Προγραμματισμός Διαδικτύου

Νικόλαος Αβούρης, Χρήστος Σιντόρης

ΚΑΛΛΙΠΟΣ
ανοικτές ακαδημαϊκές εκδόσεις

Εθνικό Πρόγραμμα Ανάπτυξης 2021-2025

ΝΙΚΟΛΑΟΣ ΑΒΟΥΡΗΣ
Καθηγητής, τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών,
Πανεπιστήμιο Πατρών

ΧΡΗΣΤΟΣ ΣΙΝΤΟΡΗΣ
ΕΔΙΠ, τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών,
Πανεπιστήμιο Πατρών

Προγραμματισμός Διαδικτύου

ΚΑΛΛΙΠΟΣ
ανοικτές ακαδημαϊκές εκδόσεις

Προγραμματισμός Διαδικτύου

Συγγραφή

Νικόλαος Αβούρης

Χρήστος Σιντόρης

Συντελεστές έκδοσης

Γλωσσική Επιμέλεια: Ντόση Ιορδανίδου

Γραφιστική Επιμέλεια: Ηλίας Σταυρόπουλος

ΚΑΛΛΙΠΟΣ

Εθνικό Μετσόβιο Πολυτεχνείο

Ηρώων Πολυτεχνείου 9, 15780 Ζωγράφου

www.kallipos.gr

ISBN: 978-618-228-104-8

Βιβλιογραφική Αναφορά: Αβούρης, Ν., & Σιντόρης, Χ. (2023). *Προγραμματισμός Διαδικτύου* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις.
http://dx.doi.org/10.57713/kallipos-337

Στη Νατάσα, στη Μαρίνα και στον Ευριπίδη.
Νικόλαος Αβούρης

Στην Αγγελική.
Χρήστος Σιντόρης

Πίνακας Περιεχομένων

Πίνακας Περιεχομένων **11**
Πίνακας Εικόνων **23**
Κεφάλαιο 1. Εισαγωγή **29**
Σύνοψη 29
Προαπαιτούμενη γνώση 29
1.1 Προοίμιο 29
1.2 Εισαγωγή στην αρχιτεκτονική του διαδικτύου 30
1.2.1 Η αρχιτεκτονική του διαδικτύου 31
1.2.2 Διευθύνσεις υπολογιστών διαδικτύου 32
1.2.3 Ασκήσεις 33
1.2.4 Η έκδοση IPv6 33
1.2.5 Σύστημα ονομασίας περιοχών DNS 33
1.2.6 Πρωτόκολλα του διαδικτύου 34
1.2.7 Εφαρμογές του διαδικτύου 35
1.3 Ο παγκόσμιος ιστός 36
1.3.1 Ο εξυπηρετητής ιστού 37
1.3.2 Ο φυλλομετρητής ιστού 39
1.3.3 Οι διευθύνσεις URL 40
1.3.4 Ειδικά σύμβολα και κωδικοποίηση 41
1.3.5 Απλοποίηση URL 41
1.4 Το πρωτόκολλο HTTP 41
1.4.1 Διαχείριση ενδιάμεσων μνημών 42
1.4.2 Η ροή επικοινωνίας HTTP 42
1.4.3 Αιτήματα εξυπηρέτησης HTTP 43
1.4.4 Μήνυμα απόκρισης HTTP 45
1.4.5 Διαχείριση σύνδεσης σε αιτήματα HTTP 46
1.4.6 Πολυπλεξία μηνυμάτων με την HTTP 2.0 47
1.4.7 Διαπραγμάτευση περιεχομένου 48
1.4.8 Διαχείριση Cookies 48
1.5 Η αρχιτεκτονική των τεχνολογιών του βιβλίου 49
1.6 Ερωτήσεις αυτοαξιολόγησης 51
1.7 Βιβλιογραφία και Αναφορές 53
Κεφάλαιο 2. Εισαγωγή στη γλώσσα HTML **55**
Σύνοψη 55
Προαπαιτούμενη γνώση 55

2.1 Εισαγωγή ... 55
2.1.1 Ιστορική αναφορά ... 55
2.2 Η πρώτη ιστοσελίδα ... 56
2.2.1 Μορφοποίηση της πρώτης ιστοσελίδας ... 58
2.3 Σύνταξη της HTML ... 59
2.3.1 Στοιχεία ... 59
2.3.2 Γνωρίσματα ... 59
2.3.3 Τυπικά γνωρίσματα στοιχείων ... 59
2.3.4 Κενά στοιχεία ... 60
2.3.5 Ειδικοί χαρακτήρες – διαχείριση κενών ... 60
2.3.6 Σχόλια στον κώδικα HTML ... 61
2.4 Μοντέλο αντικειμένων του εγγράφου HTML (DOM) ... 61
2.5 Στοιχεία του τμήματος <head> ... 63
2.5.1 Το στοιχείο <base> ... 63
2.5.2 Το στοιχείο <link> ... 63
2.5.3 Το στοιχείο <script> ... 63
2.5.4 Το στοιχείο <style> ... 64
2.5.5 Το στοιχείο <title> ... 64
2.5.6 Το στοιχείο <meta> ... 64
2.6 Τα στοιχεία του περιεχομένου: <body> ... 65
2.6.1 Στοιχεία κειμένου με σημασιολογία <h1>, ... <h6>, <p> ... 65
2.6.2 Στοιχεία κειμένου χωρίς σημασιολογία , ... 67
2.6.3 Έμφαση σε κείμενο , ... 68
2.6.4 Άνω και κάτω δείκτης κειμένου <sup>, <sub> ... 69
2.6.5 Οριζόντια γραμμή και αλλαγή γραμμής <hr>,
 ... 69
2.6.6 Στοιχεία μορφοποίησης κειμένου <pre>, <blockquote> ... 70
2.6.7 Λίστες <ol>, <ul> ... 71
2.6.8 Υπερσύνδεσμοι <a> ... 73
2.6.9 Πίνακες (στοιχείο <table>) ... 75
2.7 Ασκήσεις ... 78
2.8 Ερωτήσεις αυτοαξιολόγησης ... 79
2.9 Βιβλιογραφία και Αναφορές ... 84
Κεφάλαιο 3. Προχωρημένα θέματα της HTML ... 85
Σύνοψη ... 85
Προαπαιτούμενη γνώση ... 85
3.1 Διαχείριση εικόνων ... 85
3.2 Μορφότυποι εικόνων ... 85
3.2.1 Η σημασία του γνωρίσματος alt ... 86
3.2.2 Άλλα γνωρίσματα του <img> ... 86

3.2.3 Θέματα πνευματικών δικαιωμάτων ... 87
3.2.4 Εικόνες με το στοιχείο <figure> ... 87
3.2.5 Πολλαπλές εικόνες με το στοιχείο <picture> ... 87
3.3 Βίντεο και ήχος, τα στοιχεία <video>, <audio> ... 87
3.3.1 Το στοιχείο <video> ... 88
3.3.2 Το στοιχείο <audio> ... 89
3.4 Ενσωμάτωση στοιχείων <iframe> και <object> ... 89
3.4.1 Ενσωμάτωση στοιχείων <iframe> ... 89
3.4.2 Ενσωμάτωση στοιχείων <object> ... 90
3.5 Δομικά στοιχεία HTML ... 91
3.6 Φόρμες: το στοιχείο <form> ... 92
3.7 Η διαδικασία υποβολής φόρμας ... 92
3.8 Περιεχόμενο στοιχείου <form> ... 94
3.9 Παράδειγμα φόρμας ... 94
3.10 Τύποι στοιχείων μιας φόρμας ... 95
3.10.1 Πεδία εισαγωγής κειμένου από τον χρήστη ... 96
3.10.2 Στοιχεία επιλογής από λίστα ή γραφική διεπαφή ... 96
3.10.3 Πλήκτρα ... 96
3.10.4 Άλλα στοιχεία ... 96
3.10.5 Κύρια γνωρίσματα των στοιχείων φόρμας ... 96
3.10.6 Στοιχεία επιλογής ... 97
3.10.7 Επιλογή μέσω γραφικής διεπαφής ... 100
3.10.8 Έλεγχος εγκυρότητας της φόρμας ... 101
3.10.9 Οδηγίες σχεδίασης φόρμας ... 103
3.11 Ασκήσεις ... 104
3.12 Ερωτήσεις αυτοαξιολόγησης ... 105
3.13 Βιβλιογραφία και Αναφορές ... 107
Κεφάλαιο 4. CSS ... 109
Σύνοψη ... 109
Προαπαιτούμενη γνώση ... 109
4.1 Η γλώσσα CSS ... 109
4.2 Κανόνες CSS ... 109
4.3 Άλλες μορφές δηλώσεων CSS ... 110
4.3.1 at-rules ... 110
4.4 Επιλογείς CSS ... 110
4.4.1 Καθολικός επιλογέας (Universal selector) ... 111
4.4.2 Επιλογέας τύπου (Type selector) ... 111
4.4.3 Επιλογέας κλάσης (Class selector) ... 111
4.4.4 Επιλογέας id (Id selector) ... 112

4.4.5 Επιλογέας ιδιότητας (Attribute selector) ... 112
4.4.6 Επιλογέας ψευδοκλάσης (Pseudoclass selector) ... 113
4.4.7 Συνδυασμοί επιλογέων ... 114
4.5 Αλληλουχία και σειρά εφαρμογής των κανόνων ... 115
4.5.1 Προέλευση κανόνων ... 115
4.5.2 Ειδικότητα ... 117
4.5.3 Σειρά εμφάνισης ... 117
4.5.4 Κληρονομικότητα ... 118
4.6 Τιμές και μονάδες απόστασης και χρώματος ... 118
4.6.1 Απόσταση ... 119
4.6.2 Χρώμα ... 120
4.7 Το μοντέλο Box ... 121
4.7.1 Κανονική ροή ... 122
4.7.2 Διαστάσεις στοιχείων ... 123
4.7.3 Υπερχείλιση ... 124
4.8 Διακόσμηση στοιχείων ... 124
4.8.1 Επιλογή γραμματοσειράς ... 124
4.8.2 Διακόσμηση λίστας ... 126
4.8.3 Διακόσμηση υποβάθρου ... 126
4.9 Αντικαθιστάμενα στοιχεία ... 127
4.10 Μεταβάσεις ... 128
4.11 Ερωτήσεις ... 129
4.12 Βιβλιογραφία και Αναφορές ... 133
Κεφάλαιο 5. Διάταξη περιεχομένου με τη CSS ... 135
Σύνοψη ... 135
Προαπαιτούμενη γνώση ... 135
5.1 Διάταξη περιεχομένου ... 135
5.2 Η ιδιότητα position ... 135
5.2.1 Στατική τοποθέτηση (static positioning) ... 136
5.2.2 Σχετική τοποθέτηση (relative positioning) ... 136
5.2.3 Sticky positioning ... 137
5.2.4 Absolute positioning ... 137
5.2.5 Fixed positioning ... 138
5.3 Το stacking context και το z-index ... 139
5.4 Float ... 140
5.4.1 Flow-root και Block formatting context ... 141
5.5 CSS Grid και Flexbox ... 141
5.6 Flexbox ... 142
5.6.1 Ιδιότητες του υποδοχέα flex ... 142

5.6.2 Ιδιότητες των flex items 144
5.6.3 Στοίχιση των στοιχείων flex 146
5.6.4 Στοίχιση στον υποδοχέα flex 146
5.6.5 Στοίχιση στο κάθε στοιχείο 149
5.7 CSS Grid 149
5.7.1 Ορισμός του πλέγματος 149
5.7.2 Διαστάσεις των σειρών ή των στηλών του πλέγματος 150
5.7.3 Σιωπηρή δημιουργία του πλέγματος 150
5.7.4 Τοποθέτηση ενός στοιχείου στο πλέγμα 151
5.7.5 Τοποθέτηση σε περισσότερα κελιά 151
5.7.6 Στοίχιση στο CSS Grid 151
5.8 Ερωτήσεις αυτοαξιολόγησης 153
5.9 Βιβλιογραφία και Αναφορές 157
Κεφάλαιο 6. Bootstrap 159
Σύνοψη 159
Προαπαιτούμενη γνώση 159
6.1 Η βιβλιοθήκη Bootstrap 159
6.2 Ενσωμάτωση της Bootstrap 159
6.3 Bootstrap containers 160
6.4 Το πλέγμα της Bootstrap (grid) 160
6.4.1 Προσαρμόσιμες κλάσεις 162
6.4.2 Offset 163
6.4.3 Διάταξη των στηλών 164
6.4.4 Ταξινόμηση 164
6.4.5 Αναδίπλωση 164
6.5 Βοηθήματα 164
6.5.1 Περιθώρια 164
6.5.2 Πλάτος 164
6.5.3 Περίγραμμα 164
6.6 Φόρμες 165
6.6.1 Input 165
6.6.2 Κείμενα στη φόρμα 166
6.6.3 Input group 168
6.7 Έλεγχος εγκυρότητας 168
6.7.1 Έλεγχος εγκυρότητας με την Bootstrap 169
6.7 Ερωτήσεις αυτοαξιολόγησης 172
6.8 Βιβλιογραφία και Αναφορές 174
Κεφάλαιο 7. Εισαγωγή στην JavaScript 175
Σύνοψη 175

Προαπαιτούμενη γνώση 175
7.1 Εισαγωγή – Ιστορικό σημείωμα 175
7.1.1 Κύρια χαρακτηριστικά της JavaScript 175
7.1.2 Χρήση της JavaScript 176
7.1.3 Εκδόσεις της JavaScript 177
7.1.4 Μηχανές εκτέλεσης κώδικα JavaScript 178
7.2 Ανάπτυξη και αποσφαλμάτωση κώδικα JavaScript 178
7.3 Σύνταξη ενός προγράμματος JavaScript 179
7.3.1 Δήλωση μεταβλητών let, const, var 180
7.3.2 Πολλαπλές δηλώσεις 180
7.3.3 Εμβέλεια μεταβλητών let 181
7.3.4 Κλήση μεταβλητής πριν τη δήλωσή της 181
7.3.5 Τελεστές και εκφράσεις 181
7.3.6 Τελεστές πράξεων δυαδικών αριθμών 182
7.3.7 Διεπαφή με τον χρήστη 182
7.4 Η JavaScript στον φυλλομετρητή 184
7.5 Διαχωρισμός κώδικα HMTL, CSS, JavaScript 185
7.6 Η ακολουθία συμβάντων κατά το φόρτωμα μιας σελίδας 186
7.6.1 Ένα παράδειγμα 187
7.7 Το περιβάλλον εκτέλεσης της JavaScript 188
7.7.1 Δραστηριότητα 189
7.8 Βασικά στοιχεία του αντικείμενου window 189
7.8.1 Συναρτήσεις 189
7.8.2 Βασικά αντικείμενα 189
7.8.3 Αντικείμενα αριθμών 189
7.8.4 Κείμενο 190
7.8.5 Συλλογές αντικειμένων 190
7.8.6 Σφάλματα 190
7.9 Διεπαφή με το Document Object Model (DOM) 190
7.9.1 Ανάκτηση στοιχείων του DOM 191
7.9.2 Διαχείριση των γνωρισμάτων στοιχείων DOM 192
7.9.3 Διαχείριση περιεχομένου στοιχείων DOM 192
7.9.4 Διαχείριση του στιλ των στοιχείων DOM 192
7.9.5 Διαπέραση δένδρου DOM 192
7.9.6 Εισαγωγή και διαγραφή στοιχείων του DOM 193
7.10 Ερωτήσεις αυτοαξιολόγησης 195
7.11 Βιβλιογραφία και Αναφορές 197
Κεφάλαιο 8. Πρωτογενείς τύποι δεδομένων και εντολές της JavaScript 199
Σύνοψη 199

Προαπαιτούμενη γνώση 199
8.1 Τύποι δεδομένων 199
8.2 Πρωτογενείς τύποι δεδομένων: Number 199
8.3 Πρωτογενείς τύποι δεδομένων: Συμβολοσειρές 200
8.3.1 Δείκτης συμβολοσειράς 201
8.3.2 Μετατροπές από συμβολοσειρές σε αριθμούς 202
8.3.3 Κύριες μέθοδοι και τελεστές του String 202
8.3.4 Συμβολοσειρές-πρότυπα (template literals) 203
Ασκήσεις 203
8.4 Πρωτογενείς τύποι δεδομένων: null και undefined 204
8.5 Πρωτογενείς τύποι δεδομένων: Boolean 204
8.5.1 Τιμή λογικών εκφράσεων 205
8.6 Προγραμματιστικές δομές της JavaScript 205
8.6.1 Εντολή if-else 205
8.6.2 Εντολή switch 206
8.6.3 Υπό συνθήκη εκχώρηση τιμής (τριαδικός τελεστής) 207
8.6.4 Διαχείριση σφαλμάτων με try-catch 207
8.7 Δομές επανάληψης 208
8.7.1 Εντολή while 209
8.7.2 Εντολή do/while 209
8.7.3 Εντολή for 210
8.7.4 Εντολή for/of 211
8.7.5 Εντολή for/in 211
8.8 Ερωτήσεις αυτοαξιολόγησης 213
8.9 Βιβλιογραφία και Αναφορές 217
Κεφάλαιο 9. Τύποι δεδομένων αναφοράς: Πίνακες, Συναρτήσεις και Αντικείμενα στην JavaScript 219
Σύνοψη 219
Προαπαιτούμενη γνώση 219
9.1 Τύποι δεδομένων αναφοράς 219
9.2 Πίνακες 219
9.2.1 Δημιουργία πίνακα με ορισμό της τιμής του 220
9.2.2 Δημιουργία πίνακα με new Array() 220
9.2.3 Αραιοί πίνακες 220
9.2.4 Αρχικοποίηση πίνακα 221
9.2.5 Τροποποίηση στοιχείων πίνακα 221
9.2.6 Μέθοδοι πινάκων 221
9.2.7 Μέθοδοι υποπινάκων 222
9.2.8 Μέθοδοι αναζήτησης 223
9.2.9 Μέθοδοι ταξινόμησης 223

9.2.10 Μέθοδοι μετατροπής πίνακα σε συμβολοσειρά ... 224
9.3 Συναρτήσεις ... 224
9.3.1 Εισαγωγή ... 224
9.3.2 Ορισμός συνάρτησης με χρήση της λέξης function ... 225
9.3.3 Ορισμός συνάρτησης με χρήση βέλους => ... 226
9.3.4 Εμφώλευση συναρτήσεων ... 226
9.3.5 Εμβέλεια μεταβλητών ... 227
9.3.6 Προαιρετικά ορίσματα συναρτήσεων ... 228
9.3.7 Ο τελεστής ... στα ορίσματα συνάρτησης ... 229
9.3.8 Στατικές μεταβλητές συνάρτησης ... 230
9.3.9 Οι συναρτήσεις ορίζουν μοναδικό χώρο ονομάτων ... 230
9.3.10 Μέθοδοι επεξεργασίας πινάκων ... 231
9.4 Αντικείμενα και κλάσεις ... 234
9.4.1 Μετατροπή αντικειμένων σε JSON ... 236
9.4.2 Δημιουργία κλάσεων και αντικειμένων ... 236
9.4.3 Ορισμός μεθόδων ... 237
9.4.4 Ορισμός κλάσεων με τη λέξη-κλειδί class ... 238
9.4.5 Κληρονομικότητα κλάσεων ... 240
9.5 Ερωτήσεις αυτοαξιολόγησης ... 241
9.6 Βιβλιογραφία και Αναφορές ... 246
Κεφάλαιο 10. Ασύγχρονη JavaScript – διαχείριση συμβάντων ... 247
Σύνοψη ... 247
Προαπαιτούμενη γνώση ... 247
10.1 Ασύγχρονη εκτέλεση κώδικα ... 247
10.1.1 Συναρτήσεις setTimeout() και setInterval() ... 247
10.1.2 Ο βρόχος ελέγχου συμβάντων ... 249
10.2 Ορισμός χειριστή συμβάντων ... 251
10.2.1 Κατηγορίες συμβάντων ... 251
10.2.2 Καταχώριση χειριστών συμβάντων ... 251
10.3 Προγραμματισμός με κλήση συναρτήσεων επιστροφής ... 253
10.4 Ο μηχανισμός Promise ... 255
10.4.1 Καταστάσεις του αντικείμενου Promise ... 255
10.4.2 Καταναλωτές υποσχέσεων ... 257
10.4.3 Παράδειγμα αλυσίδας καταναλωτών υποσχέσεων ... 258
10.4.4 Promises στον βρόχο ελέγχου συμβάντων ... 259
10.5 Η διεπαφή fetch ... 260
10.5.1 Παράδειγμα χρήσης της fetch ... 260
10.5.2 Παράδειγμα: Οι καλοί γείτονες ... 262
10.6 Διαχείριση συμβάντων ... 264

10.6.1 Κλήση χειριστή συμβάντος ... 264
10.6.2 Ο μηχανισμός φυσαλίδας ... 264
10.7 Ερωτήσεις αυτοαξιολόγησης ... 267
10.8 Βιβλιογραφία και Αναφορές ... 271
Κεφάλαιο 11. Προγραμματισμός στην πλευρά του εξυπηρετητή ... 273
Σύνοψη ... 273
Προαπαιτούμενη γνώση ... 273
11.1 Εισαγωγή ... 273
11.2 Λήψη και εγκατάσταση της Node.js ... 274
11.2.1 Μια απλή εφαρμογή σε Node.js ... 275
11.2.2 Διαφορές με την JavaScript στον φυλλομετρητή ... 275
11.2.3 Ένας απλός εξυπηρετητής σε Node.js ... 275
11.3 Βιβλιοθήκες και πακέτα στη Node.js ... 276
11.3.1 Ενσωματωμένες βιβλιοθήκες ... 276
11.3.2 Τοπικές βιβλιοθήκες ... 276
11.3.3 Βιβλιοθήκες ECMAScript6 και CommonJS ... 276
11.3.4 Πακέτα και το εργαλείο npm ... 278
11.3.5 Εγκατάσταση πακέτων από το npmjs.com ... 279
11.3.6 Αρίθμηση εκδόσεων κατά SemVer ... 280
11.4 Βασικές εργασίες με αρχεία ... 281
11.4.1 Εγγραφή σε αρχείο ... 281
11.5 Async/await ... 284
11.5.1 Η λέξη-κλειδί async ... 284
11.5.2 Η λέξη-κλειδί await ... 285
11.5.3 Χειρισμός σφαλμάτων ... 287
11.6 Η Node.js ως εξυπηρετητής διαδικτύου ... 288
11.7 ES6 export και import ... 289
11.7.1 Ονοματισμένα export ... 290
11.7.2 Μετονομασία των στοιχείων ... 290
11.7.3 Default exports ... 290
11.7.4 Εισαγωγή της βιβλιοθήκης ως αντικειμένου ... 291
11.7.5 Φόρτωση και εκτέλεση βιβλιοθήκης ... 291
11.7.6 Δυναμικά import ... 292
11.7.7 Ιδιότητες και περιορισμοί ... 292
11.8 Βοηθητικά εργαλεία για την ανάπτυξη εφαρμογών Node.js ... 293
11.8.1 Dependencies και DevDependencies ... 293
11.8.2 Το πακέτο nodemon ... 293
11.8.3 Ορισμός και χρήση μεταβλητών περιβάλλοντος ... 294
11.8.4 Μεταβλητές περιβάλλοντος μέσω dotenv ... 294

11.8.5 Φιλοξενία της εφαρμογής στο Heroku ... 295
11.8.6 Αποσφαλμάτωση με το Visual Studio Code ... 296
11.9 Ερωτήσεις ... 301
11.10 Βιβλιογραφία και Αναφορές ... 304
Κεφάλαιο 12. Ανάπτυξη εφαρμογών με το πακέτο Express.js ... 305
Σύνοψη ... 305
Προαπαιτούμενη γνώση ... 305
12.1 Εισαγωγή ... 305
12.2 Μια απλή εφαρμογή με το Express.js ... 305
12.3 Ο κύκλος επεξεργασίας του αιτήματος ... 306
12.3.1 Τα middleware του Express.js ... 306
12.3.2 Καθορισμός της αλυσίδας επεξεργασίας ... 306
12.3.3 Δρομολόγηση ... 307
12.3.4 Προσάρτηση σε συγκεκριμένο μονοπάτι ... 308
12.3.5 Προσάρτηση σε συγκεκριμένη μέθοδο ... 308
12.3.6 Δρομολόγηση με τη μέθοδο app.route() ... 308
12.3.7 Δρομολόγηση με την κλάση Router ... 309
12.4 Τα αντικείμενα αιτήματος και απόκρισης ... 309
12.4.1 Το αντικείμενο αιτήματος ... 310
12.4.2 Ονοματισμένες παράμετροι ... 310
12.4.3 Πρόσβαση στα δεδομένα POST ... 310
12.4.4 Το αντικείμενο απόκρισης ... 311
12.5 Μηχανές template ... 311
12.5.1 Βασική χρήση της Handlebars ... 311
12.5.2 Το αντικείμενο res.locals ... 314
12.6 Βοηθητικά εργαλεία της Handlebars ... 315
12.7 Handlebars - Expressions ... 315
12.8 Ερωτήσεις ... 316
12.9 Βιβλιογραφία και Αναφορές ... 319
Κεφάλαιο 13. Σύνδεση του εξυπηρετητή με βάση δεδομένων ... 321
Σύνοψη ... 321
Προαπαιτούμενη γνώση ... 321
13.1 Σχεδίαση μιας διαδικτυακής εφαρμογής – δεδομένα ... 321
13.2 Η αρχιτεκτονική MVC (model-view-controller) ... 324
13.2.1 Ο Ελεγκτής της εφαρμογής ... 325
13.2.2 Το Μοντέλο της εφαρμογής myBooks ... 327
13.3 Υλοποίηση με μόνιμη αποθήκευση σε αρχεία JSON ... 328
13.4 Υλοποίηση με μόνιμη αποθήκευση σε βάση δεδομένων PostgreSQL ... 329
13.4.1 Εγκατάσταση της PostgreSQL ... 330

13.4.2 Σύνδεση με την PostgreSQL ... 332
13.4.3 Υλοποίηση της διεπαφής με τη βάση δεδομένων PostgreSQL ... 333
13.5 ORM (Object Relational Mapper) ... 335
13.6 Φιλοξενία της εφαρμογής και της βάσης δεδομένων στο Heroku ... 336
13.7 Σύνδεση με βάση δεδομένων MongoDB ... 337
13.8 Ερωτήσεις αυτοαξιολόγησης ... 340
13.9 Βιβλιογραφία και Αναφορές ... 342
Κεφάλαιο 14. Πρόσβαση χρηστών και αυθεντικοποίηση ... 343
Σύνοψη ... 343
Προαπαιτούμενη γνώση ... 343
14.1 Εισαγωγή ... 343
14.2 Αυθεντικοποίηση ... 343
14.2.1 Απαιτήσεις από το σύστημα διαχείρισης χρηστών ... 344
14.2.2 Φύλαξη των διαπιστευτηρίων ... 345
14.2.3 Προστασία διαπιστευτηρίων ... 345
14.2.4 Κατατεμαχισμός με «αλάτισμα» ... 347
14.2.5 Η συνάρτηση κατατεμαχισμού bcrypt ... 347
14.3 Διαχείριση συνεδρίας ... 349
14.3.1 Μια τυπική συνεδρία ... 349
14.3.2 Διαχείριση συνεδρίας με το express-session ... 352
14.3.3 Αποθετήριο συνεδρίας ... 353
14.3.4 Μεταβλητές συνεδρίας ... 353
14.4 Προστασία περιοχής με το Express.js ... 355
14.5 Ασφαλής μεταφορά ... 357
14.5.1 Επίπεδα επικύρωσης πιστοποιητικών ... 358
14.5.2 Ασφαλής μεταφορά με Let’s Encrypt στο Heroku ... 359
14.6 Ερωτήσεις ... 359
14.7 Βιβλιογραφία και Αναφορές ... 360
Παράρτημα Απαντήσεις στις ερωτήσεις ... 361

Πίνακας Εικόνων

Εικόνα 1.1 Αρχιτεκτονική του διαδικτύου ως ιεραρχία δικτύων ISP (παρόχων υπηρεσιών διαδικτύου).....31
Εικόνα 1.2 Χάρτης των υποβρύχιων καλωδίων που παρέχει η ιστοσελίδα https://www.submarinecablemap.com/. Οι συνδέσεις αυτές, που έχουν ποντιστεί στον βυθό της θάλασσας ή άλλες αντίστοιχες στην ξηρά, περιλαμβάνουν οπτικές ίνες μετάδοσης δεδομένων υψηλής ταχύτητας που συνδέονται με κόμβους που αποτελούν τον κορμό του διαδικτύου. 32
Εικόνα 1.3 Διαστρωμάτωση επιπέδων πρωτοκόλλων του διαδικτύου.....35
Εικόνα 1.4 Ο Tim Berners-Lee μπροστά στον πρώτο φυλλομετρητή ιστού. http://cds.cern.ch/record/39437#31 36
Εικόνα 1.5 Πλήθος εξυπηρετητών διασυνδεδεμένων στο διαδίκτυο, περίοδος 1970-2020. https://commons.wikimedia.org/wiki/File:Internet_Hosts_Count_log.svg.....37
Εικόνα 1.6 Ο δεύτερος πόλεμος των φυλλομετρητών: ποσοστό χρήσης των φυλλομετρητών κατά την περίοδο 2008-2019. https://commons.wikimedia.org/wiki/File:BrowserUsageShare.png 39
Εικόνα 1.7 Παράδειγμα URL. 40
Εικόνα 1.8 Σύνδεση πελάτη-εξυπηρετητή με παρεμβολή ενδιάμεσων. 42
Εικόνα 1.9 Στα αριστερά φαίνεται η κατάσταση των αιτημάτων HTTP όπως εμφανίζονται στην καρτέλα Network των εργαλείων σχεδιαστή του Chrome. Ενώ στα δεξιά εμφανίζεται η ιστοσελίδα. Αν επιλέξουμε ένα από τα αιτήματα, μπορούμε να δούμε το raw HTTP μήνυμα (αίτημα και απόκριση). 43
Εικόνα 1.10 Η δομή του μηνύματος αίτησης HTTP. 44
Εικόνα 1.11 Παράδειγμα μηνύματος απόκρισης HTTP που θα μπορούσε να στείλει ο εξυπηρετητής για να απαντήσει στο προηγούμενο μήνυμα. 45
Εικόνα 1.12 Επίμονη σύνδεση.....47
Εικόνα 1.13 Διαστρωμάτωση εφαρμογής ιστού, αριστερά HTTP 1.1, δεξιά HTTP 2.0.....48
Εικόνα 1.14 Ροή δυαδικών πλαισίων δεδομένων στο πρωτόκολλο HTTP 2.0. 48
Εικόνα 1.15 Αρχιτεκτονική μιας τυπικής εφαρμογής που θα αναπτύξουμε στο πλαίσιο του βιβλίου.....49
Εικόνα 2.1 Εμφάνιση του εγγράφου στον φυλλομετρητή.....57
Εικόνα 2.2 Τροποποίηση της σελίδας της εικόνας **Εικόνα 2.1**.....58
Εικόνα 2.3 Η σελίδα της εικόνας **Εικόνα 2.1** με εισαγωγή εντολής CSS.....58
Εικόνα 2.4 Εμφάνιση των δύο παραγράφων στον φυλλομετρητή. 61
Εικόνα 2.5 Αρχιτεκτονική του φυλλομετρητή. 62
Εικόνα 2.6 Αναπαράσταση του DOM της αρχικής ιστοσελίδας.....62
Εικόνα 2.7 Χειρόγραφο του ποιητή Κ.Καβάφη. https://cavafy.onassis.org/el/object/39eh-xggr-mk3e/ 65
Εικόνα 2.8 Παρουσίαση του ποιήματος χωρίς οργάνωσή του σε στοιχεία HTML.....66
Εικόνα 2.9 Παρουσίαση του ποιήματος μετά την οργάνωσή του σε στοιχεία HTML.....66
Εικόνα 2.10 Παρουσίαση του ποιήματος με εφαρμογή κανόνα CSS για διπλάσιο μέγεθος πρώτου γράμματος κάθε στροφής. 68
Εικόνα 2.11 Παράδειγμα στοιχείου 68
Εικόνα 2.12 Παράδειγμα στοιχείου 69
Εικόνα 2.13 Κείμενα που περιλαμβάνουν δείκτες και κάτω δείκτες.....69
Εικόνα 2.14 Εμφάνιση κώδικα με χρήση <pre>.....70
Εικόνα 2.15 Παράδειγμα χρήσης στοιχείου <blockquote>. 71
Εικόνα 2.16 Παρουσίαση του περιεχομένου του γνωρίσματος title, όταν το ποντίκι αιωρείται πάνω από το στοιχείο (hover). 74
Εικόνα 2.17 Παρουσίαση του πίνακα των ελληνικών βουνών (έχει περιληφθεί εντολή CSS για το πλαίσιο των κελιών). 76
Εικόνα 2.18 Παράδειγμα πίνακα με εκτεινόμενα κελιά: ο πληθυσμός μιας χώρας. 76
Εικόνα 2.19 Πίνακας κατηγοριοποίησης ζωντανών οργανισμών. 79
Εικόνα 3.1 Παράδειγμα εισαγωγής στοιχείου <iframe>: ο χάρτης.....90
Εικόνα 3.2 (α) Οργάνωση περιεχομένου με μη σημασιολογικά στοιχεία, (β) Οργάνωση με χρήση

σημασιολογικών στοιχείων. 91
Εικόνα 3.3 Η υποβολή μιας φόρμας συνεπάγεται αίτημα HTTP προς τον εξυπηρετητή.....93
Εικόνα 3.4 Παράδειγμα φόρμας που περιλαμβάνει δύο στοιχεία input και ένα πλήκτρο υποβολής. 94
Εικόνα 3.5 Μήνυμα σφάλματος που παράγεται από το στοιχείο input τύπου email. 95
Εικόνα 3.6 Παράδειγμα φόρμας με επιλογή απάντησης από λίστα στοιχείων select/option. 98
Εικόνα 3.7 Παράδειγμα στοιχείου κύλισης από πεδίο τιμών. 98
Εικόνα 3.8 Παράδειγμα φόρμας που περιλαμβάνει στοιχεία checkbox. 99
Εικόνα 3.9 Παλιό ραδιόφωνο με πλήκτρα επιλογής μπάντας συχνοτήτων, από αυτά προέρχεται ο όρος radio buttons. 99
Εικόνα 3.10 Παράδειγμα αμοιβαία αποκλειόμενων επιλογών, στοιχεία radio. 100
Εικόνα 3.11 Παράδειγμα ομαδοποίησης στοιχείων φόρμας με το <fieldset>. 100
Εικόνα 3.12 Παράδειγμα αλληλεπίδρασης με γραφικό στοιχείο για την επιλογή ημερομηνίας (Chrome). 101
Εικόνα 3.13 Τα τρία επίπεδα ελέγχου εγκυρότητας των δεδομένων που εισάγονται σε μια φόρμα. 102
Εικόνα 4.1 Τα συστατικά ενός κανόνα CSS. 110
Εικόνα 4.2 Επειδή η ιδιότητα font-size είναι κληρονομούμενη (ενότητα 4.5.4), η αλλαγή στο μέγεθος της γραμματοσειράς ενός στοιχείου κληροδοτείται στους απογόνους του και δρα αθροιστικά. Με τη μονάδα rem μπορούμε να το αποφύγουμε αυτό. 120
Εικόνα 4.3 Στην αναπαράσταση HSL η γωνία h επιλέγει την απόχρωση, ξεκινώντας από τις 0° για το κόκκινο. Στο συγκεκριμένο σχήμα επιλέγονται 12 χρώματα, ένα χρώμα κάθε 30°. Όσο μειώνεται ο κορεσμός (s) από το 100% προς το 0%, το χρώμα πλησιάζει προς το γκρι – εδώ σε 5 βήματα του 20% από έξω προς το κέντρο του δίσκου. Ο παράγοντας l καθορίζει τη φωτεινότητα του χρώματος, 50% στο αριστερό σχήμα, 75% στο μεσαίο και 25% στο δεξί. Ο τρόπος αυτός επιλογής χρώματος είναι κάπως πιο προβλέψιμος σε σχέση με την αναπαράσταση RGB. 121
Εικόνα 4.4 Το μοντέλο box. 122
Εικόνα 4.5 Διάταξη των κουτιών inline και block στην κανονική ροή για συστήματα γραφής όπως τα δυτικά, όπου γράφουμε από τα αριστερά προς τα δεξιά και από πάνω προς τα κάτω. 123
Εικόνα 4.6 Τα εξωτερικά περιθώρια δύο γειτονικών στοιχείων block συμπτύσσονται. 123
Εικόνα 4.7 Αποτέλεσμα του παραπάνω κώδικα για τιμές της ιδιότητας line-height=1.0, 1.5, 2.0 αντίστοιχα. 126
Εικόνα 4.8 Παρουσίαση στοιχείου <ul> με list-style-type=square και circle. 126
Εικόνα 4.9 Παρουσίαση στοιχείου <ol> με list-style-type: lower-roman και lower-greek. 126
Εικόνα 4.10 Παρουσίαση του στοιχείου της λίστας με εφαρμογή εικόνας υποβάθρου – Η εικόνα αφορά την ποιήτρια Υπατία, από την Αλεξάνδρεια. 127
Εικόνα 4.11 Η επίδραση τεσσάρων από τις πιθανές τιμές της ιδιότητας object-fit (Τοπίο με την πτώση του Ίκαρου, Πίτερ Μπρίγκελ ο Πρεσβύτερος). 128
Εικόνα 4.12 Παραδείγματα για 3 από τις 5 προκαθορισμένες καμπύλες Μπεζιέ. Οι παράμετροι της συνάρτησης cubic-bezier() είναι οι συντεταγμένες των σημείων ελέγχου. 129
Εικόνα 5.1 Σχετική τοποθέτηση. 136
Εικόνα 5.2 Με τις ιδιότητες top, right, bottom, left ορίζουμε μια νέα θέση για το στοιχείο, μετακινώντας το από την εκάστοτε πλευρά. Μπορούμε να χρησιμοποιήσουμε και τις inset-block-start, inset-block-end για μετατόπιση στην κατεύθυνση block ή τις inset-inline-start, inset-inline-end για μετατόπιση στην κατεύθυνση inline. 137
Εικόνα 5.3 Απόλυτη τοποθέτηση. 138
Εικόνα 5.4 Fixed positioning. 138
Εικόνα 5.5 Άξονας z. Η τιμή του z-index καθορίζει τη σειρά των στοιχείων πάνω στον άξονα z και συνεπώς το ποιο στοιχείο θα εμφανίζεται πιο κοντά. 139
Εικόνα 5.6 Τα δύο λευκά κουτιά έχουν γίνει float. 140
Εικόνα 5.7 Με την ιδιότητα clear: right στο float, η περιοχή στα δεξιά του θα μείνει ελεύθερη από περιεχόμενο. 140
Εικόνα 5.8 Το στοιχείο που είναι float (λευκό κουτί) δεν συμμετέχει στον υπολογισμό του ύψους του στοιχείου που το περιέχει. 141
Εικόνα 5.9 Όταν το ύψος του στοιχείου μέσα στο οποίο περιέχεται το float είναι μικρότερο από το ύψος του float, το float θα επικαλύψει τα επόμενα στοιχεία. 141
Εικόνα 5.10 Ο κύριος άξονας του υποδοχέα flex. 142

Εικόνα 5.11 Ο κάθετος άξονας του υποδοχέα flex. 143
Εικόνα 5.12 Οι άξονες του υποδοχέα flex αλλάζουν κατεύθυνση με την ιδιότητα **flex-direction**. Οι αριθμοί στα κουτιά δείχνουν τη σειρά με την οποία τοποθετούνται. 143
Εικόνα 5.13 Με τη flex-wrap επιτρέπουμε ή απαγορεύουμε στη σειρά των στοιχείων flex να αναδιπλώνεται. 144
Εικόνα 5.14 Στο παράδειγμα έχουμε 4 στοιχεία flex με flex-grow: 1 και ο διαθέσιμος χώρος είναι 100px. Τότε καθένα από τα 4 θα μεγαλώσει κατά (100px / 4) x 1 = 25px. 145
Εικόνα 5.15 Στο παράδειγμα αυτό έχουμε το 2ο στοιχείο με flex-grow: 2 και τα υπόλοιπα τρία με flex-grow: 1. Ο διαθέσιμος χώρος των 100px θα μοιραστεί σε 5 μονάδες ως εξής: Το στοιχείο με το flex-grow: 2 θα πάρει (100 / 5) x 2 = 40px και τα άλλα 3 στοιχεία από (100px / 5) x 1 = 20px. 145
Εικόνα 5.16 Με την **justify-content** τροποποιούμε τη διάταξη στον **κύριο άξονα**. 147
Εικόνα 5.17 Με την **align-items** τροποποιούμε τη διάταξη των στοιχείων στον **κάθετο άξονα**. 147
Εικόνα 5.18 Με την **align-content** στοιχίζουμε τις γραμμές του Flexbox στον **κάθετο άξονα**. 148
Εικόνα 5.19 Ένας υποδοχέας πλέγματος περιέχει στοιχεία πλέγματος. 149
Εικόνα 5.20 Οι γραμμές του πλέγματος σχηματίζουν τις σειρές ή τις στήλες μέσα στις οποίες τοποθετούνται τα στοιχεία. Στο παράδειγμα η δεύτερη σειρά βρίσκεται ανάμεσα από τη 2η και την 3η οριζόντια γραμμή. 150
Εικόνα 5.21 Τοποθέτηση του στοιχείου. 151
Εικόνα 5.22 Με την justify-self τροποποιούμε τη διάταξη στον κύριο άξονα. 152
Εικόνα 6.1 Τα breakpoints της Bootstrap. 160
Εικόνα 6.2. Τρεις ισόπαχες στήλες στο πλέγμα της Bootstrap (έχει προστεθεί με CSS γκρι χρώμα και πλαίσιο σε κάθε κελί). 161
Εικόνα 6.3. Το ίδιο παράδειγμα, αυτή τη φορά με .container-fluid. 161
Εικόνα 6.4 Bootstrap: ενδεικτικά επιθήματα για τον καθορισμό του πλάτους των στηλών. 162
Εικόνα 6.5. Η γραμμή αναδιπλώνεται σε δύο σειρές για να χωρέσει στο διαθέσιμο πλάτος. 162
Εικόνα 6.6. Οι τρεις στήλες με επίθημα -md εμφανίζονται σε μια σειρά όσο το διαθέσιμο πλάτος είναι μεγαλύτερο από md. 163
Εικόνα 6.7. Αν το διαθέσιμο πλάτος γίνει μικρότερο από md, η γραμμή αναδιπλώνεται. Το αποτέλεσμα είναι οι τρεις στήλες της γραμμής να εμφανίζονται η καθεμία σε δική της σειρά. 163
Εικόνα 6.8. Στη δεύτερη στήλη έχει εφαρμοστεί offset τριών μονάδων. 163
Εικόνα 6.9 Η φόρμα Bootstrap με δύο στοιχεία input, με placeholder και label. 165
Εικόνα 6.10. Οι κλάσεις .form-control-lg και .form-control-sm μάς επιτρέπουν να αλλάξουμε εύκολα το μέγεθος ενός στοιχείου της φόρμας μας. 166
Εικόνα 6.11. Στη φόρμα έχει προστεθεί με την κλάση .form-text ένα επεξηγηματικό κείμενο για το πεδίο του συνθηματικού. 167
Εικόνα 6.12 Το στοιχείο **radio** στην Bootstrap. 167
Εικόνα 6.13 Το στοιχείο **checkbox** στην Bootstrap. 168
Εικόνα 6.14 Το στοιχείο **switch** της Bootstrap. 168
Εικόνα 6.15 Το **input group** της Bootstrap. 168
Εικόνα 6.16 Ο χρήστης έχει πατήσει το κουμπί «Έλεγχος» χωρίς όμως να συμπληρώσει τιμή στο πεδίο Όνομα, το οποίο είναι απαραίτητο (required). Κλήθηκε η συνάρτηση reportValidity() του constraint validation API και εμφανίστηκε ένα μήνυμα που προειδοποιεί τον χρήστη. 169
Εικόνα 6.17 Το μήνυμα έχει αλλάξει με τη συνάρτηση setCustomValidity() του constraint validation API. 169
Εικόνα 6.18 Στο πεδίο «Όνομα» έχει εφαρμοστεί από το constraint validation API η ψευδοκλάση :invalid. 171
Εικόνα 7.1 Δημοφιλία της JavaScript σύμφωνα με την ετήσια επισκόπηση του StackOverflow. https://survey.stackoverflow.co/2022/#technology 176
Εικόνα 7.2 Εκτέλεση κώδικα JavaScript στην κονσόλα του Chrome. 178
Εικόνα 7.3 Εκτέλεση κώδικα JavaScript στο περιβάλλον RunJS. 179
Εικόνα 7.4 Η 'alert()' όπως εκτελείται στον φυλλομετρητή Firefox. 183
Εικόνα 7.5 Η prompt() όπως εκτελείται στον φυλλομετρητή Firefox. 183
Εικόνα 7.6 Φόρτωμα κώδικα JavaScript με μπλοκάρισμα φορτώματος της HTML/DOM. 184
Εικόνα 7.7 Φόρτωμα κώδικα JavaScript με την ιδιότητα async, ασύγχρονα με την HTML. 185
Εικόνα 7.8 Φόρτωμα κώδικα JavaScript με την ιδιότητα defer: η εκτέλεση του κώδικα μεταφέρεται

μετά την ολοκλήρωση του DOM. 185
Εικόνα 7.9 Συνήθης οργάνωση των αρχείων ενός πρότζεκτ ανάπτυξης ιστοσελίδας. 186
Εικόνα 7.10 Τυπική φόρτωση της σελίδας. 188
Εικόνα 7.11 Κατηγορίες αντικειμένων που έχει πρόσβαση η JavaScript. 188
Εικόνα 7.12 Το Document Object Model (DOM) είναι μια ιεραρχία από κόμβους. 191
Εικόνα 7.13 Απόσπασμα ενός δέντρου DOM. 193
Εικόνα 9.1 Δημιουργός, πρωτότυπο για την κλάση Car. 239
Εικόνα 10.1 Δομές μνήμης κατά την εκτέλεση ενός προγράμματος JavaScript: στοίβα κλήσεων και σωρός. Η main() έχει καλέσει τη squar και αυτή με τη σειρά της τη mult(), η οποία εκτελείται τώρα. Όταν τελειώσει η εκτέλεσή της, το πλαίσιο της mult() θα αφαιρεθεί από τη στοίβα κλήσεων και ο έλεγχος θα περάσει στη squar(). 249
Εικόνα 10.2 Δομές μνήμης κατά την εκτέλεση ενός προγράμματος JavaScipt με εκτέλεση ασύγχρονων τμημάτων κώδικα: χρήση ουράς κλήσεων συναρτήσεων επιστροφής. 250
Εικόνα 10.3 Διάγραμμα καταστάσεων ενός αντικείμενου Promise (υπόσχεση). 256
Εικόνα 10.4 Δομές μνήμης κατά την εκτέλεση κώδικα JavaScript που περιλαμβάνει ασύγχρονα τμήματα κώδικα και χειρισμό υποσχέσεων. 260
Εικόνα 10.5 Παράδειγμα ιεραρχίας στοιχείων και αντίστοιχων χειριστών συμβάντων. 265
Εικόνα 11.1 Αρχιτεκτονική μιας τυπικής εφαρμογής που θα αναπτύξουμε στο πλαίσιο του βιβλίου. 274
Εικόνα 11.2 Οι εκδόσεις της Node.js που διατίθενται κάθε Απρίλιο υποστηρίζονται για τρία χρόνια (LTS) και έχουν ζυγό αριθμό. Αυτές που διατίθενται κάθε Οκτώβριο έχουν μονό αριθμό. 274
Εικόνα 11.3 Στιγμιότυπο από την τεκμηρίωση των ενσωματωμένων βιβλιοθηκών της Node.js. Με τον διακόπτη μπορούμε να επιλέξουμε τον κώδικα για την εισαγωγή του module στο πρόγραμμά μας. 277
Εικόνα 11.4 Η αρίθμηση κατά SemVer χρησιμοποιεί ακεραίους σε τρεις θέσεις για να σηματοδοτήσει το είδος των αλλαγών μεταξύ εκδόσεων. 280
Εικόνα 11.5 *Το τερματικό είναι προσβάσιμο από το μενού Terminal->New Terminal.* 281
Εικόνα 11.6 Η εφαρμογή παράγει διαφορετική απόκριση με βάση την τιμή της παραμέτρου onoma. 289
Εικόνα 11.7 Βήματα για τη δημιουργία του αρχείου .vscode/launch.json, αν δεν υπάρχει ήδη. 297
Εικόνα 11.8 Δημιουργούμε ένα νέο launch configuration. 297
Εικόνα 11.9 Επιλογή του προτιμώμενου launch configuration. 298
Εικόνα 11.10 Προσαρμογή των επιμέρους ρυθμίσεων του launch configuration. Προσοχή στο όνομα του αρχείου που θέλουμε να εκσφαλματώσουμε. 299
Εικόνα 11.11 Το εργαλείο αποσφαλμάτωσης του Visual Studio Code. 300
Εικόνα 11.12 Η εκτέλεση του κώδικα έχει σταματήσει προσωρινά στο breakpoint. 301
Εικόνα 12.1 Η αλυσίδα επεξεργασίας του Express.js. 306
Εικόνα 12.2 Η μηχανή template συνθέτει την τελική απάντηση πριν αποσταλεί στον πελάτη. 312
Εικόνα 12.3 Στην ορολογία του Handlebars ένα template «τοποθετείται» μέσα σε ένα "view". Το template μπορεί να συνοδεύεται από επιμέρους "partial" τμήματα. 313
Εικόνα 13.1 Διάγραμμα μετάβασης καταστάσεων της εφαρμογής myBooks. 322
Εικόνα 13.2 Η βασική οθόνη της εφαρμογής myBooks. 323
Εικόνα 13.3 Το μοντέλο δεδομένων της εφαρμογής myBooks με μορφή διαγράμματος οντοτήτων-συσχετίσεων (ERD). 323
Εικόνα 13.4 Σχεσιακό σχήμα της βάσης δεδομένων myBooks (έχει παραχθεί με την εφαρμογή app.diagrams.net). 324
Εικόνα 13.5 Στιγμιότυπο από το περιβάλλον διαχείρισης pgAdmin4 κατά τη δημιουργία της βάσης δεδομένων books στο RDBMS PostgreSQL, αριστερά η επιλογή του πίνακα books, δεξιά η γραφική αναπαράσταση των πινάκων. 332
Εικόνα 13.6 Σύνδεση στη Heroku Postgres: (α) το αίτημα σύνδεσης, (β) η οθόνη με τα στοιχεία της νέας σύνδεσης. 337
Εικόνα 14.1 Αυθεντικοποίηση χρήστη χρησιμοποιώντας κάποιο είδος διαπιστευτηρίων. 344
Εικόνα 14.2 Κάποια από τα περιεχόμενα της ιστοσελίδας μπορεί να βρίσκονται σε προστατευμένη περιοχή. Μόνο χρήστες που έχουν αυθεντικοποιηθεί μπορούν να έχουν πρόσβαση. 344
Εικόνα 14.3 Η κρυπτογράφηση ενός μηνύματος παράγει κείμενο ανάλογου μεγέθους με το αρχικό. 345
Εικόνα 14.4 Ο μονόδρομος κατατεμαχισμός (one-way hash) παράγει αλφαριθμητικό σταθερού μήκους. 345
Εικόνα 14.5 Η αυθεντικοποίηση γίνεται συγκρίνοντας τις περιλήψεις. 346

Εικόνα 14.6 Κατατεμαχισμός με αλάτι. Η περίληψη θα προκύψει από συνθηματικό + αλάτι, όπου το αλάτι είναι γνωστό στο σύστημα και διαφορετικό για κάθε συνθηματικό. 347
Εικόνα 14.7 Μετά την επιτυχή αυθεντικοποίηση, παράγεται ένα μοναδικό "session ID" (sid) (2) και φυλάσσεται για περιορισμένο χρόνο στον εξυπηρετητή (3). Σε ένα sid μπορεί να αντιστοιχούν μεταβλητές συνεδρίας. 350
Εικόνα 14.8 Το session ID επιστρέφεται στον πελάτη (4). Ο πελάτης το αποθηκεύει σε ένα cookie. Το cookie περιέχει, εκτός από το session ID, και πληροφορίες σχετικά με το domain στο οποίο ισχύει, τον χρόνο ισχύος του κ.ά. (5).....350
Εικόνα 14.9 Στο εξής, σε κάθε αίτημα του φυλλομετρητή στέλνεται και το cookie (6). Ο εξυπηρετητής θεωρεί πως τα αιτήματα που γίνονται σε κοντινά χρονικά διαστήματα (για παράδειγμα, 15 λεπτά) ανήκουν στην ίδια συνεδρία. 351
Εικόνα 14.10 Στην αποσύνδεση (logout) (7) ή όταν το διάστημα μεταξύ δύο διαδοχικών αιτημάτων ξεπεράσει κάποιο όριο, το session ID διαγράφεται από τον εξυπηρετητή (9) και πλέον το cookie δεν θα αναγνωρίζεται. 351
Εικόνα 14.11 Ο φυλλομετρητής Firefox προειδοποιεί έντονα όταν πρόκειται να υποβάλουμε ευαίσθητα δεδομένα με μια φόρμα με το πρωτόκολλο HTTP. 357
Εικόνα 14.12 Η χρήση HTTPS απαιτεί ο εξυπηρετητής να διαθέτει ψηφιακό πιστοποιητικό και ζεύγος κλειδιών με το οποίο να κρυπτογραφείται η επικοινωνία. 358

Κεφάλαιο 1. Εισαγωγή

Σύνοψη

Στο πρώτο αυτό εισαγωγικό κεφάλαιο εστιάζουμε στα κύρια χαρακτηριστικά του διαδικτύου με ιδιαίτερη έμφαση στον παγκόσμιο ιστό (World Wide Web) και το πρωτόκολλο επικοινωνίας υπολογιστών που τον αφορά, το πρωτόκολλο μετάδοσης υπερκειμένου ***HyperText Transfer Protocol (HTTP)****. Με βάση τη γενική αυτή εικόνα, θα δοθεί μια συνοπτική περιγραφή των κεφαλαίων που ακολουθούν, που θα μας βοηθήσουν να αναπτύξουμε μια σύνθετη διαδικτυακή εφαρμογή. Αρχή και βασική τεχνολογία κάθε διαδικτυακής εφαρμογής είναι η γλώσσα σήμανσης υπερκειμένου (HyperText Markup Language, HTML) που αποτελεί αντικείμενο των επόμενων κεφαλαίων. Επίσης, στην αρχική ενότητα που ακολουθεί περιέχονται οδηγίες χρήσης του βιβλίου ως διδακτικού εγχειριδίου.*

Προαπαιτούμενη γνώση

Για τη μελέτη αυτού του κεφαλαίου δεν υπάρχει προαπαιτούμενη γνώση.

1.1 Προοίμιο

Το διαδίκτυο αποτελεί μια τεχνολογική υποδομή που έχει επηρεάσει τον τρόπο ζωής σε ένα μεγάλο τμήμα του πλανήτη μας, είναι ο βασικός μοχλός της 4ης βιομηχανικής επανάστασης και πάνω στην υποδομή αυτή έχουν αναπτυχθεί εφαρμογές που διευκολύνουν την επικοινωνία, την εργασία, την εκπαίδευση, το εμπόριο, τις οικονομικές συναλλαγές και πολλές άλλες πλευρές της σύγχρονης ζωής.

Η κατανόηση της αρχιτεκτονικής των σύγχρονων εφαρμογών του διαδικτύου είναι σημαντική για τους επιστήμονες των υπολογιστών και της πληροφορικής αλλά και για το ευρύ κοινό. Το βιβλίο αυτό απευθύνεται κυρίως σε φοιτητές τμημάτων Υπολογιστών και Πληροφορικής, για να χρησιμοποιηθεί ως διδακτικό βοήθημα ενός εξαμηνιαίου μαθήματος με τίτλο «Εισαγωγή στον προγραμματισμό του Διαδικτύου» ή «Εισαγωγή στον προγραμματισμό του Παγκόσμιου Ιστού». Στόχος είναι η εισαγωγή στις τρεις βασικές τεχνολογίες του Ιστού, HTML, CSS, JavaScript. Επιχειρείται επίσης εισαγωγή στην αρχιτεκτονική του διαδικτύου, ξεκινώντας από το μοντέλο πελάτη/εξυπηρετητή και το πρωτόκολλο HTTP. Επίσης, στο πλαίσιο των μαθημάτων θα γίνει περιγραφή σύγχρονων τεχνολογιών που επιτρέπουν και διευκολύνουν την ανάπτυξη εφαρμογών, όπως η βιβλιοθήκη Bootstrap που επεκτείνει τη CSS και την JavaScript στην πλευρά του φυλλομετρητή, ενώ στην πλευρά του εξυπηρετητή βιβλιοθήκες της JavaScript, όπως η Node.js, το Express.js κ.λπ.

Ο φοιτητής που έχει μελετήσει τα θέματα του βιβλίου και έχει εκπονήσει τις σχετικές εργασίες θα είναι σε θέση αφενός να κατανοεί τις τεχνολογίες που χρησιμοποιούνται στις διαδικτυακές εφαρμογές σήμερα και διέπουν γενικότερα τη λειτουργία του διαδικτύου, αφετέρου να σχεδιάσει, να αναπτύξει και να ελέγξει την ορθότητα λειτουργίας μιας σύνθετης διαδικτυακής εφαρμογής, που περιλαμβάνει ένα σύνολο διασυνδεομένων ιστοσελίδων, που υποστηρίζει διαφορετικούς ρόλους χρηστών και επιτρέπει τη μόνιμη αποθήκευση δεδομένων. Ο φοιτητής θα πρέπει να είναι σε θέση να κατανοεί τα θέματα ασφάλειας που σχετίζονται με αυτή την εφαρμογή και να ικανοποιεί τις απαιτήσεις ευχρηστίας και προσβασιμότητας.

Το βιβλίο οργανώνεται σε 14 κεφάλαια, το καθένα από τα οποία αντιστοιχεί, όπως περιγράφεται στη συνέχεια, συνήθως σε ένα εβδομαδιαίο μάθημα. Κάθε κεφάλαιο συνοδεύεται ακόμη από προτάσεις για πρακτικές ασκήσεις που μπορούν να αποτελέσουν το εργαστηριακό σκέλος του μαθήματος.

Στο σύγγραμμα αυτό έχουμε ενσωματώσει την εμπειρία διδασκαλίας του αντικειμένου για πάνω από είκοσι χρόνια στο Πανεπιστήμιο Πατρών. Το αντικείμενο του μαθήματος άλλαξε ριζικά με την πάροδο του χρόνου, αφού είμαστε υποχρεωμένοι να κάνουμε συνεχείς τροποποιήσεις ώστε να συμβαδίσουμε με τις τεχνολογικές εξελίξεις στο αντικείμενο του προγραμματισμού διαδικτυακών εφαρμογών, καθώς και να αντιμετωπίσουμε με όσο καλύτερο τρόπο την πρόκληση εισαγωγής των φοιτητών μας στον σύντομο χρόνο ενός εξαμηνιαίου μαθήματος, σε ένα αρκετά σύνθετο και απαιτητικό τεχνολογικό πεδίο. Τα τελευταία χρόνια, με την καθιέρωση των HTML5, CSS3 και JavaScript ES6, το τεχνολογικό πεδίο είναι πλέον αρκετά ώριμο και αποτελεί τη βάση για πιο σύνθετα εργαλεία. Το μάθημα εστιάζει σε αυτές ακριβώς τις τεχνολογίες.

Επίσης, μια καινοτομία, που έχουμε εισαγάγει στο σχετικό μάθημα και στο βιβλίο αυτό, είναι να ακολουθήσουμε προσέγγιση βασιζόμενοι πλήρως στη γλώσσα JavaScript. Στο παρελθόν, στις διδακτικές

ενότητες που αφορούν τον εξυπηρετητή, κάναμε εισαγωγή στη γλώσσα PHP, μία σύνθετη τεχνολογία με ευρεία διάδοση. Διαπιστώσαμε όμως ότι η κάλυψη του αντικειμένου ήταν επιφανειακή, δεδομένου του περιορισμένου χρόνου. Συνεπώς, αποφασίσαμε να διαθέσουμε περισσότερο χρόνο στην JavaScript δίνοντας βαρύτητα στην τεχνολογία αυτή που έχει ωριμάσει, ιδιαίτερα με την έκδοση ES6, και η χρήση της επεκτείνεται ραγδαία και στον εξυπηρετητή. Όπως θα διαπιστώσει ο αναγνώστης, προτείνονται κεφάλαια για προχωρημένα θέματα JavaScript, όπως ασύγχρονη λειτουργία και διαχείριση συμβάντων, ενώ διατίθεται επαρκής χρόνος για περιγραφή της Node.js και του πλαισίου Express.js στον εξυπηρετητή.

Σε ένα τυπικό εξαμηνιαίο μάθημα ελληνικού πανεπιστημίου, προτείνουμε να διατεθεί 1 εβδομάδα για το πρωτόκολλο HTTP και την αρχιτεκτονική του διαδικτύου (κεφάλαιο 1), 2 εβδομάδες για εισαγωγή στην HTML (κεφάλαια 2 και 3), 2 εβδομάδες για εισαγωγή στη CSS και το BootStrap (κεφάλαια 4, 5, 6), 4 εβδομάδες για εισαγωγή στην JavaScript (κεφάλαια 7, 8, 9, 10) και, τέλος, 4 εβδομάδες για εισαγωγή στον προγραμματισμό στον εξυπηρετητή με Node.js/Express.js και άλλες βιβλιοθήκες της JavaScript που χρειάζονται για αυθεντικοποίηση του χρήστη, σύνδεση στη βάση δεδομένων, διαχείριση συνεδρίας κ.λπ. (κεφάλαια 11, 12, 13, 14).

Άλλα σενάρια διδασκαλίας είναι: *Σενάριο έμφασης στον φυλλομετρητή*: Σε αυτό το σενάριο δίνεται περισσότερος χρόνος στα κεφάλαια 2-6, καθώς και 7-9 και λιγότερη έμφαση στα κεφάλαια 10 έως 14. Σε αυτή την περίπτωση θα πρέπει το υλικό να συμπληρωθεί με αρχές σχεδίασης ιστοσελίδων / ευχρηστία διαδραστικών συστημάτων.

Τέλος, ένα άλλο σενάριο είναι το μάθημα να *εστιάσει στον εξυπηρετητή*. Στην περίπτωση αυτή προτείνεται να αφιερωθούν 2 εβδομάδες για την αρχιτεκτονική και το HTTP, να αφιερωθεί 1 εβδομάδα για HTML και 1 ακόμη για CSS και Bootstrap, 4 εβδομάδες για την JavaScript, ενώ οι υπόλοιπες 5 εβδομάδες να αφιερωθούν για εμβάθυνση στο node.js και τις τεχνολογίες του εξυπηρετητή.

Αναπόσπαστο τμήμα αυτής της προσέγγισης είναι το μάθημα να συνοδεύεται από πρακτικές εργασίες (εργαστηριακές ασκήσεις) κάθε βδομάδα, και μια ατομική ή ομαδική εργασία (πρότζεκτ) που απαιτεί χρόνο περίπου 8 εβδομάδων. Η εργασία έχει τελικό στόχο την ανάπτυξη και διάθεση σε δημόσια ιστοσελίδα μιας πλήρους διαδικτυακής εφαρμογής, η οποία υποστηρίζει πολλαπλούς ρόλους χρηστών, και χρήση όλων των τεχνολογιών που καλύπτονται στο μάθημα.

Τα εργαλεία που απαιτούνται για την παρακολούθηση του μαθήματος και ιδιαίτερα την εκπόνηση των ασκήσεων και της εργασίας είναι όλα ελεύθερα διαθέσιμα, προτείνεται η χρήση του Visual Studio Code για ανάπτυξη του κώδικα ή, εναλλακτικά, κάποιου άλλου σύγχρονου εργαλείου ανάπτυξης κώδικα (sublime, atom κ.λπ.), το περιβάλλον ανάπτυξης (developer tools) που είναι ενσωματωμένο στις πιο πρόσφατες εκδόσεις των φυλλομετρητών Chrome και Firefox, το Github για την αποθήκευση του κώδικα της εργασίας, και η πλατφόρμα Heroku για τη φιλοξενία της εφαρμογής, ή κάποια άλλη πλατφόρμα που επιτρέπει τη φιλοξενία διαδικτυακών εφαρμογών.

Το μάθημα για το οποίο έχει γραφτεί το τρέχον εγχειρίδιο διδάσκεται στο 4ο ή 5ο έτος σπουδών και προϋποθέτει να έχουν προηγηθεί άλλα μαθήματα, όπως εισαγωγή στον προγραμματισμό (π.χ. Python και Java), εισαγωγή στους αλγόριθμους και σε δομές δεδομένων, εισαγωγή στις βάσεις δεδομένων (σχεσιακό μοντέλο). Επίσης, χρήσιμο μάθημα είναι ένα μάθημα αλληλεπίδρασης ανθρώπου-υπολογιστή, που εστιάζει σε κανόνες ευχρηστίας και προσβασιμότητας, με ιδιαίτερη έμφαση στη σχεδίαση και στην ανάπτυξη διαδικτυακών εφαρμογών. Παρ' ότι το μάθημα προϋποθέτει εξοικείωση με βασικές έννοιες της επιστήμης και της τεχνολογίας υπολογιστών, προσπαθούμε να καλύψουμε τις απαιτήσεις και του αρχάριου αναγνώστη, με απλά εισαγωγικά παραδείγματα και ασκήσεις. Το αντικείμενο του μαθήματος, εξάλλου, συχνά προκαλεί την περιέργεια και του μη ειδικού, αφού το διαδίκτυο έχει υπεισέλθει σε κάθε πτυχή της ζωής μας.

Το βιβλίο αυτό συνοδεύουν τρία μαθήματα στην πλατφόρμα mathesis.cup.gr, με αντικείμενο την ανάπτυξη ιστοσελίδων, ενώ οι ασκήσεις και τα παραδείγματα που περιγράφονται στο βιβλίο είναι διαθέσιμα από τη σχετική ιστοσελίδα του βιβλίου στο Github.

1.2 Εισαγωγή στην αρχιτεκτονική του διαδικτύου

Το διαδίκτυο ως τεχνολογική υποδομή κλείνει ζωή 40 χρόνων. Τα πρωτόκολλα TCP/IP ορίστηκαν στα μέσα της δεκαετίας του 1970, ενώ τα πρώτα μηνύματα ηλεκτρονικού ταχυδρομείου στάλθηκαν πριν από πάνω από 50 χρόνια (1972). Ο παγκόσμιος ιστός, η εφαρμογή του διαδικτύου που συνέτεινε αποφασιστικά στη σημερινή του εξάπλωση και χρήση, έχει συμπληρώσει 30 χρόνια ζωής από τη δημιουργία του το 1992 από τον Tim Berners-Lee. Συνεπώς, υπάρχουν σήμερα ολόκληρες γενιές πολιτών, ιδιαίτερα στις δυτικές κοινωνίες, που δεν έχουν γνωρίσει τον κόσμο και δεν μπορούν να φανταστούν τον κόσμο χωρίς το διαδίκτυο.

Το διαδίκτυο στον ανεπτυγμένο κόσμο, όπου η πρόσβαση αγγίζει το 90% του πληθυσμού, θεωρείται βασική υποδομή, όπως τα δίκτυα ενέργειας και τηλεπικοινωνίας. Η διάδοση στη χρήση του συνεχίζει να αυξάνεται ραγδαία, αφού και ο αναπτυσσόμενος κόσμος αυξάνει ταχύτατα τη χρήση του. Μάλιστα, σε συνδυασμό με την αυξημένη φορητότητα στις υπολογιστικές συσκευές των χρηστών και στην ευρεία διάδοση ασύρματων δικτύων υψηλής ταχύτητας, η πρόσβαση σε διαδικτυακούς πόρους αποτελεί πλέον αναπόσπαστο τμήμα της καθημερινής μας ζωής, στο σπίτι, στη δουλειά, στις κοινωνικές συναναστροφές.

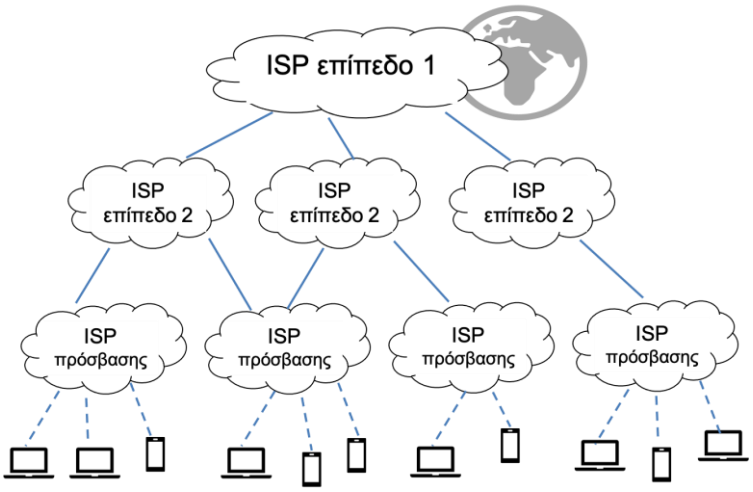
Θα ξεκινήσουμε το εισαγωγικό αυτό κεφάλαιο με μια περιγραφή βασικών εννοιών της αρχιτεκτονικής του διαδικτύου.

1.2.1 Η αρχιτεκτονική του διαδικτύου

Το **διαδίκτυο** είναι ένα σύνολο από διασυνδεδεμένα δίκτυα υπολογιστών, τα οποία χρησιμοποιούν τα πρότυπα πρωτόκολλα που είναι γνωστά ως **TCP/IP** (Πρωτόκολλο Ελέγχου Μετάδοσης, Transmission Control Protocol, TCP / Πρωτόκολλο Διαδικτύου, Internet Protocol, IP). Στα πρωτόκολλα αυτά θα γίνει ιδιαίτερη αναφορά σε επόμενη ενότητα.

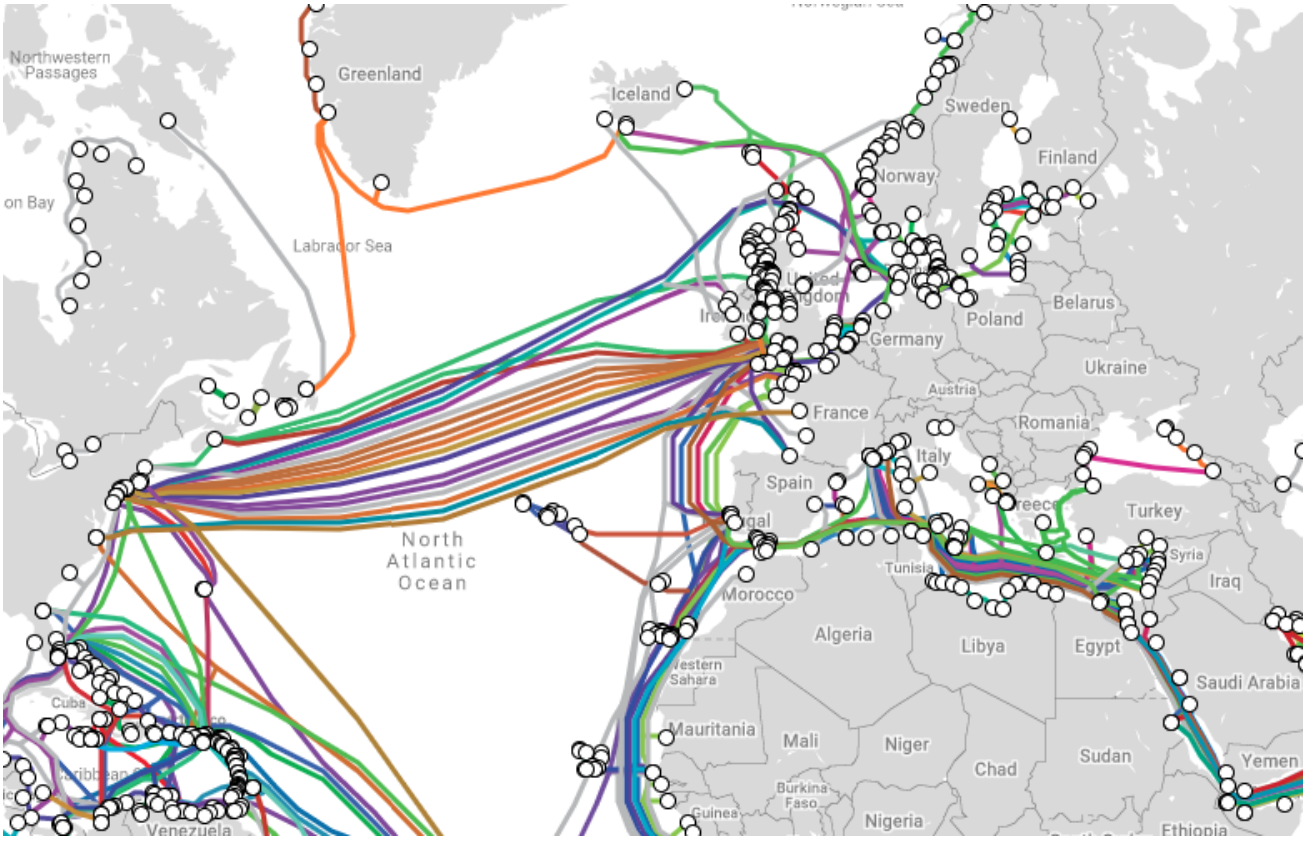
Μια σύντομη παρουσίαση της αρχιτεκτονικής του διαδικτύου θα πρέπει να κάνει αναφορά στα δίκτυα **Παρόχων Υπηρεσιών Διαδικτύου** (Internet Service Providers, ISP), τα οποία ορίζουν μια ιεραρχία, με καθένα από τα δίκτυα αυτά να έχει διαφορετικό ρόλο και τεχνικά χαρακτηριστικά, όπως η ταχύτητα μετάδοσης και ο όγκος πληροφορίας που διακινούν. Στην κορυφή αυτής της ιεραρχίας είναι ένα σύνολο από δίκτυα που συναποτελούν τον **κορμό του διαδικτύου** (Internet backbone). Πρόκειται για σχετικά μικρό αριθμό δικτύων υψηλής ταχύτητας και χωρητικότητας, τα λεγόμενα δίκτυα ISP πρώτου επιπέδου (βλέπε **Εικόνα 1.1**). Σε αυτά τα δίκτυα συνδέονται δίκτυα με πιο τοπικό χαρακτήρα και διαφορετικά χαρακτηριστικά όσον αφορά τις δυνατότητές τους (δίκτυα ISP επιπέδου 2). Τα δίκτυα των δύο αυτών κατηγοριών στην ουσία αποτελούν δίκτυα εξυπηρετητών (server), οι οποίοι παρέχουν τη βασική τηλεπικοινωνιακή υποδομή του διαδικτύου.

Στους εξυπηρετητές αυτούς συνδέονται τα δίκτυα **πρόσβασης ISP** (access ISP), τα οποία παρέχουν υπηρεσίες πρόσβασης στο διαδίκτυο σε μεμονωμένους πελάτες (τερματικούς σταθμούς, host). Οι τερματικοί σταθμοί μάλιστα δεν είναι σε όλες τις περιπτώσεις κλασσικοί υπολογιστές, αλλά σήμερα μπορεί να είναι κάμερες ασφαλείας, οικιακές συσκευές, αυτοκίνητα κ.λπ. που διαθέτουν σύνδεση στο διαδίκτυο.



Εικόνα 1.1 *Αρχιτεκτονική του διαδικτύου ως ιεραρχία δικτύων ISP (παρόχων υπηρεσιών διαδικτύου).*

Η επικοινωνία μεταξύ δύο τερματικών σταθμών πιθανόν να εμπλέξει ενδιάμεσους εξυπηρετητές που ανήκουν σε διαφορετικά δίκτυα ISP. Το πρώτο πρόβλημα που παρουσιάζεται είναι πώς θα επικοινωνήσουν δύο συνδεδεμένοι υπολογιστές. Σημαντική παράμετρος είναι η ύπαρξη μοναδικών διευθύνσεων όλων των συνδεδεμένων κόμβων. Στη συνέχεια θα αναφερθούμε στον τρόπο διευθυνσιοδότησης των υπολογιστών στο διαδίκτυο.



Εικόνα 1.2 *Χάρτης των υποβρύχιων καλωδίων που παρέχει η ιστοσελίδα https://www.submarinecablemap.com/. Οι συνδέσεις αυτές, που έχουν ποντιστεί στον βυθό της θάλασσας ή άλλες αντίστοιχες στην ξηρά, περιλαμβάνουν οπτικές ίνες μετάδοσης δεδομένων υψηλής ταχύτητας που συνδέονται με κόμβους που αποτελούν τον κορμό του διαδικτύου.*

1.2.2 Διευθύνσεις υπολογιστών διαδικτύου

Το διαδίκτυο στηρίζεται σε έναν μηχανισμό μοναδικών διευθύνσεων των διασυνδεδεμένων συσκευών, είτε πρόκειται για εξυπηρετητές είτε για τερματικούς σταθμούς. Αυτές οι διευθύνσεις είναι γνωστές ως διευθύνσεις **IP** (**Internet Protocol**). Σύμφωνα με το πρότυπο **IPv4**, μια διεύθυνση IP έχει μήκος 32 bits, που σημαίνει ότι ο συνολικός αριθμός διευθύνσεων με αυτό το πρότυπο δεν μπορεί να είναι μεγαλύτερος από 2^{32}, λίγο περισσότερες από 4 δισεκατομμύρια διευθύνσεις.

Οι διευθύνσεις IPv4 απαρτίζονται από 4 τμήματα των 8 bits (το κάθε τμήμα παίρνει τιμές 0-255), π.χ. 172.217.18.100. Τα τμήματα αυτά ορίζουν ιεραρχικά: την κατηγορία του δικτύου, τη διεύθυνση του δικτύου και υποδικτύου (ορίζεται από τη μάσκα υποδικτύου), και τη διεύθυνση του τερματικού σταθμού (host).

Η **μάσκα υποδικτύου** καθορίζει τον αριθμό bits που κατανέμονται στη διεύθυνση υποδικτύου και εκείνων που αφορούν τη διεύθυνση του υπολογιστή. Αν n bits αφορούν τη διεύθυνση του υπολογιστή, τότε 32-n bits διατίθενται για τη διεύθυνση του δικτύου και υποδικτύου.

Με τη χρήση μάσκας υποδικτύου είναι δυνατόν σε κάποιο δίκτυο να ορίσουμε για τα διαφορετικά υποδίκτυα το πλήθος των υπολογιστών που αυτά περιλαμβάνουν, ανάλογα με τις ανάγκες του οργανισμού ιδιοκτήτη του δικτύου. Είναι φανερό από τα παραπάνω ότι τα τελευταία n bits μιας διεύθυνσης είναι η ταυτότητα του υπολογιστή, που βρίσκεται στο πιο δεξιά τμήμα της διεύθυνσης.

Συνεπώς, με τη μάσκα υποδικτύου καθορίζουμε ποιο τμήμα της διεύθυνσης IP είναι η διεύθυνση δικτύου (netid) και ποιο τμήμα είναι η διεύθυνση του υπολογιστή (hostid). Η μάσκα είναι ένας αριθμός 32 bits που έχει ψηφίο 1 για netid και 0 για hostid.

1.2.3 Ασκήσεις

Άσκηση 1

Ζητείται να βρεθεί το μέγεθος ενός υποδικτύου του οποίου η μάσκα είναι η 255.255.255.192

Απάντηση

Η μάσκα σε δυαδική αναπαράσταση είναι:

```
11111111 11111111 11111111 11000000
```

Αυτό σημαίνει ότι οι διευθύνσεις των υπολογιστών του δικτύου αυτού έχουν μήκος 6 bits, άρα συνολικά το δίκτυο αυτό μπορεί να έχει $2^6 = 64$ υπολογιστές.

Άσκηση 2

Έστω υπολογιστής με διεύθυνση IP 150.251.110.200 που ανήκει στο παραπάνω δίκτυο. Ποια είναι η ταυτότητα του υπολογιστή αυτού (hostid);

Απάντηση

Επειδή, όπως είπαμε πιο πάνω, τα τελευταία 6 bits της διεύθυνσης αφορούν την ταυτότητα του υπολογιστή, αρκεί να εξετάσουμε το τελευταίο byte της διεύθυνσης (200) και να το μετατρέψουμε στον αντίστοιχο δυαδικό αριθμό, $200_{10} = 11001000$, Τα 6 τελευταία bits του αριθμού αυτού, και συνεπώς τα bits που αντιστοιχούν σε hostid, είναι 001000, που είναι και η διεύθυνση του υπολογιστή.

1.2.4 Η έκδοση IPv6

Δεδομένης της μεγάλης ανάπτυξης του διαδικτύου και συνεπώς των απαιτήσεων για μοναδικές διευθύνσεις, είμαστε σε σταδιακή μετάβαση στην έκδοση 6 του πρωτοκόλλου IP (**IPv6**) που προβλέπει διευθύνσεις 128 bits, άρα μέχρι 2^{128} διευθύνσεις, ένας αριθμός συντριπτικά μεγαλύτερος από αυτόν του IPv4 (340 τρισ. τρισ. τρισ. διευθύνσεις).

Μια διεύθυνση IPv6 είναι της εξής μορφής:

```
2001:0db8:85a3:0000:0000:8a2e:0370:7334
```

Παρατηρούμε ότι ο αριθμός αναπαρίσταται από οκτώ (8) τετράδες δεκαεξαδικών ψηφίων, χωρισμένων με το σύμβολο ":", η καθεμία από τις οποίες αντιστοιχεί σε 16 bits.

Ο οργανισμός που συντονίζει τη λειτουργία του διαδικτύου και καθορίζει το εύρος συνεχόμενων διευθύνσεων που διατίθεται σε κάθε περιοχή του διαδικτύου είναι ο ICANN, Internet Corporation for Assigned Names and Numbers. Η διαχείριση των διευθύνσεων γίνεται σε συνεργασία με την Internet Assigned Numbers Authority (IANA), που συντηρεί τους σχετικούς καταλόγους.

1.2.5 Σύστημα ονομασίας περιοχών DNS

Δεδομένου ότι η αναπαράσταση διευθύνσεων των συσκευών που βρίσκονται συνδεδεμένες στο διαδίκτυο με δυαδικούς ή δεκαδικούς/δεκαεξαδικούς αριθμούς δεν είναι εύχρηστες για τους ανθρώπους, υπάρχει ένας εναλλακτικός μηχανισμός διευθυνσιοδότησης υπολογιστών στο διαδίκτυο, που στηρίζεται σε μια ιεραρχική συγκρότηση μνημονικών διευθύνσεων, οργανωμένων σε «περιοχές» (domains).

Στο πιο υψηλό επίπεδο της ιεραρχίας αυτής έχουμε συνήθως γεωγραφικές περιοχές, όπως η περιοχή "**.gr**" για διευθύνσεις στην Ελλάδα. Σε καθεμία από τις περιοχές αυτές ορίζονται υποπεριοχές με ευθύνη του ιδιοκτήτη της περιοχής υψηλού επιπέδου, και κατοχυρώνονται μοναδικά ονόματα που εκπροσωπούν φορείς ιδιωτικούς ή δημόσιους, αλλά ακόμη και φυσικά πρόσωπα. Κάποιες διευθύνσεις, εύκολες στην απομνημόνευση, μπορεί να αποτελέσουν αντικείμενο εμπορικών δοσοληψιών όταν γίνονται διαθέσιμες.

Οι γεωγραφικές περιοχές που έχουν κατοχυρωθεί στο υψηλότερο επίπεδο (δύο χαρακτήρες ASCII, σύμφωνα με την προτυποποίηση ISO 3166-1) είναι 255, μεταξύ των οποίων και περιοχές όπως η .eu για την Ευρωπαϊκή Ένωση. Τα τελευταία χρόνια δόθηκε η δυνατότητα χρήσης χαρακτήρων πέραν του ASCII στα

ονόματα περιοχών, με αποτέλεσμα να έχουν κατοχυρωθεί 316 γεωγραφικές περιοχές (στοιχεία του 2020), μεταξύ των οποίων και η περιοχή "**.ελ**", για την Ελλάδα.

Πέραν αυτών, υπάρχουν ακόμη οι επτά αρχικές περιοχές, εκ των οποίων οι τρεις είναι γενικής χρήσης: **.com** - commercial (ιδιωτικές επιχειρήσεις), **.org** - organization (οργανισμοί), **.net** - network (δίκτυα), ενώ υπάρχουν τέσσερις ακόμη με περιορισμούς στη χρήση: **.int** (περιορισμένο σε διακρατικές ενώσεις), **.edu** - education (κύρια αμερικανικά πανεπιστήμια), **.gov** - government (αμερικανική κυβέρνηση), **.mil** (αμερικανικός στρατός).

Τα τελευταία χρόνια έχει αναπτυχθεί ένας μακρύς κατάλογος από περιοχές πρώτου επιπέδου δημόσια διαθέσιμες ή δεσμευμένες από επιχειρήσεις, για παράδειγμα η περιοχή **.bank** για τον τραπεζικό τομέα, **.museum** για μουσεία κ.λπ.

Στην ελληνική περιοχή ".gr" επιτρέπεται η εισαγωγή νέων ονομάτων δευτέρου η τρίτου επιπέδου, μετά από αίτηση στην αρμόδια αρχή. Στο δεύτερο επίπεδο υπάρχουν κατοχυρωμένα ονόματα, όπως το gov.gr για δημόσιες υπηρεσίες, sch.gr για το σχολικό δίκτυο και edu.gr για εκπαιδευτικούς οργανισμούς.

Μια διεύθυνση με το σύστημα αυτό έχει το πιο ειδικό στοιχείο στο αριστερό άκρο, ενώ το πιο γενικό στο δεξί άκρο, αντίθετα από τις διευθύνσεις IP που είδαμε στην προηγούμενη ενότητα. Έτσι, η διεύθυνση www.ece.upatras.gr ορίζει ότι στην περιοχή .gr έχει οριστεί η υποπεριοχή δευτέρου επιπέδου upatras (Πανεπιστήμιο Πατρών), στην περιοχή αυτή έχει οριστεί η περιοχή ece (Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών) στο πλαίσιο της οποίας υπάρχει ο εξυπηρετητής www.

Μια διεύθυνση με το σύστημα αυτό αντιστοιχεί σε μια ή περισσότερες διευθύνσεις IP. Για παράδειγμα, στο τερματικό (ή γραμμή εντολών στο λειτουργικό Windows) μπορούμε να δώσουμε την εντολή:

```
$ ping www.ece.upatras.gr
64 bytes from 150.140.189.12: time=18.285 ms
```

Σημειώνεται ότι η εντολή ping εκτελεί ένα πρόγραμμα για έλεγχο διαθεσιμότητας ενός εξυπηρετητή και μέτρησης του χρόνου απόκρισής του.

Από την απόκριση αντιλαμβανόμαστε ότι η συγκεκριμένη μνημονική διεύθυνση αντιστοιχεί σε διεύθυνση IP 150.140.189.12.

Καλείστε να χρησιμοποιήσετε την εντολή αυτή για να ελέγξετε τη διεύθυνση IP γνωστών διευθύνσεων εξυπηρετητών διαδικτύου.

Το ερώτημα είναι πώς ένας υπολογιστής που θέλει να στείλει ένα μήνυμα σε έναν εξυπηρετητή (π.χ. όταν πληκτρολογούμε www.google.com στον φυλλομετρητή μας) θα ξέρει σε ποια διεύθυνση IP να στείλει το μήνυμα;

Το διαδίκτυο διαθέτει μια υπηρεσία αντίστοιχη των τηλεφωνικών καταλόγων, που διατηρεί πίνακες αντιστοίχισης μνημονικών ονομάτων σε διευθύνσεις IP. Η υπηρεσία αυτή είναι κατανεμημένη ιεραρχικά και στηρίζεται στην ύπαρξη **εξυπηρετητών ονομάτων** (**name servers**), που διαθέτουν καταλόγους, οι οποίοι συντηρούνται συνήθως από τους εξυπηρετητές δικτύων ISP. Η αναζήτηση στους εξυπηρετητές ονομάτων λέγεται **DNS lookup**.

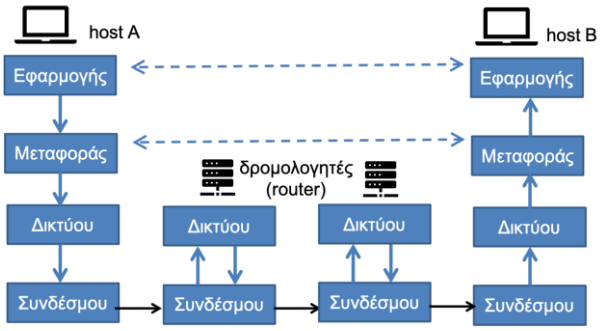
1.2.6 Πρωτόκολλα του διαδικτύου

Η επικοινωνία υπολογιστών απαιτεί κοινά πρωτόκολλα, δηλαδή συμβάσεις για τον τρόπο επικοινωνίας και τον τρόπο δόμησης της πληροφορίας που ανταλλάσσεται.

Το διαδίκτυο στηρίζεται σε ένα σύνολο πρωτοκόλλων οργανωμένων σε επίπεδα. Διακρίνονται συγκεκριμένα 4 επίπεδα, που φαίνονται στην ***Εικόνα 1.3***.

Κάθε επίπεδο επικοινωνεί με εκείνο που βρίσκεται πιο πάνω ή πιο κάτω από αυτό, ανάλογα με τη φορά διάδοσης της πληροφορίας.

Ας υποθέσουμε ότι δύο υπολογιστές (host A και host B) επικοινωνούν. Συγκεκριμένα, ο host A αποστέλλει δεδομένα στον host B. Τα δεδομένα παράγονται από μια διαδικτυακή εφαρμογή στον A, η εφαρμογή (που βρίσκεται στο **επίπεδο εφαρμογής - application layer**) είναι υπεύθυνη για επικοινωνία με τον χρήστη. Ανάλογα με την εφαρμογή, χρησιμοποιούνται διαφορετικά πρωτόκολλα. Το πιο συνηθισμένο είναι το πρωτόκολλο HTTP για επικοινωνία της εφαρμογής του παγκόσμιου ιστού, ενώ υπάρχουν ακόμη το πρωτόκολλο SMTP για το ηλεκτρονικό ταχυδρομείο, VoIP για ομότιμη σύνδεση και αποστολή δεδομένων φωνής κ.λπ. Συνήθως, οι εφαρμογές του επιπέδου αυτού ακολουθούν το μοντέλο πελάτη/εξυπηρετητή. Το επίπεδο εφαρμογής μεταφέρει τα δεδομένα στο πιο κάτω επίπεδο, που είναι το επίπεδο μεταφοράς, για να επικοινωνήσει με την αντίστοιχη εφαρμογή στον υπολογιστή B.



Εικόνα 1.3 *Διαστρωμάτωση επιπέδων πρωτοκόλλων του διαδικτύου.*

Το **επίπεδο μεταφοράς (transport layer)** διαιρεί τα δεδομένα που πρέπει να αποσταλούν σε μικρά πακέτα, τα αριθμεί ώστε να είναι δυνατή η επανασύνθεσή τους στον παραλήπτη και τα μεταφέρει στον παρακάτω επίπεδο δικτύου. Στο επίπεδο μεταφοράς έχουμε δύο πρωτόκολλα:

- Το **Πρωτόκολλο ελέγχου μετάδοσης TCP (Transmission Control Protocol)**, το οποίο εγκαθιστά μια σύνδεση με τον απομακρυσμένο υπολογιστή πριν την έναρξη αποστολής δεδομένων (ορίζεται η θύρα σύνδεσης), ενώ αν κάποιο μήνυμα δεν παραδοθεί επαναποστέλλεται.
- Το **Πρωτόκολλο αυτοδύναμων πακέτων χρήστη (User Datagram Protocol, UDP)**, το οποίο είναι πιο γρήγορο από το TCP, αφού δεν εγκαθιστά σύνδεση και δεν κάνει έλεγχο παραλαβής των απεσταλμένων πακέτων. Ο κίνδυνος απώλειας πληροφορίας για κάποιες εφαρμογές, όπως αυτές της ροής πολυμέσων και της μετάδοσης φωνής, δεν είναι απαγορευτικός, ενώ η γρήγορη μετάδοση είναι το κυρίως ζητούμενο.

Το επόμενο επίπεδο είναι το **Επίπεδο δικτύου (Internet Protocol IP)**, το οποίο παραλαμβάνει τα πακέτα που ετοίμασε το επίπεδο μεταφοράς και αναλαμβάνει να τα δρομολογήσει στον επόμενο υπολογιστή, ώστε να προωθηθούν εγγύτερα στον προορισμό τους με χρήση πινάκων δρομολόγησης. Τα πακέτα είναι παντελώς ανεξάρτητα και πιθανόν να ακολουθήσουν διαφορετικές διαδρομές.

Όταν αποφασιστεί ο επόμενος υπολογιστής στον οποίο θα αποσταλεί το πακέτο, αυτό προωθείται στο πιο κάτω επίπεδο, το **Επίπεδο συνδέσμου δεδομένων (data link layer)**, το οποίο αναλαμβάνει την αποστολή του πακέτου, ανάλογα με το δίκτυο στο οποίο είναι συνδεδεμένος ο υπολογιστής, Ethernet, WiFi κ.λπ. Στους ενδιάμεσους υπολογιστές (δρομολογητές), το μήνυμα παραλαμβάνεται στο επίπεδο συνδέσμου, μεταφέρεται στο επίπεδο δικτύου και ελέγχεται ο πίνακας δρομολόγησης ώστε να αποφασιστεί ο υπολογιστής στον οποίο πρέπει να προωθηθεί το πακέτο. Τέλος, στον σταθμό παραλήπτη το μήνυμα μεταφέρεται αντίστροφα, από το επίπεδο συνδέσμου προς το επίπεδο εφαρμογής.

1.2.7 Εφαρμογές του διαδικτύου

Το διαδίκτυο χρησιμοποιείται σήμερα από πολλές εφαρμογές, κάποιες από τις οποίες έχουν οριστεί από τα πρώτα χρόνια του διαδικτύου, κάποιες είναι πιο πρόσφατες. Η εφαρμογή που κυριαρχεί και έχει συμβάλει στη διάδοση της χρήσης του είναι βεβαίως ο **παγκόσμιος ιστός (World Wide Web)**, που θα αποτελέσει αντικείμενο ξεχωριστών ενοτήτων στη συνέχεια.

Ας κάνουμε μια σύντομη επισκόπηση άλλων εφαρμογών εδώ. Το **Ηλεκτρονικό ταχυδρομείο (email)** είναι μια από τις πιο παλιές αλλά παραμένει ακόμη μια πολύ δημοφιλής εφαρμογή του διαδικτύου. Η εφαρμογή αυτή, όπως και πολλές άλλες στο διαδίκτυο, ακολουθεί το μοντέλο πελάτη/εξυπηρετητή. Ο **Εξυπηρετητής αλληλογραφίας (email server)** λαμβάνει μηνύματα από τους χρήστες που έχουν εγγραφεί σε αυτό και τα προωθεί στους εξυπηρετητές των αποδεκτών του κάθε μηνύματος. Τα μηνύματα αποθηκεύονται στον εξυπηρετητή αυτό και ο αποδέκτης μπορεί να συνδεθεί στον εξυπηρετητή του για να δει τα μηνύματα που έχουν έλθει για αυτό.

Τα πρωτόκολλα που σχετίζονται με το email αφορούν τη λήψη και την αποστολή email. Το **Απλό Πρωτόκολλο Μεταφοράς Ταχυδρομείου (Simple Mail Transfer Protocol, SMTP)** ευθύνεται για την αποστολή ενός μηνύματος από εξυπηρετητή σε εξυπηρετητή, αλλά και για την αποστολή του μηνύματος από τον υπολογιστή του χρήστη στον εξυπηρετητή του. Για πρόσβαση στην αλληλογραφία ενός χρήστη (mailbox)

μπορεί να χρησιμοποιηθούν δύο πρωτόκολλα:

- Το **Post Office Protocol version 3 (POP3)**, το πιο απλό από τα δύο, που απλά κατεβάζει το μήνυμα στον υπολογιστή του χρήστη.
- Το **Internet Mail Access Protocol (IMAP)**, το οποίο επιτρέπει στον χρήστη να αποθηκεύει και να χειρίζεται μηνύματα ταχυδρομείου στον εξυπηρετητή.

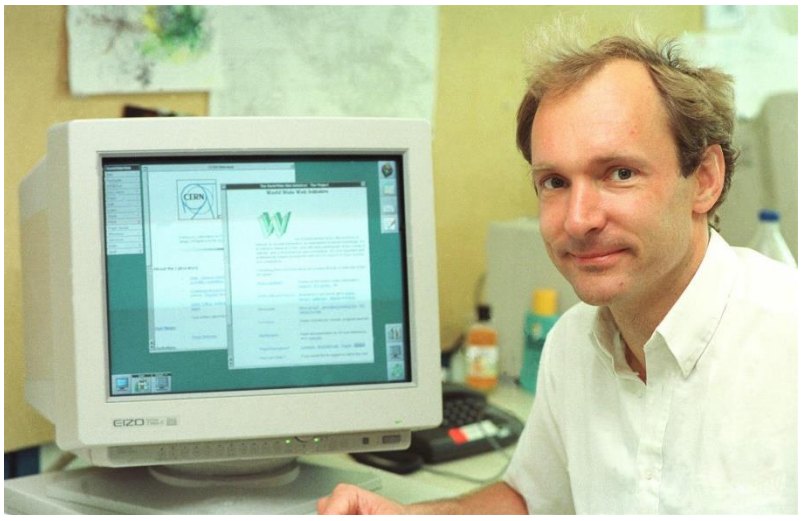
Ένα ακόμη πρωτόκολλο με μεγάλη χρήση στην εφαρμογή αυτή, αλλά και με γενικότερη χρήση για κωδικοποίηση πληροφορίας, είναι το **Multipurpose Internet Mail Extension (MIME)**. Το πρωτόκολλο αυτό αντιμετωπίζει το πρόβλημα που έχει το SMTP, που έχει σχεδιαστεί για μεταφορά μηνυμάτων με κωδικοποίηση ASCII και επιτρέπει την αποστολή πολυμέσων, κειμένου με κωδικοποίηση utf-8 κ.λπ.

Το δεύτερο παράδειγμα εφαρμογών είναι πιο πρόσφατο και σχετίζεται με εφαρμογές επικοινωνίας, όπως τα Skype, Viber, WhatsApp, Messenger κ.λπ. Όλες αυτές οι εφαρμογές χρησιμοποιούν την υποδομή του διαδικτύου για να επικοινωνήσουν (κλήση) με μια άλλη συνδεδεμένη συσκευή και στη συνέχεια να εγκαταστήσουν μια σύνδεση για την αποστολή μιας ροής δεδομένων ήχου ή/και βίντεο. Τα πρωτόκολλα που χρησιμοποιούν στηρίζονται στο **Voice over Internet Protocol (VoIP)** που επιτρέπει τη χρήση του διαδικτύου για παροχή φωνητικής επικοινωνίας μέσω ομότιμης σύνδεσης (peer connection). Τα δεδομένα αποστέλλονται με χρήση του πρωτοκόλλου UDP του επιπέδου μεταφοράς για αποστολή πακέτων φωνής/βίντεο, ενός πρωτοκόλλου το οποίο, όπως περιγράφηκε σε προηγούμενη ενότητα, επιτυγχάνει μεγαλύτερη ταχύτητα στη μετάδοση δεδομένων από το συνηθισμένο TCP που χρησιμοποιείται στον παγκόσμιο ιστό, αλλά δεν διασφαλίζει μη απώλεια δεδομένων.

Το διαδίκτυο, όπως γνωρίζουμε, τείνει να υποκαταστήσει άλλα μέσα, όπως το ραδιόφωνο και την τηλεόραση, και προς την κατεύθυνση αυτή συμβάλλουν εφαρμογές **πολυεκπομπής (multicasting)**, οι οποίες απαιτούν από τους δρομολογητές του διαδικτύου να κατευθύνουν ροές δεδομένων προς πολλαπλούς αποδέκτες, κάτι που η νέα έκδοση του πρωτοκόλλου IPv6 υποστηρίζει σε μεγάλο βαθμό.

1.3 Ο παγκόσμιος ιστός

Ο παγκόσμιος ιστός ως εφαρμογή του διαδικτύου δημιουργήθηκε στις αρχές της δεκαετίας του 1990. Στην εφεύρεση της τεχνολογίας αυτής συνέβαλε σε μεγάλο βαθμό ο Βρετανός ερευνητής τότε του Κέντρου Θεωρητικής Φυσικής CERN στην Ελβετία Tim Berners-Lee (**Εικόνα *1.4***).



Εικόνα 1.4 *Ο Tim Berners-Lee μπροστά στον πρώτο φυλλομετρητή ιστού.* http://cds.cern.ch/record/39437#31

Ο ερευνητής πρότεινε τον συνδυασμό της έννοιας του **υπερκειμένου (hypertext)** με το διαδίκτυο. Για τον λόγο αυτό, εισήγαγε τρεις τεχνολογίες που επιτρέπουν τη διασύνδεση κειμένων που βρίσκονται σε διάφορους εξυπηρετητές του διαδικτύου μέσω **υπερσυνδέσμων (hyperlinks)**. Οι τεχνολογίες αυτές ήταν το πρωτόκολλο μεταφοράς υπερκειμένων **HyperText Transfer Protocol HTTP**, η γλώσσα σύνταξης υπερκειμένων **HyperText Markup Language HTML** και ο μηχανισμός ονοματοδοσίας των πόρων του ιστού **Universal Resource Locator URL**. Με τις τεχνολογίες αυτές, οι οποίες έκτοτε αναπτύχθηκαν και αποτελούν και σήμερα βασικά συστατικά των εφαρμογών του διαδικτύου, θα ασχοληθούμε στη συνέχεια του βιβλίου. Βεβαίως,

σήμερα μια ιστοσελίδα δεν περιλαμβάνει μόνο κείμενο, αλλά και πολυμέσα, ενώ η έλευση της γλώσσας προγραμματισμού JavaScript το 1995 έδωσε στην αλληλεπίδραση με μια ιστοσελίδα πιο διαδραστικό χαρακτήρα.

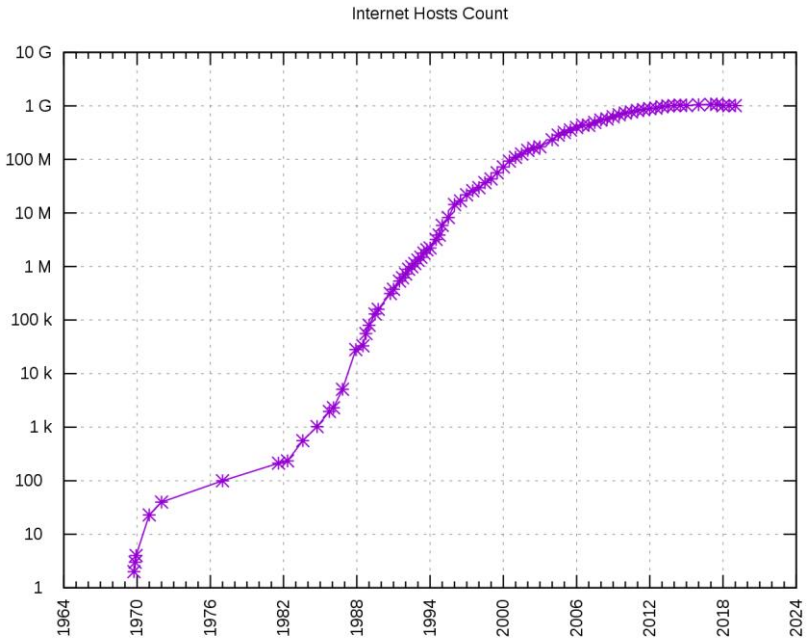
Τι είναι ένα υπερκείμενο; Είναι ένα σύνολο από έγγραφα, το καθένα από τα οποία περιέχει ενσωματωμένους συνδέσμους προς άλλα έγγραφα. Η έννοια του υπερκειμένου ήταν γνωστή πολύ πριν το διαδίκτυο, όμως με το πρωτόκολλο HTTP έγινε δυνατή η διασύνδεση των εγγράφων τα οποία βρίσκονται σε διαφορετικούς υπολογιστές του διαδικτύου, δημιουργώντας στην ουσία έναν ενιαίο πληροφοριακό χώρο.

Η αλληλεπίδραση ενός χρήστη με ένα υπερκείμενο περιλαμβάνει την επιλογή ενός υπερσυνδέσμου με το ποντίκι συνήθως, ενέργεια που οδηγεί αυτόματα στην κλήση και το φόρτωμα της ιστοσελίδας-στόχου. Η ανάκτηση του εγγράφου αυτού γίνεται με χρήση του πρωτοκόλλου HTTP. Η διαδοχική φόρτωση διασυνδεδεμένων «σελίδων» υπερκειμένου παράγει την αίσθηση της «πλοήγησης» στον παγκόσμιο ιστό.

Η αρχιτεκτονική της εφαρμογής του ιστού είναι αυτή του πελάτη/εξυπηρετητή, όπως εξάλλου οι περισσότερες εφαρμογές του διαδικτύου. Ο ίδιος ο Berners-Lee ανέπτυξε τον πρώτο πελάτη της εφαρμογής, τον **φυλλομετρητή ιστού (web browser)**, και ο πρώτος αυτός φυλλομετρητής ονομάστηκε WorldWideWeb. Είναι ένα λογισμικό που λειτουργεί στο επίπεδο εφαρμογών του διαδικτύου και το οποίο με χρήση του πρωτοκόλλου HTTP αποστέλλει αιτήματα προς έναν εξυπηρετητή, παραλαμβάνει την απόκριση και την παρουσιάζει στον χρήστη. Επίσης, ο Berners-Lee ανέπτυξε τον πρώτο **εξυπηρετητή ιστού (web server)**, ο οποίος ονομάστηκε CERN httpd. Ο εξυπηρετητής είναι το λογισμικό που δέχεται αιτήματα από τον πελάτη και αποστέλλει περιεχόμενο, συνήθως ιστοσελίδες γραμμένες σε HTML. Σήμερα ένας εξυπηρετητής μπορεί να αποστείλει πολλές διαφορετικές μορφές δεδομένων, όπως θα δούμε στη συνέχεια.

Στη γρήγορη διάδοση του παγκόσμιου ιστού συνέβαλε το γεγονός ότι η ιδέα εξαρχής διατέθηκε ελεύθερα, και άλλοι ερευνητές την ανέπτυξαν περαιτέρω. Η εφαρμογή του παγκόσμιου ιστού συνέτεινε στη ραγδαία ανάπτυξη χρήσης του διαδικτύου. Σήμερα οι περισσότεροι χρήστες του διαδικτύου είναι χρήστες ουσιαστικά της εφαρμογής του παγκόσμιου ιστού και για τον λόγο αυτό οι όροι διαδίκτυο και ιστός συγχέονται.

Στην **Εικόνα *1.5*** φαίνεται η εκθετική αύξηση των διασυνδεδεμένων υπολογιστών στο διαδίκτυο. Από το γράφημα προκύπτει ότι η περίοδος 1980-90 ήταν η περίοδος με τη μεγαλύτερη αύξηση.



Εικόνα 1.5 *Πλήθος εξυπηρετητών διασυνδεδεμένων στο διαδίκτυο, περίοδος 1970-2020.*
https://commons.wikimedia.org/wiki/File:Internet_Hosts_Count_log.svg

1.3.1 Ο εξυπηρετητής ιστού

Ένας **εξυπηρετητής ιστού (web server)** είναι μια εφαρμογή που δέχεται αιτήματα μέσω του πρωτοκόλλου

HTTP ή της ασφαλούς παραλλαγής του HTTPS. Μόλις ένας πελάτης, τυπικά ένας φυλλομετρητής, κάνει αίτημα για έναν πόρο προς τον εξυπηρετητή, το αίτημα θα πρέπει να ικανοποιηθεί, αν ο πόρος είναι διαθέσιμος, ή να σταλεί ένα μήνυμα σφάλματος. Επίσης, ο εξυπηρετητής παραλαμβάνει δεδομένα από τον χρήστη, αν αυτό επιτρέπεται, και τα αποθηκεύει στον υπολογιστή του εξυπηρετητή.

Εργασίες που αναλαμβάνει ο εξυπηρετητής είναι η μετάφραση του URL path σε όνομα αρχείου στο σύστημα αρχείων του εξυπηρετητή, η αυθεντικοποίηση του χρήστη, η διαχείριση της ροής δεδομένων για να αποφεύγεται κορεσμός του διαδικτύου και να είναι δυνατή η εξυπηρέτηση πολλών πελατών.

Ας δούμε ένα παράδειγμα μετάφρασης URL path. Αν ο εξυπηρετητής λάβει ένα αίτημα για τον πόρο:

```
http://www.example.com/path/file.html
```

Ο εξυπηρετητής ιστού στο www.example.com θα προσαρτήσει τη δεδομένη διαδρομή στη διαδρομή του καταλόγου της ρίζας του ιστότοπου (host). Σε ένα εξυπηρετητή Apache, αυτός είναι συνήθως /home/www/website. Το αποτέλεσμα είναι ο πόρος του τοπικού συστήματος αρχείων

```
/home/www/www.example.com/path/file.html
```

να αποσταλεί ως απάντηση στο αίτημα.

Η ιστορία των εξυπηρετητών, όπως ήδη αναφέρθηκε, ξεκίνησε από τον **CERN HTTPd** (HTTP deamon). Μια βελτιωμένη έκδοση αναπτύχθηκε το 1993 στο Εθνικό Κέντρο Εφαρμογών Υπερυπολογιστών NCSA στο Πανεπιστήμιο Illinois Urbana-Champaign, με το όνομα **NCSA HTTPd**. Στο πλαίσιο αυτού του εξυπηρετητή, δόθηκε η δυνατότητα, μέσω των Common Gateway Interface (CGI), διασύνδεσης με ένα εξωτερικό πρόγραμμα (γραμμένο σε οποιαδήποτε γλώσσα), και συνεπώς η δυναμική εξυπηρέτηση αιτημάτων ιστοσελίδων.

Όταν η ανάπτυξη αυτού του εξυπηρετητή σταμάτησε, ο κώδικας μεταφέρθηκε στο πρότζεκτ ανοικτού κώδικα **Apache**, που σύντομα έγινε ο πιο δημοφιλής εξυπηρετητής και έχει διατηρήσει αυτή τη θέση σχεδόν ως τις μέρες μας. Το όνομα αναφέρεται στην ινδιάνικη φυλή των Απάτσι, φημισμένη για τις δεξιότητές τους στη στρατηγική του πολέμου και την ανεξάντλητη αντοχή τους, αλλά κάνει επίσης ένα χαριτωμένο λογοπαίγνιο για έναν εξυπηρετητή που έχει φτιαχτεί με διαδοχικά «μπαλώματα» (patches).

Σήμερα υπάρχουν αρκετές διαφορετικές επιλογές εξυπηρετητών, μεταξύ των πιο δημοφιλών είναι ο Apache που ήδη αναφέρθηκε, ο Nginx που, σύμφωνα με κάποιες στατιστικές, είναι ο πιο δημοφιλής σήμερα εξυπηρετητής, ο IIS της εταιρείας Microsoft κ.λπ.

Μια εφαρμογή εξυπηρέτησης αιτημάτων ιστού, όπως θα δούμε στη συνέχεια, μπορεί να γραφτεί σε οποιαδήποτε γλώσσα προγραμματισμού. Στα κεφάλαια 11 ως 14 θα δούμε βήμα προς βήμα την ανάπτυξη του εξυπηρετητή μιας εφαρμογής ιστού με χρήση της γλώσσας JavaScript. Εκεί θα χρησιμοποιήσουμε το περιβάλλον Node.js και στη συνέχεια το πλαίσιο Express.js. Στο Node.js μπορούμε να ενσωματώσουμε έναν εξυπηρετητή ιστού με χρήση της βιβλιοθήκης (module) http ως εξής:

```
const  http = require('http');

http.createServer(function (req, res) {
  res.write('Καλή σας μέρα!');
  res.end();
}).listen(8080); // ο εξυπηρετητής αποκρίνεται στη θύρα 8080
```

Ένας εξυπηρετητής ιστού περιέχεται επίσης στο Express.js που θα χρησιμοποιήσουμε στο κεφάλαιο 12.

Τέλος, ένα ακόμη παράδειγμα είναι ο εξυπηρετητής που περιέχεται στο πλαίσιο Flask για ανάπτυξη εφαρμογών ιστού με χρήση της Python.

Ένα παράδειγμα:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return '<h1>καλή σας μέρα!</h1>'
app.run(debug=True) # ο εξυπηρετητής αποκρίνεται στη θύρα 5000
```

Στην περίπτωση αυτή η εφαρμογή χρησιμοποιεί τον εξυπηρετητή που είναι ενσωματωμένος στο πλαίσιο Flask

και είναι επαρκής για εξυπηρέτηση αιτημάτων κατά την ανάπτυξη της εφαρμογής (ένα αίτημα κάθε φορά). Όμως, όταν η εφαρμογή διατεθεί σε παραγωγική χρήση, ένας άλλος εξυπηρετητής πρέπει να χρησιμοποιηθεί, όπως ο Gunicorn ή ο Nginx ή ο Apache, μέσω της διεπαφής mod_wsgi.

Μια συζήτηση για το θέμα αυτό μπορείτε να αναζητήσετε στο reddit.com.

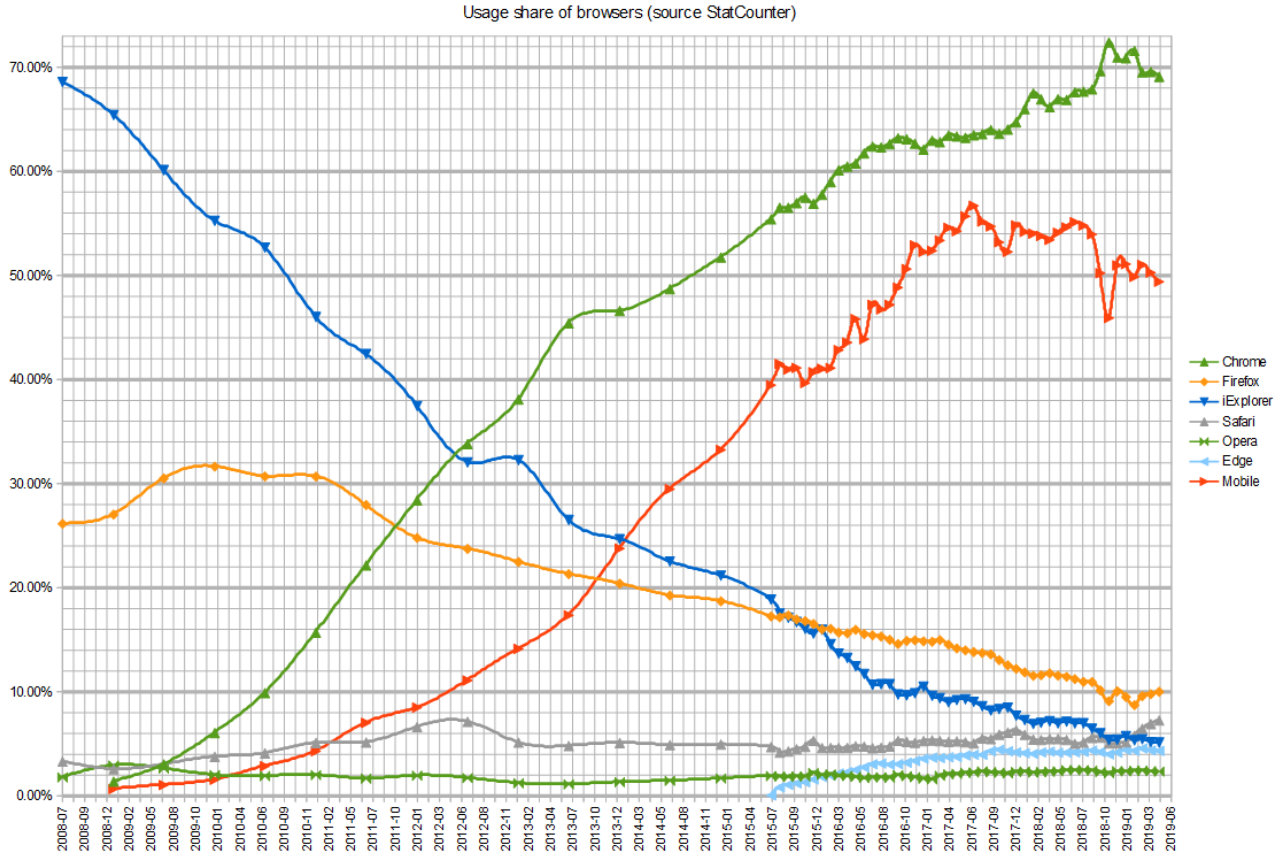
1.3.2 Ο φυλλομετρητής ιστού

Στην ενότητα αυτή θα δούμε τα κύρια χαρακτηριστικά των φυλλομετρητών ιστού (web browsers).

Ιστορικά, ο πρώτος φυλλομετρητής δημιουργήθηκε από τον ίδιο τον εφευρέτη του ιστού Berners-Lee το 1990 και ονομάστηκε **WorldWideWeb**. Στο ερευνητικό κέντρο NCSA του Πανεπιστημίου Illinois Urbana-Champaign, ο Marc Andreessen, τότε νέος ερευνητής, πήρε την ιδέα του πρώτου φυλλομετρητή και την εξέλιξε, κυρίως με δυνατότητες γραφικής διεπαφής, για να δημιουργηθεί ο **NCSA Mosaic**, ο πρώτος φυλλομετρητής με σημαντική διάδοση. Ο ίδιος ο Andreessen σύντομα άφησε το πανεπιστήμιο για να δημιουργήσει μια εταιρεία που εξέλιξε περαιτέρω την ιδέα, για να δημιουργήσει τον φυλλομετρητή **Netscape Navigator**, που υπήρξε ο πρώτος φυλλομετρητής με τεράστια διάδοση, αλλά και συνεισφορά στην ανάπτυξη της τεχνολογίας των φυλλομετρητών και των εφαρμογών διαδικτύου γενικότερα. Για παράδειγμα, η γλώσσα προγραμματισμού JavaScript, που θα μας απασχολήσει σε σημαντικό βαθμό στο βιβλίο αυτό, προέρχεται από την ομάδα του Netscape (1995).

Ο φυλλομετρητής αυτός το 1995 είχε το 90% των χρηστών του διαδικτύου, όμως έχασε στον λεγόμενο «πρώτο πόλεμο των φυλλομετρητών» από τον ανταγωνισμό του **Internet Explorer** της εταιρείας Microsoft, και έφτασε το 2006 να χρησιμοποιείται μόνο από το 1% των χρηστών. Θα πρέπει να σημειωθεί ότι ο Netscape Navigator διατέθηκε στη συνέχεια δημόσια και αποτέλεσε τη βάση του κώδικα του οργανισμού Mozilla, που στη συνέχεια ανέπτυξε τον φυλλομετρητή **Firefox** με βάση τη μηχανή σχεδίασης ιστοσελίδας Gecko.

Η διαμάχη μεταξύ των κατασκευαστών φυλλομετρητών ακολούθησε και την εξέλιξη της τεχνολογίας, και ιδιαίτερα τα νέα πρότυπα HTML5, CSS3, καθώς και την εκτεταμένη χρήση φορητών συσκευών που επηρέασε τις επιλογές των χρηστών.



Εικόνα 1.6 *Ο δεύτερος πόλεμος των φυλλομετρητών: ποσοστό χρήσης των φυλλομετρητών κατά την περίοδο 2008-2019.* *https://commons.wikimedia.org/wiki/File:BrowserUsageShare.png*

Ο «δεύτερος πόλεμος των φυλλομετρητών», που αφορά την περίοδο από τα μέσα της δεκαετίας του 2000 μέχρι σήμερα, είδε την πτώση του Internet Explorer από το 95% της αγοράς που κατείχε το 2003 σε κάτω από το 1% σήμερα. Την ίδια περίοδο αναπτύχθηκε ο **Mozzila Firefox** και στη συνέχεια ο **Google Chrome**, ο οποίος σήμερα είναι ο πιο δημοφιλής φυλλομετρητής, με πάνω από το 70% των χρηστών.

Ο Chrome χρησιμοποιεί τη μηχανή παρουσίασης της ιστοσελίδας **Blink**, η οποία βασίστηκε στη **WebKit**, που είχε αναπτυχθεί στον φυλλομετρητή Safari και η οποία με τη σειρά της βασίστηκε στη μηχανή **KHTML**, που αναπτύχθηκε από τον οργανισμό ανοιχτού κώδικα KDE. Ο Chrome χρησιμοποιεί τη μηχανή εκτέλεσης κώδικα JavaScript **V8**, που επίσης είναι η μηχανή JavaScript που τρέχει στο περιβάλλον Node.js, εκτός του φυλλομετρητή. Τα τελευταία χρόνια η υποδομή του Chrome αποτελεί τη βασική τεχνολογική βάση για πολλούς δημοφιλείς φυλλομετρητές, όπως του Microsoft Edge ή του Opera, και συνεπώς η Blink και η V8 έχουν μεγάλη διάδοση και χρήση.

Οι φυλλομετρητές των φορητών συσκευών (έξυπνα τηλέφωνα και ταμπλέτες) έχουν σημαντικές διαφορές από αυτούς των επιτραπέζιων υπολογιστών, ενώ η πρόσβαση στο διαδίκτυο μέσω των συσκευών αυτών αυξάνει συνεχώς.

Θα πρέπει να σημειώσουμε ότι σήμερα οι κύριοι φυλλομετρητές είναι συμβατοί με τις πιο σημαντικές προδιαγραφές των προτύπων HTML5, CSS3, ES6. Όμως, παραμένει ένας αριθμός χρηστών σε παλαιότερες συσκευές με εκδόσεις φυλλομετρητών που δεν υποστηρίζουν τα πρότυπα. Ο σχεδιαστής μιας διαδικτυακής εφαρμογής βρίσκεται πάντα μπροστά στο δίλημμα να χρησιμοποιήσει τις πιο τελευταίες δυνατότητες της τεχνολογίας, μη επιτρέποντας σε ένα μικρό ποσοστό των χρηστών να έχει πρόσβαση στην εφαρμογή του, ή να παραμείνει σε παλαιότερες τεχνολογίες. Η ιστοσελίδα caniuse επιτρέπει να ελέγξουμε τη διάδοση μιας συγκεκριμένης τεχνολογίας σε διαφορετικές εκδόσεις φυλλομετρητών.

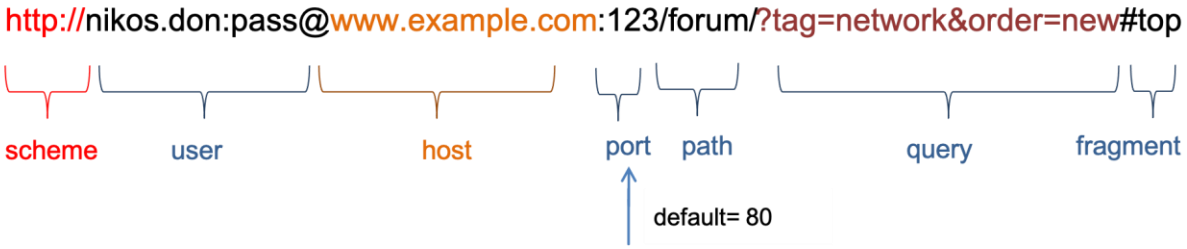
1.3.3 Οι διευθύνσεις URL

Ως τώρα έχουμε διερευνήσει τη βασική λειτουργία και τις τρέχουσες τεχνολογίες των εξυπηρετητών ιστού και των φυλλομετρητών ιστού. Ωστόσο, η επικοινωνία μεταξύ των δύο αυτών λογισμικών δεν θα ήταν δυνατή αν δεν είχε οριστεί ένα πρωτόκολλο μοναδικής ονομασίας ενός διαδικτυακού πόρου, όπου πόρος ορίζεται ένα έγγραφο HTML, μια εικόνα ή ακόμη και μια παράγραφος σε ένα έγγραφο, ώστε να μπορέσει ο πελάτης να ζητήσει τον πόρο από τον εξυπηρετητή.

Η μοναδική αυτή ονομασία ενός πόρου ονομάζεται **Ενιαίος Εντοπιστής Πόρων (Uniform Resource Locator, URL)**.

Το όνομα αυτό περιέχει πληροφορίες για (α) τον εξυπηρετητή στον οποίο είναι αποθηκευμένος ο πόρος (εδώ θα χρησιμοποιήσουμε τα μνημονικά ονόματα των υπολογιστών που ορίζονται από το σύστημα ονομασίας περιοχών DNS), (β) το πρωτόκολλο μέσω του οποίου θα ανακτήσουμε τον πόρο, (γ) τη διαδρομή μέσα στο σύστημα αρχείων του εξυπηρετητή που έχει αποθηκευτεί ο πόρος, (δ) τον χρήστη ο οποίος ζητάει πρόσβαση, (ε) τη θύρα επικοινωνίας με την εφαρμογή, (ζ) πιθανές μεταβλητές και τις τιμές που τους έχουν δοθεί ως αποτέλεσμα συμπλήρωσης μιας φόρμας και τέλος (η) το επιμέρους στοιχείο του εγγράφου (παράγραφος). Από τα στοιχεία αυτά μόνο το όνομα του εξυπηρετητή είναι υποχρεωτικό.

Ας δούμε ένα παράδειγμα:



Εικόνα 1.7 *Παράδειγμα URL.*

Στο παράδειγμα αυτό βλέπουμε ένα πλήρες όνομα πόρου:

- http:// ορίζει ότι το πρωτόκολλο ανάκτησης του πόρου είναι το HTTP.
- nikos.don:pass το δεύτερο στοιχείο καθορίζει ότι ο χρήστης που ζητάει τον πόρο είναι ο nikos.don με μυστικό κωδικό pass.
- www.example.com είναι το μνημονικό όνομα του εξυπηρετητή στον οποίο διατίθεται ο πόρος.
- 123 είναι ο αριθμός της θύρας στην οποία αποκρίνεται ο εξυπηρετητής, συνήθως για τις εφαρμογές του

ιστού, η θύρα είναι η θύρα 80, οπότε αν δεν έχει καθοριστεί θεωρείται η προκαθορισμένη αυτή τιμή.
- /forum/ αυτή είναι η διαδρομή στο σύστημα αρχείων του εξυπηρετητή όπου θα αναζητηθεί ο πόρος, αν, όπως στο παράδειγμα, δεν υπάρχει όνομα αρχείου, αλλά μόνο φάκελος, τότε ο εξυπηρετητής κάνει υπόθεση για προκαθορισμένα ονόματα αρχείων, όπως index.html.
- ?tag=network&order=new με το τμήμα αυτό του URL περνάμε ζευγάρια μεταβλητών/τιμών στον εξυπηρετητή. Στο παράδειγμα έχουμε ότι tag = "network" και order = "new".
- #top το στοιχείο αυτό ορίζει ότι ζητάμε το στοιχείο με ταυτότητα #top του πόρου. Συνήθως οι φυλλομετρητές σε αυτές τις περιπτώσεις κάνουν κύλιση της σελίδας ώστε να τεθεί στην αρχή της σελίδας το στοιχείο με αυτή την ταυτότητα.

1.3.4 Ειδικά σύμβολα και κωδικοποίηση

Κάποιοι χαρακτήρες όπως &, +, % και ο κενός χαρακτήρας (space) αν υπάρχουν σε ονόματα πόρων ή διαδρομών κωδικοποιούνται ως "%xx", όπου xx είναι η τιμή ASCII σε δεκαεξαδική μορφή, π.χ. & = "%26", κενό = "%20". Οι παράμετροι που περιλαμβάνονται στο query string δίδονται ως λίστα παραμέτρων/τιμών που διαχωρίζονται με τον χαρακτήρα &.

```
var1=value1&var2=value2&var3=value3
```

1.3.5 Απλοποίηση URL

Στους περισσότερους φυλλομετρητές, αν παραληφθεί το πρωτόκολλο, υπονοείται το http:// ή το https://. Επίσης, αν παραληφθεί η διαδρομή (path), υπονοείται η ρίζα του καταλόγου, και μέσα σε αυτό ο προκαθορισμένος πόρος index.html, οπότε είναι συχνή πρακτική τα URL να περιέχουν μόνο το μνημονικό όνομα του εξυπηρετητή, π.χ. google.com.

1.4 Το πρωτόκολλο HTTP

Με βάση τη γενική συζήτηση που προηγήθηκε, προκύπτει η μεγάλη σημασία του πρωτοκόλλου HTTP για τις εφαρμογές του παγκόσμιου ιστού. Στην ενότητα αυτή θα δούμε τα κύρια χαρακτηριστικά του.

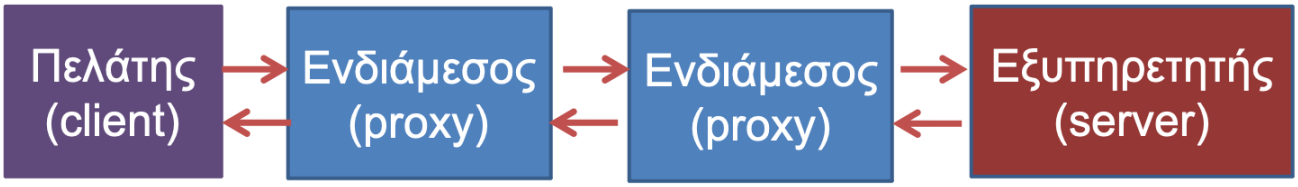
Το πρωτόκολλο **HyperText Transfer Protocol (HTTP)** είναι πρωτόκολλο επιπέδου εφαρμογών του διαδικτύου, που καθορίζει τον τρόπο που επικοινωνεί ένας πελάτης με τον εξυπηρετητή σε μια εφαρμογή του παγκόσμιου ιστού.

Οι εκδόσεις του HTTP είναι: HTTP 0.9 (1991), HTTP 1.0 (1996), HTTP 1.1 (1997), HTTP 2.0 (2015). Η πιο διαδεδομένη έκδοση είναι η HTTP 1.1, ενώ η HTTP 2.0 επιφέρει σημαντικές αλλαγές, όπως η κωδικοποίηση των μηνυμάτων σε δυαδική μορφή.

Τα κύρια χαρακτηριστικά του είναι:
- Ακολουθεί την αρχιτεκτονική Πελάτη-Εξυπηρετητή, συνεπώς υπάρχουν δύο είδη μηνυμάτων: αιτήματα και αποκρίσεις.
- Χρησιμοποιεί τις υπηρεσίες του πρωτοκόλλου TCP του επιπέδου μεταφοράς, ή μιας σύνδεσης TCP κρυπτογραφημένης με το πρωτόκολλο Transport Layer Security (TLS) για την περίπτωση του HTTPS (Ενότητα 14.5).
- Πρωτόκολλο ανταλλαγής δεδομένων σε μορφή κειμένου μέχρι την έκδοση 1.1.
- Από την έκδοση HTTP 1.1 επιτρέπονται επίμονες συνδέσεις TCP για πολλαπλές μεταφορές δεδομένων στην ίδια σύνδεση.
- Το πρωτόκολλο δεν θυμάται την προηγούμενη επικοινωνία (Stateless Protocol). Αυτή η ιδιότητα μπορεί να δημιουργήσει πρόβλημα σε μια εφαρμογή τύπου eshop στην οποία απαιτείται συνέχεια, π.χ., για το περιεχόμενο του καλαθιού αγορών. Για να αντιμετωπιστεί το πρόβλημα αυτό χρησιμοποιούνται συνεδρίες (Ενότητα 14.3).
- Από την έκδοση HTTP 1.1 επιτρέπει Proxy caching, δηλαδή τη χρήση διαμεσολαβητών για προσωρινή αποθήκευση περιεχομένου.
- Επιτρέπει τη διαπραγμάτευση του περιεχομένου (content negotiation) ως προς τα προτιμώμενα χαρακτηριστικά που θα έχει ο πόρος (γλώσσα, κωδικοποίηση κ.λπ.).

1.4.1 Διαχείριση ενδιάμεσων μνημών

Στην ανταλλαγή μηνυμάτων HTTP μεταξύ πελάτη και εξυπηρετητή μπορεί να παρεμβληθούν ενδιάμεσοι υπολογιστές οι οποίοι μπορεί να παίξουν διάφορους ρόλους. Μια εικόνα αυτής της αρχιτεκτονικής φαίνεται στη συνέχεια:



Εικόνα 1.8 *Σύνδεση πελάτη-εξυπηρετητή με παρεμβολή ενδιάμεσων.*

Παραδείγματα αυτών των ενδιαμέσων είναι:

- **Προσωρινή μνήμη**, είτε δημόσια ή ιδιωτική, όπως η προσωρινή μνήμη του φυλλομετρητή.
- **Φιλτράρισμα**, όπως η ανίχνευση ιών ή γονικοί έλεγχοι.
- **Εξισορρόπηση φορτίου**, για να επιτρέπεται σε διαφορετικούς υπολογιστές του εξυπηρετητή να εξυπηρετούν τα διαφορετικά αιτήματα.
- **Έλεγχος ταυτότητας**, για έλεγχο πρόσβασης σε διαφορετικούς πόρους.
- **Καταγραφή**, ώστε να αποθηκεύονται ιστορικές πληροφορίες.

Μέσω των μηνυμάτων HTTP μπορεί να ελεγχθεί η λειτουργία της επικοινωνίας πελάτη/εξυπηρετητή.

Μερικές από τις παραμέτρους που μπορεί να ελεγχθούν μέσω μηνυμάτων HTTP είναι:

- ενδιάμεση αποθήκευση (cashing),
- ταυτοποίηση του χρήστη,
- διαχείριση συνεδρίας με cookies,
- διαχείριση του περιορισμού same origin.

1.4.2 Η ροή επικοινωνίας HTTP

Ας παρακολουθήσουμε την επικοινωνία του φυλλομετρητή μας με έναν εξυπηρετητή, στον οποίο στέλνει ένα αίτημα εξυπηρέτησης, π.χ. να ζητήσει τον πόρο http://www.upatras.gr που είναι η κεντρική ιστοσελίδα του Πανεπιστημίου Πατρών στην ελληνική γλώσσα.

Θα γίνουν τα εξής βήματα:

- Θα ανοίξει μια **σύνδεση TCP** μεταξύ των δύο υπολογιστών. Όπως θα δούμε στη συνέχεια, η διαχείριση της σύνδεσης αυτής μπορεί να γίνει με διαφορετικούς τρόπους.
- Θα σταλεί το **αίτημα HTTP**. Το μήνυμα στην έκδοση HTTP 1.1 είναι αναγνώσιμο από τον χρήστη, ενώ από την έκδοση HTTP 2.0, αν και το περιεχόμενο παραμένει το ίδιο, το μήνυμα κωδικοποιείται σε πλαίσια δυαδικών δεδομένων (frames).

Το μήνυμα για το παράδειγμά μας είναι:

```
GET /el HTTP/1.1
Host: www.upatras.gr
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en;q=0.9,el-GR;q=0.8
```

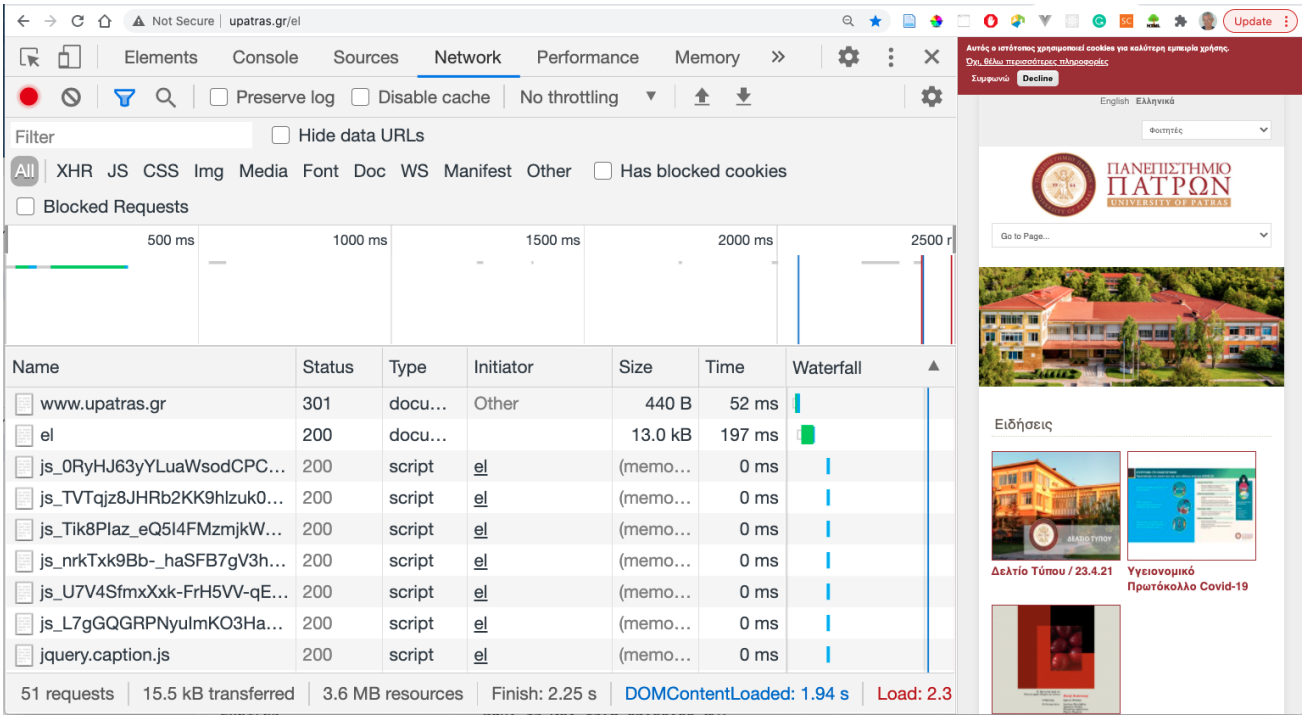
- Η απόκριση στο μήνυμα μας είναι:

```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Wed, 28 Apr 2021 19:00:28 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
```

```
Connection: keep-alive
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Cache-Control: no-cache, must-revalidate
X-Content-Type-Options: nosniff
Content-Language: el
X-Frame-Options: SAMEORIGIN
X-Generator: Drupal 7 (http://drupal.org)
Link: <http://www.upatras.gr/el>
X-XSS-Protection: 1; mode=block
Content-Encoding: gzip

<!DOCTYPE html> .... το υπόλοιπο της σελίδας
```

- Η σύνδεση αυτή θα χρησιμοποιηθεί και για τους άλλους πόρους της ιστοσελίδας.

Για να παρατηρήσουμε τη διαδικασία αυτή και να δούμε τα σχετικά μηνύματα HTTP, αρκεί να ανοίξουμε τα εργαλεία σχεδιαστή (επιλογή inspect) στον φυλλομετρητή Chrome ή Firefox και να επιλέξουμε την επιλογή "Network". Θα παρατηρήσουμε ότι για τη συγκεκριμένη ιστοσελίδα στάλθηκαν 51 αιτήματα HTTP και η συνολική διάρκεια που η σύνδεση παρέμεινε ανοικτή ήταν 2,25 δευτερόλεπτα.



Εικόνα 1.9 *Στα αριστερά φαίνεται η κατάσταση των αιτημάτων HTTP όπως εμφανίζονται στην καρτέλα Network των εργαλείων σχεδιαστή του Chrome. Ενώ στα δεξιά εμφανίζεται η ιστοσελίδα. Αν επιλέξουμε ένα από τα αιτήματα, μπορούμε να δούμε το raw HTTP μήνυμα (αίτημα και απόκριση).*

1.4.3 Αιτήματα εξυπηρέτησης HTTP

Ο πρώτος τύπος μηνυμάτων που θα δούμε είναι τα αιτήματα (requests) HTTP. Ένα αίτημα περιλαμβάνει:

- Τη **μέθοδο HTTP**, όπως GET, POST ή HEAD, η οποία καθορίζει τη λειτουργία που επιθυμεί ο πελάτης. Συνήθως, ένας πελάτης θέλει να ανακτήσει έναν πόρο (χρησιμοποιώντας την GET) ή να αποστείλει το περιεχόμενο μιας φόρμας HTML (χρησιμοποιώντας την POST).
- Τη **διαδρομή του προς ανάκτηση πόρου**. Αυτή είναι η διεύθυνση URL του πόρου, αφού έχουν αφαιρεθεί τα στοιχεία που είναι προφανή, όπως για παράδειγμα το πρωτόκολλο (http://), το μνημονικό όνομα του εξυπηρετητή (www.upatras.gr) ή η θύρα TCP (80).
- Την **έκδοση του πρωτοκόλλου HTTP**.

- Ακολουθούν οι **κεφαλίδες** του μηνύματος, εκ των οποίων η μόνη που είναι υποχρεωτική είναι η κεφαλίδα Host που αφορά τη διεύθυνση του εξυπηρετητή. Υπάρχουν ακόμη προαιρετικές κεφαλίδες που μεταφέρουν επιπλέον πληροφορίες στον εξυπηρετητή.
- Ακολουθεί το **σώμα του μηνύματος**, το οποίο για ορισμένες μεθόδους όπως η POST περιέχει τον πόρο που αποστέλλεται (όπως γίνεται και στα μηνύματα απόκρισης).

Οι διαφορετικές μέθοδοι (τύποι) αιτημάτων είναι:

- **GET**: ανάκτηση πόρου. Η μεγάλη πλειονότητα των αιτημάτων που διακινούνται είναι αυτού του τύπου.
- **HEAD**: ανάκτηση μεταδεδομένων της κεφαλίδας (π.χ., mod time).
- **POST**: υποβολή δεδομένων φόρμας στο σώμα του μηνύματος.
- **PUT**: αποθήκευση επισυναπτόμενου εγγράφου ως URI (νέο στο HTTP/ 1.1).
- **DELETE**: διαγραφή πόρου (resource) (νέο στο HTTP/ 1.1).
- **TRACE**: http “echo” για debugging (νέο στο HTTP/ 1.1).
- **CONNECT**: χρήση από proxies (νέο στο HTTP/ 1.1).
- **OPTIONS**: επιλογές του proxy (νέο στο HTTP/ 1.1).

Η δομή του μηνύματος αίτησης HTTP είναι η εξής:

Εικόνα 1.10 *Η δομή του μηνύματος αίτησης HTTP.*

Κεφαλίδες μηνυμάτων αίτησης

Οι κεφαλίδες ενός μηνύματος αίτησης είναι ζεύγη ιδιότητα-τιμή και καταλαμβάνουν μια ξεχωριστή γραμμή η καθεμία. Δεν υπάρχει διάκριση μεταξύ κεφαλαίων και πεζών χαρακτήρων.

Οι σπουδαιότερες κεφαλίδες είναι:

Host: Δείχνει το μνημονικό όνομα του εξυπηρετητή και της θύρας σύνδεσης TCP, όπως αυτά ορίζονται στο URL κλήσης του πελάτη. Είναι υποχρεωτική κεφαλίδα στην έκδοση HTTP/1.1.

Accept: Οι κεφαλίδες αυτού του τύπου καθορίζουν επιθυμητούς και αποδεκτούς τύπους δεδομένων από τον πελάτη. Οι διαφορετικοί τύποι διαχωρίζονται με κόμματα. Συχνά ακολουθούνται από έναν αριθμό μεταξύ 0 και 1, που ορίζει τη σχετική βαρύτητα του τύπου. Όταν δεν ορίζεται βαρύτητα, τότε θεωρείται η τιμή 1. Για παράδειγμα:

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

Υπάρχουν διάφοροι τύποι κεφαλίδων accept, **Accept-Encoding** (κωδικοποίηση), **Accept-Language** (γλώσσα), **Accept** (τύποι δεδομένων, κωδικοποίηση MIME, π.χ. text/html).

Connection: keep-alive (για επίμονες συνδέσεις) ή close (για βραχύβιες συνδέσεις).

Content-Length: xxxx, η κεφαλίδα αυτή χρησιμοποιείται μόνο σε αιτήσεις POST. Ο αριθμός xxxx εκφράζει το μέγεθος των αποστελλόμενων δεδομένων του μηνύματος σε bytes (δηλαδή χωρίς τις κεφαλίδες). Απαιτείται στο HTTP 1.1 για να καθορίζονται τα όρια τερματισμού και έναρξης κάθε ξεχωριστής αίτησης μέσα από μια επίμονη σύνδεση.

If-Modified-Since: <timestamp>. Η κεφαλίδα αυτή δηλώνει ότι ο πελάτης διαθέτει ήδη ένα αντίγραφο της σελίδας με χρονική σήμανση τροποποίησης timestamp και ζητάει από τον εξυπηρετητή να επιστρέψει την αιτούμενη σελίδα μόνο εφόσον αυτή έχει τροποποιηθεί μεταγενέστερα της δηλωμένης χρονικής στιγμής.

User-Agent: Περιγραφή φυλλομετρητή. Η κεφαλίδα αυτή περιέχει πληροφορίες σχετικά με τον πελάτη (φυλλομετρητή) που κάνει την αίτηση. Δεν χρησιμοποιείται πάντοτε, ενώ το περιεχόμενό της δεν μπορεί να θεωρείται αξιόπιστο.

1.4.4 Μήνυμα απόκρισης HTTP

Ο εξυπηρετητής αποκρίνεται σε ένα αίτημα με ένα μήνυμα απόκρισης. Η δομή του μηνύματος αυτού είναι η εξής:

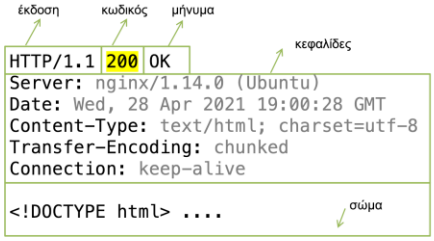
Η αρχική γραμμή της απόκρισης ονομάζεται **γραμμή κατάστασης**.

Η αρχική γραμμή αποτελείται από τα εξής τρία στοιχεία χωρισμένα με κενά:

- **Έκδοση πρωτοκόλλου HTTP** (π.χ. HTTP/1.1).
- **Κωδικός κατάστασης απάντησης** (Status code).
- **Μήνυμα περιγραφής** του κωδικού κατάστασης (reason phrase).

Ακολουθούν οι **γραμμές κεφαλίδων** οι οποίες είναι προαιρετικές. Στη συνέχεια ακολουθεί μια κενή γραμμή δύο χαρακτήρων CRLF = \r\n (χαρακτήρας ASCII 13, ακολουθούμενος από τον χαρακτήρα ASCII 10), και μετά το (προαιρετικό) σώμα του μηνύματος (π.χ. ένα αρχείο που μεταφέρεται στον φυλλομετρητή, δεδομένα αναζήτησης ή αποτελέσματα αναζήτησης).

Ακολουθεί το παράδειγμα του μηνύματος απόκρισης της προηγούμενης ενότητας.



Εικόνα 1.11 *Παράδειγμα μηνύματος απόκρισης HTTP που θα μπορούσε να στείλει ο εξυπηρετητής για να απαντήσει στο προηγούμενο μήνυμα.*

Οι δυνατές τιμές που μπορεί να πάρουν οι κωδικοί κατάστασης, που είναι ακέραιοι αριθμοί τριών χαρακτήρων, εκ των οποίων ο πρώτος καθορίζει την κατηγορία της απάντησης, είναι:

- **1xx**: μήνυμα πληροφορίας.
- **2xx**: κατάσταση επιτυχούς επεξεργασίας.
- **3xx**: προώθηση του μηνύματος σε διαφορετικό παραλήπτη.
- **4xx**: σφάλμα προερχόμενο από τον πελάτη.
- **5xx**: σφάλμα προερχόμενο από τον εξυπηρετητή.

Πιο ειδικά, μερικά από τα πιο συνήθη μηνύματα που μπορεί να επιστρέψει ένας εξυπηρετητής είναι:

- 100 Continue
- 200 OK
- 301 Moved Permanently
- 302 Moved Temporarily
- 304 Not Modified
- 404 Not Found
- 403 Forbidden
- 503 Service Unavailable
- 500 Internal Server Error

Κεφαλίδες μηνυμάτων απόκρισης

Ένα μήνυμα απόκρισης μπορεί να περιέχει κεφαλίδες γενικού σκοπού ή ειδική επεξήγηση της απόκρισης.

Αν εξετάσουμε ως παράδειγμα την απόκριση της προηγούμενης ενότητας, σχολιάζοντας τις κεφαλίδες:

- Server: nginx/1.14.0 (Ubuntu)
  Η κεφαλίδα αυτή παρέχει πληροφορία για το λογισμικό του εξυπηρετητή ιστού με τη μορφή Server : product | comment, αν η απόκριση διαβιβαστεί μέσω ενός proxy, αυτή η πληροφορία δεν τροποποιείται.
- Date: Wed, 28 Apr 2021 19:00:28 GMT
  Η κεφαλίδα αυτή περιέχει πληροφορίες για την ημέρα και ώρα κατά την οποία δημιουργήθηκε το μήνυμα.
- Content-Type: text/html; charset=utf-8
  Η κεφαλίδα αυτή παρέχει πληροφορία για τον τύπο δεδομένων του πόρου. Περιέχει τον τύπο MIME των δεδομένων, το πρότυπο κωδικοποίησης των χαρακτήρων charset, ενώ μπορεί να περιέχει ακόμη πληροφορίες για περιπτώσεις πολλαπλών ομάδων δεδομένων, τις συμβολοσειρές οριοθέτησής τους:

Content-Type: multipart/form-data; Boundary = συμβολοσειρά.

- Content-Encoding: gzip
  Η κεφαλίδα αυτή συνδυάζεται με την κεφαλίδα Content-type και την κεφαλίδα Transfer-encoding που ακολουθεί, για να ορίσει τον τρόπο συμπίεσης της πληροφορίας που στέλνεται. Αφορά ιδιαίτερα πολυμέσα, εικόνες, ήχο κ.λπ. Βοηθάει τον πελάτη να αποσυμπιέσει την πληροφορία που λαμβάνει σύμφωνα με τον τύπο MIME των δεδομένων. Τιμές που μπορεί να πάρει αυτή η κεφαλίδα είναι gzip, compress, deflate, br ή και συνδυασμός τους.
- Content-Language: el
  Η κεφαλίδα αυτή ορίζει το ακροατήριο για το οποίο προορίζεται η σελίδα. Οι κωδικοί των γλωσσών ορίζονται σύμφωνα με το πρότυπο ISO 639-1 που περιλαμβάνει συχνά και την υποκατηγορία σε μια γλώσσα, π.χ. “en-CA” περιγράφει αγγλικά για τον Καναδά. Περισσότερες από μια γλωσσικές ομάδες μπορεί να οριστούν χωρισμένες με κόμματα.
- Transfer-Encoding: chunked
  Η κεφαλίδα αυτή καθορίζει τον τρόπο μετάδοσης της πληροφορίας σε τεμάχια. Συχνά συνοδεύεται από την κεφαλίδα Content-Encoding: που αφορά τον τρόπο συμπίεσης της πληροφορίας που στέλνεται σε τεμάχια. Συνηθίζεται για την αποστολή μεγάλου όγκου πληροφορίας. Στην περίπτωση αυτή παραλείπεται η κεφαλίδα Content-Length: η οποία καθορίζει το μέγεθος της πληροφορίας που επιστρέφει. Η αποστολή των τεμαχίων τερματίζει με την αποστολή ενός άδειου τεμαχίου. Θα πρέπει να σημειωθεί ότι αυτή η επιλογή δεν έχει νόημα στο πρωτόκολλο HTTP/2.0 όπου ο τεμαχισμός και κωδικοποίηση της πληροφορίας ακολουθεί μια άλλη λογική, όπως θα εξηγηθεί στη συνέχεια.
- Connection: keep-alive
  Η κεφαλίδα αυτή καθορίζει ότι πρόκειται για «επίμονη σύνδεση» και συνεπώς είναι σε θέση να ικανοποιήσει και άλλα αιτήματα μετά την απόκριση αυτή (σε επόμενη ενότητα θα γίνει πιο λεπτομερής περιγραφή των τύπων σύνδεσης). Άλλη εναλλακτική τιμή που θα μπορούσε να πάρει η κεφαλίδα είναι close, που υποδεικνύει βραχύβια σύνδεση.
- Expires: Sun, 19 Nov 1978 05:00:00 GMT
  Η κεφαλίδα αυτή χρησιμοποιείται για να οριστεί η ημερομηνία μετά την οποία η πληροφορία θεωρείται ότι παύει να ισχύει. Η συγκεκριμένη τιμή είναι στο παρελθόν, συνεπώς η κεφαλίδα αγνοείται.
- Cache-Control: no-cache, must-revalidate
  Η κεφαλίδα αυτή καθορίζει την πολιτική ανανέωσης του περιεχομένου σε ενδιάμεσες μνήμες. Η παράμετρος no-cache καθορίζει ότι η πληροφορία μπορεί να αποθηκευτεί σε ενδιάμεσες μνήμες αλλά θα πρέπει να ελεγχθεί με τους μηχανισμούς που διαθέτουν οι ενδιάμεσες μνήμες για την εγκυρότητά της. Αυτό ορίζεται από τη δεύτερη παράμετρο, must-revalidate. Ανάλογα με το περιεχόμενο που αποστέλλει ο εξυπηρετητής, μπορεί να οριστούν διαφορετικές τιμές για τον έλεγχο ενδιάμεσων μνημών. Για παράδειγμα, no-store σημαίνει ότι η πληροφορία δεν πρέπει να αποθηκευτεί σε ενδιάμεση μνήμη, ενώ η δεύτερη παράμετρος, αν πάρει την τιμή immutable, δηλώνει ότι η πληροφορία δεν αλλάζει με τον χρόνο.
- Link: <http://www.upatras.gr/el>
  Η κεφαλίδα αυτή επιτρέπει στον εξυπηρετητή να συνδέσει τον πόρο που περιέχεται στην απόκριση σε κάποιο σύνδεσμο, ο οποίος περιέχει μεταδεδομένα για τον πόρο, συχνά συνοδεύει πολυμέσα, όπως βίντεο ή εικόνες, και περιλαμβάνει πληροφορίες, όπως Creative Commons, οδηγίες κ.λπ. Η τιμή της κεφαλίδας αυτής πρέπει να περιλαμβάνεται σε ετικέτες < > και μπορεί να συνοδεύεται από παραμέτρους όπως rel=meta/preconnect κ.λπ.

Τέλος, στην απόκριση υπάρχουν μια σειρά από κεφαλίδες που αρχίζουν με τον χαρακτήρα «X-», μεταξύ των οποίων η κεφαλίδα X-Generator: Drupal 7 (<http://drupal.org>), που υποδεικνύει το λογισμικό που έχει παραγάγει την απόκριση. Οι κεφαλίδες αυτές όμως δεν είναι στάνταρ και βαίνουν προς κατάργηση, όπως περιγράφεται στο σχετικό λήμμα της ιστοσελίδας MDN Web Docs: HTTP headers.

1.4.5 Διαχείριση σύνδεσης σε αιτήματα HTTP

Ένα πρόβλημα είναι πώς θα διαχειριστούμε τη σύνδεση TCP που θα πρέπει να ανοίξει για την προώθηση ενός αιτήματος HTTP.

Θα πρέπει να σημειωθεί ότι η αποκατάσταση σύνδεσης TCP είναι μια χρονοβόρα διαδικασία γιατί απαιτεί ανταλλαγή τριών μηνυμάτων (SYN, SYN/ACK, ACK) μεταξύ των δύο υπολογιστών. Μια αρχική προσέγγιση, που ήταν η τυπική συμπεριφορά στο πρωτόκολλο HTTP/ 1.0, ήταν να αποκαταστήσουμε μια

βραχύχρονη σύνδεση (short lived connection), όπου έπρεπε για κάθε αίτημα πόρου να ανοίγει νέα σύνδεση TCP η οποία έμενε ανοικτή μέχρι την παραλαβή απάντησης μεταξύ πελάτη και εξυπηρετητή και στη συνέχεια έκλεινε.

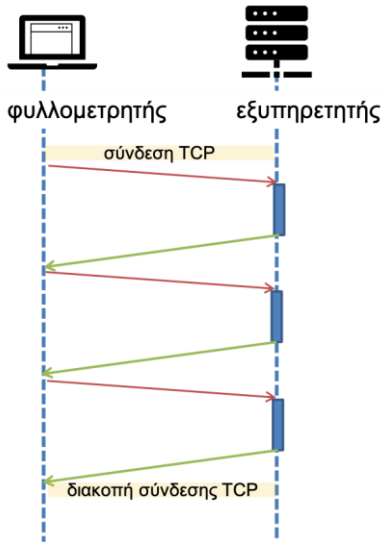
Από την έκδοση HTTP 1.1 η προκαθορισμένη συμπεριφορά είναι να εγκαθίσταται μια **επίμονη σύνδεση (persistant connection)**, η οποία παραμένει ενεργή για αρκετό χρόνο ώστε να καταστεί εφικτή η αποστολή όλων των αρχείων που συγκροτούν την ιστοσελίδα (εικόνες, αρχεία css, js κ.λπ.). Με την παράμετρο Keep-Alive της κεφαλίδας ενός αιτήματος HTTP μπορούμε να ορίσουμε τη διάρκεια της επίμονης σύνδεσης.

Για παράδειγμα, με τη δήλωση:

```
Keep-Alive: timeout=5, max=100
```

ορίζουμε ότι η σύνδεση θα μείνει ενεργή για 5 δευτερόλεπτα και στο πλαίσιό της να επιτρέψει μέχρι 100 αιτήματα για πόρους.

Στην **Εικόνα *1.12*** φαίνεται αυτός ο τύπος σύνδεσης.



Εικόνα 1.12 *Επίμονη σύνδεση.*

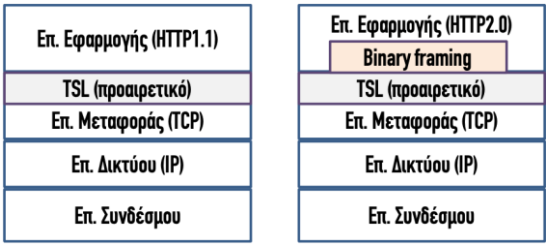
Αυτός ο τρόπος διαχείρισης της σύνδεσης είναι πιο αποδοτικός από τον προηγούμενο της βραχύχρονης σύνδεσης.

Πρέπει να σημειωθεί ότι υπάρχει μια ακόμη, εναλλακτική μέθοδος της **διεκπεραίωσης (pipelining)** σύμφωνα με την οποία ο πελάτης στέλνει ανεξάρτητες μεταξύ τους αιτήσεις, διαδοχικά μέσα από το επίμονο κανάλι επικοινωνίας, χωρίς να περιμένει πρώτα τον εξυπηρετητή να απαντήσει στην προηγούμενη αίτηση. Σε αυτή τη διαδικασία, εξαιρείται η αίτηση και απάντηση για τον πόρο του εγγράφου HTML, καθώς αυτός ορίζει ποιοι θα είναι οι επόμενοι πόροι που θα ζητήσει ο φυλλομετρητής. Η μέθοδος αυτή όμως μπορεί να προκαλέσει θέματα σε περίπτωση που κάποιο από τα παράλληλα αιτήματα αποτύχει, και για τον λόγο αυτό δεν είναι η προκαθορισμένη συμπεριφορά σε κανένα φυλλομετρητή.

Μια εναλλακτική τεχνική είναι η **πολυπλεξία των αιτημάτων** που έχει εισαγάγει η έκδοση HTTP/ 2.0, και θα περιγραφεί στη συνέχεια.

1.4.6 Πολυπλεξία μηνυμάτων με την HTTP 2.0

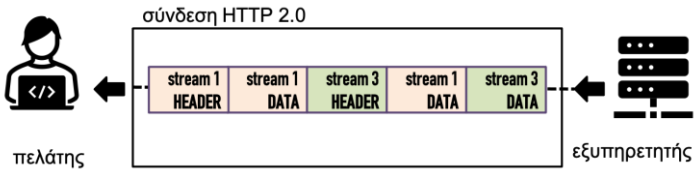
Το νέο πρωτόκολλο HTTP/2.0 δεν αλλάζει τη σημασιολογία του HTTP/1.1 καθώς και τις μεθόδους του πρωτοκόλλου. Όμως, αλλάζει τον τρόπο που κωδικοποιούνται και μεταδίδονται τα μηνύματα, ουσιαστικά βελτιώνοντας την απόδοση στη μετάδοση μεγάλου όγκου πληροφορίας που περιέχεται στις σύγχρονες ιστοσελίδες. Η διαστρωμάτωση των πρωτοκόλλων με το HTTP/1.1 και το HTTP/2.0 φαίνονται στην παρακάτω εικόνα.



Εικόνα 1.13 *Διαστρωμάτωση εφαρμογής ιστού, αριστερά HTTP 1.1, δεξιά HTTP 2.0.*

Η μόνη διαφορά είναι ότι παρεμβάλλεται μια διαδικασία κωδικοποίησης και τεμαχισμού του μηνύματος σε μια σειρά από δυαδικά πλαίσια (binary frames), πριν τα δεδομένα μεταφερθούν στο πιο χαμηλό επίπεδο μεταφοράς.

Κάθε μήνυμα κωδικοποιείται και χωρίζεται σε binary frames, ξεχωριστά για το τμήμα της κεφαλίδας και του σώματος δεδομένων. Τα πλαίσια αυτά δυαδικών δεδομένων στέλνονται ανεξάρτητα το ένα από το άλλο, πολλά δε μηνύματα μπορούν να περιπλεχτούν στο ίδιο κανάλι.



Εικόνα 1.14 *Ροή δυαδικών πλαισίων δεδομένων στο πρωτόκολλο HTTP 2.0.*

1.4.7 Διαπραγμάτευση περιεχομένου

Στο πλαίσιο ενός αιτήματος HTTP μπορεί να γίνει διαπραγμάτευση του περιεχομένου ενός αιτήματος. Ο πελάτης και ο εξυπηρετητής μπορούν να συμφωνήσουν π.χ. στην αποστολή συμπιεσμένης μορφής gzip ενός εγγράφου υπερκειμένου ή άλλου πόρου, εφόσον και οι δύο το υποστηρίζουν.

Στην περίπτωση αυτή, η συμπίεση γίνεται αδιαφανώς μέσα από το κανάλι επικοινωνίας, ενώ το περιεχόμενο του πόρου εμφανίζεται στον τελικό χρήστη ως ασυμπίεστο. Επίσης, άλλα σημεία για τα οποία μπορεί να γίνει διαπραγμάτευση είναι η κωδικοσελίδα χαρακτήρων εγγράφου, ο τύπος εικόνων του εγγράφου, η εξυπηρέτηση της αίτησης, μόνο εφόσον ο πόρος έχει τροποποιηθεί κ.λπ.

Οι ιδιότητες της κεφαλίδας του μηνύματος που χρησιμοποιούνται για τη διαπραγμάτευση αυτή είναι:

- Accept: maintype/subtype, όπου ορίζονται οι προτιμήσεις ως προς τον τύπο περιεχομένου με χρήση κωδικοποίησης MIME, π.χ. text/html, image/jpeg, application/octet-stream.
- Accept-Encoding, ορίζει τον τύπο κωδικοποίησης συμπίεσης, π.χ. Accept-Encoding: br, gzip;q=0.8.
- Accept-Language, ορίζει την προτιμώμενη γλώσσα, ορίζοντας μάλιστα την ποιότητα: Accept-Language: fr-CH, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5

1.4.8 Διαχείριση Cookies

Cookies ορίζονται τα δεδομένα, συνήθως μικρού μεγέθους, τα οποία ο εξυπηρετητής αποστέλλει στον φυλλομετρητή για προσωρινή αποθήκευση ώστε να χρησιμοποιηθούν στη συνέχεια. Τα cookies χρησιμεύουν για να εξασφαλίσουν τη διατήρηση της συνέχειας σε διαφορετικά αιτήματα από τον ίδιο φυλλομετρητή, για παράδειγμα σε μια εφαρμογή ηλεκτρονικού καταστήματος. Το πρόβλημα που προσπαθεί να αντιμετωπίσει η χρήση των cookies είναι ότι το πρωτόκολλο HTTP δεν «θυμάται» την προηγούμενη επικοινωνία (stateless protocol), άρα κάθε νέο αίτημα από έναν φυλλομετρητή δεν μπορεί να συνδεθεί με την ως τότε επικοινωνία.

Ο τρόπος να λυθεί το πρόβλημα είναι το cookie που αρχικά στάλθηκε από τον εξυπηρετητή να επισυναφθεί σε κάθε νέο αίτημα του φυλλομετρητή ως απόδειξη της ταυτότητάς του. Άλλες χρήσεις της τεχνικής αυτής είναι για προσωποποίηση της πληροφορίας, αφού στα cookies μπορεί να αποθηκευτούν οι προτιμήσεις του συγκεκριμένου χρήστη, ενώ μπορεί ακόμη να χρησιμοποιηθούν για καταγραφή της συμπεριφοράς του χρήστη, κάτι που ενδεχομένως να παραβιάζει την αρχή προστασίας των προσωπικών δεδομένων. Για τον λόγο αυτό, η Ευρωπαϊκή Ένωση στη νομοθεσία για προσωπικά δεδομένα (GDPR) απαιτεί

την ενημέρωση του χρήστη κάθε φορά που χρησιμοποιείται η τεχνολογία αυτή.

Δημιουργία cookie

Η δημιουργία cookie από τον εξυπηρετητή γίνεται με χρήση της κεφαλίδας Set-Cookie στην απόκριση του εξυπηρετητή. Στο παρακάτω παράδειγμα η κεφαλίδα του μηνύματος κοινοποιεί στον πελάτη ότι θα πρέπει να αποθηκευτούν δύο cookies, το καθένα με τιμή μια συμβολοσειρά:

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: cookie1="τραγανιστό μπισκότο"
Set-Cookie: cookie2="μουστοκούλουρο"
```

Στη συνέχεια, σε κάθε νέο αίτημα του πελάτη προς τον εξυπηρετητή αποστέλλονται τα δύο cookies.

```
GET /sample_page.html HTTP/2.0
Host: www.example.org
Cookie: cookie1="τραγανιστό μπισκότο"; cookie2="μουστοκούλουρο"
```

Η παράμετρος Expires ενός cookie ορίζει τη διάρκεια ζωής του. Για παράδειγμα:

```
Set-Cookie: cookie2="μουστοκούλουρο"; Expires=Thu, 31 Oct 2021 07:28:00
GMT;
```

Επίσης, η παράμετρος Secure επιβάλει την πρόσβαση στο cookie μέσω ασφαλούς σύνδεσης https και όχι απλής http. Τέλος, η παράμετρος HttpOnly ορίζει ότι στο cookie η πρόσβαση γίνεται μόνο μέσω της HTTP και όχι μέσω της JavaScript.

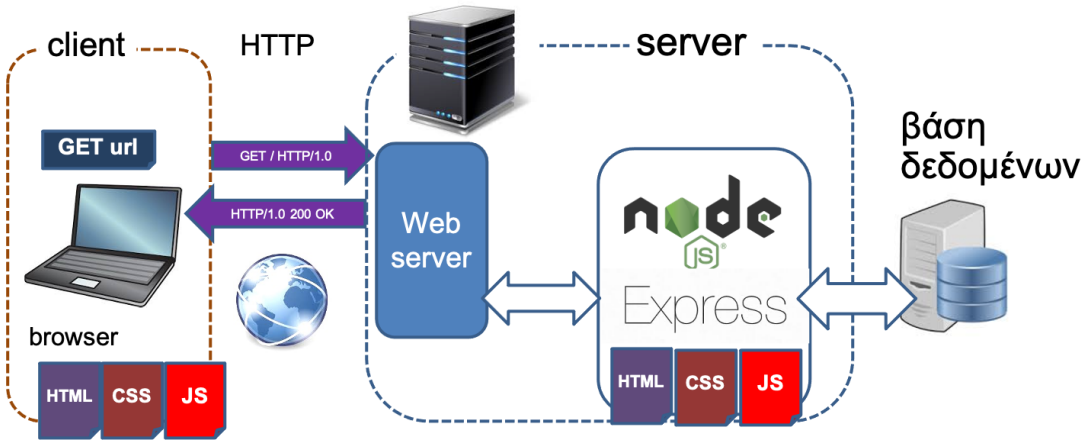
Τα cookies μπορούν να δημιουργηθούν μέσω της ιδιότητας document.cookie από την JavaScript ως εξής:

```
document.cookie = "cookie2=μουστοκούλουρο";
```

Τα cookies που δημιουργούνται με αυτό τον τρόπο δεν μπορούν να δεχτούν ως παράμετρο την HttpOnly.

1.5 Η αρχιτεκτονική των τεχνολογιών του βιβλίου

Με βάση τη γενική επισκόπηση της ιστορίας και της εξέλιξης του διαδικτύου που έγινε σε αυτό το πρώτο κεφάλαιο, στο οποίο δόθηκε ιδιαίτερη έμφαση στην εφαρμογή του παγκόσμιου ιστού και στο πρωτόκολλο που διέπει την επικοινωνία πελάτη/εξυπηρετητή στο πλαίσιό του, το HTTP, θα περιγράψουμε στη συνέχεια το αντικείμενο του βιβλίου αυτού.



Εικόνα 1.15 *Αρχιτεκτονική μιας τυπικής εφαρμογής που θα αναπτύξουμε στο πλαίσιο του βιβλίου.*

Για την περιγραφή των επόμενων κεφαλαίων θα χρησιμοποιήσουμε την παραπάνω εικόνα της αρχιτεκτονικής μιας διαδικτυακής εφαρμογής, μιας ιστοσελίδας που θα αναπτύξουμε.

Η εφαρμογή μας θα έχει δύο μέρη: ένα μέρος που θα κατεβαίνει στον φυλλομετρητή (στον πελάτη), όταν ο χρήστης καλέσει την ιστοσελίδα μας, και ένα μέρος που θα εκτελείται στον εξυπηρετητή, για να αποκριθεί στο αίτημα. Η ιστοσελίδα που θα σταλεί στον φυλλομετρητή θα αποτελείται από έναν συνδυασμό αρχείων HTML, CSS και JavaScript. Στον εξυπηρετητή θα τρέχει μια εφαρμογή JavaScript (στο περιβάλλον Node.js) που σε συνδυασμό με έναν εξυπηρετητή ιστού ικανοποιεί τα αιτήματα του χρήστη της ιστοσελίδας.

Αρχικά θα δούμε στα κεφάλαια 2 και 3 πώς να περιγράφουμε τη δομή και το περιεχόμενο μιας ιστοσελίδας με τη γλώσσα HTML5.

Στη συνέχεια, στα κεφάλαια 4 και 5 θα δούμε πώς θα ορίσουμε την εμφάνιση του περιεχομένου που καθορίσαμε με την HTML, χρησιμοποιώντας τη γλώσσα CSS3.

Στο κεφάλαιο 6 θα δούμε πώς η παρουσίαση του περιεχομένου θα γίνει πιο απλά με χρήση μιας βιβλιοθήκης παρουσίασης περιεχομένου, που είναι η BootStrap.

Στα κεφάλαια 7.4 ως 10 θα δούμε με αυξανόμενη λεπτομέρεια τη σύνταξη και χρήση της γλώσσας προγραμματισμού **JavaScript**, την οποία θα χρησιμοποιήσουμε στην αρχή σε συνδυασμό με τις τεχνολογίες που είδαμε στα πρώτα 6 κεφάλαια για να δώσουμε μεγαλύτερη διαδραστικότητα στην ιστοσελίδα μας όταν αυτή εκτελείται στον φυλλομετρητή.

Από το κεφάλαιο 11 και εξής θα μελετήσουμε τις τεχνολογίες που απαιτούνται για να εξυπηρετήσουμε τα αιτήματα από τον φυλλομετρητή. Θα εισαχθούμε στο περιβάλλον της **Node.js**, που είναι το περιβάλλον εκτέλεσης της JavaScript έξω από τον φυλλομετρητή. Στη συνέχεια θα δούμε το πλαίσιο **Express.js** (12) που μας επιτρέπει να δρομολογήσουμε αιτήματα σε έναν εξυπηρετητή και σε συνδυασμό με τη μηχανή template **handlebars** να ετοιμάσουμε το περιεχόμενο των ιστοσελίδων που θα στείλουμε στον πελάτη. Στο κεφάλαιο 13 θα δούμε πώς ο εξυπηρετητής μας μπορεί να επικοινωνήσει με μια βάση δεδομένων για μόνιμη αποθήκευση δεδομένων της εφαρμογής μας και ανάκτηση δεδομένων που ζητάει ο χρήστης. Τέλος, στο κεφάλαιο 14 θα δούμε πώς θα διαχειριστούμε τη συνεδρία του χρήστη και θα ελέγξουμε την ταυτότητά του.

Κλείνοντας αυτή την επισκόπηση, μπορούμε να παρατηρήσουμε ότι στο τέλος αυτής της διαδρομής θα έχουμε μια πλήρη εικόνα σχετικά με το πώς δομείται και λειτουργεί μια σύγχρονη διαδικτυακή εφαρμογή, με κατανόηση θεμάτων εμφάνισης περιεχομένου, διαχείρισης της διάδρασης με τον χρήστη, ασφάλειας και αποθήκευσης δεδομένων, συνδυάζοντας σύγχρονες τεχνολογίες και έχοντας ένα ισχυρό υπόβαθρο για περαιτέρω ανάπτυξη δεξιοτήτων και γνώσεων της λειτουργίας του διαδικτύου και των εφαρμογών του.

Άσκηση

Ακολουθεί ένα πρόγραμμα Python που μας επιτρέπει να εισαγάγουμε μια διεύθυνση URL και μας επιστρέφει την κεφαλίδα της απόκρισης HTTP που λαμβάνουμε. Χρησιμοποιεί τη βιβλιοθήκη **requests**.

```
# πρόγραμμα που ζητάει ένα url και επιστρέφει την κεφαλίδα της απόκρισης
HTTP
import requests

while True:
    url = input('url:')
    if url == '': break
    if not url.startswith('http'):
        url = 'http://'+ url
    try:
        with requests.get(url) as response:
            print("\nRESPONSE STATUS: ", response.status_code)
            print("RESPONSE HEADER")
            for key, value in response.headers.items():
                print("{:30s} {}".format(key, value))
    except:
        print('error opening', url)
```

Ένα παράδειγμα χρήσης:

```
url:http://hci.ece.upatras.gr/
RESPONSE STATUS:  200
RESPONSE HEADER
Date                           Sat, 27 Mar 2021 12:30:29 GMT
```

```
Server                          Apache/2.4.18 (Ubuntu)
Set-Cookie                      language=en; expires=Sun, 27-Mar-2022
12:30:29 GMT; Max-Age=31536000; path=/
Link                            <https://hci.ece.upatras.gr/wp-json/>;
rel="https://api.w.org/"
Vary                            Accept-Encoding
Content-Encoding                gzip
Content-Length                  9848
Keep-Alive                      timeout=5, max=100
Connection                      Keep-Alive
Content-Type                    text/html; charset=UTF-8
```

Πειραματιστείτε με το πρόγραμμα αυτό στέλνοντας μηνύματα σε διάφορες διευθύνσεις του διαδικτύου, μελετήστε προσεκτικά την κεφαλίδα της απόκρισης που πήρατε. Μπορείτε να βγάλετε συμπεράσματα για το τι τύπου είναι ο εξυπηρετητής που σας απάντησε (είναι η κεφαλίδα Server), τι τύπου είναι η απόκριση (Content-Type), αν επισυνάπτονται αρχεία cookies και τι διάρκεια ζωής έχουν (Set-Cookie) κ.λπ.

Επίσης, σκεφτείτε με ποιον τρόπο θα μπορούσατε να χρονομετρήσετε τον συνολικό χρόνο από την αποστολή του αιτήματος μέχρι την παραλαβή της απόκρισης (χρήση της time.process_time).

1.6 Ερωτήσεις αυτοαξιολόγησης

Σημείωση: Οι απαντήσεις στις ερωτήσεις αυτοαξιολόγησης που βρίσκονται στο τέλος κάθε κεφαλαίου δίνονται σε παράρτημα στο τέλος του βιβλίου.

1. Ο Tim Berners-Lee είναι ο εφευρέτης του διαδικτύου.
   – Σωστό/Λάθος
2. Το πρωτόκολλο ανταλλαγής μηνυμάτων του παγκόσμιου ιστού (web) είναι:
   1. TCP/IP
   2. HTML
   3. HTTP
   4. URL
3. Ο παγκόσμιος ιστός είχε εφευρεθεί ήδη από τη δεκαετία του 1960.
   – Σωστό/Λάθος
4. Ο web server είναι ένα πρόγραμμα που ξέρει να λαμβάνει και να στέλνει μηνύματα τύπου:
   1. HTTP
   2. HTML
   3. TCP/IP
5. Τα μηνύματα που επιστρέφει ένας web server στον πελάτη μπορεί να περιέχουν (σημειώστε όλα όσα επιτρέπονται):
   1. έγγραφο HTML
   2. εικόνες
   3. κώδικα JavaScript
   4. εντολές CSS
   5. απλό κείμενο
6. Το πρωτόκολλο HTTP :
   1. Ορίστηκε το 1989 από τον Berners-Lee.
   2. Υπήρχε ήδη από τη δεκαετία του 1960 μαζί με το TCP/IP.
   3. Είναι η πιο πρόσφατη εξέλιξη της γλώσσας HTML που δημιουργήθηκε από τον Berners-Lee το 1989.
7. Κατηγορίες αιτημάτων HTTP είναι (σημειώστε όλα όσα ταιριάζουν):
   1. SEND
   2. PUT
   3. GET
   4. POST
   5. MAIL
8. Ένα μήνυμα απόκρισης HTTP αρχίζει με την εξής πληροφορία:
   1. έναν κωδικό απόκρισης

2. την έκδοση του πρωτοκόλλου HTTP.
3. τη διεύθυνση του αποστολέα.
4. την ακολουθία χαρακτήρων CR LF.

9. Όταν ένα μήνυμα απόκρισης HTTP περιέχει τον κωδικό κατάστασης 301, τι συμπέρασμα βγάζουμε;
   1. Ότι ο πόρος που ζητήσαμε έχει έρθει σωστά.
   2. Ότι ο πόρος που ζητήσαμε δεν υπάρχει.
   3. Ότι ο πόρος που ζητήσαμε θα έρθει σε λίγο.
   4. Ότι ο πόρος που ζητήσαμε βρίσκεται σε άλλον εξυπηρετητή.
   5. Ότι ο εξυπηρετητής είναι απασχολημένος.
10. Όταν λάβουμε ένα μήνυμα HTTP με κωδικό 501 θα πρέπει:
    1. Να φροντίσουμε να διορθώσουμε τη διεύθυνση του πόρου που ζητάμε.
    2. Να ξαναπροσπαθήσουμε γιατί ο εξυπηρετητής έχει προσωρινά πρόβλημα.
    3. Κανένα πρόβλημα, η ιστοσελίδα θα έρθει με μικρή καθυστέρηση στον εξυπηρετητή.
11. Μια διεύθυνση URL πρέπει να περιλαμβάνει υποχρεωτικά το όνομα του host.
    – Σωστό/Λάθος
12. Αν ένα URL περιέχει στο τέλος το στοιχείο #http, αυτό αφορά:
    1. Το πρωτόκολλο (scheme) που θα χρησιμοποιηθεί (δηλαδή το http).
    2. Είναι ένδειξη αναζήτησης με λέξη-κλειδί http που έχει προκύψει από μια φόρμα της ιστοσελίδας.
    3. Είναι ένδειξη ότι ζητάμε να κατευθυνθούμε σε ένα τμήμα της ιστοσελίδας που έχει ταυτότητα http.
13. Αν ένα URL περιέχει το εξής τμήμα ?name=&last=, τι συμπέρασμα βγάζετε για την ιστοσελίδα από την οποία έχει παραχθεί το αίτημα;
    1. Είναι σφάλμα, θα πρέπει να υπάρχουν τιμές μετά τα σύμβολα =
    2. Είναι ένδειξη ότι υπάρχει φόρμα με πεδία με ονόματα name και last τα οποία όμως δεν έχουν συμπληρωθεί.
    3. Είναι ένδειξη ότι στην ιστοσελίδα υπάρχουν δύο περιοχές που ονομάζονται name και last και ζητάμε να κατευθυνθούμε σε αυτές με αυτή τη σειρά.
14. Αν ζητάμε να συνδεθούμε σε έναν υπολογιστή με όνομα www.host.com στη θύρα 5050, ποια θα ήταν η URL; (συμπληρώστε)
    http:// _____
15. Αν μια ιστοσελίδα βρίσκεται σε εξυπηρετητή που χρησιμοποιεί το πρωτόκολλο secure HTTP, ποιο θα είναι το αρχικό τμήμα του URL ενός πόρου που διαθέτει; (συμπληρώστε)
    _____

1.7 Βιβλιογραφία και Αναφορές

Υπάρχουν πολλές πηγές για τον αναγνώστη που θα ήθελε να εμβαθύνει στα θέματα αυτού του κεφαλαίου. Σχετικά με την ιστορία του διαδικτύου και του παγκόσμιου ιστού, ένα βιβλίο που περιγράφει το ιστορικό πλαίσιο και την εξέλιξη του διαδικτύου και του παγκόσμιου ιστού είναι του McCullough (2018). Επίσης, ενδιαφέρον είναι το βιβλίο του Berners-Lee (1999), που κάνει αναφορά στις αρχικές ιδέες και εκφράζει προβληματισμό για την εξέλιξη του παγκόσμιου ιστού. Μάλιστα, το βιβλίο αυτό έχει μεταφραστεί στα ελληνικά (2002). Επίσης, ενδιαφέρον ανάγνωσμα για την περίοδο της ανάπτυξης του διαδικτύου και των μηχανικών που συνέβαλλαν στον ορισμό των πρωτοκόλλων TCP/IP είναι αυτό των Hafner και Lyon (1998). Μια σύντομη αναφορά στην ιστορία του διαδικτύου περιέχει το άρθρο των Leiner κ.ά. (2009). Ένα πιο τεχνικό κείμενο αναφοράς είναι το άρθρο των Cerf και Kahn (1974) για τον ορισμό του πρωτοκόλλου TCP που περιέχει την ιδέα της μεταγωγής πακέτων.

Αξιοσημείωτη πηγή περιγραφής της αρχικής φάσης του διαδικτύου είναι η περιγραφή του Len Kleinrock (Severance, 2014), πρωτοπόρου στη σχεδίαση του διαδικτύου, που περιγράφει το πείραμα αποστολής των πρώτων πακέτων σύνδεσης μεταξύ του Πανεπιστημίου UCLA και του Stanford Research Institute (SRI), και το σχετικό βίντεο (IEEE Computer Society, 2014). Τέλος, στην περιορισμένη σχετικά ελληνική βιβλιογραφία ξεχωρίζει το βιβλίο των Δουληγέρη κ.ά. (2021) που καλύπτει στο αρχικό κεφάλαιο εκτενώς τα σχετικά πρωτόκολλα του διαδικτύου.

Ως προς τις ενότητες του κεφαλαίου που αφορούν τα πρωτόκολλα και το HTTP, χρήσιμη πηγή είναι η σελίδα τεκμηρίωσης του HTTP. Επίσης, η σελίδα του Mozilla περιέχει αναφορές στο πρωτόκολλο καθώς και tutorial. Σχετικό άρθρο των Garsiel και Irish (2011) περιγράφει πώς οι σύγχρονοι φυλλομετρητές λειτουργούν και χειρίζονται μηνύματα HTTP.

Επιπλέον, οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από το οποίο έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Για αυτό το κεφάλαιο το σχετικό μάθημα είναι οι εισαγωγικές διαλέξεις του μαθήματος «Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, Javascript».

Α. Ξενόγλωσσες

Cerf, V. G., & Kahn, R. E. (1974). A protocol for packet network interconnection. *IEEE Trans. Comm. Tech.*, *22*(5), pp. 627-641. https://doi.org/10.1109/TCOM.1974.1092259

Garsiel, T., & Irish, P. (2011). *How browsers work, Behind the scenes of modern web browsers*. Διαθέσιμο στο https://web.dev/howbrowserswork/

Hafner, K., & Lyon, M. (1998). *Where wizards stay up late: The origins of the Internet*. Touchstone, New York.

IEEE Computer Society (2014). *Len Kleinrock: The First Two Packets on the Internet*. YouTube. https://www.youtube.com/watch?v=uY7dUJT7OsU

Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., Postel, J., Roberts, L. G., & Wolff, S. (2009). A brief history of the internet. *SIGCOMM Comput. Commun. Rev.*, *39*(5), pp. 22-31. https://doi.org/10.1145/1629607.1629613

McCullough, B. (2018). *How the internet happened: from Netscape to the iPhone*. Liveright Publishing.

Severance, C. (2014). Len Kleinrock: The First Two Packets on the Internet. *Computer*, *47*(3), pp. 10-11. https://doi.org/10.1109/MC.2014.65

Β. Ελληνόγλωσσες

Αβούρης, Ν. (2018). *Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, JavaScript. Ανοικτό διαδικτυακό μάθημα*, https://mathesis.cup.gr

Berners-Lee, T., (2002). *Υφαίνοντας τον παγκόσμιο ιστό - Το παρελθόν, το παρόν και το μέλλον του παγκόσμιου ιστού από τον εφευρέτη*. Εκδόσεις Γκοβόστη.

Δουληγέρης, Χ., Μαυροπόδη, Ρ., Κοπανάκη, Ε., & Καραλής, Α. (2021). *Τεχνολογίες και Προγραμματισμός στον Παγκόσμιο Ιστό* (2η έκδ.). Εκδόσεις Νέων Τεχνολογιών. Κωδικός βιβλίου στον Εύδοξο: 102125023.

Κεφάλαιο 2. Εισαγωγή στη γλώσσα HTML

Σύνοψη

Στο κεφάλαιο αυτό γίνεται μια σύντομη εισαγωγή στη γλώσσα HTML, που είναι η θεμελιώδης τεχνολογία για δημιουργία ιστοσελίδων. Είναι το πρώτο από τα δύο κεφάλαια για την HTML. Θα κάνουμε μια σύντομη εισαγωγή στη γλώσσα και την ιστορία της, με αναφορές στο πρότυπο που την καθορίζει.

Η πρώτη επαφή με τη γλώσσα γίνεται μέσω μια απλής ιστοσελίδας που είναι και η αφετηρία για να εισαχθούν οι βασικές έννοιες: ***στοιχεία (elements)****,* ***ετικέτες*** *(tags),* ***γνωρίσματα*** *(attributes) και η δομή της σελίδας. Παρουσιάζονται βασικά στοιχεία της γλώσσας (<head>, <body>, <p>, <h1> ... <h6>, <hr>,
, , <li>, <ol>, <ul>, <a> κ.λπ.). Επίσης, γίνεται εισαγωγή στο Document Object Model (DOM), ενώ η τελευταία ενότητα παρουσιάζει το στοιχείο πίνακας (<table>).*

Στο επόμενο κεφάλαιο θα συνεχίσουμε με πιο προχωρημένα στοιχεία της HTML, όπως τα πολυμέσα και οι φόρμες.

Προαπαιτούμενη γνώση

Συνιστάται η μελέτη του κεφαλαίου 1.

2.1 Εισαγωγή

Όπως έχει ήδη αναφερθεί, τρεις ήταν οι βασικές τεχνολογίες που συνέθεσαν αρχικά την πρόταση δημιουργίας του παγκόσμιου ιστού. Αφενός το πρωτόκολλο HTTP επικοινωνίας πελάτη/εξυπηρετητή και ο μηχανισμός παγκόσμιου εντοπισμού πόρων (Universal Resource Locator, URL), που είδαμε στο προηγούμενο κεφάλαιο, αφετέρου η γλώσσα **HTML (Hyper Text Markup Language)**, η γλώσσα δημιουργίας και σύνταξης περιεχομένου ιστοσελίδων. Η HTML θα μας απασχολήσει εκτενώς στη συνέχεια.

2.1.1 Ιστορική αναφορά

Αρχικά η HTML ήταν μια γλώσσα που καθόριζε την εμφάνιση του περιεχομένου της ιστοσελίδας. Σήμερα η HTML5 (έκδοση 5) παίζει τον ρόλο της βάσης μιας ιστοσελίδας και καθορίζει το περιεχόμενό της και τη δομή της, ενώ συμπληρώνεται ουσιαστικά από δύο ακόμη τεχνολογίες, τις οποίες θα δούμε σε επόμενα κεφάλαια, τη CSS3, γλώσσα που ορίζει μέσω κανόνων την παρουσίαση του περιεχομένου, και τη γλώσσα JavaScript, μια γλώσσα προγραμματισμού γενικού σκοπού, που επιτρέπει σε μια ιστοσελίδα να παρουσιάζει δυναμική συμπεριφορά, δηλαδή να αλλάζει μετά από ενέργειες του χρήστη.

Η HTML είναι μια *γλώσσα επισημείωσης (markup)* που καθορίζει τη δομή και το περιεχόμενο μιας ιστοσελίδας. Ορίζει, για παράδειγμα, τα κείμενα της σελίδας, αλλά και τη δομή τους, τις παραγράφους στις οποίες αυτά χωρίζονται, τις κεφαλίδες τους, τις εικόνες, βίντεο που αυτή περιέχει κ.λπ. Η CSS έκδοση 3 είναι μια γλώσσα που ορίζει τον τρόπο παρουσίασης του περιεχομένου της ιστοσελίδας. Καθορίζει το στιλ παρουσίασης των διαφόρων στοιχείων, το χρώμα τους, το μέγεθος τους, τη γραμματοσειρά με την οποία παρουσιάζεται το κείμενο κ.λπ. Τέλος, η JavaScript ή JS είναι μια διερμηνευόμενη (interpreted) γλώσσα προγραμματισμού, η οποία εκτελείται μέσα στο περιβάλλον του φυλλομετρητή και η εκτέλεσή της επηρεάζει την εμφάνιση της ιστοσελίδας. Επιτρέπει να ανταποκρίνεται η ιστοσελίδα σε ενέργειες του χρήστη, δηλαδή να μην είναι απλά μια στατική σελίδα αλλά να προσφέρει μια αλληλεπιδραστική εμπειρία.

Η HTML ορίστηκε αρχικά ως μια απλοποιημένη έκδοση της SGML, η οποία είναι ένα πρότυπο (ISO 8879:1986) για επισημείωση κειμένων. Η γλώσσα πέρασε μια σειρά από εκδόσεις, με τις πιο σημαντικές την έκδοση HTML 3.2 που έγινε πρότυπο από το W3C το 1997 και την έκδοση HTML4 που ακολούθησε την ίδια εποχή. Η τρέχουσα έκδοση 5 είναι η πρώτη που πια δεν συνδέεται με τη SGML. Η ομάδα εργασίας Web Hypertext Application Technology Working Group (WHATWG) σε συνεργασία με το W3C World Wide Web Consortium καθορίζει σήμερα την προτυποποίηση της HTML, εξασφαλίζοντας ότι σε όλο το διαδίκτυο υλοποιείται η ίδια γλώσσα.

Αξίζει να επισκεφτεί κανείς την ιστοσελίδα https://html.spec.whatwg.org/ όπου συντηρείται η τρέχουσα έκδοση του προτύπου από την ομάδα εργασίας WHATWG, η οποία, σε συμφωνία με την W3C, έχει αναλάβει

να συντηρεί και να αναπτύσσει το πρότυπο της γλώσσας HTML. Μια ιστορική σημείωση που θα πρέπει να γίνει εδώ είναι ότι η ομάδα εργασίας WHATWG δημιουργήθηκε το 2004, όταν σε συνάντηση της επιτροπής του W3C δεν ψηφίστηκε η πρόταση από τη Mozilla και την Opera για ανάπτυξη της HTML5 προς την κατεύθυνση που τελικά πήρε, αλλά αντίθετα αποφασίστηκε να προχωρήσει το W3C στην ανάπτυξη της XHTML, δηλαδή της HTML σε κωδικοποίηση XML. Εκπρόσωποι μεγάλων εταιρειών και οργανισμών του διαδικτύου (Mozilla, Opera, Apple) προχώρησαν στην ίδρυση της WHATWG που έκτοτε αναπτύσσει το πρότυπο και, μάλιστα, από το 2019, με συμφωνία και της W3C, αποτελεί το μοναδικό σημείο αναφοράς του προτύπου. Στην ομάδα εργασίας σήμερα συμμετέχουν και άλλες εταιρείες διαδικτύου, όπως η Google και η Microsoft.

Θα πρέπει να σημειωθεί ότι η HTML5 υπάρχει ως W3C Recommendation από το 2014, ενώ αρχική πρόταση εκδόθηκε το 2008. Χρησιμοποιείται ήδη αρκετά χρόνια. Παρ' όλα αυτά, υπάρχουν ακόμη πολλές ιστοσελίδες και ιστότοποι που είναι υλοποιημένα σε προγενέστερες εκδόσεις της HTML, όπως η HTML4, η οποία υπάρχει ήδη από το 1997 και η ακόμα προγενέστερη HTML3.2. Επίσης, υπάρχουν φυλλομετρητές (κυρίως παλαιότερες εκδόσεις του Internet Explorer) που δεν υποστηρίζουν την HTML5. Αυτά θα πρέπει να τα λάβουμε υπόψη μας όταν αναπτύσσουμε εφαρμογές, όπως θα συζητηθεί στη συνέχεια.

Τα κύρια χαρακτηριστικά της HTML5 σε σύγκριση με προηγούμενες εκδόσεις της HTML είναι:

- Έχει απλοποιήσει συντακτικά τη γλώσσα και έχει βελτιώσει τα δομικά στοιχεία της, όπως τα στοιχεία <header>, <footer>, <nav>, <article>, <section> κ.λπ.
- Έχει βελτιώσει τις δυνατότητες που έχουν οι φόρμες για υποστήριξη του χρήστη στην εισαγωγή πληροφορίας – έχουμε, για παράδειγμα, τα καινούργια στοιχεία <input type='number'> ή τύπου date, time, calendar κ.λπ.
- Έχει εισαγάγει δυνατότητες για δημιουργία γραφικών στοιχείων, κάτι που χρησιμοποιείται ιδιαίτερα στα παιχνίδια και σε άλλες εφαρμογές, όπως είναι το στοιχείο <canvas>.
- Έχει βελτιώσει και έχει προτυποποιήσει την παρουσίαση πολυμεσικού περιεχομένου με το στοιχείο <audio> και το στοιχείο <video>, που είχαν ως συνέπεια, ουσιαστικά, να εκλείψουν άλλες τεχνολογίες, όπως ήταν η τεχνολογία Flash που χρησιμοποιούνταν παλαιότερα
- Έχει εισαγάγει API, διεπαφές της γλώσσας, που της δίνουν ιδιαίτερα ισχυρά χαρακτηριστικά, όπως, παραδείγματος χάρη, δυνατότητα γεωεντοπισμού, δυνατότητες για σύρε-άφησε (drag and drop) κ.λπ.

Αυτά είναι τα κύρια σημεία που θα τα δούμε στη συνέχεια.

2.2 Η πρώτη ιστοσελίδα

Ας δούμε τώρα πώς θα κατασκευάσουμε την πρώτη μας ιστοσελίδα. Έστω σε ένα περιβάλλον ανάπτυξης, όπως το Visual Studio Code, δημιουργούμε ένα καινούργιο αρχείο. Το οποίο ονομάζουμε ex00.html, όπου θα αποθηκεύσουμε ένα έγγραφο HTML.

```
<!DOCTYPE html>
<html lang="el">
    <head>
        <meta charset="utf-8">
        <title>τίτλος</title>
    </head>
    <body>
        <h1>Επικεφαλίδα</h1>
        <p> Πρώτη παράγραφος.</p>
    </body>
</html>
```

Το έγγραφο αυτό όταν το ανοίξουμε σε έναν φυλλομετρητή έχει την εξής εμφάνιση:

Εικόνα 2.1 *Εμφάνιση του εγγράφου στον φυλλομετρητή.*

Όπως παρατηρούμε, ένα έγγραφο HTML περιλαμβάνει και το περιεχόμενο που εμφανίζεται στην ιστοσελίδα (το κείμενο *«Επικεφαλίδα»* και το κείμενο *«Πρώτη παράγραφος»*), αλλά και ετικέτες που οργανώνονται ιεραρχικά και ορίζουν τα **στοιχεία** του εγγράφου.

Η πρώτη εντολή <!DOCTYPE html> είναι η δήλωση του τύπου εγγράφου που ακολουθεί. Δεν είναι υποχρεωτική, όμως είναι καλή πρακτική να μπαίνει ως πρώτη εντολή ενός εγγράφου HTML5. Σε προηγούμενες εκδόσεις της HTML ήταν η εντολή που συνέδεε το έγγραφο με τους κανόνες που έλεγχαν τη συντακτική του εγκυρότητα, για παράδειγμα ενός αρχείου DTD (Document Type Definition).

Η ρίζα της ιεραρχίας από στοιχεία του εγγράφου HTML5 είναι το στοιχείο <html>. Η αρχική ετικέτα <html> συνοδεύεται από την τερματική ετικέτα </html> ορίζοντας το στοιχείο αυτό. Το έγγραφο περιλαμβάνει μια ιεραρχία από στοιχεία. Η ρίζα έχει δύο παιδιά, το στοιχείο <head> και το στοιχείο <body>. Το <body> περιέχει το περιεχόμενο του εγγράφου που θα δει ο χρήστης, πριν από αυτό όμως ορίζουμε ένα στοιχείο <head>.

Μέσα στο στοιχείο <head> περιέχονται μεταδεδομένα του εγγράφου. Πολλά μεταδεδομένα ορίζονται από την ετικέτα <meta>. Παρατηρούμε ότι η ετικέτα αυτή δεν έχει αντίστοιχη ετικέτα τερματισμού, ορίζει, όπως θα αναφερθεί στη συνέχεια, ένα **κενό στοιχείο** (void element). Μάλιστα, η συγκεκριμένη ετικέτα περιλαμβάνει ένα γνώρισμα με την τιμή του charset="utf-8". Με αυτό τον τρόπο ορίζουμε ότι η κωδικοποίηση των χαρακτήρων του εγγράφου είναι utf-8, ώστε ο φυλλομετρητής να εμφανίσει σωστά χαρακτήρες εκτός του λατινικού αλφαβήτου, όπως είναι, παραδείγματος χάρη, τα ελληνικά. Ένα άλλο στοιχείο που περιέχεται στο <head> είναι το <title>, που μας επιτρέπει να ορίσουμε τον τίτλο που θα εμφανιστεί στο παράθυρο του φυλλομετρητή.

Ας δούμε στη συνέχεια το κυρίως σώμα της ιστοσελίδας, το οποίο περιλαμβάνει το περιεχόμενο μέσα σε νέα στοιχεία. Η συγκεκριμένη ιστοσελίδα περιλαμβάνει το στοιχείο <h1> που σημαίνει επικεφαλίδα (header) πρώτου επιπέδου, και στη συνέχεια το στοιχείο το οποίο αφορά μια παράγραφο (paragraph) κειμένου. Το κείμενο που περιλαμβάνεται ανάμεσα στις ετικέτες <h1> και </h1> καθώς και αυτό ανάμεσα στις ετικέτες και έχει διαφορετική εμφάνιση στον φυλλομετρητή. Η επικεφαλίδα επιπέδου 1 εμφανίζεται με πιο έντονα και μεγάλα γράμματα. Όμως, η ακριβής εμφάνιση του περιεχομένου των στοιχείων αυτών, όπως το χρώμα των χαρακτήρων και η γραμματοσειρά, καθορίζεται σε αυτή την περίπτωση από τον φυλλομετρητή, αφού δεν έχουμε ορίσει με κατάλληλες εντολές την εμφάνισή τους. Αν επιθυμούμε να ορίσουμε την εμφάνιση, θα πρέπει να εισάγουμε εντολές της CSS, όπως θα δούμε στη συνέχεια.

Επίσης, θα πρέπει να προσέξουμε ότι το περιεχόμενο του στοιχείου **<h1>** καθώς και του στοιχείου ** ** ξεκινάνε από την αρχή της γραμμής, αυτό γιατί και τα δύο αυτά στοιχεία είναι **στοιχεία μπλοκ (block elements)**, καθένα από τα οποία ορίζει έναν ορθογώνιο χώρο (block) της σελίδας. Αντίθετα, υπάρχουν άλλα στοιχεία, όπως το στοιχείο , τα οποία εισάγονται μέσα στην παράγραφο και επηρεάζουν μόνο τμήματα ενός στοιχείου block, τα οποία λέγονται **στοιχεία γραμμής (inline elements)**. Για παράδειγμα, αν τροποποιήσουμε το στοιχείο ως ακολούθως:

```
<p> <strong>Πρώτη</strong> παράγραφος.</p>
```

το αποτέλεσμα θα είναι το παρακάτω (παρατηρούμε ότι η λέξη *Πρώτη* εμφανίζεται πιο έντονα):

Εικόνα 2.2 *Τροποποίηση της σελίδας της εικόνας* **Εικόνα 2.1**.

Υπάρχουν και άλλα στοιχεία γραμμής (inline elements), στοιχεία δηλαδή που εντάσσονται στο πλαίσιο μιας γραμμής και συνήθως αφορούν ένα μικρό τμήμα του περιεχομένου. Τέτοια είναι ακόμη το στοιχείο (emphasis) και το <a> για anchor, δηλαδή υπερσύνδεσμος και ούτω καθεξής. Θα πρέπει να προσέξουμε ότι μέσα σε ένα στοιχείο γραμμής δεν μπορούμε να εισαγάγουμε ένα στοιχείο block. Ο κανόνας συνεπώς είναι: Ένα block δεν χωράει μέσα σε ένα στοιχείο inline, ενώ αντίθετα ένα στοιχείο inline χωράει σε ένα block.

Μπορούμε επίσης να παρατηρήσουμε στην αρχική ετικέτα <html> ότι έχουμε εισαγάγει το γνώρισμα lang με τιμή “el”. Το γνώρισμα αυτό δηλώνει τη γλώσσα του εγγράφου και είναι πολύ χρήσιμο σε μηχανές αναζήτησης. Το γνώρισμα lang μπορεί να εισαχθεί σε διαφορετικά στοιχεία που περιέχουν κείμενο. Για παράδειγμα:

```
<p lang="fr">Ceci est un paragraphe.</p>
```

Η παράγραφος αυτή δηλώνεται ότι είναι στη γαλλική γλώσσα.

Θα πρέπει να σημειωθεί ότι τυπικοί κώδικες γλωσσών είναι: “en” (english), “es” (spanish), “pt” (portuguese), “de” (german), “fr” (french), ενώ μια γενικότερη καταγραφή των κωδικών αυτών μπορεί κανείς να βρει στο language codes ISO 639-1.

2.2.1 Μορφοποίηση της πρώτης ιστοσελίδας

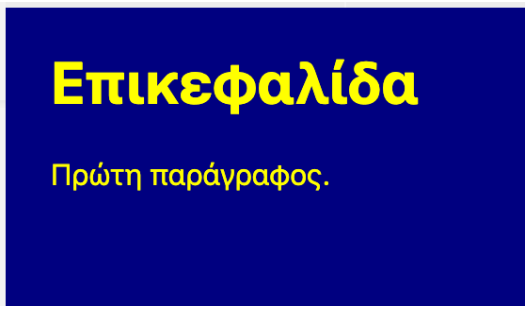
Έστω τώρα ότι θέλουμε να καθορίσουμε εμείς σε κάποιο βαθμό την εμφάνιση του εγγράφου. Έστω ότι επιθυμούμε να εμφανίζονται τα γράμματα με κίτρινο χρώμα και το υπόβαθρο της σελίδας μπλε.

Αρκεί να προσθέσουμε στην περιοχή <head> του εγγράφου το εξής στοιχείο:

```
<style>
    body { background: navy; color: yellow;}
</style>
```

Η ετικέτα <style> επιτρέπει την εισαγωγή εντολών CSS που σχετίζονται με τα στοιχεία του εγγράφου. Εδώ για το στοιχείο <body> (δηλαδή για όλο το περιεχόμενο του εγγράφου) ορίζουμε ότι το υπόβαθρό του να είναι χρώματος navy, ενώ το χρώμα του κειμένου να είναι yellow.

Το αποτέλεσμα αν ξανανοίξουμε το έγγραφο HTML στον φυλλομετρητή είναι το εξής:



Εικόνα 2.3 *Η σελίδα της εικόνας* **Εικόνα 2.1** *με εισαγωγή εντολής CSS.*

2.3 Σύνταξη της HTML

Θα δούμε στην ενότητα αυτή τα κύρια στοιχεία της σύνταξης ενός εγγράφου HTML. Όπως ήδη συζητήσαμε, σε ένα έγγραφο HTML περιέχονται κείμενα που θα εμφανιστούν στον χρήστη, καθώς και **στοιχεία HTML (HTML elements)**.

2.3.1 Στοιχεία

Τα στοιχεία ενός εγγράφου HTML έχουν εν γένει την εξής μορφή:

```
<tag> περιεχόμενο </tag>
```

Τα <tag> και </tag> είναι οι ετικέτες οριοθέτησης της αρχής και τέλους του στοιχείου. Να σημειωθεί ότι στα ονόματα των στοιχείων δεν υπάρχει διάκριση μεταξύ κεφαλαίων και μικρών, άρα η ετικέτα <head>μπορεί επίσης να γραφτεί ως <HEAD> ή <Head>.

Το περιεχόμενο ενός στοιχείου μπορεί να είναι κείμενο που θα εμφανιστεί στον χρήστη ή άλλα στοιχεία που εμπεριέχονται στο συγκεκριμένο στοιχείο, για παράδειγμα, είδαμε ότι στο στοιχείο περιέχεται το στοιχείο <h1>.

Η ιεραρχική παράθεση των στοιχείων θα πρέπει να είναι αυστηρή. Για παράδειγμα, ο παρακάτω κώδικας δεν είναι συντακτικά ορθός, αφού η ιεράρχηση του στοιχείου και του στοιχείου <p> δεν είναι αυστηρή, και θα δημιουργήσει σύγχυση στον συντακτικό αναλυτή της HTML:

```
<p>Το χωριό μου είναι <strong>πολύ όμορφο.</p></strong>
```

Θα πρέπει να σημειωθεί εν γένει ότι ο συντακτικός αναλυτής της HTML δεν είναι αυστηρός όσον αφορά συντακτικά σφάλματα, και χειρίζεται τα σφάλματα τέτοιου τύπου με τον καλύτερο δυνατό τρόπο, ώστε το έγγραφο να εμφανιστεί στον χρήστη.

2.3.2 Γνωρίσματα

Η ετικέτα αρχής ενός στοιχείου μπορεί να περιέχει και γνωρίσματα (attributes), όπως στο παράδειγμα:

```
<tag class="myclass"> περιεχόμενο </tag>
```

Τα γνωρίσματα χωρίζονται μεταξύ τους με κενά και ακολουθούνται από το = και την τιμή, συνήθως μέσα σε εισαγωγικά. Η τιμή μπορεί να περιέχεται σε εισαγωγικά ή όχι, αν όμως η τιμή περιλαμβάνει τον κενό χαρακτήρα, τότε είναι υποχρεωτικά τα εισαγωγικά, αφού ο κενός χαρακτήρας είναι διαχωριστικός χαρακτήρας μεταξύ των γνωρισμάτων. Εν γένει είναι καλή πρακτική να θέτουμε τις τιμές των γνωρισμάτων εντός εισαγωγικών, αν και αυτό, όπως αναφέρθηκε, δεν είναι υποχρεωτικό πάντα.

Υπάρχουν γνωρίσματα που είναι λογικές μεταβλητές, συνεπώς δεν παίρνουν τιμή. Για παράδειγμα:

```
<input type="text" disabled>
```

Το στοιχείο <input> έχει δύο γνωρίσματα, εκ των οποίων το δεύτερο είναι λογική μεταβλητή που παίρνει την τιμή true.

Τα γνωρίσματα αφορούν ιδιότητες του συγκεκριμένου στοιχείου, όπως, για παράδειγμα, η μοναδική ταυτότητα του στοιχείου, η κλάση στοιχείων στην οποία ανήκει κ.λπ.

2.3.3 Τυπικά γνωρίσματα στοιχείων

Στη συνέχεια παρατίθενται κάποια τυπικά γνωρίσματα στοιχείων της HTML.

- **id**. Ορίζει μοναδική ταυτότητα του στοιχείου.
- **class**. Ορίζει την κλάση στιλ που μοιράζεται το στοιχείο με άλλα στοιχεία.
- **hidden**. Ορίζει αν το στοιχείο εμφανίζεται στον χρήστη (λογικό γνώρισμα).
- **lang**. Ορίζει γλώσσα περιεχομένου του στοιχείου.
- **style**. Ορίζει ένα inline στιλ CSS για το στοιχείο.
- **contenteditable**. Αποφασίζει αν επιτρέπεται η τροποποίηση του περιεχομένου του στοιχείου.
- **data-...**. Αποθήκευση κρυφών δεδομένων, π.χ. <li data-animal-type="bird">Owl.

- **dir**. (direction) Κατεύθυνση κειμένου σε στοιχείο που περιέχει κείμενο.
- **draggable**. Ορίζει αν στο στοιχείο μπορεί να εφαρμοστεί ενέργεια σύρε-και-άφησε.
- **dropzone**. Ορίζει τι γίνεται αν τα δεδομένα που αφήνονται στο στοιχείο (σύρε-και-άφησε) αντιγράφονται, μετακινούνται ή συνδέονται.
- **spellcheck**. Ορίζει αν το στοιχείο μπορεί να ελεγχθεί για γλωσσική ορθότητα.
- **tabindex**. Ορίζει τη σειρά επίσκεψης των στοιχείων μέσω του πλήκτρου tab.
- **title**. Πρόσθετη πληροφορία, π.χ. <abbr title="World Health Organization">WHO</abbr>.
- **translate**. Ορίζει αν το περιεχόμενο πρέπει να μεταφραστεί.

Εκ των γνωρισμάτων αυτών, τα id και class είναι ιδιαίτερα σημαντικά και χρησιμοποιούνται το πρώτο ως μοναδική ταυτότητα του στοιχείου, για παράδειγμα για αναφορά σε αυτό σε κώδικα JavaScript, ενώ το δεύτερο δηλώνει την κλάση στην οποία ανήκει το στοιχείο, και συνεπώς ποιοι κοινοί κανόνες εμφάνισής του (εντολές CSS) το αφορούν.

2.3.4 Κενά στοιχεία

Υπάρχουν κάποια στοιχεία τα οποία εισάγονται με μόνο μια ετικέτα αρχής, χωρίς να περιλαμβάνουν περιεχόμενο, δηλαδή είναι αυτάρκη, και τα ονομάζουμε **κενά στοιχεία (void elements)** γιατί ακριβώς δεν έχουν περιεχόμενο. Τέτοια στοιχεία είναι το <hr> (horizontal rule), δηλαδή μια διαχωριστική οριζόντια γραμμή,
 (break), στοιχείο αλλαγής γραμμής, <img> για εισαγωγή εικόνας κ.λπ. Όπως είδαμε στο παράδειγμα της προηγούμενης ενότητας, το στοιχείο <meta> ήταν ένα κενό στοιχείο.

Σε κάποιες προηγούμενες εκδόσεις της HTML οι ετικέτες των κενών στοιχείων έπρεπε να αυτοτερματιστούν ως <tag/>, υποδεικνύοντας ότι η ετικέτα τερματίζει στον εαυτό της. Αυτό πια δεν απαιτείται στην HTML5.

2.3.5 Ειδικοί χαρακτήρες – διαχείριση κενών

Ένα άλλο θέμα που θα πρέπει να προσέξουμε είναι το πώς να χειριστούμε τους ειδικούς χαρακτήρες που χρησιμοποιούνται στη σύνταξη των ετικετών HTML όταν αυτοί πρέπει να εισαχθούν ως απλό κείμενο μέσα στο έγγραφο. Ας πάρουμε ένα παράδειγμα. Έστω ότι επιθυμούμε να εμφανίσουμε στον χρήστη το εξής κείμενο:

```
Στην HTML για παράγραφο χρησιμοποιούμε την ετικέτα <p>.
```

Αν θέλουμε αυτό το κείμενο να το εμφανίσουμε στον χρήστη, αυτό δεν είναι δυνατόν, γιατί η ακολουθία χαρακτήρων <p> συνιστά η ίδια μια ετικέτα της HTML. Συνεπώς, ο συντακτικός αναλυτής θα αντιληφθεί την ακολουθία αυτή ως εντολή HTML και όχι ως κείμενο.

Το πρόβλημα αυτό αντιμετωπίζεται με τη χρήση ειδικών ακολουθιών χαρακτήρων (character reference) που χρησιμοποιούνται για την εμφάνιση των ειδικών αυτών χαρακτήρων. Οι ακολουθίες αυτές περιλαμβάνουν το σύμβολο “ampersand” **&** και εν συνεχεία μια λεκτική περιγραφή του συμβόλου που ακολουθείται από το σύμβολο **;**.

Σημείωση: Το σύμβολο αυτό χρησιμοποιείται και στα ελληνικά ως συντομογραφία της λέξης «και», προέρχεται δε από παράφραση της λατινικής φράσης ‘& per se and’, δηλαδή «& μόνο του σημαίνει και».

Η παρακάτω λίστα περιλαμβάνει τους πιο βασικούς χαρακτήρες που εισάγονται με αυτό τον τρόπο.

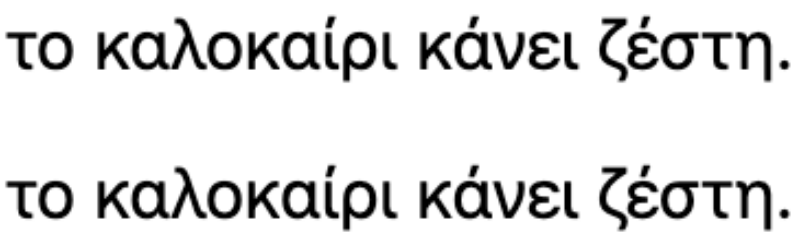
- **<** -> [< less than, μικρότερο]
- **>** -> [> greater than, μεγαλύτερο]
- **"** -> [" quote, διπλό εισαγωγικό]
- **'** -> [' apostrophe, μονό εισαγωγικό]
- **&** -> [**&** ampersand, χαρακτήρας &]
- ** ** -> [non breaking space, κενό]

Ιδιαίτερη μνεία πρέπει να γίνει για την τελευταία ακολουθία χαρακτήρων   που εκπροσωπεί τον κενό χαρακτήρα μη αλλαγής γραμμής. Η ακολουθία αυτή μπορεί να χρησιμοποιηθεί για την εισαγωγή κενών στο περιεχόμενο ενός στοιχείου της HTML. Η HTML εμφανίζει ακολουθίες μη εκτυπώσιμων χαρακτήρων, όπως το κενό, ο χαρακτήρας new line ή το tab, ως ένα μόνο κενό χαρακτήρα. Έτσι, για παράδειγμα, ένα κενό ακολουθούμενο από ένα άλλο κενό και έναν χαρακτήρα tab συμπτύσσονται σε έναν μόνο χαρακτήρα κενού. Με την ακολουθία   μπορούμε να εμφανίσουμε διαδοχικά περισσότερους από έναν κενούς χαρακτήρες. Επιπλέον, μπορούμε να χωρίσουμε λέξεις με την ακολουθία   έτσι ώστε να εμφανίζονται πάντα η μία δίπλα στην άλλη, καθώς το non breaking space δεν αναδιπλώνεται σε νέα γραμμή.

Για παράδειγμα, αν εισαγάγουμε σε ένα έγγραφο HTML τα εξής στοιχεία:

```
<p>το καλοκαίρι κάνει ζέστη.</p>

<p>το καλοκαίρι    κάνει
  ζέστη.</p>
```

το αποτέλεσμα στον φυλλομετρητή για τις δύο παραγράφους θα είναι ακριβώς το ίδιο:



Εικόνα 2.4 *Εμφάνιση των δύο παραγράφων στον φυλλομετρητή.*

Όπως βλέπουμε στην ***Εικόνα 2.4***, τα δύο στοιχεία εμφανίζονται με ακριβώς τον ίδιο τρόπο. Αυτό γιατί την ακολουθία των κενών ή μη εκτυπώσιμων χαρακτήρων του δεύτερου στοιχείου, ακόμη και τον χαρακτήρα αλλαγής γραμμής (new line), θα την αντικαταστήσει ο αναλυτής της HTML με ένα μόνο κενό.

Όμως, υπάρχουν περιπτώσεις που επιθυμούμε να υπάρχουν πολλά κενά. Τότε μπορούμε να χρησιμοποιήσουμε τον ειδικό χαρακτήρα ** ** ενώ με το στοιχείο **
** μπορούμε να επιβάλουμε αλλαγή γραμμής.

2.3.6 Σχόλια στον κώδικα HTML

Σε ένα έγγραφο HTML μπορούμε να εισαγάγουμε σχόλια με τον εξής τρόπο:

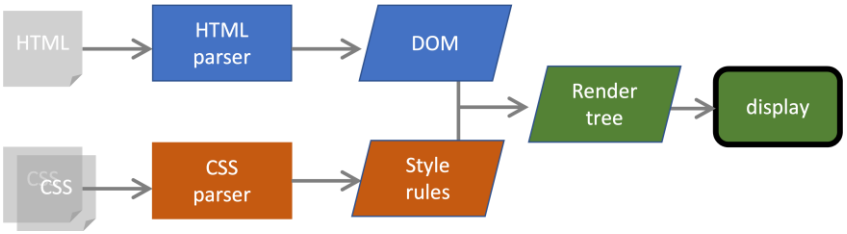
```
<!-- αυτό είναι ένα σχόλιο
που μπορεί να εκτείνεται
σε πολλές γραμμές ...
-->
```

Παρατηρούμε ότι και τα σχόλια περιλαμβάνονται μεταξύ των συμβόλων < και > όπως όλα τα στοιχεία HTML, και μετά ακολουθεί η ακολουθία θαυμαστικό - παύλα-παύλα, δηλώνοντας ότι αυτό που ακολουθεί είναι ένα σχόλιο το οποίο τερματίζεται με την ακολουθία παύλα-παύλα.

2.4 Μοντέλο αντικειμένων του εγγράφου HTML (DOM)

Στην ενότητα αυτή θα δοθεί μια σύντομη περιγραφή της λειτουργίας του συντακτικού αναλυτή της HTML στο περιβάλλον του φυλλομετρητή και της διαδικασίας δημιουργίας του **Μοντέλου Αντικειμένων του Εγγράφου (Document Object Model, DOM)**.

Στο παρακάτω διάγραμμα φαίνεται η διαδικασία συντακτικής ανάλυσης ενός εγγράφου HTML, που έχει ως ενδιάμεσο βήμα τη δημιουργία μιας αναπαράστασης της ιεραρχίας των στοιχείων του εγγράφου στη μνήμη του υπολογιστή, ως Μοντέλο Αντικειμένων του Εγγράφου (DOM). Στη συνέχεια, τα αντικείμενα αυτά, σε συνδυασμό με φύλλα στιλ που συνοδεύουν το έγγραφο, παράγουν την εμφάνιση του εγγράφου στο παράθυρο του φυλλομετρητή.

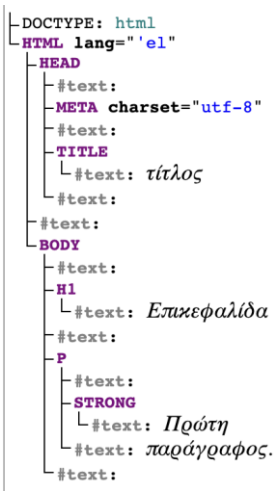


Εικόνα 2.5 *Αρχιτεκτονική του φυλλομετρητή.*

Το DOM δημιουργείται κατά τη συντακτική ανάλυση του εγγράφου HTML. Στην ουσία είναι μια ιεραρχική δομή δεδομένων που αφορά μια νέου τύπου αναπαράσταση ενός εγγράφου HTML. Κάθε κόμβος του DOM αντιστοιχεί σε ένα στοιχείο της HTML ή αφορά δεδομένα προς παρουσίαση. Το έγγραφο HTML φορτώνεται στον φυλλομετρητή, με κύριο στόχο να παρουσιαστεί στον χρήστη, αυτό επιτυγχάνεται με συντακτική ανάλυση του εγγράφου HTML. Ενώ γίνεται η συντακτική ανάλυση, παράγεται αυτή η ιεραρχική δομή αντικειμένων που έχει ως ρίζα το αντικείμενο document του φυλλομετρητή. Αυτή η ιεραρχική δομή δεδομένων ορίζεται ως **Document Object Model** και, αμέσως μόλις φορτωθεί και ολοκληρωθεί η συντακτική ανάλυση, γίνεται διαθέσιμο μέσω κάποιων διεπαφών στα προγραμματιστικά περιβάλλοντα, δηλαδή στην JavaScript. Μέσω της διεπαφής, η JavaScript βλέπει αυτά τα αντικείμενα που είναι τα στοιχεία της ιστοσελίδας και μπορεί να τα επεξεργαστεί, να τα τροποποιήσει, να τα μεταφέρει από μια θέση του δέντρου αυτού σε μια άλλη, να τους αλλάξει τις ιδιότητες, το στιλ, να τα εμφανίσει, να τα αποκρύψει, και ούτω καθεξής.

Άρα, η ύπαρξη του Document Object Model και των διεπαφών του είναι πολύ σημαντική γιατί επιτρέπει σε μια γλώσσα προγραμματισμού, όπως η JavaScript, να αλληλοεπιδρά και να επηρεάζει τα στοιχεία της ιστοσελίδας και, συνεπώς, να της προσδώσει δυναμική συμπεριφορά.

Υπάρχουν διάφορα εργαλεία που μας επιτρέπουν να δούμε το DOM ενός εγγράφου. Για παράδειγμα, στον live DOM viewer, αν φορτώσουμε τον κώδικα της προηγούμενης ενότητας, παράγεται η εξής αναπαράσταση του DOM:



Εικόνα 2.6 *Αναπαράσταση του DOM της αρχικής ιστοσελίδας.*

Παρατηρούμε ότι στο DOM περιέχονται με δομή δένδρου κόμβοι, οι οποίοι είναι είτε περιεχόμενο (εμφανίζονται ως #text) είτε στοιχεία HTML. Επίσης, μπορούμε να δούμε ένα παράδειγμα χρήσης της προγραμματιστικής διεπαφής του DOM.

Ανοίγουμε το παραπάνω αρχείο στον φυλλομετρητή και ενεργοποιούμε τα εργαλεία ανάπτυξης (στο Chrome, επιλογή inspection), από τα οποία επιλέγουμε την κονσόλα JavaScript (console)

Αν δώσουμε την εντολή

```
document.querySelector('body')
```

θα πάρουμε ως απόκριση το υποσύνολο της ιεραρχίας του DOM που αφορά το στοιχείο <body> και τα παιδιά του.

Αν στη συνέχεια θέλουμε να βρούμε τα παιδιά του στοιχείου body, μπορούμε να ζητήσουμε την τιμή της ιδιότητας children του αντίστοιχου κόμβου:

```
document.querySelector('body').children
```

Η απόκριση σε αυτή την περίπτωση είναι ένας πίνακας με δύο στοιχεία: [h1, p].

Αν, τέλος θέλουμε να βρούμε το πλήθος των παιδιών του body, μπορούμε να ζητήσουμε την ιδιότητα length του πίνακα αυτού

```
document.querySelector('body').children.length
```

και η απάντηση βεβαίως στην περίπτωση αυτή είναι 2.

Αυτά είναι παραδείγματα της προγραμματιστικής διεπαφής του DOM, η οποία περιλαμβάνει πολλές δυνατότητες. Θα επανέλθουμε στη διεπαφή του DOM στο κεφάλαιο που συζητάμε την JavaScript (Ενότητα 7.9).

2.5 Στοιχεία του τμήματος <head>

Όπως έχουμε ήδη συζητήσει, ένα έγγραφο HTML περιέχει δύο παιδιά της ρίζας του, το <head> και το <body>. Το στοιχείο <head> είναι το πρώτο από τα παιδιά της ρίζας html και περιέχει μεταδεδομένα για το έγγραφο. Είναι στοιχεία αντίστοιχα με τις κεφαλίδες ενός μηνύματος HTTP, που είδαμε στο πρώτο κεφάλαιο. Περιλαμβάνει πληροφορίες οι οποίες δεν θα εμφανιστούν στον χρήστη αλλά είναι χρήσιμες για τον φυλλομετρητή, για να ξέρει πώς να χειριστεί το συγκεκριμένο έγγραφο. Επίσης, πολλά μεταδεδομένα βοηθάνε τις μηχανές αναζήτησης να κατανοήσουν το αντικείμενο της σελίδας.

Τα στοιχεία που μπορεί να περιέχονται στο τμήμα αυτό του εγγράφου HTML είναι: <base>, <link>, <script>, <style>, <title>, <meta>.

Ας δούμε καθένα από αυτά ξεχωριστά.

2.5.1 Το στοιχείο <base>

Ορίζει την URL, με βάση την οποία δημιουργούνται τα σχετικά URL που συναντώνται στο έγγραφο. Δέχεται ως γνωρίσματα το href με τη διεύθυνση (υποχρεωτικό) και το target που καθορίζει, όταν επιλεγεί το URL, αν θα ανοίξει σε νέο target="_blank" ή στο ίδιο target="_self" παράθυρο του φυλλομετρητή. Παράδειγμα.

```
<base href="https://www.upatras.gr/" target="_blank">
```

2.5.2 Το στοιχείο <link>

Με το στοιχείο αυτό γίνεται σύνδεση του εγγράφου HTML με έναν εξωτερικό πόρο. Η πιο συνηθισμένη χρήση του είναι για σύνδεση με εξωτερικά αρχεία στιλ. Παράδειγμα:

```
<link href="main.css" rel="stylesheet">
```

Μια άλλη χρήση του είναι για σύνδεση με το εικονίδιο της ιστοσελίδας μεγέθους 16px, που εμφανίζεται στην πάνω αριστερή γωνία του αντίστοιχου παράθυρου ή καρτέλας, το **favicon (favorite icon)**.

```
<link rel="icon" href="favicon.ico" type="image/x-icon">
```

2.5.3 Το στοιχείο <script>

Το στοιχείο αυτό χρησιμοποιείται για ενσωμάτωση κώδικα JavaScript στο έγγραφο, καθώς επίσης και δεδομένων. Ο κώδικας μπορεί να περιέχεται εντός του στοιχείου <script> ή να βρίσκεται σε ένα εξωτερικό αρχείο και μέσω του γνωρίσματος src να γίνεται αναφορά σε αυτό.

Αν ο κώδικας JavaScript βρίσκεται σε εξωτερικό αρχείο, η ενσωμάτωσή του γίνεται ως εξής:

```
<script src="myscript.js defer"></script>
```

Θα πρέπει να σημειωθεί ότι το γνώρισμα defer (λογική μεταβλητή) δηλώνει ότι ο κώδικας θα φορτωθεί αφού ολοκληρωθεί το φόρτωμα του εγγράφου HTML. Θα συζητήσουμε αυτή την επιλογή στο κεφάλαιο της

JavaScript (Ενότητα 7.4).

Αν τώρα επιθυμούμε να ενσωματώσουμε τον κώδικα μέσα στο ίδιο το έγγραφο HTML, αυτό μπορεί να γίνει ως εξής:

```
<script>
    alert("Καλώς ήρθατε!")
</script>
```

Θα πρέπει να σημειωθεί εδώ ότι το στοιχείο αυτό μπορεί να ενσωματωθεί και στο <body> του εγγράφου, ενώ είναι σημαντικό να ορίσουμε πότε θα εκτελεστεί ο κώδικας, δηλαδή πριν ή μετά την ολοκλήρωση δημιουργίας του DOM, όπως θα συζητήσουμε σε επόμενο κεφάλαιο που αφορά την εκτέλεση του κώδικα JavaScript.

2.5.4 Το στοιχείο <style>

Το στοιχείο αυτό επιτρέπει την εισαγωγή εντολών CSS για το έγγραφο. Όπως είδαμε, εξωτερικά αρχεία CSS μπορούν να συνδεθούν με χρήση του στοιχείου link, όμως μερικές φορές κάποιες πιο ειδικές εντολές μορφοποίησης του περιεχομένου μπορεί να εισαχθούν μέσω του στοιχείου <style> στο τμήμα <head>.

Σε προηγούμενη ενότητα είδαμε πώς χρησιμοποιήθηκε το στοιχείο αυτό για να αλλάξουμε το χρώμα της σελίδας και του κειμένου στο εισαγωγικό μας παράδειγμα.

Εδώ θα προχωρήσουμε σε ένα πιο προχωρημένο παράδειγμα, όπου θα ορίσουμε ότι, αν το παράθυρο του φυλλομετρητή έχει πλάτος μικρότερο ή ίσο με 600px, αλλάζει το χρώμα του παράθυρου σε μπλε και τα γράμματα γίνονται λευκά.

Η εντολή είναι η εξής:

```
<style>
        @media (max-width: 600px) {body {
            background-color: navy; color: white}}
</style>
```

Αυτό είναι ένα παράδειγμα “media query” που ορίζει ότι, όταν ισχύει η πρώτη συνθήκη, τότε εκτελείται η εντολή. Θα γίνει εκτενής αναφορά στη σύνταξη των εντολών CSS στα κεφάλαια (4 και 5).

2.5.5 Το στοιχείο <title>

Το στοιχείο αυτό ορίζει το κείμενο στην άνω μπάρα του φυλλομετρητή ή της καρτέλας στην οποία παρουσιάζεται το έγγραφο HTML. Ένα παράδειγμα:

```
<title> Η πρώτη μου ιστοσελίδα </title>
```

Θα πρέπει να σημειωθεί ότι ο τίτλος της σελίδας βοηθάει τις μηχανές αναζήτησης, ιδιαίτερα αν είναι περιγραφικός του περιεχομένου της.

2.5.6 Το στοιχείο <meta>

Το στοιχείο αυτό παρέχει τη δυνατότητα να εισαγάγουμε μεταδεδομένα στην ιστοσελίδα. Το στοιχείο αυτό είναι ένα *κενό στοιχείο*, δηλαδή δεν χρειάζεται ετικέτα τερματισμού του, όπως ήδη είδαμε στο εισαγωγικό παράδειγμα. Στο παράδειγμα εκείνο χρησιμοποιήσαμε το στοιχείο αυτό για περιγραφή της κωδικοποίησης των χαρακτήρων της σελίδας (<**meta** charset="utf-8">). Αυτή είναι η πιο κλασική χρήση του στοιχείου αυτού.

Υπάρχουν όμως ακόμη δύο χρήσεις με ευρεία διάδοση. Το στοιχείο, όταν περιλαμβάνει τα γνωρίσματα name, content, τότε περιέχει μεταδεδομένα τα οποία επιτρέπουν σε μηχανές αναζήτησης να ευρετηριάσουν τη σελίδα πιο αποδοτικά. Ένα παράδειγμα:

```
<meta name="Description" content="author: N.Avouris -C.Sintoris, title:
Introduction to web programming">
<meta name="robots" conent="noindex, nofollow">
```

Στο παράδειγμα, το πρώτο στοιχείο <meta> παρέχει μια περιγραφή του περιεχομένου της σελίδας, ενώ το δεύτερο ορίζει στις μηχανές αναζήτησης να μην ευρετηριάσουν τη σελίδα. Στις οδηγίες προς τους

προγραμματιστές η Google κάνει ιδιαίτερη αναφορά στις δύο αυτές κατηγορίες μεταδεδομένων.

Ένα άλλο παράδειγμα μεταδεδομένων αφορά την εμφάνιση της σελίδας σε διαφορετικούς φυλλομετρητές, όπως για παράδειγμα στην περίπτωση κινητών με περιορισμένες διαστάσεις οθόνης και αυτόματη σμίκρυνση της σελίδας ώστε να χωράει το περιεχόμενο σε αυτή. Σε αυτή την περίπτωση θα πρέπει να χρησιμοποιηθεί το στοιχείο <meta> με γνώρισμα name ="viewport". Για παράδειγμα, αν επιθυμούμε η σελίδα να ακολουθεί τις διαστάσεις της συσκευής και να έχουμε ως αρχική κλίμακα 100% (δηλαδή να μη γίνεται σμίκρυνση της σελίδας), αυτό ορίζεται ως εξής:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Άλλες παράμετροι που μπορούν να προσδιοριστούν πέραν των width, initial-scale του παραδείγματος είναι: height, minimum-scale, maximum-scale, user-scalable.

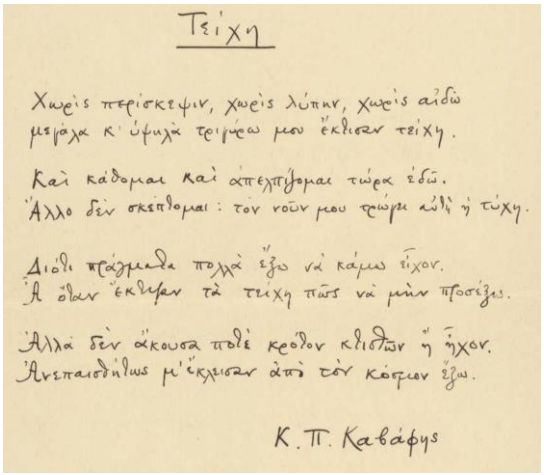
2.6 Τα στοιχεία του περιεχομένου: <body>

Είδαμε στην προηγούμενη ενότητα το περιεχόμενο του <head> ενός εγγράφου HTML. Το δεύτερο τμήμα του εγγράφου, το <body>, έχει ακόμη περισσότερο ενδιαφέρον, γιατί εδώ θα βρούμε το περιεχόμενο της σελίδας μας. Θα δούμε τι τρόπους μάς παρέχει η HTML για την οργάνωση του περιεχομένου.

2.6.1 Στοιχεία κειμένου με σημασιολογία <h1>, ... <h6>, <p>

Ένα οποιοδήποτε κείμενο οργανώνεται σε παραγράφους, οι οποίες έχουν επικεφαλίδες διαφορετικών επιπέδων, όπως γίνεται για παράδειγμα με τα κεφάλαια και τις ενότητες αυτού του βιβλίου. Τα στοιχεία που περιέχουν τις επικεφαλίδες διαφόρων επιπέδων φέρουν τα ονόματα <h1>, <h2>, …, <h6> (heading 1 … 6). Η επικεφαλίδα του υψηλότερου επιπέδου είναι η <h1>, που την είδαμε και στο εισαγωγικό μας παράδειγμα. Ενώ το απλό κείμενο εκτός των κεφαλίδων περιέχεται σε στοιχεία <p> (paragraph).

Έστω το παρακάτω χειρόγραφου του ποιητή μας Κώστα Καβάφη, που περιέχεται στο ψηφιακό αρχείο του Καβάφη του Ιδρύματος Ωνάση.


Τείχη

Χωρὶς περίσκεψιν, χωρὶς λύπην, χωρὶς αἰδὼ
μεγάλα κ' ὑψηλὰ τριγύρω μου ἔκτισαν τείχη.

Καὶ κάθομαι καὶ ἀπελπίζομαι τώρα ἐδῶ.
Ἄλλο δὲν σκέπτομαι: τὸν νοῦν μου τρώει αὐλὴ ἡ τύχη.

Διόλου πράγματα πολλὰ ἔξω νὰ κάμω εἶχον.
Ἂ ὅλων ἔκλεψαν τὰ τείχη πῶς νὰ μὴν προσέξω.

Ἀλλὰ δὲν ἄκουσα πολὺ κρεῖττον κλίδων ἢ ἦχον.
Ἀνεπαισθήτως μ' ἔκλεισαν ἀπὸ τὸν κόσμον ἔξω.

Κ. Π. Καβάφης

Εικόνα 2.7 *Χειρόγραφο του ποιητή Κ.Καβάφη.* https://cavafy.onassis.org/el/object/39eh-xggr-mk3e/

Ο ποιητής έχει δώσει τον τίτλο στο ποίημα, τον οποίο έχει υπογραμμίσει για να του δώσει βαρύτητα, έχει αφήσει κενό και έχει στη συνέχεια δώσει ξεχωριστά τις τέσσερις στροφές του ποιήματος και, τέλος, υπογράφει ιδιόχειρα.

Αν το κείμενο αυτού του ποιήματος δεν το είχαμε εισαγάγει σε στοιχεία HTML, θα ήταν ως εξής:

```
<body>
    Τείχη
    Χωρίς περίσκεψιν, χωρίς λύπην, χωρίς αιδώ
    μεγάλα κ’ υψηλά τριγύρω μου έκτισαν τείχη.
    Και κάθομαι και απελπίζομαι τώρα εδώ.
```

```
    Άλλο δεν σκέπτομαι: τον νουν μου τρώγει αυτή η τύχη·
    διότι πράγματα πολλά έξω να κάμω είχον.
    Α όταν έκτιζαν τα τείχη πώς να μην προσέξω.
    Αλλά δεν άκουσα ποτέ κρότον κτιστών ή ήχον.
    Ανεπαισθήτως μ’ έκλεισαν από τον κόσμον έξω.
</body>
```

Το αποτέλεσμα στον φυλλομετρητή είναι:

Τείχη Χωρίς περίσκεψιν, χωρίς λύπην, χωρίς αιδώ μεγάλα κ′ υψηλά τριγύρω μου έκτισαν τείχη. Και κάθομαι και απελπίζομαι τώρα εδώ. Άλλο δεν σκέπτομαι: τον νουν μου τρώγει αυτή η τύχη· διότι πράγματα πολλά έξω να κάμω είχον. Α όταν έκτιζαν τα τείχη πώς να μην προσέξω. Αλλά δεν άκουσα ποτέ κρότον κτιστών ή ήχον. Ανεπαισθήτως μ′ έκλεισαν από τον κόσμον έξω.

Εικόνα 2.8 *Παρουσίαση του ποιήματος χωρίς οργάνωσή του σε στοιχεία HTML.*

Ας οργανώσουμε τώρα το κείμενο με χρήση στοιχείων HTML ως ακολούθως:

```
<body>
    <h1>Τείχη</h1>
    <p>Χωρίς περίσκεψιν, χωρίς λύπην, χωρίς αιδώ<br>
        μεγάλα κ’ υψηλά τριγύρω μου έκτισαν τείχη.</p>

    <p>Και κάθομαι και απελπίζομαι τώρα εδώ.<br>
    Άλλο δεν σκέπτομαι: τον νουν μου τρώγει αυτή η τύχη·</p>

    <p>διότι πράγματα πολλά έξω να κάμω είχον.<br>
    Α όταν έκτιζαν τα τείχη πώς να μην προσέξω.</p>

    <p>Αλλά δεν άκουσα ποτέ κρότον κτιστών ή ήχον.<br>
    Ανεπαισθήτως μ’ έκλεισαν από τον κόσμον έξω. </p>
</body>
```

Το αποτέλεσμα στον φυλλομετρητή μοιάζει περισσότερο με το αρχικό ποίημα.

Τείχη

Χωρίς περίσκεψιν, χωρίς λύπην, χωρίς αιδώ
μεγάλα κ′ υψηλά τριγύρω μου έκτισαν τείχη.

Και κάθομαι και απελπίζομαι τώρα εδώ.
Άλλο δεν σκέπτομαι: τον νουν μου τρώγει αυτή η τύχη·

διότι πράγματα πολλά έξω να κάμω είχον.
Α όταν έκτιζαν τα τείχη πώς να μην προσέξω.

Αλλά δεν άκουσα ποτέ κρότον κτιστών ή ήχον.
Ανεπαισθήτως μ′ έκλεισαν από τον κόσμον έξω.

Εικόνα 2.9 *Παρουσίαση του ποιήματος μετά την οργάνωσή του σε στοιχεία HTML.*

Παραμένει θέμα υπό εξέταση, αφού μάλιστα δούμε σε επόμενα κεφάλαια τη χρήση CSS, πώς θα

μπορούσαμε να προχωρήσουμε σε τροποποίηση της εμφάνισης των στοιχείων <h1> και <p> ώστε το ποίημα να μοιάζει σε μεγαλύτερο βαθμό με το αρχικό χειρόγραφο του ποιητή.

Η χρήση των στοιχείων αυτών που φέρουν σημασιολογικό περιεχόμενο θα πρέπει να γίνεται με προσοχή. Τα στοιχεία <h1> έως <h6>, όπως αναφέρθηκε, αναπαριστούν επικεφαλίδες με φθίνουσα σπουδαιότητα. Η <h1> είναι η υψηλότερη επικεφαλίδα. Για μια ιστοσελίδα θα πρέπει να υπάρχει μόνο μια <h1> ή, αν η ιστοσελίδα περιέχει πολλαπλά άρθρα, το καθένα να έχει μόνο ένα <h1>, όπως ο τίτλος ενός βιβλίου είναι ένας. Η <h2> είναι το δεύτερο επίπεδο, που θα μπορούσε να αντιστοιχεί στα κεφάλαια του βιβλίου. Η <h3> σε ενότητες του κάθε κεφαλαίου και ούτω καθεξής μέχρι το επίπεδο <h6>. Καθεμία από αυτές θα πρέπει να αντιστοιχεί σε διαφορετικού μεγέθους γραμματοσειρά.

Θα πρέπει να προσέξουμε ώστε να χρησιμοποιούνται τα στοιχεία αυτά όχι για την εμφάνιση του κειμένου, αλλά για να το οργανώσουμε ως μια ιεραρχία στοιχείων με νόημα. Αυτό έχει ιδιαίτερη σημασία για τα άτομα με ειδικές ανάγκες, αφού επηρεάζει την προσβασιμότητα της σελίδας.

Κάποιοι σχεδιαστές χρησιμοποιούν την <h1> ή την <h2> απλά για μορφοποίηση, για να τονίσουν ένα τμήμα του κειμένου. Αυτός είναι εσφαλμένος τρόπος χρήσης των στοιχείων αυτών. Τα στοιχεία αυτά έχουν σημασιολογία. Κάποιοι χρήστες με δυσκολίες όρασης μπορεί να πλοηγούνται γρήγορα σε κάποιο επίπεδο για να τις εξετάσουν, όπως κάνουμε όταν ξεφυλλίζουμε ένα βιβλίο, ώστε να ελέγξουν αν υπάρχει ένα κεφάλαιο που τους ενδιαφέρει. Αυτό γίνεται για παράδειγμα με τους screen readers, δηλαδή τους αναγνώστες της οθόνης, που είναι συσκευές που χρησιμοποιούνται για άτομα με ειδικές ανάγκες, με δυσκολίες όρασης. Αν χρησιμοποιήσουμε τα στοιχεία <h1> έως <h6> καταχρηστικά, απλώς για παρουσίαση μεγέθους γραμματοσειρών, ουσιαστικά καταργούμε αυτή την αρχή της ιεράρχησης του κειμένου και ο αναγνώστης της οθόνης δεν μπορεί να εξυπηρετήσει το άτομο με ειδικές ανάγκες.

2.6.2 Στοιχεία κειμένου χωρίς σημασιολογία ,

Δύο στοιχεία με ευρεία χρήση είναι τα και . Το ορίζει ένα μπλοκ κειμένου, αντίστοιχα με το <p>, ενώ το ορίζει μια ακολουθία χαρακτήρων μέσα σε ένα μπλοκ, είναι, όπως λέγεται, *στοιχείο γραμμής (inline element)*. Τα δύο αυτά στοιχεία δεν φέρουν σημασιολογία αλλά ορίζουν στοιχεία στα οποία μπορούμε να επιβάλουμε ορισμένο στιλ εμφάνισης. Η χρήση τους είναι ιδιαίτερα εκτεταμένη από τότε που χρησιμοποιήθηκε εκτενώς η CSS για καθορισμό της μορφής του περιεχομένου της σελίδας.

Θα πρέπει να σημειωθεί ότι παλαιότερα η HTML είχε πολλές ετικέτες, οι οποίες τείνουν να καταργηθούν και αφορούσαν τη μορφοποίηση. Όριζαν, για παράδειγμα, στοιχεία <b> για έντονο κείμενο (bold), <i> για πλάγια γραφή (italics) και <u> για υπογραμμισμένο κείμενο (underline). Αυτές τείνουν να καταργηθούν γιατί αντικαθίστανται από τα στιλ CSS όπου ορίζουμε τρόπο εμφάνισης (κεφάλαια 4 και 5). Τα στιλ επιβάλλονται σε στοιχεία και , τα οποία ορίζουν περιοχές στις οποίες εφαρμόζουμε ένα στιλ εμφάνισης με κατάλληλες εντολές CSS. Και αυτό γίνεται με το να δώσουμε ταυτότητα σε κάθε τέτοιο στοιχείο με το γνώρισμα id ή να ορίσουμε μια κλάση που επιβάλλεται σε ένα σύνολο από στοιχεία με το γνώρισμα class.

Η παράγραφος <p> μοιάζει αρκετά με το . Υπάρχουν κάποιες διαφορές. Η παράγραφος ορίζει μια παράγραφο κειμένου, ένα τμήμα κειμένου που ξεχωρίζει από το υπόλοιπο κείμενο. Ο φυλλομετρητής προσθέτει ένα περιθώριο από το κείμενο που υπάρχει πριν ή μετά την παράγραφο. Μία ακόμη διαφορά είναι ότι **σε ένα στοιχείο <p> δεν μπορούμε να εισαγάγουμε άλλα στοιχεία μπλοκ**, ενώ αυτό μπορεί να γίνει σε ένα . Άρα ένα μπορεί να περιέχει παραγράφους ή άλλα στοιχεία , ενώ μια παράγραφος δεν μπορεί να περιέχει ή άλλες παραγράφους.

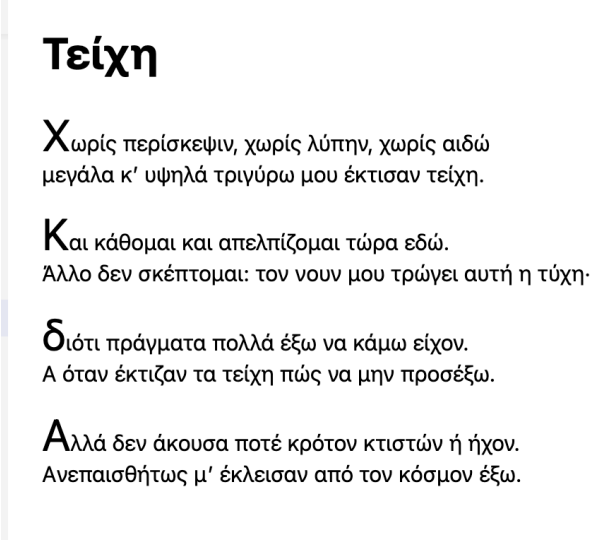
Ας δούμε ένα παράδειγμα. Έστω ότι επιθυμούμε να διαφοροποιήσουμε τον πρώτον χαρακτήρα κάθε στροφής του ποιήματος. Αυτό γίνεται ως εξής: Σε κάθε πρώτη σειρά εισάγουμε τον πρώτον χαρακτήρα σε ένα στοιχείο . Ως παράδειγμα, η πρώτη στροφή του ποιήματος γίνεται:

```
<p><span class="first-letter">Χ</span>ωρίς περίσκεψιν, χωρίς λύπην, χωρίς
αιδώ<br>
μεγάλα κ’ υψηλά τριγύρω μου έκτισαν τείχη.</p>
```

Αυτό επαναλαμβάνεται για όλες τις στροφές. Επίσης, στο στοιχείο <head> εισάγουμε έναν κανόνα μορφοποίησης για την κλάση "fist-letter", με βάση τον οποίο το περιεχόμενό του είναι διπλάσιο σε μέγεθος από το υπόλοιπο κείμενο (να σημειωθεί ότι το όνομα μιας κλάσης προσδιορίζεται με τον χαρακτήρα "." στις εντολές CSS, όπως θα δούμε στο κεφάλαιο 4).

```
<style>
    .first-letter {font-size: 2em;}
</style>
```

Το αποτέλεσμα είναι το εξής:



Εικόνα 2.10 *Παρουσίαση του ποιήματος με εφαρμογή κανόνα CSS για διπλάσιο μέγεθος πρώτου γράμματος κάθε στροφής.*

2.6.3 Έμφαση σε κείμενο ,

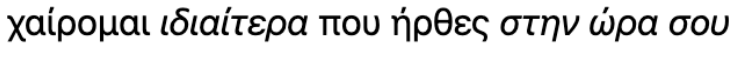
Δύο στοιχεία που μπορούμε να χρησιμοποιήσουμε για να δώσουμε ιδιαίτερη έμφαση σε τμήματα ενός κειμένου είναι τα στοιχεία και .

Και τα δύο αυτά στοιχεία χρησιμοποιούνται για να δώσουμε μια μεγαλύτερη έμφαση σε ένα κείμενο και είναι στοιχεία γραμμής (inline elements). Δηλαδή, δεν δημιουργούν ένα νέο μπλοκ κειμένου, αλλά αλλάζουν την έμφαση σε κάποια τμήματα ενός κειμένου.

Ας πούμε ότι στην παρακάτω φράση «χαίρομαι ιδιαίτερα που ήρθες στην ώρα σου» επιθυμούμε να τονίσουμε τη λέξη **ιδιαίτερα** και τις λέξεις **στην ώρα σου** και για αυτό τα βάζουμε μέσα σε ένα στοιχείο (έμφαση).

```
<p>χαίρομαι <em>ιδιαίτερα</em> που ήρθες <em>στην ώρα σου</em></p>
```

Όταν παρουσιαστεί αυτό το κείμενο στον φυλλομετρητή, θα δούμε ότι η λέξη «ιδιαίτερα» και η φράση «στην ώρα σου» εμφανίζονται με πλάγιους χαρακτήρες (italics).



Εικόνα 2.11 *Παράδειγμα στοιχείου .*

Το στοιχείο χρησιμοποιείται για να δοθεί ακόμα πιο μεγάλη έμφαση. Αν θέλουμε να συγκρίνουμε το με το , το δίνει με τον ίδιο τρόπο που το κάνει το ακόμα μεγαλύτερη έμφαση στο τμήμα που αναφέρεται, μάλιστα αυτό λαμβάνεται υπόψη και από τους αναγνώστες για άτομα με ειδικές ανάγκες. Ένα παράδειγμα είναι, έστω, ότι θέλουμε να πούμε «αυτό το υγρό είναι πολύ τοξικό». Επειδή επιθυμούμε να δώσουμε πολύ μεγάλη έμφαση, ώστε να προσέχουν όσοι το χειρίζονται, το διαμορφώνουμε ως εξής:

```
<p>αυτό το υγρό είναι <strong> πολύ τοξικό </strong></p>
```

Αυτή η φράση θα εμφανιστεί στον φυλλομετρητή συνήθως με έντονη γραφή (bold).

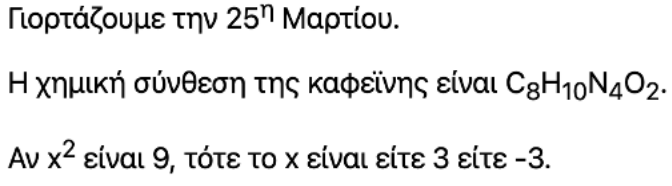
Εικόνα 2.12 *Παράδειγμα στοιχείου .*

Η έντονη γραφή (bold), λοιπόν, έχει πιο μεγάλη βαρύτητα από ό,τι η πλάγια γραφή (italics). Ένα ερώτημα που τίθεται είναι αν συνεχίζουν να έχουν χρήση τα στοιχεία <b> (bold) και <i> (italics) που είναι παλαιότερα και οι αντίστοιχες ετικέτες πιο σύντομες. Η απάντηση είναι ότι τα στοιχεία και , επειδή έχουν σημασιολογικό περιεχόμενο, εκφράζοντας έμφαση διαφορετικής βαρύτητας, υποστηρίζονται από τους αναγνώστες της οθόνης (screen readers), οι οποίοι τονίζουν με πιο μεγάλη έμφαση τη φωνή του ομιλητή και έτσι και το άτομο με ειδικές ανάγκες μπορεί να αντιληφθεί καλύτερα το νόημα. Άρα, συστήνονται πολύ περισσότερο από τα <b> και <i> που είναι μεν συντομογραφίες ως ετικέτες του ενός χαρακτήρα, όμως δεν φέρουν σημασιολογικό περιεχόμενο, αλλά απλά αλλάζουν την εμφάνιση.

2.6.4 Άνω και κάτω δείκτης κειμένου <sup>, <sub>

Μία ανάγκη που έχουμε είναι να παρουσιάσουμε κάποιους χαρακτήρες σαν δείκτες άνω ή κάτω από την κανονική σειρά του κειμένου.

Για παράδειγμα, έστω ότι επιθυμούμε να παρουσιάσουμε τις εξής φράσεις:



Εικόνα 2.13 *Κείμενα που περιλαμβάνουν δείκτες και κάτω δείκτες.*

Για τη λέξη *25η Μαρτίου* ο χαρακτήρας «η» είναι δείκτης με μικρότερη γραμματοσειρά. Το ίδιο για τη χημική σύνθεση όπου χρησιμοποιούμε κάτω δείκτες. Πώς μπορούμε να κάνουμε αυτό στην HTML;

Ας δούμε τον κώδικα που παράγει αυτή την ημερομηνία.

```
<p>Γιορτάζουμε την 25<sup>η</sup> Μαρτίου.</p>
```

Παρατηρούμε ότι στο κείμενο ορίζουμε ως στοιχείο <sup> τον χαρακτήρα δείκτη (superscript). Έτσι, ο χαρακτήρας αυτός εμφανίζεται ως άνω δείκτης για να μπορέσουμε να εκφράσουμε την ημερομηνία με αυτό τον τρόπο.

Ας δούμε τώρα το δεύτερο παράδειγμα της χημικής ένωσης.

```
<p>Η χημική σύνθεση της καφεΐνης είναι :
C<sub>8</sub>H<sub>10</sub>N<sub>4</sub>O<sub>2</sub>.</p>
```

Εδώ παρατηρώ ότι τα στοιχεία που επιθυμώ να θέσω ως κάτω δείκτες τα εντάσσω σε στοιχεία <sub> με αντίστοιχο τρόπο.

Το ίδιο και για την περίπτωση της μαθηματικής έκφρασης:

```
<p>Αν x<sup>2</sup> είναι 9, τότε το x είναι είτε 3 είτε -3.</p>
```

2.6.5 Οριζόντια γραμμή και αλλαγή γραμμής <hr>,

Επίσης, σημαντικά στοιχεία είναι αυτά που μας επιτρέπουν είτε να ορίσουμε μια οριζόντια γραμμή που διαχωρίζει περιοχές του κειμένου είτε να αλλάξουμε γραμμή. Και τα δύο αυτά στοιχεία είναι κενά στοιχεία.

Η ετικέτα με την οποία δημιουργούμε ένα στοιχείο οριζόντιας γραμμής είναι η <hr> (horizontal rule) που διαχωρίζει περιοχές. Μάλιστα, αξίζει να αναφερθεί ότι μπορούμε να ορίσουμε με κάποιες παραμέτρους το πλάτος αυτής της γραμμής, το χρώμα της κ.λπ.

Η ετικέτα αλλαγής γραμμής είναι η
 (break) που είναι, όπως έχει ήδη αναφερθεί, μια κενή ετικέτα. Το στοιχείο αυτό μπορεί να μας επιτρέψει να αλλάξουμε γραμμή μέσα σε ένα στοιχείο μπλοκ. Επίσης, να σημειωθεί ότι ο χαρακτήρας ASCII αλλαγής γραμμής δεν λαμβάνεται υπόψη από τον συντακτικό αναλυτή της HTML, αφού αντικαθίσταται από έναν κενό χαρακτήρα.

Θυμίζουμε ότι ένα στοιχείο μπλοκ, όπως είναι ένα ή <p>, ξεκινάει σε μια καινούργια γραμμή.

Όπως είδαμε όμως και στο παράδειγμα του ποιήματος, αν επιθυμούμε να αλλάξουμε γραμμή σε ένα μπλοκ στοιχείο, τότε πρέπει να εισαγάγουμε ένα στοιχείο
.

2.6.6 Στοιχεία μορφοποίησης κειμένου <pre>, <blockquote>

Το στοιχείο <pre> που σημαίνει “pre formatted” (προ-μορφοποιημένο κείμενο) ορίζει μια περιοχή κειμένου στην οποία μπορούμε να διατηρήσουμε τους κενούς χαρακτήρες, χαρακτήρες αλλαγής γραμμής κ.λπ., καθώς και να εμφανίζεται το κείμενο με χαρακτήρες με σταθερό διάστημα ανά χαρακτήρα (monospace).

Ας δούμε ένα παράδειγμα. Αυτό το τμήμα κειμένου είναι κώδικας Python που πρέπει να διατηρήσουμε τη στοίχιση:

```
<h1>Python code</h1>
<pre>
def add_mult(*x):
    a, m = 0, 1
    for i in x:
        a += i
        m *= i
    return (a, m)
</pre>
```

Το αποτέλεσμα είναι:

Python code

```
def add_mult(*x):
    a, m = 0, 1
    for i in x:
        a += i
        m *= i
    return (a, m)
```

Εικόνα 2.14 *Εμφάνιση κώδικα με χρήση <pre>.*

Για να εμφανίσουμε κώδικα μέσα στη γραμμή, μπορούμε να χρησιμοποιήσουμε και το εξειδικευμένο στοιχείο γραμμής **<code>**.

Το επόμενο ενδιαφέρον στοιχείο είναι το <blockquote>, όπου στην ουσία είναι μια παράγραφος με αριστερή στοίχιση, η οποία όμως είναι στοιχισμένη πιο δεξιά σε σχέση με το υπόλοιπο κείμενο. Αυτό το στιλ εμφάνισης κειμένου προέρχεται από επιστημονικές δημοσιεύσεις, όπου γίνεται κατά λέξη αναφορά ενός άλλου κειμένου. Το κείμενο αυτό το βάζουμε σε εισαγωγικά και το παρουσιάζουμε με έναν διαφορετικό τρόπο, ώστε να φαίνεται ότι δεν είναι δικό μας, αλλά ότι είναι αναφορά την οποία έχουμε περιλάβει στο κείμενο. Αυτή λοιπόν η παράθεση (quotting), όπως λέμε την αναφορά σε κάποιον άλλον, χρησιμεύει εδώ για παρουσίαση κειμένου με αυτό τον τρόπο.

Βλέπουμε εδώ ένα παράδειγμα. Ένα απόσπασμα από τον λόγο του Κολοκοτρώνη προς μαθητές του γυμνασίου το 1838. Επειδή τα λόγια είναι όπως καταγράφηκαν, εισάγονται ως στοιχείο <blockquote>.

```
    <body>
        <h1>Ομιλία του Κολοκοτρώνη στην Πνύκα</h1>
        <p>Ο γέρος του Μωριά επισκέφθηκε το Βασιλικό Γυμνάσιο της Αθήνας,
            ανέβηκε με τους μαθητές στην Πνύκα και τους είπε τα εξής
λόγια:
        </p>
        <blockquote>
            Ὅταν ἀποφασίσαμε νὰ κάμωμε τὴν Ἐπανάσταση, δὲν ἐσυλλογισθήκαμε
            οὔτε πόσοι εἴμεθα οὔτε πὼς δὲν ἔχομε ἄρματα οὔτε ὅτι οἱ
Τοῦρκοι
            ἐβαστοῦσαν τὰ κάστρα καὶ τὰς πόλεις οὔτε κανένας φρόνιμος μᾶς
            εἶπε «ποῦ πᾶτε ἐδῶ νὰ πολεμήσετε μὲ σιταροκάραβα βατσέλα»,
            ἀλλὰ ὡς μια βροχὴ ἔπεσε εἰς ὅλους μας ἡ ἐπιθυμία τῆς
ἐλευθερίας μας
        </blockquote>
    </body>
```

Το έγγραφο αυτό στον φυλλομετρητή εμφανίζεται ως εξής:

Ομιλία του Κολοκτρώνη στην Πνύκα

Ο γέρος του Μωριά επισκέφθηκε το Βασιλικό Γυμνάσιο της Αθήνας, ανέβηκε με τους μαθητές στην Πνύκα και τους είπε τα εξής λόγια:

> Ὅταν ἀποφασίσαμε νὰ κάμωμε τὴν Ἐπανάσταση, δὲν ἐσυλλογισθήκαμε οὔτε πόσοι εἴμεθα οὔτε πὼς δὲν ἔχομε ἄρματα οὔτε ὅτι οἱ Τοῦρκοι ἐβαστοῦσαν τὰ κάστρα καὶ τὰς πόλεις οὔτε κανένας φρόνιμος μᾶς εἶπε «ποῦ πᾶτε ἐδῶ νὰ πολεμήσετε μὲ σιταροκάραβα βατσέλα», ἀλλὰ ὡς μία βροχὴ ἔπεσε εἰς ὅλους μας ἡ ἐπιθυμία τῆς ἐλευθερίας μας

Εικόνα 2.15 *Παράδειγμα χρήσης στοιχείου <blockquote>.*

2.6.7 Λίστες <ol>,

Ένα άλλο στοιχείο που έχει ενδιαφέρον είναι αυτό που χρησιμεύει για να παρουσιάσουμε λίστες από αντικείμενα, από οδηγίες, από εικόνες κ.λπ.

Για παράδειγμα, πριν πάμε στο μανάβικο έχουμε σημειώσει ότι θέλουμε να αγοράσουμε τα εξής φρούτα:

- Μήλα
- Πορτοκάλια
- Μανταρίνια
- Μπανάνες

Σε αυτή τη λίστα έχουμε συμπεριλάβει τέσσερα φρούτα, το ένα κάτω από το άλλο, χωρίς να υπάρχει συγκεκριμένη σειρά. Έχουμε λοιπόν μια μη ταξινομημένη λίστα. Σε αυτού του τύπου τις λίστες χρησιμοποιούμε κάποιο σύμβολο, όπως είναι εδώ η βούλα, που ξεχωρίζει τα στοιχεία της λίστας.

Πώς ορίζουμε λοιπόν μια λίστα στην HTML; Αν δούμε κώδικα που έχει παραγάγει αυτή εδώ την εικόνα, θα διαπιστώσουμε ότι η λίστα ορίζεται ως ένα στοιχείο <ul> (unordered list, μη ταξινομημένη λίστα), δηλαδή μια λίστα χωρίς σειρά. Το στοιχείο <ul> περιέχει τα επιμέρους στοιχεία της λίστας, τα φρούτα, τα οποία ορίζονται ως στοιχεία της λίστας (list item) και εδώ φαίνονται τα στοιχεία αυτά.

```
<ul>
    <li>Μήλα</li>
    <li>Πορτοκάλια</li>
    <li>Μανταρίνια</li>
    <li>Μπανάνες</li>
</ul>
```

Θα πρέπει να σημειώσουμε ότι ως επιμέρους στοιχείο μιας λίστας μπορεί να είναι και μια άλλη λίστα. Ας πούμε ότι κάτω από τα πορτοκάλια θέτουμε ένα καινούργιο στοιχείο, το οποίο είναι μια καινούργια λίστα η οποία

περιέχει τα δικά της στοιχεία (συγκεκριμένα είδη πορτοκαλιών).

- Μήλα
- Πορτοκάλια
  - Σαγκουίνια
  - Μέρλιν
  - Βαλένθια
- Μανταρίνια
- Μπανάνες

Αυτό το νέο στοιχείο λοιπόν είναι ένα αντικείμενο που υπάρχει στην αρχική λίστα και περιέχει με τη σειρά του δικά του στοιχεία. Αν θέλουμε να δούμε την καινούργια εικόνα, βλέπουμε τώρα ότι έχουμε τα αρχικά στοιχεία και ένα από τα στοιχεία είναι μια υπολίστα κάτω από το αντικείμενο αυτό.

Ο κώδικας που παράγει αυτή την εικόνα είναι ο εξής:

```
<ul>
 <li>Μήλα</li>
 <li>Πορτοκάλια
        <ul>
            <li>Σαγκουίνια</li>
            <li>Μέρλιν</li>
            <li>Βαλένθια</li>
        </ul>
    </li>
    <li>Μανταρίνια</li>
 <li>Μπανάνες</li>
</ul>
```

Μια δεύτερη περίπτωση λίστας είναι εκείνη που περιέχει αντικείμενα που έχουν μια ορισμένη σειρά. Έστω ότι θέλουμε να δώσουμε οδηγίες μαγειρικής. Σε αυτή την περίπτωση δημιουργούμε ένα στοιχείο <ol>, που σημαίνει “ordered list” (ταξινομημένη λίστα).

1. Βάζουμε το αυγό στο νερό
2. Βράζουμε για 5΄
3. Απομακρύνουμε από τη φωτιά
4. Καθαρίζουμε το αυγό
5. Σερβίρουμε με αλάτι, πιπέρι

Ο κώδικας που παράγει αυτή την αναπαράσταση είναι ο εξής:

```
<ol>
    <li>Βάζουμε το αυγό στο νερό</li>
    <li>Βράζουμε για 5'</li>
    <li>Απομακρύνουμε από τη φωτιά</li>
    <li>Καθαρίζουμε το αυγό</li>
    <li>Σερβίρουμε με αλάτι, πιπέρι</li>
</ol>
```

Αν για κάποιο λόγο επιθυμούμε η αρίθμηση των στοιχείων της λίστας να μην αρχίσει από το 1 αλλά από άλλον αριθμό, το ορίζουμε με το γνώρισμα start του στοιχείου <ol>.

```
<ol start="5">
```

Επίσης, μπορούμε να ορίσουμε αν η φορά αρίθμησης θα είναι αύξουσα ή φθίνουσα. Η τελευταία επιλογή ορίζεται με το γνώρισμα reversed.

Για παράδειγμα, η εντολή:

```
<ol start="5" reversed>
```

θα αριθμήσει τα στοιχεία της ταξινομημένης λίστας από το 5 με φθίνουσα σειρά, 4, 3, 2 και τα λοιπά.

2.6.8 Υπερσύνδεσμοι <a>

Σε αυτή την ενότητα θα δούμε πώς σε ένα έγγραφο HTML μπορούμε να εισαγάγουμε υπερσυνδέσμους.

Οι υπερσύνδεσμοι (hyperlinks) είναι μια θεμελιώδης έννοια στο διαδίκτυο και στην HTML ιδιαίτερα. Να λάβουμε υπόψη μας ότι η λέξη *hyper* αποτελεί συστατικό του hypertext hyperlink που είναι συστατικό του ίδιου του ονόματος της γλώσσας HTML και σημαίνει Hypertext Markup Language, επίσης, αποτελεί συστατικό του πρωτοκόλλου που χρησιμοποιείται για τη μετάδοση του περιεχομένου HTML που είναι το HTTP Hypertext Transfer Protocol και ούτω καθεξής. Ακόμη, να σημειώσουμε ότι το hyper προέρχεται από την ελληνική λέξη *ὑπέρ* που, ως δασυνόμενη, αρχίζει στα αγγλικά από το γράμμα h. Ας δούμε τώρα πώς εισάγουμε έναν υπερσύνδεσμο μέσα σε μια ιστοσελίδα.

Ο υπερσύνδεσμος εισάγεται ως στοιχείο <a> (“anchor”, άγκυρα). Είναι μια «αγκύρωση» ενός στοιχείου του εγγράφου, συνήθως, σε ένα άλλο έγγραφο HTML, σε μια άλλη ιστοσελίδα.

Με αυτό τον τρόπο, με την εισαγωγή ενός στοιχείου <a>, το οποίο περιέχει ως γνώρισμα href τη διεύθυνση του πόρου-στόχου, η ιστοσελίδα μας μπορεί να συνδεθεί με εκείνη την ιστοσελίδα, αρκεί ο χρήστης να διαλέξει το αντικείμενο που περιέχεται στο στοιχείο <a>.

Ας δούμε ένα παράδειγμα.

```
<a href="https://www.upatras.gr">The University of Patras</a>
```

Εδώ έχουμε το στοιχείο <a> το οποίο περιέχει το γνώρισμα href, το οποίο έχει ως τιμή τη διεύθυνση ενός πόρου, μια ιστοσελίδα-στόχο. Το περιεχόμενο του στοιχείου είναι το κείμενο “The University of Patras”, είναι αυτό που θα δει ο χρήστης και όταν το επιλέξει θα ξεκινήσει ένα αίτημα HTTP για ανάκτηση του πόρου και, αν όλα πάνε καλά, θα φορτωθεί η ιστοσελίδα-στόχος με την οποία θα αλληλεπιδράσει ο χρήστης. Οι φυλλομετρητές έχουν καθορισμένη μορφοποίηση των υπερσυνδέσμων (π.χ. χρώμα μπλε με υπογράμμιση), ώστε ο χρήστης να αντιληφθεί άμεσα ότι αυτό το αντικείμενο, αυτό το τμήμα του κειμένου ή αυτή η εικόνα, αποτελεί υπερσύνδεσμο και, αν επιλεγεί, θα οδηγηθεί σε μια άλλη σελίδα.

Ας δούμε μερικές βασικές αρχές που διέπουν τη σχεδίαση υπερσυνδέσμων στις ιστοσελίδες μας.

Κανόνας νούμερο 1: Θα πρέπει να προσέξουμε ιδιαίτερα το κείμενο του υπερσυνδέσμου. Το κείμενο του υπερσυνδέσμου πρέπει να είναι περιγραφικό, δηλαδή να περιγράφει τον στόχο, γιατί είναι βασική πληροφορία που θα οδηγήσει τον χρήστη να επιλέξει έναν υπερσύνδεσμο έναντι κάποιου άλλου και για να αποφεύγονται κοινοτυπίες τύπου «πατήστε εδώ», «εκεί» κ.λπ. Ας δούμε ένα παράδειγμα. Εδώ έχω δύο υπερσυνδέσμους: (α) *Πανεπιστήμιο Πατρών* και (β) *Πατήστε εδώ για το Πανεπιστήμιο Πατρών*. Ποια είναι η καλύτερη επιλογή; Η πρώτη είναι πιο περιγραφική και σαφής, η υπόδειξη «πατήστε εδώ» δεν χρειάζεται, αφού αυτό προκύπτει από την εμφάνιση του υπερσυνδέσμου.

Κανόνας νούμερο 2: Θα πρέπει να παρέχονται πληροφορίες στον χρήστη για τη συνέπεια επιλογής υπερσυνδέσμου, για παράδειγμα, αν πατώντας έναν υπερσύνδεσμο θα κατεβάσει ένα αρχείο, θα χρειάζεται μια επέκταση του φυλλομετρητή (plug-in), θα πρόκειται να οδηγηθεί σε ένα νέο παράθυρο κ.λπ. Ακολουθούν σχετικά παραδείγματα. Αν ο υπερσύνδεσμος “download report” συνεπάγεται κατέβασμα ενός αρχείου τύπου pdf, μέγεθος 10 Mb, αυτή είναι μια χρήσιμη πληροφορία για να ξέρει ο χρήστης αν πρέπει να επιλέξει αυτό τον υπερσύνδεσμο. Το ίδιο αν ο υπερσύνδεσμος που περιγράφεται με τις λέξεις *«Δες το βίντεο»* θα ενεργοποιήσει μια ροή βίντεο υψηλής ανάλυσης σε ένα καινούργιο παράθυρο.

Κανόνας νούμερο 3: Δεν είναι σωστό να χρησιμοποιούμε την ίδια τη διεύθυνση του πόρου ως υπερσύνδεσμο. Αυτή είναι μια κακή πρακτική, αφού η ανάγνωση του URL ακούγεται ιδιαίτερα άσχημα.

Κανόνας νούμερο 4: Το κείμενο του υπερσυνδέσμου δεν θα πρέπει να είναι ιδιαίτερα μεγάλο. Σκεφτείτε αν, για ένα άτομο με ειδικές ανάγκες το οποίο θα ακούσει τον υπερσύνδεσμο, πρέπει ο screen reader να διαβάσει ένα μακρύ κείμενο που αποτελεί το κείμενο του υπερσυνδέσμου. Τέλος, σκεφτείτε ότι αν χρησιμοποιήσετε την ίδια κοινότοπη λέξη, όπως «πατήστε εδώ», για μια σειρά από υπερσυνδέσμους σε μια σελίδα, τότε θα είναι δύσκολο για ένα άτομο με ειδικές ανάγκες είτε για έναν απλό χρήστη να ξεχωρίσει τους υπερσυνδέσμους.

Γνωρίσματα της ετικέτας <a>

Το πρώτο γνώρισμα είναι το γνώρισμα href που σημαίνει *hyperlink reference* και ορίζει τη διεύθυνση του στόχου. Έχουμε ήδη δει μερικά παραδείγματα. Ένα δεύτερο βασικό γνώρισμα που είναι χρήσιμο είναι το γνώρισμα title. Είναι προαιρετικό γνώρισμα, το οποίο περιέχει ένα κείμενο περιγραφής του στόχου και εμφανίζεται ως “tooltip”, ένα μικρό αναδυόμενο κείμενο, όταν με τη δεικτική συσκευή ο χρήστης βρεθεί πάνω από τον υπερσύνδεσμο.

Ένα παράδειγμα:

```
<a href="https://www.upatras.gr" title="Το τρίτο ελληνικό
Πανεπιστήμιο">Πανεπιστήμιο Πατρών</a>.
```

Όταν ο χρήστης περάσει με το ποντίκι πάνω από τον υπερσύνδεσμο, βλέπει την εξής πρόσθετη πληροφορία:

Εικόνα 2.16 *Παρουσίαση του περιεχομένου του γνωρίσματος title, όταν το ποντίκι αιωρείται πάνω από το στοιχείο (hover).*

Ένα άλλο γνώρισμα που είναι χρήσιμο είναι το γνώρισμα target. Αυτό ορίζει πώς θα συμπεριφερθεί ο φυλλομετρητής όταν κατέβει η νέα ιστοσελίδα. Για παράδειγμα, όταν πάρει την τιμή _blank, τότε ο πόρος στον οποίο απευθυνόμαστε θα ανοιχτεί σε ένα νέο παράθυρο ή σε μια νέα καρτέλα του φυλλομετρητή. Η τιμή _self που είναι η εξ ορισμού τιμή, φορτώνει τον πόρο στο ίδιο παράθυρο.

Άλλα γνωρίσματα πιο εξειδικευμένα είναι το γνώρισμα download. Αυτό ορίζει ότι η σύνδεση αυτή μάς οδηγεί στο να κατεβάσουμε έναν πόρο, ένα αρχείο συνήθως.

Ένα παράδειγμα:

```
<a href="www.example.com" download="manual.pdf"> Κατεβάστε το εγχειρίδιο
(pdf) </a>
```

Εδώ ορίζουμε ότι αν επιλεγεί αυτός ο υπερσύνδεσμος απλά θα κατέβει το αρχείο manual.pdf.

Μια άλλη περίπτωση υπερσυνδέσμου είναι ο υπερσύνδεσμος που δεν έχει στόχο μια άλλη ιστοσελίδα, αλλά το τοπικό πρόγραμμα αποστολής ηλεκτρονικού ταχυδρομείου. Όταν επιλέξει ο χρήστης αυτό τον υπερσύνδεσμο, τότε, χρησιμοποιώντας το πρωτόκολλο mailto, αναζητείται το τοπικό πρόγραμμα μέιλ και αναδύεται ένα παράθυρο του προγράμματος για να αποστείλουμε ένα ηλεκτρονικό μήνυμα στη διεύθυνση, στον στόχο που ορίζεται από τον υπερσύνδεσμο.

Ένα παράδειγμα:

```
<a href= "mailto:announces@ece.upatras.gr"> μήνυμα στο Τμήμα</a>
```

Μια άλλη ειδική περίπτωση που αξίζει να αναφέρουμε είναι ο υπερσύνδεσμος να αναφέρεται σε ένα σημείο, όχι σε μια άλλη ιστοσελίδα, αλλά σε κάποιο άλλο σημείο στην ίδια σελίδα.

Υπάρχουν διάφοροι τρόποι για να γίνει αυτό. Ας δούμε ένα παράδειγμα. Έστω ότι αναφερόμαστε σε έναν υπερσύνδεσμο που ορίζεται ως #locationA (υπενθυμίζουμε ότι στο URL το σύμβολο # ορίζει ένα επιμέρους στοιχείο ενός πόρου· αν η διεύθυνση αρχίζει με #, εννοείται ότι αποτελεί επιμέρους στοιχείο της ίδιας της σελίδας).

```
<a href="#locationA ">Πηγαίνετε στο Α</a>
```

Πρέπει όμως κάπου στην ιστοσελίδα να υπάρχει αυτός ο στόχος. Ένας τρόπος για να ορίσουμε αυτό τον στόχο είναι να υπάρχει ένα στοιχείο <a> που έχει ως όνομα, με το γνώρισμα name, ακριβώς αυτό το όνομα, το locationA.

```
<a name= "locationA"> το σημείο Α είναι εδώ</a>
```

Μια άλλη μέθοδος είναι να ορίσουμε με ένα στοιχείο το οποίο έχει ως ιδιότητα id το όνομα της αγκύρωσης. Ας δούμε ένα παράδειγμα.

Εάν υπάρχει ένα στοιχείο που έχει ως ταυτότητα id=mailing_address:

```
<h2 id="Mailing_address">Ηλεκτρονικό ταχυδρομείο</h2>
```

Σε αυτό το στοιχείο μπορούμε να αναφερθούμε ως εξής:

```
<p>Η διεύθυνση <a href="#Mailing_address"> μέιλ της εταιρείας μας</a>
βρίσκεται στο τέλος της σελίδας.</p>
```

Στη συνέχεια, ας εξετάσουμε το URL τιμή στο γνώρισμα href ενός στοιχείου <a>. Υπάρχουν απόλυτες διευθύνσεις, που αρχίζουν από το πρωτόκολλο https:// και ορίζουν μια πλήρη διεύθυνση πόρου ή σχετικές διευθύνσεις, οπότε έχουμε μόνο το όνομα του αρχείου αλλά αυτό σημαίνει ότι το αρχείο βρίσκεται στον ίδιο φάκελο που βρίσκεται η ιστοσελίδα.

Εάν ορίσουμε μόνο κάποιον υποφάκελο, όπως για παράδειγμα /a/b/, τότε ο εξυπηρετητής θα αναζητήσει εκεί κάποια από τα προκαθορισμένα ονόματα αρχείων. Το πιο συνηθισμένο είναι το αρχείο index.html. Μπορεί όμως να υπάρχουν και άλλα ονόματα, όπως default.html κ.λπ.

Εάν δεν υπάρχει το προκαθορισμένο αρχείο, δηλαδή ένα από τα παραπάνω, τότε μπορεί να μας επιστρέψει, αν αυτό επιτρέπεται, τον κατάλογο των αρχείων που βρίσκονται σε αυτό τον υποφάκελο. Τέλος, αν δεν υπάρχει κάποιο αρχείο b στον υποφάκελο a, τότε μπορεί να μας επιστρέψει τον κατάλογο a, εφόσον επιτρέπεται. Σε αυτή την περίπτωση το μήνυμα http που θα μας επιστρέψει τον κατάλογο θα έχει τον κωδικό 301, γιατί έχουμε την περίπτωση ανακατεύθυνσης σε διαφορετικό πόρο (Ενότητα 1.4.4).

2.6.9 Πίνακες (στοιχείο <table>)

Σε αυτή την ενότητα θα δούμε το στοιχείο <table> που μας επιτρέπει να δημιουργήσουμε πίνακες, δισδιάστατους υποδοχείς δεδομένων, αντίστοιχους με ένα φύλλο Excel που περιέχουν τα δεδομένα σε κελιά.

Στοιχεία <table>, <tr>, <td>, <th>

Για να ορίσουμε έναν πίνακα στην HTML χρειαζόμαστε έναν συνδυασμό από στοιχεία που είναι παιδιά του στοιχείου <table>. Ως παιδιά του πίνακα ορίζουμε τα στοιχεία <tr> (table rows), που αφορούν τις γραμμές του πίνακα. Κάθε γραμμή περιέχει με τη σειρά της κελιά. Τα κελιά ορίζονται ως στοιχεία <td>, δηλαδή table data.

Ας δούμε ένα παράδειγμα. Έστω ότι επιθυμούμε να δημιουργήσουμε έναν πίνακα δύο γραμμών και δύο στηλών (με 4 κελιά). Αυτό γίνεται με τον παρακάτω κώδικα.

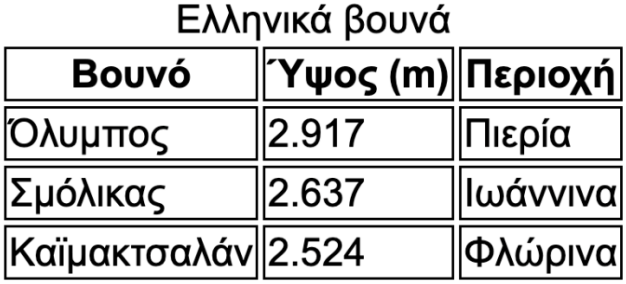
```
<table>
    <tr>
        <td> κελί11 </td>
        <td> κελί12 </td>
    </tr>
    <tr>
        <td> κελί21 </td>
        <td> κελί22 </td>
    </tr>
</table>
```

Παρατηρούμε εδώ λοιπόν στην πρώτη γραμμή <tr> περιέχονται δύο κελιά <td>, με περιεχόμενο *κελί11* και *κελί12*. Στη συνέχεια έχουμε μια δεύτερη γραμμή που πάλι ορίζεται από ένα στοιχείο <tr> που περιέχει επίσης δύο κελιά με περιεχόμενο *κελί21* και *κελί22*. Το πλαίσιο γύρω από κάθε κελί και το συνολικό πλαίσιο γύρω από τον πίνακα, καθορίζονται από φύλλα στιλ, όπως θα δούμε στη συνέχεια.

Ένα στοιχείο <table> δέχεται επίσης έναν άλλο τύπο κελιών αντί για τα <td>, που είναι τα στοιχεία <th> (table header). Είναι τα κελιά της κεφαλίδας του πίνακα, που συνήθως υπάρχουν στην πρώτη σειρά (κελιά που περιγράφουν το περιεχόμενο των αντίστοιχων στηλών) και έχουν πιο έντονη εμφάνιση. Επίσης, ο πίνακας δέχεται ένα ακόμη στοιχείο, το <caption>, το οποίο αφορά τη λεζάντα περιγραφής του πίνακα.

Ας δούμε ένα παράδειγμα. Έστω ότι θέλουμε να δημιουργήσουμε ένα πίνακα με τα στοιχεία των τριών ψηλότερων ελληνικών βουνών, που περιέχει τρεις πληροφορίες για κάθε βουνό, το όνομά του, το ύψος του και την περιοχή που βρίσκεται.

Η τελική εμφάνιση του πίνακα φαίνεται στην ***Εικόνα 2.17***.



Εικόνα 2.17 *Παρουσίαση του πίνακα των ελληνικών βουνών (έχει περιληφθεί εντολή CSS για το πλαίσιο των κελιών).*

Ο πίνακας περιλαμβάνει 4 γραμμές, την πρώτη, που είναι η κεφαλίδα με τα ονόματα των στηλών, και 3 γραμμές με δεδομένα των βουνών. Επίσης, στο πάνω μέρος περιλαμβάνει τη λεζάντα.

Ο κώδικας που παράγει τον παραπάνω πίνακα είναι ο εξής:

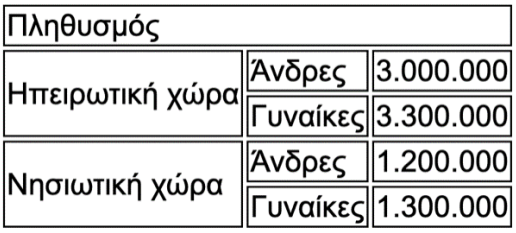
```
<table>
    <caption>Ελληνικά βουνά</caption>
    <tr>
        <th>Βουνό</th>
        <th>Ύψος (m) </th>
        <th>Περιοχή</th>
    </tr>
    <tr>
        <td> Όλυμπος </td>
        <td> 2.917 </td>
        <td> Πιερία </td>
    </tr>
    <tr>
        <td> Σμόλικας </td>
        <td> 2.637 </td>
        <td> Ιωάννινα </td>
    </tr>
    <tr>
        <td> Καϊμακτσαλάν </td>
        <td> 2.524 </td>
        <td> Φλώρινα </td>
    </tr>
</table>
```

Παρατηρούμε ότι οι κεφαλίδες εμφανίζονται με έντονους χαρακτήρες και με κεντρική στοίχιση. Βλέπουμε δηλαδή τα *Βουνό, Ύψος, Περιοχή* ότι είναι με έντονα γράμματα και στο κέντρο των αντίστοιχων στηλών.

Εκτεινόμενα κελιά, γνωρίσματα collspan, rowspan

Ένα άλλο ενδιαφέρον θέμα είναι να πρέπει σε έναν πίνακα κάποια κελιά να εκτείνονται σε περισσότερες από μια στήλες ή επίσης να εκτείνονται σε περισσότερες από μια γραμμές.

Ας δούμε ένα παράδειγμα. Ας πούμε ότι θέλουμε να φτιάξουμε τον παρακάτω πίνακα με στοιχεία από τον πληθυσμό κάποιας χώρας.



Εικόνα 2.18 *Παράδειγμα πίνακα με εκτεινόμενα κελιά: ο πληθυσμός μιας χώρας.*

Παρατηρούμε ότι το πρώτο κελί εκτείνεται και στις τρεις στήλες του πίνακα. Στη συνέχεια υπάρχει στην πρώτη στήλη ένα κελί που περιέχει τις λέξεις *«Ηπειρωτική χώρα»* που εκτείνεται σε δύο γραμμές, και παρακάτω το κελί *«Νησιωτική χώρα»* που επίσης εκτείνεται σε δύο γραμμές. Συνολικά, ο πίνακας περιλαμβάνει πέντε γραμμές και τρεις στήλες, αλλά μόνο 11 κελιά. Πώς γίνεται αυτό με τη χρήση των ετικετών της HTML και των γνωρισμάτων των ετικετών αυτών;

Όπως θα δούμε, θα χρησιμοποιήσουμε το γνώρισμα collspan, που επιτρέπει την έκταση του αντίστοιχου κελιού σε πολλές στήλες, και του γνωρίσματος rowspan, που επιτρέπει την επέκταση του αντίστοιχου κελιού σε πολλές γραμμές.

Ο παρακάτω κώδικας παράγει αυτό το αποτέλεσμα:

```
<table>
    <tr >
        <td colspan="3">Πληθυσμός</td>
    </tr>
    <tr>
        <td rowspan="2">Ηπειρωτική χώρα</td>
        <td>Άνδρες</td>
        <td>3.000.000</td>
    </tr>
    <tr>
        <td>Γυναίκες</td>
        <td>3.300.000</td>
    </tr>
    <tr>
        <td rowspan="2">Νησιωτική χώρα</td>
        <td>Άνδρες</td>
        <td>1.200.000</td>
    </tr>
    <tr>
        <td>Γυναίκες</td>
        <td>1.300.000</td>
    </tr>
</table>
```

Όπως βλέπουμε, θέτοντας το γνώρισμα collspan="3" στο πρώτο στοιχείο, έχει ως συνέπεια το κελί να εκταθεί σε τρεις στήλες. Το ίδιο με το πρώτο στοιχείο της δεύτερης γραμμής που παίρνει το γνώρισμα rowspan="2", με αποτέλεσμα αυτό να καλύπτει δύο γραμμές. Με αυτό τον τρόπο, λοιπόν, τα κελιά εκτείνονται είτε οριζόντια, καλύπτοντας πολλές στήλες είτε κατακόρυφα, καλύπτοντας πολλές γραμμές, κάτι που πολύ συχνά χρειαζόμαστε και σε φύλλα Excel κ.λπ.

Κάποια ακόμη στοιχεία που χρησιμοποιούνται σε πίνακες είναι τα στοιχεία <thead> (table head) για την κεφαλίδα του πίνακα, <tfoot> (table foot) για το υποσέλιδο του πίνακα και <tbody> (table body) για το σώμα του πίνακα. Τα στοιχεία αυτά είναι δομικά στοιχεία του πίνακα. Αυτά μπορεί να συνδυαστούν με στιλ έτσι ώστε αυτά τα τμήματα του πίνακα να έχουν ορισμένο στιλ εμφάνισης. Επίσης, μπορεί να επαναλαμβάνονται. Παραδείγματος χάρη, δομικά στοιχεία που μπορεί να επαναλαμβάνονται είναι ένας πίνακας που εκτείνεται σε πολλές σελίδες και σε κάθε σελίδα έχει κεφαλίδα και υποσέλιδο που μας υπενθυμίζει το περιεχόμενο του.

Μορφοποίηση πινάκων

Θα κάνουμε στη συνέχεια μια σύντομη αναφορά στον τρόπο που ορίζουμε το στιλ εμφάνισης των στοιχείων ενός πίνακα. Περισσότερες λεπτομέρειες για την ακριβή σύνταξη των εντολών CSS είναι αντικείμενο επόμενου κεφαλαίου (κεφάλαιο 4).

Παράδειγμα 1

```
table, th, td {border: 2px solid black;}
```

Στην εντολή αυτή ορίζουμε τα στοιχεία <table>, <th> και <td>, που είναι στοιχεία που ορίζουν τον πίνακα και τα κελιά του. Για τα στοιχεία αυτά ο κανόνας ορίζει ότι οι το πλαίσιό τους είναι πλάτους 2 πίξελ, με συνεχή γραμμή μαύρου χρώματος, άρα κάθε κελί θα έχει το δικό του πλαίσιο.

Παράδειγμα 2

```
th, td {padding: 10px;}
```

Η εντολή αυτή ορίζει ότι τα στοιχεία <th> και <td> θα έχουν εσωτερικό περιθώριο μεταξύ των δεδομένων και του πλαισίου του αντίστοιχου κελιού 10 πίξελ.
Παράδειγμα 3

```
table {border-spacing: 5px;}
```

Η απόσταση μεταξύ κελιών ορίζεται από την παράμετρο border-spacing που εδώ ορίζεται να είναι 5 πίξελ.
Παράδειγμα 4

```
th { text-align: left;}
```

Εδώ ορίζουμε τα κελιά της κεφαλίδας να στοιχηθούν αριστερά, όπως και τα κελιά δεδομένων, η παράμετρος text-align μάς επιτρέπει να στοιχίσουμε το περιεχόμενο των κελιών δεξιά, αριστερά ή στο κέντρο, ενώ υπάρχει και η vertical-align η οποία μάς επιτρέπει να στοιχίσουμε πάνω, κάτω ή στο κέντρο το περιεχόμενο των κελιών κατά την κατακόρυφη διάσταση.

Χρήση του στοιχείου <table>

Θα κλείσουμε αυτή την ενότητα με ένα σχόλιο για τη χρήση του στοιχείου <table>. Το στοιχείο αυτό χρησιμοποιείται από κάποιους σχεδιαστές εκτός από δημιουργία πινάκων και για διάταξη του περιεχομένου της σελίδας, μια πρακτική που ήταν συνηθισμένη στα πρώτα χρόνια του διαδικτύου αλλά που τείνει να εξαλειφθεί. Η σελίδα μπορεί να οριστεί ως ένας πίνακας και τα κελιά να περιέχουν τα επιμέρους στοιχεία της σελίδας, εικόνες, κείμενα κ.λπ. Αυτή είναι όμως μια πολύ κακή πρακτική και θα πρέπει να αποφεύγεται. Υπάρχουν πολλοί σοβαροί λόγοι για τους οποίους το <table> δεν πρέπει να χρησιμοποιείται για διάταξη περιεχομένου. Διάταξη περιεχομένου, όπως θα δούμε στα επόμενα κεφάλαια, μπορεί να γίνει χρησιμοποιώντας μηχανισμούς της CSS (κεφάλαιο 5). Μερικοί λόγοι για τους οποίους αυτή η πρακτική πρέπει να αποφεύγεται είναι οι εξής:

- Τα στοιχεία <table> μειώνουν την προσβασιμότητα για χρήστες με προβλήματα όρασης. Ο ορισμός στιλ πινάκων είναι πιο περίπλοκος από ό,τι με τις τεχνικές διάταξης μέσω CSS, με αποτέλεσμα σύγχυση των screen reader.
- Οι πίνακες είναι πιο δύσκολο να συντηρηθούν, αφού οι διατάξεις των στοιχείων πίνακα είναι σύνθετες.
- Οι πίνακες δεν προσαρμόζονται σε διαφορετικές συσκευές, γιατί έχουν σταθερό μέγεθος που ορίζεται από το περιεχόμενό τους.

Για όλους αυτούς τους λόγους, οι πίνακες θα πρέπει να χρησιμοποιούνται μόνο για αυτό το οποίο δημιουργήθηκαν, δηλαδή για την οργανωμένη παρουσίαση δεδομένων.

2.7 Ασκήσεις

Άσκηση 1

Να δημιουργήσετε μια σελίδα που περιέχει το βιογραφικό σας.

Στο πάνω μέρος να υπάρχει ένας πίνακας περιεχομένων με συνδέσμους στις επιμέρους ενότητες του βιογραφικού, να περιλάβετε ενότητες για προσωπικά στοιχεία, σπουδές, εργασιακή εμπειρία, ενδιαφέροντα. Να περιλάβετε υπερσυνδέσμους σε σελίδες που σας αρέσουν.

Άσκηση 2

Έστω ο πίνακας στην ***Εικόνα 2.19***. Ζητείται:

1. Να δημιουργήσετε ιστοσελίδα με τον πίνακα.
2. Να αλλάξετε την εμφάνιση αυξάνοντας το πλάτος του πλαισίου των κελιών σε 4px.
3. Να προσθέστε λεζάντα και υποσέλιδο.

Σπονδυλωτά	Πτηνά		Κότα
	Ερπετά		Σαύρα
	Αμφίβια		Βάτραχος
	Ψάρια		Καρχαρίας
	Θηλαστικά	Τρωκτικά	Ποντίκι
		Προβοσκιδωτά	Ελέφαντας
		Πρωτεύοντα	Πίθηκος
		Κητώδη	Δελφίνι
		Σαρκοφάγα	Λιοντάρι
		Οπληφόρα	Άλογο
		Χειρόπτερα	Νυχτερίδα
Ασπόνδυλα	Μαλάκια		Καλαμάρι
	Εχινόδερμα		Αχινός
	Αρθρόποδα		Καβούρι
	Σπόγγοι		Σφουγγάρι
	Κνιδόζωα		Τσούχτρα
	Σκώληκες		Σκουλήκι

Εικόνα 2.19 *Πίνακας κατηγοριοποίησης ζωντανών οργανισμών.*

2.8 Ερωτήσεις αυτοαξιολόγησης

1. Μια ιστοσελίδα δεν μπορεί να υπάρχει αν δεν έχει οριστεί ένα αρχείο HTML που να συνοδεύεται από αντίστοιχο CSS και JS.
   – Σωστό/Λάθος
2. Οι διαφορές της HTML5 από προηγούμενες εκδόσεις περιλαμβάνουν (σημειώστε όλα όσα ισχύουν):
   1. πιο σύνθετη σύνταξη.
   2. νέα στοιχεία φόρμας, όπως date, time κ.λπ.
   3. νέα στοιχεία για πολυμέσα, όπως video.
   4. νέα σύνταξη που στηρίζεται στην XML (XHTML).
   5. νέα στοιχεία δόμησης της σελίδας, όπως header, footer.
3. Το πρότυπο HTML5 ορίστηκε από το W3C
   1. το 1994
   2. το 1997
   3. το 2014
   4. το 2017
4. Παλαιότερες εκδόσεις της HTML, όπως η HTML4 και η HTML3, δεν υποστηρίζονται πλέον από τους σύγχρονους φυλλομετρητές.
   – Σωστό/Λάθος
5. Η πρώτη ετικέτα ενός αρχείου html είναι
   1. <start>
   2. <body>
   3. <head>
   4. <html>
6. Στο περιβάλλον *VS Code*, μόλις σώσουμε το αρχείο html, αυτό εμφανίζεται αυτόματα στον φυλλομετρητή
   – Σωστό/Λάθος
7. Για να αλλάξουμε το χρώμα της ιστοσελίδας από λευκό σε πράσινο δημιουργήσαμε ένα στοιχείο <style>, σε ποιο τμήμα του αρχείου hmtl θα πρέπει να το τοποθετήσουμε;
   1. <body>
   2. <head>
8. Μπορούμε να βάλουμε το στοιχείο <p> που περιλαμβάνει μια παράγραφο κειμένου μέσα στο τμήμα

<head>;
   – Ναι/Όχι
9. Η ετικέτα και η ετικέτα <P> είναι διαφορετικές ετικέτες.
   – Σωστό/Λάθος
10. Ποια θα πρέπει να είναι η τελευταία ετικέτα ενός αρχείου html;
   1. </head>
   2. </html>
   3. </doctype>
   4. </body>
11. Ποιο από τα παρακάτω στοιχεία θα εμφανιστεί διαφορετικά από τα άλλα;
   1. “ This is my Paragraph ”
   2. “ This is my Paragraph ”
   3. “ This is my Paragraph ”
   4. “ This is my Paragraph ”
   5. κανένα, όλα θα εμφανιστούν το ίδιο
12. Τι θα δει ο χρήστης στη σελίδα ως αποτέλεσμα του κώδικα <h1>This is a heading & it is the "biggest" one in this page>/h1<
   1. This is a heading & it is the "biggest" one in this page
   2. <h1>This is a heading & it is the "biggest" one in this page>/h1<
   3. <h1>This is a heading & it is the "biggest" one in this page>/h1<
   4. <h1>This is a heading & it is the "biggest" one in this page</h1>
13. <!-- Αν αυτή η γραμμή ήταν στο πρόγραμμά σας, θα την έβλεπε ο χρήστης -->;
   – Ναι / Όχι
14. Το DOM είναι μια ιεραρχική δομή από αντικείμενα που αντιπροσωπεύουν τα στοιχεία μιας ιστοσελίδας.
   – Σωστό/Λάθος
15. Το DOM παρέχει διεπαφές προς γλώσσες προγραμματισμού όπως η JavaScript.
   – Σωστό/Λάθος
16. Η JavaScript μπορεί να αλληλεπιδράσει με τους κόμβους του DOM ως εξής (σημειώστε όλα όσα ισχύουν):
   1. Να μετακινήσει έναν κόμβο του DOM.
   2. Να καταργήσει έναν κόμβο του DOM.
   3. Να τροποποιήσει την εμφάνιση του αντίστοιχου στοιχείου του κόμβου.
   4. Να μετρήσει όλους τους κόμβους που ικανοποιούν μια συνθήκη.
17. Η ετικέτα <title> :
   1. Ορίζει τον τίτλο πριν από το όνομα του δημιουργού της ιστοσελίδας, π.χ. “Mr”, “Miss”, “Mrs” κ.λπ.
   2. Ορίζει τον τίτλο που θα εμφανίζεται στο πάνω μέρος της ιστοσελίδας στον φυλλομετρητή.
   3. Ορίζει το όνομα της ιστοσελίδας όπως την αναγνωρίζει μια μηχανή αναζήτησης, όπως η google.
   4. Ορίζει το όνομα του αρχείου που θα έχει η ιστοσελίδα αν αυτή αποθηκευτεί στον τοπικό δίσκο με την επιλογή «Αποθήκευση ως».
18. Η ετικέτα <meta>:
   1. Επιτρέπει την εισαγωγή μεταδεδομένων, όπως η κωδικοποίηση χαρακτήρων, το όνομα του δημιουργού, περιγραφή του περιεχομένου κ.λπ.
   2. Είναι η ετικέτα που χρησιμοποιείται για τον ορισμό της γλώσσας του εγγράφου HTML, όπως “fr”, “en”, “el” κ.λπ.
   3. Είναι η ετικέτα που ορίζει την κωδικοποίηση χαρακτήρων όπως “utf-8”.
   4. Ορίζει τι ακολουθεί μετά (“meta”) το φόρτωμα της ιστοσελίδας.
19. Η ετικέτα <link> χρησιμοποιείται:
   1. Για τη δημιουργία υπερσυνδέσμων με την ιστοσελίδα.
   2. Για τη σύνδεση με εξωτερικά αρχεία, όπως αρχεία css.
   3. Για τη σύνδεση με εξωτερικά προγράμματα, όπως αρχεία JavaScript ή Python.
20. Η ετικέτα <script> χρησιμοποιείται για (επιλέξτε όλα όσα ισχύουν):
   1. Την εισαγωγή κώδικα JavaScript στην ιστοσελίδα.

    2. Την εισαγωγή ενός συνδέσμου σε ένα εξωτερικό αρχείο που περιέχει κώδικα JavaScript.
    3. Την εισαγωγή ενός κώδικα σε οποιαδήποτε γλώσσα, αρκεί αυτή να οριστεί με την παράμετρο type.
21. Η εικόνα favicon, που εμφανίζεται στην πάνω αριστερά γωνία μιας ιστοσελίδας, εισάγεται μέσω του στοιχείου:
    1. <img>
    2. <favicon>
    3. <link>
    4. <favorite-icon>
22. Το γνώρισμα lang που ορίζει τη γλώσσα του περιεχομένου ενός τμήματος ή και ολόκληρης της ιστοσελίδας μπορεί να οριστεί σε οποιοδήποτε επίπεδο, από την ετικέτα <html> μέχρι οποιοδήποτε στοιχείο.
    – Σωστό/Λάθος
23. Πώς ορίζουμε ότι μια ιστοσελίδα ακολουθεί τις διαστάσεις της οθόνης;
    1. <html width=100%>
    2. <meta viewport="device-width">
    3. <meta name="viewport" content="width=device-width">
    4. <viewport width="device-width">
24. Το στοιχείο <head> είναι υποχρεωτικό σε ένα έγγραφο HTML.
    – Σωστό/Λάθος
25. Σημειώστε ποια από τα παρακάτω στοιχεία είναι block elements.
    1. <a>
    2.
    3.
    4.
    5. <h3>
26. Ένα στοιχείο <h6> περιέχει κείμενο με μεγαλύτερη γραμματοσειρά από ένα στοιχείο <h3>.
    – Σωστό/Λάθος
27. Ένα στοιχείο μπορεί να περιέχει άλλα στοιχεία ως υπο-παραγράφους.
    – Σωστό/Λάθος
28. Αν θέλουμε να παρουσιάσουμε ένα κώδικα python σε μια ιστοσελίδα, θα πρέπει να τον ενσωματώσουμε σε περιοχή:
    1. <code></code>
    2. <script></script>
    3. <pre></pre>
    4. <blockquote></blockquote>
    5.
29. Για να δώσουμε έμφαση με έντονους χαρακτήρες σε μια λέξη:
    1. Πρέπει να τη βάλουμε σε ένα στοιχείο .
    2. Πρέπει να τη βάλουμε σε ένα στοιχείο .
    3. Πρέπει να τη βάλουμε σε ένα στοιχείο <emphasis>.
    4. Πρέπει να τη βάλουμε σε ένα στοιχείο <h1>.
30. Για να γράψω μια δευτεροβάθμια εξίσωση χρειάζομαι:
    1. την ετικέτα <sub>.
    2. την ετικέτα <sup>.
    3. την ετικέτα <equation>.
31. Οι ετικέτες <sup> και <sub> αλλάζουν τη θέση του κειμένου που περικλείουν χωρίς να αλλάζουν το μέγεθος της γραμματοσειράς.
    – Σωστό/Λάθος
32. Μια λίστα με αρίθμηση των στοιχείων της αρχίζει με την ετικέτα:
    1.
    2.
    3.
    4. <al>
33. Ένα στοιχείο που περιέχεται στις ετικέτες θα έχει την ίδια εμφάνιση ανεξάρτητα από τις

ετικέτες που το περιβάλλουν.
   – Σωστό/Λάθος

34. Έστω μια λίστα που ορίζεται με την ετικέτα <ol reversed> και περιέχει 8 στοιχεία. Τι θα υπάρχει στην αρχή του πρώτου στοιχείου;
    1. μια βούλα (bullet)
    2. ο αριθμός 1
    3. ο αριθμός 8
    4. μια παύλα (-)
35. Έστω μια λίστα που ορίζεται με την ετικέτα <ul start="8" reversed> και περιέχει 4 στοιχεία. Τι θα υπάρχει στην αρχή του δεύτερου στοιχείου;
    1. μια βούλα (bullet)
    2. ο αριθμός 8
    3. ο αριθμός 7
    4. μια παύλα (-)
36. Ποια ιδιότητα του στοιχείου <a> είναι απαραίτητη;
    1. href
    2. alt
    3. src
    4. target
37. Αν έχουμε ένα στοιχείο b, ο χρήστης θα δει τον εξής υπερσύνδεσμο:
    1. b.html
    2. b
    3. href
38. Το γνώρισμα title σε ένα στοιχείο <a> ορίζει:
    1. Τον τίτλο του παράθυρου στόχου του υπερσυνδέσμου.
    2. Τον τίτλο του ίδιου του αρχείου που περιέχει το στοιχείο <a>.
    3. Το κείμενο περιγραφής του υπερσυνδέσμου-στόχου.
    4. Το κείμενο που θα πρέπει να επιλέξει ο χρήστης για να ακολουθήσει τον υπερσύνδεσμο.
39. Αν ορίσουμε href = "/dir1/dir2/" σε ένα στοιχείο <a>, ποια σελίδα θα μας επιστρέψει ο server αν επιλέξουμε τον υπερσύνδεσμο;
    1. Τον κατάλογο αρχείων του dir2.
    2. Το προκαθορισμένο αρχείο index.html στον φάκελο dir2 αν υπάρχει.
    3. Ένα μήνυμα HTTP 404 file not found.
    4. Ένα μήνυμα HTTP 301 redirect.
40. Έστω ο παρακάτω πίνακας:

```
<table>
    <tr>
        <td rowspan=2> a </td>
        <td> b </td>
    </tr>
    <tr>
        <td> c </td>
        <td> d </td>
    </tr>
</table>
```

    Πόσες στήλες έχει ο πίνακας;
    Απάντηση: _____
41. Πόσες σειρές έχει ο πίνακας της ερώτησης 40;
    Απάντηση: _____
42. Πόσα κελιά έχει ο πίνακας της ερώτησης 40;
    Απάντηση: _____
43. Τι διαφορά έχει ένα στοιχείο <th> από ένα στοιχείο <td>;
    1. Καμία και τα δύο αφορούν κελιά ενός πίνακα.
    2. Τα στοιχεία <th> προηγούνται των στοιχείων <td>.

3. Το περιεχόμενο των <th> εμφανίζεται με έντονους χαρακτήρες.
4. Τα στοιχεία <th> ανήκουν στο στοιχείο <thead>.

44. Γιατί δεν συνιστάται η χρήση της <table> για οργάνωση του περιεχομένου μιας ιστοσελίδας; (επιλέξτε όλα όσα ταιριάζουν)
    1. Τα στοιχεία <table> μειώνουν την προσβασιμότητα του περιεχομένου της σελίδας.
    2. Τα στοιχεία του <table> δεν έχουν τόσο πλούσιες δυνατότητες εμφάνισης (border, στοίχιση κ.λπ.) όσο άλλα στοιχεία.
    3. Το περιεχόμενο των <table> είναι δύσκολο να συντηρηθεί λόγω της μεγάλης πολυπλοκότητάς τους.
    4. Τα στοιχεία του <table> δεν προσαρμόζονται εύκολα σε διαφορετικές διαστάσεις συσκευών.

2.9 Βιβλιογραφία και Αναφορές

Το πρότυπο HTML συντηρείται από την κοινότητα WHATWG.

Στο διαδίκτυο υπάρχουν πολλές πηγές για εκμάθηση της HTML, καθώς και για αναφορά στα στοιχεία της γλώσσας. Η MDN είναι μια καλή πηγή για εισαγωγικά και προχωρημένα μαθήματα. Επίσης, η w3schools.

Βιβλία για την HTML συνήθως περιλαμβάνουν ένα σύνολο από άλλες τεχνολογίες, όπως η CSS και η JavaScript. Στην ελληνική βιβλιογραφία, εκτός από το βιβλίο των Δουληγέρη κ.ά. (2021), υπάρχει ακόμη το βιβλίο των Μαρκατσέλα και Ξαρχάκου (2013), το οποίο όμως δίνει κυρίως έμφαση στην CSS, ενώ έχουν μεταφραστεί κάποια συγγράμματα και διατίθενται από τον Εύδοξο. Παράδειγμα, το βιβλίο των Kyrnin και Morrison (2021) και το βιβλίο των Lemay, Coburn και Kyrnin (2016).

Επιπλέον, οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Για αυτό το κεφάλαιο το σχετικό μάθημα είναι οι εισαγωγικές διαλέξεις του μαθήματος «Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, Javascript».

Ελληνόγλωσσες

Αβούρης, Ν. (2018). Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, JavaScript. Ανοικτό διαδικτυακό μάθημα. https://mathesis.cup.gr

Δουληγέρης, Χ., Μαυροπόδη, Ρ., Κοπανάκη, Ε., & Καραλής, Α. (2021). *Τεχνολογίες και Προγραμματισμός στον Παγκόσμιο Ιστό* (2η έκδ.). Εκδόσεις Νέων Τεχνολογιών. Κωδικός βιβλίου στον Εύδοξο: 102125023.

Kyrnin, J., & Meloni, J. (2021). *Sams Teach Yourself HTML5, CSS and Javascript* (3rd ed.). Εκδόσεις Γκιούρδας.

Lemay, L., Coburn, R., & Kyrnin, J. (2016). *Sams Teach Yourself HTML, CSS & JavaScript* (7th ed). Εκδόσεις Γκιούρδας. Κωδικός Βιβλίου στον Εύδοξο: 59357307.

Μαρκατσέλας, Μ., & Ξαρχάκος, Κ. (2013). *CSS3 & Εφαρμογές.* Εκδόσεις Άβακας, Αθήνα. Κωδικός βιβλίου στον Εύδοξο: 68369452.

Κεφάλαιο 3. Προχωρημένα θέματα της HTML

Σύνοψη

Το δεύτερο αυτό κεφάλαιο για την HTML παρουσιάζει τον χειρισμό και την ενσωμάτωση πολυμέσων και άλλων αντικειμένων (εικόνες, βίντεο, ήχος, iframe, object) σε μια ιστοσελίδα. Συγκεκριμένα, θα εμβαθύνουμε στις δυνατότητες που έχει η HTML5, η οποία υποστηρίζει πολυμέσα σε πολύ μεγαλύτερο βαθμό από ό,τι οι προηγούμενες εκδόσεις.

Θα δούμε επίσης στοιχεία με σημασιολογικό περιεχόμενο που επιτρέπουν τη διάταξη και την κατηγοριοποίηση του περιεχομένου μιας σελίδας. Επίσης, θα γίνει αναφορά στις φόρμες HTML και θα παρουσιαστούν οι πλούσιες επιλογές που παρέχει η HTML5 για δημιουργία φορμών σήμερα. Η παράθεση οδηγιών σχεδιασμού φορμών και η συζήτηση για τον ορθό σχεδιασμό τους είναι η τελευταία ενότητα αυτού του κεφαλαίου.

Στο πλαίσιο αυτού και του προηγούμενου κεφαλαίου, είδαμε ήδη μερικά αποσπάσματα από εντολές CSS και κώδικα JavaScript. Στο επόμενο κεφάλαιο θα δούμε πιο συστηματικά τη γλώσσα παρουσίασης περιεχομένου CSS και στη συνέχεια τη βιβλιοθήκη της Bootstrap.

Προαπαιτούμενη γνώση

Μελέτη των κεφαλαίων 1 και 2.

3.1 Διαχείριση εικόνων

Όπως γνωρίζουμε, μια ιστοσελίδα, εκτός από κείμενο, που ήταν το αντικείμενο του προηγούμενου κεφαλαίου, περιλαμβάνει και άλλα μέσα, όπως εικόνες, βίντεο και ήχους. Θα ξεκινήσουμε από τις εικόνες.

Μια εικόνα μπορεί να ενσωματωθεί σε μια σελίδα ως στοιχείο <img> image, δηλαδή, που να μας επιτρέπει να ενσωματώσουμε ένα εξωτερικό αρχείο που περιέχει εικόνα. Επίσης, θα δούμε το στοιχείο <picture> που χρησιμοποιείται λιγότερο, όμως μας επιτρέπει να ενσωματώσουμε πολλαπλές εικόνες διαφορετικών διαστάσεων ούτως ώστε να ανταποκρινόμαστε σε διαφορετικά μεγέθη συσκευών.

Ας δούμε πρώτα το στοιχείο <img>. Το στοιχείο αυτό είναι κενό στοιχείο (δεν απαιτείται ετικέτα τερματισμού του).

Το κύριο γνώρισμα του στοιχείου αυτού είναι το src (source), δηλαδή η πηγή, η διεύθυνση του διαδικτύου που υπάρχει η εικόνα. Αν η εικόνα είναι στον ίδιο φάκελο που είναι η ιστοσελίδα, αρκεί το όνομα του αρχείου. Διαφορετικά, θα πρέπει να δοθεί το URL ή το σχετικό μονοπάτι.

Ένα άλλο γνώρισμα το οποίο είναι ιδιαίτερα σημαντικό, όπως θα εξηγηθεί στη συνέχεια, και θεωρείται υποχρεωτικό είναι το alt (alternate text). Το γνώρισμα αυτό αφορά μια εναλλακτική λεκτική περιγραφή, μέσω κειμένου, της εικόνας. Είναι πολύ σημαντικό γνώρισμα για πληρότητα παρουσίασης και ιδιαίτερα για εξυπηρέτηση ατόμων με ειδικές ανάγκες.

Εδώ βλέπουμε ένα παράδειγμα εισαγωγής ενός στοιχείου <img> που, όπως αναφέρθηκε, δεν απαιτεί ετικέτα τερματισμού </img>.

```
<img src="images/dog.png" alt="το σκυλί μου">
```

3.2 Μορφότυποι εικόνων

Ας δούμε τώρα τι τύπους εικόνων μπορούμε να εισαγάγουμε σε μια ιστοσελίδα. Αυτοί είναι οι τέσσερις πιο βασικοί τύποι:

- **GIF** (Graphic Interchange Format) – επιτρέπει μόνο 256 χρώματα, επιτρέπει κίνηση (animated gif).
- **JPEG** (Joint Photographic Experts Group) – υποστηρίζει υψηλή συμπίεση εικόνων, είναι συνεπώς η πιο συνήθης επιλογή. Όμως, δεν υποστηρίζει κίνηση και διαφάνεια.
- **PNG** (Portable Network Graphics) – υποστηρίζει διαφάνεια, συνεπώς είναι χρήσιμο και για μη ορθογώνιες εικόνες.
- **SVG** (Scalable Vector Graphics), μαθηματικά ορισμένο περιεχόμενο σε XML – υποστηρίζει κίνηση, είναι χρήσιμο για μεγάλες κλίμακες.

Υπάρχουν κι άλλοι τύποι, όπως PDF, TIFF κ.λπ., αλλά οι παραπάνω είναι αυτοί που χρησιμοποιούνται περισσότερο.

Ο μορφότυπος GIF είναι ο πιο παλιός, επιτρέπει animation, με τη μορφή animated GIF, όμως είναι πολύ περιορισμένος ως προς τον αριθμό χρωμάτων.

Ο JPEG είναι ο πιο συνηθισμένος μορφότυπος εικόνων στο διαδίκτυο. Επιτρέπει εκατομμύρια χρώματα και αποχρώσεις, υψηλή συμπίεση εικόνων και είναι η πιο συνηθισμένη επιλογή. Όμως, δεν υποστηρίζει κίνηση και επίσης –κάτι σημαντικό– δεν υποστηρίζει διαφάνεια. Δηλαδή, δεν μας επιτρέπει να έχουμε τμήματα της εικόνας διαφανή για να φαίνεται το υπόβαθρο, και αυτό σημαίνει ότι είμαστε υποχρεωμένοι μέσω JPEG να έχουμε μόνο ορθογώνιες εικόνες.

Ο PNG είναι μια εναλλακτική επιλογή του JPEG. Ο PNG είναι μορφότυπος ο οποίος έχει τα ίδια χαρακτηριστικά όσον αφορά το πλήθος των χρωμάτων, υποστηρίζει όμως διαφάνεια και συνεπώς είναι χρήσιμος για μη ορθογώνιες εικόνες.

Τέλος, το SVG επιτρέπει τον ορισμό μέσω γραφικών εξισώσεων των εικονικών στοιχείων. Στην ουσία, το περιεχόμενο της εικόνας αποτυπώνεται ως διανύσματα, είναι σε μορφή XML και υποστηρίζει κίνηση. Έχει ενδιαφέρον, αφού υποστηρίζει μεγέθυνση πολύ μεγάλης κλίμακας εικόνων, γιατί διατηρεί τα βασικά στοιχεία, αφού πρόκειται για διανυσματική αναπαράσταση, ανεξαρτήτως κλίμακας. Οι μορφότυποι αυτοί υποστηρίζονται από όλους τους σημαντικούς φυλλομετρητές σήμερα, βλέπε τον σχετικό πίνακα από το λήμμα της Βικιπαίδειας.

3.2.1 Η σημασία του γνωρίσματος alt

Το γνώρισμα alt του στοιχείου <img>, όπως ήδη αναφέρθηκε, περιέχει ένα εναλλακτικό κείμενο αντί για την εικόνα. Είναι πολύ σημαντικό, γιατί για ένα άτομο με ειδικές ανάγκες, όπως για παράδειγμα με δυσκολία στην όραση, ο screen reader, ο αυτόματος αναγνώστης του περιεχομένου της ιστοσελίδας, στη θέση της εικόνας θα διαβάσει το περιεχόμενο του γνωρίσματος alt. Αν δεν υπάρχει το γνώρισμα αυτό, το άτομο με προβλήματα όρασης δεν μπορεί να έχει καμία αίσθηση για το περιεχόμενο της εικόνας και συνεπώς της ιστοσελίδας.

Επίσης, εάν η εικόνα αργεί να κατέβει, αφού οι εικόνες είναι πολύ πιο μεγάλα αρχεία από το έγγραφο HTML που είναι αρχείο κειμένου, το γνώρισμα alt βοηθάει τον χρήστη να αντιληφθεί το περιεχόμενο της εικόνας που δεν έχει κατέβει.

Ακόμη, το γνώρισμα alt είναι πολύ σημαντικό για τις μηχανές αναζήτησης. Οι μηχανές αναζήτησης κυρίως επεξεργάζονται το λεκτικό περιεχόμενο της ιστοσελίδας, οπότε το κείμενο του γνωρίσματος alt προσφέρει μια πιο πλήρη περιγραφή του περιεχομένου της σελίδας.

Για όλους αυτούς τους λόγους, το alt είναι υποχρεωτικό να το ορίζουμε και να του δίνουμε τιμή που αντιστοιχεί στην περιγραφή της εικόνας κάθε φορά που εισάγουμε μια εικόνα στο έγγραφο HTML.

3.2.2 Άλλα γνωρίσματα του <img>

Ας δούμε μερικά ακόμα γνωρίσματα του στοιχείου <img> που είναι χρήσιμα. Δύο γνωρίσματα που έχουν ιδιαίτερη χρήση είναι το width και το height. Το width και το height δίνουν είτε σε πίξελ είτε επί τοις εκατό ή σε άλλες μονάδες μέτρησης (θα μιλήσουμε για μονάδες μέτρησης στο κεφάλαιο 4.6.1) το μέγεθος που θα πάρει η εικόνα, το πλάτος δηλαδή και το ύψος της.

Εάν το μέγεθος της εικόνας που πρόκειται να τοποθετήσουμε σε μια θέση είναι διαφορετικό, τότε η εικόνα μεγεθύνεται ή μικραίνει, ανάλογα με την τιμή που θα πάρει το width ή το height. Θα πρέπει να προσέξουμε ότι δεν πρέπει να τα ορίζουμε και τα δύο. Πρέπει να ορίζουμε αυτό που έχει σημασία για τη διάταξη περιεχομένου της σελίδας μας, συνήθως το πλάτος, γιατί αν ορίσουμε και τα δύο και ο λόγος width προς height δεν είναι ο λόγος των διαστάσεων της εικόνας, τότε η εικόνα παραμορφώνεται.

Ένα άλλο γνώρισμα που έχει ενδιαφέρον είναι το γνώρισμα title. Αυτό το γνώρισμα, όπως και το alt, παίρνει ως τιμή ένα κείμενο που περιέχει πρόσθετα στοιχεία για την εικόνα. Θα πρέπει να προσέξουμε ότι δεν είναι σωστό να θέτουμε την ίδια ακριβώς πληροφορία στο γνώρισμα alt και title. Τα δύο αυτά γνωρίσματα παίζουν διαφορετικό ρόλο. Το μεν title προσθέτει πληροφορία συμπληρωματική για κάποιον που βλέπει μεν την εικόνα αλλά θέλει κάτι περισσότερο, μια επεξήγηση, και συνήθως εμφανίζεται σαν “tooltip” πάνω από την εικόνα όταν το ποντίκι υπερίπταται της εικόνας. Ενώ το alt, όπως είπαμε, είναι η εναλλακτική περιγραφή της εικόνας, άρα δίνει την ουσία της εικόνας για την περίπτωση που η εικόνα δεν φαίνεται ή δεν μπορεί να τη δει ο χρήστης.

3.2.3 Θέματα πνευματικών δικαιωμάτων

Κάτι που πρέπει να προσέξουμε ιδιαίτερα και αξίζει να το αναφέρουμε είναι ότι πρέπει να κάνουμε πολύ προσεκτική χρήση των εικόνων που βρίσκονται στο διαδίκτυο.

Η ενσωμάτωση εικόνας σε ένα έγγραφο HTML μέσω της ετικέτας <img>, όπως είδαμε, είναι μια απλή διαδικασία. Μπορούμε να χρησιμοποιήσουμε τη διεύθυνση URL μιας εικόνας που βρίσκεται σε οποιονδήποτε ιστότοπο. Κάτι τέτοιο όμως θα πρέπει να γίνεται αφού ακολουθηθούν κάποια απαραίτητα βήματα, που σχετίζονται με τα πνευματικά δικαιώματά της.

Πρέπει να βεβαιωθούμε ότι ο κάτοχος της εικόνας έχει παραχωρήσει ρητή γραπτή άδεια ή το έχει αναφέρει στην ιστοσελίδα του ότι το υλικό είναι ελεύθερο και με κάποια άδεια χρήσης Creative Commons που επιτρέπει τη χρήση. Εάν αυτό δεν υπάρχει ή δεν είναι σαφές, πρέπει να αποφεύγουμε τη χρήση του υλικού. Υπάρχει αρκετό υλικό ελεύθερα διαθέσιμο στο διαδίκτυο, όπως, για παράδειγμα, οι περισσότερες εικόνες στη Βικιπαίδεια, όμως ακόμη και σε αυτή την περίπτωση θα πρέπει να ακολουθούμε τις οδηγίες για αναφορά στον δημιουργό που επισυνάπτονται στο έργο.

3.2.4 Εικόνες με το στοιχείο <figure>

Στοιχεία που σχετίζονται με την παρουσίαση εικόνων είναι τα <figure> και <figcaption>. Θεωρούνται σημασιολογικά στοιχεία (περισσότερα τέτοια στοιχεία θα δούμε σε επόμενη ενότητα). Το στοιχείο <figure> επιτρέπει να ομαδοποιήσουμε μια εικόνα <img>, καθώς και τη λεζάντα της μέσω του στοιχείου <figcaption>. Ας δούμε ένα παράδειγμα:

```
<figure>
    <img src="images/myhouse.jpg"
        alt="Εικόνα ενός σπιτιού"
        width="300" >
    <figcaption>Το σπίτι μου στην οδό Φιλικών.  </figcaption>
</figure>
```

3.2.5 Πολλαπλές εικόνες με το στοιχείο <picture>

Το στοιχείο <picture> χρησιμοποιείται για να υποστηρίξει πολλαπλές εικόνες. Ας δούμε ένα παράδειγμα.

```
<picture>
    <source media="(max-width: 650px)" srcset="img-600.jpg">
    <source media="(max-width: 900px)" srcset="img-900.jpg">
    <img
        src="img3.jpg"
        alt="Το σπίτι δίπλα στη θάλασσα"
        title="Το σπίτι">
</picture>
```

Το στοιχείο <picture> περιέχει ένα πλήθος από στοιχεία <source>, το καθένα από τα οποία ορίζει μια διαφορετική εικόνα ανάλογα με τις συνθήκες που ορίζονται από το γνώρισμα media, το οποίο περιέχει ένα "media query" (καθορίζει τη συνθήκη με βάση την οποία η εικόνα εμφανίζεται). Στο παράδειγμα, αν το μέγιστο πλάτος της οθόνης είναι 650px ή μικρότερο, τότε επιλέγεται η εικόνα img-600.jpg. Αν είναι μέχρι 900px, επιλέγεται η εικόνα img-900.jpg, αν καμιά από τις δύο συνθήκες δεν ικανοποιείται, τότε υπάρχει η προκαθορισμένη εικόνα που περιέχεται στο στοιχείο <img> που είναι η εικόνα img3.jpg. Αυτή η εικόνα θα επιλεγεί, επίσης, αν ο φυλλομετρητής δεν υποστηρίζει το στοιχείο <picture>, οπότε θα αγνοηθούν τα στοιχεία <source>.

Να σημειωθεί ότι τα στοιχεία <source> χρησιμοποιούνται εκτός από το πλαίσιο του στοιχείου <picture>, που είδαμε εδώ, και στα στοιχεία <audio> και <video> που θα δούμε στη συνέχεια.

3.3 Βίντεο και ήχος, τα στοιχεία <video>, <audio>

Στην ενότητα αυτή θα δούμε δύο σημαντικά στοιχεία που αφορούν τα πολυμέσα που χρησιμοποιούνται ολοένα και περισσότερο σήμερα στο διαδίκτυο. Όπως βλέπουμε, ενώ τείνει να λιγοστέψει η χρήση του κειμένου στο

διαδίκτυο, η εικόνα και το βίντεο χρησιμοποιούνται όλο και περισσότερο.

Η HTML5 έχει βοηθήσει πάρα πολύ την άνοδο χρήσης αυτών των μέσων, υποκαθιστώντας διάφορες άλλες τεχνολογίες που δεν ήταν ενσωματωμένες σε προηγούμενες εκδόσεις της HTML και χρησιμοποιούνταν για τα πολυμέσα παλαιότερα. Σήμερα η HTML5 με τα στοιχεία audio και video μάς επιτρέπει πολύ εύκολα να εισαγάγουμε πολυμέσα στη σελίδα μας. Και εδώ, όπως με το στοιχείο img, χρησιμοποιούμε το γνώρισμα src=, με τιμή τη διεύθυνση που είναι το αρχείο ήχου ή βίντεο αντίστοιχα.

```
<video src="myvideo.mp4" controls>
    <p> δεν υποστηρίζεται βίντεο </p>
</video>
<audio src="myaudio.mp3" autoplay controls  loop>
```

Όπως φαίνεται στο παράδειγμα, τα στοιχεία αυτά υποστηρίζουν διάφορα γνωρίσματα, πολλά από τα οποία είναι κοινά και στα δύο:

- Το γνώρισμα autoplay σημαίνει ότι θα αρχίσει να εκτελείται το αρχείο βίντεο ή το αρχείο ήχου όταν έχει κατέβει επαρκές περιεχόμενο, ακόμα και πριν ολοκληρωθεί το φόρτωμα.
- Το γνώρισμα controls σημαίνει ότι θα εμφανιστούν τα γραφικά στοιχεία ελέγχου της ροής του πολυμεσικού περιεχομένου, όπως stop, start, play.
- Το γνώρισμα loop σημαίνει ότι, όταν τελειώσει να παίζει ο ήχος ή το βίντεο, θα ξαναρχίσουν από την αρχή.

Μέσα στο στοιχείο <audio> καθώς και στο <video> μπορούμε να εισαγάγουμε κάποιο κείμενο το οποίο θα το δείξει ο φυλλομετρητής στην περίπτωση που δεν υποστηρίζει το στοιχείο αυτό (αν, για παράδειγμα, δεν υποστηρίζει την HTML5). Σε αυτή την περίπτωση, συνήθως εισάγουμε έναν σύνδεσμο για να κατεβάσει το αρχείο, αφού ο ίδιος ο φυλλομετρητής δεν μπορεί να το παίξει.

3.3.1 Το στοιχείο <video>

Πριν από την HTML5, η ενσωμάτωση βίντεο σε ιστοσελίδες γινόταν με διάφορες τεχνολογίες, εξωτερικές της HTML, όπως το Flash, το Silverlight κ.λπ. Η HTML5 εισήγαγε το στοιχείο video. Σήμερα θα πρέπει να λάβουμε υπόψη μας ότι ένα μεγάλο ποσοστό των δεδομένων που διακινούνται στο διαδίκτυο είναι τύπου βίντεο, μια τάση που αναπτύσσεται περαιτέρω λόγω της αύξησης της ταχύτητας των δικτύων, της σύνδεσης συσκευών TV στο δίκτυο, της ευρείας χρήσης βίντεο σε μέσα κοινωνικής δικτύωσης, της ανάγκης επικοινωνίας και συνεργασίας μέσω τηλεδιασκέψεων κ.λπ., ενώ υπάρχει μεγαλύτερη ζήτηση να χρησιμοποιήσουμε το βίντεο σε παρουσίαση προϊόντων, στη διδασκαλία, σε παρουσίαση ερευνητικών αποτελεσμάτων κ.λπ.

Όπως είδαμε, το στοιχείο <video> της HTML5 μάς επιτρέπει να παρέχουμε πρόσβαση σε διαφορετικούς μορφότυπους βίντεο, ανάλογα με τις δυνατότητες που υποστηρίζει ο φυλλομετρητής του χρήστη.

Έστω το παρακάτω παράδειγμα:

```
<video controls>
    <source src="myvideo.mp4" type="video/mp4">
    <source src="myvideo.webm" type="video/webm">
    <p>O browser δεν υποστηρίζει video HTML5. Συνδεθείτε <a
href="myvideo.mp4 ">με το βίντεο</a></p>
</video>
```

Παρατηρούμε ότι μέσω του στοιχείου <source>, όπως είδαμε σε προηγούμενη ενότητα, που παρέχει εναλλακτικές επιλογές εικόνων, δίνεται η δυνατότητα στον φυλλομετρητή να εμφανίσει διαφορετικούς μορφότυπους βίντεο, για παράδειγμα, να διαθέτουμε εναλλακτικές εκδόσεις του ίδιου βίντεο για να υποστηρίζουμε περισσότερους φυλλομετρητές. Η πρώτη πηγή βίντεο είναι μορφότυπου mp4, που είναι ο πιο συνηθισμένος μορφότυπος στο διαδίκτυο, ενώ η δεύτερη πηγή είναι μορφότυπου webm.

Οι διαφορές μεταξύ webm και mp4 σχετίζονται με τον τρόπο που γίνεται το πακετάρισμα της ροής audio και ροής video μέσα σε μια ροή. Ο μορφότυπος webm έχει τον ήχο Ogg Vorbis με codec βίντεο VP8/VP9 και υποστηρίζεται από Firefox και Chrome, ενώ ο mp4, που είναι πιο διαδεδομένος, έχει ήχο AAC mp3 και codec video H.264. που είναι πρότυπο.

Τα γνωρίσματα που μπορούμε να ορίσουμε σε ένα στοιχείο <video> είναι παρόμοια με αυτά που είδαμε προηγουμένως στο στοιχείο <img> για τις εικόνες. Παραδείγματος χάρη, όπως είπαμε, το γνώρισμα controls μάς επιτρέπει να εμφανίσουμε τα χειριστήρια του βίντεο (play/stop κ.λπ.), το γνώρισμα width μάς παρέχει τη

δυνατότητα να ορίσουμε το πλάτος του παραθύρου του βίντεο, το height το ύψος κ.λπ. Αν ορίσουμε και τα δύο αυτά γνωρίσματα, θα αγνοηθούν, γιατί το βίντεο συντηρεί τον λόγο προβολής (πλάτος/ύψος) και δεν παραμορφώνεται όπως η εικόνα. Επίσης, υπάρχουν τα γνωρίσματα autoplay (άμεση έναρξη του βίντεο ενώ το αρχείο κατεβαίνει), loop (επανεκκίνηση του βίντεο μετά το τέλος του) και muted (ο ήχος σε σίγαση). Ακόμη, το γνώρισμα poster μάς επιτρέπει να ορίσουμε ένα αρχείο εικόνας που μπαίνει στη θέση του παραθύρου του βίντεο όταν το βίντεο δεν παίζει. Όπως είπαμε, ακόμη ως περιεχόμενο του στοιχείου <video> μπορούμε να βάλουμε, εκτός από διάφορα στοιχεία <source>, και κείμενο το οποίο θα φαίνεται αν ο φυλλομετρητής δεν δείχνει αυτό το video, το οποίο μπορεί να περιλαμβάνει υπερσύνδεσμο σε αρχείο βίντεο.

3.3.2 Το στοιχείο <audio>

Αντίστοιχα με το στοιχείο <video> είναι η λειτουργία του στοιχείου <audio>, που αφορά τον ήχο. Θα πρέπει να είμαστε ιδιαίτερα προσεκτικοί με τη χρήση του ήχου (μουσική, μηνύματα κ.λπ.), όπως βεβαίως και του βίντεο, χωρίς τη συγκατάθεση του χρήστη. Αν, για παράδειγμα, ο χρήστης συνδεθεί σε μια ιστοσελίδα σε έναν δημόσιο χώρο (μέσο μεταφοράς ή μια δημόσια βιβλιοθήκη) και απρόσμενα η σελίδα που επισκέφθηκε αρχίσει να παράγει ήχους, αυτό μπορεί να ενοχλήσει τον χρήστη και τους γύρω του. Και σε αυτή την περίπτωση μπορούμε να χρησιμοποιήσουμε το γνώρισμα controls, ενώ, όπως και στην προηγούμενη περίπτωση με το βίντεο και την εικόνα, μπορούμε να ορίσουμε πολλαπλές εναλλακτικές πηγές ήχου, καλύπτοντας την περίπτωση που ο φυλλομετρητής δεν υποστηρίζει κάποιο από αυτά.

Ένα παράδειγμα:

```
<audio controls>
    <source src="mysong.mp3" type="audio/mp3">
    <source src="mysong.ogg" type="audio/ogg">
    <p>
        Ο φυλλομετρητής δεν επιτρέπει ήχο HTML5. Ακολουθήστε a <a
href="mysong.mp3">σύνδεσμο στο αρχείο ήχου</a>
    </p>
</audio>
```

Ο τύπος mp3 είναι ο πιο συνηθισμένος, ενώ εναλλακτικά χρησιμοποιείται ο τύπος ogg. Και στο στοιχείο <audio> μπορούμε να εισαγάγουμε κάποιο κείμενο που εμφανίζεται εναλλακτικά, για την περίπτωση που ο φυλλομετρητής δεν υποστηρίζει το στοιχείο αυτό.

3.4 Ενσωμάτωση στοιχείων <iframe> και <object>

Σε αυτή την ενότητα θα δούμε κάποιες πολύ σημαντικές δυνατότητες που μας δίνει η HTML5, η οποία μάς επιτρέπει να ενσωματώσουμε άλλα αντικείμενα, όπως μια άλλη ιστοσελίδα ή αρχεία διαφορετικών τύπων στη σελίδα μας. Αυτές είναι δυνατότητες πολύ ισχυρές, τις οποίες όμως θα πρέπει να χρησιμοποιούμε με ιδιαίτερη προσοχή και μέτρο.

3.4.1 Ενσωμάτωση στοιχείων <iframe>

Η πρώτη δυνατότητα που μας δίνεται είναι η ενσωμάτωση στοιχείων <iframe> (inline frame). Το στοιχείο αυτό μας επιτρέπει να εισαγάγουμε ένα frame το οποίο ορίζουμε να λαμβάνει περιεχόμενο από έναν εξωτερικό πόρο και να ενσωματώνεται σε μια δική μας σελίδα.

Ας δούμε ένα παράδειγμα:

```
<body>
    <h1> Ο χάρτης του Πανεπιστημίου Πατρών</h1>
    <iframe id="inlineFrameExample"
        title="Το Πανεπιστήμιο Πατρών"
        width="500"
        height="300"
        frameborder="2"

src="https://www.openstreetmap.org/export/embed.html?bbox=21.7742%2C38.281
```

```
4%2C21.7987%2C38.2943&layer=mapnik">
    </iframe>
</body>
```

Το αποτέλεσμα του παραπάνω εγγράφου είναι το εξής:

Εικόνα 3.1 *Παράδειγμα εισαγωγής στοιχείου <iframe>: ο χάρτης.*

Εδώ παρατηρούμε ότι έχουμε ενσωματώσει ένα στοιχείο <iframe> το οποίο έχει τις δικές του διαστάσεις (width, height) και ως πηγή του περιεχομένου του ένα τμήμα του χάρτη openstreetmap (σημείωση: το απόσπασμα του χάρτη είναι εύκολο να ανακτηθεί, καθώς και ο κώδικας από την ιστοσελίδα openStreetMap, με την επιλογή «κοινοποίηση» ή "share" και ακολουθούμε τις οδηγίες για ανάκτηση κώδικα iframe για το περιεχόμενο του χάρτη που έχουμε επιλέξει).

Ας δούμε τα κύρια χαρακτηριστικά του στοιχείου <iframe>. Αυτό παρουσιάζει αρκετές ομοιότητες με άλλα στοιχεία που είδαμε προηγούμενα. Δέχεται ως γνώρισμα ένα source, μια διεύθυνση URL ενός πόρου από το οποίο θα αντλήσουμε το περιεχόμενο. Επίσης, πρέπει να ορίσουμε τις διαστάσεις του frame, δηλαδή του τμήματος της δικής μας ιστοσελίδας όπου θα τοποθετηθεί το περιεχόμενο από αυτό τον εξωτερικό πόρο. Μπορούμε επίσης να ορίσουμε αν θα υπάρχει πλαίσιο, με το γνώρισμα frameborder, αν θα επιτρέπεται ή όχι η πλήρης κάλυψη της οθόνης με το λογικό γνώρισμα allowfullscreen. Ακόμη ένα γνώρισμα με ιδιαίτερο ενδιαφέρον είναι το sandbox, που επιτρέπει αυξημένη ασφάλεια του περιβάλλοντος του φυλλομετρητή μας από τον εξωτερικό πόρο. Επίσης, μέσα στο στοιχείο <iframe> είναι καλό να εισαγάγουμε περιεχόμενο το οποίο θα δει ο χρήστης σε περίπτωση που ο φυλλομετρητής του δεν υποστηρίζει το στοιχείο iframe.

Για παράδειγμα, αν στον παραπάνω κώδικα προσθέσουμε το γνώρισμα sandbox, ο φυλλομετρητής Chrome δεν επιτρέπει την εμφάνιση του χάρτη και παράγει το εξής διαγνωστικό: "Blocked script execution in 'https://…' because the document's frame is sandboxed and the 'allow-scripts' permission is not set".

Θέλει πολλή προσοχή εδώ, βεβαίως, γιατί θα πρέπει να εξασφαλίσουμε, όπως γίνεται και με άλλα πολυμεσικά υλικά με βίντεο, ήχους και εικόνες, ότι έχουμε δικαίωμα να χρησιμοποιήσουμε το περιεχόμενο και να το προβάλουμε στη δική μας ιστοσελίδα. Θα πρέπει επίσης να προσέξουμε ότι πολλές ιστοσελίδες δεν επιτρέπουν να εισαγάγουμε σαν iframe το περιεχόμενό τους, ενώ κάποιες άλλες το επιτρέπουν.

Τέλος, θα πρέπει να λάβουμε υπόψη μας ότι κάθε iframe που ενσωματώνουμε στη σελίδα μας θα έχει τη δική του ιστορία πλοήγησης και document object model (DOM), επιβαρύνοντας τη μνήμη του φυλλομετρητή μας. Άρα, θα πρέπει να γίνεται με φειδώ η ενσωμάτωση περιεχομένου με τη μέθοδο αυτή.

3.4.2 Ενσωμάτωση στοιχείων <object>

Μια εναλλακτική επιλογή ενσωμάτωσης περιεχομένου εξωτερικών πόρων είναι η χρήση στοιχείων <object>. Είναι ένα στοιχείο γενικού σκοπού το οποίο μάς επιτρέπει ενσωμάτωση του εξωτερικού πόρου είτε ως εικόνας είτε ως πόρου που θα τον χειριστεί μια επέκταση (plugin ή extension).

Ας δούμε ένα παράδειγμα.

```
<object data="mypdf.pdf" type="application/pdf" width="800"
height="1200">
    <p>Μπορείτε να κατεβάσετε το αρχείο από
    <a href="mypdf.pdf">τον σύνδεσμο (800KB).</a>
</object>
```

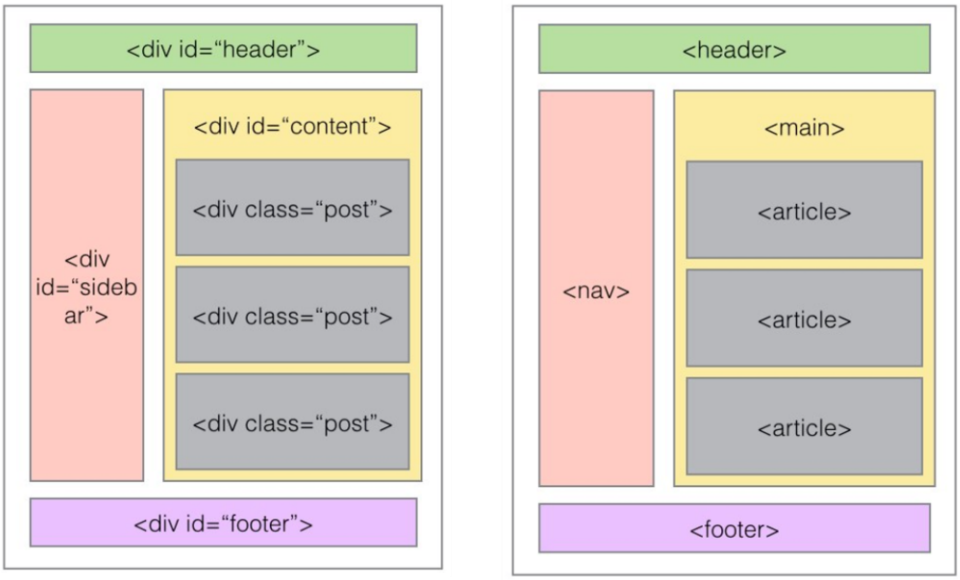
Στο παράδειγμα αυτό ενσωματώνουμε σε μια περιοχή 800x1200px της σελίδας μας ένα αρχείο pdf. Για τον σκοπό αυτό, ορίζουμε ένα στοιχείο <object>, στο οποίο περνάμε τον πόρο μέσω του γνωρίσματος data. Είναι απαραίτητο μέσω του γνωρίσματος type να ορίσουμε τον τύπο του πόρου, χρησιμοποιώντας κατηγοριοποίηση MIME. Αυτό ώστε ο φυλλομετρητής να γνωρίζει πώς θα χειριστεί τον πόρο, εφόσον υπάρχει αυτή η δυνατότητα.

Άλλα γνωρίσματα που μπορούμε να χρησιμοποιήσουμε στο στοιχείο αυτό είναι το πλάτος του πλαισίου (border) και οι διαστάσεις του width και height.

Άσκηση
Πηγαίνετε στο YouTube και ελέγξτε τον κώδικα που παράγεται αν επιλέξετε για κάποιο βίντεο την επιλογή «κοινοποίηση»/“share” και στη συνέχεια «Ενσωμάτωση βίντεο». Ποιο στοιχείο HTML παράγεται; Πώς μπορείτε να τροποποιήσετε τον κώδικα που σας προτείνει ώστε να μην ξεκινάει το βίντεο αυτόματα; Μελετήστε τις παραμέτρους του στοιχείου που προτείνεται.

3.5 Δομικά στοιχεία HTML

Σε αυτή την ενότητα θα δούμε κάποια ακόμα στοιχεία που έχουν εισαχθεί στην HTML5, τα οποία μας επιτρέπουν να δομήσουμε σημασιολογικά το περιεχόμενο μιας ιστοσελίδας.



Εικόνα 3.2 *(α) Οργάνωση περιεχομένου με μη σημασιολογικά στοιχεία, (β) Οργάνωση με χρήση σημασιολογικών στοιχείων.*

Έστω ότι το περιεχόμενο μιας σελίδας επιθυμούμε να το δομήσουμε όπως φαίνεται στην ***Εικόνα 3.2***. Δηλαδή, μια κεφαλίδα, ένα υποσέλιδο, μια πλαϊνή μπάρα που χρησιμεύει ως μενού πλοήγησης και κάποιο κυρίως περιεχόμενο το οποίο δομείται σε επιμέρους τμήματα. Αυτή μπορεί να είναι μια ιστοσελίδα με ειδήσεις ή ένα ηλεκτρονικό κατάστημα που μπορεί να δομηθεί με αυτό τον τρόπο.

Ο κλασικός τρόπος για να οργανώσουμε τους υποδοχείς περιεχομένου, σε προηγούμενες εκδόσεις της HTML, είναι να ορίσουμε στοιχεία , καθένα από τα οποία έχει μια ταυτότητα id ή ανήκει σε μια κλάση class, στην περίπτωση που το στοιχείο αυτό επαναλαμβάνεται. Αυτός ο τρόπος φαίνεται στο αριστερό σχήμα της εικόνας. Για παράδειγμα, η πλαϊνή μπάρα πλοήγησης ορίζεται ως , ενώ καθένα από τα επιμέρους άρθρα ορίζεται ως . Το στοιχείο , όμως, δεν φέρει ιδιαίτερη σημασιολογία, είναι απλά ένας υποδοχέας, ένα block element. Με αυτή τη μέθοδο, όλα τα στοιχεία της σελίδας είναι .

Αυτός παραμένει ένας τρόπος που πολλοί προγραμματιστές χρησιμοποιούν ακόμα και σήμερα.

Ας δούμε την καινούργια δυνατότητα που μας παρέχει η HTML5, η οποία εισάγει νέα στοιχεία που επιτρέπουν να προσδιορίζουμε τη σημασιολογία καθενός από αυτά τα δομικά στοιχεία.

Τέτοια στοιχεία είναι το στοιχείο <header> για την κεφαλίδα της σελίδας, <nav> για την μπάρα πλοήγησης (Navigation bar), η οποία μπορεί να είναι στο πάνω μέρος ή στο πλάι, ανάλογα με τη διάταξη της σελίδας, ένα στοιχείο <main>, το οποίο πρέπει να είναι ένα και μοναδικό, και μέσα σε αυτό στοιχεία <article> καθώς και <section>. Ακόμη, μπορεί να υπάρχει μια πλαϊνή μπάρα που περιγράφεται από το στοιχείο <aside> και συνήθως περιέχεται στο στοιχείο <main>. Τέλος, έχουμε ένα στοιχείο <footer> για το υποσέλιδο. Αυτά λοιπόν είναι τα κύρια δομικά στοιχεία που μας προσφέρονται, και ένα παράδειγμα χρήσης τους φαίνεται στα δεξιά στην ***Εικόνα 3.2***.

Μια συζήτηση που γίνεται αφορά το πώς δομούνται τα στοιχεία section και article. Για κάποιους σχεδιαστές ένα section περιέχει πολλά στοιχεία article. Αυτή η άποψη προτείνει ένα section ως θεματική ομαδοποίηση περιεχομένου μέσα στο οποίο περιέχονται αυτόνομα τμήματα πληροφορίας, που είναι τα article. Καθένα από αυτά τα article θα πρέπει να έχει μέσα του header <h1> <h2> κ.λπ., με οργάνωση τέτοια ώστε το κάθε article να έχει μόνο ένα header <h1>, μέσα στο οποίο να υπάρχουν πολλά <h2> και ούτω καθεξής. Αυτός ο τρόπος οργάνωσης έχει υιοθετηθεί στην παραπάνω εικόνα.

Για κάποιους άλλους σχεδιαστές υπάρχει η αντίστροφη δομή, ότι το article είναι το κυρίως στοιχείο και οι επιμέρους περιοχές του είναι τα section. Θα πρέπει όμως να λάβουμε υπόψη ότι για τα επιμέρους στοιχεία ενός article έχουμε τα άλλα δομικά στοιχεία, τα οποία υπήρχαν και παλαιότερα, τα <h1>, <h2> κ.λπ., που μας επιτρέπουν να δομήσουμε ιεραρχικά το περιεχόμενο ενός article.

Τα και , όπως είπαμε, είναι στοιχεία που υπάρχουν ήδη και χρησιμοποιούνται εκτεταμένα. Μάλιστα, μερικοί σχεδιαστές τα προτιμούν, έχουν συνηθίσει να τα χρησιμοποιούν. Όμως, η νέα πρόταση της HTML5 είναι να χρησιμοποιήσουμε τα και μόνο για την περίπτωση που δεν μας ταιριάζει να χρησιμοποιήσουμε κάποιο από τα άλλα δομικά στοιχεία που ήδη αναφέραμε. Υπενθυμίζεται ότι το είναι ένα στοιχείο inline που αφορά τμήμα του κειμένου, ενώ το είναι ένα στοιχείο block. Τα στοιχεία αυτά όμως δεν φέρουν σημασιολογία, αλλά πρέπει εμείς να τους δώσουμε σημασιολογία, συνήθως με ένα id. Υπάρχουν περιπτώσεις που ένα στοιχείο έχει ειδική σημασιολογία η οποία δεν καλύπτεται από τα δομικά στοιχεία που αναφέρθηκαν, τότε χρησιμοποιούνται τα ή . Ας δούμε ένα παράδειγμα.

Ας πούμε ότι έχουμε μια παράγραφο <p> μέσα στην οποία θέλουμε να εισαγάγουμε μια σημείωση του επιμελητή της έκδοσης. Η σημείωση εισάγεται μέσω ενός στοιχείου , στο οποίο επισυνάπτουμε την κλάση "editor-note".

```
<p> Η βασίλισσα μπήκε ακολουθούμενη από την πομπή της στο δωμάτιο <span
class="editor-note">[Σημείωση: Τα φώτα χαμηλώνουν καθώς εισέρχεται η
πομπή]</span>.</p>
```

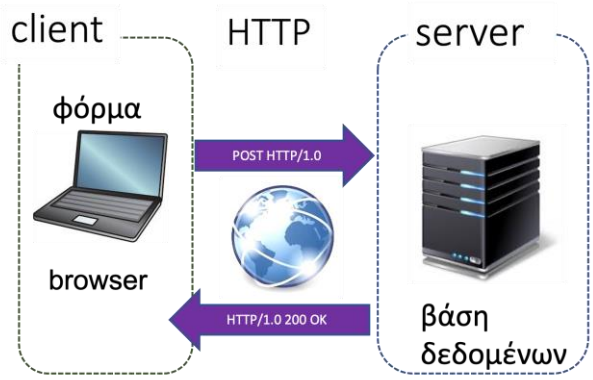
Στην κλάση αυτή θα επισυνάψουμε κάποιο στιλ ειδικό για αυτού του τύπου τις σημειώσεις.

3.6 Φόρμες: το στοιχείο <form>

Αυτή είναι η τελευταία ενότητα του κεφαλαίου, στην οποία θα συζητήσουμε ένα πολύ σημαντικό στοιχείο της HTML, το <form>. Είναι το στοιχείο που μας επιτρέπει να δημιουργήσουμε φόρμες μέσω των οποίων ο χρήστης εισάγει δεδομένα σε ένα σύστημα. Συμπληρώνουμε συνεχώς φόρμες στη ζωή μας, είτε διαδικτυακά, όταν συμπληρώνουμε όνομα χρήστη και συνθηματικό ή όταν ζητούνται τα προσωπικά μας στοιχεία για να κάνουμε εγγραφή σε μια καινούργια υπηρεσία, είτε συμπληρώνουμε έντυπες φόρμες όταν συναλλασσόμαστε με το δημόσιο ή με άλλους φορείς (συμπληρώνουμε ένα ερωτηματολόγιο, υπογράφουμε μια υπεύθυνη δήλωση κ.λπ.). Η σχεδίαση μιας καλής φόρμας διέπεται από κάποιες βασικές αρχές, τις οποίες θα συζητήσουμε στο πλαίσιο αυτής της ενότητας.

3.7 Η διαδικασία υποβολής φόρμας

Ας αρχίσουμε όμως με την αρχιτεκτονική μιας διαδικτυακής εφαρμογής που περιλαμβάνει μια φόρμα. Αυτή φαίνεται στην ***Εικόνα 3.3***.



Εικόνα 3.3 *Η υποβολή μιας φόρμας συνεπάγεται αίτημα HTTP προς τον εξυπηρετητή.*

Οι εφαρμογές του διαδικτύου, όπως έχει ήδη αναφερθεί, ακολουθούν το μοντέλο πελάτη-εξυπηρετητή. Ο πελάτης (ο φυλλομετρητής) είναι εκεί που παρουσιάζεται η φόρμα στον χρήστη. Όταν ο χρήστης συμπληρώσει τα στοιχεία της φόρμας, τότε πατάει ένα πλήκτρο **υποβολή φόρμας (submit)**. Όταν ο χρήστης εισάγει δεδομένα ή υποβάλλει τη φόρμα, τα στοιχεία ελέγχονται στον ίδιο τον φυλλομετρητή ως προς την εγκυρότητά τους.

Για παράδειγμα, αν υπάρχει ένα υποχρεωτικό στοιχείο της φόρμας που ζητάει την *ηλικία*, ελέγχεται αν συμπληρώθηκε και αν η ηλικία που έδωσε ο χρήστης είναι αποδεκτή. Αφού γίνουν οι έλεγχοι, εάν κάποιος από τους ελέγχους δεν ικανοποιηθεί, ένα μήνυμα στέλνεται στον χρήστη και ζητείται να συμπληρώσει και να υποβάλει ξανά τη φόρμα. Αυτοί οι έλεγχοι μπορεί να γίνουν από τα ίδια τα στοιχεία της HTML5, ενώ συχνά χρησιμοποιείται κώδικας JavaScript.

Αν υποθέσουμε τώρα ότι η φόρμα συμπληρώθηκε σωστά, τότε τα στοιχεία της στέλνονται μέσα από το πρωτόκολλο HTTP προς τον εξυπηρετητή είτε με τη μέθοδο GET είτε με τη μέθοδο POST. Ο εξυπηρετητής λαμβάνει τα στοιχεία, ελέγχει την εγκυρότητά τους και τα συσχετίζει με στοιχεία που διαθέτει ήδη. Ο εξυπηρετητής, για παράδειγμα, μπορεί να ελέγξει αν το όνομα που δίνει κάποιος ως όνομα χρήστη έχει ήδη χρησιμοποιηθεί από άλλους χρήστες. Εάν τα στοιχεία είναι έγκυρα, τότε ο εξυπηρετητής παράγει μια απόκριση η οποία επιστρέφει προς τον χρήστη. Η απόκριση αυτή μπορεί να είναι: *εντάξει, ο κωδικός που έδωσες ήταν σωστός, μπορείς να μπεις στην υπηρεσία μας*, π.χ. στην ιστοσελίδα του μέσου κοινωνικής δικτύωσης. Ή μπορεί να είναι μια ανάδραση: *λάθος, τα στοιχεία που έδωσες δεν είναι σωστά*. Επίσης, ο εξυπηρετητής μπορεί να αποθηκεύσει τα στοιχεία που έδωσε ο χρήστης σε μια βάση δεδομένων.

Η σύνταξη του στοιχείου <form> φαίνεται στη συνέχεια:

```
<form action="/my-handling-form-page" method="post">
    ... τα στοιχεία της φόρμας ...
</form>
```

Δύο γνωρίσματα που καθορίζουν τη συμπεριφορά του στοιχείου form είναι το action και το method.

- Το γνώρισμα action παίρνει ως τιμή τη διεύθυνση URL του πόρου του διαδικτύου στην οποία θα αποσταλεί η φόρμα για επεξεργασία όταν την υποβάλει ο χρήστης. Εάν δεν έχει δοθεί τιμή στο γνώρισμα action, αυτό σημαίνει ότι η φόρμα θα σταλεί στην ίδια διεύθυνση URL της ιστοσελίδας στην οποία υπάρχει η ίδια η φόρμα.
- Το γνώρισμα method περιγράφει τη μέθοδο της HTTP που θα χρησιμοποιηθεί για αποστολή της φόρμας στον εξυπηρετητή και αυτή είναι είτε post είτε get. Αν δεν δοθεί τιμή στο γνώρισμα method, τότε θεωρείται ότι η μέθοδος είναι η get.

Ένα από τα ερωτήματα που τίθενται είναι ποια από τις μεθόδους get/post να χρησιμοποιηθεί για υποβολή της φόρμας.

Η διαφορά είναι ότι η get περιλαμβάνει τα δεδομένα που εισήγαγε ο χρήστης της φόρμας στο URL που βρίσκεται στην κεφαλίδα του μηνύματος ως query string. Είδαμε στο πρώτο κεφάλαιο πώς συντάσσεται μια διεύθυνση URL στην οποία εισάγουμε τα δεδομένα της φόρμας. Για παράδειγμα, η διεύθυνση https://www.google.gr/search?q=python έχει προκύψει από αποστολή μιας φόρμας με ένα πεδίο με όνομα q στο οποίο ο χρήστης έδωσε την τιμή python. Το μέγεθος του URL είναι περιορισμένο (2.048 χαρακτήρες), άρα δεν μπορούμε να στείλουμε πολλά δεδομένα με αυτό τον τρόπο. Ένα θετικό όμως αυτής της μεθόδου είναι ότι μπορούμε να αποθηκεύσουμε το αποτέλεσμα της συμπλήρωσης της φόρμας ως διεύθυνση URL.

Η post, από την άλλη μεριά, είναι η πιο συνηθισμένη μέθοδος για αποστολή στον εξυπηρετητή των δεδομένων της φόρμας. Τα δεδομένα σε αυτή την περίπτωση στέλνονται στο σώμα του μηνύματος. Θα πρέπει να σημειωθεί ότι δεν υπάρχουν περιορισμοί στο μέγεθος δεδομένων που στέλνονται με τη μέθοδο post, όμως δεν αποθηκεύονται στο URL, όπως στην περίπτωση της μεθόδου get.

3.8 Περιεχόμενο στοιχείου <form>

Στην ενότητα αυτή θα συζητήσουμε τα στοιχεία που μπορεί να περιέχονται μέσα σε ένα στοιχείο <form>.

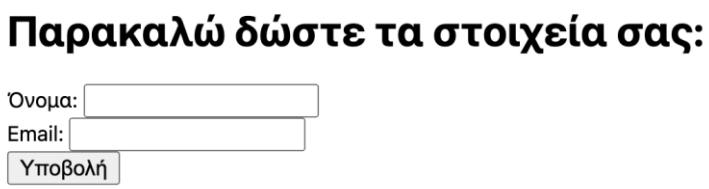
Όπως ξέρουμε από την εμπειρία μας, μια φόρμα μπορεί να περιέχει πεδία εισαγωγής δεδομένων, που αντιστοιχούν σε διάφορους τύπους στοιχείων <input>, καθώς και πεδία με δυνατότητα επιλογής από εναλλακτικές τιμές κ.λπ. Επίσης, η φόρμα εμπεριέχει επεξηγηματικά κείμενα τα οποία είναι απαραίτητα ως υπόδειξη στον χρήστη, τα στοιχεία <label>.

Το πιο συνηθισμένο στοιχείο <input> είναι ένα πεδίο κειμένου μέσα στο οποίο καλείται να εισαγάγει ο χρήστης δεδομένα, σύμφωνα με μια υπόδειξη. Εναλλακτικά, μπορεί η φόρμα να περιέχει ένα μεγαλύτερο πλαίσιο πολλών γραμμών, στο οποίο ο χρήστης μπορεί να εισαγάγει κείμενο, για παράδειγμα, να γράψει τα σχόλιά του. Αυτό είναι το στοιχείο <textarea> της HTML.

Επίσης, σε μια φόρμα θα δούμε στοιχεία επιλογής ναι/όχι (checkbox) ή αμοιβαία αποκλειόμενες μεταξύ τους επιλογές (radio buttons), στοιχεία επιλογής τιμής από εύρος τιμών (range). Ακόμη, η φόρμα μπορεί να περιέχει πιο σύνθετα στοιχεία, πεδία επιλογής ημερομηνίας, χρώματος, αρχείου κ.λπ. Τέλος, περιέχει πλήκτρα τα οποία έχουν σχέση με τη φόρμα, όπως το πλήκτρο *Υποβολή* που όταν πατηθεί η φόρμα υποβάλλεται στον εξυπηρετητή, ή το πλήκτρο *Εκκαθάριση* για επανεκκίνηση συμπλήρωσης της φόρμας.

3.9 Παράδειγμα φόρμας

Ακολουθεί το πρώτο παράδειγμα φόρμας που θα συζητήσουμε.



Εικόνα 3.4 *Παράδειγμα φόρμας που περιλαμβάνει δύο στοιχεία input και ένα πλήκτρο υποβολής.*

Η φόρμα αυτή περιέχει δύο πεδία εισαγωγής κειμένου, ένα για το όνομα και ένα για το email. Επίσης, περιέχει έναν τίτλο, υποδείξεις και ένα πλήκτρο υποβολής.

Ο κώδικας που παράγει αυτή τη φόρμα είναι ο εξής:

```
<form>
    <h1>Παρακαλώ δώστε τα στοιχεία σας:</h1>
    <div>
        <label for="name">Όνομα:</label>
        <input type="text" id="name" name="user_name">
    </div>
    <div>
        <label for="mail">Email:</label>
        <input type="email" id="mail" name="user_mail">
    </div>
    <div>
        <input type="submit" value="Υποβολή">
    </div>
</form>
```

Το στοιχείο <form> περιέχει ένα στοιχείο <h1> και τρία στοιχεία , το καθένα από τα οποία περιέχει ένα

στοιχείο <input>.

Αν εξετάσουμε το πρώτο στοιχείο , περιέχει ένα <label> που αφορά την υπόδειξη προς τον χρήστη και ένα στοιχείο <input type="text"> που είναι το πεδίο στο οποίο εισάγει ο χρήστης τα δεδομένα. Τα δύο αυτά στοιχεία συνδέονται μεταξύ τους μέσω του γνωρίσματος for του στοιχείου <label>, το οποίο έχει τιμή την ταυτότητα id του στοιχείου <input>.

Όπως θα δούμε, υπάρχουν πολλοί διαφορετικοί τύποι στοιχείων <input>, το πιο συνηθισμένο είναι το στοιχείο τύπου text. Το στοιχείο <input> έχει ως γνώρισμα name="user_name". Το γνώρισμα αυτό είναι το όνομα της μεταβλητής στην οποία εκχωρείται η τιμή που θα δώσει ο χρήστης στο αντίστοιχο πεδίο. Το γνώρισμα name είναι υποχρεωτικό αν θέλουμε να έχουμε πρόσβαση στα δεδομένα του στοιχείου, η μεταβλητή αυτή χρησιμοποιείται από την JavaScript και στέλνεται ως ζεύγος μεταβλητή=τιμή στον εξυπηρετητή.

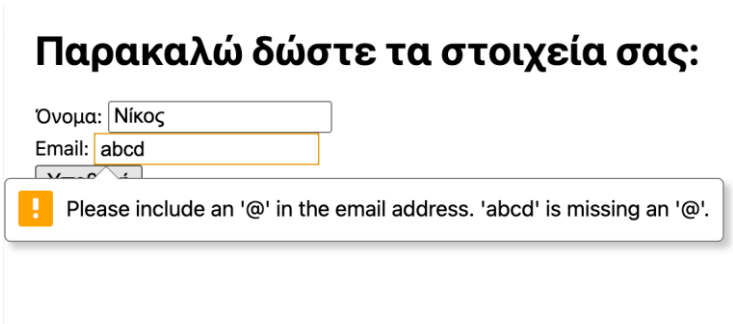
Το ίδιο θα παρατηρήσουμε να γίνεται και στο επόμενο πεδίο <input type="email">, στο οποίο δίνεται το όνομα name="user_mail".

Όταν υποβάλουμε τη φόρμα, αν υποθέσουμε ότι ο χρήστης έχει συμπληρώσει στο πρώτο πεδίο το όνομα “Nikos” και στο δεύτερο πεδίο “nikos@mail.com”, θα παρατηρήσουμε ότι η φόρμα θα υποβληθεί (method=get) στο URL:

...?user_name=Nikos&user_mail=nikos%40mail.com

Παρατηρούμε ότι τα ζεύγη τιμών που στέλνονται στον εξυπηρετητή μέσω του URL απαρτίζονται από το όνομα του κάθε πεδίου και την τιμή που εισήγαγε ο χρήστης.

Ένα ερώτημα που τίθεται είναι σε τι διαφέρει το στοιχείο <input type="email"> από το πρώτο στοιχείο <input type="text">, αφού και τα δύο έχουν ακριβώς την ίδια εμφάνιση και τα δύο επιτρέπουν την εισαγωγή κειμένου στον χρήστη. Η διαφορά φαίνεται αν βάλουμε ένα κείμενο, για παράδειγμα “abcd”, στο δεύτερο πεδίο και προσπαθήσουμε να υποβάλουμε τη φόρμα, τότε θα πάρουμε ένα μήνυμα σαν αυτό που φαίνεται στην εικόνα:



Εικόνα 3.5 *Μήνυμα σφάλματος που παράγεται από το στοιχείο input τύπου email.*

Αυτό το μήνυμα προκύπτει από τον έλεγχο που γίνεται κατά την υποβολή της φόρμας στο πεδίο τύπου “email”, από τον οποίο προκύπτει ότι η συμβολοσειρά που έδωσε ο χρήστης δεν είναι αποδεκτή διεύθυνση email αφού δεν περιέχει το σύμβολο “@”.

Το τρίτο στοιχείο input της φόρμας είναι τύπου “submit”. Το στοιχείο αυτό έχει τη μορφή ενός πλήκτρου και έχει την ειδική χρήση ότι, όταν πατηθεί από τον χρήστη, γίνονται έλεγχοι και αν δεν προκύψει κάποιο πρόβλημα όπως αυτό που συζητήσαμε παραπάνω, η φόρμα υποβάλλεται στο URL που ορίζει το γνώρισμα action του στοιχείου <form>, με τη μέθοδο που ορίζει το γνώρισμα method. Αν, για παράδειγμα, έχουμε:

<form action="/form-handler.html" method="post">

τότε η φόρμα θα υποβληθεί στο σχετικό URL /form-handler.html με τη μέθοδο POST.

3.10 Τύποι στοιχείων μιας φόρμας

Στην ενότητα αυτή θα κάνουμε μια σύντομη επισκόπηση των κυριότερων τύπων στοιχείων <input>, αλλά και άλλων στοιχείων. Στο παράδειγμα της προηγούμενης ενότητας είδαμε στοιχεία <input> τύπου text, email και submit. Παρακάτω θα έχουμε μια πιο πλήρη εικόνα.

3.10.1 Πεδία εισαγωγής κειμένου από τον χρήστη

- **<input type="text">**: πεδίο εισαγωγής κειμένου,
- **<input type="email">**: εισαγωγή email,
- **<input type="number">**: εισαγωγή αριθμητικής τιμής,
- **<input type="password">**: εισαγωγή κωδικού (****), οι χαρακτήρες δεν φαίνονται,
- **<input type="search">**: πεδίο αναζήτησης,
- **<input type="url">**: εισαγωγή διεύθυνσης URL,
- **<input type="tel">**: εισαγωγή αριθμού τηλεφώνου,
- **<textarea>**: ορίζει πεδίο εισαγωγής κειμένου πολλών γραμμών.

3.10.2 Στοιχεία επιλογής από λίστα ή γραφική διεπαφή

- **<select>** : ορίζει μια λίστα από drop-down επιλογές οι οποίες περιέχονται σε στοιχεία <option> ή σε ομάδα επιλογών <optgroup>,
- **<datalist>**: ορίζει μια προκαθορισμένη λίστα από στοιχεία για ένα στοιχείο <input>,
- **<input type="checkbox">**: επιλογή ναι/όχι,
- **<input type="radio">**: επιλογή μιας από εναλλακτικές περιπτώσεις,
- **<input type="range">**: επιλογή με κύλιση σε ράβδο τιμών,
- **<input type="date">**: εισαγωγή/επιλογή ημερομηνίας από ημερολόγιο,
- **<input type="datetime-local">**: εισαγωγή/επιλογή τοπικής ημερομηνίας από ημερολόγιο,
- **<input type="month">**: εισαγωγή/επιλογή μήνα από ημερολόγιο,
- **<input type="time">**: εισαγωγή/επιλογή χρονικής στιγμής,
- **<input type="week">**: εισαγωγή/επιλογή εβδομάδας,
- **<input type="color">**: επιλογή χρώματος από παλέτα,
- **<input type="file">**: επιλογή αρχείου από το σύστημα αρχείων.

3.10.3 Πλήκτρα

- **<input type="button">**: γενικού σκοπού πλήκτρο,
- **<input type="image">**: εικόνα ως πλήκτρο υποβολής,
- **<input type="reset">**: πλήκτρο επαναφοράς της φόρμας στην αρχική κατάσταση,
- **<input type="submit">**: πλήκτρο υποβολής της φόρμας,
- **<button>**: ορίζει ένα πλήκτρο εναλλακτικά του στοιχείου <input type="button">.

Για να συμπληρωθεί η εικόνα των στοιχείων που μπορεί να βρούμε σε μια φόρμα, εκτός των παραπάνω μπορεί να υπάρχουν και τα εξής στοιχεία:

3.10.4 Άλλα στοιχεία

- **<input type="hidden">**: κρυφό πεδίο για αποθήκευση και υποβολή δεδομένων,
- **<label>**: επεξηγηματικό κείμενο για κάποιο στοιχείο,
- **<fieldset>** και **<legend>**: προσδιορίζουν μια ομάδα στοιχείων,
- **<output>**: στοιχείο που περιέχει το αποτέλεσμα υπολογισμού.

Όπως βλέπουμε, η HTML5 διαθέτει ένα πλούσιο σύνολο από στοιχεία φόρμας, πολλά από τα οποία (όπως, για παράδειγμα, το <input type="date">) βοηθούν τον χρήστη στην εισαγωγή δεδομένων, ενώ άλλα (όπως το <input type="email">) κάνουν έλεγχο της εισόδου του χρήστη σύμφωνα με κάποιους βασικούς κανόνες.

Στην επόμενη ενότητα θα δούμε κάποια από τα γνωρίσματα που μπορεί να πάρουν τα στοιχεία αυτά.

3.10.5 Κύρια γνωρίσματα των στοιχείων φόρμας

Ήδη έχουμε δει κάποια βασικά **γνωρίσματα** των στοιχείων της φόρμας, όπως το name που ορίζει τη μεταβλητή στην οποία θα εκχωρηθεί η τιμή του χρήστη. Επίσης, είδαμε το γνώρισμα value στο πλήκτρο υποβολής. Στη συνέχεια κάνουμε μια σύντομη αναφορά στα κυριότερα γνωρίσματα.

- **name**: η μεταβλητή που παίρνει την τιμή του στοιχείου στην JavaScript ή στον εξυπηρετητή,
- **value**: η αρχική τιμή του στοιχείου ή, για τα πλήκτρα, το κείμενό τους,
- **autocomplete**, “on”/“off”: επιτρέπει/απαγορεύει στον φυλλομετρητή να βοηθήσει με τη συμπλήρωση

της εισόδου του χρήστη,

- **autofocus**: υποδηλώνει το στοιχείο που αποκτάει την εστίαση όταν φορτωθεί η φόρμα,
- **max, min**: μέγιστη και ελάχιστη επιτρεπόμενη τιμή αριθμητικού στοιχείου,
- **maxlength, minlength**: μέγιστος και ελάχιστος αριθμός χαρακτήρων που επιτρέπονται στα δεδομένα που εισάγονται σε στοιχείο <input>,
- **required**: υποχρεωτικό πεδίο που πρέπει να συμπληρωθεί πριν υποβληθεί η φόρμα,
- **placeholder**: σύντομη περιγραφή της τιμής που αναμένεται να εισαγάγει ο χρήστης, εντός του πεδίου.,
- **size**: ορατό μήκος σε αριθμό χαρακτήρων του στοιχείου <input>,
- **disabled**: ανενεργό στοιχείο, δεν αποστέλλεται όταν γίνεται υποβολή της φόρμας,
- **readonly**: μη τροποποιήσιμο πεδίο, πεδίο μόνο για ανάγνωση.

Άσκηση 1

Το γνώρισμα value το είδαμε ήδη ως γνώρισμα του πλήκτρου υποβολής, όπου στην περίπτωση αυτή ορίζει το κείμενο που εμφανίζεται στο πλήκτρο. Όμως, θα πρέπει να προσέξουμε ότι η σημασιολογία του γνωρίσματος είναι αρκετά διαφορετική για στοιχεία input, όπου το γνώρισμα προσδιορίζει την αρχική τιμή του πεδίου όταν φορτωθεί η φόρμα.

Ποια η εμφάνιση της φόρμας όταν φορτωθεί και τι θα γίνει αν ο χρήστης εισαγάγει την τιμή 15;

```
<label for="age">Ηλικία:</label>
<input type="number" id="age" name="age" min=18, max=99, value=20><br>
```

Απάντηση

Στην περίπτωση αυτή, όταν φορτωθεί η φόρμα στο πεδίο της ηλικίας, το πεδίο θα έχει την τιμή 20. Τα γνωρίσματα min και max ορίζουν την ελάχιστη και μέγιστη επιτρεπόμενη τιμή, συνεπώς, αν ο χρήστης δώσει την τιμή 15, η τιμή δεν θα επιτραπεί και θα πάρει ένα μήνυμα λάθους.

Άσκηση 2

Το γνώρισμα placeholder ορίζει την υπόδειξη προς τον χρήστη που θα εμφανιστεί στο πεδίο. Έστω το παρακάτω στοιχείο της φόρμας:

```
<label for="fname">Όνομα:</label><br>
<input type="text" size="10" id="fname" name="fname" placeholder='γράψτε
το όνομά σας'><br>
```

Περιγράψτε τι θα δει ο χρήστης όταν φορτωθεί η φόρμα.

Απάντηση

Ο χρήστης θα δει το πεδίο Όνομα. Το κείμενο που θα εμφανιστεί εντός του πεδίου είναι αυτό που ορίζει το γνώρισμα placeholder, συνεπώς το κείμενο-υπόδειξη: *γράψτε το όνομά σας*. Όμως, έχει οριστεί ότι το μήκος του πεδίου είναι 10 χαρακτήρες (γνώρισμα size). Συνεπώς, ο χρήστης θα δει μόνο τους 10 πρώτους χαρακτήρες της υπόδειξης: *γράψτε το ό*.

3.10.6 Στοιχεία επιλογής

Ως τώρα έχουμε δει στοιχεία εισαγωγής δεδομένων από τον χρήστη, όπως τα στοιχεία τύπου text, password, email, number κ.λπ.

Όμως, όταν η τιμή που περιμένουμε από τον χρήστη ανήκει σε ένα κλειστό σύνολο, είναι προτιμότερο να ζητήσουμε από τον χρήστη να επιλέξει μια από τις τιμές, αντί να την εισαγάγει.

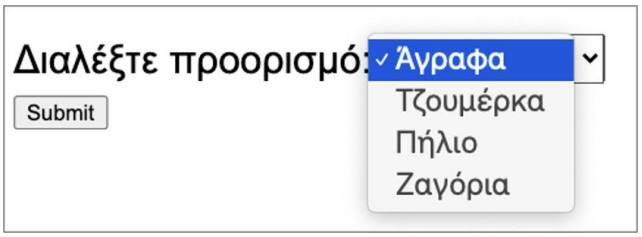
Ας δούμε τους διαφορετικούς τρόπους που έχουμε για να δώσουμε αυτή τη δυνατότητα στον χρήστη.

Λίστα επιλογών select/option

Ο πιο κλασικός τρόπος επιλογής είναι από αναδυόμενη λίστα επιλογών. Ας υποθέσουμε ότι ένας ορειβατικός σύλλογος ζητάει από τα μέλη του να επιλέξουν τον προορισμό της επόμενης εξόρμησης ανάμεσα σε 4

υποψήφιους προορισμούς.

Η εμφάνιση της ιστοσελίδας είναι:



Εικόνα 3.6 *Παράδειγμα φόρμας με επιλογή απάντησης από λίστα στοιχείων select/option.*

Ακολουθεί ο σχετικός κώδικας:

```
<form action="/action_page.html">
    <label for="places"> Διαλέξτε προορισμό:</label>
    <select id="places" name="place">
        <option value="agrafa" selected> Άγραφα</option>
        <option value="tzoumerka"> Τζουμέρκα</option>
        <option value="pilio"> Πήλιο</option>
        <option value="zagoria"> Ζαγόρια</option>
    </select><br>
    <input type="submit">
</form>
```

Παρατηρούμε ότι το στοιχείο που διαμορφώνει το πεδίο επιλογής είναι το στοιχείο <select>. Το όνομα του πεδίου είναι name=place.

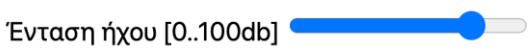
Εντός του στοιχείου <select> ορίζουμε τις επιλογές ως στοιχεία <option>. Καθένα από αυτά τα στοιχεία έχει ως περιεχόμενο το κείμενο που θα δει ο χρήστης. Το γνώρισμα value της αντίστοιχης επιλογής είναι η τιμή που θα πάρει το στοιχείο αν την επιλέξει ο χρήστης. Έτσι, αν ο χρήστης επιλέξει το *Πήλιο*, η μεταβλητή θα λάβει την τιμή place = pilio.

Επίσης, θα πρέπει να προσέξουμε ότι η επιλογή *Άγραφα* είναι προεπιλεγμένη όταν φορτώνεται η φόρμα και, αν ο χρήστης δεν κάνει καμιά ενέργεια επιλογής κατά την υποβολή της φόρμας, θα είναι η τιμή που θα επιστραφεί. Συνήθως θέτουμε ως προεπιλεγμένη τιμή εκείνη που είναι πιο πιθανόν να επιλέξουν οι περισσότεροι χρήστες.

Κύλιση σε πεδίο τιμών range

Ένας εναλλακτικός τρόπος να επιλέξει ο χρήστης μια τιμή είναι μέσω μιας ράβδου κύλισης. Το στοιχείο φόρμας που επιτρέπει κάτι τέτοιο είναι το <input type="range">. Ας δούμε ένα παράδειγμα.

Έστω ότι ζητάμε από τον χρήστη να επιλέξει την τιμή της έντασης του ήχου μεταξύ 0 και 100 db με κύλιση επί της ράβδου.



Εικόνα 3.7 *Παράδειγμα στοιχείου κύλισης από πεδίο τιμών.*

Ο παρακάτω κώδικας υλοποιεί αυτό το στοιχείο.

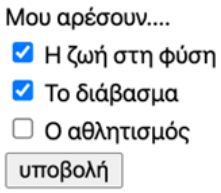
```
<form>
    <label for="volume">Ένταση ήχου [0..100db]</label>
    <input type="range" id="volume" name="vol" min="0" max="100"
value="80">
</form>
```

Θα πρέπει να σημειωθεί ότι η αρχική τιμή και αντίστοιχα η θέση της ράβδου κύλισης είναι 80 (γνώρισμα value).

Περισσότερες από μια επιλογές checkbox

Υπάρχουν περιπτώσεις που θα επιθυμούσαμε ο χρήστης να μπορεί να επιλέξει περισσότερες από μια επιλογές. Αυτό μπορεί να γίνει με χρήση πολλαπλών στοιχείων <input type="checkbox">, τα οποία επιτρέπουν το καθένα να επιλεγεί ή όχι.

Ας δούμε ένα παράδειγμα. Έστω μια ιστοσελίδα που ζητάει από τον χρήστη τα ενδιαφέροντά του και του επιτρέπει να επιλέξει κάποια από τα παρακάτω τρία ενδιαφέροντα:



Εικόνα 3.8 *Παράδειγμα φόρμας που περιλαμβάνει στοιχεία checkbox.*

Ο χρήστης μπορεί να επιλέξει κανένα, ένα, δύο ή τρία από τα ενδιαφέροντα αυτά. Ο παρακάτω κώδικας υλοποιεί αυτή τη φόρμα:

```
<form>
<div id ='likes'>Μού αρέσουν.... </div>
    <input type="checkbox" id="item1" name="item1" value="nature">
    <label for="item1">Η ζωή στη φύση </label><br>
    <input type="checkbox" id="item2" name="item2" value="reading">
    <label for="item2"> Το διάβασμα </label><br>
    <input type="checkbox" id="item3" name="item3" value="sports">
    <label for="item3">Ο αθλητισμός </label><br>
    <input type="submit" value='υποβολή'>
</form>
```

Το καθένα από τα στοιχεία τύπου checkbox έχει γνωρίσματα όνομα (name) και τιμή (value). Αν ο χρήστης επιλέξει το συγκεκριμένο στοιχείο, τότε η μεταβλητή που αντιστοιχεί στο όνομά του λαμβάνει την τιμή που ορίζει το γνώρισμα value.

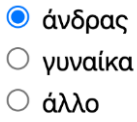
Για παράδειγμα, στην παραπάνω εικόνα έχουν επιλεγεί τα δύο πρώτα ενδιαφέροντα. Όταν υποβάλουμε τη φόρμα με αυτή την επιλογή, το URL θα περιέχει το εξής query string: ?item1=nature&item2=reading.

Αμοιβαία αποκλειόμενες επιλογές radio

Μια ακόμη περίπτωση στοιχείου φόρμας είναι το radio, που περιλαμβάνει 2 ή περισσότερες αμοιβαία αποκλειόμενες επιλογές. Το όνομα του στοιχείου αυτού προέρχεται από τα παλιά ραδιόφωνα που είχαν πλήκτρα επιλογής μπάντας συχνότητας, όπως στην ***Εικόνα 3.9***.

Εικόνα 3.9 *Παλιό ραδιόφωνο με πλήκτρα επιλογής μπάντας συχνοτήτων, από αυτά προέρχεται ο όρος radio buttons.*

Ας δούμε ένα παράδειγμα. Έστω μια φόρμα που ζητάει από τον χρήστη να δηλώσει το φύλο του και επιτρέπει αυτές τις τρεις επιλογές.



Εικόνα 3.10 *Παράδειγμα αμοιβαία αποκλειόμενων επιλογών, στοιχεία radio.*

Θα χρησιμοποιήσουμε τα στοιχεία τύπου radio τα οποία θα έχουν κοινό όνομα της μεταβλητής (γνώρισμα name). Μια ομάδα τέτοιων γνωρισμάτων η οποία τυπικά θα είναι κοντά το ένα με το άλλο θα επιτρέπει στον χρήστη να επιλέξει ένα από αυτά.

Ο κώδικας που υλοποιεί αυτή τη φόρμα είναι:

```
<form>
    <input type="radio" name="gender" value="male" checked> άνδρας<br>
    <input type="radio" name="gender" value="female"> γυναίκα<br>
    <input type="radio" name="gender" value=”other"> άλλο<br>
    <br>
</form>
```

Όπως βλέπουμε σε αυτό το παράδειγμα, έχει προεπιλεγεί (λογικό γνώρισμα checked) το πρώτο από τα στοιχεία.

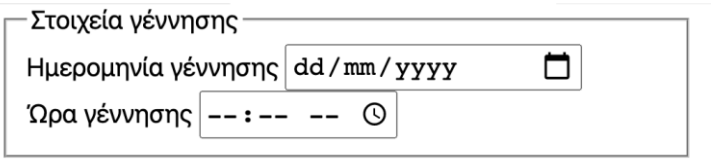
Τα στοιχεία συνδέονται μέσω του γνωρίσματος name="gender", το οποίο είναι κοινό και για τα τρία στοιχεία τύπου radio.

3.10.7 Επιλογή μέσω γραφικής διεπαφής

Υπάρχει μια ομάδα από στοιχεία φόρμας τα οποία εισήχθησαν στην HTML5 και βοηθάνε τον χρήστη να εισαγάγει τιμή μέσω γραφικού παράθυρου, χρησιμοποιώντας το εγγενές γραφικό σύστημα του υπολογιστή – επιλογή στοιχείων όπως ημερομηνίας από ένα ημερολόγιο, χρώματος από μια παλέτα χρωμάτων, αρχείο από μια διεπαφή του συστήματος αρχείων κ.λπ. Τα στοιχεία αυτά είναι εξαιρετικά χρήσιμα και διευκολύνουν τον χρήστη να εισαγάγει τα αντίστοιχα δεδομένα, όπως μια ημερομηνία, με σωστή μορφοποίηση, χωρίς σφάλματα πληκτρολόγησης. Το πρόβλημα με τα στοιχεία αυτά είναι ότι δεν υποστηρίζονται ακόμη από όλους τους φυλλομετρητές. Θα πρέπει να ελέγξουμε αν ο φυλλομετρητής υποστηρίζει το στοιχείο με κατάλληλο κώδικα JavaScript και να δώσουμε εναλλακτικές επιλογές, π.χ. να δημιουργήσουμε ένα στοιχείο select/option με τους μήνες του χρόνου ή τις ημερομηνίες στην περίπτωση του στοιχείου τύπου date.

Ας δούμε ένα παράδειγμα μέσω του οποίου θα εισαγάγουμε και ένα άλλο σημαντικό στοιχείο, που είναι το <fieldset>/<legend> και χρησιμεύει για την ομαδοποίηση των στοιχείων. Το παράδειγμα αφορά την εισαγωγή της ημερομηνίας και ώρας γέννησης ενός βρέφους.

Η ιστοσελίδα έχει την εξής εμφάνιση:



Εικόνα 3.11 *Παράδειγμα ομαδοποίησης στοιχείων φόρμας με το <fieldset>.*

Ας δούμε τον κώδικα της αντίστοιχης φόρμας:

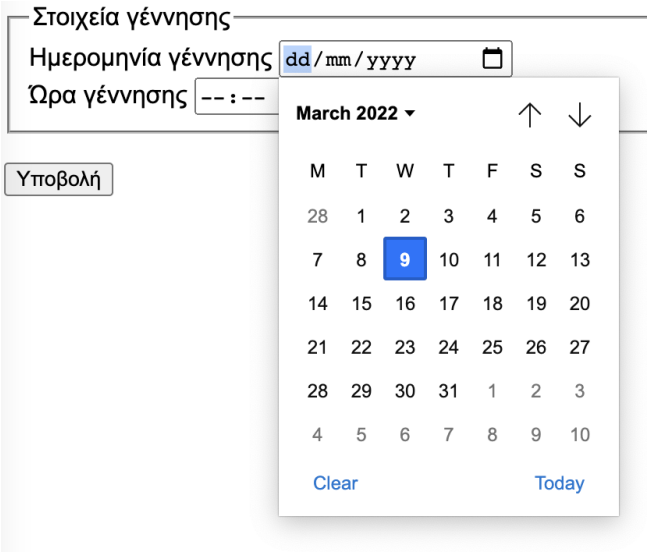
```
<fieldset>
    <legend>Στοιχεία γέννησης</legend>
    <label for="date">Ημερομηνία γέννησης</label>
    <input type="date" id="date" name="date-birth"><br>
    <label for="time">Ώρα γέννησης</label>
    <input type="time" id="time" name="time-birth">
</fieldset>
```

Το πρώτο στοιχείο που χρησιμοποιούμε σε αυτό το παράδειγμα είναι το στοιχείο <fieldset>, το οποίο περικλείει μια ομάδα από συγγενή στοιχεία. Επίσης, το στοιχείο αυτό συνοδεύεται από το στοιχείο <legend>, το οποίο μας επιτρέπει να δώσουμε μια επικεφαλίδα στην ομάδα. Ένα σχετικό σχόλιο είναι ότι η ομαδοποίηση των στοιχείων της φόρμας βοηθάει τον χρήστη να κατανοήσει καλύτερα τι του ζητάμε και είναι πολύ καλή πρακτική να χρησιμοποιήσουμε το στοιχείο ομαδοποίησης.

Παρατηρούμε ότι η φόρμα περιέχει δύο στοιχεία <input> τύπου date και time.

Παρατηρούμε ότι για τα στοιχεία αυτά υπάρχει η δυνατότητα είτε να εισαγάγει ο χρήστης μια ημερομηνία ή ώρα, πληκτρολογώντας την, είτε να επιλέξει το σχετικό εικονίδιο και να αλληλεπιδράσει από το σχετικό γραφικό στοιχείο του φυλλομετρητή.

Στην ***Εικόνα 3.12*** φαίνεται η αλληλεπίδραση με το πρώτο στοιχείο στον φυλλομετρητή Chrome. Η εμφάνιση του στοιχείου επιλογής ημερομηνίας μπορεί να διαφέρει σε διάφορους φυλλομετρητές.



Εικόνα 3.12 *Παράδειγμα αλληλεπίδρασης με γραφικό στοιχείο για την επιλογή ημερομηνίας (Chrome).*

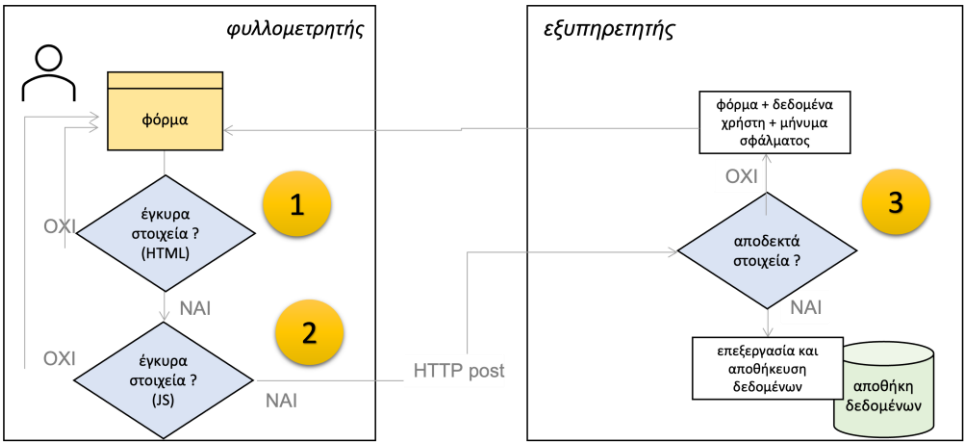
Κάτι που θα πρέπει να προσέξουμε είναι ότι, ανεξάρτητα από την εμφάνιση του στοιχείου στη διεπαφή χρήστη, η τιμή που θα αποθηκευτεί είναι πάντα στη μορφή yyyy-mm-dd. Έτσι, για παράδειγμα, αν ο χρήστης εισαγάγει 09/03/2022, θα αποθηκευτεί η τιμή date-birth=2022-03-09.

Επίσης, θα πρέπει να σημειωθεί ότι φυλλομετρητές που δεν υποστηρίζουν αυτό το στοιχείο το αντιλαμβάνονται ως απλό στοιχείο <input type="text">. Βεβαίως, σε τέτοια περίπτωση δεν γίνεται κανένας έλεγχος για τη συμβολοσειρά που εισάγει ο χρήστης, αν είναι αποδεκτή ημερομηνία και σύμφωνη με την προδιαγραφή μορφοποίησής της.

Αντίστοιχη είναι η συμπεριφορά του δεύτερου στοιχείου <input>. Να σημειωθεί και εδώ ότι η ώρα αποθηκεύεται ως συμβολοσειρά σε 24ωρο ρολόι, ανεξάρτητα από τη διεπαφή χρήστη. Για παράδειγμα, αν ο χρήστης δώσει την τιμή 08:36 PM, αυτή θα αποθηκευτεί ως συμβολοσειρά 20:36, οπότε το σχετικό query string για το στοιχείο αυτό θα είναι time-birth=20%3A36.

3.10.8 Έλεγχος εγκυρότητας της φόρμας

Ένα ιδιαίτερα σημαντικό θέμα είναι πώς θα γίνει ο έλεγχος εγκυρότητας μιας φόρμας. Αν ζητήσουμε από τον χρήστη την ηλικία του και απαντήσει 999, είναι μεν συντακτικά ορθό (number), όμως εννοιολογικά δεν έχει νόημα. Υπάρχουν πολλαπλά επίπεδα ελέγχου εγκυρότητας της φόρμας, όπως φαίνεται στην ***Εικόνα 3.13***.



Εικόνα 3.13 *Τα τρία επίπεδα ελέγχου εγκυρότητας των δεδομένων που εισάγονται σε μια φόρμα.*

Οι φάσεις ελέγχου εγκυρότητας είναι:

1. Έλεγχοι που γίνονται στον φυλλομετρητή από τα ίδια τα στοιχεία της φόρμας (HTML),
2. έλεγχοι που γίνονται στον φυλλομετρητή από κώδικα JavaScript,
3. έλεγχοι που γίνονται στον εξυπηρετητή στη γλώσσα που είναι γραμμένος ο εξυπηρετητής της ιστοσελίδας.

Βεβαίως, μπορεί να γίνει συνδυασμός των παραπάνω.

Η πρώτη φάση θα πρέπει να μη θεωρηθεί δεδομένη, αφού, όπως αναφέρθηκε, τα νέα στοιχεία HTML, τα οποία περιλαμβάνουν έλεγχο εγκυρότητας, δεν υποστηρίζονται από όλους τους φυλλομετρητές. Επίσης, να σημειωθεί ότι τα στοιχεία αυτά δεν περιλαμβάνουν τη γνώση της εφαρμογής αλλά γενικούς κανόνες. Όμως, παραμένει ένα πρώτο επίπεδο ελέγχου, το οποίο θα πρέπει να υποστηρίζεται, όπως για παράδειγμα η ύπαρξη του γνωρίσματος required σε ένα στοιχείο επιβάλλει την εισαγωγή μιας τιμής πριν την υποβολή της φόρμας.

Υπάρχει η δυνατότητα επιβολής πιο συγκεκριμένων κανόνων εγκυρότητας, μέσω του ορισμού ενός προτύπου (pattern) στο γνώρισμα pattern ενός στοιχείου <input>.

```
<input type=" " pattern ="RegEx" >
```

Μπορούμε να ορίσουμε ένα πρότυπο της συμβολοσειράς που θα εισαγάγει ο χρήστης μέσω του γνωρίσματος pattern, το οποίο δέχεται μια τυπική έκφραση (regular expression).

Ακολουθεί ένα παράδειγμα για ένα πρότυπο που αφορά τους ελληνικούς ταχυδρομικούς κώδικες, υποθέτοντας ότι οι ελληνικοί ταχυδρομικοί κώδικες είναι της μορφής XXX XX ή XXXXX, όπου X αριθμητικό ψηφίο.

```
<label for="postalCode">Ταχυδρομικός κώδικας:</label><br>
<input type="text" id="postalCode" name="pcode"
      pattern="^[0-9]{3}[ ]?[0-9]{2}$"><br><br>
```

Μια δεύτερη φάση ελέγχου εγκυρότητας είναι αυτή να γίνει στον φυλλομετρητή με χρήση κώδικα JavaScript. Συνήθως ορίζουμε μια συνάρτηση, η οποία καλείται όταν ο χρήστης πατήσει το πλήκτρο υποβολής της φόρμας. Η συνάρτηση αυτή καλείται πριν την υποβολή, μέσω του χειριστή συμβάντος onsubmit που ορίζεται στο στοιχείο <form>. Επίσης, μπορεί να υπάρχουν επιμέρους συναρτήσεις που ελέγχουν την τιμή που εισάγεται σε επιμέρους πεδία.

```
<form id="myForm" onsubmit="return(validate());">
```

Η συνάρτηση validate() πρέπει να επιστρέψει true αν ο έλεγχος εγκυρότητας είναι επιτυχής και false αν δεν είναι επιτυχής.

Ένα παράδειγμα συνάρτησης validate() για μια φόρμα που ζητάει την ηλικία του χρήστη με τον περιορισμό ότι η είσοδος δεν επιτρέπεται για χρήστες κάτω των 18 ετών είναι το εξής:

```
function validate(){
    if (document.querySelector("#age").value<18){
        document.querySelector("#message").innerHTML=
```

```
            "Δεν είναι επιτρεπτή η είσοδος σε άτομο ηλικίας μικρότερης των 18
ετών";
            // διαμόρφωση του μηνύματος σφάλματος
            document.querySelector("#message").style.color="#ef0000";
            document.querySelector("#message").autofocus = "on";
            document.querySelector("#myForm").reset();
            document.querySelector("#age").style.backgroundColor="white";
            return false;
        }
        return true;
    }
```

Να σημειωθεί εδώ ότι ο κώδικας αυτός θα γίνει περισσότερο κατανοητός όταν σε επόμενα κεφάλαια γίνει εισαγωγή στη γλώσσα προγραμματισμού JavaScript.

Το τελευταίο βήμα στον έλεγχο εγκυρότητας γίνεται στον εξυπηρετητή. Ο έλεγχος αυτός είναι απαραίτητος, γιατί υπάρχει πάντα το ενδεχόμενο κάποιος να προσπέρασε τους ελέγχους στην πλευρά του φυλλομετρητή και να έστειλε στον εξυπηρετητή μη έγκυρα δεδομένα, με κίνδυνο την ασφάλεια του συστήματος. Επίσης, στο επιχείρημα γιατί δεν αρκούμαστε στον έλεγχο αυτής της φάσης και απαιτούμε να γίνουν έλεγχοι και στον φυλλομετρητή, θα πρέπει να σημειώσουμε ότι είναι καλό η ανάδραση στον χρήστη να παρέχεται άμεσα και όχι αφού σταλούν τα δεδομένα στον εξυπηρετητή, όπως περιγράφεται και στις οδηγίες καλής σχεδίασης φορμών της επόμενης ενότητας.

3.10.9 Οδηγίες σχεδίασης φόρμας

Αντικείμενο της ενότητας αυτής είναι να γίνει μια σύντομη αναφορά στις βασικές οδηγίες που πρέπει να εφαρμόζουμε για την καλή σχεδίαση φορμών. Δεδομένης της σημασίας των φορμών στις εφαρμογές διαδικτύου, οι οδηγίες αυτές θα πρέπει να αποτελούν οδηγό μας όταν σχεδιάζουμε φόρμες.

Μια σύντομη λίστα από κανόνες είναι η εξής:

1. Η φόρμα να έχει πάντα κατατοπιστικό τίτλο.
2. Οι υποδείξεις να είναι λιτές και σαφείς.
3. Να είναι σαφές στον χρήστη ποια πεδία είναι υποχρεωτικά και ποια προαιρετικά.
4. Τα μηνύματα σφάλματος όταν προκύψουν να είναι υποστηρικτικά.
5. Να γίνεται λογική ομαδοποίηση και διάταξη πεδίων (βλέπε στοιχείο <fieldset>).
6. Η διάταξη περιεχομένου να είναι ελκυστική (χρώματα, αποστάσεις κ.λπ.).
7. Η πυκνότητα πληροφορίας στη φόρμα να είναι ομοιόμορφη.
8. Να παρέχεται εύκολη πρόσβαση σε πληροφορίες για νέους χρήστες.
9. Να υπάρχει ομοιομορφία σε υποδείξεις, μηνύματα σφάλματος και ορολογία.
10. Να παρέχεται ένδειξη για τα όρια, αν υπάρχουν, στο μέγεθος της πληροφορίας που θα εισαχθεί σε ένα πεδίο.
11. Ο έλεγχος ορθότητας τιμών να γίνεται έγκαιρα, κατά προτίμηση πριν την υποβολή της συνολικής φόρμας, αλλά κατά την εισαγωγή της μη έγκυρης τιμής.
12. Να παρέχονται συγκεκριμένες οδηγίες για την υποβολή της φόρμας.

Θα ολοκληρώσουμε τις οδηγίες για καλή σχεδίαση φορμών με επιλογή *20 κανόνων για εύχρηστες φόρμες* από τη σελίδα του D. Travis για την **ευχρηστία ιστοσελίδων**, ενότητα *Forms and data entry usability guidelines*. Όπως θα δούμε, κάποιες από τις οδηγίες ευχρηστίας επαναλαμβάνουν κανόνες καλής σχεδίασης που αναφέρθηκαν πιο πάνω, ενώ παρέχονται κανόνες που σχετίζονται με την ίδια τη διαδικασία συμπλήρωσης της φόρμας και πιθανά προβλήματα που ο χρήστης μπορεί να συναντήσει.

1. Τα πεδία της φόρμας πρέπει να περιέχουν προεπιλεγμένες τιμές όπου είναι δυνατόν.
2. Όταν μια διαδικασία περιλαμβάνει έντυπα, η φόρμα πρέπει να είναι συμβατή με τη δομή τους.
3. Θα πρέπει να προσφέρεται αυτόματη μορφοποίηση, π.χ. δύο πεδία με “,” μεταξύ τους για ακέραιο και δεκαδικό μέρος τιμής.
4. Οι υποδείξεις πρέπει να εξηγούν με σαφήνεια τι αναμένεται να εισαχθεί.
5. Τα πεδία κειμένου πρέπει να έχουν κατάλληλο μήκος για την αναμενόμενη απάντηση.
6. Να υπάρχει σαφής διάκριση μεταξύ «υποχρεωτικών» και «προαιρετικών» πεδίων.
7. Θα πρέπει να υπάρχει έγκαιρη προειδοποίηση εάν απαιτούνται εξωτερικές πληροφορίες για την ολοκλήρωση (π.χ. αριθμός διαβατηρίου) πριν ο χρήστης αρχίσει τη διαδικασία.

8. Τα πεδία στις φόρμες πρέπει να ομαδοποιούνται λογικά και κάθε ομάδα να έχει μια επικεφαλίδα.
9. Τα πεδία στις φόρμες θα πρέπει να περιέχουν υποδείξεις, παραδείγματα ή τυπικές απαντήσεις που να υποδεικνύουν την αναμενόμενη εισαγωγή.
10. Όταν οι υποδείξεις έχουν τη μορφή ερωτήσεων, οι ερωτήσεις αναφέρονται σε σαφή, απλή γλώσσα.
11. Τα αναπτυσσόμενα μενού, τα κουμπιά επιλογής και οι επιλογές ναι/όχι προτιμώνται από πεδία εισαγωγής κειμένου, γιατί ελαχιστοποιούν το γνωστικό φορτίο του χρήστη.
12. Ο δρομέας τοποθετείται εκεί όπου απαιτείται η είσοδος (γνώρισμα autofocus).
13. Οι αναμενόμενες μορφές δεδομένων υποδεικνύονται σαφώς (π.χ. ημερομηνίες) και ορίζονται οι μονάδες τιμών.
14. Οι χρήστες πρέπει να είναι σε θέση να ολοκληρώσουν απλές εργασίες εισάγοντας μόνο τις βασικές πληροφορίες (όχι άχρηστα υποχρεωτικά πεδία).
15. Η φόρμα πρέπει να επιτρέπει στους χρήστες να παραμένουν στην ίδια μέθοδο αλληλεπίδρασης για όσο το δυνατόν περισσότερο (να μη χρειάζεται να κάνουν συχνά μετάβαση από πληκτρολόγιο σε ποντίκι και πάλι σε πληκτρολόγιο).
16. Τα πεδία εισαγωγής κειμένου πρέπει να υποδεικνύουν το ποσό και τη μορφή των δεδομένων που πρέπει να εισαχθούν.
17. Να γίνεται έλεγχος των στοιχείων πριν από την υποβολή της φόρμας και να γίνεται έλεγχος σε επίπεδο πεδίου και σε επίπεδο φόρμας την κατάλληλη στιγμή.
18. Να παρέχεται διευκόλυνση στη διόρθωση σφαλμάτων (π.χ. όταν μια φόρμα δεν είναι πλήρης, τοποθετώντας τον δρομέα στην τοποθεσία όπου απαιτείται διόρθωση).
19. Να υπάρχει συνέπεια μεταξύ της εισαγωγής και της εμφάνισης δεδομένων.
20. Οι υποδείξεις να βρίσκονται κοντά στα πεδία εισαγωγής δεδομένων (π.χ. στοιχισμένες δεξιά).

3.11 Ασκήσεις

Άσκηση 1

Να δημιουργήσετε μια σελίδα στην οποία θα περιέχονται πληροφορίες για σας και την περιοχή που ζείτε. Να ενσωματώσετε φωτογραφία και έναν χάρτη της περιοχής σας. Να χρησιμοποιήσετε σημασιολογικά δομικά στοιχεία.

Άσκηση 2

Να δημιουργήσετε μια ιστοσελίδα με πληροφορίες για τον πλανήτη Αφροδίτη. Να περιέχει εικόνες και βίντεο καθώς και πληροφορίες για τον πλανήτη σε μορφή κειμένου. Να χρησιμοποιήσετε σημασιολογικά δομικά στοιχεία.

Άσκηση 3

Να κατασκευάσετε μια ιστοσελίδα εισαγωγής στοιχείων για κράτηση δωματίου σε μια αλυσίδα ξενοδοχείων.

Η φόρμα να περιέχει τις εξής ομάδες στοιχείων:

Προσωπικά στοιχεία – όνομα, επώνυμο, διεύθυνση, τηλέφωνο, email.

Επιλογή ξενοδοχείου – από μια λίστα ξενοδοχείων να επιλεχθεί ένα.

Τύπος δωματίου – να επιλέξτε ανάμεσα στους τύπους: μονόκλινο/δίκλινο/τρίκλινο/σουίτα.

Ημερομηνίες κράτησης – να περιέχει ημερομηνία άφιξης και ημερομηνία αναχώρησης.

Τρόπος πληρωμής – να περιέχει μια από τις παρακάτω επιλογές:

- Κατάθεση σε τραπεζικό λογαριασμό
- Πιστωτική κάρτα

Η δεύτερη επιλογή να έχει ακόμη τις εξής επιλογές: Visa/Mastercard/American Express.

Τέλος, η φόρμα να έχει πλήκτρο υποβολής και καθαρίσματος της φόρμας.

3.12 Ερωτήσεις αυτοαξιολόγησης

1. Επιθυμούμε να ενσωματώσουμε την εικόνα myfig.jpg που βρίσκεται στον υποφάκελο figs της ιστοσελίδας μας. Να συμπληρώσετε τον παρακάτω κώδικα: <img src=".....">
Απάντηση: _____
2. Ποια η διαφορά μεταξύ των γνωρισμάτων alt και title ενός στοιχείου <img>;
    1. Το περιεχόμενο του title εμφανίζεται ως tooltip με πρόσθετη πληροφορία στον χρήστη, ενώ το alt εμφανίζεται ως πληροφορία περιγραφής της εικόνας όταν αυτή λείπει.
    2. Το περιεχόμενο του title εμφανίζεται ως τίτλος στο πάνω μέρος της εικόνας ενώ το alt είναι εναλλακτική πληροφορία περιγραφής της εικόνας όταν αυτή λείπει.
    3. Δεν υπάρχει διαφορά, και τα δύο αφορούν πρόσθετη πληροφορία για την εικόνα.
3. Γιατί δεν είναι καλή ιδέα να δώσουμε τιμή και στα δύο γνωρίσματα height και width μιας εικόνας <img>;
    1. Γιατί φορτώνουμε το αρχείο html με περιττά γνωρίσματα, μόνο το ένα από τα δύο θα ληφθεί υπόψη.
    2. Γιατί υπάρχει κίνδυνος να παραμορφωθεί η εικόνα αν ο λόγος τους δεν είναι αυτός της αρχικής εικόνας.
    3. Δεν επιτρέπονται τα γνωρίσματα αυτά, η εικόνα θα καταλάβει τον χώρο που προσδιορίζει η αρχική της μορφοποίηση.
4. Αν θέλουμε να εισαγάγουμε ένα κόκκινο πλαίσιο με διάφανο εσωτερικό, ποιους τύπους εικόνας θα πρέπει να επιλέξουμε; (Επιλέξτε τύπους εικόνων που επιτρέπουν διαφάνεια.) Σημειώστε όλες τις απαντήσεις που ταιριάζουν.
    1. gif
    2. jpeg
    3. png
    4. svg
5. Αν μια εικόνα διατίθεται σε μια ιστοσελίδα με άδεια creative commons και ένδειξη attribution-share alike (CC BY-SA),
    1. δεν επιτρέπεται να την περιλάβουμε στην ιστοσελίδα μας αφού προστατεύεται ο δημιουργός της.
    2. επιτρέπεται να την περιλάβουμε στην ιστοσελίδα μας αφού η άδεια αυτή επιτρέπει την ελεύθερη χρήση.
    3. επιτρέπεται να την περιλάβουμε στην ιστοσελίδα μας αρκεί να μην την τροποποιήσουμε.
    4. επιτρέπεται να την περιλάβουμε στην ιστοσελίδα μας αρκεί να αναφέρουμε τον δημιουργό της.
6. Έστω το παρακάτω στοιχείο:

    ```
    <picture>
      <source srcset="pix1.png" media="(min-width: 800px)">
      <source srcset="pix2.png" media="(min-width: 600px)">
      <img src="pix3.png" alt="this is me">
    </picture>
    ```

    Ποια εικόνα θα δει ο χρήστης αν η οθόνη του έχει πλάτος 400px;
Απάντηση: _____
7. Αν θέλουμε το βίντεο που εισάγουμε με την ετικέτα <video> να μη σταματάει να παίζει, ποιο γνώρισμα πρέπει να προσθέσουμε;
Απάντηση: _____
8. Αν θέλουμε το βίντεο να παίξει όταν φορτωθεί η σελίδα, ποιο γνώρισμα πρέπει να προσθέσουμε;
Απάντηση: _____
9. Έστω <video src="video.mp4" width="400" height="400" > και το βίντεο video.mp4 έχει λόγο προβολής widescreen 16:9. Ποια η διάσταση του παράθυρου προβολής του βίντεο (πλάτος x ύψος);
    1. 400x400 px
    2. 400x300 px
    3. 1600x900 px
    4. 400x225 px
10. Για να προστατέψουμε το περιβάλλον του υπολογιστή μας από το περιεχόμενο μιας άλλης πηγής που

ενσωματώνουμε με την ετικέτα <iframe> θα πρέπει να προσθέσουμε το γνώρισμα:
Απάντηση: _____

11. Θα πρέπει να είμαστε ιδιαίτερα προσεκτικοί στη χρήση στοιχείων <iframe> γιατί:
    1. το περιεχόμενο του εξωτερικού στοιχείου μπορεί να αποτελεί απειλή ασφάλειας για τον υπολογιστή μας.
    2. το περιεχόμενο του εξωτερικού στοιχείου μπορεί να υπόκειται σε περιορισμούς προστασίας πνευματικών δικαιωμάτων.
    3. το περιεχόμενο του εξωτερικού στοιχείου μπορεί να είναι διαφορετικής αισθητικής από την υπόλοιπη ιστοσελίδα μας.
12. Ένα αρχείο myfile.pdf μπορεί να ενσωματωθεί σε μια ιστοσελίδα με χρήση της ετικέτας:
    Απάντηση: _____
13. Αν ένα αρχείο mypage.html περιέχει το εξής στοιχείο: <form method="post">, τι θα γίνει όταν ο χρήστης της φόρμας επιλέξει το πλήκτρο submit της φόρμας αυτής;
    1. Το περιεχόμενο της φόρμας θα σταλεί στη σελίδα mypage.html του server.
    2. Το περιεχόμενο της φόρμας δεν θα σταλεί στον server γιατί δεν έχει οριστεί η τιμή του γνωρίσματος action στο στοιχείο <form>.
    3. Το περιεχόμενο της φόρμας θα σταλεί στην προκαθορισμένη σελίδα index.html στον server, αφού δεν έχει οριστεί η τιμή του γνωρίσματος action στο στοιχείο <form>.
    4. Ο χρήστης θα πάρει μήνυμα σφάλματος 304 resource not modified.
14. Αν ένα αρχείο mypage.html περιέχει το στοιχείο <form> χωρίς άλλα γνωρίσματα, τι θα γίνει όταν ο χρήστης της φόρμας επιλέξει το πλήκτρο submit της φόρμας αυτής;
    1. Η φόρμα δεν θα σταλεί στον server γιατί δεν έχει οριστεί το γνώρισμα method.
    2. Η φόρμα θα σταλεί στον server με τη μέθοδο get.
    3. Η φόρμα θα σταλεί στον server με τη μέθοδο post.
    4. Η φόρμα θα σταλεί στον server με τη μέθοδο put.
    5. Η φόρμα θα σταλεί στον server με τη μέθοδο head.
15. Έστω ιστοσελίδα mypage.html που περιέχει την παρακάτω φόρμα

```
<form>
  <input type="text" id="myname" name="name">
  <input type="submit" value="submit">
</form>
```

στην οποία ο χρήστης επιλέγει το πλήκτρο submit αφού έχει εισαγάγει τη συμβολοσειρά “Maria” στο πλαίσιο εισαγωγής κειμένου.
Περιγράψτε την εντολή του μηνύματος HTTP που θα παραχθεί:

1. GET mypage.html?name=Maria HTTP/1.1
2. POST index.html?name=Maria HTTP/1.1
3. GET mypage.html?myname=Maria HTTP/1.1
4. POST mypage.html?myname=Maria HTTP/1.1
5. GET mypage.html?myname=Maria HTTP/1.1
6. POST mypage.html HTTP/1.1
7. GET index.html?myname=Maria HTTP/1.1

3.13 Βιβλιογραφία και Αναφορές

Η βιβλιογραφία αυτού του κεφαλαίου είναι αντίστοιχη με αυτή του προηγούμενου εισαγωγικού κεφαλαίου στην HTML. Βασική αναφορά είναι το πρότυπο HTML, το οποίο συντηρείται από την κοινότητα WHATWG.

Στο διαδίκτυο υπάρχουν πολλές πηγές για εκμάθηση της HTML καθώς και για αναφορά στα στοιχεία της γλώσσας. Η MDN είναι μια καλή πηγή για εισαγωγικά και προχωρημένα μαθήματα, όπως επίσης η w3schools.

Βιβλία για την HTML συνήθως περιλαμβάνουν ένα σύνολο από άλλες τεχνολογίες, όπως η CSS και η JavaScript. Στην ελληνική βιβλιογραφία, εκτός από το βιβλίο των Δουληγέρη κ.ά. (2021), υπάρχει ακόμη το βιβλίο των Μαρκατσέλα και Ξαρχάκου (2013), το οποίο όμως δίνει κυρίως έμφαση στη CSS, ενώ έχουν μεταφραστεί κάποια συγγράμματα και διατίθενται από τον Εύδοξο. Παράδειγμα το βιβλίο των Kyrnin και Morrison (2021) και το βιβλίο των Lemay, Coburn και Kyrnin (2016).

Επιπλέον, οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Για αυτό το κεφάλαιο το σχετικό μάθημα είναι οι εισαγωγικές διαλέξεις του μαθήματος «Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, Javascript». Επίσης, στο μάθημα «Προχωρημένα θέματα ανάπτυξης ιστοσελίδων» γίνεται εμβάθυνση στη λειτουργία των φορμών της HTML, αφού όμως έχει γίνει αναφορά στην JavaScript και την CSS.

Ελληνόγλωσσες

Αβούρης, Ν. (2018). Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, JavaScript. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr

Αβούρης, Ν., & Σιντόρης, Χ. (2020). Προχωρημένα θέματα ανάπτυξης ιστοσελίδων. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr

Δουληγέρης, Χ., Μαυροπόδη, Ρ., Κοπανάκη, Ε., & Καραλής, Α. (2021). *Τεχνολογίες και Προγραμματισμός στον Παγκόσμιο Ιστό* (2η έκδ.). Εκδόσεις Νέων Τεχνολογιών. Κωδικός βιβλίου στον Εύδοξο: 102125023.

Kyrnin, J., & Meloni, J. (2021). *Sams Teach Yourself HTML5, CSS and Javascript* (3rd ed.). Εκδόσεις Γκιούρδας.

Lemay, L., Coburn, R., & Kyrnin, J. (2016). *Sams Teach Yourself HTML, CSS & JavaScript* (7th ed). Εκδόσεις Γκιούρδας. Κωδικός Βιβλίου στον Εύδοξο: 59357307.

Μαρκατσέλας, Μ., & Ξαρχάκος, Κ. (2013). *CSS3 & Εφαρμογές*. Εκδόσεις Άβακας, Αθήνα. Κωδικός βιβλίου στον Εύδοξο: 68369452.

Κεφάλαιο 4. CSS

Σύνοψη

Μετά την HTML (κεφάλαια 2 και 3) παρουσιάζεται η CSS, η δεύτερη σημαντική τεχνολογία προγραμματισμού διαδικτυακών εφαρμογών. Το κεφάλαιο ξεκινά με σύντομη ιστορική αναδρομή και παρουσίαση της γλώσσας και των προτύπων της. Στη συνέχεια παρουσιάζονται οι πρώτες βασικές έννοιες των επιλογέων και των οδηγιών CSS, καθώς και η έννοια της ιεραρχίας επάλληλων φύλλων στιλ. Το κεφάλαιο συνεχίζει με την εισαγωγή σε βασικές οδηγίες της CSS και με τη συζήτηση για τους μηχανισμούς που διαθέτει η γλώσσα σχετικά με μονάδες για τα μεγέθη. Έπειτα παρουσιάζεται διεξοδικά το μοντέλο box της CSS και συζητούνται έννοιες και οδηγίες που αφορούν το υπόβαθρο, το πλαίσιο, την υπερχείλιση, τις εικόνες, τις λίστες κ.ά. Το κεφάλαιο κλείνει με την παρουσίαση του μηχανισμού των μεταβάσεων (transitions).

Είναι απαραίτητη η εξοικείωση με την HTML (κεφάλαια 2 και 3), καθώς οι δύο αυτές τεχνολογίες αλληλοσυμπληρώνονται. Η μεν HTML αφορά τη δομή και το περιεχόμενο ιστοσελίδων, ενώ η CSS την εμφάνιση και παρουσίασή τους.

Προαπαιτούμενη γνώση

Μελέτη των κεφαλαίων 2 και 3.

4.1 Η γλώσσα CSS

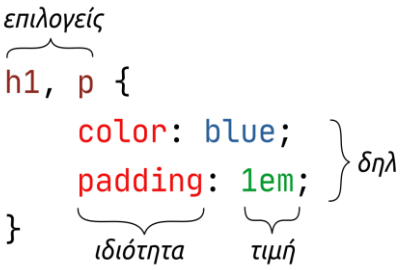
Η CSS (*Cascading Style Sheets, Επάλληλα φύλλα στιλ*) είναι γλώσσα με την οποία περιγράφεται η *παρουσίαση* του περιεχομένου ενός εγγράφου και εφαρμόζεται σε έγγραφα που είναι γραμμένα σε κάποια γλώσσα, όπως σε HTML, XML, SVG κ.ά. Με τη χρήση της CSS πραγματοποιούμε τον διαχωρισμό του περιεχομένου της ιστοσελίδας (που περιέχεται στο έγγραφο HTML) από την παρουσίασή του (που περιέχεται στις εντολές CSS που συνήθως ενσωματώνονται σε ξεχωριστά αρχεία css). Παλιότερα, οι οδηγίες περιγραφής της παρουσίασης περιλαμβάνονταν μέσα στην ίδια την HTML, όπως για παράδειγμα με χρήση των ετικετών και <b> (bold), που στους φυλλομετρητές εμφανίζονται με έντονα γράμματα. Η χρήση όμως των ετικετών αυτών για να παρουσιάσουμε κείμενο με έντονα γράμματα αποθαρρύνεται σήμερα και αυτός ο ρόλος ανατίθεται στη γλώσσα CSS, π.χ. με την ιδιότητα `font-weight`.

Η πρώτη έκδοση της CSS, η CSS1, δημοσιεύτηκε το 1996 ως W3C CSS Recommendation (CSS1) και ήταν αποτέλεσμα της συνεργασίας που είχε ξεκινήσει δύο χρόνια νωρίτερα μεταξύ του Νορβηγού Håkon Wium Lie και του Ολλανδού Bert Bos. Η CSS1, όπως και η επόμενη έκδοση, η CSS Επίπεδο 2, περιγράφονται σε ένα ενιαίο κείμενο χωρισμένο σε κεφάλαια. Η τρέχουσα πρόταση του W3C είναι η CSS2.1 (Level 2 Revision 1). Η CSS Επίπεδο 3 αποτελείται από διαφορετικές, ανεξάρτητες μεταξύ τους, ενότητες. Κάθε ενότητα της CSS3 επεκτείνει τις αντίστοιχες δυνατότητες της CSS2. Έτσι, κάθε ενότητα της CSS3 βρίσκεται σε διαφορετικό στάδιο ωρίμανσης, που μπορεί να χαρακτηρίζεται ως Πρόχειρη (Working Draft), Υποψήφια Πρόταση (Candidate Recommendation) και Πρόταση (Recommendation), καθώς και σε διαφορετικό επίπεδο. Για παράδειγμα, η ενότητα CSS Fonts βρίσκεται στο επίπεδο 3, ενώ η ενότητα CSS Flexible Box Module είναι στο επίπεδο 1. Οι ενότητες CSS3 που ολοκληρώνονται ενσωματώνονται σταδιακά στην τρέχουσα CSS2.1.

4.2 Κανόνες CSS

Οι οδηγίες της CSS γράφονται σε αρχεία κειμένου που ονομάζονται CSS Stylesheets, φύλλα CSS, και αποτελούνται από έναν ή περισσότερους **κανόνες** (CSS rules). Κάθε τέτοιος κανόνας δίνει οδηγίες στον φυλλομετρητή για το πώς να εμφανίσει συγκεκριμένα στοιχεία της σελίδας. Ένας τέτοιος κανόνας αποτελείται από τα εξής στοιχεία (***Εικόνα 4.1***):

- **Επιλογέας** (selector). Καθορίζει το στοιχείο ή τα στοιχεία στα οποία έχει ισχύ ο κανόνας.
- **Δήλωση** (declaration). Αποτελείται από μια **ιδιότητα** (property) και την **τιμή** της (value).



Εικόνα 4.1 *Τα συστατικά ενός κανόνα CSS.*

Στο παράδειγμα στην ***Εικόνα 4.1*** ορίζεται ότι τα στοιχεία <h1> και <p> του εγγράφου θα περιέχουν κείμενο με μπλε χρώμα και το περιθώριό τους από τα υπόλοιπα στοιχεία της σελίδας θα είναι μεγέθους 1em (θα δούμε με περισσότερη λεπτομέρεια τις μονάδες μέτρησης, καθώς και τα περιθώρια ενός στοιχείου box στη συνέχεια αυτού του κεφαλαίου).

Ας δούμε ένα ακόμη παράδειγμα. Ο παρακάτω κανόνας καθορίζει πως όλες οι παράγραφοι (τα στοιχεία <p>) του εγγράφου θα περιέχουν κείμενο με σκούρο γκρι χρώμα:

```
p {
    color: darkgray;
}
```

4.3 Άλλες μορφές δηλώσεων CSS

Οι κανόνες CSS είναι ο πιο συνήθης τρόπος που έχουμε για να δώσουμε στον φυλλομετρητή οδηγίες για την εμφάνιση της σελίδας μας. Υπάρχουν ωστόσο και άλλες μορφές κανόνων στη γλώσσα CSS.

4.3.1 at-rules

Οι κανόνες αυτοί αρχίζουν με το σύμβολο @ (at) και δηλώνουν μεταδεδομένα ή άλλες πληροφορίες. Συνήθεις at-rules είναι:

- **@import**. Φορτώνει κανόνες που βρίσκονται σε εξωτερικό αρχείο, π.χ.

  ```
  @import url('page-url.css')
  ```

- **@media**. Εφαρμόζει το στιλ ανάλογα με το μέσο (media) προβολής, π.χ.

  ```
  @media screen and (min-width: 900px) {
      ...
  }
  ```

- **@supports**. Εφαρμόζει το στιλ ανάλογα με το επίπεδο CSS που υποστηρίζει ο φυλλομετρητής, π.χ.

  ```
  @supports not (display: grid) {
      ...
  }
  ```

- **@font-face**. Καθορίζει μια γραμματοσειρά.

4.4 Επιλογείς CSS

Οι επιλογείς ενός κανόνα CSS καθορίζουν σε ποια στοιχεία θα εφαρμοστούν οι δηλώσεις. Οι βασικοί επιλογείς είναι ο **καθολικός επιλογέας**, ο **επιλογέας τύπου**, ο **επιλογέας κλάσης**, ο **επιλογέας id** και ο **επιλογέας ιδιότητας**. Απλοί επιλογείς μπορούν να συνδυαστούν σε πιο πολύπλοκους, όπως θα δούμε στη συνέχεια. Η τρέχουσα πρόταση της W3C για τους επιλογείς είναι αυτή του επιπέδου 3 (Selectors Level 3).

Στη συνέχεια αυτής της ενότητας θα παρουσιαστούν σύντομα οι διαθέσιμοι επιλογείς καθώς και το πώς μπορούν να συνδυαστούν. Η παράθεση των επιλογέων έχει έναν κάπως εγκυκλοπαιδικό χαρακτήρα, καθώς οι

επιλογείς είναι πολλοί και παρουσιάζονται σε συντομία, συνοδευόμενοι από σύντομα παραδείγματα. Συνιστάται να αποκτηθεί επίγνωση όλων των επιλογέων, ακόμη και όσων φαίνονται «εξωτικοί» με την πρώτη ματιά. Σε κάθε περίπτωση, ο αναγνώστης μπορεί να ανατρέξει σε αυτή την ενότητα για να διαλευκάνει τον τρόπο λειτουργίας των πιο ιδιαίτερων επιλογέων και αργότερα.

4.4.1 Καθολικός επιλογέας (Universal selector)

Ο καθολικός επιλογέας, που συμβολίζεται με *, επιλέγει όλα τα στοιχεία, οποιουδήποτε τύπου. Επιλέγονται όλα τα στοιχεία της σελίδας, π.χ.

```
* {
    color: black; /*όλα τα στοιχεία της σελίδας θα έχουν μαύρα γράμματα*/
}
```

4.4.2 Επιλογέας τύπου (Type selector)

Ο επιλογέας τύπου επιλέγει όλα τα στοιχεία του συγκεκριμένου τύπου, π.χ.

```
p, h1, ul {
    color: green; /*όλα τα p, h1 και ul θα έχουν πράσινα γράμματα*/
}
```

Συμπληρωματικά με τον επιλογέα τύπου έχουμε και τον επιλογέα ψευδο-στοιχείου (pseudo-element), με τον οποίο μπορούμε να επιλέξουμε ένα *τμήμα* του στοιχείου. Για παράδειγμα, με το ψευδο-στοιχείο **::first-line** μπορούμε να επιλέξουμε την πρώτη γραμμή μιας παραγράφου. Αυτή η γραμμή μπορεί στην οθόνη του χρήστη να περιλαμβάνει διαφορετικό πλήθος λέξεων, που αλλάζει ανάλογα με το διαθέσιμο πλάτος στον εκάστοτε φυλλομετρητή. Άλλα ενδιαφέροντα ψευδο-στοιχεία είναι τα **::before** και **::after**, που εισάγουν περιεχόμενο στην αρχή ή στο τέλος του επιλεγμένου στοιχείου:

```
/* Εισάγει τη λέξη "Προσοχή: " στην αρχή κάθε στοιχείου p με κλάση
.warning  */
p.warning::before {
    content: "Προσοχή: "
}
```

4.4.3 Επιλογέας κλάσης (Class selector)

Επιλέγονται όλα τα στοιχεία που στην ιδιότητα class τους αναγράφεται ο επιλογέας. Ο επιλογέας κλάσης αρχίζει με μια τελεία, ακολουθούμενη από το όνομα μιας κλάσης, π.χ.

```
.first {
    font-weight: bold; /* τα στοιχεία με κλάση first θα έχουν έντονα
γράμματα */
}

<!-- το li έχει δύο κλάσεις, τη first και τη my-list. Στην HTML η κλάση
γράφεται χωρίς τελεία -->
<li class="first my-list">Πορτοκάλια</li>
```

Ο επιλογέας κλάσης μπορεί να συνδυαστεί με τον επιλογέα τύπου, για παράδειγμα:

```
p.introductory {
    font-style: italic; /* τα στοιχεία <p> με κλάση introductory θα
εμφανίζονται με πλάγια γράμματα */
}

<!-- το <p> έχει την κλάση introductory -->
<p class="anotherclass introductory">Η εισαγωγική παράγραφος θα
εμφανίζεται με πλάγια γράμματα, καθώς έχει την κλάση introductory.</pi>
```

Τέλος, μπορούν να επιλεγούν στοιχεία που έχουν περισσότερες από μια συγκεκριμένες κλάσεις, όπως φαίνεται στο παράδειγμα:

```
/* επιλέγονται τα στοιχεία που έχουν και τις δύο κλάσεις dark και
important */
*.dark.important {
    background-color: darkgray;
    color: white;
    font-style: italic;
}

<p class="anotherclass dark important">Η παράγραφος θα εμφανίζεται με
άσπρα πλάγια γράμματα σε σκούρο γκρι φόντο, καθώς έχει και τις δύο κλάσεις
του επιλογέα που ορίστηκε πιο πάνω.</li>
```

Η χρήση του καθολικού επιλογέα * δεν ήταν υποχρεωτική στο παραπάνω παράδειγμα. Ο επιλογέας θα μπορούσε να είναι και **.dark.important**, ωστόσο συχνά γράφουμε ρητά τον καθολικό επιλογέα για να είναι πιο ευανάγνωστος ο κώδικας CSS.

4.4.4 Επιλογέας id (Id selector)

Επιλέγεται το στοιχείο που έχει την ιδιότητα id. Ο επιλογέας id αρχίζει με δίεση # ακολουθούμενη από το id του στοιχείου που θέλουμε να επιλεγεί, π.χ.

```
/* τα στοιχεία με id friendly θα έχουν έντονα γράμματα */
#friendly {
    font-weight: bold;
}

<!-- κάθε id πρέπει να είναι μοναδικό στη σελίδα -->
<span id="friendly">Καλωσήλθατε</span>
```

Σημειώστε πως ένα στοιχείο μπορεί να έχει μόνο μια τιμή στην ιδιότητα id, δηλαδή δεν επιτρέπεται να έχουμε τιμή στο id που να περιέχει το κενό διάστημα. Επίσης, σύμφωνα με το πρότυπο, η τιμή του id πρέπει να είναι μοναδική στη σελίδα.

Αν χρησιμοποιήσουμε το ίδιο id σε περισσότερα στοιχεία στη σελίδα μας, ο φυλλομετρητής δεν θα τα αγνοήσει αλλά θα εφαρμόσει τον κανόνα CSS σε όλα τα στοιχεία με το ίδιο id. Ωστόσο, ενθαρρύνεται ο κώδικας να είναι σύμφωνος με τα πρότυπα.

4.4.5 Επιλογέας ιδιότητας (Attribute selector)

Ο επιλογέας ιδιότητας επιτρέπει την επιλογή με βάση την ύπαρξη και την τιμή που μπορεί να έχει κάποια ιδιότητα των στοιχείων. Ο επιλογέας αυτός είναι πιο σύνθετος, καθώς υπάρχει σε τέσσερις μορφές, ανάλογα με την τιμή που μπορεί να έχει η ιδιότητα:

- **[ιδιότητα]**
  Στην πιο απλή μορφή του, μπορεί να χρησιμοποιηθεί για να επιλέξει τα στοιχεία που έχουν κάποια ιδιότητα, ανεξάρτητα από την τιμή της, π.χ.

  ```
  /* τα στοιχεία που έχουν την ιδιότητα disabled θα εμφανίζονται με μια
  οριζόντια διαγράμμιση */
  [disabled] {
      text-decoration: line-through;
  }
  ```

- **[ιδιότητα**=τιμή**]**
  Τα στοιχεία τα οποία στην ιδιότητα έχουν ακριβώς την καθορισμένη τιμή.

  ```
  /* τα στοιχεία με την ιδιότητα type='button' θα εμφανίζονται με μαύρο
  περίγραμμα */
  [type=button] {
  ```

```
    border-color: black;
}
```

- **[ιδιότητα~=τιμή]**
  Σε μερικές ιδιότητες μπορεί να καταχωριστεί μια λίστα με τιμές χωρισμένες με κενό διάστημα. Ο επιλογέας ~= επιλέγει τα στοιχεία που στη λίστα τιμών τους περιέχεται η τιμή.
- **[ιδιότητα|=τιμή]**
  Ο επιλογέας |= επιλέγει όλα τα στοιχεία που η ιδιότητά τους είναι είτε ακριβώς τιμή είτε αρχίζει με τιμή-. Συχνό παράδειγμα είναι τιμές που αφορούν κωδικό γλώσσας, όπως en-US, en-UK, el-GR, el-CY κ.λπ.

```
/* τα στοιχεία <a> με την ιδιότητα hreflang='el' ή οτιδήποτε αρχίζει με
hreflang='el-.....' εμφανίζονται υπογραμμισμένα */
a[hreflang|="el"] {
    text-decoration: underline;
}
```

Επιπλέον, ο επιλογέας ιδιότητας μπορεί να επιλέξει και με βάση τμήμα της τιμής (substring matching) με τρεις επιπλέουν τρόπους:

- **[ιδιότητα^=τιμή]** Επιλέγει στοιχεία που η τιμή της ιδιότητας **αρχίζει** με τιμή.
- **[ιδιότητα$=τιμή]** Επιλέγει στοιχεία που η τιμή της ιδιότητας **τελειώνει** με τιμή.
- **[ιδιότητα*$*=τιμή]** Επιλέγει στοιχεία που η τιμή της ιδιότητας **περιέχει** τουλάχιστον μια φορά το αλφαριθμητικό τιμή.

4.4.6 Επιλογέας ψευδοκλάσης (Pseudoclass selector)

Οι επιλογείς ψευδοκλάσης μάς επιτρέπουν να επιλέξουμε στοιχεία με άλλο τρόπο, πέρα από την αναζήτηση στο DOM. Οι επιλογείς αυτοί αρχίζουν πάντα με `:` ακολουθούμενο από το όνομα της ψευδοκλάσης. Οι ψευδοκλάσεις μπορούν μεταξύ άλλων να αφορούν τη δυναμική ή στατική κατάσταση ενός στοιχείου, τη θέση του μέσα στο DOM κ.λπ.

Δυναμικές ψευδοκλάσεις

Οι δυναμικές ψευδοκλάσεις είναι οι:

- **`:link`** Επιλέγει τους συνδέσμους που δεν έχει επισκεφτεί ο χρήστης του φυλλομετρητή.
- **`:visited`** Επιλέγει τους συνδέσμους που έχει επισκεφτεί ο χρήστης του φυλλομετρητή.
- **`:hover`** Επιλέγει τον σύνδεσμο τον οποίο δείχνει ο χρήστης, πάνω από τον οποίο δηλαδή αιωρείται ο δείκτης της συσκευής κατάδειξης (π.χ. του ποντικιού).
- **`:active`** Επιλέγει τον σύνδεσμο που ενεργοποιείται από τον χρήστη, δηλαδή τον υπερσύνδεσμο στον οποίο έχει πατήσει ο χρήστης για όσο διαρκεί το κλικ.
- **`:focus`** Επιλέγει τον σύνδεσμο που είναι εστιασμένος, που μπορεί να έχει επιλεγεί με το πληκτρολόγιο ή το ποντίκι.

```
a:link { ... } /* σύνδεσμοι που δεν έχει επισκεφτεί ο χρήστης */
a:visited { ... } /* σύνδεσμοι που έχει επισκεφτεί ο χρήστης */
a:hover { ... } /* σύνδεσμοι που από πάνω τους αιωρείται ο δείκτης του
ποντικιού */
a:active { ... } /* ενεργοποιημένος σύνδεσμος */
a:focus:hover { ... } /* σύνδεσμοι που έχουν την εστίαση και που πάνω τους
αιωρείται ο δείκτης του ποντικιού */
```

Δομικές ψευδοκλάσεις

Οι δομικές ψευδοκλάσεις επιλέγουν στοιχεία με βάση τη θέση τους στο DOM, με τρόπο που δεν παρέχεται από άλλους επιλογείς.

- **`:root`** Επιλέγει το αρχικό στοιχείο, που σε μια σελίδα HTML είναι το στοιχείο html.
- **`:empty`** Επιλέγει στοιχεία που δεν έχουν παιδιά.

Μια πολύ χρήσιμη κατηγορία ψευδοκλάσεων είναι αυτές που επιλέγουν στοιχεία με βάση τη θέση τους

στο DOM, με πιο χαρακτηριστικό τον επιλογέα **`:nth-child(An+B)`**. Τα A και B είναι σταθερές που δίνονται από τον χρήστη. Επιλέγονται όλα τα στοιχεία που βρίσκονται στη θέση *i* που υπολογίζεται από την έκφραση *i=An+B* για μη αρνητικό ακέραιο *n*. Οι θέσεις των στοιχείων ξεκινούν από το 1. Ας δούμε μερικά παραδείγματα στο παρακάτω απόσπασμα HTML:

```
<body>
    <p>1ο στοιχείο</p>
    <p>2ο στοιχείο</p>
    <p>3ο στοιχείο</p>
    <p>4ο στοιχείο</p>
    <p>5ο στοιχείο</p>
    <p>6ο στοιχείο</p>
    <p>7ο στοιχείο</p>
    <p>8ο στοιχείο</p>
    <p>9ο στοιχείο</p>
</body>
```

Ο επιλογέας **:nth-child(3n)** θα επιλέξει κάθε 3ο στοιχείο. Ο επιλογέας αυτός ισοδυναμεί με **:nth-child(3n+0)**. Ξεκινώντας από το *n=0*, η έκφραση i=3*0+0 ισούται με 0 και συνεπώς θα επιλεγούν τα στοιχεία για n=1, n=2 και n=3 στις θέσεις 3, 6 και 9 αντίστοιχα. Ο επιλογέας **:nth-child(3n+1)** θα επιλέξει το 1ο, το 4ο και το 7ο στοιχείο. Αντίστοιχα, ο επιλογέας **:nth-child(3n-1)** θα επιλέξει το 2ο, το 5ο και το 8ο στοιχείο.

Όλα τα στοιχεία που βρίσκονται σε ζυγές ή μονές θέσεις μπορούμε να τα επιλέξουμε και με τις λέξεις κλειδιά even και odd, π.χ. **:nth-child(even)**. Μπορούμε να κάνουμε την επιλογή μετρώντας ανάποδα, από το τέλος προς την αρχή, με τον επιλογέα **:nth-last-child(An+B)**.

Τέλος, μπορούμε να χρησιμοποιήσουμε και τους επιλογείς **:nth-of-type**, **:nth-last-of-type**, **:first-of-type**, **:last-of-type** που λειτουργούν παρόμοια, με τη διαφορά πως επιλέγονται μόνο στοιχεία ενός συγκεκριμένου τύπου. Για παράδειγμα, ο επιλογέας **p:nth-of-type(2n)** επιλέγει μόνο κάθε 2η παράγραφο.

Άσκηση

Ποια στοιχεία θα επιλεγούν με τον επιλογέα `:nth-child(-3n+1);`

Απάντηση

Μόνο το 1ο στοιχείο, δηλαδή για *n=0*, που βρίσκεται στη θέση i=-3*0+1=1, καθώς για *n>=1* το *i* θα είναι αρνητικό.

Άσκηση

Γράψτε έναν επιλογέα που να επιλέγει όλα τα στοιχεία σε ζυγές θέσεις και έναν που να τα επιλέγει σε μονές θέσεις.

Απάντηση

Τα στοιχεία σε ζυγές θέσεις μπορούν να επιλεγούν για *A*=2 και *B*=0, δηλαδή `:nth-child(2n)`. Αντίστοιχα, τα στοιχεία σε μονές θέσεις επιλέγονται με τον επιλογέα `:nth-child(2n+1)`. Το ίδιο αποτέλεσμα μπορούμε να έχουμε με τους επιλογείς `nth-child(even)` ή `nth-child(odd)`.

4.4.7 Συνδυασμοί επιλογέων

Υπάρχει λοιπόν ένας μεγάλος αριθμός από επιλογείς με τους οποίους μπορούμε να επιλέξουμε στοιχεία και να στοχεύσουμε έτσι το πού θα εφαρμοστούν οι κανόνες μας. Συνδυάζοντάς τους, μπορούμε να έχουμε ακόμη πιο σύνθετους επιλογείς.

Στη συνέχεια, με A και B σημειώνονται οι επιλογείς που συνδυάζονται.

- **A B** Είναι ο πιο συνήθης συνδυασμός και επιλέγει τα στοιχεία B που είναι απόγονοι, άμεσοι ή και όχι των στοιχείων A. Οι δύο επιλογείς A και B χωρίζονται με το κενό διάστημα.
- **A>B** Σε αυτή την περίπτωση επιλέγονται τα B που είναι **άμεσοι** απόγονοι των A.
- **A~B** Επιλέγονται τα B που είναι αδέρφια των A, όχι απαραίτητα άμεσα. Αδέρφια είναι τα στοιχεία που

έχουν τον ίδιο άμεσο πρόγονο.

- **A+B** Επιλέγονται τα B που είναι άμεσα επόμενα αδέρφια των A, δηλαδή όσα B που ακολουθούν αμέσως μετά από ένα A.

Μπορούμε επίσης, με το "," (κόμμα) να κατασκευάσουμε μια λίστα επιλογέων, στους οποίους θα εφαρμοστεί ο κανόνας:

- **A,B** Επιλέγονται τα στοιχεία που επιλέγονται από τον A καθώς και τα στοιχεία που επιλέγονται από τον B. Αν έστω και ένας από τους επιλογείς της λίστας δεν είναι συντακτικά έγκυρος, τότε είναι άκυρη όλη η λίστα. Για παράδειγμα, το

```
h1 {font-color: rebeccapurple}
h2 {font-color: rebeccapurple}
h3 {font-color: rebeccapurple}
```

ισοδυναμεί με

```
h1,h2,h3 {font-color: rebeccapurple}
```

4.5 Αλληλουχία και σειρά εφαρμογής των κανόνων

Ένας κανόνας CSS περιέχει ιδιότητες και τις τιμές τους και ο επιλογέας καθορίζει σε ποια στοιχεία θα εφαρμοστούν τα ζεύγη ιδιοτήτων-τιμών. Πολύ συχνά, περισσότεροι από έναν κανόνες (***Εικόνα 4.1***) μπορεί να περιέχουν δηλώσεις που να αφορούν τις ίδιες ακριβώς ιδιότητες για το ίδιο στοιχείο. Ο μηχανισμός που αποφασίζει ποιες τιμές τελικά θα επικρατήσουν εξετάζει την προέλευση του κανόνα και αν η ιδιότητα έχει σημανθεί ως σημαντική ή όχι, το πόσο στοχευμένος είναι ο επιλογέας, καθώς και τη σειρά με την οποία φορτώθηκε. Επίσης, κάποιες από τις ιδιότητες κληρονομούν την τιμή τους από τα στοιχεία που βρίσκονται πιο πριν στο δέντρο, ενώ άλλες ιδιότητες όχι. Αυτό που εκ πρώτης όψεως μπορεί να φαίνεται χαοτικό μάς επιτρέπει ωστόσο να γράφουμε τις ελάχιστες δυνατές δηλώσεις. Ο μηχανισμός εφαρμογής των δηλώσεων είναι τόσο κεντρικός στη CSS που είναι μέρος του ονόματος: "*Cascading*" στο Cascading Style Sheets σημαίνει ακριβώς «φύλλα στιλ» που εφαρμόζονται σε «αλληλουχία». Η λειτουργία του μηχανισμού αυτού περιγράφεται στην πρόταση Cascading and Inheritance Level 3.

Μπορούμε να συμπεράνουμε πως έχει σημασία η σειρά με την οποία συναντά ο φυλλομετρητής τους κανόνες. Στο τέλος, βέβαια, **όλες** οι ιδιότητες **όλων** των στοιχείων θα έχουν τιμές. Αν έχουν οριστεί τιμές σε περισσότερα από ένα σημεία, τότε μόνο μια από αυτές θα επικρατήσει. Ο τρόπος με τον οποίο καθορίζεται η τιμή που θα επικρατήσει εξαρτάται από την εξής αλληλουχία, από το πιο σημαντικό στο λιγότερο σημαντικό:

- Την **προέλευση** του κανόνα όπου δηλώνεται η τιμή και αν η τιμή έχει σημανθεί ως σημαντική ή όχι,
- την **ειδικότητα**, δηλαδή το πόσο στοχευμένος είναι ο επιλογέας που περιέχει τη δήλωση,
- τη **σειρά εμφάνισης** του κανόνα στον κώδικα,
- το αν η τιμή της ιδιότητας **κληρονομείται** ή όχι.

Στη συνέχεια θα δούμε τα επιμέρους τμήματα της αλληλουχίας με περισσότερη λεπτομέρεια.

4.5.1 Προέλευση κανόνων

Ένας κανόνας μπορεί να έχει μια από τις εξής τρεις προελεύσεις:

1. Στο λεγόμενο **στιλ φυλλομετρητή** (*user agent style*). Κάθε φυλλομετρητής έχει ενσωματωμένο ένα σύνολο κανόνων CSS, το λεγόμενο και προκαθορισμένο στιλ CSS, το οποίο έχει οριστεί από τον κατασκευαστή του. Το στιλ αυτό καθορίζει πώς θα εμφανιστούν τα στοιχεία, αν δεν υπάρχει άλλη CSS. Για παράδειγμα, το ότι τα γράμματα θα είναι μαύρα σε λευκό φόντο ή ότι τα <b> θα εμφανίζονται με έντονη γραμματοσειρά, ακόμη και αν δεν έχουμε ορίσει δικό μας στιλ, καθορίζεται στο στιλ φυλλομετρητή.
2. Στο **στιλ χρήστη** (*user defined style*). Ο χρήστης μπορεί να ορίσει τα δικά του αρχεία CSS και να ζητήσει από τον φυλλομετρητή να χρησιμοποιεί αυτά αντί για το προκαθορισμένο στιλ φυλλομετρητή. Όπως και το στιλ φυλλομετρητή, το στιλ χρήστη αφορά όλες τις ιστοσελίδες που φορτώνει ο χρήστης στον φυλλομετρητή. Καθώς το στιλ χρήστη είναι ρητή ενέργεια του χρήστη, οι κανόνες που περιέχει έχουν μεγαλύτερη προτεραιότητα από το στιλ φυλλομετρητή.
3. Στα στιλ που παρέχει ο δημιουργός της ιστοσελίδας, με τρεις τρόπους.
    1. **Εξωτερικό αρχείο CSS.** Ο δημιουργός της ιστοσελίδας ορίζει ένα ή περισσότερα αρχεία CSS

και τοποθετεί το URL τους στο τμήμα <head> της σελίδας, γράφοντας για παράδειγμα

```
<link rel="stylesheet" href="filename-url.css">
```

2. **Εσωτερικό CSS** (ή CSS σελίδας). Ο δημιουργός της σελίδας γράφει τους κανόνες CSS απευθείας στο τμήμα <head> μέσα στην ετικέτα <style>, π.χ.

```
<head>
    ...
    <style>
        p {
            color: darkgray;
        }
    </style>
    ..
</head>
```

Το εσωτερικό CSS αφορά μόνο τη συγκεκριμένη σελίδα και συνεπώς είναι πιο συγκεκριμένο από το εξωτερικό αρχείο CSS.

3. **Ενσωματωμένο ή inline στιλ**. Δηλώσεις CSS μπορούν να γραφτούν και μέσα σε μια ετικέτα HTML, ως τιμή της ιδιότητας <style>. Οι δηλώσεις αυτές έχουν εφαρμογή μόνο για το συγκεκριμένο στοιχείο, για παράδειγμα:

```
<p style="color: darkgray"> ... </p>
```

Οι παραπάνω πηγές κανόνων CSS είναι ταξινομημένες από τη λιγότερο στην περισσότερο σημαντική: στιλ φυλλομετρητή < στιλ χρήστη < εξωτερικό στιλ < εσωτερικό στιλ < ενσωματωμένο στιλ. Ο δημιουργός μιας ιστοσελίδας δεν έχει έλεγχο στις δύο πρώτες πηγές κανόνων CSS, το προκαθορισμένο στιλ και το στιλ χρήστη.

Από τις τρεις επιλογές που έχει ο δημιουργός ιστοσελίδων, το εξωτερικό CSS είναι η προτιμώμενη πρακτική. Ο λόγος είναι πως το εξωτερικό CSS μπορεί να αντιστοιχεί σε πολλές σελίδες HTML που συνθέτουν έναν ιστότοπο. Έτσι, αν θέλουμε να διορθώσουμε κάποιον κανόνα CSS που να αφορά όλες τις σελίδες του ιστοτόπου, π.χ. στο μενού, αρκεί τότε να τον αλλάξουμε σε ένα μόνο σημείο, στο εξωτερικό CSS. Στο άλλο άκρο, αποθαρρύνεται η χρήση του ενσωματωμένου (inline) στιλ, καθώς οι δηλώσεις αυτές έχουν τη μεγαλύτερη προτεραιότητα. Συνεπώς, αν θέλουμε να διορθώσουμε κάτι, θα πρέπει να το κάνουμε σε όλα τα στοιχεία HTML του ιστοτόπου μας στα οποία έχουμε εφαρμόσει τον ενσωματωμένο κανόνα ή να χρησιμοποιήσουμε τη λέξη !important.

Όλοι οι κανόνες CSS είναι είτε μη σημαντικοί (normal) είτε σημαντικοί (important). Σημαντικοί είναι αυτοί που η τιμή τους τελειώνει με `!important`, για παράδειγμα:

```
span .serious {
    color: red !important;
}
```

Μια σημαντική (!important) ιδιότητα έχει αυτομάτως μεγαλύτερο βάρος, δηλαδή, ανεξάρτητα σε ποια πηγή τη συναντήσαμε, η τιμή της θα επικρατήσει. Ωστόσο, τι γίνεται σε περίπτωση που η ίδια ιδιότητα έχει σημανθεί σαν σημαντική σε περισσότερες από μια πηγές; Τότε η προτεραιότητα επικράτησης θα ***αντιστραφεί****. Συνολικά λοιπόν έχουμε τη σειρά προτεραιότητας μιας δήλωσης, από την περισσότερο προς τη λιγότερο σημαντική, που φαίνεται στον*

Πίνακας 4.*1*:

Σειρά προτεραιότητας	Πηγή κανόνων
1 (μεγαλύτερη)	Δηλώσεις μετάβασης (transition)
2	!important στο στιλ του φυλλομετρητή
3	!important στο στιλ του χρήστη
4	!important στο στιλ του δημιουργού
5	Animation
6	Στο στιλ του δημιουργού
7	Στο στιλ του χρήστη
8 (μικρότερη)	Στο στιλ του φυλλομετρητή

Πίνακας 4.1 *Προτεραιότητες δηλώσεων ανάλογα με την πηγή στην οποία βρίσκονται. Οι δηλώσεις που έχουν να κάνουν με ενεργές μεταβάσεις (transitions) (Ενότητα 4.10) έχουν τη μεγαλύτερη προτεραιότητα (1), ενώ αυτές που αφορούν τα ενεργά animation (5) έχουν μεγαλύτερη προτεραιότητα μόνο από τις δηλώσεις δημιουργού, χρήστη και φυλλομετρητή.*

Συνεπώς, όταν ο φυλλομετρητής συναντήσει δύο διαφορετικές τιμές για την ίδια ιδιότητα ενός στοιχείου, θα επιλέξει αυτή με τη μεγαλύτερη προτεραιότητα, που προκύπτει από την πηγή προέλευσης του κανόνα.

4.5.2 Ειδικότητα

Αν η προέλευση δυο ή περισσότερων κανόνων είναι η ίδια, τότε θα επικρατήσει αυτός με τον πιο συγκεκριμένο επιλογέα. Το πόσο συγκεκριμένος είναι ο επιλογέας μετριέται με την **ειδικότητά** του (specificity). Η ειδικότητα ενός επιλογέα συντίθεται από τρία συστατικά στοιχεία:

- Το πλήθος των **επιλογέων id**,
- το πλήθος των **επιλογέων κλάσης** (π.χ. `.first`), των **επιλογέων ιδιότητας** (π.χ. **[type**=button**]**) και **επιλογέων ψευδοκλάσης** (π.χ. `:link`),
- το πλήθος των **επιλογέων τύπου** (π.χ. `p`) και των **επιλογέων ψευδοστοιχείων** (όπως το `::first-line`).

Για παράδειγμα, ο επιλογέας #friendly έχει ειδικότητα (1, 0, 0), καθώς αποτελείται από έναν επιλογέα id, ενώ ο #friendly #animal έχει ειδικότητα (2, 0, 0). Αντίστοιχα, ο επιλογέας `p.first  em` έχει ειδικότητα (0, 1, 2) καθώς αποτελείται από δύο επιλογείς τύπου (<p> και) και έναν επιλογέα κλάσης (`.first`). Επικρατεί πάντα ο επιλογέας που έχει τη μεγαλύτερη τιμή από αριστερά προς τα δεξιά, δηλαδή η ειδικότητα (1, 0, 0) είναι πιο ισχυρή από την (0, 1, 2). Στην περίπτωση που δύο επιλογείς έχουν την ίδια ειδικότητα θα επικρατήσει αυτός που συναντιέται τελευταίος.

4.5.3 Σειρά εμφάνισης

Αν δεν αρκούν τα δύο προηγούμενα κριτήρια για να αποφασιστεί η τιμή που θα επικρατήσει, δηλαδή αν οι κανόνες βρίσκονται στο ίδιο επίπεδο προέλευσης [
Πίνακας 4.1 Προτεραιότητες δηλώσεων ανάλογα με την πηγή στην οποία βρίσκονται. Οι δηλώσεις που έχουν να κάνουν με ενεργές μεταβάσεις (transitions) (Ενότητα 4.10) έχουν τη μεγαλύτερη προτεραιότητα (1), ενώ αυτές που αφορούν τα ενεργά animation (5) έχουν μεγαλύτερη προτεραιότητα μόνο από τις δηλώσεις δημιουργού, χρήστη και φυλλομετρητή.] και οι επιλογείς έχουν την ίδια ειδικότητα, τότε θα επικρατήσει η πιο πρόσφατη δήλωση, δηλαδή αυτή που συνάντησε τελευταία ο φυλλομετρητής όταν φόρτωσε την ιστοσελίδα:

```
/* Τελικά θα έχουμε κίτρινα γράμματα σε μαύρο φόντο  */
p {
    color: yellow;
    background-color: navy;
}
p {
    background-color: black;
}
```

4.5.4 Κληρονομικότητα

Όπως αναφέρθηκε, όλες οι ιδιότητες όλων των στοιχείων πρέπει υποχρεωτικά να έχουν μια τιμή. Αν η αλληλουχία δεν αποδώσει τιμή σε μια ιδιότητα, τότε θα χρησιμοποιηθεί η κληρονομικότητα. Όλες οι ιδιότητες έχουν προδιαγραφεί με μια **αρχική τιμή** (initial value). Για παράδειγμα, η αρχική τιμή της ιδιότητας font-size είναι medium. Αν λοιπόν δεν προκύψει τιμή από την αλληλουχία, τότε, ανάλογα με την ιδιότητα, η τιμή αυτή θα είναι είτε η αρχική τιμή της είτε θα κληρονομηθεί η τιμή (inherited value) από τον άμεσο πρόγονο στο DOM. Το αν θα γίνει αυτό εξαρτάται από το αν η ιδιότητα είναι κληρονομούμενη ή όχι, το οποίο προδιαγράφεται από το πρότυπο της CSS. Ένας εμπειρικός κανόνας είναι πως, γενικά, οι τιμές ιδιοτήτων που σχετίζονται με την εμφάνιση του κειμένου κληρονομούνται. Αντίστοιχα, δεν κληρονομούνται γενικά οι τιμές ιδιοτήτων που αφορούν την εμφάνιση των στοιχείων, π.χ. τα περιθώρια ή το περίγραμμα.

Για παράδειγμα, ας ορίσουμε στη CSS μας τις εξής τιμές για τη γραμματοσειρά και το μέγεθός της:

```
body {
    font-size: 1.2em;
    font-family: serif;
}
```

Οι τιμές αυτές θα εφαρμοστούν σε όλο το κείμενο της σελίδας μας, π.χ. στα στοιχεία <p> που πιθανώς περιέχονται στη σελίδα μας[1], καθώς τόσο το font-size όσο και το font-family κληρονομούνται, όπως μπορούμε να διαπιστώσουμε στο CSS Fonts Module Level 3 ή στη σχετική τεκμηρίωση της MDN. Ωστόσο, αν ορίσουμε επιπλέον τον εξής κανόνα, που αφορά το εξωτερικό περιθώριο του στοιχείου <body>

```
body {
    margin: 1em;
}
```

τότε αυτό θα εφαρμοστεί μόνο για το στοιχείο <body>, καθώς οι τιμές των ιδιοτήτων padding, border, margin δεν κληρονομούνται, όπως μπορούμε να διαπιστώσουμε στο αντίστοιχο πρότυπο CSS Box Model Module Level 3 ή την τεκμηρίωση του MDN.

Η συμπεριφορά αυτή μπορεί να αλλάξει ρητά. Μπορούμε, δηλαδή, να ζητήσουμε η ιδιότητα να πάρει την αρχική ή την κληρονομημένη τιμή χρησιμοποιώντας τις τιμές initial και inherit:

```
p {
    padding: inherit;
    background-color: initial;
}
```

Ερώτηση

Γιατί το στιλ χρήστη έχει μικρότερη προτεραιότητα από το στιλ δημιουργού, παρ' όλο που το πρώτο είναι ρητή ενέργεια του χρήστη;

Απάντηση

Το στιλ του χρήστη, όπως είπαμε, αφορά όλες τις σελίδες που φορτώνονται στον φυλλομετρητή, ενώ το στιλ δημιουργού αφορά μόνο τη συγκεκριμένη σελίδα. Συνεπώς, το στιλ δημιουργού είναι πιο ειδικό (specific) από το στιλ χρήστη.

4.6 Τιμές και μονάδες απόστασης και χρώματος

Κάθε ιδιότητα έχει έναν *τύπο* αποδεκτών τιμών. Οι τύποι αυτοί καλύπτουν διάφορες κατηγορίες, όπως κείμενο, απόσταση, αριθμητική τιμή, γωνία, χρόνο, συχνότητα, ανάλυση κ.λπ. Τους περισσότερους τύπους θα τους συναντήσουμε σε αυτό και στο επόμενο κεφάλαιο μέσα από τα παραδείγματα και σε αυτή την ενότητα θα δούμε με λεπτομέρεια τις μονάδες που εκφράζουν την απόσταση και το χρώμα.

Φυσικά, για να εκφράσουμε κάποιο μέγεθος, χρησιμοποιούμε αριθμούς είτε ακεραίους είτε

[1] Εκτός, φυσικά, και αν στη CSS μας υπάρχουν πιο στοχευμένες οδηγίες για τα στοιχεία <p>.

πραγματικούς. Οι αριθμοί αυτοί μπορεί να εμφανίζονται μόνοι τους είτε με κάποια μονάδα μέτρησης, όπως π.χ. κάποια μονάδα που μετράει απόσταση. Αριθμητικοί τύποι μπορούν να εκφραστούν και ως ποσοστά κάποιας άλλης ποσότητας, π.χ. του πλάτους ενός στοιχείου ή του μεγέθους μιας γραμματοσειράς. Οι μονάδες και τιμές που χρησιμοποιούνται στη CSS περιγράφονται στην υποψήφια πρόταση CSS Values and Units Module Level 3. Στη συνέχεια περιγράφονται διάφορες κατηγορίες τιμών.

4.6.1 Απόσταση

Οι τιμές που εκφράζουν απόσταση μετρούν είτε τη **σχετική** είτε την **απόλυτη** απόσταση. Οι σχετικές αποστάσεις υπολογίζουν την απόσταση με βάση κάποια άλλη απόσταση αναφοράς, που μπορεί να είναι είτε σε σχέση με το μέγεθος γραμματοσειράς είτε σε σχέση με τις διαστάσεις του παραθύρου προβολής (viewport), δηλαδή τις διαστάσεις του *κουτιού* μέσα στο οποίο περιέχεται η σελίδα μας. Οι σχετικές αποστάσεις είναι χρήσιμες όταν δεν γνωρίζουμε τις διαστάσεις στις οποίες θα προβληθεί η ιστοσελίδα μας. Διαδεδομένη είναι η χρήση της μονάδας em. Ένα em είναι ίσο με το μέγεθος της γραμματοσειράς που έχει το στοιχείο. Αν, για παράδειγμα, στο στοιχείο έχει οριστεί:

```
p {
    font-size: 16px;
    border: 0.5em solid black;
}
```

τότε 1em=10px και συνεπώς το πλάτος του περιγράμματος θα είναι 8px.

s	Σχετική με
em	το μέγεθος γραμματοσειράς του στοιχείου (ορισμένο ή κληρονομημένο)
ex	το ύψος του “x” στη γραμματοσειρά του στοιχείου
ch	το συνολικό πλάτος του “0” (ZERO, U+0030)
rem	το μέγεθος γραμματοσειράς του ριζικού στοιχείου (<body> ή :root)
vw	1% του πλάτους του παραθύρου προβολής (viewport width)
vh	1% του ύψους του παραθύρου προβολής (viewport height)

Πίνακας 4.2 *Σύνοψη των σχετικών μονάδων στο επίπεδο 3 του CSS Values and Units Module. Στο επίπεδο 4 της ίδιας ενότητας, που είναι όμως ακόμη πρόχειρη (working draft), έχουν εισαχθεί και άλλες μονάδες.*

Για παράδειγμα, ο κανόνας

```
p {
    padding: 1em;
    font-size: 1.2em;
}
```

ορίζει πως το μέγεθος της γραμματοσειράς του στοιχείου <p> θα είναι 120% του κληρονομημένου μεγέθους, ενώ το εσωτερικό περιθώριο θα είναι ακριβώς όσο και το μέγεθος της γραμματοσειράς (που είναι πλέον το 120% του κληρονομημένου μεγέθους). Ποια θα είναι όμως τελικά η απόλυτη τιμή αυτού του μεγέθους όταν θα εμφανιστεί στην οθόνη; Αν δεν έχουμε αλλάξει σε κανέναν πρόγονο του <p> το μέγεθος της γραμματοσειράς, τότε θα κληρονομηθεί η τιμή που έχει οριστεί για το <body>, η οποία θα είναι η τιμή που ορίζει το στιλ φυλλομετρητή. Αυτό συνήθως είναι 16px και συνεπώς τόσο το `padding` όσο και το `font-size` θα έχουν τιμή 16*1,2=19,2px.

Εκτός από την `em`, συνήθης είναι και η χρήση της μονάδας `rem`. Η διαφορά μεταξύ `em` και `rem` είναι πως η δεύτερη αναφέρεται πάντα στο μέγεθος της γραμματοσειράς του <body>. Η χρήση της `rem` είναι ίσως προτιμότερη, καθώς οι αλλαγές του μεγέθους της γραμματοσειράς με τη μονάδα `em` διαδίδονται αθροιστικά στο DOM:

```
p.warning { font-size: 2em; }

span.unit { font-size:2em; }
```

```
<!-- Η παράγραφος θα έχει μέγεθος γραμματοσειράς 32px ενώ το span 64px -->
<p class="warning">Οι αλλαγές του μεγέθους της γραμματοσειράς εμφωλευμένων
στοιχείων με μονάδες <span class="unit">em</span> δρουν αθροιστικά.</p>
```

Οι αλλαγές του μεγέθους της γραμματοσειράς εμφωλευμένων στοιχείων με μονάδες em δρουν αθροιστικά.

Εικόνα 4.2 *Επειδή η ιδιότητα font-size είναι κληρονομούμενη (ενότητα 4.5.4), η αλλαγή στο μέγεθος της γραμματοσειράς ενός στοιχείου κληροδοτείται στους απογόνους του και δρα αθροιστικά. Με τη μονάδα rem μπορούμε να το αποφύγουμε αυτό.*

Άλλες μονάδες που καθορίζονται από κάποιο μέγεθος γραμματοσειράς είναι οι ex και ch, η πρώτη σε σχέση με το μέγεθος του γράμματος x και η δεύτερη σε σχέση με το μέγεθος του 0.

Σχετικές αποστάσεις μπορούν να πάρουν τιμές και σε σχέση με τις διαστάσεις του παραθύρου προβολής (viewport). Οι μονάδες αυτές είναι οι vw και vh που είναι ίσες με το 1% του πλάτους ή του ύψους του viewport. Αν μας ενδιαφέρει πάντα η μικρότερη ή η μεγαλύτερη από αυτές τις δύο διαστάσεις, μπορούμε να χρησιμοποιήσουμε τις vmin και vmax.

Πρέπει να σημειωθεί πως, στην περίπτωση της κληρονομικότητας, οι σχετικές τιμές δεν κληρονομούνται, αλλά κληρονομείται η απόλυτη τιμή στην οποία τελικά θα καταλήξει ο υπολογισμός της σχετικής απόστασης.

Απόλυτες αποστάσεις μπορούν να οριστούν επίσης με πολλές μονάδες. Οι απόλυτες αποστάσεις είναι χρήσιμες όταν γνωρίζουμε τις διαστάσεις προβολής της ιστοσελίδας. Η πιο συχνά χρησιμοποιούμενη απόλυτη μονάδα είναι το πίξελ (px), που αντιστοιχεί σε 1/96 της ίντσας. Όλες οι απόλυτες αποστάσεις φαίνονται στον

Πίνακας 4.3.

Μονάδα	**Όνομα**	**Αντιστοιχία**
cm	εκατοστά	1cm = 96px/2.54
mm	χιλιοστά	1mm = 1/10 του εκατοστού
Q	τέταρτα του χιλιοστού	1Q = 1/40 του εκατοστού
in	ίντσες	1in = 2.54cm = 96px
pc	πίκα	1pc = 1/6 της ίντσας
pt	σημεία	1pt = 1/72 της ίντσας
px	πίξελ	1px = 1/96 της ίντσας

Πίνακας 4.3 *Οι απόλυτες μονάδες και η αντιστοιχία τους (CSS Values and Units Module Level 3).*

4.6.2 Χρώμα

Χρωματικές τιμές μπορούν να καθοριστούν είτε με κάποιο από τα 16 διαθέσιμα βασικά ονόματα χρωμάτων (web colors: black, silver, gray, white, maroon, red, purple, fuchsia, green, lime, olive, yellow, navy, blue, teal, aqua) είτε χρησιμοποιώντας κάποιο από τα *εκτεταμένα* χρώματα (extended colors). Η πλήρης λίστα των εκτεταμένων χρωμάτων μπορεί να βρεθεί μεταξύ άλλων στο πρότυπο CSS Color Module Level 3.

Είναι όμως δυνατό, αντί για τα ονοματισμένα χρώματα, να οριστεί το χρώμα με αριθμητικές τιμές. Με τις τιμές αυτές επιλέγεται ένα χρώμα από τα χρώματα που είναι διαθέσιμα στο χρωματικό χώρο sRGB (sRGB colorspace), που είναι με τη σειρά του ένας από τους πολλούς χρωματικούς χώρους που βασίζονται στο μοντέλο RGB. Αν και το sRGB δεν είναι σε θέση να αποτυπώσει όλα τα δυνατά χρώματα και ούτε καν όλα τα χρώματα που μπορούν να αποτυπώσουν οι σύγχρονες οθόνες, είναι ωστόσο το πρότυπο που επικρατεί και σήμερα για την κωδικοποίηση χρωμάτων στο διαδίκτυο. Αναμειγνύοντας τα τρία συστατικά χρώματα, κόκκινο, πράσινο και μπλε (**r**ed, **g**reen, **b**lue) του sRGB μπορούμε να παράξουμε όλα τα διαθέσιμα χρώματα. Στη CSS έχουμε στη διάθεσή μας 1 byte για καθένα από τα τρία χρώματα και άρα μπορούμε να επιλέξουμε τιμές από 0 ως 255

(ή από 0 ως FF στο δεκαεξαδικό σύστημα) και αυτό μπορούμε να το κάνουμε με διαφορετικούς τρόπους:

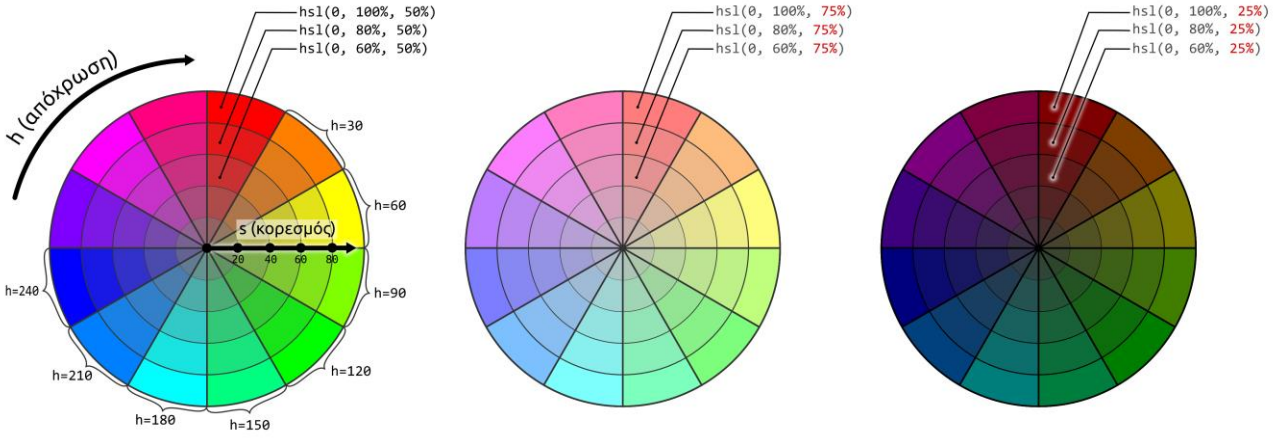
```
p { color: #f10; } /* ισοδυναμεί με #ff1100*/
p { color: #ff1100; }
p { color: rgb(255, 17, 0); }
p { color: rgb(100%, 6.7%, 0%); }
```

Συμπληρωματικά με την rgb() μπορούμε να χρησιμοποιήσουμε την rgba() για να δηλώσουμε και διαφάνεια με έναν πραγματικό αριθμό από 0, για πλήρη διαφάνεια, μέχρι 1, για πλήρη αδιαφάνεια. Μπορούμε να χρησιμοποιήσουμε και ποσοστό:

```
p { color: rgba(255, 17, 0, 1); } /* τελείως αδιαφανές */
p { color: rgba(100%, 6.7%, 0%, 50%); /* ημιδιαφανές */ }
```

Αντί για τα τρία βασικά χρώματα, κόκκινο, πράσινο και μπλε, μπορούμε να ορίσουμε χρώμα με την αναπαράσταση HSL (**h**ue, **s**aturation, **l**uminance – απόχρωση, κορεσμός, φωτεινότητα) (***Εικόνα 4.3***). Η απόχρωση **h** (το ποιο χρώμα θέλουμε) δηλώνεται με έναν ακέραιο αριθμό από 0 ως 360. Ο αριθμός αυτός δηλώνει γωνία στον χρωματικό κύκλο HSL, όπου 0=360=κόκκινο, 120=πράσινο και 240=μπλε. Ο κορεσμός **s**, δηλαδή το πόσο χρώμα θέλουμε, δηλώνεται με ένα ποσοστό, όπου το 0% είναι το γκρι. Αντίστοιχα και η φωτεινότητα **l**, όπου 0% είναι το μαύρο, 100% το λευκό και 50% η «κανονική» φωτεινότητα. Όπως και με την rgb, έχουμε στη διάθεσή μας δύο τρόπους για να ορίσουμε χρώματα hsv, χωρίς ή με διαφάνεια:

```
p { color: hsl(255, 100%, 50%); } /* κόκκινο */
p { color: hsla(255, 100%, 50%, 0.5); /* ημιδιαφανές κόκκινο */ }
```



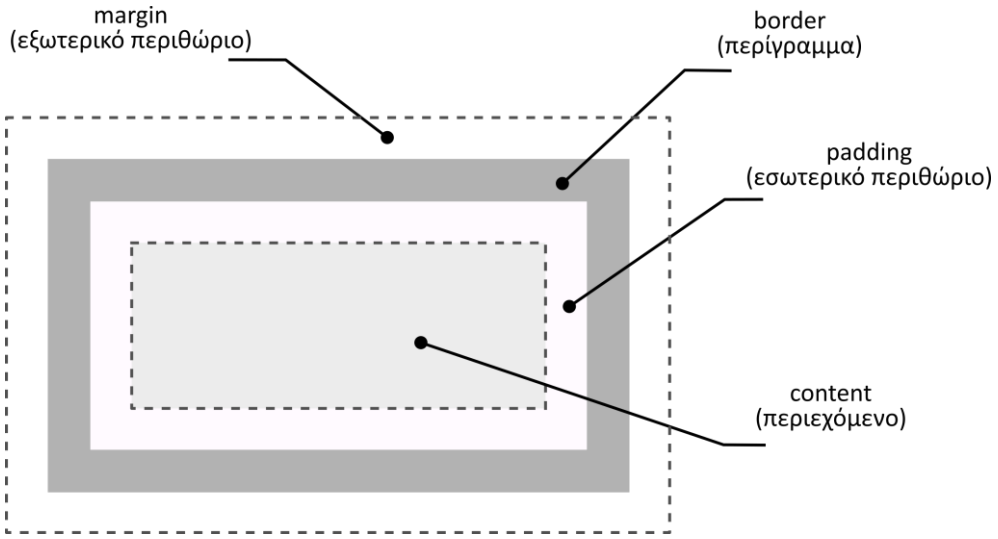
Εικόνα 4.3 *Στην αναπαράσταση HSL η γωνία h επιλέγει την απόχρωση, ξεκινώντας από τις 0° για το κόκκινο. Στο συγκεκριμένο σχήμα επιλέγονται 12 χρώματα, ένα χρώμα κάθε 30°. Όσο μειώνεται ο κορεσμός (s) από το 100% προς το 0%, το χρώμα πλησιάζει προς το γκρι – εδώ σε 5 βήματα του 20% από έξω προς το κέντρο του δίσκου. Ο παράγοντας l καθορίζει τη φωτεινότητα του χρώματος, 50% στο αριστερό σχήμα, 75% στο μεσαίο και 25% στο δεξί. Ο τρόπος αυτός επιλογής χρώματος είναι κάπως πιο προβλέψιμος σε σχέση με την αναπαράσταση RGB.*

4.7 Το μοντέλο Box

Το μοντέλο box είναι το θεμέλιο της διαδικασίας με την οποία ο φυλλομετρητής διατάσσει τα στοιχεία μιας ιστοσελίδας στην οθόνη. Στο μοντέλο αυτό τα στοιχεία HTML που συνθέτουν την ιστοσελίδα περιγράφονται σαν ορθογώνια κουτιά, τα οποία καταλαμβάνουν συγκεκριμένο χώρο στην οθόνη και τα οποία μπορούμε να χειριστούμε με τη γλώσσα CSS.

Κάθε τέτοιο κουτί αποτελείται, από μέσα προς τα έξω, από το **περιεχόμενο** (content), το **εσωτερικό περιθώριο** (padding), το **περίγραμμα** (border) και το **εξωτερικό περιθώριο** (margin) (**Εικόνα 4.3**).

Όπως είδαμε, αν δεν εφαρμόσουμε το δικό μας στιλ CSS και αφήσουμε τον φυλλομετρητή να διακοσμήσει τη σελίδα, τότε θα χρησιμοποιηθούν οι τιμές που ορίζονται στο προκαθορισμένο στιλ φυλλομετρητή (default stylesheet), το οποίο είναι ενσωματωμένο σε κάθε φυλλομετρητή.



Εικόνα 4.4 *Το μοντέλο box.*

Για κάθε κουτί μπορούμε να ορίσουμε την εξωτερική και την εσωτερική του συμπεριφορά, χρησιμοποιώντας την ιδιότητα display. Η εξωτερική συμπεριφορά περιγράφει το πώς το κουτί θα συμπεριφερθεί σε σχέση με τα γειτονικά του κουτιά. Υπάρχουν δύο τύποι κουτιών όσον αφορά τον εξωτερικό τύπο: τα κουτιά inline και τα κουτιά block.

Ένα στοιχείο τύπου block καταλαμβάνει από μόνο του όλο τον διαθέσιμο οριζόντιο χώρο, όλη δηλαδή τη γραμμή στην οποία είναι τοποθετημένο. Τέτοια στοιχεία είναι τα <p>, , <h1> κ.λπ. Αντίθετα, ένα στοιχείο τύπου inline καταλαμβάνει οριζόντια μόνο όσο χώρο χρειάζεται για να προβληθεί. Τέτοια στοιχεία είναι τα , κ.λπ. (***Εικόνα 4.5***).

Οι δύο αυτοί τύποι κουτιών διαφέρουν και σε άλλα σημεία. Ένα στοιχείο block θα είναι μόνο του στην οριζόντια γραμμή όπου βρίσκεται και θα καταλαμβάνει όλο το οριζόντιο εύρος αυτής της γραμμής, χωρίς άλλα στοιχεία αριστερά ή δεξιά του. Τα εσωτερικά και εξωτερικά περιθώρια και το περίγραμμα (padding, border και margin) του κουτιού τύπου block γίνονται σεβαστά και «σπρώχνουν» τα γειτονικά του στοιχεία που έπονται, έτσι ώστε να χωρέσει το κουτί. Επίσης, σεβαστές γίνονται και οι τιμές των ιδιοτήτων width και height.

Αντίθετα, ένα στοιχείο τύπου inline εμφανίζεται δίπλα σε άλλα στοιχεία inline, το ένα μετά το άλλο στην οριζόντια γραμμή. Αν δεν χωράει στην οριζόντια γραμμή, το στοιχείο inline αναδιπλώνεται σε νέα γραμμή και συνεχίζει από κάτω. Τα εσωτερικά και εξωτερικά περιθώρια και το περίγραμμα (padding, border και margin) του κουτιού τύπου inline γίνονται σεβαστά, αλλά προκαλούν τη μετατόπιση των γειτονικών στοιχείων μόνο στον οριζόντιο άξονα. Οι τιμές στις ιδιότητες width και height δεν έχουν κάποιο αποτέλεσμα.

Το αν ένα στοιχείο είναι block ή inline ορίζεται από την ιδιότητα display. Οι αντίστοιχες δηλώσεις είναι display: block και display: inline.

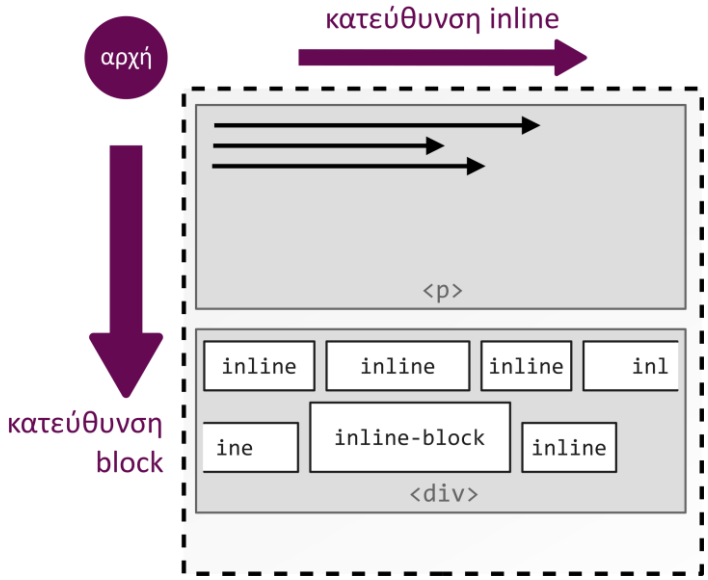
Η τρίτη βασική τιμή της ιδιότητας display είναι η none, που αφαιρεί τελείως το στοιχείο από την προβολή. Το στοιχείο δεν εμφανίζεται στην οθόνη, αλλά ούτε και δεσμεύεται ο χώρος που θα καταλάμβανε.

Όλα τα στοιχεία έχουν μια προκαθορισμένη τιμή για το display. Για παράδειγμα, το <p> ή το <h1> είναι στοιχεία block. Τα στοιχεία ή είναι τύπου inline.

Μια άλλη τιμή που μπορεί να πάρει η ιδιότητα display είναι η inline-block. Τα στοιχεία αυτά συμπεριφέρονται σαν inline, αλλά οι ιδιότητες padding και margin έχουν αποτέλεσμα τη μετατόπιση των γειτονικών στοιχείων και στις τέσσερις πλευρές (***Εικόνα 4.5***).

4.7.1 Κανονική ροή

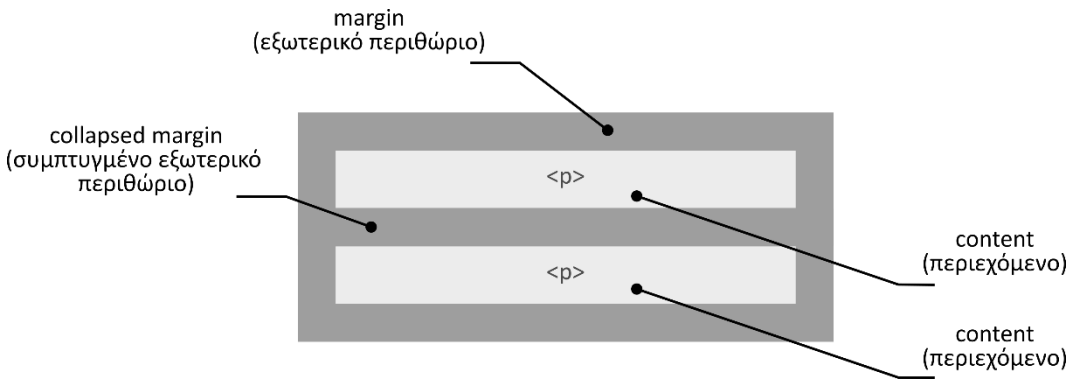
Η τοποθέτηση στοιχείων `block` από πάνω προς τα κάτω σε ξεχωριστές γραμμές και στοιχείων `inline` από τα δεξιά προς τα αριστερά το ένα δίπλα στο άλλο ονομάζεται **normal flow**, κανονική ροή.



Εικόνα 4.5 *Διάταξη των κουτιών inline και block στην κανονική ροή για συστήματα γραφής όπως τα δυτικά, όπου γράφουμε από τα αριστερά προς τα δεξιά και από πάνω προς τα κάτω.*

Η **κατεύθυνση inline** και η **κατεύθυνση block** εξαρτώνται από τη γλώσσα που χρησιμοποιούμε για το περιεχόμενο. Στα ελληνικά και στις υπόλοιπες δυτικές γλώσσες γράφουμε από τα αριστερά προς τα δεξιά και από πάνω προς τα κάτω και αντίστοιχη είναι η φορά των δύο αυτών κατευθύνσεων. Η κατεύθυνση αυτή αλλάζει με την ιδιότητα writing-mode, με την οποία μπορούμε να καθορίσουμε αν το κείμενό μας διατάσσεται οριζόντια, από πάνω προς τα κάτω, ή κάθετα, όπως για παράδειγμα γίνεται σε συστήματα γραφής γλωσσών της Ασίας.

Στην κανονική ροή, γειτονικά εξωτερικά περιθώρια (margin) γειτονικών μπλοκ θα συμπτυχθούν, με κάποιες εξαιρέσεις (CSS 2.1 Specification). Η συμπεριφορά αυτή είναι γνωστή ως **margin collapse**. Το συμπτυγμένο περιθώριο έχει μέγεθος ίσο με το μεγαλύτερο από τα δύο περιθώρια ή ίσο με ένα από τα περιθώρια αν τα δύο είναι ίσα. Η σύμπτυξη περιθωρίου συμβαίνει μόνο στην κατεύθυνση block (**Εικόνα *4.6***). Τα περιθώρια του στοιχείου `:root` δεν συμπτύσσονται.



Εικόνα 4.6 *Τα εξωτερικά περιθώρια δύο γειτονικών στοιχείων block συμπτύσσονται.*

4.7.2 Διαστάσεις στοιχείων

Για τον προσδιορισμό του μεγέθους που θα έχει ένα στοιχείο έχουν ιδιαίτερη σημασία δύο διαστάσεις, το εξωτερικό του μέγεθος και το εσωτερικό του μέγεθος. Το εξωτερικό μέγεθος ενός στοιχείου καθορίζει το πόσο χώρο χρειάζεται για να προβληθεί το στοιχείο, ενώ το εσωτερικό μέγεθος καθορίζει τις διαστάσεις του περιεχομένου του στοιχείου. Με άλλα λόγια, το εξωτερικό μέγεθος αντιστοιχεί στο εξωτερικό περιθώριο, ενώ το εσωτερικό μέγεθος στο περιεχόμενο (**Εικόνα *4.4***).

Ένα στοιχείο τύπου block παίρνει αυτόματα όλο το πλάτος που του διαθέτει ο άμεσος πρόγονός του,

δηλαδή το πλάτος του κουτιού που το περιέχει (το κουτί περιεχομένου στην **Εικόνα *4.4***). Αντίθετα, ένα στοιχείο τύπου inline έχει τόσο πλάτος όσο χρειάζεται για να εμφανιστεί το περιεχόμενό του.

Μπορούμε να καθορίσουμε τις εξωτερικές διαστάσεις ενός στοιχείου με τις ιδιότητες width και height. Με τις ιδιότητες max-width, min-width, max-height, min-height μπορούμε να θέσουμε συμπληρωματικά όρια ώστε το στοιχείο να μη γίνει μεγαλύτερο ή μικρότερό τους.

4.7.3 Υπερχείλιση

Όταν το περιεχόμενο ενός στοιχείου χρειάζεται περισσότερο χώρο για να προβληθεί από αυτόν που είναι διαθέσιμος, τότε έχουμε υπερχείλιση (overflow). Για τη διαχείριση αυτής της κατάστασης έχουμε διάφορες λύσεις.

Η ιδιότητα overflow (και οι αντίστοιχες overflow-x και overflow-y) καθορίζει τι θα γίνεται με το περιεχόμενο που ξεπερνά τα όρια του κουτιού του. Οι τιμές που παίρνει είναι

overflow: **visible** | hidden | clip | scroll | auto

- Η προκαθορισμένη τιμή είναι η visible και σημαίνει πως το περιεχόμενο θα υπερχειλίσει και θα ζωγραφιστεί έξω από τα όρια του στοιχείου.
- Αν πάρει την τιμή hidden, τότε το περιεχόμενο θα εκταθεί μέχρι και τα όρια του εσωτερικού περιθωρίου. Οτιδήποτε ξεπερνά τα όρια αυτά ψαλιδίζεται (clipping). Ωστόσο, το περιεχόμενο σε αυτή την περίπτωση είναι διαθέσιμο για κύλιση (scroll) μέσω κάποιου σκριπτ.
- Αυτό δεν είναι δυνατό με την τιμή clip, με την οποία το περιεχόμενο ψαλιδίζεται ώστε να μην ξεπεράσει τα όρια.
- Με την τιμή scroll το περιεχόμενο ψαλιδίζεται (clipped) μεν, αλλά εμφανίζονται μπάρες κύλισης (scrollbars).

4.8 Διακόσμηση στοιχείων

4.8.1 Επιλογή γραμματοσειράς

Ο ορισμός της γραμματοσειράς ενός στοιχείου γίνεται με την ιδιότητα font-family, με την οποία ορίζουμε μια ή περισσότερες γραμματοσειρές για το ίδιο στοιχείο. Η ιδιότητα font-family είναι κληρονομούμενη (ενότητα 4.5.4), που σημαίνει πως, αν οριστεί, για παράδειγμα, στο στοιχείο :root, θα ισχύει για όλα τα υπόλοιπα στοιχεία του DOM. Η ιστοσελίδα μας μπορεί να προβληθεί σε υπολογιστή με πολύ διαφορετικά χαρακτηριστικά από αυτά που περιμένουμε, και δεν μπορούμε να ξέρουμε ποιες γραμματοσειρές θα είναι διαθέσιμες στους φυλλομετρητές των τελικών χρηστών. Η CSS μάς δίνει διάφορους τρόπους για να αντιμετωπίσουμε αυτό το πρόβλημα και να έχουμε και κάποιο έλεγχο στην τελική εμφάνιση του κειμένου στον φυλλομετρητή.

Καταρχήν, μπορούμε να ορίσουμε περισσότερες από μια γραμματοσειρές και τότε θα εφαρμοστεί η πρώτη από αυτές που είναι διαθέσιμη στον φυλλομετρητή που προβάλλεται η ιστοσελίδα:

```
/* Αν δεν βρεθεί η Helvetica, θα εφαρμοστεί η Verdana, αλλιώς η
προκαθορισμένη sans-serif στον φυλλομετρητή */
body { font-family: Helvetica, Verdana, sans-serif; }
```

Επιτρέπεται όμως να δηλώσουμε και κάποιον από πέντε *γενικούς τύπους* γραμματοσειρών, τους serif, sans-serif, monospace, cursive, fantasy. Για αυτούς τους γενικούς τύπους, ο εκάστοτε φυλλομετρητής θα χρησιμοποιήσει τις αντίστοιχες προεπιλεγμένες γραμματοσειρές, που μπορεί να διαφέρουν από υπολογιστή σε υπολογιστή. Γι' αυτό τον λόγο οι γενικοί τύποι γραμματοσειρών συνιστάται να χρησιμοποιούνται σαν εφεδρική λύση, όπως φαίνεται στο παραπάνω παράδειγμα όπου η sans-serif είναι η ύστατη επιλογή.

Ακόμη πιο χρήσιμη είναι η δυνατότητα να ορίσουμε γραμματοσειρά που βρίσκεται στο διαδίκτυο, αντί για τον υπολογιστή του τελικού χρήστη. Δημοφιλείς βιβλιοθήκες online γραμματοσειρών είναι η Google Fonts, η Font Library, που προσφέρει γραμματοσειρές κυρίως με άδεια CC BY-SA, η Font Squirrel κ.ά. Οι υπηρεσίες αυτές παράγουν και τον κώδικα που χρειάζεται για να ενσωματωθούν οι γραμματοσειρές στη CSS μας.

Για παράδειγμα, αν επιθυμούμε να χρησιμοποιήσουμε τη δημοφιλή γραμματοσειρά Roboto Thin 100 της βιβλιοθήκης Google Fonts, αυτό μπορεί να γίνει με την παρακάτω εντολή CSS:

```
<style>
    @import
```

```
url('https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap
');
</style>
```

Στη συνέχεια, μπορούμε να χρησιμοποιήσουμε τη γραμματοσειρά σε κάποιο στοιχείο ως εξής:

```
<style>
    div {font-family: 'Roboto', sans-serif;}
</style>
```

Μερικές χρήσιμες ιδιότητες που μπορούν να χρησιμοποιηθούν σε συνάρτηση με τις γραμματοσειρές είναι:

- **font-size**. Καθορίζει το μέγεθος της γραμματοσειράς. Στην ενότητα 4.6.1 περιγράφονται οι σχετικές μονάδες.
- **font-style**. Δυνατές τιμές είναι normal, italic, oblique, για κανονική, πλάγια και πλαγιασμένη γραμματοσειρά. Η τελευταία είναι εξομοιωμένη πλάγια που παράγεται δίνοντας κλίση στην κανονική γραμματοσειρά.
- **font-weight**. Το βάρος της γραμματοσειράς μπορεί να καθοριστεί είτε αριθμητικά, από 100, 200 έως 900, είτε σε σχέση με το τρέχον βάρος της γραμματοσειράς ως lighter ή bolder, είτε με τις λέξεις normal και bold που ισοδυναμούν με 400 και 700 αντίστοιχα.
- **text-decoration**. Προσθέτει μια οριζόντια γραμμή στο κείμενο. Δυνατές τιμές είναι οι none, underline (υπογράμμιση), overline (υπεργράμμιση), line-through (διαγράμμιση) και μάλιστα μπορεί να πάρει και δύο τιμές ταυτόχρονα, π.χ. {**text-decoration**: underline overline;}. Μπορεί να αλλάξει και το στιλ της γραμμής με την text-decoration-style καθώς και το χρώμα της με text-decoration-color.

```
p {
    text-decoration-line: underline;
    text-decoration-style: dotted; /* ή double, dashed, wavy, solid */
    text-decoration-color: blue;
    /* ισοδυναμεί με */
    text-decoration: underline dotted blue;
}
```

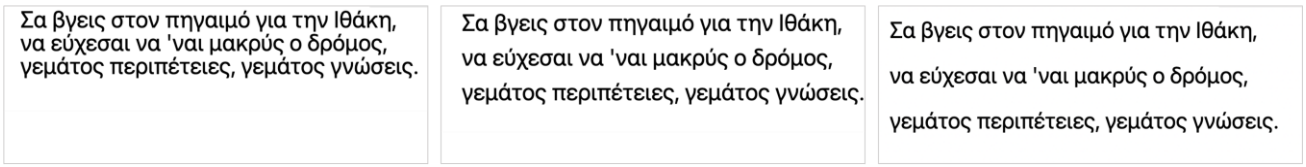
- **text-transform**. Μετασχηματίζει το κείμενο σε όλα κεφαλαία: uppercase, όλα μικρά: lowercase, τα πρώτα γράμματα κάθε λέξης κεφαλαία: capitalize, όλα τα γράμματα με το ίδιο πάχος: full-width.

Πέρα από τη γραμματοσειρά, μια σειρά από ιδιότητες επηρεάζουν και άλλα χαρακτηριστικά της εμφάνισης του κειμένου και βοηθούν να έχουμε καλύτερο τυπογραφικό έλεγχο. Από τις πιο σημαντικές ιδιότητες, η line-height ορίζει το ύψος της γραμμής στην οποία περιέχεται το κείμενο. Η line-height δέχεται τιμές σε οποιαδήποτε από τις διαθέσιμες μονάδες μέτρησης (ενότητα 4.6.1), αλλά και τιμές χωρίς μονάδα μέτρησης. Σε αυτή την περίπτωση, το line-height θα γίνει <τιμή>*<μέγεθος γραμματοσειράς>. Αυτός είναι και ο προτιμώμενος τρόπος, καθώς το ύψος της γραμμής θα είναι συνεπές, ακόμη και αν αλλάξει το μέγεθος της γραμματοσειράς του στοιχείου λόγω κληρονομικότητας.

Ακολουθεί ένα παράδειγμα. Έστω το παρακάτω έγγραφο.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <style>
        div { line-height: 1.0; font-size: 10pt; }
    </style>
</head>
<body>
    <div>
        Σα βγεις στον πηγαιμό για την Ιθάκη,<br>
        να εύχεσαι να 'ναι μακρύς ο δρόμος,<br>
        γεμάτος περιπέτειες, γεμάτος γνώσεις.<br>
    </div>
</body>
</html>
```

Όταν τροποποιήσουμε τον κανόνα ορισμού στιλ και δώσουμε τιμές 1, 1.5 και 2.0, αντίστοιχα, στην ιδιότητα line-height, θα έχουμε το αποτέλεσμα που φαίνεται στην **Εικόνα *4.7***.



Εικόνα 4.7 *Αποτέλεσμα του παραπάνω κώδικα για τιμές της ιδιότητας line-height=1.0, 1.5, 2.0 αντίστοιχα.*

Ενδιαφέρον παρουσιάζουν, τέλος, και οι ιδιότητες text-align και text-indent. Η πρώτη στοιχίζει το κείμενο μέσα στον υποδοχέα (κοντέινερ) και παίρνει τις τιμές left, right, center και justify. Η δεύτερη ιδιότητα δημιουργεί μια εσοχή στην πρώτη γραμμή της παραγράφου, χρησιμοποιώντας κάποια απόλυτη ή σχετική τιμή (ενότητα 4.6.1).

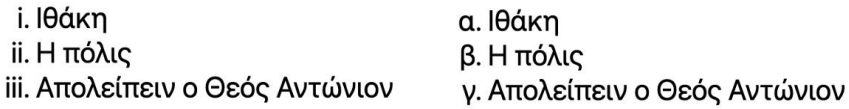
4.8.2 Διακόσμηση λίστας

Όπως είδαμε, στην HTML έχουμε δύο είδη λίστας, τη μη ταξινομημένη και την ταξινομημένη, <ul> και <ol>. Για τις μη ταξινομημένες λίστες η ιδιότητα list-style-type μπορεί να πάρει τις τιμές disc (αρχική τιμή), circle, square ή none.

Εικόνα 4.8 *Παρουσίαση στοιχείου <ul> με list-style-type=square και circle.*

Για τα στοιχεία των ταξινομημένων λιστών υπάρχει μια σειρά από διαφορετικές απαριθμήσεις που μπορούν να χρησιμοποιηθούν, όπως decimal (η αρχική τιμή), decimal-leading-zero, lower-roman, upper-roman, lower-greek κ.ά.

Έτσι, για παράδειγμα, αν η λίστα με τίτλους ποιημάτων του Αλεξανδρινού ποιητή μας γίνει στοιχείο <ol> και δώσουμε list-style-type: lower-roman ή lower-greek αντίστοιχα, θα πάρουμε το εξής αποτέλεσμα:



Εικόνα 4.9 *Παρουσίαση στοιχείου <ol> με list-style-type: lower-roman και lower-greek.*

4.8.3 Διακόσμηση υποβάθρου

Το υπόβαθρο κάθε στοιχείου της σελίδας μπορεί να μεταβληθεί με διάφορους τρόπους, π.χ. να καθορίσουμε για το υπόβαθρο ένα χρώμα ή μια διαβάθμιση (gradient) ή μια εικόνα. Ο καθορισμός του χρώματος γίνεται με την ιδιότητα background-color. Κάθε στοιχείο είναι αρχικά διαφανές. Η ιδιότητα background-color δέχεται οποιαδήποτε έγκυρη τιμή χρώματος (Ενότητα 4.6.2) και χρωματίζει το υπόβαθρο του κουτιού-περιεχομένου και του εσωτερικού περιθωρίου (***Εικόνα 4.4***).

Εναλλακτικά, ή και συμπληρωματικά, πάνω από το χρώμα υποβάθρου μπορούμε να ορίσουμε και μια εικόνα υποβάθρου, με την ιδιότητα background-image. Η ιδιότητα αυτή δέχεται μια ή περισσότερες εικόνες, που ζωγραφίζονται η μια κάτω από την άλλη. Για παράδειγμα, ο παρακάτω κανόνας θα δημιουργήσει στο υπόβαθρο μια στοίβα (stacking context, ενότητα 5.3) που περιέχει την εικόνα και από πάνω της τη διαβάθμιση.

```
div {
    background-image:
        radial-gradient(rgba(255, 136, 0, 0.425), rgba(238, 255, 0, 0.5)),
        url("https://openclipart.org/image/800px/319076");
    width: 100vh;
    height: 100vh;
    border: 5px dashed rgba(255,255,255,0.5);
    padding: 10px;
    margin: 5px;
    background-clip: border-box;
    background-origin: content-box;
    font-size: 1.5em;
}
```

Η εικόνα αυτή θα ψαλιδιστεί στο κουτί του πλαισίου (border), που είναι και η προκαθορισμένη τιμή της background-clip. Εναλλακτικά, μπορεί να ψαλιδιστεί και στο κουτί του περιεχομένου (content) ή στο κουτί του εσωτερικού περιθωρίου (padding). Με την ιδιότητα background-origin, που παίρνει αντίστοιχες τιμές, ορίζεται ποια θα είναι η αφετηρία της εικόνας, το πλαίσιο, το εσωτερικό περιθώριο ή το κουτί περιεχομένου.

Η τελική εικόνα φαίνεται στη συνέχεια (**Εικόνα *4.10***).

Άσκηση

Πειραματιστείτε με τις παραμέτρους του κανόνα CSS που παρήγαγε αυτό το αποτέλεσμα.

Εικόνα 4.10 *Παρουσίαση του στοιχείου της λίστας με εφαρμογή εικόνας υποβάθρου – Η εικόνα αφορά την ποιήτρια Υπατία, από την Αλεξάνδρεια.*

Πολύ χρήσιμες είναι οι τιμές της ιδιότητας background-repeat, που καθορίζουν τι θα γίνει αν οι διαστάσεις του κουτιού είναι μεγαλύτερες από την εικόνα. Η εικόνα μπορεί να επαναλαμβάνεται (repeat), που είναι η προκαθορισμένη συμπεριφορά, να επαναλαμβάνεται μόνο κάθετα ή οριζόντια (repeat-y, repeat-x) ή να μην επαναλαμβάνεται (no-repeat). Με τις τιμές round και space έχουμε ακόμη περισσότερο έλεγχο.

4.9 Αντικαθιστάμενα στοιχεία

Οι εικόνες, αλλά και άλλα μέσα που ενσωματώνονται στην ιστοσελίδα, όπως π.χ. ένα βίντεο ή ένα ενσωματωμένο έγγραφο PDF, είναι στοιχεία που το περιεχόμενό τους δεν μπορεί να επηρεαστεί από τη CSS. Μπορούμε να αλλάξουμε βέβαια τη θέση αυτών των στοιχείων ή την εμφάνιση του κουτιού που τα περιέχει, αλλά όχι το ίδιο το περιεχόμενο. Τα στοιχεία αυτά ονομάζονται replaced elements. Συχνά, οι διαστάσεις των

στοιχείων αυτών είναι *φυσικές*, δηλαδή προκύπτουν από το ίδιο το στοιχείο, όπως για παράδειγμα οι διαστάσεις μιας εικόνας.

Οι φυσικές διαστάσεις μιας εικόνας σπάνια είναι κατάλληλες για την τελική προβολή. Το σύνηθες, αντίθετα, είναι να θέλουμε να προβάλουμε μια εικόνα έτσι ώστε να χωράει σε συγκεκριμένες διαστάσεις. Μπορούμε να ορίσουμε τις διαστάσεις της εικόνας σε σχέση με τις διαστάσεις του κουτιού που την περικλείει με την ιδιότητα object-fit. Η object-fit παίρνει τις τιμές fill, none, contain και cover. Η επίδραση της object-fit φαίνεται στην ***Εικόνα 4.11***.

object-fit: **fill** | none | contain | cover

- Η προκαθορισμένη τιμή είναι η fill και σημαίνει πως το περιεχόμενο (η εικόνα) θα παραμορφωθεί ώστε να γεμίσει όλο τον χώρο του κουτιού που το περικλείει.
- Αν πάρει την τιμή none, τότε το περιεχόμενο δεν θα αλλάξει διαστάσεις.
- Με την τιμή contain η εικόνα θα χωρέσει ολόκληρη στο κουτί που την περιέχει, χωρίς να παραμορφωθεί, διατηρώντας δηλαδή τις αναλογίες της.
- Αντίθετα με την τιμή cover, η εικόνα θα κλιμακωθεί ώστε να καλυφθεί όλη η επιφάνεια του κουτιού που την περιέχει, πάλι χωρίς να παραμορφωθεί.

Αρχική εικόνα

Εικόνα 4.11 *Η επίδραση τεσσάρων από τις πιθανές τιμές της ιδιότητας object-fit (Τοπίο με την πτώση του Ίκαρου, Πίτερ Μπρίγκελ ο Πρεσβύτερος).*

4.10 Μεταβάσεις

Οι μεταβάσεις (transitions) είναι ο πιο απλός τρόπος που έχουμε για κινούμενα εφέ. Αν και είναι σχετικά απλός μηχανισμός, καλύπτει αρκετές ανάγκες.

Όταν αλλάζουμε την τιμή μιας ιδιότητας, π.χ. όταν αλλάζουμε το χρώμα της γραμματοσειράς με την ψευδοκλάση **:hover** (ενότητα 4.4.6), τότε η μετάβαση από την αρχική στην τελική κατάσταση είναι ακαριαία.

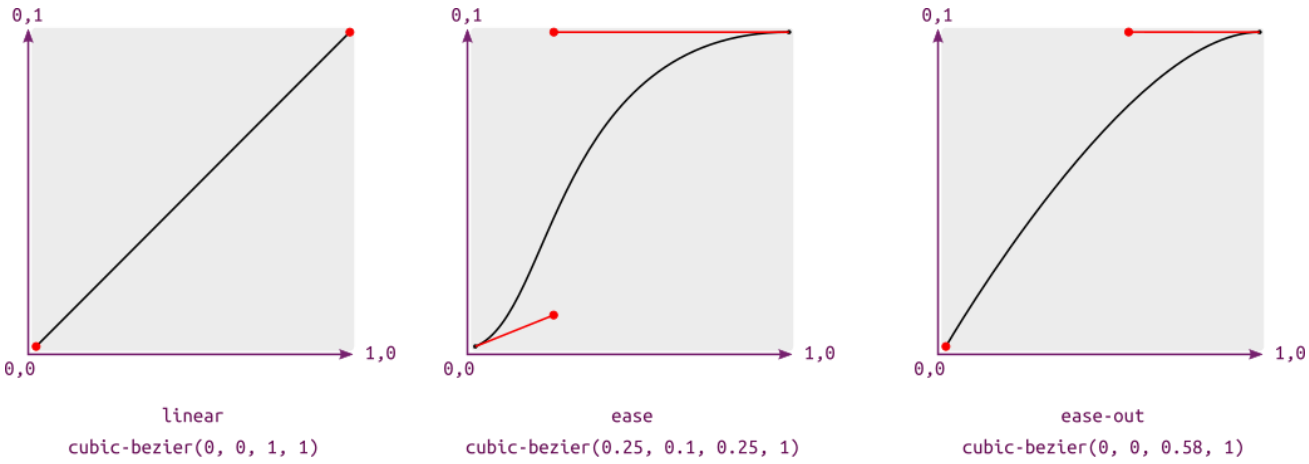
Με την ιδιότητα **transition** μπορούμε να περιγράψουμε πιο ομαλές μεταβάσεις ή, με άλλα λόγια, να αλλάξουμε τις τιμές τους σταδιακά και όχι απότομα. Για παράδειγμα:

```
<a href="">Ένας σύνδεσμος</a>

a {
    /* color: blue;
    background-color: white; */
```

```
    transition-property: background-color color ;
    transition-duration: 500ms;
    transition-timing-function: ease-in-out;
    transition-delay: 0s;
}

a:hover {
    background-color: navy;
    color: lightyellow;
}
```

Όταν ο δείκτης του ποντικιού αιωρηθεί πάνω από το <a>, θα γίνει σταδιακή μετάβαση από τις αρχικές τιμές των ιδιοτήτων background-color και color στις τελικές τιμές. Η μετάβαση αυτή θα ξεκινήσει χωρίς καθυστέρηση (transition-delay) και θα διαρκέσει 500ms (transition-duration). Ο υπολογισμός όλων των ενδιάμεσων καταστάσεων της μετάβασης θα γίνει από τον φυλλομετρητή χρησιμοποιώντας τη συνάρτηση χρονισμού (transition-timing-function).

Η ιδιότητα transition-timing-function μπορεί να πάρει κάποιες έτοιμες συναρτήσεις, όπως ease, ease-in, ease-out, ease-in-out, linear, step-start, step-end. Μπορούμε να καθορίσουμε τη δική μας συνάρτηση χρονισμού ορίζοντας μια παραμετρική καμπύλη Μπεζιέ με τη συνάρτηση CSS cubic-bezier() ή μια βαθμιδωτή συνάρτηση με τη συνάρτηση CSS steps(). Το online εργαλείο cubic-bezier.com της Λίας Βέρου μάς βοηθά αν θέλουμε να κατασκευάσουμε τις δικές μας καμπύλες Μπεζιέ.



Εικόνα 4.12 *Παραδείγματα για 3 από τις 5 προκαθορισμένες καμπύλες Μπεζιέ. Οι παράμετροι της συνάρτησης cubic-bezier() είναι οι συντεταγμένες των σημείων ελέγχου.*

Αν και δεν μπορούμε να χρησιμοποιήσουμε τη δυνατότητα αυτή σε όλες τις ιδιότητες της CSS, ωστόσο, η λίστα των διαθέσιμων ιδιοτήτων είναι μεγάλη.

4.11 Ερωτήσεις

1. Ο προτεινόμενος τρόπος εισαγωγής κανόνων CSS σε ένα έγγραφο HTML είναι:
    1. μέσω εξωτερικού αρχείου CSS.
    2. μέσω της ετικέτας <style> στο τμήμα <head> του αρχείου HTML.
    3. μέσω της ιδιότητας style σε επιμέρους στοιχεία HTML.
2. Ποια η ετικέτα HTML μέσω της οποίας συνδέουμε ένα εξωτερικό αρχείο CSS;
    1. <style>
    2. <link>
    3. <script>
    4. <css>
3. Κάθε κανόνας CSS αφορά την παρουσίαση ενός μόνο στοιχείου HTML.
    – Σωστό/Λάθος

4. Αν μια ιδιότητα ενός κανόνα CSS παίρνει πολλές τιμές, ποιος είναι ο χαρακτήρας που διαχωρίζει τις τιμές αυτές;
    1. ο κενός χαρακτήρας ” ”
    2. το κόμμα “,”
    3. το ερωτηματικό “;”
    4. η παύλα “-”
5. Έστω ο κανόνας p, li { color : red; }. Ποια είναι η σημασία του;
    1. Τα στοιχεία <li> που περιέχονται σε στοιχεία είναι κόκκινα.
    2. Τα στοιχεία <li> και τα στοιχεία είναι κόκκινα.
    3. Τα στοιχεία που περιέχονται σε στοιχεία <li> είναι κόκκινα.
    4. Τα στοιχεία που είναι άμεσα παιδιά των στοιχείων <li> είναι κόκκινα.
6. Έστω ο κανόνας p > a {color: yellow}. Να σημειώσετε σε ποιες περιπτώσεις η λέξη «παράδειγμα» εμφανίζεται στον χρήστη με χρώμα κίτρινο.
    1. <html>παράδειγμα</html>
    2. <html> παράδειγμα </html>
    3. <html> παράδειγμα
</html>
    4. <html>
παράδειγμα
</html>
7. Έστω ο κανόνας p a {color: green}. Να σημειώσετε σε ποιες περιπτώσεις η λέξη «παράδειγμα» εμφανίζεται στον χρήστη με χρώμα πράσινο.
    1. <html>παράδειγμα</html>
    2. <html> παράδειγμα </html>
    3. <html> παράδειγμα
</html>
    4. <html>
παράδειγμα
</html>
8. Έστω ο κανόνας h3::after{content:"*"; color:red;}. Ποιο το αποτέλεσμα;
    1. Όλα τα στοιχεία <h3> που ακολουθούνται από “*” εμφανίζονται κόκκινα.
    2. Όλα τα στοιχεία <h3> που περιέχουν έναν χαρακτήρα “*” εμφανίζονται κόκκινα.
    3. Όλα τα στοιχεία <h3> κοκκινίζουν (το “*” σημαίνει ανεξάρτητα του περιεχομένου τους).
    4. Όλα τα στοιχεία <h3> εμφανίζονται με ένα κόκκινο “*” στο τέλος.
9. Έστω ο κανόνας p[xyz^="1"] {color: green;}. Να σημειώσετε όλες τις περιπτώσεις στις οποίες η λέξη καλημέρα είναι πράσινη.
    1. καλημέρα
    2. καλημέρα
    3. καλημέρα
    4. <h2 xyz="1"> καλημέρα </h2>
10. Έστω ο κανόνας p, [xyz^="1"] {color: red;}. Να σημειώσετε όλες τις περιπτώσεις στις οποίες η λέξη καλημέρα είναι κόκκινη.
    1. καλημέρα
    2. καλημέρα
    3. καλημέρα
    4. <h2 xyz="1"> καλημέρα </h2>
11. Έστω ο κανόνας [xyz] {color: yellow;}. Να σημειώσετε όλες τις περιπτώσεις στις οποίες η λέξη καλημέρα είναι κίτρινη.
    1. καλημέρα
    2. καλημέρα
    3. καλημέρα
    4. <h2 xyz="2"> καλημέρα </h2>
12. Έστω ο κανόνας div [xyz] {color: red;}. Να σημειώσετε όλες τις περιπτώσεις στις οποίες η λέξη καλημέρα είναι κόκκινη.
    1. καλημέρα
    2. καλημέρα
    3. καλημέρα
    4. <h2 xyz="1"> καλημέρα </h2>
13. Έστω ο κανόνας div h2:hover {color: green;}. Ποια είναι τα στοιχεία του αρχείου τα οποία θα γίνουν πράσινα όταν το ποντίκι βρίσκεται πάνω τους; (σημειώστε όλα όσα ισχύουν)
    1. όλα τα στοιχεία και <h2>

2. όλα τα στοιχεία , <h2> και <hover>
3. όλα τα στοιχεία <h2> που είναι παιδιά ενός
4. όλα τα στοιχεία <h2> που είναι απόγονοι ενός
5. όλα τα στοιχεία που είναι απόγονοι ενός <h2>
6. όλα τα στοιχεία <h2>, επίσης όλα τα θα είναι πάντα πράσινα

14. Έστω ο κανόνας p:nth-child(even){color:green;}. Πόσα πράσινα ονόματα θα εμφανιστούν από τον παρακάτω κώδικα;

```
<div>
    <p>Maria</p>
    <p>Kostas</p>
    <p>Katerina</p>
</div>
```

15. Έστω οι παρακάτω κανόνες CSS:
    - p:nth-child(odd):hover{color:yellow;}
    - p:nth-child(even):hover{color:green;}

    Τι συνέπεια θα έχουν οι κανόνες αυτοί στον παρακάτω κώδικα;

```
<div>
    <p>Maria</p>
    <p>Kostas</p>
</div>
```

    1. Όταν το ποντίκι περνάει πάνω από τα ονόματα, αυτά γίνονται πότε κίτρινα, πότε πράσινα, εναλλάξ την 1η, 3η κ.λπ. φορά κίτρινα, την 2η, 4η κ.λπ. πράσινα.
    2. Καμιά συνέπεια, ο κανόνας έχει συντακτικό λάθος.
    3. Όταν το ποντίκι περνάει πάνω από τη Maria, το όνομα γίνεται κίτρινο και από τον Kostas πράσινο,
    4. Το χρώμα των ονομάτων είναι εξαρχής κίτρινο για τη Maria και πράσινο για τον Kostas. Το ποντίκι δεν επηρεάζει αυτή την εμφάνιση.

16. Έστω ο κανόνας CSS #mybox {padding: 5px 8px; margin: 3px 5px; border: solid 1px black;} και έστω ότι το στοιχείο αυτό περιέχει ένα αντικείμενο με διαστάσεις width: 3em, height: 2em, και το root element είναι γνωστό ότι έχει font-size 16px. Ποιο το πλάτος του στοιχείου #mybox σε πίξελ;
17. Έστω αρχείο HTML στο οποίο έχει οριστεί ο κανόνας CSS #mybox {box-sizing: border-box; padding: 5px; margin: 10px; height:50px; width:50px}. Ποιο το πλάτος του περιεχομένου του στοιχείου #mybox;
18. Έστω ο κανόνας {box-sizing: border-box; }. Ποια είναι η συνέπεια του παραπάνω κανόνα για τα στοιχεία που ακολουθούν το μοντέλο εγκιβωτισμού;
    1. Οι ιδιότητες width και height αφορούν τη συνολική διάσταση του στοιχείου, συμπεριλαμβανομένων του περιεχομένου, του πλαισίου και του εσωτερικού περιθωρίου.
    2. Οι ιδιότητες width και height αφορούν τη συνολική διάσταση του στοιχείου, συμπεριλαμβανομένου του περιεχομένου, του πλαισίου και του εσωτερικού και εξωτερικού περιθωρίου
    3. Οι ιδιότητες width και height αφορούν μόνο το περιεχόμενο του στοιχείου.
    4. Οι ιδιότητες width και height αφορούν το περιεχόμενο του στοιχείου και το εσωτερικό περιθώριο.
19. Αν ένα στοιχείο έχει την ιδιότητα padding 10px 20px; αυτό σημαίνει ότι:
    1. Το εσωτερικό περιθώριο είναι 10px και το εξωτερικό περιθώριο 20px.
    2. Το εσωτερικό περιθώριο είναι προς τα αριστερά και δεξιά 10px, ενώ προς τα πάνω και κάτω 20px.
    3. Το εσωτερικό περιθώριο είναι προς τα πάνω και κάτω 10px, ενώ προς τα αριστερά και δεξιά 20px
20. Έστω ο κανόνας #b {padding: 5px 0 8px 3px; margin: 3px 4px 0 2px;}. Ποιο είναι το ύψος του στοιχείου σε px αν το περιεχόμενό του είναι ύψους 20px;
21. Για ποιες από τις τιμές της ιδιότητας display μπορούμε να ορίσουμε την ιδιότητα margin-top ενός στοιχείου; (σημειώστε όλες όσες ταιριάζουν)

1. block
2. inline
3. inline-block

22. Ένα στοιχείο inline δεν μπορεί να περιέχει στοιχεία block.
    – Σωστό/Λάθος
23. Ποια η διαφορά μεταξύ em και rem;
    1. Είναι το ίδιο, και τα δύο αφορούν το μέγεθος του χαρακτήρα M.
    2. Το em αφορά το μέγεθος του χαρακτήρα M του γονικού στοιχείου, ενώ το rem το μέγεθος του ριζικού στοιχείου.
    3. Το em αφορά το μέγεθος του χαρακτήρα M, ενώ το rem του χαρακτήρα R.
24. Το χρώμα rgba(0, 0, 0, 0.8) είναι:
    1. Το λευκό χρώμα με διαφάνεια 80%, δηλαδή σχεδόν πλήρως διαφανές.
    2. Το μαύρο χρώμα με διαφάνεια 80%, δηλαδή σχεδόν πλήρως διαφανές.
    3. Το μαύρο χρώμα με διαφάνεια 80%, δηλαδή σχεδόν πλήρως αδιαφανές.
    4. Το λευκό χρώμα με διαφάνεια 80%, δηλαδή σχεδόν πλήρως αδιαφανές.
25. Οι παρακάτω δύο ιδιότητες είναι απολύτως ισοδύναμες:
    a. background-color: rgba(255,0,0,0.5);
    b. background-color: rgb(255,0,0); opacity: 0.5;
    – Σωστό/Λάθος
26. Έστω στοιχείο και έστω ότι ισχύουν οι εξής κανόνες για το έγγραφο αυτό:

```
div {color: red;}
#id1 {color: green;}
.cl1 {color: blue;}
```

Με τι χρώμα θα εμφανιστεί το περιεχόμενο του στοιχείου αυτού;

1. κόκκινο
2. πράσινο
3. κίτρινο
4. μπλε

4.12 Βιβλιογραφία και Αναφορές

Ο ιστότοπος mdn web docs αποτελεί μια καλή αρχή για εμβάθυνση στη γλώσσα CSS. Εκεί μπορεί κανείς να βρει αναλυτικούς οδηγούς για διάφορα τμήματα της γλώσσας CSS, όπως π.χ. πώς μορφοποιείται το κείμενο ή πώς εντοπίζονται και εξαλείφονται τα σφάλματα στη CSS. Επιπλέον, ο ιστότοπος παρέχει αναλυτική περιγραφή των ιδιοτήτων CSS, έχει δηλαδή τον ρόλο του σημείου αναφοράς για την τρέχουσα κατάσταση της γλώσσας. Για παράδειγμα, μπορεί κανείς να διαβάσει την εξαντλητική περιγραφή της ιδιότητας position.

Στο MDN θα βρει κανείς και τη σύνδεση με τα πρότυπα, τα κείμενα που περιγράφουν τη γλώσσα CSS με μεγάλη λεπτομέρεια και τα οποία είναι πολλές φορές υπό ανάπτυξη. Ο αναγνώστης που θέλει να εμβαθύνει στη CSS ενθαρρύνεται να ανατρέξει στα πρότυπα αυτά. Ωστόσο πρέπει να σημειώσουμε πως τα πρότυπα αυτά μπορεί να είναι για τον αρχάριο αναγνώστη, σε ορισμένα σημεία, δυσνόητα, καθώς περιέχουν λεπτομερείς οδηγίες που απευθύνονται και σε όσους αναπτύσσουν τις σχετικές δυνατότητες των φυλλομετρητών. Ένα καλό σημείο εκκίνησης είναι το τρέχον στιγμιότυπο της CSS (snapshot) που αναφέρεται σε όλα τα επιμέρους πρότυπα που συνθέτουν τη CSS σήμερα, όπως για παράδειγμα το CSS Box Model Level 3 ή το CSS Color Level 3. Το (ψηφιακό ή έντυπο) βιβλίο του Paul McFedries "Web Design Playground - HTML & CSS The Interactive Way", εκδόσεις Manning (2019), παρέχει μια καλή εισαγωγή βήμα προς βήμα στην HTML και τη CSS.

Μια πολύ καλή διαδικτυακή πηγή για προχωρημένα θέματα της CSS είναι η ιστοσελίδα CSS-Tricks, ενώ πιο απλούς οδηγούς μπορεί να βρει κανείς στο HTML & CSS is hard.

Τέλος, να επισημάνουμε πως οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Τα σχετικά μαθήματα για τα κεφάλαια 1-6 αυτού του βιβλίου είναι το «Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, Javascript» και το «Προχωρημένα θέματα ανάπτυξης ιστοσελίδων».

A. Ξενόγλωσσες

Atkins Jr. T, Rivoal, F., & Etemad E. (2021). *CSS Snapshot 2021* (White paper). W3C.

McFedries, P. (2019). *Web Design Playground - HTML & CSS The Interactive Way*. Manning.

Verou, L. (2015). *CSS Secrets*. O'Reilly Media, Inc.

B. Ελληνόγλωσσες

Αβούρης, Ν. (2018). Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, JavaScript. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr.

Αβούρης, Ν., & Σιντόρης, Χ. (2020). Προχωρημένα θέματα ανάπτυξης ιστοσελίδων. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr

Δουληγέρης, Χ., Μαυροπόδη, Ρ., Κοπανάκη, Ε., & Καραλής, Α. (2021). *Τεχνολογίες και Προγραμματισμός στον Παγκόσμιο Ιστό* (2η έκδ.). Εκδόσεις Νέων Τεχνολογιών. Κωδικός βιβλίου στον Εύδοξο: 102125023.

Kyrnin, J., & Meloni, J. (2021). *Sams Teach Yourself HTML5, CSS and Javascript* (3rd ed.). Εκδόσεις Γκιούρδας.

Lemay, L., Coburn, R., & Kyrnin, J. (2016). *Sams Teach Yourself HTML, CSS & JavaScript* (7th ed). Εκδόσεις Γκιούρδας. Κωδικός Βιβλίου στον Εύδοξο: 59357307.

Μαρκατσέλας, Μ., & Ξαρχάκος, Κ. (2013). *CSS3 & Εφαρμογές*. Εκδόσεις Άβακας, Αθήνα. Κωδικός βιβλίου στον Εύδοξο: 68369452.

Κεφάλαιο 5. Διάταξη περιεχομένου με τη CSS

Σύνοψη

Το κεφάλαιο αυτό, δεύτερο για τη CSS μετά το κεφάλαιο 4, εισάγει τους μηχανισμούς που διαθέτει η γλώσσα CSS για τη διάταξη περιεχομένου. Αρχικά συζητιούνται δύο από τους πιο παλιούς μηχανισμούς της γλώσσας, η τοποθέτηση με την ιδιότητα position και ο μηχανισμός των float. Η ιδιότητα position μάς επιτρέπει να τροποποιήσουμε τον προκαθορισμένο μηχανισμό τοποθέτησης στη CSS, την κανονική ροή, και να τοποθετήσουμε ένα στοιχείο σε διαφορετική θέση. Η ιδιότητα float δίνει τη δυνατότητα να κατασκευάσουμε απλές διατάξεις όπου κάποια στοιχεία στοιχίζονται στα δεξιά ή αριστερά του κυρίως κειμένου, ακολουθώντας την αισθητική που συνηθίζεται σε έντυπα άρθρα εφημερίδων ή περιοδικών. Με αυτούς τους δύο μηχανισμούς παρουσιάζεται και η έννοια του "stacking context", του μηχανισμού με τον οποίο τα στοιχεία τοποθετούνται σε έναν νοητό άξονα και στοιβάζονται το ένα μπροστά από το άλλο, ξεκινώντας από την επιφάνεια προβολής με κατεύθυνση τον χρήστη.

Στη συνέχεια, παρουσιάζονται οι δύο πιο βασικοί μηχανισμοί της σύγχρονης CSS για τη διάταξη περιεχομένου, του Flexbox και του CSS Grid. Οι μηχανισμοί αυτοί της CSS μάς δίνουν για πρώτη φορά μεγάλο βαθμό ελευθερίας στη στοίχιση των στοιχείων των σελίδων μας και μάλιστα με προβλέψιμη συμπεριφορά, ανάλογα με τον διαθέσιμο χώρο στην επιφάνεια προβολής.

Προαπαιτούμενη γνώση

Για την κατανόηση αυτού του κεφαλαίου είναι απαραίτητη η εξοικείωση με το υλικό του πρώτου μέρους σχετικά με τη CSS (κεφάλαιο 4), καθώς και καλή γνώση της HTML που παρουσιάζεται στα προηγούμενα κεφάλαια (2 και 3).

5.1 Διάταξη περιεχομένου

Η διάταξη του περιεχομένου των ιστοσελίδων γίνεται σύμφωνα με την κανονική ροή (Ενότητα 4.7.1), στην οποία τα στοιχεία μπλοκ τοποθετούνται το ένα μετά το άλλο, από πάνω προς τα κάτω, ενώ τα στοιχεία inline που βρίσκονται μέσα στα στοιχεία μπλοκ τοποθετούνται από τα δεξιά προς τα αριστερά. Αυτός ο μηχανισμός είναι επαρκής για να διατάξουμε το περιεχόμενο με πολύ απλό τρόπο.

Αρκετά νωρίς στην ιστορία του παγκόσμιου ιστού προέκυψε η επιθυμία και η ανάγκη να μπορούμε να τοποθετήσουμε τα στοιχεία μας με πιο ευφάνταστο ή πολύπλοκο τρόπο, για παράδειγμα να τοποθετήσουμε δύο ή περισσότερα στοιχεία μπλοκ το ένα δίπλα στο άλλο ώστε να δημιουργήσουμε στήλες ή ακόμα και πλέγματα. Καθώς αρχικά η HTML και η CSS δεν είχαν αντίστοιχους μηχανισμούς, η πρώτη λύση ήταν η χρήση των πινάκων, του στοιχείου <table> δηλαδή, για τον έλεγχο της διάταξης. Ωστόσο, η προσέγγιση αυτή έχει πολύ σοβαρά προβλήματα και γρήγορα εγκαταλείφτηκε, όπως ήδη αναφέρθηκε στο κεφάλαιο 2.

Τη θέση των πινάκων κατέλαβε σύντομα η χρήση της ιδιότητας float, που επέτρεψε τη δημιουργία σύνθετων διατάξεων σε στήλες. Ωστόσο, καθώς ούτε και αυτή η ιδιότητα είχε σχεδιαστεί για να υποστηρίζει πολύπλοκες διατάξεις, η χρήση της έφτασε γρήγορα σε όρια, όπου πολύπλοκες ιστοσελίδες ήταν δύσκολο να συντηρηθούν και να εξασφαλιστεί πως θα εμφανίζονταν σωστά και προβλέψιμα σε διαφορετικούς φυλλομετρητές και σε διάφορα εύρη οθονών. Η κατάσταση αυτή βελτιώθηκε δραματικά τα τελευταία χρόνια με την έλευση των προτύπων της CSS για το Flexbox και το Grid, που υποστηρίζονται πολύ καλά από τους σύγχρονους φυλλομετρητές.

Θα ξεκινήσουμε όμως την ενότητα αυτή με την παρουσίαση της ιδιότητας position, που επιτρέπει την τοποθέτηση των στοιχείων σε διαφορετική θέση σε σχέση με τη θέση που θα τοποθετούνταν στην κανονική ροή. Στη συνέχεια θα δούμε την έννοια της στοίβας (stacking context) και έπειτα την ιδιότητα float, χωρίς όμως να εμβαθύνουμε σε πολύπλοκες διατάξεις με αυτή, καθώς θα παρουσιάσουμε τους σύγχρονους μηχανισμούς διάταξης Flexbox και Grid.

5.2 Η ιδιότητα position

Η ιδιότητα position είναι ένας τρόπος που μας δίνει η CSS από το επίπεδο 2 για να διατάξουμε τα στοιχεία μας έτσι ώστε να τοποθετούνται σε διαφορετική θέση από αυτή στην οποία θα τοποθετούνταν στην κανονική ροή.

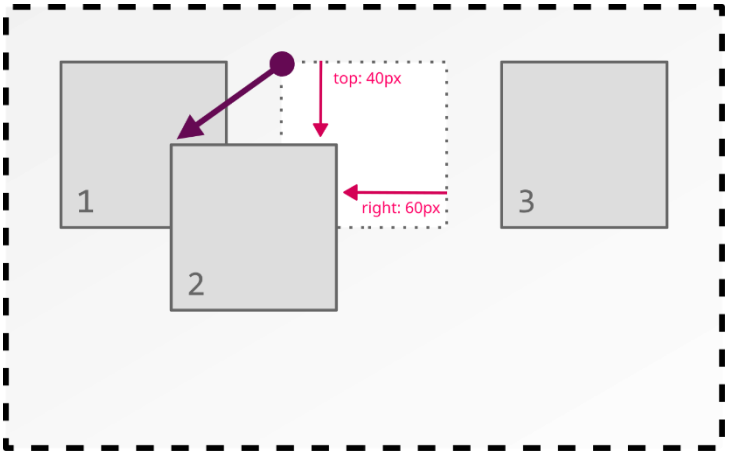
Με την ιδιότητα position μπορούμε να τοποθετήσουμε στοιχεία σε οποιαδήποτε θέση στη σελίδα. Έχουμε στη διάθεσή μας πέντε τρόπους τοποθέτησης, τη στατική, τη δυναμική, την απόλυτη, τη σταθερή τοποθέτηση, καθώς και την τοποθέτηση sticky.

5.2.1 Στατική τοποθέτηση (static positioning)

Όπως είδαμε, στην κανονική ροή (ενότητα 4.7.1) η CSS τοποθετεί τα στοιχεία το ένα μετά το άλλο, ώστε να μην υπερκαλύπτονται, με αυτό που ονομάζουμε **στατική τοποθέτηση**. Η στατική τοποθέτηση δεν είναι τίποτα άλλο παρά η κανονική, συνήθης τοποθέτηση με την οποία τα στοιχεία τοποθετούνται το ένα μετά το άλλο και καταλαμβάνουν τον χώρο που χρειάζονται για να προβληθούν πριν εφαρμόσουμε τις δικές μας οδηγίες CSS. Η στατική τοποθέτηση ενεργοποιείται με τη δήλωση position: static. Ουσιαστικά, σε αυτά τα στοιχεία δεν έχουμε εφαρμόσει κάποια αλλαγή, αφού η static είναι η προκαθορισμένη τιμή της ιδιότητας position.

5.2.2 Σχετική τοποθέτηση (relative positioning)

Με την ιδιότητα position: relative δεσμεύεται ο χώρος που ούτως ή άλλως θα δεσμευόταν για να εμφανιστεί το στοιχείο, αλλά επιπλέον μπορούμε να τον μετατοπίσουμε, χωρίς να αλλάξει η θέση των υπόλοιπων στοιχείων. Αυτό είναι σημαντικό: Η θέση των υπόλοιπων στοιχείων δεν αλλάζει και θα μείνουν εκεί που θα ήταν σαν το στοιχείο που μετατοπίζουμε να ήταν static.

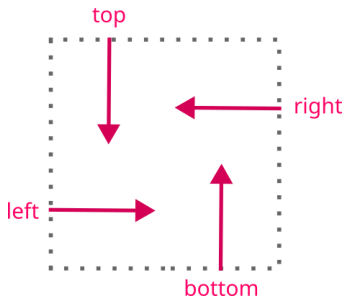


Εικόνα 5.1 *Σχετική τοποθέτηση.*

Με τις ιδιότητες top, right, bottom, left ορίζουμε μια νέα θέση για το στοιχείο, μετακινώντας το **από την εκάστοτε πλευρά** (***Εικόνα 5.2***). Για παράδειγμα, η οδηγία

```
#repositioned {
    position: relative;
    top: 40px;
    right: 60px;
}
```

θα μετακινήσει το στοιχείο #repositioned κατά 40px προς τα κάτω, γιατί θα το σπρώξει από την πάνω πλευρά και 60px προς τα αριστερά γιατί θα το σπρώξει από τη δεξιά πλευρά. Τα επόμενα στοιχεία δεν αλλάζουν θέση και το στοιχείο #repositioned, που είναι relative, εμφανίζεται πιο κοντά στον χρήστη από ό,τι το προηγούμενο στοιχείο, δηλαδή το στοιχείο #repositioned θα υπερκαλύπτει το προηγούμενό του στοιχείο.



Εικόνα 5.2 *Με τις ιδιότητες top, right, bottom, left ορίζουμε μια νέα θέση για το στοιχείο, μετακινώντας το από την εκάστοτε πλευρά. Μπορούμε να χρησιμοποιήσουμε και τις inset-block-start, inset-block-end για μετατόπιση στην κατεύθυνση block ή τις inset-inline-start, inset-inline-end για μετατόπιση στην κατεύθυνση inline.*

Αντί για τις ιδιότητες top, right, bottom, left, μπορούμε να χρησιμοποιήσουμε και τις inset-block-start, inset-block-end για μετατόπιση στην κατεύθυνση block ή τις inset-inline-start, inset-inline-end για μετατόπιση στην κατεύθυνση inline. Αυτές οι τέσσερις δηλώσεις αποτελούν και ένα καλό παράδειγμα του τρόπου με τον οποίο εξελίσσεται η CSS. Οι τέσσερις αυτές ιδιότητες προδιαγράφονται στο CSS Logical Properties and Values Level 1 και έχει ήδη υλοποιηθεί από όλους τους μεγάλους φυλλομετρητές.

Συνοψίζοντας, η ιδιότητα position με την τιμή relative:

- Μετακινεί το στοιχείο σε σχέση με τη θέση που θα είχε αν δεν επεμβαίναμε.
- Δεν επηρεάζεται η θέση κανενός άλλου στοιχείου.
- Η μετακίνηση γίνεται προσδιορίζοντας ποια πλευρά θέλουμε να μετακινήσουμε.
- Ο υπολογισμός γίνεται σε σχέση με τη θέση του εξωτερικού περιθωρίου (margin).

Αν έχουμε δηλώσει αντιφατικές τιμές, π.χ. left: 10px; right: 10px, τότε θα ισχύσει το left, γιατί στην κανονική ροή η κατεύθυνση inline στο δικό μας σύστημα γραφής είναι από αριστερά προς τα δεξιά. Αντίστοιχα και με τα top/bottom, γιατί η κατεύθυνση block είναι από πάνω προς τα κάτω.

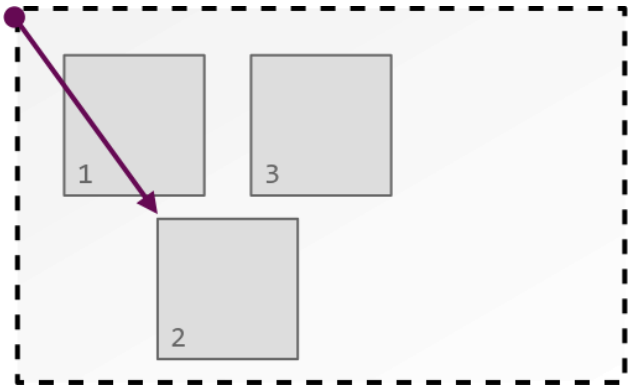
Επιπλέον, τα στοιχεία που είναι τοποθετημένα relative δημιουργούν δικό τους stacking context, που περιγράφεται στην ενότητα 5.3.

5.2.3 Sticky positioning

Το στοιχείο τοποθετείται αρχικά όπως και με την position: relative. Η διαφορά είναι πως η θέση προσδιορίζεται με βάση το παράθυρο του scroll. Το sticky συμπεριφέρεται σαν relative, δηλαδή τοποθετείται σχετικά σε σχέση με τον πρόγονό του. Όταν ο πρόγονος κυλήσει (scroll) πέρα από μια συγκεκριμένη θέση, το στοιχείο sticky δεν θα ακολουθήσει την κύλιση, αλλά θα σταθεροποιηθεί στην τελευταία θέση, θα συμπεριφερθεί δηλαδή σαν στοιχείο fixed.

5.2.4 Absolute positioning

Αντίθετα με τη σχετική τοποθέτηση, η δήλωση position: absolute αφαιρεί το στοιχείο από την κανονική ροή τοποθέτησης. Αυτό σημαίνει πως δεν θα δεσμευτεί χώρος για το στοιχείο που τοποθετείται απόλυτα.



Εικόνα 5.3 *Απόλυτη τοποθέτηση.*

Οι ιδιότητες top, right, bottom, left προσδιορίζουν τη μετατόπιση της θέσης του στοιχείου σε σχέση με το στοιχείο που το περικλείει, δηλαδή σε σχέση με **το πιο κοντινό πρόγονο στοιχείο που δεν έχει** position: static. Το στοιχείο τοποθετείται σε σχέση **με την εκάστοτε πλευρά** του πρόγονου στοιχείου. Για παράδειγμα, η οδηγία

```
#absolute-positioned {
    position: absolute;
    bottom: 16px;
}
```

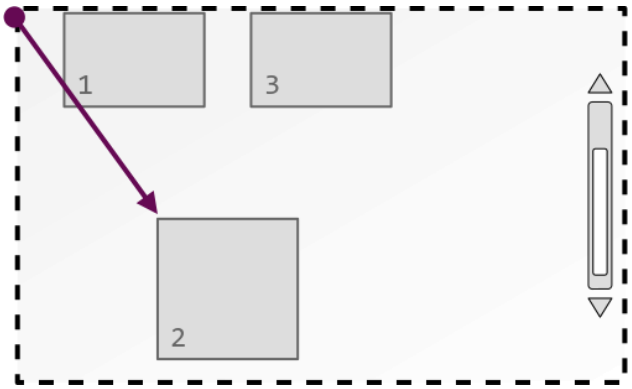
θα μετακινήσει το στοιχείο #absolute-positioned κατά 16px προς τα πάνω από την κάτω πλευρά του στοιχείου που το περικλείει.

Συνοψίζοντας, με την τιμή absolute:

- Το στοιχείο τοποθετείται σε σχέση με το πρόγονο μπλοκ.
- Το στοιχείο δεν συμμετέχει στην κανονική ροή, δεν δεσμεύεται χώρος για αυτό.
- Η σύμπτυξη περιθωρίου (margin collapse) (Ενότητα 4.7.1) δεν ισχύει για στοιχεία με position: absolute.

5.2.5 Fixed positioning

Η διαφορά της δήλωσης position: fixed από την absolute είναι πως η τοποθέτηση του στοιχείου γίνεται σε σχέση με το παράθυρο προβολής (viewport) του φυλλομετρητή, δηλαδή την περιοχή που είναι ορατή μέσα στο παράθυρο του φυλλομετρητή. Η κύλιση του εγγράφου δεν επηρεάζει καθόλου τη θέση προβολής του στοιχείου. Με αυτό τον τρόπο, μπορούμε να φτιάξουμε, για παράδειγμα, μενού πλοήγησης που να είναι πάντα στην ίδια θέση ή να δείξουμε σημαντικές (ή ενοχλητικές) πληροφορίες. Όπως και με την απόλυτη τοποθέτηση, ούτε εδώ το στοιχείο συμμετέχει στην κανονική ροή.



Εικόνα 5.4 *Fixed positioning.*

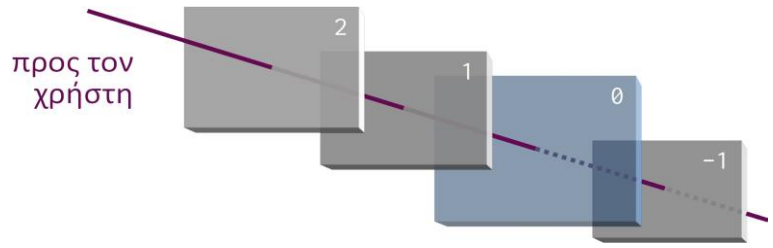
5.3 Το stacking context και το z-index

Σύμφωνα με τους κανόνες της κανονικής ροής (Ενότητα 4.7.1), ο φυλλομετρητής τοποθετεί τα στοιχεία το ένα μετά το άλλο. Αν όμως κάποια στοιχεία επικαλύπτονται, πρέπει να αποφασιστεί ποιο στοιχείο θα τοποθετηθεί πιο μπροστά, δηλαδή πιο κοντά στον χρήστη. Επικάλυψη, για παράδειγμα, μπορεί να συμβεί με τη σχετική τοποθέτηση (ενότητα 5.2.2).

Τα στοιχεία που αλληλοεπικαλύπτονται σχηματίζουν μια **στοίβα**, ένα **stack**. Τα στοιχεία αυτής της στοίβας που είναι πιο κοντά μας μπορούν να κρύψουν τα στοιχεία που βρίσκονται πιο πίσω. Το ποιο στοιχείο βρίσκεται πιο μπροστά καθορίζεται από μια σειρά κανόνων, ο πιο βασικός από τους οποίους είναι πως στοιχεία που βρίσκονται πιο βαθιά στο DOM εμφανίζονται πιο μπροστά, πιο κοντά στον χρήστη.

Μια στοίβα στοιχείων, ένα **stacking context**, είναι μια ομάδα στοιχείων που έχουν έναν *κοινό πρόγονο*. Αυτός ο πρόγονος είναι η ρίζα της στοίβας και τα στοιχεία τοποθετούνται με σειρά μέσα σε αυτή τη στοίβα. Το στοιχείο <html> είναι η ρίζα όλων των στοιχείων και συνεπώς σχηματίζει μια στοίβα με τα στοιχεία που περιέχει.

Μπορούμε να χρησιμοποιήσουμε την ιδιότητα `z-index` για να ορίσουμε τη θέση κάποιων στοιχείων στη στοίβα σε σχέση τα υπόλοιπα στοιχεία. Μπορούμε να θεωρήσουμε μια φανταστική γραμμή, έναν νοητό άξονα, τον **άξονα z**, που ξεκινά από τον παρατηρητή μιας σελίδας, κατευθύνεται προς την επιφάνεια του φυλλομετρητή και συνεχίζει προς τα πίσω. Τα στοιχεία που είναι πιο κοντά στον χρήστη εμφανίζονται πιο μπροστά από τα στοιχεία που είναι πιο μακριά από τον χρήστη.



Εικόνα 5.5 *Άξονας z. Η τιμή του z-index καθορίζει τη σειρά των στοιχείων πάνω στον άξονα z και συνεπώς το ποιο στοιχείο θα εμφανίζεται πιο κοντά.*

Ένα στοιχείο που μπορούμε να μετακινήσουμε πάνω στον άξονα z είναι με τη σειρά του ρίζα **για τη δική του στοίβα στοιχείων**. Οπότε, αλλάζοντας την τιμή της z-index ενός παιδιού του <html>, μετακινούμε προς τα μπροστά ή προς τα πίσω όχι μόνο αυτό το στοιχείο, αλλά μια ολόκληρη υπο-στοίβα στοιχείων, που αποτελείται από τους απογόνους του και που έχουν τη δική τους διάταξη στον δικό τους άξονα z.

Δεν μπορούμε να αλλάξουμε τη θέση όλων των στοιχείων του DOM με τη z-index, αλλά μόνο όσων *έχουν τοποθετηθεί*. Για ένα στοιχείο λέμε πως *έχει τοποθετηθεί* όταν έχει αλλάξει η θέση του με την ιδιότητα position. Μπορούμε να το δούμε αυτό εξετάζοντας τη σειρά με την οποία ο φυλλομετρητής στοιβάζει τα στοιχεία από το πιο μπροστά (προς τον χρήστη) στο πιο πίσω (μακριά από τον χρήστη):

- Τοποθετημένα στοιχεία για τα οποία έχουμε δηλώσει z-index >=0, όπου μεγαλύτερη τιμή σημαίνει πως το στοιχείο βρίσκεται πιο μπροστά,
- στοιχεία inline που δεν έχουν τοποθετηθεί,
- στοιχεία float που δεν έχουν τοποθετηθεί,
- στοιχεία block που δεν έχουν τοποθετηθεί,
- τοποθετημένα στοιχεία με αρνητικό z-index,
- το στοιχείο :root, που υπάρχει πάντα και περιέχει όλα τα στοιχεία της σελίδας.

Για παράδειγμα, όταν για ένα στοιχείο οριστεί μια τιμή για το z-index, τότε σχηματίζεται μια **νέα στοίβα στοιχείων** (stacking context) για ***τα απόγονα στοιχεία*** του . Δηλαδή, οι απόγονοι του τοποθετούνται σε έναν άξονα z που ανήκει στο . Όλη αυτή η στοίβα στοιχείων τοποθετείται με τη σειρά της στον άξονα z της στοίβας στην οποία ανήκει και το .

Βέβαια, δεν είναι μόνο το z-index που μπορεί να προκαλέσει τον σχηματισμό μιας νέας στοίβας στοιχείων. Οι κανόνες είναι πολλοί, αλλά οι πιο σημαντικοί τρόποι δημιουργίας στοίβας στοιχείων είναι:

- Το στοιχείο :root
- τοποθετημένο στοιχείο:
  - με absolute ή relative positioning και z-index με αριθμητική τιμή,

– με fixed ή sticky positioning.

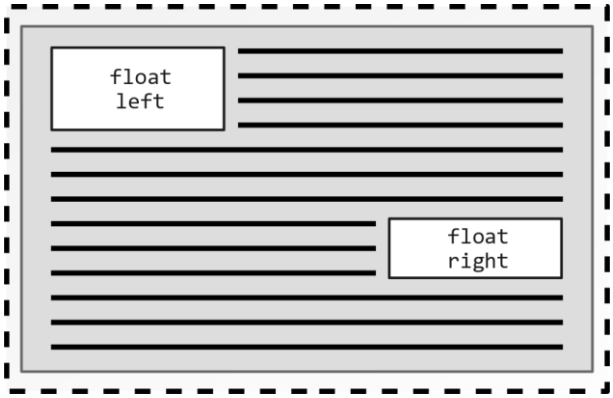
5.4 Float

Μέχρι τώρα έχουμε δει πώς να χρησιμοποιούμε την κανονική ροή για να τοποθετούμε στοιχεία από πάνω προς τα κάτω, το ένα κάτω από το άλλο και πώς να αλλάζουμε τη θέση τους με την τοποθέτηση (Ενότητα 5.2).

Με την ιδιότητα float έχουμε έναν επιπλέον μηχανισμό για να μετακινήσουμε ένα αντικείμενο ώστε να τοποθετηθεί στην αριστερή ή τη δεξιά πλευρά της σελίδας ή, καλύτερα, του υποδοχέα (κοντέινερ) που το περιέχει και να δημιουργήσουμε πιο σύνθετες διατάξεις, που να μοιάζουν με έντυπα άρθρα περιοδικών όπου το κείμενο πλαισιώνεται με εικόνες. Αυτός ήταν και ο αρχικός στόχος των floats και όχι η δημιουργία πολύπλοκων διατάξεων. Καθώς όμως δεν υπήρχαν εναλλακτικές, οι σχεδιαστές ιστοσελίδων χρησιμοποίησαν τον μηχανισμό των floats για να κατασκευάσουν πολύπλοκα πλέγματα (grid), όπως για παράδειγμα συνέβαινε με το δημοφιλές Bootstrap μέχρι την έκδοση 3.

Σήμερα που έχουμε στη διάθεσή μας καλύτερους μηχανισμούς διάταξης, όπως το Flexbox ή το Grid (Ενότητα 5.5), τα floats δεν χρησιμοποιούνται πια για τη δημιουργία πολύπλοκων διατάξεων. Είναι ωστόσο χρήσιμα όταν θέλουμε να τοποθετήσουμε ένα στοιχείο μπλοκ στη μια ή την άλλη πλευρά.

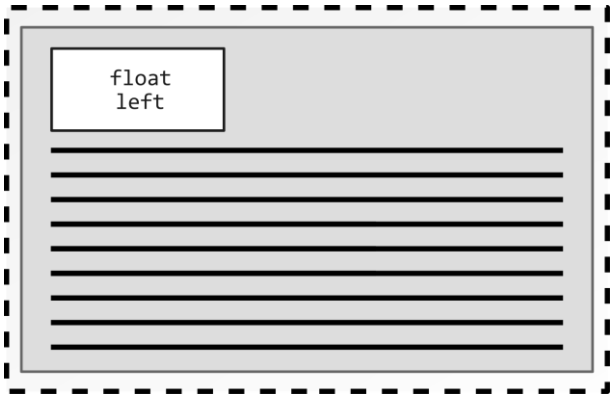
Ένα στοιχείο που είναι float θεωρείται πως συμμετέχει στην κανονική ροή όπως ήταν πριν ενεργοποιηθεί το float. Το περιεχόμενο που ακολουθεί το στοιχείο θα τυλιχτεί γύρω από το στοιχείο αυτό όπως φαίνεται στην ***Εικόνα 5.6***.



Εικόνα 5.6 *Τα δύο λευκά κουτιά έχουν γίνει float.*

Η ιδιότητα float μπορεί να έχει τιμή left, right ή none, που είναι και η προκαθορισμένη τιμή.

Μπορούμε όμως να ζητήσουμε να μην υπάρχει περιεχόμενο στην πλευρά του στοιχείου που είναι float. Δηλαδή, να μετακινηθεί μεν το float στην άκρη, αλλά το περιεχόμενο που το ακολουθεί να μην ανέβει προς τα πάνω και τυλιχτεί γύρω από το float (***Εικόνα 5.7***). Αυτό το πετυχαίνουμε με την ιδιότητα clear, η οποία μπορεί να πάρει με τη σειρά της τις τιμές left, right, both ή none, που είναι και η προκαθορισμένη τιμή.

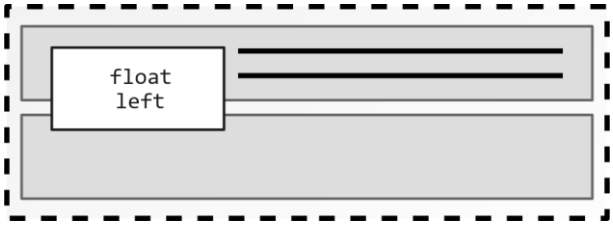


Εικόνα 5.7 *Με την ιδιότητα clear: right στο float, η περιοχή στα δεξιά του θα μείνει ελεύθερη από περιεχόμενο.*

Ένα σημαντικό σημείο είναι πως ένα στοιχείο που είναι float δεν συνεισφέρει στον υπολογισμό του ύψους του κοντέινέρ του. Ο υποδοχέας αυτός παίρνει το ύψος που χρειάζεται για να δείξει το περιεχόμενό του (χωρίς να υπολογίζεται πια το float) και το float μπορεί να εξέχει προς τα κάτω αν είναι πιο ψηλό από το υπόλοιπο περιεχόμενο (**Εικόνα *5.8***).

Εικόνα 5.8 *Το στοιχείο που είναι float (λευκό κουτί) δεν συμμετέχει στον υπολογισμό του ύψους του στοιχείου που το περιέχει.*

Μια παρενέργεια είναι να ζωγραφίζεται το float πάνω από το περιεχόμενο που ακολουθεί (***Εικόνα 5.9***). Η λύση αυτού του προβλήματος παρουσιάζεται στην αμέσως επόμενη ενότητα.



Εικόνα 5.9 *Όταν το ύψος του στοιχείου μέσα στο οποίο περιέχεται το float είναι μικρότερο από το ύψος του float, το float θα επικαλύψει τα επόμενα στοιχεία.*

5.4.1 Flow-root και Block formatting context

Όταν το ύψος του στοιχείου μέσα στο οποίο περιέχεται το float είναι μικρότερο από το ύψος του float, το float θα επικαλύψει τα επόμενα στοιχεία (***Εικόνα 5.9***). Αυτό το πρόβλημα παλιότερα λυνόταν πιο δύσκολα, με μια λύση που είναι γνωστή ως clearfix. Σήμερα λύνεται πιο εύκολα, δίνοντας την τιμή flow-root στην ιδιότητα display. Ορίζοντας λοιπόν display: flow-root στο στοιχείο-υποδοχέα που περιέχει το float, η διάταξη των παιδιών του γίνεται πλέον σε σχέση με αυτό τον υποδοχέα. Το αποτέλεσμα είναι πως ο υποδοχέας flow-root επεκτείνεται για να περικλείσει όλα του τα παιδιά, ακόμη και αυτά που είναι float. Λέμε λοιπόν πως το στοιχείο με τη δήλωση display: flow-root δημιουργεί ένα *block formatting context*, μια περιοχή διάταξης κανονικής ροής. Χωρίς αυτή τη δήλωση, τον ρόλο του block formatting context τον παίζει το στοιχείο :root.

Επίσης, νέο block formatting context δημιουργούν όσα στοιχεία είναι float. Αυτό σημαίνει πως τα παιδιά ενός float θα διαταχθούν σε σχέση με το float αυτό.

5.5 CSS Grid και Flexbox

Από τα πρώτα χρόνια της χρήσης της CSS φάνηκε πως οι προγραμματιστές ιστοσελίδων χρειάζονταν δυνατότητες που δεν τους παρείχε η γλώσσα, κυρίως για τη δημιουργία σύνθετων διατάξεων με ευκολία και ακρίβεια, καθώς και την προσαρμογή των ιστοσελίδων σε διαφορετικές διαστάσεις οθόνης. Τα προβλήματα αυτά λύνονταν με πολύ κόπο και όχι πάντα ικανοποιητικά με τις τότε υπάρχουσες δυνατότητες της CSS. Η τοποθέτηση στοιχείων με την κανονική ροή μπορεί να μην είναι ικανοποιητική και ακόμη και η χρήση τεχνικών όπως το position ή τα floats μπορεί να μη μας βοηθήσουν ή να μας οδηγήσουν σε αποτέλεσμα που δεν συμπεριφέρεται πάντα όπως θέλουμε. Το Flexbox και το CSS Grid είναι δύο δυνατότητες που προστέθηκαν στη CSS σχετικά πρόσφατα, την τελευταία δεκαετία, και δημιουργήθηκαν για να λύσουν αυτού του είδους τα προβλήματα.

Καθώς σχεδιάστηκαν το ένα μετά το άλλο, σε κοντινό διάστημα, οι προδιαγραφές για τα δύο αυτά συστήματα έχουν κάποια κοινά στοιχεία που μπορούν να βοηθήσουν στη γρηγορότερη εξοικείωση. Η βασική διαφορά μεταξύ αυτών των δύο είναι πως το Flexbox είναι σχεδιασμένο για τη διάταξη σε μία διάσταση, ενώ το CSS Grid για τη διάταξη σε δύο διαστάσεις.

Με λίγα λόγια, αν θέλουμε να διατάξουμε στοιχεία σε μία γραμμή, όπως π.χ. τα στοιχεία μιας οριζόντιας μπάρας πλοήγησης ή μια στήλη με κάποιες χαρακτηριστικές εικόνες που θα συνοδεύουν το άρθρο μας, θα **προτιμήσουμε το Flexbox**. Αν ο στόχος μας είναι η διάταξη σε δύο διαστάσεις, σε γραμμές και στήλες, όπως μπορεί να είναι η διάταξη μιας ολόκληρης σελίδας, θα κάνουμε τη ζωή μιας πιο εύκολη αν **προτιμήσουμε το CSS Grid**. Μπορούμε βέβαια να τα αναμείξουμε και να έχουμε κάποια τμήματα με Flexbox και κάποια άλλα με Grid, ώστε να εκμεταλλευτούμε τα δυνατά σημεία κάθε τεχνικής.

5.6 Flexbox

Το Flexbox είναι ένας πολύ ευέλικτος μηχανισμός και λύνει με απλό τρόπο πολλά από τα προβλήματα που μας ταλάνιζαν πριν την εμφάνισή του. Η βασική ιδέα είναι πως τα στοιχεία ενός υποδοχέα Flexbox είναι εύκαμπτα κουτιά, **flexible boxes**.

Για να χρησιμοποιήσουμε το Flexbox χρειάζεται να επιλέξουμε ένα στοιχείο που θα περιέχει τα εύκαμπτα στοιχεία, τον υποδοχέα flex (**flex container**). Αυτό το κάνουμε ορίζοντας την τιμή display : flex σε κάποιο στοιχείο, όπως για παράδειγμα ένα , το <body> ή οτιδήποτε άλλο χρησιμοποιούμε και στην κανονική ροή για να περιέχει άλλα στοιχεία. Τα παιδιά του υποδοχέα flex ονομάζονται στοιχεία flex, **flex items** και διατάσσονται με τους κανόνες του Flexbox.

Για να τροποποιήσουμε τις ιδιότητες της διάταξής μας μπορούμε να επέμβουμε σε δύο σημεία: στον **υποδοχέα flex** και στα **στοιχεία flex**.

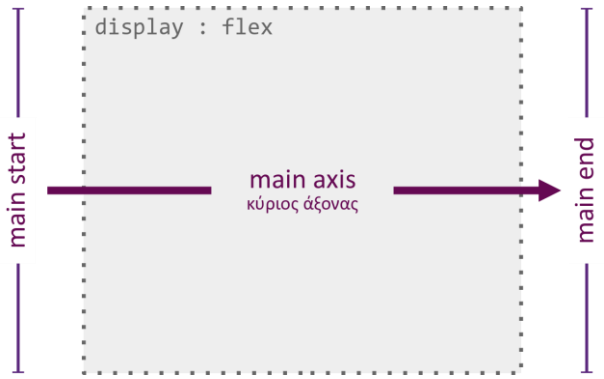
5.6.1 Ιδιότητες του υποδοχέα flex

Πριν δούμε τις ιδιότητες που αφορούν τον υποδοχέα flex, είναι χρήσιμο να εξοικειωθούμε με τις βασικές έννοιες των αξόνων στο Flexbox.

Άξονες flex

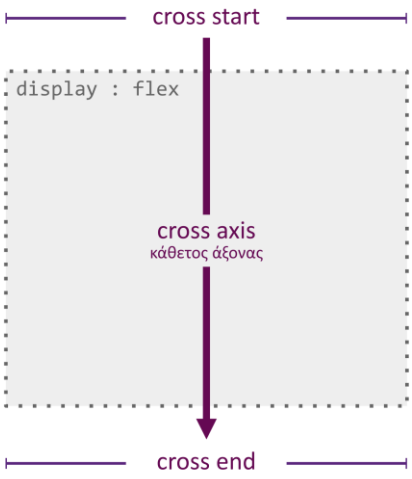
Σε έναν υποδοχέα flex τα στοιχεία τοποθετούνται σε δύο άξονες, τον **κύριο άξονα** και τον **κάθετο άξονα** (**main** και **cross axis**). Οι κατευθύνσεις των αξόνων εξαρτώνται από τον τρόπο γραφής (writing mode), που στα ελληνικά και στις δυτικές γλώσσες είναι από τα αριστερά προς τα δεξιά και από πάνω προς τα κάτω.

Ο κύριος άξονας αρχίζει από την πλευρά **main-start** και καταλήγει στη **main-end**. Αν δεν αλλάξουμε κάτι στον τρόπο γραφής, η προκαθορισμένη κατεύθυνση του κύριου άξονα συμπίπτει με την κατεύθυνση inline (**Εικόνα *5.10***).



Εικόνα 5.10 *Ο κύριος άξονας του υποδοχέα flex.*

Αντίστοιχα, ο κάθετος άξονας αρχίζει και τελειώνει στις πλευρές **cross-start**/**cross-end**. Αν δεν αλλάξουμε τις προκαθορισμένες τιμές, η τοποθέτηση στον κάθετο άξονα συμπίπτει με την τοποθέτηση των στοιχείων block (**Εικόνα *5.11***).

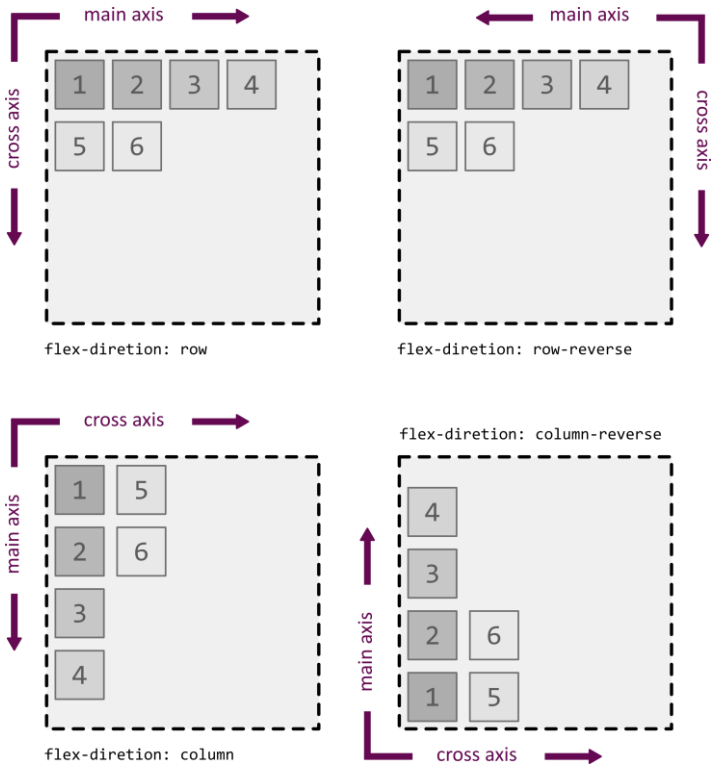


Εικόνα 5.11 *Ο κάθετος άξονας του υποδοχέα flex.*

Flex-direction

Τα στοιχεία flex διατάσσονται πάνω στον κύριο άξονά του, τον main axis. Αν δεν επέμβουμε, η τοποθέτηση γίνεται από τα αριστερά προς τα δεξιά, όπως και με τα στοιχεία inline στην κανονική ροή. Με την ιδιότητα flex-direction μπορούμε όμως να αλλάξουμε την κατεύθυνση των αξόνων του υποδοχέα flex (***Εικόνα 5.12***) και συνεπώς την κατεύθυνση με την οποία τοποθετούνται τα στοιχεία flex.

- **flex-direction: row** Η προκαθορισμένη τιμή. Ο κύριος άξονας είναι οριζόντια, από τα αριστερά προς τα δεξιά.
- **flex-direction: row-reverse** Ο κύριος άξονας είναι οριζόντια, από τα δεξιά προς τα αριστερά.
- **flex-direction: column** Ο κύριος άξονας είναι κάθετα, από πάνω προς τα κάτω και
- **flex-direction: column-reverse** Ο κύριος άξονας είναι κάθετα, από κάτω προς τα πάνω.



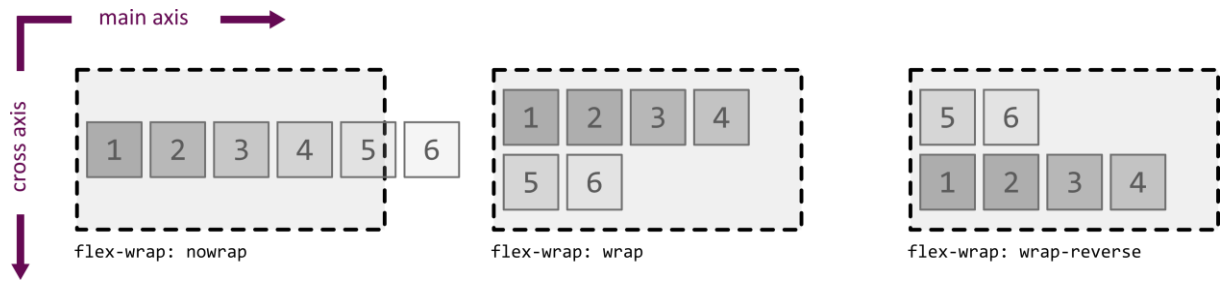
Εικόνα 5.12 *Οι άξονες του υποδοχέα flex αλλάζουν κατεύθυνση με την ιδιότητα* ***flex-direction****. Οι αριθμοί στα κουτιά δείχνουν τη σειρά με την οποία τοποθετούνται.*

Αναδίπλωση σειράς

Όπως είπαμε στην εισαγωγή (Ενότητα 5.5), το Flexbox τοποθετεί τα στοιχεία του σε μία μόνο σειρά, στον κύριο άξονα. Δηλαδή, ο υποδοχέας flex μας είναι μία μόνο σειρά (ή μία μόνο στήλη).

Με την ιδιότητα flex-wrap (***Εικόνα 5.13***) μπορούμε να ζητήσουμε η σειρά αυτή να αναδιπλώνεται. Με την αναδίπλωση ίσως να μας φαίνεται πως τα στοιχεία τοποθετούνται σε πολλές γραμμές, αλλά δεν συμβαίνει αυτό. Στην πραγματικότητα, αναδιπλώνεται η μοναδική σειρά που έχουμε ώστε να χωρέσει μέσα στον υποδοχέα flex:

- **flex-wrap: nowrap** Χωρίς αναδίπλωση, προκαθορισμένη τιμή.
- **flex-wrap: wrap** Αναδίπλωση.
- **flex-wrap: wrap-reverse** Αναδίπλωση, αλλά αντιστρέφεται η φορά του κάθετου άξονα.



Εικόνα 5.13 *Με τη flex-wrap επιτρέπουμε ή απαγορεύουμε στη σειρά των στοιχείων flex να αναδιπλώνεται.*

Θα μπορούσε κανείς να πει πως με τη flex-wrap έχουμε τη δυνατότητα να δημιουργήσουμε σχάρες, πλέγματα (grid). Ωστόσο, αν χρειαζόμαστε πλέγμα για να τοποθετήσουμε τα στοιχεία μας, τότε ας καταφύγουμε στο CSS Grid, καθώς με το Flexbox συνεχίζουμε να έχουμε μόνο μια σειρά, παρ' όλο που μπορεί αυτή να αναδιπλώνεται.

5.6.2 Ιδιότητες των flex items

Το χαρακτηριστικό του Flexbox είναι πως τα στοιχεία είναι εύκαμπτα. Η συμπεριφορά αυτή επηρεάζεται από τις ιδιότητες flex-grow, flex-shrink, flex-basis και τη σύνθετη ιδιότητα flex, που συγκεντρώνει και τις τρεις προηγούμενες. Οι ιδιότητες αυτές είναι ίσως οι πιο δύσκολες στον χειρισμό.

Το μέγεθος λοιπόν ενός στοιχείου flex στον κύριο άξονα είναι το main size, το **βασικό μέγεθος**. Συνήθως τοποθετούμε τα στοιχεία σε μια σειρά, οπότε το βασικό μέγεθος είναι το πλάτος, αλλιώς, αν τα τοποθετήσουμε κάθετα σε μια στήλη, είναι το ύψος.

Η ιδιότητα flex-basis

Η ιδιότητα flex-basis καθορίζει το βασικό μέγεθος του στοιχείου flex στον κύριο άξονα. Μπορεί να πάρει τις συνήθεις τιμές που χρησιμοποιούμε για τις διαστάσεις (Ενότητα 4.6.1) καθώς και την τιμή auto, που είναι η προκαθορισμένη τιμή (αλλά όχι πάντα, όπως θα δούμε). Με την προκαθορισμένη τιμή auto η τιμή της flex-basis γίνεται όση και η τιμή της ιδιότητας width του στοιχείου. Αν δεν έχει οριστεί width για το στοιχείο, τότε γίνεται αυτόματος προσδιορισμός με βάση το μέγεθος του περιεχομένου. Αν όμως η flex-basis έχει οριστεί και δεν είναι auto, τότε η flex-basis έχει προτεραιότητα σε σχέση με τη width.

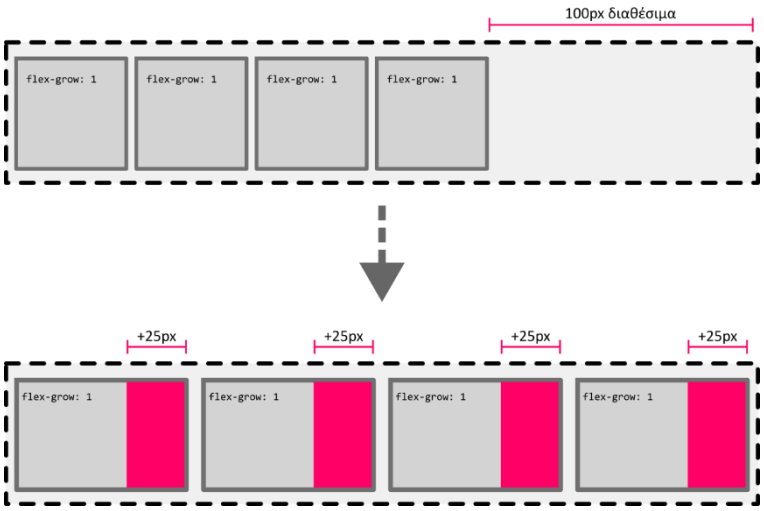
Οι ιδιότητες flex-grow και flex-shrink

Στο Flexbox το μέγεθος του στοιχείου μπορεί να αυξάνει αυτόματα, ανάλογα με τον διαθέσιμο χώρο, με την ιδιότητα flex-grow. Αυτή παίρνει για τιμή έναν θετικό αριθμό χωρίς μονάδες μέτρησης. Αν δεν ορίσουμε την ιδιότητα, τότε η flex-grow έχει την **προκαθορισμένη τιμή 0**. Αυτό σημαίνει πως δεν θα αυξηθεί το μέγεθος του στοιχείου, ακόμη και αν υπάρχει διαθέσιμος χώρος.

Αν ορίσουμε σε όλα τα στοιχεία ενός υποδοχέα flex πως το flex-grow θα είναι ίσο με 1, τότε όσο αυξάνει το μέγεθος του υποδοχέα θα μεγαλώνουν και τα στοιχεία flex με τον ίδιο ρυθμό.

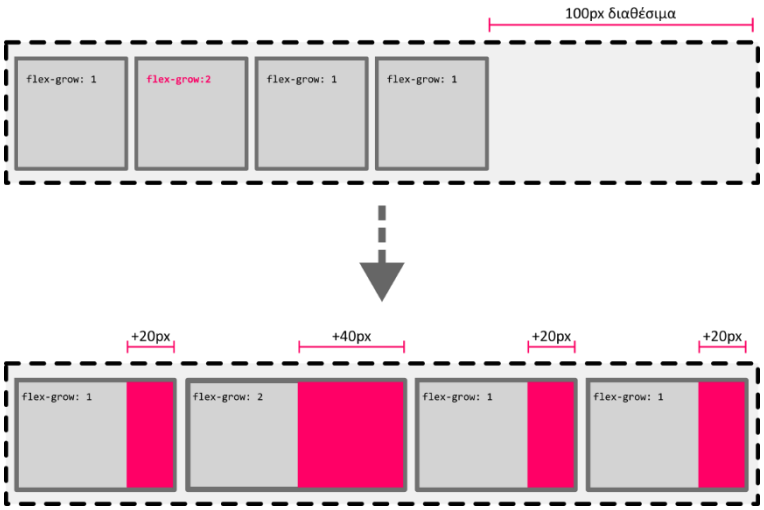
Έστω, για παράδειγμα, πως έχουμε 4 τέτοια στοιχεία flex με flex-grow: 1 και ο διαθέσιμος χώρος είναι 100px. Τότε καθένα από τα 4 θα μεγαλώσει κατά (100px / 4) x 1 = 25px (***Εικόνα 5.14***). Αν ορίσουμε για ένα μόνο από τα 4 στοιχεία το flex-grow: 2 και αφήσουμε τα υπόλοιπα τρία με flex-grow: 1, τότε ο διαθέσιμος

χώρος θα μοιραστεί σε 5 μονάδες ως εξής: Το στοιχείο με το flex-grow: 2 θα πάρει (100 / 5) x 2 = 40px και τα άλλα 3 στοιχεία από (100px / 5) x 1 = 20px (***Εικόνα 5.15***).



Εικόνα 5.14 *Στο παράδειγμα έχουμε 4 στοιχεία flex με flex-grow: 1 και ο διαθέσιμος χώρος είναι 100px. Τότε καθένα από τα 4 θα μεγαλώσει κατά (100px / 4) x 1 = 25px.*

Αντίστοιχα, η flex-shrink προσδιορίζει το πώς θα μειώνεται το μέγεθος ενός στοιχείου flex. **Η προκαθορισμένη τιμή είναι 1**. Αυτό σημαίνει πως όλα τα στοιχεία flex θα μειώνουν το μέγεθός τους με τον ίδιο ρυθμό. Ο υπολογισμός είναι αντίστοιχος με αυτόν του flex-grow. Ο υπολογισμός ενεργοποιείται όταν το συνολικό μέγεθος των στοιχείων flex ξεπερνά το μέγεθος του υποδοχέα flex. Αν, για παράδειγμα, 4 στοιχεία έχουν flex-shrink: 1 και λείπουν 100px από τον υποδοχέα, τότε όλα θα μικρύνουν κατά 25px.



Εικόνα 5.15 *Στο παράδειγμα αυτό έχουμε το 2ο στοιχείο με flex-grow: 2 και τα υπόλοιπα τρία με flex-grow: 1. Ο διαθέσιμος χώρος των 100px θα μοιραστεί σε 5 μονάδες ως εξής: Το στοιχείο με το flex-grow: 2 θα πάρει (100 / 5) x 2 = 40px και τα άλλα 3 στοιχεία από (100px / 5) x 1 = 20px.*

Η αύξηση και η μείωση του μεγέθους γίνεται ως προς το βασικό μέγεθος, το οποίο ορίζεται, όπως είπαμε, με την ιδιότητα flex-basis. Ο υπολογισμός της flex-basis γίνεται πρώτος, δηλαδή πριν προστεθούν ή αφαιρεθούν οι τιμές που ορίζονται με τη flex-grow και τη flex-shrink. Οπότε, πρώτα προσδιορίζεται η τιμή της flex-basis και στη συνέχεια, αν ο υποδοχέας flex έχει διαθέσιμο χώρο, αυτός αποδίδεται στα στοιχεία flex με βάση τη flex-grow. Αν, αντίθετα ο διαθέσιμος χώρος (δηλαδή το μέγεθος του υποδοχέα στην κύρια διάσταση) είναι μικρότερος από το άθροισμα όλων των βασικών μεγεθών, τότε τα στοιχεία μικραίνουν σύμφωνα με τη flex-shrink.

Όπως είπαμε, η προκαθορισμένη τιμή είναι auto. Αν πάρει την τιμή content, η flex-basis θα γίνει ίση με

τον χώρο που χρειάζεται το περιεχόμενο, ανεξάρτητα από την τιμή της width του στοιχείου. Μπορεί να πάρει και τιμή %, που σημαίνει ποσοστό του μεγέθους του υποδοχέα flex.

Η σύνθετη ιδιότητα flex

Αν και είδαμε τις τρεις ιδιότητες ξεχωριστά, η πιο συνηθισμένη και η συνιστώμενη από το πρότυπο μορφή αυτών των ιδιοτήτων είναι η σύνθετη ιδιότητα flex.

Η ιδιότητα flex παίρνει τιμές με τη σειρά για το flex-grow flex-shrink flex-basis. Η flex έχει αρχική τιμή 0 1 auto. Ωστόσο, αν δώσουμε μόνο την τιμή για το βασικό μέγεθος, αν πούμε δηλαδή flex: 4em, τότε αυτό ισοδυναμεί με flex: 1 1 4em. Δηλαδή, όταν χρησιμοποιούμε τη σύνθετη ιδιότητα flex, η flex-grow έχει αρχική τιμή 1 και όχι 0, όπως θα περιμέναμε.

Μερικά παραδείγματα των τιμών που μπορεί να πάρει η flex:

Τιμή	**Ισοδυναμεί με**
`flex: 0 auto`	`flex: 0 1 auto`
`flex: initial`	`flex: 0 1 auto`
`flex: auto`	`flex: 1 1 auto`
`flex: none`	`flex: 0 0 auto`
`flex: n` (θετικός αριθμός)	`flex: n 1 0%`

Πίνακας 5.4 *Παραδείγματα τιμών της ιδιότητας flex.*

5.6.3 Στοίχιση των στοιχείων flex

Η στοίχιση των στοιχείων στο Flexbox γίνεται με τη βοήθεια του CSS Box Alignment Module. Η ίδια προδιαγραφή του CSS χρησιμοποιείται και από το CSS Grid (Ενότητα 5.7). Μπορούμε να ορίσουμε τη στοίχιση σε δύο σημεία:

- Στο κιβώτιο στοίχισης, δηλαδή στον υποδοχέα flex,
- στο στοιχιζόμενο κιβώτιο, δηλαδή σε κάθε στοιχείο flex ξεχωριστά.

Οι ιδιότητες αυτές έχουν κάποια συμμετρία. Έχουμε τρεις ιδιότητες για τη στοίχιση στον κάθετο ή τον block άξονα (**Πίνακας 5.5**) και άλλες τρεις για τον κύριο ή τον inline άξονα (**Πίνακας 5.6**). Ωστόσο, χρειάζεται λίγη προσοχή στο Flexbox, όπου μόνο η justify-content από τις τρεις ιδιότητες έχει ισχύ.

	Κάθετος ή block άξονας ↕	**Εφαρμογή**
Στοίχιση στον υποδοχέα:	align-content	CSS Grid, Flexbox
	align-items	CSS Grid, Flexbox
Στοίχιση στο στοιχείο:	align-self	CSS Grid, Flexbox

Πίνακας 5.5 *Για τη στοίχιση στον κάθετο ή block άξονα έχουμε στη διάθεσή μας 3 ιδιότητες.*

	Κύριος ή inline άξονας ↔	**Εφαρμογή**
Στοίχιση στον υποδοχέα:	justify-content	CSS Grid, Flexbox
	justify-items	CSS Grid
Στοίχιση στο στοιχείο:	justify-self	CSS Grid

Πίνακας 5.6 *Για τη στοίχιση στον κύριο ή inline άξονα έχουμε στη διάθεσή μας 3 ιδιότητες. Προσοχή όμως, καθώς στο Flexbox έχουμε μόνο την* ***justify-content*** *στη διάθεσή μας.*

5.6.4 Στοίχιση στον υποδοχέα flex

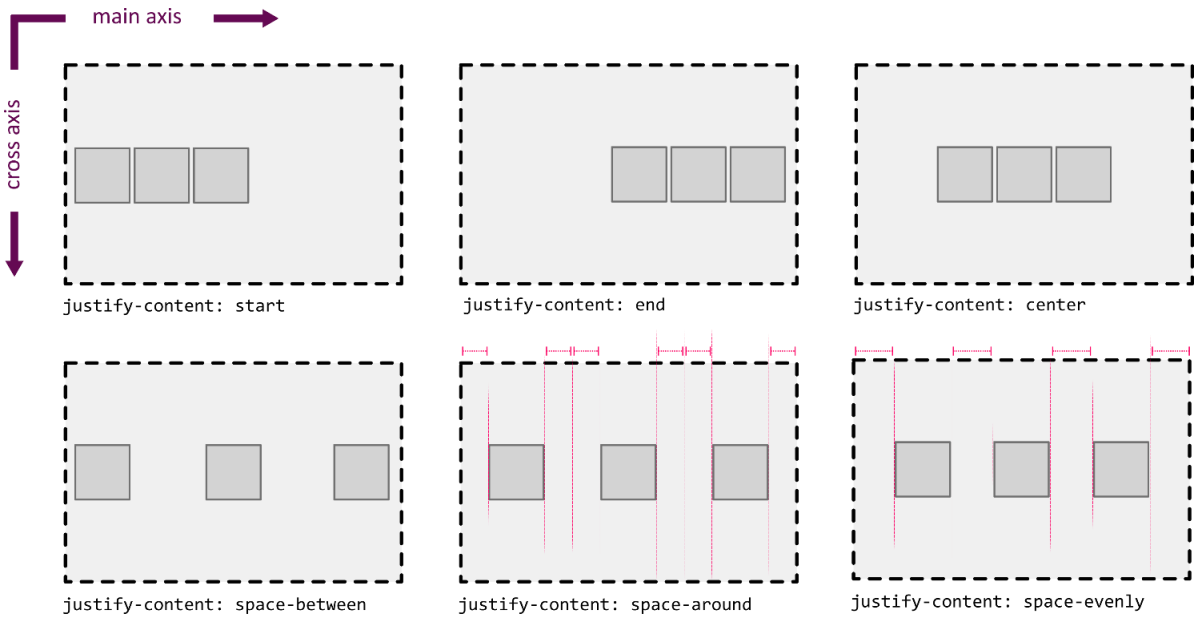
Έχουμε λοιπόν τη δυνατότητα να αλλάξουμε τον τρόπο με τον οποίο στοιχίζονται τα στοιχεία flex μέσα στον υποδοχέα flex. Αυτή τη συμπεριφορά μπορούμε να την αλλάξουμε για κάθε άξονα ξεχωριστά.

Για τον **κύριο άξονα** χρησιμοποιούμε την ιδιότητα justify-content (***Εικόνα 5.16***):

- **justify-content: flex-start** Τα στοιχεία τοποθετούνται στην αρχή του **κύριου άξονα**, το πρώτο

ακουμπάει την πλευρά **main-start** και τα επόμενα είναι το ένα δίπλα στο άλλο.

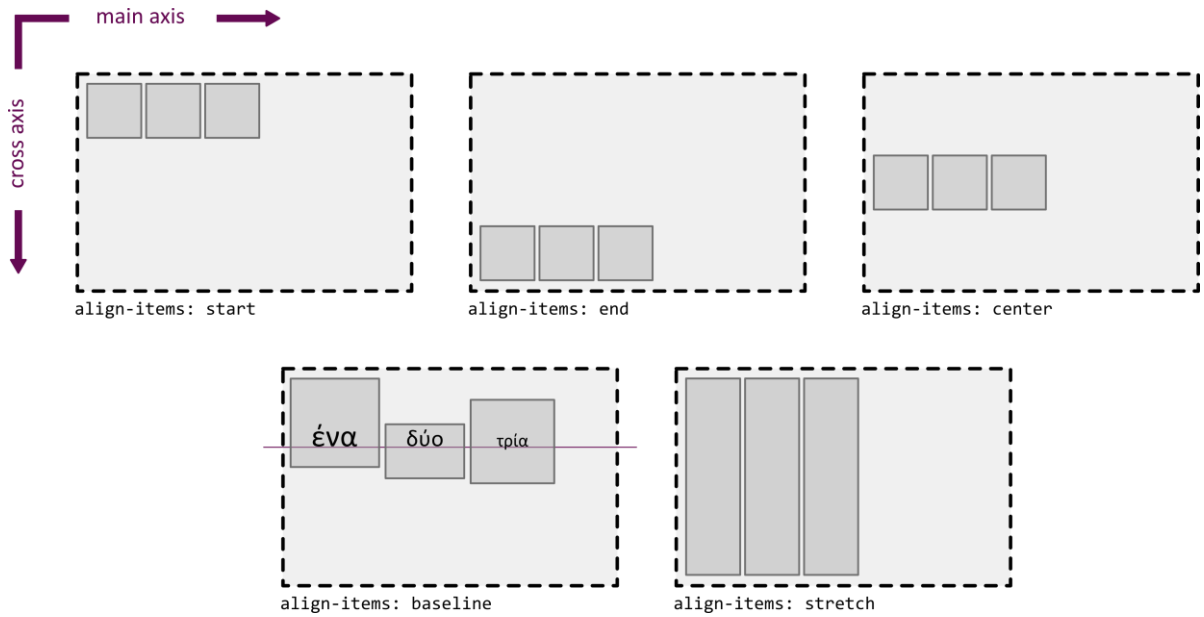
- **justify-content: flex-end** Τα στοιχεία τοποθετούνται στο τέλος του **κύριου άξονα**, το τελευταίο ακουμπάει την πλευρά **main-end** και τα προηγούμενα είναι το ένα δίπλα στο άλλο.
- **justify-content: center** Τα στοιχεία τοποθετούνται στο κέντρο, το ένα δίπλα στο άλλο.
- **justify-content: space-between** Το πρώτο και το τελευταίο στοιχείο ακουμπάνε την αντίστοιχη πλευρά και όλος ο διαθέσιμος κενός χώρος μοιράζεται ίσα ανάμεσα στα στοιχεία.
- **justify-content: space-around** Ο ίδιος διαθέσιμος κενός χώρος μοιράζεται σε όλα τα στοιχεία.
- **justify-content: space-evenly** Όπως το space-around, με τη διαφορά πως ο χώρος στην αρχή και στο τέλος είναι διπλός.



Εικόνα 5.16 *Με την **justify-content** τροποποιούμε τη διάταξη στον **κύριο άξονα**.*

Για τον **κάθετο άξονα** χρησιμοποιούμε την ιδιότητα align-items (***Εικόνα 5.17***):

- **align-items: flex-start** Τα στοιχεία τοποθετούνται στην αρχή του **κάθετου άξονα**, το πρώτο ακουμπάει την πλευρά **cross-start** και τα επόμενα είναι το ένα δίπλα στο άλλο.
- **align-items: flex-end** Τα στοιχεία τοποθετούνται στο τέλος του **κάθετου άξονα**, το τελευταίο ακουμπάει την πλευρά **cross-end** και τα προηγούμενα είναι το ένα δίπλα στο άλλο.



Εικόνα 5.17 *Με την **align-items** τροποποιούμε τη διάταξη των στοιχείων στον **κάθετο άξονα**.*

- **align-items: center** Τα στοιχεία τοποθετούνται στο κέντρο του **κάθετου άξονα**, το ένα δίπλα στο άλλο.
- **align-items: baseline** Στοιχίζονται οι baselines των στοιχείων. Στην πλευρά **cross-start** ακουμπάει το στοιχείο του οποίου η baseline έχει τη μεγαλύτερη απόσταση από την **cross-start**. Τα υπόλοιπα στοιχεία στοιχίζονται έτσι ώστε οι γραμμές τους να είναι όλες σε μια ευθεία. Η baseline είναι η γραμμή πάνω στην οποία τοποθετούνται τα γράμματα στο σύστημα γραφής.
- **align-items: stretch** Τα στοιχεία τεντώνονται ώστε το εξωτερικό περιθώριό (margin) τους να ταυτίζεται με την έκταση του κάθετου άξονα.

Εικόνα 5.18 Με την **align-content** στοιχίζουμε τις γραμμές του Flexbox στον **κάθετο άξονα**.

Τέλος, με την align-content (***Εικόνα 5.18***) στοιχίζουμε τις **γραμμές** σε σχέση με τον υποδοχέα flex.

- **align-content: flex-start** Τα στοιχεία τοποθετούνται στην αρχή του **κάθετου άξονα**, το πρώτο ακουμπάει την πλευρά **cross-start** και τα επόμενα είναι το ένα δίπλα στο άλλο.
- **align-content: flex-end** Τα στοιχεία τοποθετούνται στο τέλος του **κάθετου άξονα**, το τελευταίο ακουμπάει την πλευρά **cross-end** και τα προηγούμενα είναι το ένα δίπλα στο άλλο.
- **align-content: center** Τα στοιχεία τοποθετούνται στο κέντρο του **κάθετου άξονα**, το ένα δίπλα στο άλλο.
- **align-content: space-between** Το πρώτο και το τελευταίο στοιχείο ακουμπάνε την αντίστοιχη πλευρά και όλος ο διαθέσιμος κενός χώρος μοιράζεται ίσα ανάμεσα στα στοιχεία.
- **align-content: space-around** Ο ίδιος διαθέσιμος κενός χώρος μοιράζεται σε όλα τα στοιχεία.
- **align-content: space-evenly** Όπως το space-around, με τη διαφορά πως ο χώρος στην αρχή και στο τέλος είναι διπλός.
- **align-content: stretch** Τα στοιχεία τεντώνονται ώστε να καταλαμβάνουν όλο τον χώρο στον κάθετο άξονα.

5.6.5 Στοίχιση στο κάθε στοιχείο

Στον κάθετο άξονα μπορούμε να στοιχίσουμε ένα στοιχείο σε σχέση με τα υπόλοιπα στοιχεία χρησιμοποιώντας την ιδιότητα align-self (***Εικόνα 5.17***):

- **align-self: flex-start** Το στοιχείο στοιχίζεται στην αρχή του **κάθετου άξονα**, προς την πλευρά **cross-start**.
- **align-self: flex-end** Το στοιχείο στοιχίζεται στο τέλος του **κάθετου άξονα**, προς την πλευρά **cross-end**.
- **align-self: center** Το στοιχείο στοιχίζεται στο κέντρο του **κάθετου άξονα**.

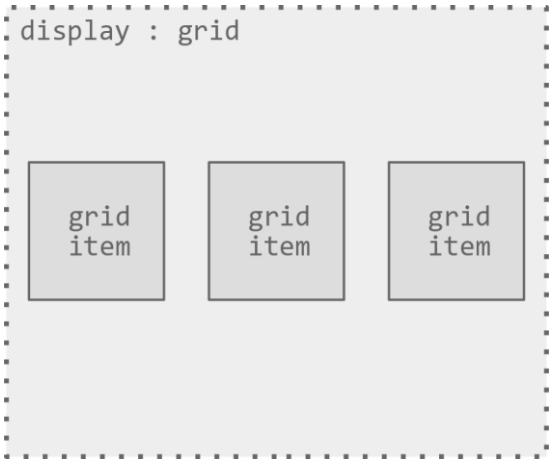
- **align-self: baseline** Στοιχίζεται η baseline του στοιχείου. Η baseline είναι η γραμμή πάνω στην οποία τοποθετούνται τα γράμματα στο σύστημα γραφής.
- **align-self: stretch** Το στοιχείο τεντώνεται ώστε το εξωτερικό περιθώριο (margin) του να ταυτίζεται με την έκταση του **κάθετου άξονα**.

Στο Flexbox δεν μπορούμε να χρησιμοποιήσουμε την ιδιότητα justify-self για να στοιχίσουμε το στοιχείο στον κύριο άξονα.

5.7 CSS Grid

Η βασική ιδέα του CSS Grid είναι η δημιουργία ενός πλέγματος που αποτελείται από σειρές και στήλες. Το **grid** είναι ένας ισχυρός μηχανισμός για τη διάταξη των στοιχείων της σελίδας μας σε δύο διαστάσεις. Η βασική ιδέα του grid είναι ότι κατασκευάζουμε ένα πλέγμα στο οποίο έπειτα μπορούμε να τοποθετήσουμε τα στοιχεία μας με μεγάλη ακρίβεια. Είναι μηχανισμός διάταξης σε δύο διαστάσεις, σε αντίθεση με το Flexbox (Ενότητα 5.6) που διατάσσει τα στοιχεία σε μια σειρά, δηλαδή σε μία μόνο διάσταση.

Ο υποδοχέας του πλέγματος (**grid container**) είναι το στοιχείο που θα περιέχει το πλέγμα πάνω στο οποίο τοποθετούμε τα στοιχεία μας. Τα στοιχεία που τοποθετούμε στο πλέγμα ονομάζονται στοιχεία πλέγματος (**grid items**). Μπορούμε να τροποποιήσουμε τις ιδιότητες του πλέγματός μας στα δύο αυτά σημεία, στο στοιχείο που είναι ο υποδοχέας του πλέγματος ή στα ίδια τα στοιχεία του πλέγματος, τα grid items.

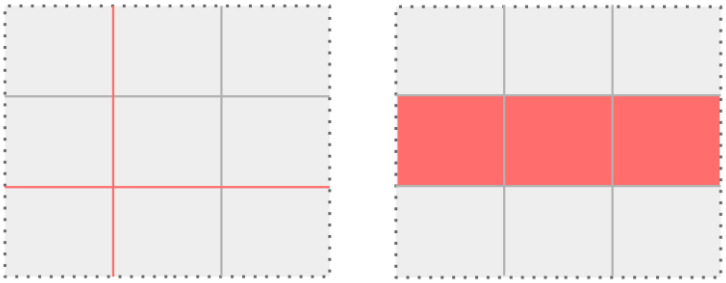


Εικόνα 5.19 *Ένας υποδοχέας πλέγματος περιέχει στοιχεία πλέγματος.*

5.7.1 Ορισμός του πλέγματος

Το πλέγμα αποτελείται από κάθετες και οριζόντιες γραμμές (**Εικόνα *5.20***). Ανάμεσα από δύο οριζόντιες γραμμές σχηματίζεται μια *σειρά* και ανάμεσα από δύο κάθετες γραμμές σχηματίζεται μια *στήλη*. Τα στοιχεία του πλέγματος τοποθετούνται στις σειρές και στις στήλες του.

Οι γραμμές του πλέγματος (**grid lines**) σχηματίζουν τις σειρές ή τις στήλες μέσα στις οποίες τοποθετούνται τα στοιχεία, για παράδειγμα, η δεύτερη σειρά στην εικόνα βρίσκεται ανάμεσα από τη 2η και την 3η οριζόντια γραμμή:



Εικόνα 5.20 *Οι γραμμές του πλέγματος σχηματίζουν τις σειρές ή τις στήλες μέσα στις οποίες τοποθετούνται τα στοιχεία. Στο παράδειγμα η δεύτερη σειρά βρίσκεται ανάμεσα από τη 2η και την 3η οριζόντια γραμμή.*

Έχουμε διάφορους τρόπους για να ορίσουμε το πλήθος των σειρών και στηλών του grid.

Συνήθως ορίζουμε το πλέγμα με την ιδιότητα `grid-template`, όπου δηλώνουμε το πλήθος και το μέγεθος των σειρών και στηλών του πλέγματός μας. Η δήλωση `grid-template: 5em 10em / 7em 7em` ορίζει ένα πλέγμα με δύο σειρές και δύο στήλες. Οι σειρές έχουν μέγεθος 5 και 10 em, ενώ οι στήλες 7 και 7 em αντίστοιχα.

Ίσως πιο διαισθητικός τρόπος είναι να ορίσουμε το πλέγμα με την ιδιότητα `grid-template-areas`. Σε αυτή την περίπτωση ορίζουμε το πλέγμα ονοματίζοντας τις περιοχές του. Για παράδειγμα, η δήλωση

```
#my-grid {
    grid-template-areas:
        "pano-a pano-m pano-d"
        "mesi   mesi   mesi"
        "kato-a kato-m kato-d";
}
```

ορίζει ένα πλέγμα 3x3 ονοματίζοντας επτά περιοχές με ονόματα που διαλέξαμε αυθαίρετα (τρεις περιοχές πάνω, μία στη μέση και τρεις κάτω).

5.7.2 Διαστάσεις των σειρών ή των στηλών του πλέγματος

Όταν ορίσαμε ένα πλέγμα πιο πάνω με τη δήλωση grid-template: 5em 10em / 7em 7em, προσδιορίσαμε τις διαστάσεις των σειρών και των στηλών του με ανελαστικό τρόπο. Αν το στοιχείο που θα τοποθετήσουμε είναι μεγαλύτερο από τις διαστάσεις της σειράς ή της στήλης, τότε θα έχουμε επικάλυψη.

Μπορούμε όμως να ορίσουμε με ελαστικό τρόπο τις διαστάσεις των σειρών και στηλών, χρησιμοποιώντας τη μονάδα **fr** (fraction). Για παράδειγμα, με τη δήλωση grid-template: 7em 7em / 1fr 2fr 2fr 1fr ορίζουμε πως χωρίζουμε το πλέγμα μας σε 4 στήλες και πως οι δύο κεντρικές στήλες θα έχουν πάντα διπλάσιο πλάτος από τις δύο ακριανές.

Σε ακόμη πιο σύνθετα πλέγματα μπορούμε να πετύχουμε το ίδιο αποτέλεσμα χρησιμοποιώντας τη συνάρτηση repeat(). Για παράδειγμα, η ακριβώς παραπάνω δήλωση γίνεται grid-template: 7em 7em / 1fr repeat(2, 2fr) 1fr.

Επίσης, χρήσιμη είναι η συνάρτηση minmax(), με την οποία μπορούμε να ορίσουμε ελάχιστη και μέγιστη διάσταση. Για παράδειγμα, με grid-template: 7em 7em / minmax (20em, 1fr) 2fr 2fr minmax (20em, 1fr) ορίζουμε πως η 1η και η 4η στήλη μας δεν θέλουμε να έχουν πλάτος λιγότερο από 20em.

Συχνά θα θέλουμε να προσδιορίσουμε την ελάχιστη διάσταση σύμφωνα με το μέγεθος του αντικειμένου. Πολύ χρήσιμες είναι οι δηλώσεις διάστασης με max-content και min-content. Η πρώτη ορίζει το μέγιστο μέγεθος που μπορεί να χρειαστεί το στοιχείο για να αναπαρασταθεί, αν είναι κείμενο έτσι ώστε να μην αναδιπλωθεί. Η δεύτερη τιμή αναφέρεται στο ελάχιστο δυνατό μέγεθος που χρειάζεται το στοιχείο για να αναπαραστήσει το περιεχόμενό του χωρίς υπερχείλιση.

5.7.3 Σιωπηρή δημιουργία του πλέγματος

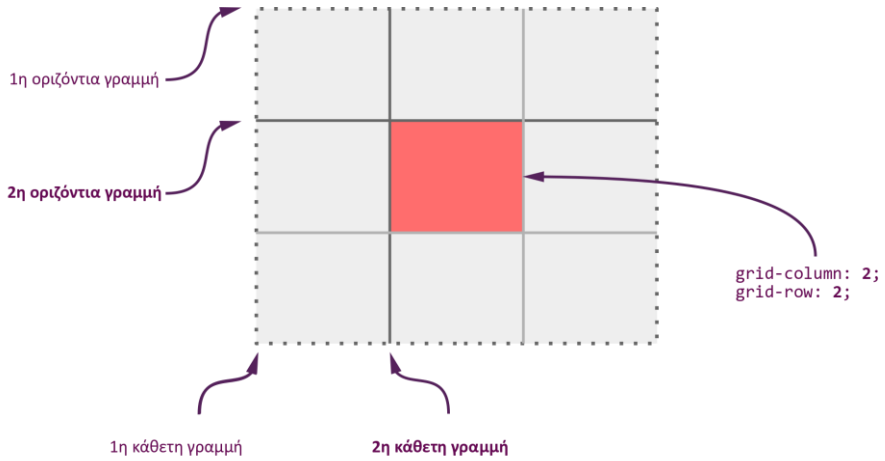
Εκτός από τη ρητή διατύπωση για τη δημιουργία του πλέγματος που είδαμε πιο πάνω, μπορούμε να δημιουργήσουμε γραμμές ή στήλες με σιωπηρό τρόπο, το λεγόμενο *implicit grid*.

Αν ένα στοιχείο του πλέγματος τοποθετηθεί σε στήλη ή σειρά που δεν έχει προσδιοριστεί, για παράδειγμα αν ζητήσουμε να τοποθετηθεί το στοιχείο στην 3η σειρά ενός πλέγματος 2x2, τότε θα δημιουργηθεί σιωπηρά μια νέα σειρά για να γίνει η τοποθέτηση.

Το μέγεθος των σειρών και των στηλών που δημιουργούνται σιωπηρά προσδιορίζεται από τις ιδιότητες grid-auto-rows και grid-auto-columns αντίστοιχα.

5.7.4 Τοποθέτηση ενός στοιχείου στο πλέγμα

Για να τοποθετήσουμε ένα στοιχείο σε συγκεκριμένες σειρές ή γραμμές έχουμε τις ιδιότητες grid-row και grid-column. Οι ιδιότητες αυτές παίρνουν για τιμές όχι τον αριθμό σειράς ή στήλης, αλλά **τον αριθμό της γραμμής** που ορίζει την αρχή μιας σειράς ή μιας στήλης.



Εικόνα 5.21 *Τοποθέτηση του στοιχείου.*

Αν έχουμε χρησιμοποιήσει την grid-template-areas για τον ορισμό του πλέγματος, μπορούμε να χρησιμοποιήσουμε την ιδιότητα grid-area για να τοποθετήσουμε το στοιχείο στη συγκεκριμένη θέση, π.χ. με grid-area: ka για να το τοποθετήσουμε στην περιοχή με όνομα ka.

5.7.5 Τοποθέτηση σε περισσότερα κελιά

Ένα στοιχείο πλέγματος μπορεί να τοποθετηθεί έτσι ώστε να καταλαμβάνει περισσότερο χώρο από όσο ένα μόνο κελί. Μπορούμε να ορίσουμε ένα εύρος κελιών, από γραμμή μέχρι γραμμή. Για παράδειγμα, η δήλωση grid-column: 1 / 3 τοποθετεί το στοιχείο στη στήλη που ξεκινά με την πρώτη κάθετη γραμμή και τελειώνει στην 3η κάθετη γραμμή (προσοχή στη διαφορά μεταξύ «γραμμής» και «σειράς»). Εναλλακτικά, η δήλωση grid-column: 1/ span 2 τοποθετεί το στοιχείο στη στήλη που ξεκινά με την πρώτη κάθετη γραμμή και καταλαμβάνει τον χώρο δύο κελιών, έχει δηλαδή το ίδιο αποτέλεσμα.

Το ίδιο μπορούμε να πετύχουμε και στην περίπτωση που χρησιμοποιούμε την grid-template-areas για τον ορισμό του πλέγματος. Στο παράδειγμά μας με την grid-template-areas πιο πάνω, η περιοχή mesi καταλαμβάνει 3 κελιά, όπως και το στοιχείο που θα τοποθετηθεί εκεί.

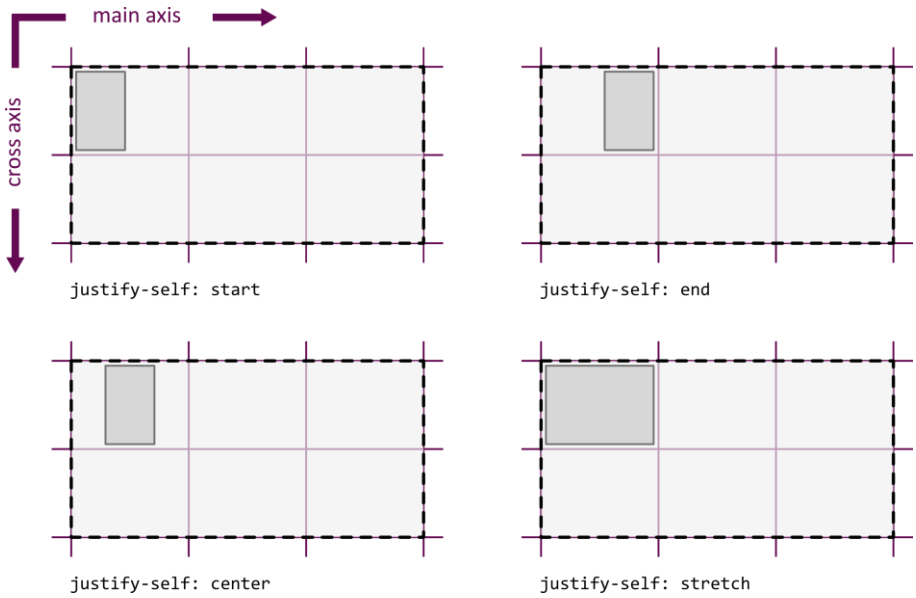
5.7.6 Στοίχιση στο CSS Grid

Η στοίχιση στο CSS Grid γίνεται με τη βοήθεια του CSS Box Alignment Module, όπως και στο Flexbox (Ενότητα 5.6.3). Μπορούμε να ορίσουμε τη στοίχιση σε δύο κατευθύνσεις, την κατεύθυνση block και την κατεύθυνση inline (Ενότητα 4.7.1). Τη στοίχιση των grid items μπορούμε να την προσδιορίσουμε, όπως και στο Flexbox, σε δύο σημεία: στο επίπεδο του υποδοχέα του πλέγματος και στο επίπεδο των στοιχείων του πλέγματος.

Για τη στοίχιση στην κατεύθυνση inline χρησιμοποιούμε την ιδιότητα justify-items, που ορίζει την τιμή που θα έχει η ιδιότητα justify-self σε όλα τα στοιχεία του πλέγματος.

- **justify-self: start** Το στοιχείο πλέγματος στοιχίζεται προς την αρχή της **κατεύθυνσης inline**.

- **justify-self: end** Το στοιχείο πλέγματος στοιχίζεται προς το τέλος της **κατεύθυνσης inline**.
- **justify-self: center** Το στοιχείο πλέγματος τοποθετείται στο κέντρο της **κατεύθυνσης inline**.
- **justify-self: stretch** Το μέγεθος του στοιχείου αυξάνεται τόσο ώστε να χωρέσει μέσα στην περιοχή όπου στοιχίζεται.
- **justify-self: normal** Όπως το strecth, εκτός αν πρόκειται για στοιχεία με δικές τους εσωτερικές διαστάσεις, π.χ. εικόνες.



Εικόνα 5.22 *Με την justify-self τροποποιούμε τη διάταξη στον κύριο άξονα.*

Για τον κάθετο άξονα χρησιμοποιούμε την ιδιότητα align-items, που τη είδαμε στο Flexbox (Ενότητα 5.6.4):

- **align-items: flex-start** Τα στοιχεία πλέγματος τοποθετούνται στην αρχή του **κύριου άξονα**, το πρώτο ακουμπάει την πλευρά **cross-start** και τα επόμενα είναι το ένα δίπλα στο άλλο.
- **align-items: flex-end** Τα στοιχεία πλέγματος τοποθετούνται στο τέλος του **κύριου άξονα**, το τελευταίο ακουμπάει την πλευρά **cross-end** και τα προηγούμενα είναι το ένα δίπλα στο άλλο.
- **align-items: center** Τα στοιχεία πλέγματος τοποθετούνται στο κέντρο του **κύριου άξονα**, το ένα δίπλα στο άλλο.
- **align-items: baseline** Στοιχίζονται οι baselines των στοιχείων πλέγματος. Στην πλευρά **cross-start** ακουμπάει το στοιχείο του οποίου η baseline έχει τη μεγαλύτερη απόσταση από την **cross-start**. Τα υπόλοιπα στοιχεία πλέγματος στοιχίζονται έτσι ώστε οι γραμμές τους να είναι όλες σε μια ευθεία.
- **align-items: stretch** Τα στοιχεία τεντώνονται ώστε το εξωτερικό περιθώριό τους (margin) να ταυτίζεται με την έκταση του κάθετου άξονα.

5.8 Ερωτήσεις αυτοαξιολόγησης

1. Στην κανονική ροή τα στοιχεία block τοποθετούνται το ένα κάτω από το άλλο.
   – Σωστό/Λάθος
2. Για να ορίσουμε ένα στοιχείο σαν inline-block χρησιμοποιούμε τον κανόνα (επιλέξτε το σωστό).

   1.
   ```
   {
       display: inline;
       position: block;
   }
   ```

   2.
   ```
   {
       position: inline;
       display: block;
   }
   ```

   3.
   ```
   {
       display: inline-block;
   }
   ```

   4.
   ```
   {
       display: inline block;
   }
   ```

3. Το στοιχείο span είναι στοιχείο (επιλέξτε το σωστό)
   1. inline
   2. block
   3. inline-block
   4. non3
4. Για το στοιχείο με id div-a, ισχύει ο κανόνας

   ```
   #div-a {
       position: relative;
       display: static;
       inset-inline-start: -10px;
   }
   ```

   Σε σχέση με τι θα τοποθετηθεί το στοιχείο αυτό;
   1. Σε σχέση με το παράθυρο του φυλλομετρητή.
   2. Σε σχέση με τη θέση που θα είχε πριν τοποθετηθεί.
   3. Σε σχέση με τον άμεσο πρόγονο.
   4. Σε σχέση με κάποιον πρόγονο.
5. Για το στοιχείο με id #div-a, ισχύει ο κανόνας

   ```
   #div-a {
       position: fixed;
       left: -10px;
   }
   ```

   Σε σχέση με τι θα τοποθετηθεί το στοιχείο αυτό;
   1. Σε σχέση με το παράθυρο του φυλλομετρητή.
   2. Σε σχέση με τη θέση που θα είχε πριν τοποθετηθεί.
   3. Σε σχέση με τον άμεσο πρόγονο.
   4. Σε σχέση με κάποιον πρόγονο.
6. Προς ποια κατεύθυνση θα μετακινηθεί το στοιχείο που τοποθετείται με τον κανόνα

   ```
   #box {
       position: absolute;
       top: -10px;
   }
   ```

    1. Προς τα πάνω κατά 10px.
    2. Προς τα κάτω κατά 10px.
    3. Προς τα πάνω κατά 20px.
    4. Δε θα μετακινηθεί.
7. Ποια ιδιότητα CSS αλλάζει τον τρόπο τοποθέτησης ενός στοιχείου;
    1. position X
    2. display
    3. set-xy
    4. static
8. Ποιος κανόνας καθορίζει τη μετατόπιση της δεξιάς πλευράς ενός στοιχείου κατά 10px, το οποίο έχει τοποθετηθεί με relative τρόπο;
    1. inset-right: 10px;
    2. inset-block-end: 10px;
    3. position-left: 10px;
    4. right-offset: 10px;
9. Ποια ιδιότητα καθορίζει τη θέση που έχει ένα τοποθετημένο στοιχείο στο stacking context;
    1. stack-axis
    2. z-index
    3. stack-position
    4. z-position
10. Έχουμε τη δυνατότητα να αλλάξουμε το z-index όλων των στοιχείων του DOM.
    – Σωστό/Λάθος
11. Στοιχεία που έχουν τον κανόνα z-index: auto δημιουργούν το δικό τους stacking context.
    – Σωστό/Λάθος
12. Με ποια ιδιότητα ορίζουμε πως ένα στοιχείο δεν θα έχει float στα αριστερά ή στα δεξιά του;
    1. clean
    2. float
    3. clear
    4. no-float
13. Ένα στοιχείο που είναι float συμμετέχει στον υπολογισμό του ύψος του υποδοχέα που το περιέχει.
    – Σωστό/Λάθος
14. Ποια από τις παρακάτω τιμές δεν γίνεται δεκτή από την ιδιότητα float;
    1. left
    2. right
    3. both
    4. none
15. Ο κανόνας display: flex ορίζει πως
    1. το στοιχείο θα είναι flexible όσον αφορά την κατεύθυνση inline.
    2. το στοιχείο, εφόσον είναι ταυτόχρονα και float, θα τοποθετηθεί μπροστά από όλα τα άλλα.
    3. το στοιχείο είναι ένας flex container.
    4. το στοιχείο είναι ένα flex item.
16. Η ιδιότητα flex-direction
    1. αφορά έναν flex container και προσδιορίζει την οριζόντια τοποθέτηση των στοιχείων του.
    2. αφορά έναν flex container και προσδιορίζει την κάθετη τοποθέτηση των στοιχείων του.
    3. αφορά ένα flex item και προσδιορίζει το πώς θα τοποθετηθεί το στοιχείο.
    4. αφορά έναν flex container και προσδιορίζει αν τα στοιχεία του θα τοποθετηθούν σε γραμμές ή σε στήλες.
17. Για να αλλάξουμε την οριζόντια διάταξη των στοιχείων ενός flex container χρησιμοποιούμε την ιδιότητα
    1. align-content
    2. align-items
    3. justify-content
    4. justify-items
18. Η ιδιότητα flex-wrap προσδιορίζει αν τα στοιχεία θα τοποθετηθούν κάθετα ή οριζόντια.
    – Σωστό/Λάθος

19. Ο κύριος άξονας ενός flex container, αν δεν ορίσουμε κάτι άλλο, είναι
    1. ο άξονας inline
    2. ο άξονας block
    3. ο άξονας z
    4. ο άξονας περιστροφής του flex container
20. Η αρχική τιμή της ιδιότητας flex-grow είναι
    1. block-start
    2. inline-start
    3. 0
    4. 1
21. Η προκαθορισμένη τιμή της ιδιότητας flex-grow είναι
    1. 0
    2. 1
    3. 100%
    4. auto
22. Η προκαθορισμένη τιμή της ιδιότητας flex-basis είναι
    1. 0
    2. 1
    3. 100%
    4. auto
23. Με τον κανόνα flex: 50%; η ιδιότητα flex-grow παίρνει την τιμή:
    1. 50%
    2. 100%
    3. 0
    4. 1
24. Ο κανόνας flex: 0; ισοδυναμεί με
    1. flex-grow: 0, flex-shrink: 0, flex-basis: 0%
    2. flex-grow: 0, flex-shrink: 1, flex-basis: 0%
    3. flex-grow: 0, flex-shrink: 0, flex-basis: 100%
    4. flex-grow: 0, flex-shrink: 0, flex-basis: auto
25. Ο κανόνας flex: 0 ορίζει πως το βασικό μέγεθος (flex-basis) του στοιχείου θα είναι 0%.
    – Σωστό/Λάθος
26. Η ιδιότητα box-sizing δεν έχει επιρροή στα flex items, παρά μόνο στον flex container.
    – Σωστό/Λάθος
27. Ο κανόνας flex: 1 1 48% ορίζει πως, εφόσον υπάρχει διαθέσιμος χώρος,
    1. το flex item θα έχει μέγεθος ίσο με το 48% του αρχικού του μεγέθους.
    2. το flex item θα έχει μέγεθος ίσο με το 48% του flex container.
    3. ο flex container θα έχει μέγεθος ίσο με το 48% του αρχικού του μεγέθους.
    4. το τρίτο flex item θα καταλαμβάνει το 48% του διαθέσιμου χώρου.
28. Ποιο θα είναι το πλάτος των στηλών που δημιουργεί το παρακάτω grid layout;

```
.mygrid {
    display: grid;
    width: 600px;
    grid-template 1fr 2fr / 2fr 1fr
}
```

    1. 100px και 200px
    2. 100px, 200px, 200px και 100px
    3. 200px και 400px
    4. 400px και 200px
29. Ο κανόνας

```
#box {
    grid-area: top-left;
}
```

    1. τοποθετεί το στοιχείο #box στην περιοχή grid με όνομα “top-left”.
    2. ορίζει πως το στοιχείο #box είναι ένας grid container με όνομα “top-left”.
    3. τοποθετεί το στοιχείο #box στο πάνω αριστερά μέρος του block μέσα στο οποίο βρίσκεται.
    4. ορίζει πως τα παιδιά του #box θα τοποθετούνται ξεκινώντας από πάνω αριστερά.
30. Η ιδιότητα grid-auto-rows ορίζει αν θα επιτρέπεται ή όχι η αυτόματη δημιουργία γραμμών σε ένα grid.
    – Σωστό/Λάθος
31. Ο κανόνας

```
#box {
    justify-self: flex-start;
}
```

    θα έχει σαν αποτέλεσμα το στοιχείο #box να
    1. τοποθετηθεί στην αριστερή πλευρά του κελιού του grid.
    2. να τοποθετηθεί στην πάνω πλευρά του κελιού του grid.
    3. να τοποθετηθεί στην αρχή της στήλης του grid.
    4. να τοποθετηθεί στην αρχή της γραμμής του grid.
32. Ο κανόνας grid-column: 1/3; ορίζει πως
    1. το grid item θα καταλαμβάνει την περιοχή μεταξύ της 1ης και της 3ης κάθετης γραμμής.
    2. το grid item θα καταλαμβάνει την 1η από τις τρεις διαθέσιμες στήλες.
    3. το grid item θα τοποθετηθεί στην τομή της 1ης γραμμής και της 3ης στήλης.
    4. χωρίζουμε το grid μας σε τρεις ισομεγέθεις περιοχές.
33. Η CSS δεν μας επιτρέπει να αναμείξουμε Flexbox και CSS Grid.
    – Σωστό/Λάθος
34. Η ιδιότητα gap μπορεί να χρησιμοποιηθεί και στο Flexbox και στο Grid.
    – Σωστό/Λάθος

5.9 Βιβλιογραφία και Αναφορές

Το υλικό αυτού του κεφαλαίου καλύπτεται και από τις πηγές και τη βιβλιογραφία που παρουσιάστηκε στο προηγούμενο κεφάλαιο (4.12). Όπως και εκεί, η σελίδα mdn μπορεί να αποτελέσει την πρωταρχική πηγή για περαιτέρω εμβάθυνση και επιπλέον ο αναγνώστης μπορεί να ανατρέξει στα αντίστοιχα πρότυπα, όπως π.χ. το CSS Flexible Box Layout Module Level 1 ή το CSS Grid Layout Module Level 2.

Επιπλέον, οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Για αυτό το κεφάλαιο το σχετικό μάθημα είναι το «Προχωρημένα θέματα ανάπτυξης ιστοσελίδων».

Καθώς το CSS Grid και το Flexbox είναι από μόνοι τους σύνθετοι μηχανισμοί, εξειδικευμένοι οδηγοί και εργαλεία παρέχουν επιπλέον υποστήριξη. Ενδεικτικά προτείνονται οι παρακάτω πηγές:

- Flexbox
  - Common CSS Flexbox Layout Patterns with Example Code
  - A Complete Guide to Flexbox
  - Solved by Flexbox
  - Flexbox: How Big Is That Flexible Box?
  - The ultimate CSS battle: Grid vs Flexbox
- CSS Grid
  - A Complete Guide to Grid
  - Learn CSS Grid
  - Introduction to CSS Grid Layout
  - Grid by example

A. Ξενόγλωσσες

Atkins Jr. T, Rivoal, F., & Etemad E. (2021). *CSS Snapshot 2021* (White paper). W3C.

McFedries, P. (2019). *Web Design Playground - HTML & CSS The Interactive Way*. Manning.

Verou, L. (2015). *CSS Secrets*. O'Reilly Media, Inc.

B. Ελληνόγλωσσες

Αβούρης, Ν. (2018). Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, JavaScript. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr

Αβούρης, Ν., & Σιντόρης, Χ. (2020). Προχωρημένα θέματα ανάπτυξης ιστοσελίδων. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr

Δουληγέρης, Χ., Μαυροπόδη, Ρ., Κοπανάκη, Ε., & Καραλής, Α. (2021). *Τεχνολογίες και Προγραμματισμός στον Παγκόσμιο Ιστό* (2η έκδ.). Εκδόσεις Νέων Τεχνολογιών. Κωδικός βιβλίου στον Εύδοξο: 102125023.

Kyrnin, J., & Meloni, J. (2021). *Sams Teach Yourself HTML5, CSS and Javascript* (3rd ed.). Εκδόσεις Γκιούρδας.

Lemay, L., Coburn, R., & Kyrnin, J. (2016). *Sams Teach Yourself HTML, CSS & JavaScript* (7th ed). Εκδόσεις Γκιούρδας. Κωδικός Βιβλίου στον Εύδοξο: 59357307.

Κεφάλαιο 6. Bootstrap

Σύνοψη

Το πλαίσιο Bootstrap αποτελεί μια από τις πιο δημοφιλείς βιβλιοθήκες για τη γρήγορη ανάπτυξη διεπαφών διαδικτυακών εφαρμογών. Αν και δεν ανήκει στις τεχνολογίες που προτυποποιούνται μέσω του W3C, ωστόσο η Bootstrap αποτελεί μια πολύ χρήσιμη εργαλειοθήκη για τη γρήγορη ανάπτυξη εύρωστων διεπαφών διαδικτύου, ειδικότερα όταν η έμφαση δίνεται στην ανάπτυξη για την πλευρά του εξυπηρετητή. Σε αυτό το κεφάλαιο παρουσιάζεται σύντομα η Bootstrap και συζητούνται διεξοδικά η διάταξη περιεχομένου με τη χρήση της και ο βασικός μηχανισμός της Bootstrap grid. Στη συνέχεια παρουσιάζονται κάποια από τα βασικά γραφικά στοιχεία της Bootstrap όπως το navigation bar, τα buttons, τα form widgets κ.ά.

Προαπαιτούμενη γνώση

Για την κατανόηση και χρήση της Bootstrap είναι απαραίτητη η εξοικείωση με τις βασικές τεχνολογίες πάνω στις οποίες βασίζεται, δηλαδή την HTML και την CSS. Επιπλέον, οι τρέχουσες εκδόσεις της Bootstrap βασίζονται στο Flexbox (Ενότητα 5.6) και το ίδιο το πλαίσιο Bootstrap παρέχει εργαλεία διάταξης που βασίζονται στον μηχανισμό αυτό. Συνεπώς, και η εξοικείωση με το Flexbox είναι επιθυμητή για την κατανόηση αυτού του κεφαλαίου. Τέλος, αν και η Bootstrap συνοδεύεται από γραφικά στοιχεία που βασίζονται στην JavaScript, αυτά δεν θα παρουσιαστούν εδώ. Ωστόσο, η JavaScript χρησιμοποιείται στην τελευταία ενότητα αυτού του κεφαλαίου, όπου συζητείται η επικύρωση δεδομένων με την Bootstrap (Ενότητα 6.7). Επομένως, για την κατανόηση της ενότητας 6.7 απαιτείται η εξοικείωση με τη γλώσσα JavaScript στην πλευρά του φυλλομετρητή, που συζητείται στα επόμενα κεφάλαια (κεφάλαια 7, 8, 9 και 10).

6.1 Η βιβλιοθήκη Bootstrap

Η ποικιλία των συσκευών που χρησιμοποιούν οι σημερινοί χρήστες του διαδικτύου είναι μεγάλη, από σταθερούς υπολογιστές με μεγάλες οθόνες και φορητούς υπολογιστές μέχρι ταμπλέτες και κινητά τηλέφωνα. Το αποτέλεσμα είναι να πρέπει οι ιστοσελίδες να προσαρμόζονται στις διαστάσεις της οθόνης που προβάλλονται, να είναι, όπως λέμε, “responsive”. Τα εργαλεία που μας δίνει η CSS3 για να το πετύχουμε αυτό, όπως το Flexbox ή το CSS Grid, είναι πολύ ισχυρά. Ωστόσο, ακόμα και με αυτά τα βοηθήματα, η δημιουργία προσαρμοστικών ιστοσελίδων συνεχίζει να είναι χρονοβόρα διαδικασία.

Μια λύση σε αυτό το πρόβλημα είναι να χρησιμοποιήσουμε μια έτοιμη βιβλιοθήκη CSS, ένα CSS framework. Μια τέτοια βιβλιοθήκη αναλαμβάνει να επιλύσει τα βασικά προβλήματα προσαρμοστικότητας και μας επιτρέπει να εστιάσουμε σε άλλα θέματα.

Η Bootstrap είναι μια τέτοια βιβλιοθήκη CSS/JavaScript που μας βοηθά να σχεδιάσουμε και να κατασκευάσουμε σε λίγο χρόνο προσαρμοστικές ιστοσελίδες.

6.2 Ενσωμάτωση της Bootstrap

Η διανομή της Bootstrap αποτελείται από αρχεία CSS και JavaScript. Κατ’ ελάχιστον, για να χρησιμοποιήσουμε την Bootstrap, αρκεί να ενσωματώσουμε το αρχείο CSS της διανομής της. Αν θέλουμε να χρησιμοποιήσουμε μόνο το πλέγμα της Bootsrap (grid), αυτό αρκεί. Αν θέλουμε να χρησιμοποιήσουμε και μερικά από τα γραφικά στοιχεία της Bootstrap, θα χρειαστούμε και το συνοδευτικό αρχείο JavaScript.

Για την ενσωμάτωση έχουμε δύο επιλογές:

- να κατεβάσουμε τη διανομή της Bootstrap και να αποθηκεύσουμε τα σχετικά αρχεία (CSS και JavaScript) μαζί με το πρότζεκτ μας ή
- να ενσωματώσουμε τα σχετικά αρχεία μέσα από το CDN (Content Delivery Network).

Θα βρούμε οδηγίες και για τα δύο στην ιστοσελίδα της Bootstrap. Αν δεν υπάρχει κάποιος σοβαρός λόγος για το αντίθετο, όπως η ανάγκη να τροποποιήσουμε τα αρχεία της Bootstrap, συνήθως είναι προτιμότερη η δεύτερη επιλογή, για λόγους ταχύτητας και κόστους. Αν ο φυλλομετρητής του χρήστη έχει ήδη φορτώσει τα αρχεία της Bootstrap από προηγούμενη επίσκεψή του σε άλλη σελίδα, έχει μεγάλες πιθανότητες να μην τα ξανακατεβάσει για τη δική μας εξαιτίας της προσωρινής αποθήκευσης. Όσον αφορά το κόστος, αν ο πάροχός μας μας χρεώνει

ανάλογα με την κίνηση, τότε είναι προτιμότερο να μην προμηθεύουμε στους χρήστες μας αυτά τα αρχεία εμείς, αλλά το CDN.

6.3 Bootstrap containers

Για τη διάταξη της σελίδας η Bootstrap μάς διαθέτει ένα πλέγμα, το Bootstrap grid. Για να το χρησιμοποιήσουμε, αρκεί να ορίσουμε ένα με την κλάση .container-fluid ή .container

```
<div class="container">
    ...
    ...
</div>
```

Η διαφορά τους είναι πως η κλάση .container-fluid ορίζει ένα πλέγμα που το μέγεθός του είναι το 100% του μεγέθους της περιοχής μέσα στην οποία βρίσκεται. Συνήθως αυτό είναι το μέγεθος του παραθύρου προβολής ή "viewport".

Από την άλλη, το πλέγμα που ορίζεται με την κλάση .container αλλάζει μέγεθος κλιμακωτά, πηδώντας από το ένα μέγεθος στο άλλο όσο μεγαλώνει η διαθέσιμη περιοχή. Όπως φαίνεται στον πίνακα στην **Εικόνα *6.1***, για παράδειγμα, όταν το διαθέσιμο εύρος γίνει ίσο ή μεγαλύτερο από 768px, τότε το grid θα πάρει τη διάσταση md, η οποία είναι 720px σύμφωνα με την τεκμηρίωση της Bootstrap. Θα παραμείνει σε αυτό το μέγεθος για όσο η διάσταση του παραθύρου προβολής είναι μικρότερη από 992px. Είναι πολύ βολικό να αναφερόμαστε σε αυτά τα βήματα με τις συντομογραφίες τους (**sm**, **md**, **lg**, **xl**, **xxl**) και όχι με τις αριθμητικές τιμές τους.

Breakpoint	Επίθημα κλάσης	Διάσταση viewport
Extra small		<576px
Small	sm	≥576px
Medium	md	≥768px
Large	lg	≥992px
Extra large	xl	≥1200px
Extra extra large	xxl	≥1400px

Εικόνα 6.1 *Τα breakpoints της Bootstrap.*

Εκτός από την κλάση .container-fluid και την .container, μπορούμε να χρησιμοποιήσουμε και μια σειρά από κλάσεις στη μορφή .container-xx, όπου **xx** είναι ένα από τα breakpoints. Για παράδειγμα, ένα grid που έχει οριστεί με .container-md θα είναι fluid όταν η διάσταση είναι από md και κάτω, ενώ από md και πάνω θα συμπεριφέρεται σαν .container, δηλαδή θα μεγαλώνει κλιμακωτά, πηδώντας από breakpoint σε breakpoint όσο μεγαλώνει ο διαθέσιμος χώρος.

6.4 Το πλέγμα της Bootstrap (grid)

Με το πλέγμα της Bootstrap, το grid, μπορούμε

- να διαιρέσουμε την οθόνη (δηλαδή τον διαθέσιμο χώρο στον φυλλομετρητή) σε τμήματα μεταβλητού μεγέθους,
- να αλλάξουμε το μέγεθος των στηλών του ανάλογα με το breakpoint,

- να έχουμε εμφωλευμένα grid, δηλαδή grid μέσα σε grid.

Το grid αποτελείται από στήλες (row) που βρίσκονται μέσα σε γραμμές (column). Οι γραμμές με τη σειρά τους περιέχονται σε .container.

Το πιο βασικό παράδειγμα είναι το grid με μια στήλη:

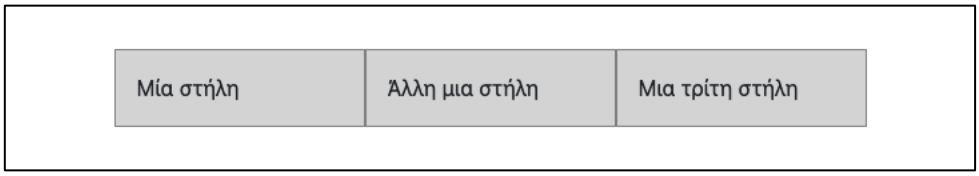
```
<div class="container">
    <div class="row">
        <div class="col">Μια στήλη</div>
    </div>
</div>
```

Οι βασικοί κανόνες χρήσης του grid, ξεκινώντας από έξω προς τα μέσα, είναι:

- οι γραμμές τοποθετούνται μέσα σε έναν .container,
- οι στήλες τοποθετούνται σε γραμμές,
- το περιεχόμενο τοποθετείται σε στήλες.

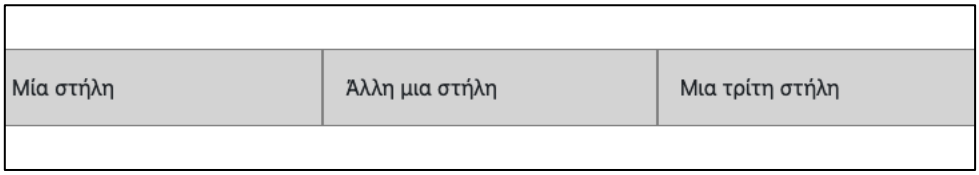
Το παρακάτω παράδειγμα δημιουργεί 3 ισοπαχείς στήλες (**Εικόνα *6.2***).

```
<div class="container">
    <div class="row">
        <div class="col">Μια στήλη</div>
        <div class="col">Άλλη μια στήλη</div>
        <div class="col">Μια τρίτη στήλη</div>
    </div>
</div>
```



Εικόνα 6.2. *Τρεις ισόπαχες στήλες στο πλέγμα της Bootstrap (έχει προστεθεί με CSS γκρι χρώμα και πλαίσιο σε κάθε κελί).*

Ο εξωτερικός .container κεντράρει το grid οριζόντια και προσθέτει κάποιο περιθώριο δεξιά και αριστερά από τα όρια του viewport. Αν δεν θέλουμε το περιθώριο, χρησιμοποιούμε την κλάση .container-fluid (**Εικόνα *6.3***).



Εικόνα 6.3. *Το ίδιο παράδειγμα, αυτή τη φορά με .container-fluid.*

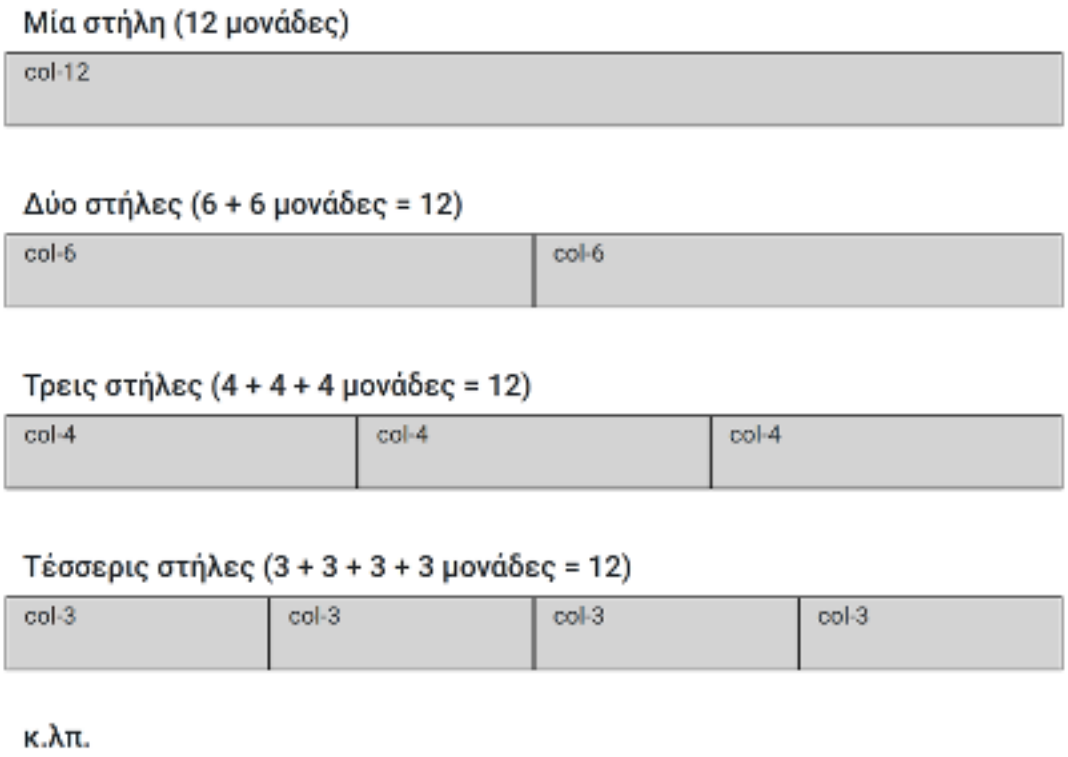
Οι γραμμές (.row) που βρίσκονται μέσα στο .container χρησιμοποιούνται για να περιέχουν στήλες. Το περιεχόμενο βρίσκεται μέσα σε στήλες (.col) και οι στήλες μέσα σε γραμμές (.row). Οι γραμμές πρέπει να περιέχουν μόνο στήλες και τίποτε άλλο.

Οι στήλες διαστασιολογούνται αυτόματα, και το διαθέσιμο πλάτος, το εύρος του .container, μοιράζεται εξίσου και στις 3 στήλες του παραδείγματός μας.

Αν ορίσουμε το πλάτος μιας στήλης, τότε το υπόλοιπο πλάτος θα κατανεμηθεί αυτόματα στις υπόλοιπες, οι οποίες θα προσαρμοστούν ανάλογα.

Για να ορίσουμε το πλάτος χρησιμοποιούμε τις διαθέσιμες μονάδες πλάτους. Οι μονάδες αυτές δηλώνονται με το κατάλληλο **επίθημα**.

Οι διαθέσιμες «μονάδες πλάτους» σε μια γραμμή (.row) είναι 12, τις οποίες μπορεί η γραμμή να μοιράσει στις στήλες της. Για παράδειγμα, μια στήλη πλάτους 6 μονάδων δηλώνεται ως .col-6 και θα έχει πάντα το μισό από το διαθέσιμο πλάτος της γραμμής (6/12 = 1/2). Οι υπόλοιπες 6 μονάδες που περισσεύουν θα κατανεμηθούν στις υπόλοιπες στήλες.

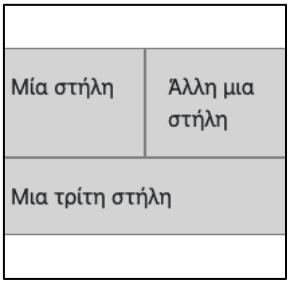


Εικόνα 6.4 *Bootstrap: ενδεικτικά επιθήματα για τον καθορισμό του πλάτους των στηλών.*

Μπορούμε και να ζητήσουμε το πλάτος της στήλης να προσαρμόζεται αυτόματα στο περιεχόμενο, χρησιμοποιώντας το επίθημα -auto, με την κλάση .col-auto.

6.4.1 Προσαρμόσιμες κλάσεις

Αν το grid μας έχει την ίδια διάταξη για όλες τις διαστάσεις του φυλλομετρητή, μπορούμε να χρησιμοποιήσουμε τις κλάσεις .col και .col-* Τέτοιο είναι το grid που φαίνεται στην **Εικόνα *6.2*** και στην **Εικόνα *6.3***. Αυτό το grid θα έχει την ίδια διάταξη, δηλαδή τρεις ισόπαχες στήλες, η μια δίπλα στην άλλη, εκτός και αν δεν αρκεί ο χώρος που είναι διαθέσιμος στον container. Σε αυτή την περίπτωση, η γραμμή θα αναδιπλωθεί και θα εμφανίσει τα κελιά της σε δύο σειρές (**Εικόνα *6.5***).



Εικόνα 6.5. *Η γραμμή αναδιπλώνεται σε δύο σειρές για να χωρέσει στο διαθέσιμο πλάτος.*

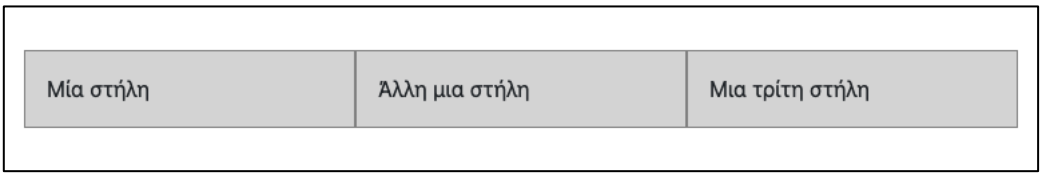
Το πιο ισχυρό χαρακτηριστικό του grid, όμως, είναι πως μπορούμε να ορίσουμε διαφορετικές διατάξεις για διαφορετικά πλάτη. Αυτό το κάνουμε με τα κατάλληλα επιθήματα breakpoint.

Για παράδειγμα, η παρακάτω δήλωση

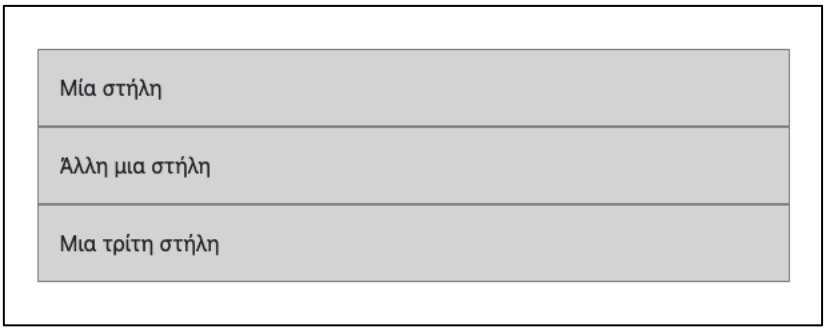
```
<div class="container">
    <div class="row">
```

```
            <div class="col-md">Μια στήλη</div>
            <div class="col-md">Άλλη μια στήλη</div>
            <div class="col-md">Μια τρίτη στήλη</div>
        </div>
    </div>
```

θα εμφανίσει τρεις στήλες όταν η διαθέσιμη διάσταση θα είναι από md και πάνω (**Εικόνα *6.6***), ενώ για κάτω από md θα εμφανίσει τις στήλες στοιβαγμένες τη μια κάτω από την άλλη (***Εικόνα 6.7***).



Εικόνα 6.6. *Οι τρεις στήλες με επίθημα -md εμφανίζονται σε μια σειρά όσο το διαθέσιμο πλάτος είναι μεγαλύτερο από md.*



Εικόνα 6.7. *Αν το διαθέσιμο πλάτος γίνει μικρότερο από md, η γραμμή αναδιπλώνεται. Το αποτέλεσμα είναι οι τρεις στήλες της γραμμής να εμφανίζονται η καθεμία σε δική της σειρά.*

6.4.2 Offset

Με την κλάση .offset-*-* μπορούμε να μετακινήσουμε τη στήλη προς τα δεξιά. Για παράδειγμα, η παρακάτω δήλωση θα μετακινήσει τη δεύτερη στήλη κατά τρεις μονάδες (offset-md-**3**) προς τα δεξιά. Το offset αυτό έχει ισχύ μόνο για διάσταση md και πάνω (offset-**md**-3). Όταν ο διαθέσιμος χώρος γίνει κάτω από md, τότε το offset δεν θα εφαρμοστεί (**Εικόνα *6.8***).

```
<div class="container">
    <div class="row">
        <div class="col-md">Μια στήλη</div>
        <div class="col-md offset-md-3">Άλλη μια στήλη</div>
        <div class="col-md">Μια τρίτη στήλη</div>
    </div>
</div>
```

Εικόνα 6.8. *Στη δεύτερη στήλη έχει εφαρμοστεί offset τριών μονάδων.*

Η κλάση .offset-0 ή .offset-*-0 αφαιρεί το offset.

6.4.3 Διάταξη των στηλών

Για να διατάξουμε κάθετα τα στοιχεία μιας γραμμής, τις στήλες δηλαδή, μπορούμε να ορίσουμε πως ο "container" μας ή το "col" μας είναι flex container με την κλάση **.d-flex**. Στη συνέχεια, θα χρησιμοποιήσουμε κάποια από τις κατάλληλες κλάσεις .align-items-start, .align-items-end, .align-items-center. Αντίστοιχα, μπορούμε να διατάξουμε και μεμονωμένες στήλες με .align-self-start, .align-self-end, .align-self-center.

Για την οριζόντια διάταξη οι αντίστοιχες κλάσεις είναι .justify-content με επιθήματα τα -start, -end -center, -around, -between.

6.4.4 Ταξινόμηση

Με τις κλάσεις .order ελέγχουμε την οπτική διάταξη των στηλών. Οι κλάσεις αυτές είναι προσαρμόσιμες και μπορούν να δηλωθούν ανά breakpoint. Υπάρχουν επίσης και οι κλάσεις .order-first και .order-last που τοποθετούν το στοιχείο στην πρώτη ή την τελευταία θέση.

6.4.5 Αναδίπλωση

Αν περισσότερες από 12 στήλες έχουν τοποθετηθεί σε μία γραμμή, τότε κάθε ομάδα στηλών θα αναδιπλωθεί.

Μπορούμε να προκαλέσουμε στοχευμένα την αναδίπλωση τοποθετώντας ένα στοιχείο με πλάτος 100%, με την κλάση .w-100. Με το βοήθημα d (display) μπορούμε να ορίσουμε και σε ποιο breakpoint θα εφαρμόζεται αυτή η συμπεριφορά, π.χ. να ορίσουμε πως από md και πάνω δεν θα εφαρμόζεται η αναδίπλωση της γραμμής (display: none):

```
<div class="d-md-none w-100"></div>
```

6.5 Βοηθήματα

Εκτός από το d του παραπάνω παραδείγματος, είναι διαθέσιμες και μια σειρά από άλλες κλάσεις με βοηθητικό ρόλο. Μερικές από αυτές περιγράφονται παρακάτω.

6.5.1 Περιθώρια

Μια σειρά από κλάσεις μας βοηθούν να ορίσουμε περιθώρια. Η ονοματολογία τους είναι απλή: Αρχίζουν με m για εξωτερικό περιθώριο (margin) και με p για εσωτερικό περιθώριο (padding).

Έπειτα ακολουθεί προαιρετικά η πλευρά, που μπορεί να είναι t, b, s, e, για πάνω, κάτω, αριστερά (start) και δεξιά (end) αντίστοιχα, ή x για αριστερά και δεξιά και y για πάνω και κάτω. Οπότε για αριστερό padding θα έχουμε μια κλάση που ξεκινά με ps-....

Στη συνέχεια μπορούμε προαιρετικά να ορίσουμε το breakpoint (ps-md-) για περιθώριο που θα εφαρμόζεται σε διάσταση md και πάνω.

Τέλος, πρέπει να δώσουμε το μέγεθος του περιθωρίου με έναν αριθμό από 0 μέχρι 5, που θέτει κάποιο προκαθορισμένο μέγεθος, π.χ. ps-md-5 θα ορίσει αριστερό padding όταν είμαστε σε md και πάνω, και το μέγεθος του padding θα είναι 5 (οι μονάδες 1-5 έχουν προκαθορισμένα μεγέθη στην Bootstrap). Μπορούμε να ορίσουμε και -auto για margin: auto.

6.5.2 Πλάτος

Για τον καθορισμό του πλάτους ενός στοιχείου (width), η Bootstrap διαθέτει τις κλάσεις w-*. Το * μπορεί να είναι 25, 50, 75, 100 ή auto, και ορίζει το μέγεθος του στοιχείου σε ποσοστό %.

6.5.3 Περίγραμμα

Ομοίως, οι κλάσεις border-* ορίζουν περίγραμμα, όπου * είναι top, end, bottom, start.

6.6 Φόρμες

Η Bootstrap παρέχει πολύ καλή υποστήριξη όσον αφορά τον ορισμό φορμών και πετυχαίνει να εμφανίζονται με μεγάλη ομοιομορφία σε διαφορετικούς φυλλομετρητές. Η διακόσμηση των γραφικών στοιχείων των φορμών, δηλαδή η προσαρμογή τους με τη χρήση της CSS, είναι μια διαβόητα δύσκολη διαδικασία. Είναι δηλαδή δύσκολο να πετύχουμε να φαίνονται με τον ίδιο ή έστω με σχεδόν ίδιο τρόπο τα γραφικά στοιχεία μιας φόρμας, δηλαδή τα στοιχεία input, select και τα κουμπιά, σε διαφορετικούς φυλλομετρητές, και η CSS δεν μας δίνει εύκολο έλεγχο της εμφάνισής τους. Χρησιμοποιώντας την Bootstrap, μπορούμε να βασιστούμε στην προεργασία που έχουν κάνει οι συγγραφείς της βιβλιοθήκης και να πετύχουμε μια αρμονική εμφάνιση της φόρμας μας σε διάφορους φυλλομετρητές. Επιπλέον, η Bootstrap παρέχει και βοηθήματα που είναι χρήσιμα στον έλεγχο εγκυρότητας της φόρμας μας και ευκολίες για σύνθετα στοιχεία ελέγχου, συνδυασμούς δηλαδή κουμπιών με input box. Όλα αυτά σε συνοδεία με το πολύ ισχυρό grid της Bootstrap.

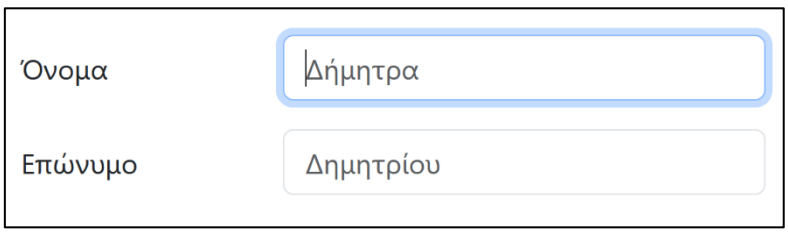
6.6.1 Input

Στα στοιχεία input αρκεί να συμπεριλάβουμε την κλάση .form-control για να εμφανιστούν με τη μορφή της Bootstrap:

```
<input type="text" class="form-control">
```

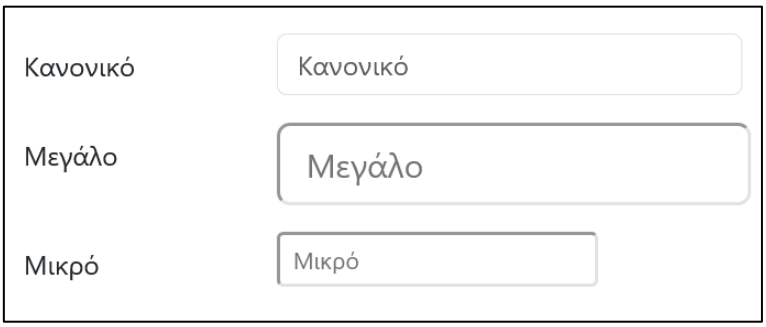
Συμπληρωματικά, οι κλάσεις .form-control-sm και .form-control-lg αλλάζουν το μέγεθος του στοιχείου σε μικρότερο ή μεγαλύτερο αντίστοιχα (**Εικόνα *6.9***).

```
<div class="container">
  <form>
    <div class="row mb-3">
      <label for="inputOnoma" class="col-2 col-form-label">
        Όνομα
      </label>
      <div class="col-10">
        <input type="text" class="form-control" id="inputOnoma"
         placeholder="Δήμητρα">
      </div>
    </div>
    <div class="row mb-3">
      <label for="inputEponymo" class="col-2 col-form-label">
        Επώνυμο
      </label>
      <div class="col-10">
        <input type="text" class="form-control" id="inputEponymo"
         placeholder="Δημητρίου">
      </div>
    </div>
  </form>
</div>
```



Εικόνα 6.9 *Η φόρμα Bootstrap με δύο στοιχεία input, με placeholder και label.*

Το μέγεθος των στοιχείων της φόρμας μπορεί να αλλάξει εύκολα χρησιμοποιώντας τις κλάσεις .form-control-lg και .form-control-sm (**Εικόνα *6.10*.** Οι κλάσεις .form-control-lg και .form-control-sm μάς επιτρέπουν να αλλάξουμε εύκολα το μέγεθος ενός στοιχείου της φόρμας μας.).



Εικόνα 6.10. *Οι κλάσεις .form-control-lg και .form-control-sm μάς επιτρέπουν να αλλάξουμε εύκολα το μέγεθος ενός στοιχείου της φόρμας μας.*

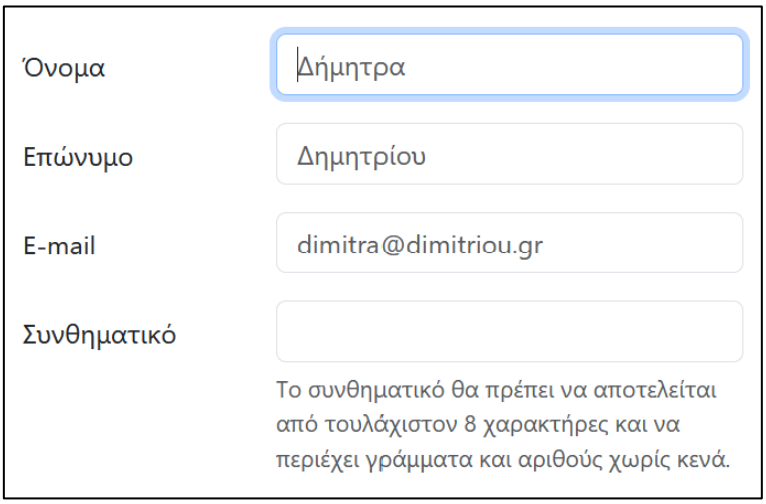
6.6.2 Κείμενα στη φόρμα

Στις φόρμες μπορούμε να προσθέσουμε κείμενα με την κλάση .form-text. Η κλάση αυτή δίνει σε βοηθητικά κείμενα (μέσα σε div ή span) της φόρμας στιλ που να ταιριάζει με την αισθητική της Bootstrap και προσθέτει αυτόματα ένα εξωτερικό περιθώριο.

```
<div class="container">
  <form>
    <div class="row mb-3">
      <label for="inputOnoma" class="col-2 col-form-label">
        Όνομα
      </label>
      <div class="col-10">
        <input type="text" class="form-control" id="inputOnoma"
         placeholder="Δήμητρα">
      </div>
    </div>
    <div class="row mb-3">
      <label for="inputEponymo" class="col-2 col-form-label">
        Επώνυμο
      </label>
      <div class="col-10">
        <input type="text" class="form-control" id="inputEponymo"
         placeholder="Δημητρίου">
      </div>
    </div>
    <div class="row mb-3">
      <label for="inputEmail" class="col-2 col-form-label">
        E-Mail
      </label>
      <div class="col-10">
        <input type="email" class="form-control" id="inputEmail">
      </div>
    </div>
    <div class="row mb-3">
      <label for="inputPassword" class="col-2 col-form-label">
        Συνθηματικό
      </label>
      <div class="col-10">
```

```
        <input type="password" class="form-control" id="inputPassword"
aria-describedby="passwordHelp">
      </div>
      <div id="passwordHelp" class="form-text col-8 offset-4">
        Το συνθηματικό θα πρέπει να αποτελείται από τουλάχιστον 8
        χαρακτήρες και να περιέχει γράμματα και αριθούς χωρίς κενά.
      </div>
    </div>
  </form>
</div>
```

Η παραπάνω φόρμα φαίνεται στην ***Εικόνα 6.11.*** Στη φόρμα έχει προστεθεί με την κλάση .form-text ένα επεξηγηματικό κείμενο για το πεδίο του συνθηματικού. Παρατηρήστε ότι το πεδίο εισαγωγής συνθηματικού συνδέεται με το επεξηγηματικό κείμενο μέσω του γνωρίσματος **aria-describedby**. Το γνώρισμα αυτό περιγράφεται στις Οδηγίες Προσβασιμότητας Περιεχομένου (Web Content Accessibility Guidelines – WCAG). Η τιμή του γνωρίσματος στο παράδειγμά μας είναι 'passwordHelp', δηλαδή το id του στοιχείου που περιγράφει το στοιχείο εισαγωγής συνθηματικού. Γενικότερα, το πεδίο αυτό μπορεί να χρησιμοποιηθεί για να συσχετίσει κείμενο με κάποιο στοιχείο. Η χρήση του δεν περιορίζεται στις φόρμες, ούτε και στην Bootstrap, αλλά μπορεί να χρησιμοποιηθεί γενικότερα.

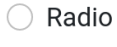


Εικόνα 6.11. *Στη φόρμα έχει προστεθεί με την κλάση .form-text ένα επεξηγηματικό κείμενο για το πεδίο του συνθηματικού.*

Radio / Checkbox

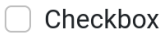
Για τα στοιχεία αυτά χρησιμοποιούμε την κλάση .form-check-input και τυλίγουμε τα στοιχεία μέσα σε ένα div με κλάση .form-check

```
<div class="form-check">
  <input class="form-check-input" type="radio"  ... >
</div>
```



Εικόνα 6.12 *Το στοιχείο **radio** στην Bootstrap.*

```
<div class="form-check">
  <input class="form-check-input" type="checkbox"   ... >
</div>
```



Εικόνα 6.13 *Το στοιχείο **checkbox** στην Bootstrap.*

Μπρούμε να έχουμε και την ιδιαίτερη εμφάνιση «διακόπτη» προσθέτοντας στο εξωτερικό την κλάση .form-switch

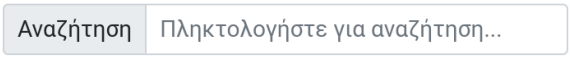
```
<div class="form-check form-switch">
  <input class="form-check-input" type="radio"   ... >
</div>
```

*Εικόνα 6.14 Το στοιχείο **switch** της Bootstrap.*

6.6.3 Input group

Τα input group είναι ένα τρόπος να δημιουργήσουμε σύνθετα στοιχεία εισόδου συνδυάζοντας, π.χ., ένα input text με κείμενο ή κουμπί.

```
<div class="input-group">
    <span class="input-group-text" id="...">Αναζήτηση</span>
    <input type="text" class="form-control" placeholder="Πληκτρολογήστε
για αναζήτηση...">
</div>
```



Εικόνα 6.15 *Το **input group** της Bootstrap.*

6.7 Έλεγχος εγκυρότητας

Η Bootstrap παρέχει βοηθήματα για τη μορφοποίηση των μηνυμάτων που προκύπτουν από τον έλεγχο εγκυρότητας μιας φόρμας. Έχουμε δει στην ενότητα 3.10 πώς γίνεται με απλά μέσα ο έλεγχος εγκυρότητας μιας φόρμας. Οι σύγχρονοι φυλλομετρητές παρέχουν έναν ενιαίο μηχανισμό που υποστηρίζει τον έλεγχο εγκυρότητας, το constraint validation API. Το API αυτό είναι βέβαια ανεξάρτητο από την Bootstrap και μπορεί να χρησιμοποιηθεί σε οποιαδήποτε φόρμα. Με το constraint validation API ο έλεγχος εγκυρότητας μπορεί να γίνει είτε για κάθε στοιχείο ξεχωριστά είτε για ολόκληρη τη φόρμα. Θα χρειαστεί να χρησιμοποιήσουμε JavaScript, η οποία παρουσιάζεται στα επόμενα κεφάλαια. Ο αναγνώστης που δεν είναι εξοικειωμένος με τη γλώσσα αυτή μπορεί να συμβουλευτεί πρώτα το κεφάλαιο Κεφάλαιο 7 και στη συνέχεια να επιστρέψει σε αυτή εδώ την ενότητα. Ωστόσο, αναγνώστες που έχουν κάποια εξοικείωση με τον προγραμματισμό θα μπορέσουν να συνεχίσουν, καθώς η γλώσσα χρησιμοποιείται σε πολύ απλά παραδείγματα.

Για ολόκληρη τη φόρμα ο έλεγχος εγκυρότητας γίνεται:

- κατά την υποβολή της φόρμας,
- με κλήση μιας από τις μεθόδους reportValidity() ή checkValidity()

Αντίστοιχα, ο έλεγχος εγκυρότητας σε ένα μόνο στοιχείο μπορεί να γίνει:

- με κλήση μιας από τις μεθόδους reportValidity() ή checkValidity() στο εκάστοτε στοιχείο αντί για όλη τη φόρμα.

Η διαφορά της reportValidity() από την checkValidity() είναι πως η πρώτη στέλνει επιπλέον ένα μήνυμα σχετικά με τον έλεγχο εγκυρότητας που μπορεί να προβάλει ο φυλλομετρητής. Στο παρακάτω παράδειγμα, με πάτημα στο κουμπί «Έλεγχος» θα κληθεί η reportValidity() για όλη τη φόρμα '#myForm'

```
<form action="..." id="myForm">
   <input type="text" class="form-control" id="inputOnoma" required>
```

```
    <button type="button" class="btn btn-primary"
onclick="document.querySelector('#myForm').reportValidity()" > Έλεγχος
</button>
</form>
```

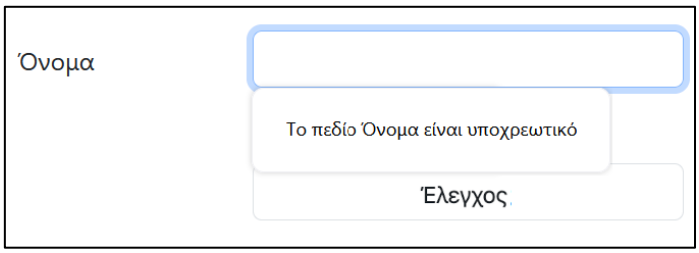
Το αποτέλεσμα θα είναι να πραγματοποιηθεί ο έλεγχος εγκυρότητας και, αν ο χρήστης δεν έχει δώσει τιμή στο πεδίο 'inputOnoma', που είναι απαιτούμενο (**required**), τότε θα εμφανιστεί στον φυλλομετρητή ένα μήνυμα που να προειδοποιεί τον χρήστη (***Εικόνα 6.16***).

Εικόνα 6.16 *Ο χρήστης έχει πατήσει το κουμπί «Έλεγχος» χωρίς όμως να συμπληρώσει τιμή στο πεδίο Όνομα, το οποίο είναι απαραίτητο (required). Κλήθηκε η συνάρτηση reportValidity() του constraint validation API και εμφανίστηκε ένα μήνυμα που προειδοποιεί τον χρήστη.*

Μπορούμε να αλλάξουμε το μήνυμα που θα εμφανίζεται για αυτό το πεδίο, χρησιμοποιώντας τη συνάρτηση του constraint validation API, setCustomValidity(). Μπορούμε, για παράδειγμα, να ορίσουμε το δικό μας μήνυμα για το πεδίο '#inputOnoma':

```
<form action="..." id="myForm">
   <input type="text" class="form-control" id="inputOnoma" required>
   <button type="button" class="btn btn-primary"
onclick="document.querySelector('#myForm').reportValidity()" > Έλεγχος
</button>
...
<script>
   document.querySelector('#inputOnoma').setCustomValidity(«Το πεδίο Όνομα
είναι υποχρεωτικό»)
</script>
```

Τώρα θα εμφανιστεί το δικό μας, προσαρμοσμένο μήνυμα (**Εικόνα *6.17***).



Εικόνα 6.17 *Το μήνυμα έχει αλλάξει με τη συνάρτηση setCustomValidity() του constraint validation API.*

6.7.1 Έλεγχος εγκυρότητας με την Bootstrap

Μπορούμε να χρησιμοποιήσουμε την Bootstrap σε συνδυασμό με το constraint validation API για να εμφανίσουμε τα δικά μας μηνύματα σε περίπτωση παραβίασης κάποιου κανόνα εγκυρότητας, διαμορφωμένα με το στιλ της Bootstrap.

Θα παρακάμψουμε λοιπόν τον μηχανισμό που παρέχει ο φυλλομετρητής για την επικύρωση. Με την παράκαμψη, ο μηχανισμός αυτός συνεχίζει να υπάρχει, αλλά δεν θα λειτουργήσει αυτόματα, με την υποβολή της φόρμας. Θα χρειαστεί να τον εκτελέσουμε γράφοντας δική μας JavaScript.

Για να ορίσουμε τη συμπεριφορά αυτή, προσθέτουμε αρχικά το γνώρισμα **novalidate** στη φόρμα, με το οποίο ο φυλλομετρητής θα υποβάλει τη φόρμα ακόμη και αν δεν ικανοποιείται ο έλεγχος εγκυρότητας, π.χ. αν δεν έχουμε βάλει τιμή σε ένα πεδίο που είναι **required**.

```
<form action="..." id="myForm" ... novalidate>
...
   <input type="text" class="form-control" id="inputOnoma" required>
   ...
   <button type="submit" class="btn btn-primary"> Υποβολή </button>
   ...
```

Χωρίς το **novalidate**, η φόρμα δεν θα υποβληθεί αν ο χρήστης αφήσει το πεδίο **inputOnoma** κενό. Προσθέτοντας όμως το γνώρισμα **novalidate** στη φόρμα μας, ο έλεγχος παρακάμπτεται και η φόρμα θα υποβληθεί ακόμη και αν δεν τηρούνται οι περιορισμοί εγκυρότητας. Πλέον, πρέπει να δημιουργήσουμε τον δικό μας μηχανισμό για τον έλεγχο και την υποβολή της φόρμας.

Για να μάθουμε πότε θα υποβληθεί η φόρμα, γράφουμε, χρησιμοποιώντας την JavaScript, μια συνάρτηση χειρισμού του συμβάντος «υποβολή» (το 'submit' event):

```
document.querySelector('#myForm').addEventListener('submit', (event) => {
    //σώμα της συνάρτησης χειρισμού του συμβάντος 'submit'
})
```

Οι χειριστές συμβάντων περιγράφονται αναλυτικά στο κεφάλαιο Κεφάλαιο 10. Στο παραπάνω παράδειγμα, όταν υποβληθεί η φόρμα, τότε θα εκτελεστεί μια ανώνυμη συνάρτηση επιστροφής (callback) που παίρνει ένα όρισμα, το 'event'. Στο σώμα αυτής της συνάρτησης επιστροφής χρειάζεται τώρα

- να ελέγξουμε αν τα δεδομένα που εισήγαγε ο χρήστης στη φόρμα είναι έγκυρα,
- αν δεν είναι έγκυρα, να διακόψουμε την υποβολή της φόρμας,
- να υποδείξουμε στον χρήστη τα στοιχεία της φόρμας που περιέχουν τα μη έγκυρα δεδομένα.

Βήμα 1

Ελέγχουμε αν τα δεδομένα που έδωσε ο χρήστης είναι έγκυρα, καλώντας συναρτήσεις του constraint validation API που διαθέτει ο φυλλομετρητής. Αυτό μπορούμε να το κάνουμε για όλα τα στοιχεία της φόρμας, καλώντας, για παράδειγμα, την **document.querySelector('#myForm').checkValidity()**.

Βήμα 2

Αν η φόρμα δεν περάσει τον έλεγχο, δηλαδή αν η **checkValidity()** επιστρέψει **false**, αυτό θα σημαίνει πως σε κάποιο από τα πεδία της φόρμας έχουν εισαχθεί μη έγκυρα δεδομένα. Θα πρέπει λοιπόν να σταματήσουμε τη συνέχιση της υποβολής της φόρμας. Αυτό το πετυχαίνουμε καλώντας τη συνάρτηση **preventDefault()** του συμβάντος (event) (άλλο ένα παράδειγμα υπάρχει στην ενότητα 10.6.2 Ο μηχανισμός φυσαλίδας). Η **preventDefault()** σταματάει τη διαδικασία υποβολής της φόρμας, που είναι και η προκαθορισμένη, η default, πράξη που ακολουθεί έπειτα από ένα γεγονός υποβολής της φόρμας (submit event). Συνεπώς, η φόρμα αυτή δεν θα υποβληθεί στον εξυπηρετητή.

```
document.querySelector('#myForm').addEventListener('submit', (event) => {
    //σώμα της συνάρτησης χειρισμού του συμβάντος 'submit'
    if !(document.querySelector('#myForm').checkValidity()) {
       event.preventDefault()
    }
})
```

Βήμα 3

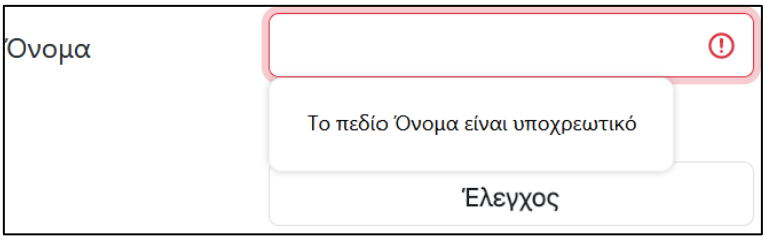
Σε αυτό το σημείο έχουμε διαπιστώσει πως τα δεδομένα που έχει εισαγάγει ο χρήστης δεν είναι έγκυρα και έχουμε σταματήσει την υποβολή της. Τώρα χρειάζεται να δείξουμε στο χρήστη τα στοιχεία που έχουν πρόβλημα. Θα χρησιμοποιήσουμε πάλι το μηχανισμό του constraint validation API. Ο φυλλομετρητής, στα στοιχεία που είναι έγκυρα, προσθέτει αυτόματα την ψευδοκλάση **:valid** ενώ στα μη έγκυρα την **:invalid**. Η

Bootstrap έχει έτοιμα στιλ για τις δύο αυτές ψευδοκλάσεις, αλλά, για να φανούν αυτά τα στιλ της Bootstrap, χρειάζεται οι δύο ψευδοκλάσεις να έχουν πρόγονο με κλάση **.was-validated** (δηλαδή να περικλείονται από κάποιο στοιχείο του DOM που έχει την κλάση **.was-validated**). Αλλιώς, αν τα στιλ για **:valid** και **:invalid** εμφανίζονταν πάντα, τότε θα φαίνονταν ακόμα και όταν η φόρμα απλά έχει φορτωθεί και δεν έχουν ακόμα συμπληρωθεί, π.χ., τα στοιχεία **required**.

Για να εμφανίσουμε λοιπόν τα σχετικά στιλ, προσθέτουμε στη φόρμα μας την κλάση **.was-validated**, που είναι και πρόγονος των στοιχείων, ανεξάρτητα από το αν το βήμα #2 επέστρεψε true ή false, και έτσι θα προβληθούν τα στιλ της Bootstrap για τα στοιχεία που είναι έγκυρα και μη έγκυρα.

```
document.querySelector('#myForm').addEventListener('submit', (event) => {
    //σώμα της συνάρτησης χειρισμού του συμβάντος 'submit'
    if !(document.querySelector('#myForm').checkValidity()) {
       event.preventDefault()
    }

    document.querySelector('#myForm').classList.add('was-validated')
})
```

Στην **Εικόνα *6.18*** φαίνεται το αποτέλεσμα. Το πεδίο «Όνομα» εμφανίζεται με κόκκινο περίγραμμα, καθώς αυτό είναι το στιλ της Bootstrap για τα στοιχεία φόρμας που επιλέγονται με τον επιλογέα CSS '.was-validated :invalid'.



Εικόνα 6.18 *Στο πεδίο «Όνομα» έχει εφαρμοστεί από το constraint validation API η ψευδοκλάση :invalid.*

6.7 Ερωτήσεις αυτοαξιολόγησης

1. Με το grid της Bootstrap έχουμε στη διάθεσή μας μέχρι 12 στήλες.
   – Σωστό/Λάθος
2. Ποιες από τις παρακάτω δεν είναι έγκυρες κλάσεις για τον ορισμό υποδοχέα της Bootstrap;
   1. .container
   2. .container-xs
   3. .container-xl
   4. .container-fluid
3. Στις γραμμές (.row) του grid της Bootstrap μπορούμε να τοποθετήσουμε υποδοχέα της Bootstrap.
   – Σωστό/Λάθος
4. Με την Bootstrap, με τις παρακάτω δηλώσεις κλάσεων, μπορούμε να κατασκευάσουμε μια γραμμή με τρεις στήλες που κατανέμονται ισομερώς στο διαθέσιμο πλάτος (επιλέξτε όσα ισχύουν).
   1. col-3 col-3 col-3
   2. col-4 col-4 col-4
   3. col col col
   4. col-1 col-1
5. Για να ορίσουμε αριστερό περιθώριο 8px χρησιμοποιούμε την κλάση
   1. .ms-2
   2. .mt-2
   3. .ml-2
   4. .me-2
6. Για να ορίσουμε padding 8px στην αριστερή πλευρά ενός στοιχείου με την Bootstrap χρησιμοποιούμε την κλάση
   1. .ms-2
   2. .ps-2
   3. .ml-2
   4. .mr-2
7. Για να ορίσουμε σε ένα div με μέγεθος ίσο με το 75% του άμεσου προγόνου του με την Bootstrap, χρησιμοποιούμε την κλάση
   1. w-75
   2. width-75
   3. w-75%
   4. width-75%
8. Για να ορίσουμε πως μια στήλη της Bootstrap θα είναι flex container χρησιμοποιούμε την κλάση
   1. .flex
   2. .display-flex
   3. .d-flex
   4. δεν χρειάζεται καμία κλάση, η Bootstrap χρησιμοποιεί ούτως ή άλλως το flexbox
9. Οι κλάσεις ".col col-sm-10" σε μια στήλη της Bootstrap θα έχουν σαν αποτέλεσμα
   1. Από μέγεθος οθόνης SM και κάτω η στήλη να πιάνει 1/10 του διαθέσιμου χώρου στο grid.
   2. Από μέγεθος οθόνης SM και πάνω η στήλη να πιάνει 1/10 του διαθέσιμου χώρου στο grid.
   3. Από μέγεθος οθόνης SM και πάνω η στήλη να πιάνει 10/12 του διαθέσιμου χώρου στο grid.
   4. Από μέγεθος οθόνης SM και κάτω η στήλη να πιάνει 10/12 του διαθέσιμου χώρου στο grid.
10. Με ποια κλάση της Bootstrap μπορούμε να μετακινήσουμε μια στήλη κατά δύο μονάδες προς τα δεξιά;
    1. margin-column-2
    2. col-offset-2
    3. col-md-2
    4. offset-2
11. Ο συνδυασμός κλάσεων "border border-1" ορίζει
    1. περίγραμμα πλάτους 1 (δηλαδή 0.25rem)
    2. περίγραμμα μόνο στην πάνω πλευρά (στην πλευρά “1”)
    3. περίγραμμα μόνο σε 1 στήλη
    4. περίγραμμα χρωματισμένο με το χρώμα 1
12. Με την κλάση .input-group της Bootstrap

1. καταφέρνουμε αυτό που χωρίς την Bootstrap θα πετυχαίναμε με το στοιχείο fieldset.
2. ορίζουμε ομάδες από checkbox ή radio buttons.
3. κατασκευάζουμε σύνθετα στοιχεία που αποτελούνται από απλά input σε συνδυασμό με κουμπιά-κείμενο.
4. συνδυάζουμε input με datalist.

13. Η ιδιότητα του στοιχείου form, με την οποία μπορούμε να ζητήσουμε να παρακαμφθεί έλεγχος εγκυρότητας, είναι η:
    1. validate="false"
    2. validate="disabled"
    3. validate="check"
    4. novalidate
14. Τα στοιχεία μιας φόρμας μπορούν να εκτελέσουν τη συνάρτηση checkValidity() του Validation API. Ωστόσο, η εκτέλεση της συνάρτησης αυτής μπορεί να γίνει και από τη φόρμα για όλα τα στοιχεία της.
    – Σωστό/Λάθος
15. Για να εμφανιστούν σωστά τα στιλ των στοιχείων μιας φόρμας που έχουν χαρακτηριστεί :valid ή :invalid από το Validation API του φυλλομετρητή χρησιμοποιούμε την κλάση της Bootstrap
    1. novalidate
    2. validated
    3. validate
    4. was-validated
16. Για να εμφανιστεί το μήνυμα επικύρωσης ενός στοιχείου που βρίσκεται μέσα σε ένα κλάσης valid-feedback χρειάζεται ο πρόγονος να είναι τοποθετημένος με position: relative.
    – Σωστό/Λάθος

6.8 Βιβλιογραφία και Αναφορές

Η κύρια πηγή για το πλαίσιο Bootstrap είναι η τεκμηρίωση που παρέχεται στην ιστοσελίδα της ίδιας της Bootstrap. Η τεκμηρίωση εκεί είναι εκτενής και αναλυτική και συνοδεύεται από πλήθος παραδειγμάτων.

Επιπλέον, οι συγγραφείς έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το κεφάλαιο. Το σχετικό μάθημα είναι το «Προχωρημένα θέματα ανάπτυξης ιστοσελίδων».

Ξενόγλωσση

Συλλογικό (2011). *Bootstrap Documentation*. Bootstrap. Ανακτήθηκε στις 14 Σεπτεμβρίου 2022 από https://github.com/twbs/bootstrap

Κεφάλαιο 7. Εισαγωγή στην JavaScript

Σύνοψη

*Στο κεφάλαιο αυτό ξεκινάμε την εισαγωγή στη γλώσσα προγραμματισμού **JavaScript**, που αποτελεί μια βασική τεχνολογία του διαδικτυακού προγραμματισμού και μια από τις γλώσσες προγραμματισμού που έχουν ευρύτατη χρήση όχι μόνο στον φυλλομετρητή αλλά και έξω από αυτόν, για προγραμματισμό του εξυπηρετητή (στο περιβάλλον Node.js, που παρουσιάζεται στο κεφάλαιο 11), καθώς και για ανάπτυξη μη διαδικτυακών διαδραστικών εφαρμογών στην επιφάνεια εργασίας του υπολογιστή μας (με το περιβάλλον electron.js).*

Η γλώσσα JavaScript είναι αντικείμενο αυτού και των επόμενων κεφαλαίων. Στο κεφάλαιο αυτό γίνεται η πρώτη γνωριμία με την JavaScript. Ξεκινάμε με μια ιστορική ανασκόπηση της εξέλιξης της γλώσσας, στη συνέχεια θα δούμε βασικά στοιχεία σύνταξης της γλώσσας, όπως τους τρόπους δήλωσης μεταβλητών και τους βασικούς τελεστές που χρησιμοποιούνται σε μαθηματικές εκφράσεις. Στη συνέχεια θα συζητήσουμε τρόπους για να ενσωματώσουμε κώδικα JavaScript σε μια ιστοσελίδα. Θα δούμε πώς η JavaScript έχει πρόσβαση στις ιδιότητες του global object (window) και στο DOM (document object model).

Στο επόμενο κεφάλαιο θα προχωρήσουμε με τους βασικούς τύπους δεδομένων και τις κύριες εντολές της JavaScript.

Προαπαιτούμενη γνώση

Για την κατανόηση αυτού του κεφαλαίου είναι επιθυμητή η γνώση της HTML (κεφάλαια 2 και 3) και της CSS (κεφ 4 και 5), καθώς συζητείται και παρουσιάζεται με παραδείγματα η χρήση της γλώσσας JavaScript ως προγραμματιστική διεπαφή με τα στοιχεία μιας ιστοσελίδας.

7.1 Εισαγωγή – Ιστορικό σημείωμα

Στην πρώτη αυτή ενότητα θα δώσουμε μια γενική εισαγωγή με κύρια την ιστορική διάσταση και τον ρόλο της JavaScript. Η JavaScript είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου, όπως η Python, η Java κ.λπ. Ακολουθεί ένα πρότυπο, το πρότυπο ECMAScript. Το επίσημο όνομά της είναι **ECMAScript** αλλά έχει καθιερωθεί να χρησιμοποιούμε το εμπορικό όνομα που αρχικά τής δόθηκε, JavaScript.

Η ιστορία της αρχίζει το 1995 από τον μηχανικό της εταιρείας Netscape Brendan Eich. Ο Netscape ήταν την εποχή εκείνη ο πιο δημοφιλής φυλλομετρητής. Στο πλαίσιο του Netscape αναπτύχθηκε η πρώιμη έκδοση της JavaScript ως γλώσσα προγραμματισμού μιας ιστοσελίδας, αφού είχε προκύψει η ανάγκη οι ιστοσελίδες να γίνουν πιο διαδραστικές.

Η JavaScript μαζί με την HTML και τη CSS, που είδαμε σε προηγούμενα κεφάλαια, είναι οι βασικές τεχνολογίες για να αναπτύσσουμε εφαρμογές στον παγκόσμιο ιστό, στο διαδίκτυο. Σύντομα θα δούμε πώς συνδέεται η JavaScript με τις προηγούμενες δύο τεχνολογίες που έχουμε ήδη δει.

Όταν κατεβάζουμε ένα αρχείο HTML (που συνοδεύεται από ένα φύλλο μορφοποίησης CSS, είτε σε ξεχωριστό αρχείο είτε στο ίδιο το αρχείο), μπορεί μέσα στο ίδιο το αρχείο HTML ή σε ένα συνοδευτικό διασυνδεδεμένο αρχείο να υπάρχει κώδικας JavaScript. Ο κώδικας αυτός, λοιπόν, πρέπει να εκτελεστεί. Αυτό γίνεται μέσα στον ίδιο τον φυλλομετρητή, ο οποίος διαθέτει μια «μηχανή» που μπορεί να κάνει συντακτική ανάλυση και μεταγλώττιση της JavaScript σε μια γλώσσα που εκτελείται από τον υπολογιστή (γλώσσα μηχανής).

Επίσης, θα πρέπει να σημειώσουμε ότι η εξαιρετική δημοφιλία αυτής της τεχνολογίας έχει οδηγήσει ώστε να αποκτήσει και άλλες χρήσεις, εκτός από γλώσσα που χρησιμοποιείται στον φυλλομετρητή, στις εφαρμογές του διαδικτύου.

Για παράδειγμα, με το runtime περιβάλλον Node.js η γλώσσα αυτή χρησιμοποιείται ως μια παραδοσιακή γλώσσα, στην ουσία για προγραμματισμό εφαρμογών στον εξυπηρετητή, να δρομολογεί αιτήματα για παροχή ιστοσελίδων, να συνδέει τις εφαρμογές αυτές με μια βάση δεδομένων, το σύστημα αρχείων κ.λπ.

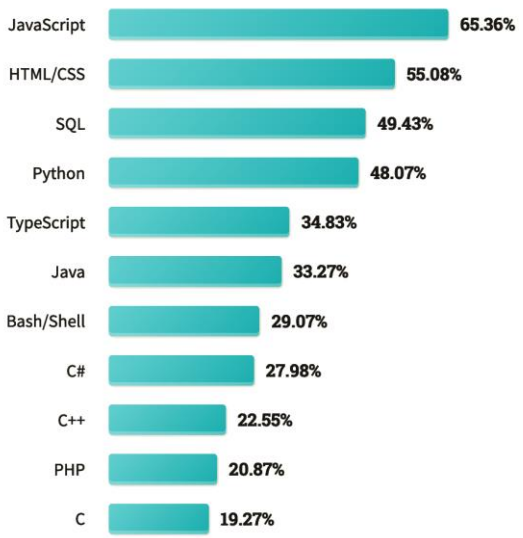
7.1.1 Κύρια χαρακτηριστικά της JavaScript

Ας δούμε τα κύρια χαρακτηριστικά της JavaScript.

- Η JavaScript χαρακτηρίζεται καταρχάς ως δυναμική γλώσσα με **ασθενή διαχείριση τύπων δεδομένων** (weakly typed language). Η JavaScript δεν είναι πολύ αυστηρή ως προς τους τύπους δεδομένων και συχνά παραβλέπει ασυμβατότητα τύπων σε εκφράσεις κάνοντας αυτόματα μετατροπές τύπων. Για παράδειγμα, η έκφραση 5 + "10" στις περισσότερες γλώσσες προγραμματισμού θα οδηγήσει σε σφάλμα, αφού δεν επιτρέπεται η πρόσθεση ασύμβατων τύπων δεδομένων (στην Python θα παραγάγει TypeError). Στην JavaScript θα δώσει το αποτέλεσμα "510", που έχει παραχθεί από τη μετατροπή του αριθμού 5 στη συμβολοσειρά “5” και στη συνέχεια στη συνένωση των δύο συμβολοσειρών.
- Συχνά αυτό το χαρακτηριστικό της JavaScript είναι αιτία κριτικής στη γλώσσα, αφού μπορεί να προκαλέσει ανεπιθύμητα σφάλματα που είναι δύσκολο να διαγνωστούν. Για αντιμετώπιση του προβλήματος αυτού έχουν δημιουργηθεί διάλεκτοι της γλώσσας, όπως η TypeScript που χειρίζονται τα δεδομένα με πιο αυστηρό τρόπο.
- Η JavaScript είναι αντικειμενοστραφής γλώσσα ως προς την αρχιτεκτονική της, τα περισσότερα δεδομένα της είναι αντικείμενα, όμως ακολουθεί ένα ιδιαίτερο μηχανισμό κληρονομικότητας που στηρίζεται σε «πρωτότυπα», όπως θα περιγραφεί σε επόμενο κεφάλαιο.
- Επίσης, η JS υποστηρίζει διαφορετικά προγραμματιστικά παραδείγματα, όπως τον προγραμματισμό με συμβάντα (**event-based programming**), συναρτησιακό προγραμματισμό (**functional programming**), διαθέτει δομές για να προγραμματίσουμε με αντικείμενα (**object-oriented programming**).
- Η JavaScript διαθέτει προγραμματιστικές διεπαφές (APIs) για χειρισμό κειμένων, πινάκων, ημερομηνιών, τυπικών εκφράσεων (regular expression), καθώς και κάτι που μας ενδιαφέρει ιδιαίτερα, διαθέτει προγραμματιστική διεπαφή προς το DOM (Document Object Model), δηλαδή προς αντικείμενα που αντιστοιχούν στα στοιχεία ενός εγγράφου HTML που προβάλλεται από τον φυλλομετρητή.
- Δεν περιλαμβάνει διεπαφή για αλληλεπίδραση απευθείας με τον χρήστη, π.χ. δεν υπάρχει αντίστοιχη εντολή print(), επίσης δεν διαθέτει διεπαφή για δικτύωση, για μόνιμη αποθήκευση στο σύστημα αρχείων ή σε βάση δεδομένων ή γραφική διεπαφή.

7.1.2 Χρήση της JavaScript

Για να έχουμε μια ιδέα του πόσο δημοφιλής είναι η JavaScript, δεν έχουμε παρά να εξετάσουμε ένα πρόσφατο ετήσιο developer survey της δημοφιλούς ιστοσελίδας StackOverflow, που έχει ευρύτατη χρήση στην κοινότητα των προγραμματιστών. Στο ερώτημα για την πιο δημοφιλή τεχνολογία οι απαντήσεις που δόθηκαν κατά την ετήσια επισκόπηση του 2022 φαίνονται στην **Εικόνα *7.1***.



Εικόνα 7.1 *Δημοφιλία της JavaScript σύμφωνα με την ετήσια επισκόπηση του StackOverflow.* *https://survey.stackoverflow.co/2022/#technology*

Μάλιστα, σύμφωνα με την ετήσια αυτή έρευνα, η γλώσσα JavaScript παραμένει στη θέση τής πιο δημοφιλούς τεχνολογίας για 10η συνεχή χρονιά, με βάση τις απαντήσεις των μελών του StackOverflow. Θα πρέπει ακόμη

να παρατηρηθεί ότι η TypeScript, που αποτελεί διάλεκτο της JavaScript, βρίσκεται επίσης στις πρώτες θέσεις των πιο δημοφιλών τεχνολογιών.

Αν εξετάσουμε ιστορικά την εξέλιξη της JavaScript, θα παρατηρήσουμε ότι αρχικά η γλώσσα αυτή δημιουργήθηκε για να υποστηρίζει και να επιτρέπει διαδραστικότητα σε έγγραφα HTML (ιστοσελίδες). Παραδείγματος χάρη, χρησιμοποιούσαμε τα σκριπτ της JavaScript για να ελέγχουμε την εγκυρότητα δεδομένων που δίνει ο χρήστης σε μια φόρμα, να υπάρχει διαδραστικότητα σε συμβάντα που προκαλεί ο χρήστης όταν χειρίζεται το ποντίκι, το πληκτρολόγιο, την οθόνη αφής κ.λπ.

Σήμερα, εκτός από αυτή τη χρήση, που παραμένει πολύ σημαντική, υπάρχει και μια άλλη χρήση που θα τη δούμε περισσότερο στη συνέχεια. Από το τέλος της δεκαετίας του ’90 και μετά, με την έλευση της AJAX (*“Asynchronous JavaScript and XML”*), που επέτρεπε μια ιστοσελίδα να κάνει ασύγχρονες κλήσεις προς τον εξυπηρετητή, άρχισε η ανάπτυξη νέου τύπου διαδικτυακών εφαρμογών, οι εφαρμογές μοναδικής ιστοσελίδας (single page applications, SPA), οι οποίες στέλνουν και λαμβάνουν δεδομένα από τον εξυπηρετητή χωρίς να απαιτείται ξαναφόρτωμα της σελίδας. Τέτοιες εφαρμογές, που συναντάμε όλο και πιο συχνά στο διαδίκτυο, είναι εφαρμογές μέσων κοινωνικής δικτύωσης, χαρτογραφικού περιεχομένου, σχεδίασης, διαχείρισης κειμένων και φύλλων εργασίας, και μοιάζουν όλο και περισσότερο με διαδραστικές εφαρμογές που έχουμε συνηθίσει στην επιφάνεια εργασίας των υπολογιστών μας.

Αυτές δεν ακολουθούν το μοντέλο των ιστοσελίδων που φορτώνονται διαδοχικά στον φυλλομετρητή, αλλά συνήθως είναι σελίδες που κατεβαίνουν, εγκαθίστανται και, με διαδοχικές κλήσεις στον εξυπηρετητή, ανανεώνεται το περιεχόμενό τους. Αυτού του τύπου οι εφαρμογές είναι γραμμένες σε JavaScript και είναι συνήθως αρκετά σύνθετες, με πολλές γραμμές κώδικα. Το τμήμα της εφαρμογής που τρέχει στον εξυπηρετητή παίζει σε αυτή την περίπτωση λιγότερο σημαντικό ρόλο και, κυρίως, παρέχει υπηρεσίες όπως σύνδεση με βάση δεδομένων και άλλες πηγές μόνιμης αποθήκευσης κ.λπ.

7.1.3 Εκδόσεις της JavaScript

Ας εξετάσουμε ιστορικά τις πιο σημαντικές εκδόσεις της JavaScript ή ECMAScript, όπως είναι το επίσημο όνομά της:

- Η πρώτη σημαντική έκδοση ήταν η **ES3 (JavaScript 3)**, που ήταν το πρώτο πλήρες πρότυπο, στα πλαίσια του οποίου εισήχθησαν σημαντικές νέες λειτουργίες, όπως κανονικές εκφράσεις (regular expression literals), χειριστής εξαιρέσεων try/catch κ.λπ.
- Το 2009 ορίζεται το πρότυπο **ES5 (JavaScript 5)**, που υποστηρίζουν σχεδόν όλες οι εκδόσεις των φυλλομετρητών και περιλαμβάνει τις **συναρτήσεις πινάκων** map() reduce() filter() forEach, υποστήριξη JSON, ορισμό getters και setters για πρόσβαση στις ιδιότητες αντικειμένων κ.λπ.
- Η πιο σημαντική νέα έκδοση ήταν η **ES6 (ES2015)**, που θεωρείται η πιο ανεπτυγμένη και ώριμη έκδοση της γλώσσας. Η έκδοση αυτή περιλαμβάνει μεγάλες βελτιώσεις όπως τον μηχανισμό υποσχέσεων (*promises*) για ασύγχρονη εκτέλεση, εισαγωγή modules, ορισμό κλάσεων με τη λέξη *class*, ορισμό μεταβλητών με εμβέλεια στο block (*let*), συναρτήσεις βέλη (arrow functions **() => { }**), ορισμό συμβολοσειρών με ενσωμάτωση μεταβλητών (template literal), τον τελεστή spread **...*** ορισμό προκαθορισμένων τιμών σε παραμέτρους συναρτήσεων, ορισμό του τύπου symbol κ.λπ.
- Έκτοτε σε ετήσια βάση ανακοινώνονται καινούργιες εκδόσεις της γλώσσας που φέρουν το όνομα του έτους, ES2020, ES2021 κ.λπ. Όμως, μετά την έκδοση ES2015 οι αλλαγές είναι λιγότερο ρηξικέλευθες και ούτως ή άλλως η πλήρης υιοθέτηση πρόσφατων αλλαγών από τους φυλλομετρητές απαιτεί χρόνο.

Ουσιαστικά, η έκδοση ES6 είναι το τρέχον πρότυπο της JavaScript και αυτή θα χρησιμοποιήσουμε στο βιβλίο αυτό, όμως επειδή υπάρχει πολύ εγκατεστημένο λογισμικό και σε προηγούμενες εκδόσεις, κυρίως την ES5, θα πρέπει να λάβουμε υπόψη μας ποια ήταν η λειτουργία της γλώσσας σε προηγούμενες εκδόσεις. Θα πρέπει να αναφερθεί επίσης ότι υπάρχουν ειδικά εργαλεία για μεταγλώττιση μεταξύ εκδόσεων (transpilers), που επιτρέπουν να γράψουμε κώδικα στην έκδοση ES6 και να τον μεταγλωττίσουμε σε ES5 ώστε να μπορεί να εκτελεστεί σχεδόν σε όλους του φυλλομετρητές.

Η JavaScript σήμερα συνοδεύεται από ένα πλούσιο οικοσύστημα από εργαλεία, βιβλιοθήκες και πλαίσια ανάπτυξης. Η πιο σημαντική βιβλιοθήκη είναι jQuery η οποία υπάρχει από το 2006 αν και τείνει να υποχωρήσει αφού οι νέες εκδόσεις της JavaScript έχουν υποκαταστήσει πολλές από τις λειτουργίες της jQuery. Αξίζει να κάνουμε επίσης αναφορά σε κάποια από τα framework για το front-end που χρησιμοποιούνται ευρύτατα σήμερα όπως είναι το React, το Angular το Vue.js και τα οποία βασίζονται στην JavaScript.

7.1.4 Μηχανές εκτέλεσης κώδικα JavaScript

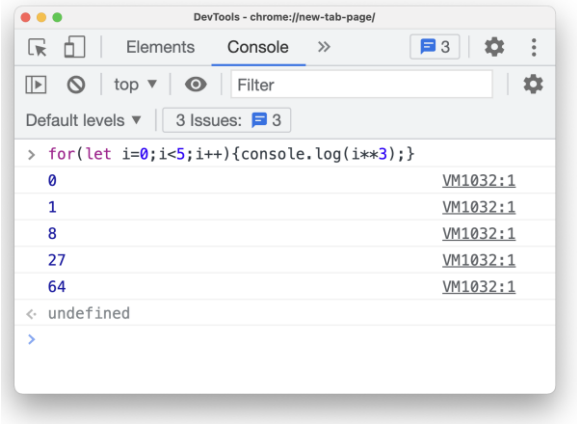
Ας δούμε τις βασικές τεχνολογίες που υπάρχουν για εκτέλεση κώδικα JavaScript. Θα πρέπει να αναφερθεί ότι κάθε φυλλομετρητής έχει διάφορες διεργασίες που εκτελούνται όταν αρχίσει το φόρτωμα μιας ιστοσελίδας. Αρχίζει μια διεργασία που κάνει συντακτική ανάλυση του κώδικα HTML και κτίζει το DOM, όπως είδαμε σε προηγούμενο κεφάλαιο. Αυτή η διεργασία, σε συνδυασμό με τον συντακτικό αναλυτή των σχετικών φύλλων CSS, τροφοδοτεί τη διεργασία που ονομάζεται *rendering engine*, που είναι η διεργασία που εμφανίζει την ιστοσελίδα στο παράθυρο του φυλλομετρητή. Επίσης, υπάρχει η *JavaScript engine* που αναλαμβάνει τη διερμηνεία και την εκτέλεση του κώδικα JavaScript που σχετίζεται με το συγκεκριμένο έγγραφο HTML. Μάλιστα, οι σύγχρονοι φυλλομετρητές δεν διερμηνεύουν την JavaScript αλλά τη μεταγλωττίζουν με τεχνολογία "just in time" για καλύτερη απόδοση.

Οι κυρίες μηχανές JavaScript που έχουμε στους φυλλομετρητές σήμερα είναι η **V8** που υπάρχει στον *Chrome* καθώς και στη **Node.js** (Ενότητα 11), που είναι το περιβάλλον εκτέλεσης εφαρμογών JavaScript στον εξυπηρετητή. Μια άλλη μηχανή JavaScript είναι η **SpiderMonkey** που είναι ενσωματωμένη στον φυλλομετρητή *FireFox*, καθώς και η **JavaScriptCore** (Nitro) που βρίσκεται στον φυλλομετρητή *Safari*.

7.2 Ανάπτυξη και αποσφαλμάτωση κώδικα JavaScript

Για την ανάπτυξη και την αποσφαλμάτωση κώδικα JavaScript μπορούμε να χρησιμοποιήσουμε οποιοδήποτε περιβάλλον ανάπτυξης, όπως το *Visual Studio Code*. Η εκτέλεση του κώδικα θα γίνει στο περιβάλλον του φυλλομετρητή αφού ενσωματώσουμε τον κώδικα σε μια άδεια σελίδα HTML. Τα μηνύματα σφάλματος εμφανίζονται στην κονσόλα στα εργαλεία προγραμματιστή του φυλλομετρητή (ενεργοποιούνται συνήθως με function-F12).

Αν επιθυμούμε να πειραματιστούμε με τμήματα κώδικα, μπορούμε να χρησιμοποιήσουμε την κονσόλα του φυλλομετρητή, την οποία μάλιστα μπορούμε να αποσπάσουμε σε ξεχωριστό παράθυρο και εκεί να γράψουμε κώδικα, ο οποίος εκτελείται αμέσως. Ένα παράδειγμα εκτέλεσης ενός βρόχου εκτύπωσης των κύβων των αριθμών 0...4 φαίνεται στην **Εικόνα *7.2*** για την κονσόλα του φυλλομετρητή Chrome.



Εικόνα 7.2 *Εκτέλεση κώδικα JavaScript στην κονσόλα του Chrome.*

Μια εναλλακτική επιλογή που επιτρέπει δοκιμές με σκριπτ της JavaScript σε περιβάλλον Node.js είναι η χρήση της εφαρμογής RunJS που διατίθεται σε δωρεάν έκδοση. Η εφαρμογή αυτή παρουσιάζει στην τυπική της μορφή δύο παράθυρα, στο ένα γράφουμε κώδικα και στο δίπλα βλέπουμε το αποτέλεσμα, όπως φαίνεται στην **Εικόνα *7.3***, για το κλασικό πρόγραμμα helloWorld.

Εικόνα 7.3 *Εκτέλεση κώδικα JavaScript στο περιβάλλον RunJS.*

Η σύνταξη της γλώσσας είναι κοινή στα δύο αυτά περιβάλλοντα (εξάλλου και τα δύο χρησιμοποιούν την ίδια μηχανή JS, τη V8), αν και στη δεύτερη περίπτωση λείπουν κάποιες διεπαφές, όπως η διεπαφή στα DOM και BOM, που παρέχονται στο περιβάλλον του φυλλομετρητή, όπως θα περιγραφεί στη συνέχεια του κεφαλαίου αυτού.

7.3 Σύνταξη ενός προγράμματος JavaScript

Όπως μπορείτε να διαπιστώσετε από τα παραδείγματα της προηγούμενης ενότητας, η σύνταξη της JavaScript μοιάζει με άλλες γλώσσες προγραμματισμού, όπως η C, Java κ.λπ. Είναι ευαίσθητη στη διαφορά κεφαλαίων και μικρών γραμμάτων και ο κώδικας του προγράμματος γράφεται με κωδικοποίηση UTF-8, κάτι που σημαίνει ότι μπορούμε να εισαγάγουμε ελληνικούς χαρακτήρες ως χαρακτήρες σε συμβολοσειρές (string) χωρίς πρόβλημα.

Ο χαρακτήρας **;** ορίζεται ως τερματικός χαρακτήρας εντολής, που σημαίνει ότι μπορούν να συνυπάρξουν πολλαπλές εντολές σε μια γραμμή. Όμως, σε αντίθεση με άλλες γλώσσες προγραμματισμού, η JavaScript δεν θεωρεί τον τερματικό χαρακτήρα υποχρεωτικό και μπορεί να παραληφθεί αν ο συντακτικός αναλυτής μπορεί να συνάγει σαφώς πού τελειώνει η κάθε εντολή. Γενικά είναι καλή πρακτική να χρησιμοποιείται πάντα ο τερματικός χαρακτήρας στο τέλος εντολών, αν και κάποιοι προγραμματιστές προτιμάνε να τον χρησιμοποιούν μόνο όταν είναι απαραίτητος, δηλαδή όπου η παράλειψή του μπορεί να προκαλέσει σύγχυση στον συντακτικό αναλυτή.

Τα σχόλια μιας γραμμής αρχίζουν με τους χαρακτήρες // ενώ σχόλια πολλών γραμμών αρχίζουν με την ακολουθία χαρακτήρων **/*** και τερματίζουν με την ακολουθία ***/**

```
// σχόλιο μιας γραμμής

/* αυτό είναι ένα
σχόλιο που εκτείνεται
σε πολλές γραμμές */
```

Τα ονόματα σταθερών, μεταβλητών, συναρτήσεων, κλάσεων, γνωρισμάτων αντικειμένων κ.λπ. (identifiers) στην JavaScript πρέπει να ακολουθήσουν κάποιους κανόνες. Τα ονόματα αυτά πρέπει να περιέχουν γράμματα, αριθμούς ή τα σύμβολα _ και **$**. Όμως, ένα όνομα δεν πρέπει να αρχίζει με αριθμητικό χαρακτήρα.

Συνεπώς, επιτρεπτά ονόματα μεταβλητών είναι τα **_**, **$**, **_1**, **$1**, **a1** ενώ δεν είναι επιτρεπτά τα **1a**, **1$**, **12**, **1_**.

Επίσης, υπάρχουν κάποιες δεσμευμένες λέξεις που δεν μπορούν να χρησιμοποιηθούν, όπως τα ονόματα εντολών ή δομών **if**, **else**, **const**, **null**, **continue**, **this**, **while**, **false**, **return**, **throw**, **with**, **break**, **in**, **true**, **case**, **for**, **instanceof**, **try**, **catch**, **let**, **super**, **typeof**, **class**, **function**, **new**, **switch**, **var** κ.λπ.

Θεωρητικά, θα μπορούσε να χρησιμοποιηθεί οποιοσδήποτε χαρακτήρας UTF-8 στο όνομα μιας μεταβλητής, άρα δεν είναι λάθος να γράψουμε:

```
μεταβλητή = 10;
console.log(μεταβλητή)
```

Όμως, δεν θεωρείται καλή πρακτική για τη μεταφερσιμότητα του κώδικά μας να χρησιμοποιήσουμε χαρακτήρες εκτός ASCII (λατινικό αλφάβητο) στα ονόματα μεταβλητών.

Κατά σύμβαση, τα ονόματα των μεταβλητών, των γνωρισμάτων αντικειμένων και των συναρτήσεων ακολουθούν τη σημειογραφία καμήλας camelNotation, δηλαδή αρχίζουν με μικρό γράμμα και, όταν περιέχουν πολλές λέξεις (συνήθως της αγγλικής γλώσσας), κάθε αρχικό των επόμενων λέξεων γράφεται με κεφαλαίο, γραφή που θυμίζει τις καμπούρες της καμήλας, εξού και το όνομα.

Ο τύπος των δεδομένων που θα εκχωρηθούν σε μια μεταβλητή ή μια σταθερά δεν ορίζεται κατά τη δήλωση της μεταβλητής, αλλά προκύπτει από την αρχική τιμή που η μεταβλητή θα πάρει. Η διάλεκτος TypeScript προσπαθεί να διορθώσει αυτό το πρόβλημα.

Θα πρέπει να δηλώσουμε τις μεταβλητές ή τις σταθερές πριν τις χρησιμοποιήσουμε με τις λέξεις **let**, **const**, **var**, όπως θα δούμε στη συνέχεια.

7.3.1 Δήλωση μεταβλητών let, const, var

Σε αυτή την ενότητα θα δούμε τρόπους που διαθέτουμε για δήλωση μεταβλητών ή σταθερών.

Μέχρι την έκδοση ES5 της JavaScript είχαμε μόνο έναν τρόπο να δηλώνουμε μεταβλητές, με τη λέξη-κλειδί **var** (για variable).

```
var myName = 'Nikos';
var myAge =  75;
```

Με την έκδοση ES6 προστέθηκαν δύο ακόμη λέξεις κλειδιά για δήλωση μεταβλητών, η **let** (let it be…) και η **const** (constant).

```
let myName = 'Nikos';
let myAge =  75;
const pi =  3.14;
```

Ποιος ο λόγος για εισαγωγή των δύο αυτών τρόπων ορισμού μεταβλητών, υπάρχουν διαφορές μεταξύ τους;

Η απάντηση είναι ότι υπάρχουν σημαντικές διαφορές και η πρόταση εξαρχής που έχουμε να κάνουμε είναι να ξεχάσουμε τη var και να χρησιμοποιούμε την const στις περισσότερες περιπτώσεις και τη let στις υπόλοιπες, που είναι μόνο για τις περιπτώσεις μεταβλητών που αναφέρονται σε **πρωτογενείς τύπους δεδομένων** (Number, String, Boolean κ.λπ.) των οποίων η τιμή πρόκειται να αλλάξει.

Ας εξετάσουμε στη συνέχεια τις κύριες διαφορές μεταξύ let και var. Αυτές εντοπίζονται στα εξής:

- Διαφορές στη δυνατότητα επαναδήλωσης μιας μεταβλητής (στη var επιτρέπεται, στη let όχι).
- Διαφορές στην εμβέλεια (η var έχει εμβέλεια και έξω από το μπλοκ στο οποίο δηλώνεται, ενώ η let όχι).
- Διαφορές στη δυνατότητα πρόσβασης στη μεταβλητή πριν τη δήλωσή της (στη var επιτρέπεται, η τιμή είναι αρχικά undefined, στη let αυτό δεν επιτρέπεται).

Ας δούμε μερικά σχετικά παραδείγματα:

7.3.2 Πολλαπλές δηλώσεις

Ο παρακάτω κώδικας είναι αποδεκτός:

```
var myName = 'Nikos';
var myName = 'Kostas';
```

Αντίθετα, ο παρακάτω κώδικας δίνει σφάλμα:

```
let myName = 'Nikos';
let myName = 'Kostas';
> SyntaxError: Identifier 'myname' has already been declared
```

Βεβαίως, μπορούμε να αλλάξουμε την τιμή μιας μεταβλητής που έχει δηλωθεί με let, χωρίς όμως να την ξαναδηλώσουμε:

```
let myName = 'Nikos';
myName = 'Kostas';
```

Αντίθετα, δεν μπορούμε να αλλάξουμε μια μεταβλητή που έχει δηλωθεί με τη λέξη-κλειδί const αν η τιμή της

είναι πρωτογενούς τύπου.

```
const myName = 'Nikos';
myName = 'Kostas';
> TypeError: Assignment to constant variable.
```

Θα πρέπει να προσέξουμε όμως ότι η δήλωση με τη λέξη-κλειδί const μεταβλητών που αναφέρονται σε **τύπους δεδομένων αναφοράς**, όπως τα αντικείμενα (object) ή οι πίνακες, δεν μας αποκλείει από τη δυνατότητα να τροποποιήσουμε στη συνέχεια το περιεχόμενο αυτών των αντικειμένων.

Για παράδειγμα, ο παρακάτω κώδικας είναι απόλυτα αποδεκτός:

```
const ourNames = ['Nikos', 'Kostas', 'Maria'];
ourNames[0] = 'Katerina';
console.log(ourNames);
> ["Katerina", "Kostas", "Maria"]
```

7.3.3 Εμβέλεια μεταβλητών let

Μια σημαντική διαφορά μεταξύ let και var είναι ότι η let ορίζει εμβέλεια της μεταβλητής μόνο στο μπλοκ στο οποίο έχει δηλωθεί (μπλοκ μπορεί να θεωρηθεί μια συνάρτηση αλλά και μια εντολή if, for κ.λπ.), ενώ η var έχει εμβέλεια σε ολόκληρο τον κώδικα.

Ας συγκρίνουμε το αποτέλεσμα των δύο αυτών παραδειγμάτων:

```
for(var i = 0; i<10; i++) {
}
console.log(i)
> 10

for(let i = 0; i<10; i++) {
}
console.log(i)
> ReferenceError: i is not defined
```

Στο δεύτερο παράδειγμα που η μεταβλητή i ορίστηκε με χρήση της λέξης κλειδί let, αυτή δεν είχε εμβέλεια εκτός του μπλοκ for μέσα στο οποίο ορίστηκε, και για αυτό πήραμε μήνυμα σφάλματος.

7.3.4 Κλήση μεταβλητής πριν τη δήλωσή της

Η let δεν μπορεί να χρησιμοποιηθεί πριν δηλωθεί, ενώ η var μπορεί να χρησιμοποιηθεί (*hoisting*) και έχει την τιμή undefined.

```
console.log('x=', typeof x, x); // x undefined NaN
console.log('y=', typeof y, y); // ReferenceError
var x=5;
let y=10;
```

7.3.5 Τελεστές και εκφράσεις

Έχουμε ήδη δώσει παραδείγματα εντολών εκχώρησης τιμής σε μεταβλητή. Αυτή γίνεται με χρήση του τελεστή =:

```
μεταβλητή = έκφραση;
```

Σε μια έκφραση μπορούν να χρησιμοποιηθούν οι δυαδικοί τελεστές αριθμητικών πράξεων που συναντώνται και σε άλλες γλώσσες προγραμματισμού: “+” για πρόσθεση, “-” για αφαίρεση, “*” για πολλαπλασιασμό, “/” για διαίρεση, “%” για υπόλοιπο διαίρεσης, ενώ από την έκδοση ES2016 προστέθηκε ο τελεστής “**” για ύψωση σε δύναμη. Επίσης, σε μια έκφραση μπορεί να χρησιμοποιηθούν οι μοναδιαίοι τελεστές:

- **a++** για απόδοση τιμής και αύξηση κατά 1
- **a--** για απόδοση τιμής και μείωση κατά 1

- **++a** για αύξηση τιμής κατά 1 και απόδοση τιμής στη συνέχεια
- **--a** για μείωση τιμής κατά 1 και απόδοση τιμής στη συνέχεια

καθώς και οι συντομογραφίες εκφράσεων:

- **a += b** Αύξηση της τιμής του a κατά b, δηλαδή a = a + b
- **a -= b** Μείωση της τιμής του a κατά b, δηλαδή a = a - b;

7.3.6 Τελεστές πράξεων δυαδικών αριθμών

Αν έχουμε δύο μεταβλητές a, b που εκφράζουν δυαδικούς αριθμούς, υπάρχουν οι εξής τελεστές που επιτρέπουν πράξεις μεταξύ τους:

- a & b Bitwise AND ανάμεσα στις μεταβλητές
- a | b Bitwise OR ανάμεσα στις μεταβλητές
- a ^ b Bitwise XOR ανάμεσα στις μεταβλητές
- ~a Bitwise NOT της μεταβλητής
- a << b Αριστερή ολίσθηση κατά b θέσεις των bit της μεταβλητής a
- a >> b Δεξιά ολίσθηση κατά b θέσεις των bit της μεταβλητής a

Θυμίζουμε ότι η αριστερή ολίσθηση στους δυαδικούς αριθμούς είναι ισοδύναμη με πολλαπλασιασμό με τη βάση 2, ενώ η δεξιά ολίσθηση είναι ισοδύναμη με διαίρεση με 2.

Στο επόμενο κεφάλαιο θα δούμε άλλους τελεστές που μπορούμε να χρησιμοποιήσουμε σε μια έκφραση, όπως λογικούς τελεστές, αλλά και τον τριαδικό τελεστή, καθώς και τελεστές σύγκρισης που χρησιμοποιούνται σε εκφράσεις που συνηθίζονται στις εντολές επιλογής.

Άσκηση
Ποιο το αποτέλεσμα;

```
let c = 5;
console.log(c++);
console.log(++c);
```

Απάντηση:

```
> 5
> 7
```

Αυτό γιατί η πρώτη εντολή εκτύπωσης στην κονσόλα τυπώνει την τιμή της μεταβλητής c πριν την αύξησή της, ενώ η δεύτερη μετά τη νέα αύξησή της.

7.3.7 Διεπαφή με τον χρήστη

Όπως ήδη αναφέρθηκε, η JavaScript δεν έχει εγγενώς διεπαφή για επικοινωνία με τον χρήστη, όπως άλλες γλώσσες προγραμματισμού. Εξάλλου, είναι προορισμένη να συνεργάζεται στενά με την HTML/CSS για τον σκοπό αυτό. Όμως, συχνά χρειαζόμαστε είτε στο περιβάλλον του εξυπηρετητή είτε στο περιβάλλον του φυλλομετρητή το πρόγραμμά μας να τυπώσει την κατάστασή του, ή να ζητήσει από τον χρήστη κάποια δεδομένα. Στη συνέχεια κάνουμε μια σύντομη αναφορά σε τρόπους που μπορεί να χρησιμοποιήσουμε για τον σκοπό αυτό.

console.log(μήνυμα)

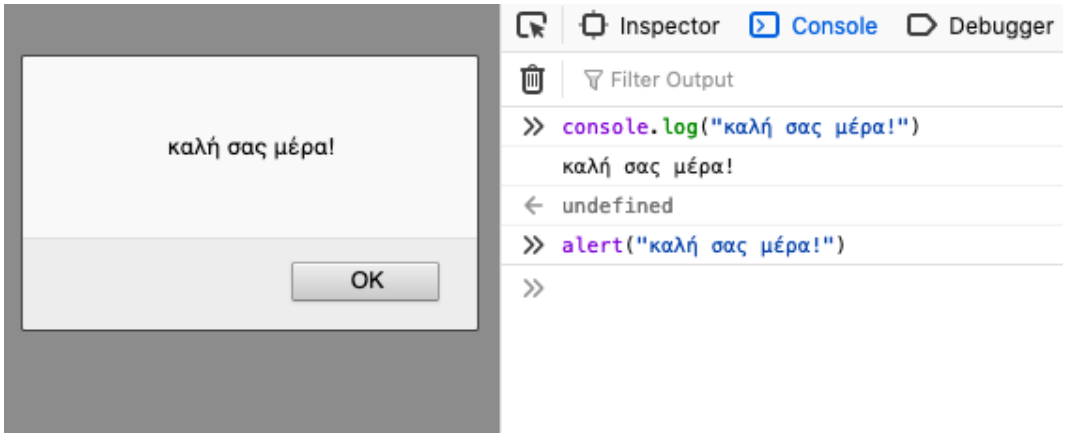
Ο πιο συνηθισμένος τρόπος είναι να χρησιμοποιήσουμε το αντικείμενο console που ανήκει στο global object και να καλέσουμε τη μέθοδο console.log(). Το αντικείμενο αυτό έχει οριστεί, όπως θα δούμε στη συνέχεια, και στο καθολικό επίπεδο της Node.js με παρόμοια συμπεριφορά με αυτή του φυλλομετρητή.

Στα ορίσματα της συνάρτησης αυτής μπορούμε να περάσουμε εκφράσεις ή συμβολοσειρές και να πάρουμε τα αποτελέσματα στην κονσόλα. Η κονσόλα βρίσκεται στα εργαλεία ανάπτυξης όλων των φυλλομετρητών (Chrome, Firefox, Safari κ.λπ.).

```
console.log("Καλημέρα");
> "Καλημέρα"
```

alert(μήνυμα)

Ένας εναλλακτικός τρόπος, μόνο για το περιβάλλον του φυλλομετρητή όμως, είναι να χρησιμοποιήσουμε τη μέθοδο alert() του window object.

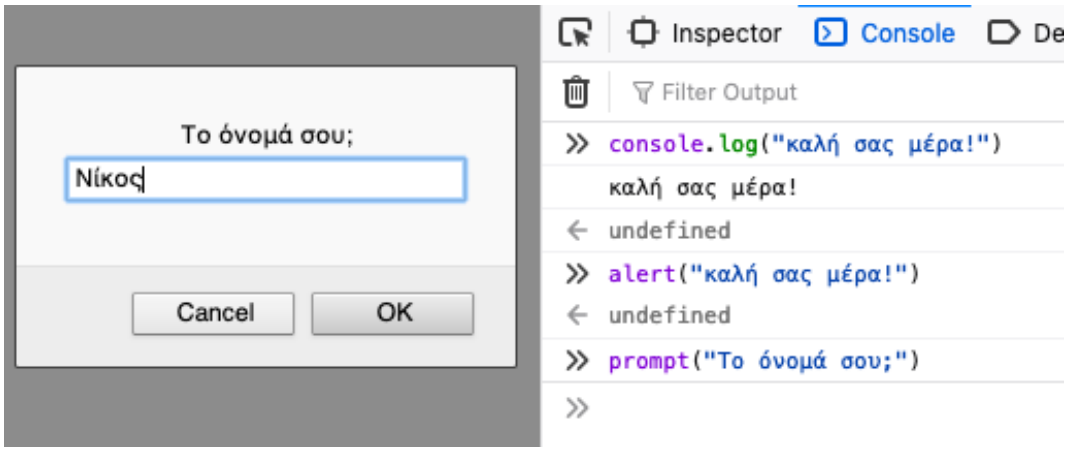


Εικόνα 7.4 *Η 'alert()' όπως εκτελείται στον φυλλομετρητή Firefox.*

Αυτή παράγει ένα αναδυόμενο μονοτροπικό παράθυρο (modal) με το μήνυμα, όπως βλέπουμε στην **Εικόνα *7.4***, από την κονσόλα του Firefox.

prompt()

Παρόμοια είναι η χρήση της μεθόδου prompt() του global object, η οποία μέσω ενός αναδυόμενου μονοτροπικού παραθύρου ζητάει μια τιμή από τον χρήστη. Παράδειγμα στην **Εικόνα *7.5***.



Εικόνα 7.5 *Η prompt() όπως εκτελείται στον φυλλομετρητή Firefox.*

Επικοινωνία μέσω της HTML

Εκτός από τους παραπάνω, υπάρχουν πολλοί τρόποι να χρησιμοποιηθεί η ίδια η ιστοσελίδα για επικοινωνία με τον χρήστη, όπως για παράδειγμα με τροποποίηση ενός στοιχείου με την ιδιότητα element.textContent ή element.innerHTML, αλλά και να πάρουμε στοιχεία από τον χρήστη μέσω στοιχείων <input> μιας φόρμας, όπως έχουμε δει στο κεφάλαιο της HTML.

Αυτές είναι οι βασικές τεχνικές που διαθέτει η JavaScript για επικοινωνία με τον χρήστη. Στη συνέχεια του κεφαλαίου θα επιχειρήσουμε μια επισκόπηση της αρχιτεκτονικής του φυλλομετρητή και του τρόπου που εκτελείται ο κώδικας JavaScript στο πλαίσιό του, ενώ θα εξετάσουμε τη διεπαφή της JavaScript με το περιβάλλον της ιστοσελίδας και του φυλλομετρητή.

7.4 Η JavaScript στον φυλλομετρητή

Ο κώδικας JavaScript μιας ιστοσελίδας σχετίζεται με το αρχείο .html. Μια ιστοσελίδα μπορεί να συνοδεύεται από πολλά τμήματα κώδικα JavaScript που μπορεί να βρίσκονται είτε ενσωματωμένα στο ίδιο το αρχείο ή σε εξωτερικά αρχεία που φορτώνονται από το αρχείο .html.

Όλα αυτά τα τμήματα του κώδικα συνυπάρχουν στον ίδιο κοινό χώρο, συναποτελώντας το *«πρόγραμμα JavaScript»* της σελίδας. Μάλιστα, όλα αυτά τα τμήματα κώδικα μοιράζονται τον κοινό χώρο ονομάτων μεταβλητών που ονομάζεται **global object**.

Ο κώδικας JavaScript που προέρχεται από το ίδιο το αρχείο .html εμφανίζεται ως περιεχόμενο στοιχείων <script>.

```
<script>
    // κώδικας JavaScript
</script>
```

Το στοιχείο αυτό μπορεί να βρίσκεται είτε στο τμήμα <head> του εγγράφου HTML ή οπουδήποτε στο τμήμα <body>.

Ένας δεύτερος τρόπος να φορτώσουμε στη σελίδα κώδικα JavaScript είναι από εξωτερικό αρχείο, στο οποίο γίνεται αναφορά μέσω του γνωρίσματος src του στοιχείου <script>.

Για παράδειγμα, ένα εξωτερικό αρχείο “scriptfile.js” φορτώνεται ως εξής:

```
<script src="scriptfile.js"></script>
```

Σε αυτή την περίπτωση το στοιχείο μπορεί να έχει μια από τις ιδιότητες async ή defer. Οι ιδιότητες αυτές δεν παίρνουν τιμή, είναι λογικές ιδιότητες και καθορίζουν τη σειρά φορτώματος και εκτέλεσης του αντίστοιχου κώδικα JavaScript.

- Η ιδιότητα **async** ορίζει ασύγχρονο φόρτωμα και εκτέλεση του κώδικα JavaScript, που μπορεί να γίνει ακόμη και πριν την ολοκλήρωση φορτώματος της HTML.
- Η ιδιότητα **defer** ορίζει ότι το φόρτωμα του κώδικα θα γίνει ασύγχρονα αλλά η εκτέλεση του κώδικα JavaScript θα γίνει μετά τη φόρτωση HTML ή άλλου κώδικα JS που επίσης έχει την ίδια ιδιότητα αλλά προηγείται.

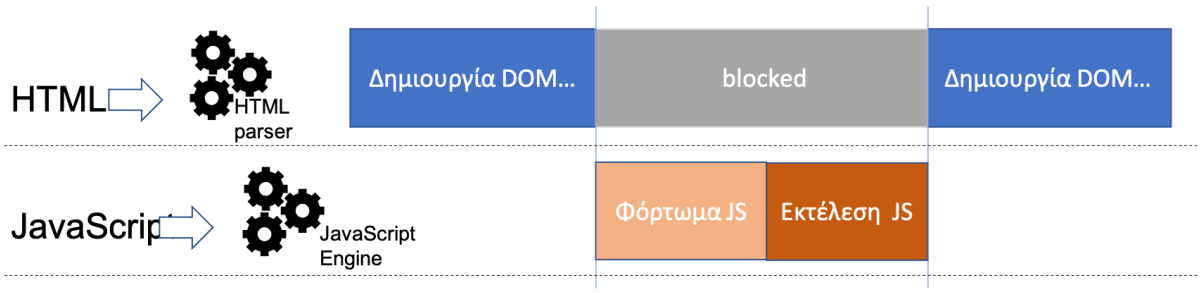
Για παράδειγμα, αν στη σελίδα μας περιέχεται ο παρακάτω κώδικας:

```
<script src="js/script2.js" defer ></script>
<script src="js/script3.js" defer ></script>
```

το script3.js θα εκτελεστεί μετά το script2.js.

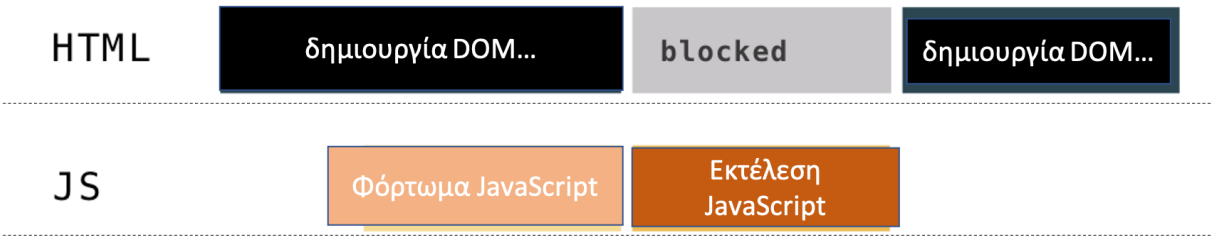
Αν δεν χρησιμοποιηθεί καμιά από αυτές τις ιδιότητες, ο κώδικας JavaScript μπλοκάρει το φόρτωμα της HTML και άρα το χτίσιμο του DOM. Αυτό γίνεται γιατί μέσα στον κώδικα JS μπορεί να κρύβονται εντολές που συμβάλλουν στο χτίσιμο της HTML, π.χ. με την εντολή document.write() που επιτρέπει εισαγωγή κώδικα HTML στο σημείο της συγκεκριμένης εντολής. Η χρήση αυτής της δυνατότητας δεν προτείνεται, όμως οι φυλλομετρητές πρέπει να φροντίσουν και για αυτό το ενδεχόμενο.

Η παρακάτω ***Εικόνα 7.6*** περιγράφει αυτό το σενάριο.



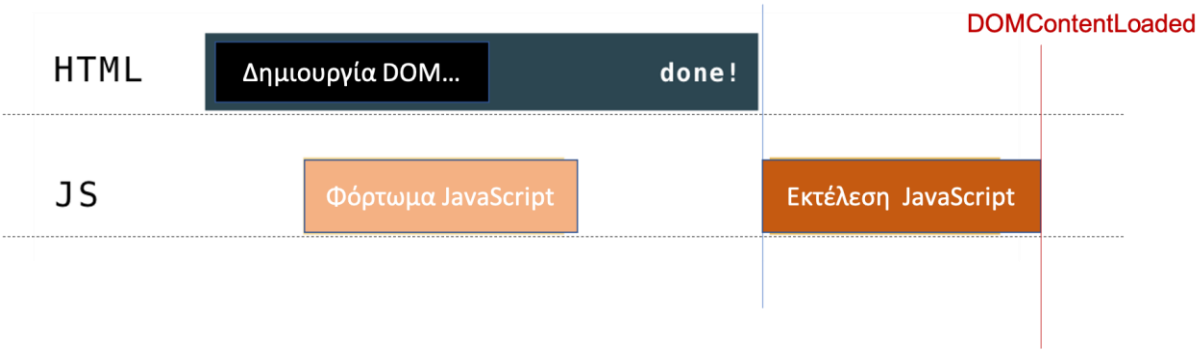
Εικόνα 7.6 *Φόρτωμα κώδικα JavaScript με μπλοκάρισμα φορτώματος της HTML/DOM.*

Η περίπτωση φορτώματος κώδικα JavaScript με την ιδιότητα async φαίνεται στην **Εικόνα *7.7***.



Εικόνα 7.7 *Φόρτωμα κώδικα JavaScript με την ιδιότητα async, ασύγχρονα με την HTML.*

Τέλος, η περίπτωση φορτώματος κώδικα JavaScript με την ιδιότητα defer φαίνεται στην ***Εικόνα 7.8***.



Εικόνα 7.8 *Φόρτωμα κώδικα JavaScript με την ιδιότητα defer: η εκτέλεση του κώδικα μεταφέρεται μετά την ολοκλήρωση του DOM.*

Ποια είναι η καλύτερη πολιτική για φόρτωμα κώδικα JS; Γενικά, τοποθετούμε το <script> μέσα στο <head> και χρησιμοποιούμε τα γνωρίσματα async και defer ώστε η JS να μην μπλοκάρει το κατέβασμα της σελίδας.

Χρησιμοποιούμε <script async ...> αν το περιεχόμενο της σελίδας είναι ανεξάρτητο από τον κώδικα JavaScript.

Χρησιμοποιούμε <script defer ...> αν επιθυμούμε ο κώδικας να τρέξει μετά την ολοκλήρωση φορτώματος της HTML, αν για παράδειγμα ο κώδικάς μας κάνει αναφορά σε στοιχεία του DOM.

Επίσης, μπορούμε να θέσουμε το <script> στο τέλος του <body> ώστε το φόρτωμα και η εκτέλεση του κώδικα να μην καθυστερεί το φόρτωμα της σελίδας, ιδιαίτερα αν ο κώδικας JavaScript είναι εκτενής.

Εναλλακτικά, μπορούμε να βάλουμε τμήμα του κώδικα μέσα σε μια συνάρτηση η οποία να κληθεί μετά από ένα συμβάν, π.χ. το συμβάν DOMContentLoaded που σηματοδοτεί την ολοκλήρωση φόρτωσης του DOM.

Στη συνέχεια θα κάνουμε ιδιαίτερη αναφορά στα συμβάντα που σχετίζονται με το φόρτωμα μιας σελίδας.

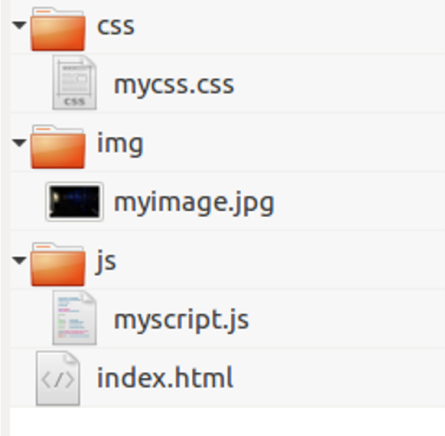
7.5 Διαχωρισμός κώδικα HMTL, CSS, JavaScript

Σε μια σύγχρονη διαδικτυακή εφαρμογή μια ιστοσελίδα περιλαμβάνει εκτός από το κυρίως αρχείο HTML και ένα σύνολο από αρχεία (π.χ. index.html). Αυτά τυπικά περιλαμβάνουν αρχείο ή αρχεία CSS, αρχεία JavaScript, καθώς και αρχεία εικόνων και άλλων πολυμέσων. Ο τρόπος που συνδέονται αυτά τα αρχεία, όπως έχει ήδη αναφερθεί, είναι αυτός που φαίνεται στο παράδειγμα.

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="css/mycss.css">
  </head>
  <body>
    <h1 id="first">Καλή σας μέρα</h1>
    <img src="img/myimage.jpg"></img>
    <script src="js/myscript.js"></script>
```

```
  </body>
</html>
```

Η οργάνωση των αρχείων του πρότζεκτ ανάπτυξης της ιστοσελίδας συνήθως έχει την παρακάτω δομή (**Εικόνα 7.9**):



Εικόνα 7.9 *Συνήθης οργάνωση των αρχείων ενός πρότζεκτ ανάπτυξης ιστοσελίδας.*

7.6 Η ακολουθία συμβάντων κατά το φόρτωμα μιας σελίδας

Η εκτέλεση της JavaScript στον φυλλομετρητή ακολουθεί τη λογική των γλωσσών προγραμματισμού που στηρίζονται στο **μοντέλο χειρισμού συμβάντων (event-based model)**. Τέτοιες είναι οι γλώσσες που λειτουργούν σε διαδραστικές συνθήκες, όπως σε γραφικά περιβάλλοντα αλληλεπίδρασης με τον χρήστη. Το μοντέλο αυτό περιλαμβάνει φόρτωμα του κώδικα και αρχικοποίηση των μεταβλητών και αντικειμένων, ορισμό των χειριστών συμβάντων, δηλαδή των τμημάτων του κώδικα (συναρτήσεων) που θα εκτελεστούν όταν συμβούν καθορισμένα συμβάντα (events). Στη συνέχεια το πρόγραμμα βρίσκεται σε αναμονή των συμβάντων αυτών, τα οποία προκύπτουν συνήθως από ενέργειες του χρήστη, αλλά και συμβάντων του συστήματος, όπως ανάκτηση δεδομένων από άλλες πηγές, ολοκλήρωση φόρτωσης τμημάτων του κώδικα κ.λπ. Όταν συμβεί ένα αναμενόμενο συμβάν, ενεργοποιείται ο αντίστοιχος χειριστής.

Ας ακολουθήσουμε στη συνέχεια την ακολουθία φάσεων και αντίστοιχων συμβάντων κατά το φόρτωμα της ιστοσελίδας.

Φάση 1: Φόρτωμα DOM. Ο φυλλομετρητής δημιουργεί ένα αντικείμενο **document** και αρχίζει τη συντακτική ανάλυση της ιστοσελίδας με προσθήκη κόμβων στο DOM καθώς αναλύει ένα προς ένα τα στοιχεία HTML και το περιεχόμενό τους. Η ιδιότητα document.readyState έχει την τιμή *loading* σε αυτή τη φάση. Αν κατά τη διάρκεια της συντακτικής ανάλυσης βρεθεί ετικέτα <script> χωρίς async, defer, φορτώνεται και εκτελείται ο κώδικας JavaScript, ενώ διακόπτεται το φόρτωμα της HTML. Αν κατά τη φάση αυτή ο αναλυτής συναντήσει ετικέτα <script async ...> με ασύγχρονο τρόπο, προχωράει στο φόρτωμα και την εκτέλεση του κώδικα JavaScript.

Φάση 2: Η ανάλυση της HTML ολοκληρώνεται, η ιδιότητα document.readyState παίρνει την τιμή *interactive*. Σε αυτή τη φάση τα τμήματα κώδικα JavaScript που ήταν σε κατάσταση defer εκτελούνται διαδοχικά το ένα μετά το άλλο. Μετά την ολοκλήρωση εκτέλεσης των τμημάτων αυτών του κώδικα, ενεργοποιείται το συμβάν DOMContentLoaded, που σηματοδοτεί τη μετάβαση της JavaScript από κατάσταση φορτώματος και εκτέλεσης κώδικα στη φάση της διαδραστικής εκτέλεσης, σε αναμονή συμβάντων.

Φάση 3: Η JavaScript έχει μπει σε κατάσταση αναμονής συμβάντων. Η σελίδα έχει φορτωθεί πλήρως, αν και τμήματα των πρόσθετων (εικόνες κ.λπ.) ίσως ακόμη φορτώνονται. Επίσης, πιθανόν τμήματα κώδικα JavaScript σε κατάσταση async να είναι ακόμη σε κατάσταση φορτώματος/εκτέλεσης. Η φάση αυτή ολοκληρώνεται όταν όλα τα πρόσθετα φορτωθούν και ολοκληρωθεί η εκτέλεση του κώδικα async. Τότε εμφανίζεται το συμβάν window.load, ενώ η ιδιότητα document.readyState = "complete". Με την ολοκλήρωση αυτής της φάσης ξεκινάει η ασύγχρονη λειτουργία της JavaScript.

7.6.1 Ένα παράδειγμα

Στο παράδειγμα αυτό θα φορτώσουμε μια ιστοσελίδα η οποία μας γνωστοποιεί τη χρονική διάρκεια κάθε φάσης φορτώματος όπως προκύπτει από τα σχετικά συμβάντα. Η σελίδα φορτώνει μια σειρά από φωτογραφίες διαφορετικής ανάλυσης από την ιστοσελίδα διάθεσης φωτογραφιών https://picsum.photos. Ο κώδικας καταγράφει τον χρόνο που απαιτείται για τα εξής συμβάντα: (α) Αλλαγή της ιδιότητας readyState σε interactive (ολοκλήρωση φορτώματος του DOM), (β) ενεργοποίηση του συμβάντος DOMContentLoaded που σηματοδοτεί την ολοκλήρωση εκτέλεσης του κώδικα JavaScript, (γ) ενεργοποίηση του συμβάντος window.load που σηματοδοτεί την ολοκλήρωση φορτώματος των πρόσθετων (εικόνων).

Η ιστοσελίδα φαίνεται στη συνέχεια. Θα πρέπει να σημειωθεί ότι το παράδειγμα περιέχει κώδικα JavaScript με συναρτήσεις και δομές που δεν έχουμε αναφέρει ακόμη, όμως παρουσιάζεται κυρίως για το αποτέλεσμα που παράγει, το οποίο φαίνεται στην εικόνα που ακολουθεί.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ex0</title>
    <script>
        const round = (v) => (v/1000).toFixed(2);
        report = "";
        function checkReady(e){
            report += `readyState=${document.readyState} --
${round(e.timeStamp)}sec <br>`;
        }
        document.addEventListener('readystatechange', checkReady);
        document.addEventListener('DOMContentLoaded', (e) => {
            report += `DOMContentLoaded -- ${round(e.timeStamp)}sec
:φόρτωμα DOM<br>`;
        });
        window.addEventListener('load', (e) => {
            report += `window.load -- ${round(e.timeStamp)}sec :φόρτωμα
σελίδας`;
            document.querySelector("#report").innerHTML = report;
            console.log(report);
        });
    </script>
</head>
<body>
    <div style="display: flex"></div>
    <div id="report" style="margin:10px;font-family:sans-serif;
    font-size:2rem;"></div>
    <script>
        const container = document.querySelector("div");
        for (let i=0;i<5;i++){
            const d = document.createElement("div");
            d.style.padding = "5px";
            const im = document.createElement("img");
            im.src = `https://picsum.photos/${i+2}00`
            d.appendChild(im);
            container.append(d);
        }
    </script>

</body>
</html>
```

Το αποτέλεσμα από μια τυπική φόρτωση της σελίδας φαίνεται στη συνέχεια (**Εικόνα *7.10***):

Εικόνα 7.10 *Τυπική φόρτωση της σελίδας.*

Προκύπτει ότι το φόρτωμα της σελίδας που περιλαμβάνει τις 5 εικόνες που φορτώνονται από το https://picsum.photos απαίτησε στη συγκεκριμένη περίπτωση συνολικά 0.42 sec.

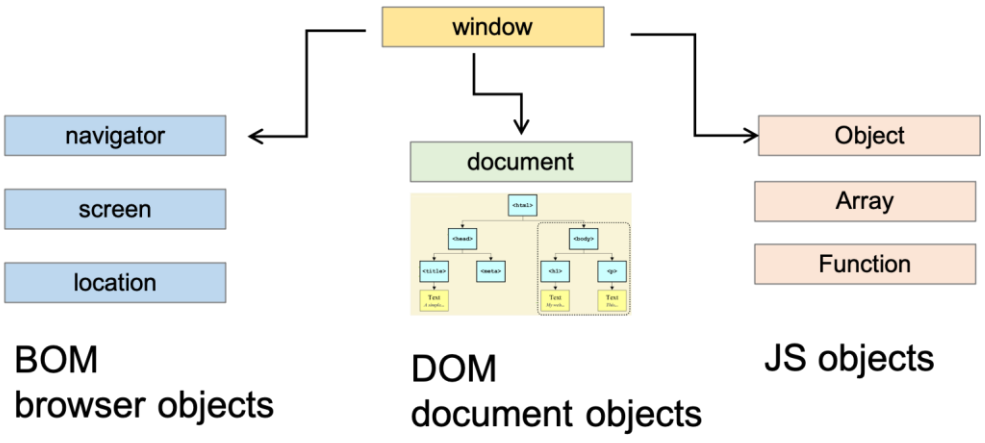
Συγκεκριμένα, το συμβάν ολοκλήρωσης της συντακτικής ανάλυσης της HTML ολοκληρώθηκε σε 0,18 δευτερόλεπτα, ενώ αμέσως ολοκληρώθηκε και το φόρτωμα του JavaScript κώδικα με κατάσταση defer (συμβάν DOMContentLoaded), αφού δεν υπήρχαν τέτοια τμήματα κώδικα JS. Τέλος, η ολοκλήρωση φορτώματος των εικόνων που προκαλεί την αλλαγή της document.readyState = "complete" καθώς και του συμβάντος window.load, απαιτεί, όπως προκύπτει, χρόνο μέχρι 0,42 δευτερόλεπτα.

Μελετήστε τον κώδικα του παραδείγματος και ιδιαίτερα την ιδιότητα του συμβάντος που επιτρέπει την καταγραφή χρόνων, που χρησιμοποιούν οι *χειριστές συμβάντων* για να μας παρουσιάσουν τη σχετική πληροφορία.

7.7 Το περιβάλλον εκτέλεσης της JavaScript

Στο περιβάλλον του φυλλομετρητή το πρόγραμμα JavaScript που φορτώθηκε με τη διαδικασία, και περιγράφηκε στην προηγούμενη παράγραφο, έχει πρόσβαση στον κοινό χώρο ονομάτων (global scope) που λέγεται *global object*. Το αντικείμενο αυτό είναι το αντικείμενο window, στο οποίο μάλιστα μπορούμε να αναφερθούμε με χρήση της λέξης-κλειδί this. Ιδιότητες του αντικειμένου αυτού είναι τα βασικά στοιχεία της γλώσσας (built-in objects), τα οποία μπορεί να χρησιμοποιηθούν είτε με σημειογραφία τελείας window.αντικείμενο είτε παραλείποντας την αναφορά στο αντικείμενο window.

Οι κατηγορίες αντικειμένων που έχει πρόσβαση η JavaScript μέσω του global object φαίνονται στο διάγραμμα στην **Εικόνα *7.11***.



Εικόνα 7.11 *Κατηγορίες αντικειμένων που έχει πρόσβαση η JavaScript.*

- **Αντικείμενα της γλώσσας JavaScript**. Μια πρώτη κατηγορία είναι τα αντικείμενα της γλώσσας, είτε τα συστατικά της στοιχεία είτε τα αντικείμενα που δημιουργεί ο κώδικας του χρήστη.
- **DOM**. Μια δεύτερη κατηγορία είναι τα αντικείμενα του *Document Object Model* που ανήκουν στο αντικείμενο document. Σε επόμενη ενότητα θα κάνουμε ιδιαίτερη αναφορά στη **διεπαφή DOM**, δηλαδή το σύνολο των μεθόδων που διαθέτει η JavaScript για αλληλεπίδραση με τα στοιχεία του DOM.
- **Browser objects**. Συχνά τα στοιχεία αυτά αναφέρονται ως BOM (browser object model), περιλαμβάνουν, μεταξύ άλλων, το αντικείμενο navigator, που περιέχει στοιχεία για το περιβάλλον του φυλλομετρητή και του λειτουργικού συστήματος του χρήστη, το αντικείμενο screen που αφορά την οθόνη του υπολογιστή του χρήστη, το αντικείμενο location που αφορά το URL της σελίδας κ.λπ.

7.7.1 Δραστηριότητα

Μεταβείτε στην κονσόλα του φυλλομετρητή σας και πληκτρολογήσετε window ή this. Επιθεωρήστε τις ιδιότητες του αντικειμένου αυτού. Ιδιαίτερα, εξετάστε τα αντικείμενα του φυλλομετρητή που αναφέρθηκαν, καθώς και το περιεχόμενο του DOM.

7.8 Βασικά στοιχεία του αντικείμενου window

Αν επιθεωρήσουμε τις ιδιότητες του αντικείμενου window, θα βρούμε, μεταξύ άλλων, αντικείμενα με ιδιότητες και μεθόδους που χρησιμοποιούνται ευρύτατα στην JavaScript.

Θα κάνουμε σύντομη αναφορά σε αυτά, ενώ σε επόμενες ενότητες θα γίνει ιδιαίτερη μνεία στις κυριότερες από αυτές.

7.8.1 Συναρτήσεις

- **Eval (έκφραση)**: Υπολογισμός του αποτελέσματος της έκφρασης. Θεωρείται κακή πρακτική η χρήση της για λόγους ασφάλειας.
- **isFinite (εκφραση)**: Επιστρέφει false αν το αποτέλεσμα της έκφρασης είναι +Infinity, -Infinity, NaN ή undefined, αλλιώς true
- **isNaN (τιμή)**: Επιστρέφει true αν η *τιμή* επιστρέφει NaN, αλλιώς false.
- **parseFloat (συμβολοσειρά)**: Επιστρέφει την αριθμητική τιμή ως δεκαδικό αριθμό. Αν όμως η συμβολοσειρά περιέχει μη αριθμητικούς χαρακτήρες από έναν χαρακτήρα και μετά, λαμβάνει υπόψη του το ως τότε τμήμα της συμβολοσειράς. Αν όμως η συμβολοσειρά αρχίζει με μη αριθμητικό χαρακτήρα, επιστρέφει NaN
- **parseInt (συμβολοσειρά, βάση)**: Παρόμοια με την parseFloat() για ακέραιο αριθμό, παίρνει ως δεύτερο προαιρετικό όρισμα τη βάση του αριθμητικού συστήματος (π.χ. 16 για το δεκαεξαδικό κ.λπ.)
- **encodeURI(URI)**: Κωδικοποίηση δεδομένων με την τεχνική URL Encoding, βάσει της κωδικοσελίδας UTF-8, λαμβάνοντας υπόψη ειδικούς χαρακτήρες, όπως ;,/?:@&=+$#, για παράδειγμα, κωδικοποιεί τα κενά ως %20.
- **encodeURIComponent()**: Όπως η προηγούμενη, μόνο που αφορά δεδομένα που θα τοποθετηθούν σε μεταβλητές PUT GET.
- **decodeURI()**: Αποκωδικοποίηση της encodeURI()
- **decodeURIComponent()**: Αποκωδικοποίηση της encodeURIComponent()

7.8.2 Βασικά αντικείμενα

- **Object**: Το αρχέτυπο αντικείμενο. Τα περισσότερα αντικείμενα της JavaScript κληρονομούν από αυτό.
- **Function**: Η κλάση των συναρτήσεων της JavaScript που είναι αντικείμενα πρώτης τάξης.
- **Boolean**: Αντικείμενο που αντιστοιχεί σε λογικές τιμές.
- **Symbol**: Αντικείμενο που αντιστοιχεί στα δεδομένα τύπου Symbol.

7.8.3 Αντικείμενα αριθμών

- **Number**: Αντικείμενο που αντιστοιχεί στον πρωτογενή τύπο δεδομένων Number.
- **BigInt**: Αντικείμενο που αντιστοιχεί στον πρωτογενή τύπο δεδομένων BigInt για αριθμούς

μεγαλύτερους από $2^{53} - 1$.

- **Math**: Αντικείμενο με ιδιότητες και μεθόδους για μαθηματικές συναρτήσεις και σταθερές.
- **Date**: Αντικείμενο που εκφράζει χρονική στιγμή σε χιλιοστά του δευτερολέπτου με αφετηρία μέτρησης 1 Ιανουαρίου 1970 UTC (τύπου Number).

7.8.4 Κείμενο

- **String**: Αντικείμενο που αφορά την αναπαράσταση και μεθόδους που αφορούν συμβολοσειρές.
- **RegExp**: Αντικείμενο που επιτρέπει την αντιστοίχιση προτύπων χαρακτήρων (κανονικές εκφράσεις) με συμβολοσειρές.

7.8.5 Συλλογές αντικειμένων

- **Array**: Αντικείμενο που αφορά την κλάση αντικειμένων τύπου Array (πίνακες). Να σημειωθεί ότι υπάρχουν ακόμη αντικείμενα για "typed" Arrays, πίνακες που δέχονται ορισμένου τύπου δεδομένα, όπως Float32Array κ.λπ.
- **Map**: Αντικείμενο που αφορά την κλάση Map, ακολουθία ζευγών «κλειδί: τιμή» που θυμάται τη σειρά δημιουργίας της.
- **Set**: Αντικείμενο που αφορά την κλάση Set, που επιτρέπει την αποθήκευση συλλογής μοναδικών στοιχείων.

7.8.6 Σφάλματα

- **Error** : αντικείμενα τα οποία δημιουργούνται όταν συμβούν σφάλματα κατά τη διάρκεια εκτέλεσης του κώδικα. Υπάρχουν ειδικοί τύποι σφαλμάτων, π.χ. ReferenceError κ.λπ.

Υπάρχουν πολλά ακόμη αντικείμενα ως ιδιότητες του αντικείμενου window.

Ακόμη, το αντικείμενο window, πέραν του να ορίζει τον καθολικό χώρο ονομάτων και να έχει ως ιδιότητες τα βασικά στοιχεία της γλώσσας (τα λεγόμενα "built-ins"), έχει έναν ακόμη ρόλο να παίξει: περιέχει πληροφορίες για το παράθυρο του φυλλομετρητή στο οποίο εμφανίζεται η ιστοσελίδα, όπως οι ιδιότητες innerHeight, innerWidth που περιέχουν τις διαστάσεις του παράθυρου.

Για πλήρη περιγραφή των αντικειμένων και ιδιοτήτων αυτών μπορείτε να συμβουλευτείτε τη MDN σχετικά με τα global objects.

7.9 Διεπαφή με το Document Object Model (DOM)

Σε αυτή την ενότητα θα εξετάσουμε τη διεπαφή της JavaScript με τα στοιχεία της ιστοσελίδας μέσω της ιδιότητας document του global object, η οποία έχει ως τιμή το αντικείμενο Document.

Το αντικείμενο αυτό αναπαριστά τα στοιχεία του εγγράφου HTML, που εμφανίζεται στο παράθυρο ή στην καρτέλα του φυλλομετρητή. Έχει οριστεί μια προγραμματιστική διεπαφή με τα στοιχεία αυτά, που ονομάζεται Document Object Model (DOM). Έχει γίνει ήδη αναφορά στην αναπαράσταση των στοιχείων ενός εγγράφου HTML ως ιεραρχία κόμβων με ρίζα το στοιχείο <html>. Το DOM API ορίζει στο περιβάλλον της JavaScript ένα σύνολο από αντικείμενα, που αντανακλούν την ιεραρχία στοιχείων της HTML. Για κάθε στοιχείο HTML υπάρχει ένα αντίστοιχο αντικείμενο (Element) JavaScript και για κάθε κείμενο του αρχείου HTML υπάρχει ένα αντικείμενο κειμένου (Text). Τα αντικείμενα αυτά είναι εξειδικεύσεις του αντικείμενου Node.

Συνεπώς, στην ιεραρχία του DOM εμπεριέχονται σαν αντικείμενα όλα τα στοιχεία της ιστοσελίδας. Να σημειωθεί ότι *αντικείμενο (object)* είναι μια βασική δομή δεδομένων της JavaScript που θα δούμε στη συνέχεια (Ενότητα 9.5). Στα αντικείμενα αυτά έχει πρόσβαση ένα πρόγραμμα JavaScript το οποίο μπορεί να τα τροποποιήσει. Συνεπώς, το document object model είναι η προγραμματιστική διεπαφή του εγγράφου HTML και είναι ο μηχανισμός που παρέχει στην JavaScript δυνατότητα πρόσβασης στο περιεχόμενο της ιστοσελίδας.

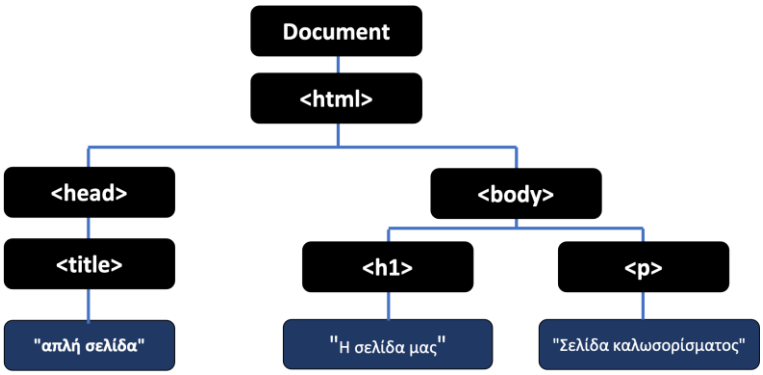
Ας πάρουμε καταρχάς ένα παράδειγμα απλού εγγράφου HTML.

```
<!DOCTYPE html>
<html lang="el">
    <head>
```

```
        <meta charset="utf-8">
        <title>απλή σελίδα</title>
    </head>
    <body>
        <h1>Η σελίδα μας</h1>
        <p>Σελίδα καλωσορίσματος</p>
    </body>
</html>
```

Σε αυτό το έγγραφο βλέπουμε ότι η ρίζα είναι το <html>, υπάρχει το στοιχείο <head> που είναι παιδί του <html>, το <body> που είναι επίσης ένα άλλο παιδί του <html> και αυτά με τη σειρά τους έχουν άλλα στοιχεία ως παιδιά τους.

Το document object model θα μπορούσαμε να το δούμε ως μια ιεραρχία από κόμβους ως εξής:



Εικόνα 7.12 *Το Document Object Model (DOM) είναι μια ιεραρχία από κόμβους.*

Στη συνέχεια θα δούμε την προγραμματιστική διεπαφή DOM API, που περιλαμβάνει τις συναρτήσεις που θα χρησιμοποιήσουμε σε ένα πρόγραμμα JavaScript για να επικοινωνήσει με αυτά τα στοιχεία.

7.9.1 Ανάκτηση στοιχείων του DOM

Η διεπαφή περιλαμβάνει διάφορες μεθόδους οι οποίες μας επιτρέπουν την ανάκτηση στοιχείων του δένδρου.

Η πιο σημαντική συνάρτηση είναι η **querySelector()**. Όπως και όλες οι συναρτήσεις του DOM API, είναι μέθοδος του αντικείμενου document, γι' αυτό την καλούμε ως document.querySelector(). Ως όρισμα περνάμε μια προδιαγραφή για το στοιχείο που αναζητούμε, χρησιμοποιώντας τη σύνταξη των επιλογέων της CSS (Ενότητα 4.4). Για παράδειγμα, αν δώσουμε όρισμα το "a", η συνάρτηση θα μας επιστρέψει το πρώτο στοιχείο τύπου <a>, δηλαδή το πρώτο anchor του εγγράφου. Η κλήση της document.querySelector(".myClass") θα επιστρέψει το πρώτο στοιχείο που έχει γνώρισμα class = "myClass", ενώ η document.querySelector("#myId") θα επιστρέψει το στοιχείο με γνώρισμα id = "myId" κ.λπ.

Μια παραλλαγή του querySelector είναι η document.querySelectorAll(). Η μέθοδος αυτή επιστρέφει μια συλλογή με στοιχεία, σε αντίθεση με την querySelector που επιστρέφει ένα μόνο στοιχείο. Θα μας επιστρέψει όλα τα στοιχεία που ικανοποιούν τον επιλογέα, για παράδειγμα η document.querySelectorAll("a") θα επιστρέψει όλα τα στοιχεία τύπου anchor <a> που υπάρχουν στην ιστοσελίδα.

Υπάρχουν και άλλες μέθοδοι για αναζήτηση στοιχείων του DOM, αυτές είναι:

- **document.getElementById("myid")** Επιστρέφει το στοιχείο που έχει id="myid".
- **document.getElementsByClassName("myClass")** Επιστρέφει τη συλλογή στοιχείων με όνομα κλάσης class = "myClass".
- **document.getElementsByTagName("tag")** Επιστρέφει συλλογή στοιχείων με όνομα ετικέτας tag.

Κατά κάποιον τρόπο, οι μέθοδοι αυτές έχουν υπερκαλυφθεί από τη διεπαφή querySelector που είδαμε, η οποία παίρνει ως όρισμα έναν επιλογέα CSS που μπορεί να αντιστοιχεί σε στοιχείο HTML, σε κλάση CSS, σε γνώρισμα (attribute), id κ.λπ.

Να σημειωθεί πως όλες οι παραπάνω μέθοδοι, εκτός από την getElementById, μπορούν να εκτελεστούν όχι μόνο στο document, αλλά και σε κάποιο στοιχείο.

Π.χ. η κλήση document.querySelector('#someId').querySelectorAll('.someClass') θα επιστρέψει μια

λίστα με τα στοιχεία που έχουν κλάση “someClass” και είναι απόγονοι του στοιχείου #someId.

7.9.2 Διαχείριση των γνωρισμάτων στοιχείων DOM

Στη συνέχεια θα δούμε τις μεθόδους της διεπαφής DOM που αφορούν τα γνωρίσματα των στοιχείων. Έστω πως ένα στοιχείο element είναι ένα αντικείμενο που μας επέστρεψε η μέθοδος querySelector. Μπορούμε με τη μέθοδο element.setAttribute(atr, val) να δώσουμε τιμή σε ένα γνώρισμά του element.getAttribute(attr), να πάρουμε την τιμή ενός γνωρίσματος, ενώ με την element.removeAttribute(attr) να καταργήσουμε ένα γνώρισμα.

7.9.3 Διαχείριση περιεχομένου στοιχείων DOM

Για τη διαχείριση του περιεχομένου των στοιχείων του DOM η διεπαφή διαθέτει δύο ιδιότητες, την innerHTML και την textContent.

Η ιδιότητα innerHTML έχει δύο χρήσεις:

(α) Η element.innerHTML επιστρέφει το περιεχόμενο του στοιχείου element, περιλαμβανομένου του κειμένου που αυτό περιέχει, καθώς και του κώδικα HTML.

(β) Επιπροσθέτως, μπορεί να χρησιμοποιηθεί για να δώσουμε τιμή στο περιεχόμενο του στοιχείου element, element.innerHTML = text.

Αντίστοιχη είναι η ιδιότητα element.textContent, η οποία μας επιστρέφει μόνο το κείμενο που περιέχεται στο στοιχείο.

Συνεπώς, θα πρέπει να τονιστεί πως η διαφορά της ιδιότητας textContent από την ιδιότητα innerHTML ενός στοιχείου είναι ότι η πρώτη περιέχει μόνο το κείμενο που περιέχεται στο στοιχείο, ενώ η δεύτερη περιέχει όλο τον κώδικα HTML που περιέχεται στο στοιχείο, τα στοιχεία παιδιά του κ.λπ.

7.9.4 Διαχείριση του στιλ των στοιχείων DOM

Η διεπαφή DOM μάς επιτρέπει να έχουμε πρόσβαση και να διαχειριστούμε το στιλ των στοιχείων.

Συγκεκριμένα, η ιδιότητα element.style μας επιστρέφει την προδιαγραφή του στιλ του συγκεκριμένου στοιχείου όπως καθορίζεται από τα φύλλα στιλ που σχετίζονται με το στοιχείο αυτό, ενώ επίσης μας επιτρέπει να τροποποιήσουμε αυτή την προδιαγραφή, αφού μπορούμε να ορίσουμε παραμέτρους στιλ αλλάζοντας την τιμή τους.

```
element.style.ιδιότητα = τιμή
```

Ένα σημείο που πρέπει να προσέξουμε, όμως, είναι η έκφραση των ιδιοτήτων στιλ της CSS ως μεταβλητών JavaScript. Επειδή στη CSS έχουμε συχνά μεταβλητές που περιέχουν τον χαρακτήρα παύλα-ενωτικό, π.χ. font-size, background-color κ.λπ., αυτές οι μεταβλητές δεν επιτρέπονται στην JavaScript, αφού ο χαρακτήρας "-" δεν επιτρέπεται στο όνομα μεταβλητής. Μια σύμβαση που ισχύει είναι να αναφερόμαστε στην ιδιότητα font-size της CSS ως fontSize στην JavaScript.

```
element.style.fontSize = "2rem";
```

7.9.5 Διαπέραση δένδρου DOM

Μια άλλη δυνατότητα που μας παρέχει η διεπαφή DOM είναι να διαπεράσουμε τα στοιχεία του δένδρου.

Ας δούμε καταρχήν τους όρους στην προγραμματιστική διεπαφή του DOM. Είναι όροι που συναντώνται γενικότερα στις δομές δεδομένων δένδρου.

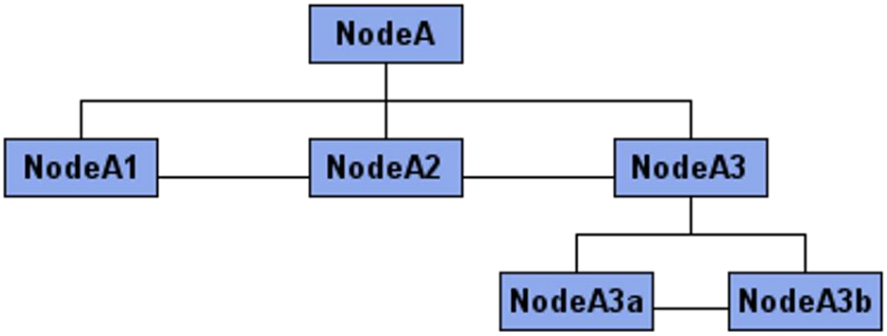
- Το **Element** αντιστοιχεί σε ένα στοιχείο.
- Το **Root** αντιστοιχεί στη ρίζα του δέντρου, το <html>.
- **Child** είναι το παιδί ενός κόμβου.
- **Descendant** είναι οποιοδήποτε κόμβος είτε παιδί είτε παιδί παιδιού, δηλαδή οποιοσδήποτε κόμβος του υποδένδρου κάτω από έναν κόμβο.
- **Parent** είναι ο πατέρας, το γονικό στοιχείο όπως ονομάζεται.
- **Sibling** είναι τα αδέλφια, είναι άλλα στοιχεία που ανήκουν στο ίδιο επίπεδο.
- **Text** node είναι ένας κόμβος που περιέχει κείμενο.

Ακολουθούν μερικά παραδείγματα διαπέρασης του δένδρου:

- element.firstChild επιστρέφει το πρώτο παιδί του στοιχείου element.
- element.lastChild επιστρέφει το τελευταίο παιδί του στοιχείου element.
- element.childNodes επιστρέφει ως συλλογή τα παιδιά του element.
- element.childNodes.length επιστρέφει το πλήθος των παιδιών του element.
- element.childNodes[0] επιστρέφει το πρώτο παιδί του element, ισοδύναμο με το element.firstChild.
- element.nextSibling είναι ο επόμενος αδελφός και element.prevSibling ο προηγούμενος αδελφός του element.
- element.parentNode είναι ο πατέρας του στοιχείου element.

Άσκηση

Έστω το παρακάτω απόσπασμα από ένα δένδρο DOM.



Εικόνα 7.13 *Απόσπασμα ενός δέντρου DOM.*

Ζητείται να βρείτε τις απαντήσεις στα παρακάτω ερωτήματα:

1. NodeA.firstChild
2. NodeA.lastChild
3. NodeA.childNodes.length
4. NodeA.childNodes[0]
5. NodeA.childNodes[1]
6. NodeA.childNodes[2]
7. NodeA1.parentNode
8. NodeA1.nextSibling
9. NodeA3.prevSibling
10. NodeA3.nextSibling
11. NodeA.lastChild.firstChild
12. NodeA3b.parentNode.parentNode

Απάντηση

(1) NodeA1, (2) NodeA3, (3) 3, (4) NodeA1, (5) NodeA2, (6) NodeA3, (7) NodeA, (8) NodeA2, (9) NodeA2, (10) null, (11) NodeA3a, (12) NodeA.

7.9.6 Εισαγωγή και διαγραφή στοιχείων του DOM

Μια ακόμη χρήσιμη δυνατότητα που μας παρέχει η διεπαφή του DOM είναι αυτή της δημιουργίας και της διαγραφής στοιχείων του δένδρου.

Αυτή η δυνατότητα υποστηρίζεται από τη μέθοδο createElement() που δημιουργεί ένα νέο στοιχείο, και στη συνέχεια από τη μέθοδο element.appendChild() για προσθήκη του στοιχείου στην ιεραρχία, ή τη μέθοδο element.removeChild() για διαγραφή στοιχείου από την ιεραρχία.

Ας δούμε ένα παράδειγμα. Έστω ότι θέλουμε να δημιουργήσουμε έναν κόμβο <p> κάτω από τον κόμβο <section> ενός εγγράφου HTML.

```
const sect = document.querySelector('section');
const para = document.createElement('p');
para.textContent = 'Ευχαριστούμε για την επίσκεψη!';
sect.appendChild(para);
```

Παρατηρούμε στο παράδειγμα αυτό ότι αρχικά στη μεταβλητή sect εκχωρήσαμε το στοιχείο <section> του εγγράφου (αν υπήρχαν πολλά στοιχεία αυτού του τύπου, μας επιστρέφει το πρώτο από αυτά), στη συνέχεια δημιουργούμε ένα νέο στοιχείο (μεταβλητή para) που του δίνουμε ένα περιεχόμενο κείμενο και, τέλος, το επισυνάπτουμε στο στοιχείο sect με τη μέθοδο appendChild().

Ας δούμε ένα ακόμη παράδειγμα, της διαγραφής του πρώτου παιδιού ενός κόμβου του δένδρου.

```
const sect = document.querySelector('section');
sect.removeChild(sect.childNodes[0]);
```

Να σημειωθεί ότι η μέθοδος removeChild() επιστρέφει ένα αντικείμενο τύπου Node μετά την επιτυχή διαγραφή του στοιχείου, ή null αν το στοιχείο δεν βρέθηκε στο δένδρο.

Άσκηση

Να εισάγετε ως πρώτο παιδί του στοιχείου με ταυτότητα id = "myID" έναν υπερσύνδεσμο με κείμενο *Πανεπιστήμιο Πατρών* προς την ιστοσελίδα του Πανεπιστημίου Πατρών.

Απάντηση

Θα χρησιμοποιήσουμε την appendChild():

```
const el = document.querySelector('#myID');
const link = document.createElement('a');
link.textContent = 'Πανεπιστήμιο Πατρών';
link.href = 'https://www.upatras.gr';
el.appendChild(link)
```

Αυτά ως μια σύντομη εισαγωγή στη διεπαφή προς το DOM και μια ιστοσελίδα. Οι μέθοδοι που είδαμε σε αυτή την ενότητα και ιδιαίτερα η querySelector θα είναι το βασικό μας εργαλείο για την επικοινωνία ανάμεσα σε ένα πρόγραμμα JavaScript και την ιστοσελίδα με την οποία θέλει να επικοινωνήσει.

7.10 Ερωτήσεις αυτοαξιολόγησης

1. Η JavaScript είναι μια παραλλαγή της Java για την κατασκευή ιστοσελίδων
   – Σωστό/Λάθος
2. Η μηχανή της JavaScript που τρέχει στο περιβάλλον του φυλλομετρητή Chrome είναι η
   Απάντηση _____
3. Το επίσημο όνομα της JavaScript είναι:
   1. JS
   2. ECMAScript
   3. node.js
   4. jQuery
   5. Vue.js
4. Δεν επιτρέπεται η ενσωμάτωση κώδικα JavaScript σε ένα αρχείο HTML.
   – Σωστό/Λάθος
5. Ποια είναι η ετικέτα μέσω της οποίας εισάγουμε κώδικα JavaScript σε ένα αρχείο HTML;
   1. <javascript>
   2. <script>
   3. <code>
   4. <link>
6. Ποια είναι η ιδιότητα που πρέπει να προσθέσουμε στο στοιχείο <script> ώστε ο κώδικας να τρέξει μετά την ολοκλήρωση φορτώματος της HTML;
   Απάντηση: _____
7. Αν θέλουμε να μάθουμε το πλάτος του παράθυρου της ιστοσελίδας, από ποιο αντικείμενο θα το πάρουμε;
   1. navigator.innerWidth
   2. window.innerWidth
   3. document.innerWidth
   4. canvas.innerWidth
8. Η μέθοδος document.querySelectorAll("p") θα επιστρέψει:
   1. όλα τα στοιχεία <p> του εγγράφου.
   2. το πρώτο στοιχείο <p> του εγγράφου.
   3. όλα τα στοιχεία του εγγράφου που αρχίζουν από "p".
   4. όλα τα στοιχεία του εγγράφου που έχουν id = "p".
   5. όλα τα στοιχεία του εγράφου που έχουν class = "p".
9. Έστω το παρακάτω έγγραφο HTML:

```
<h1>κατοικίδια</h1>
<h2>γάτες</h2>
<h2>σκύλοι</h2>
```

   Ποιο το αποτέλεσμα του document.querySelector('h2').textContent;
   Απάντηση: _____
10. Αν δώσουμε την εντολή console.log("hi") σε ένα script JS…
    1. η συμβολοσειρά ‘hi’ θα γραφτεί σε ένα αρχείο log file, στον ίδιο φάκελο με το αρχείο javaScript.
    2. η συμβολοσειρά ‘hi’ θα γραφτεί στο τέλος της ιστοσελίδας.
    3. η συμβολοσειρά ‘hi’ θα παρουσιαστεί σε ένα αναδυόμενο παράθυρο (alert window).
    4. η συμβολοσειρά ‘hi’ θα γραφτεί στην κονσόλα της JS.
11. Έστω η παρακάτω εντολή JS: document.querySelector('#a').textContent = 'info'. Ποιο το αποτέλεσμα;
    1. Ο υπερσύνδεσμος #a θα εμφανίζεται στον χρήστη ως η λέξη info.
    2. Το στοιχείο με id="a" θα περιέχει τη λέξη info που θα παρουσιάζεται στον χρήστη.
    3. Το στοιχείο <a> θα περιέχει τη λέξη info που θα παρουσιάζεται στον χρήστη.
    4. Το στοιχείο με ιδιότητα #a θα περιέχει τη λέξη info που θα παρουσιάζεται στον χρήστη.
    5. Το στοιχείο με class = "a" θα περιέχει τη λέξη info που θα παρουσιάζεται στον χρήστη.
12. Έστω ο παρακάτω κώδικας JS: let x = window.prompt('x='). Αν ο χρήστης δώσει την τιμή 33 στο αναδυόμενο παράθυρο, ποια θα είναι η τιμή που θα λάβει η μεταβλητή x;

1. Ο αριθμός 33.
2. Η συμβολοσειρά ‘33’.
3. Undefined.
4. Null.

13. Έστω οι παρακάτω εντολές JS:

```
let year = 1821;
let year = 1940;
```

Ποια η τελική τιμή της μεταβλητής year;

1. 1940.
2. 1821 γιατί η δεύτερη εντολή αγνοείται.
3. Θα πάρουμε σφάλμα, δεν επιτρέπεται ο επανορισμός της x X.

14. Έστω η παρακάτω εντολή JS : let x = 12.5; Τι τύπου είναι η μεταβλητή x;

1. real
2. integer
3. number
4. string
5. boolean

15. Ποια από τα παρακάτω ονόματα μεταβλητών είναι επιτρεπτά; (σημειώστε όλα όσα ταιριάζουν)

1. a
2. _a
3. $a
4. 12a
5. a12

16. Η JavaScript έχει τις παρακάτω χρήσεις (σημειώστε όλα όσα ισχύουν):

1. Εκτελείται στο περιβάλλον του φυλλομετρητή.
2. Ως γλώσσα εξυπηρετητή εφαρμογών.
3. Ως γλώσσα για ανάπτυξη εφαρμογών σε κινητά τηλέφωνα.
4. Ως γλώσσα προγραμματισμού εφαρμογών desktop.

17. Η JavaScript έκδοση ES6 έχει διάδοση σε πάνω από το 95% των φυλλομετρητών που χρησιμοποιούνται σήμερα σύμφωνα με την πηγή caniuse.com

– Σωστό/Λάθος

18. Ποιο το αποτέλεσμα;

```
let a=10;
let b = a++;
console.log(b);
```

Απάντηση: _____

7.11 Βιβλιογραφία και Αναφορές

Το πρότυπο EcmaScript (ECMA-262) συντηρείται από τον οργανισμό ecma.

Στο διαδίκτυο υπάρχουν πολλές πηγές για εκμάθηση της JavaScript καθώς και για αναφορά στα στοιχεία της γλώσσας. Η MDN είναι μια καλή πηγή για εισαγωγικά και προχωρημένα μαθήματα, όπως και για την HTML και CSS. Επίσης, η w3schools περιέχει μαθήματα, ενώ μια πλήρη σειρά μαθημάτων για την JavaScript με παραδείγματα περιλαμβάνει η JavaScript.info, ξεκινώντας από τη γλώσσα και προχωρώντας στη διεπαφή της με τον φυλλομετρητή.

Η JavaScript περιλαμβάνεται σε βιβλία που ήδη αναφέρθηκαν που αφορούν τις τεχνολογίες του προγραμματισμού στον ιστό όπως η HTML και η CSS. Αυτή την προσέγγιση στην ελληνική βιβλιογραφία ακολουθεί το βιβλίο των Δουληγέρη κ.ά. (2021), ενώ έχουν μεταφραστεί κάποια συγγράμματα και διατίθενται από τον Εύδοξο, όπως το βιβλίο των Kyrnin και Morrison (2021), και αυτό των Lemay, Coburn και Kyrnin (2016).

Μια άλλη προσέγγιση είναι αυτή των εγχειριδίων που ξεκινούν με τη σύνταξη της γλώσσας JavaScript και περιλαμβάνουν τη διεπαφή με το DOM σε μεταγενέστερο κεφάλαιο. Στις πηγές αυτές συχνά περιλαμβάνονται και κεφάλαια που αφορούν τη λειτουργία της γλώσσας στο περιβάλλον node.js. Αυτή την προσέγγιση ακολουθεί το βιβλίο του Λιακέα (2021). Από τη διεθνή βιβλιογραφία παρόμοια προσέγγιση ακολουθεί το βιβλίο του Flanagan (2020), ενώ υπάρχουν και πολλά άλλα, όπως Frisbie (2020) κ.λπ.

Επιπλέον, οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Για αυτό το κεφάλαιο το σχετικό μάθημα είναι οι εισαγωγικές διαλέξεις του μαθήματος «Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, Javascript».

A. Ξενόγλωσσες

Flanagan, D. (2020). *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language* (7th ed.). O'Reilly Media, Inc.

Frisbie, M. (2020). *Professional JavaScript for Web Developers*. Wiley.

McFedries, P. (2019). *Web Design Playground - HTML & CSS The Interactive Way*. Manning.

B. Ελληνόγλωσσες

Αβούρης, Ν. (2018). Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, JavaScript. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr.

Δουληγέρης, Χ., Μαυροπόδη, Ρ., Κοπανάκη, Ε., & Καραλής, Α. (2021). *Τεχνολογίες και Προγραμματισμός στον Παγκόσμιο Ιστό* (2η έκδοση). Εκδόσεις Νέων Τεχνολογιών. Κωδικός βιβλίου στον Εύδοξο: 102125023.

Kyrnin, J., & Meloni, J. (2021). *Sams Teach Yourself HTML5, CSS and Javascript* (3rd ed.). Εκδόσεις Γκιούρδας.

Lemay, L., Coburn, R., & Kyrnin, J. (2016). *Sams Teach Yourself HTML, CSS & JavaScript* (7th ed). Εκδόσεις Γκιούρδας. Κωδικός Βιβλίου στον Εύδοξο: 59357307.

Λιακέας, Γ. (2021). *Η γλώσσα JavaScript* (3η έκδ.). Κλειδάριθμος. Κωδικός βιβλίου στον Εύδοξο: 102070465.

Κεφάλαιο 8. Πρωτογενείς τύποι δεδομένων και εντολές της JavaScript

Σύνοψη

Στο δεύτερο αυτό κεφάλαιο με αντικείμενο την εισαγωγή στη γλώσσα προγραμματισμού JavaScript θα ξεκινήσουμε την επισκόπηση των διαφορετικών πρωτογενών τύπων δεδομένων που υποστηρίζει η γλώσσα. Συγκεκριμένα, θα δούμε τους πρωτογενείς τύπους δεδομένων (primitive data types), τους τύπους δεδομένων Number, String, Boolean, null, undefined.

Επίσης, στο κεφάλαιο αυτό θα γίνει η εισαγωγή στον πυρήνα των εντολών της JavaScript που υλοποιούν δομές ελέγχου ροής και επανάληψης που διαθέτει η γλώσσα. Θα προχωρήσουμε σε ανασκόπηση των βασικών προγραμματιστικών δομών της JavaScript: των εντολών ελέγχου της ροής του προγράμματος, όπως οι if και switch, των εντολών που επιτρέπουν την επαναληπτική εκτέλεση ενός κώδικα, όπως οι for, while, do/while και, τέλος, εντολών που επιτρέπουν τη μετάβαση σε άλλο τμήμα του κώδικα, όπως οι break, continue και throw.

Προαπαιτούμενη γνώση

Είναι απαραίτητη η εξοικείωση με το κεφάλαιο 7.

8.1 Τύποι δεδομένων

Υπάρχουν οι εξής πρωτογενείς (primitive) τύποι δεδομένων:

Οι τύποι **number**, **string**, **boolean** που συναντιούνται στις περισσότερες γλώσσες προγραμματισμού.

Ακόμη, ορίζονται οι ειδικοί τύποι **null** και **undefined** που θα περιγραφούν στη συνέχεια.

Επίσης, σε τελευταίες εκδόσεις της γλώσσας έχουν προστεθεί οι τύποι **bigInt** για αναπαράσταση μεγάλων ακέραιων αριθμών μεγαλύτερων από 2^{53}, καθώς και ο τύπος **symbol**.

Εκτός των πρωτογενών τύπων δεδομένων, υπάρχουν οι τύποι **δεδομένων αναφοράς**. Στην κατηγορία αυτή είναι ο τύπος δεδομένων που αφορά όλα τα δεδομένα πλην των ανωτέρω, το **αντικείμενο (object)**. Ένα αντικείμενο της JavaScript είναι μια μη ταξινομημένη συλλογή από ζευγάρια *κλειδιών-τιμών*. Η γλώσσα ορίζει επίσης ένα ειδικό είδος αντικείμενου, τον **πίνακα (array)**, που αντιπροσωπεύει μια διαταγμένη συλλογή αριθμημένων τιμών.

Οι πρωτογενείς τύποι δεδομένων θα συζητηθούν στο παρόν κεφάλαιο, ενώ οι πίνακες και τα αντικείμενα θα συζητηθούν μαζί με τις συναρτήσεις στο επόμενο κεφάλαιο.

8.2 Πρωτογενείς τύποι δεδομένων: Number

Ο τύπος δεδομένων **Number** αφορά τιμές ακεραίων με τιμές $\pm 2^{53}$ και κλασματικών αριθμών μεγέθους μέχρι $\pm 1.7976931348623157 \times 10^{308}$ και μικρών τιμών μέχρι $\pm 5 \times 10^{-324}$. Οι κλασματικοί αριθμοί αναπαρίστανται με κωδικοποίηση 64-bit IEEE 754, όπως στις περισσότερες σύγχρονες γλώσσες προγραμματισμού (αντίστοιχος τύπου double της Java, C++, float της Python κ.λπ.).

Για αναπαράσταση πολύ μεγάλων ακέραιων αριθμών πέρα από το παραπάνω όριο έχει εισαχθεί από την έκδοση ES2020 ο τύπος **BigInt**, ο οποίος αναπαριστά τον αριθμό ως ακολουθία ψηφίων και δεν έχει άνω όριο. Όμως, η υλοποίησή του δεν έχει ολοκληρωθεί στους φυλλομετρητές ακόμη.

Οι τιμές μεταβλητών τύπου Number μπορεί να είναι είτε ακέραιοι είτε κλασματικοί αριθμοί είτε αριθμοί σε δεκαεξαδική μορφή, σε δυαδική μορφή, σε μορφή ύψωσης σε δύναμη κ.λπ. Παραδείγματα ακολουθούν:

```
let x = 12345 // ακέραιος
let y = 123.456 // δεκαδικός
let z = 1.473E+32 // δύναμη 1.473 × 10³²
let k = 0xBADCAFE // δεκαεξαδικός => 195939070
```

Υπάρχει ακόμη η δυνατότητα από την έκδοση ES6 και μετά αναπαράστασης οκταδικών αριθμών (αρχίζουν με **0o**) και δυαδικών αριθμών (αρχίζουν με **0b**).

Με χρήση του τελεστή typeof μπορούμε να ελέγξουμε τον τύπο μιας μεταβλητής. Σε όλες τις παραπάνω περιπτώσεις η έκφραση, για παράδειγμα, typeof x επιστρέφει τη συμβολοσειρά 'number'.

Όπως έχει ήδη αναφερθεί, στους αριθμούς μπορούμε να εφαρμόσουμε τελεστές αριθμητικών πράξεων, +, -, *, /, %, **, καθώς και τους τελεστές ++, --, +=, -= κ.λπ.

Το global object διαθέτει το αντικείμενο Math το οποίο έχει μεθόδους για πιο σύνθετες μαθηματικές συναρτήσεις.

Οι κυριότερες από αυτές είναι:

- **Math.abs(-20)**, 20 απόλυτη τιμή
- **Math.ceil(10.5)**, 11 ο επόμενος ακέραιος
- **Math.E**, 2.718281828459045 η σταθερά *e*
- **Math.exp(5)**, e**5 = 148.413, δύναμη του *e*
- **Math.floor(8.9)**, 8 το ακέραιο μέρος
- **Math.log(3), // 1.0986122886681096
- **Math.max(10,20)**, 20 μέγιστη τιμή
- **Math.min(10,20)**, 10 ελάχιστη τιμή,
- **Math.pow(3,4)**, 81 ύψωση του x στη δύναμη y, x^y
- **Math.round(5.6)**, 6 στρογγύλεμα στον πλησιέστερο ακέραιο
- **Math.sqrt(4)**, 2 τετραγωνική ρίζα
- **Math.PI**, π = 3,14
- **Math.sin(θ)**, ημίτονο
- **Math.cos(θ)**, συνημίτονο
- **Math.tan(θ)**, εφαπτομένη

Θα πρέπει να σημειωθεί ότι η γωνία θ στις τριγωνομετρικές συναρτήσεις εκφράζεται σε rad, δηλαδή 90° = Math.PI/2, συνεπώς: Math.cos(Math.PI/2) = 0.

Επίσης, το αντικείμενο Math διαθέτει αρκετές ακόμη χρήσιμες συναρτήσεις, μεταξύ των οποίων η Math.random(), η οποία όταν κληθεί επιστρέφει έναν ψευδο-τυχαίο δεκαδικό αριθμό στο διάστημα από 0 μέχρι 1, χωρίς να περιλαμβάνεται το 1.

Αντίθετα με άλλες γλώσσες δεδομένων, στην JavaScript αν μια μαθηματική έκφραση προκαλέσει υπερχείλιση (overflow), όπως για παράδειγμα η διαίρεση με το μηδέν, δεν προκύπτει σφάλμα, αλλά το αποτέλεσμα της έκφρασης μπορεί να πάρει μια από τις παρακάτω ειδικές τιμές:

- **Infinity** (θετικό άπειρο)
- **-Infinity** (αρνητικό άπειρο)
- **NaN** μη αριθμητική τιμή (Not a Number)

Μάλιστα, το global object διαθέτει συναρτήσεις για έλεγχο αυτών των τιμών, όπως isNaN(x).

Για παράδειγμα:

```
console.log(-10/0);
> -Infinity
```

Επίσης:

```
isNaN("αβγ");
> true
```

8.3 Πρωτογενείς τύποι δεδομένων: Συμβολοσειρές

Οι συμβολοσειρές (string) είναι ο τύπος δεδομένων που αναπαριστά ακολουθία χαρακτήρων. Οι συμβολοσειρές στην JavaScript περιέχουν χαρακτήρες Unicode, 16bit ο καθένας. Οριοθετούμε συμβολοσειρές με απλά ' ή διπλά " εισαγωγικά ή με τον χαρακτήρα *backquote* `. Χρησιμοποιούμε το ένα για να οριοθετήσουμε συμβολοσειρά που περιέχει το άλλο.

Ο χαρακτήρας backquote χρησιμοποιείται για συμβολοσειρές που μπορούν να περιέχουν εκφράσεις JavaScript, όπως θα εξηγηθεί στη συνέχεια.

Ο χαρακτήρας διαφυγής \ μπορεί να χρησιμοποιηθεί για να ακυρώσει τον ειδικό ρόλο κάποιου χαρακτήρα, όπως του χαρακτήρα οριοθέτησης της συμβολοσειράς.

Για παράδειγμα:

```
console.log("του είπαν: \"τι χαμπάρια μας φέρνεις;\"")

>του είπαν: "τι χαμπάρια μας φέρνεις;"
```

Ο τελεστής “+” επιτρέπει τη συνένωση συμβολοσειρών.

```
let one = 'Καλή σας μέρα ';
let two = 'άρχοντες! ';
console.log(one + two);
>Καλή σας μέρα άρχοντες!
```

Στο επόμενο παράδειγμα, όταν ο χρήστης πατήσει το πλήκτρο, προκύπτει παράθυρο με ερώτημα για το όνομά του και ακολουθεί χαιρετισμός με σύνθεση συμβολοσειρών.

```
<button> ok </button>
let button = document.querySelector('button');
button.onclick = function() {
    let name = prompt(Πώς σε λένε;');
    alert('Καλωσήρθες ' + name); }
```

Ένα αξιοπρόσεκτο χαρακτηριστικό της γλώσσας, καθώς έχει *δυναμικό* σύστημα τύπων, είναι πως μπορούμε να κάνουμε πράξεις με τιμές διαφορετικών τύπων, π.χ., μεταξύ συμβολοσειρών και αριθμών:

```
10 + '20'
> "1020"
10+20
> 30
'abc'+10
> "abc10"
```

8.3.1 Δείκτης συμβολοσειράς

Μπορούμε να αναφερθούμε σε επιμέρους στοιχεία μιας συμβολοσειράς με χρήστη δεικτών που αρχίζουν από 0 μέχρι length-1, όπου length το πλήθος χαρακτήρων της συμβολοσειράς.

Για παράδειγμα:

```
const hero = "Αθανάσιος Διάκος";
console.log(hero[1]);
> 'θ'
```

Το μήκος της συμβολοσειράς μπορούμε να το πάρουμε με χρήση της ιδιότητας length

```
const hero = "Αθανάσιος Διάκος";
console.log(hero.length);
> 16
```

επειδή η συμβολοσειρά hero περιέχει 16 χαρακτήρες.

Μια λεπτομέρεια που θα πρέπει να προσέξουμε είναι ότι, αν η συμβολοσειρά περιέχει σύμβολα που συντίθενται από ακολουθίες Unicode 16bit, η ιδιότητα length θα μετρήσει το πλήθος αυτών των χαρακτήρων. Για παράδειγμα, emoji σημαιών χωρών:

```
const greekFlag = "🇬🇷";
greekFlag.length;
> 4
```

Θα πρέπει όμως να σημειωθεί ότι κείμενα στις περισσότερες γλώσσες, συμπεριλαμβανομένων των ευρωπαϊκών γλωσσών αλλά και των ιδεογραμμάτων των ασιατικών γλωσσών, περιλαμβάνονται στην κωδικοποίηση Unicode 16bit, συνεπώς το μήκος της συμβολοσειράς μπορεί να μετρηθεί.

```
const hello = "你好" // Nǐ hǎo (γεια σας στα κινεζικά)
hello.length;
> 2
```

8.3.2 Μετατροπές από συμβολοσειρές σε αριθμούς

Για τη μετατροπή μιας συμβολοσειράς με αριθμητικούς χαρακτήρες σε αριθμό μπορούμε να χρησιμοποιήσουμε τον δημιουργό του αντικείμενου Number().

```
// μετατροπή από string σε αριθμό
let myString = '123';
let myNum = Number(myString);
typeof myNum;
>"number"
```

Αν το όρισμα του δημιουργού Number() δεν εκφράζει έναν αριθμό, τότε η συνάρτηση επιστρέφει ένα αντικείμενο τύπου number με την τιμή NaN.

Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις parseInt() και parseFloat(), που ανήκουν στο global object, για μετατροπή συμβολοσειράς σε ακέραιο ή κλασματικό αριθμό αντίστοιχα.

Θα πρέπει να προσέξουμε μια ιδιαιτερότητα των συναρτήσεων αυτών. Αν η συμβολοσειρά αρχίζει με αριθμητικούς χαρακτήρες και στη συνέχεια περιέχει μη αριθμητικούς χαρακτήρες, αγνοεί τους μη αριθμητικούς και επιστρέφει τον αριθμό που προκύπτει, και όχι NaN που επιστρέφει η δημιουργός Number().

```
Number('123abc'); \\ NaN
parseInt('123abc'); \\ 123
```

Η αντίστροφη μετατροπή από αριθμό στην αντίστοιχη συμβολοσειρά γίνεται με κλήση της μεθόδου toString() του αντικείμενου Number.

```
// μετατροπή από αριθμό σε string
let myNum = 123;
let myString = myNum.toString();
```

8.3.3 Κύριες μέθοδοι και τελεστές του String

Έχουμε ήδη δει πώς αναφερόμαστε σε επιμέρους χαρακτήρες μιας συμβολοσειράς και πώς βρίσκουμε το μήκος μια συμβολοσειράς.

```
let myName = 'Nikos';
myName.length;    // 5

//πρώτος χαρακτήρας
myName[0];     // "N"

//τελευταίος χαρακτήρας
myName[myName.length-1];  // "s"
```

Στη συνέχεια θα δούμε δύο ενδιαφέρουσες μεθόδους του αντικείμενου String, τη slice() και την indexOf():

```
//τμήμα, από 0 έως 3
myName.slice(0,3); // returns 'Nik'

//βρες τη θέση της υπο-συμβολοσειράς  -1 αν δεν βρεθεί
myName.indexOf('kos');     // 2

//τεμάχιο συμβολοσειράς από θέση i1, έως i2
myName.slice(3); // επιστρέφει 'os'
myName.slice(1,4); // επιστρέφει 'iko'
```

Όπως φαίνεται από τα παραδείγματα, μπορούμε να αναφερθούμε σε τμήμα της συμβολοσειράς από τη θέση index1 μέχρι τη θέση index2, χωρίς να περιλαμβάνεται η τελευταία με χρήση της s.slice(index1, index2).

Επίσης, με τη μέθοδο s.indexOf (υπο-συμβολοσειρά) βρίσκουμε τη θέση που για πρώτη φορά συναντάμε την υπο-συμβολοσειρά στην s. Αν αυτή δεν υπάρχει, τότε μας επιστρέφει την τιμή -1.

Μπορούμε να διαχωρίσουμε μια συμβολοσειρά σε έναν πίνακα των επιμέρους χαρακτήρων της με χρήση

της μεθόδου split(). Η μέθοδος μοιάζει με την αντίστοιχη της Python με μια διαφορά, ότι πρέπει να δώσουμε οπωσδήποτε όρισμα. Για παράδειγμα:

```
//διαχωρισμός συμβολοσειράς σε πίνακα στοιχείων
//split(delimiter_string)
'αβγ'.split('');
> [ 'α', 'β', 'γ' ]
'αβγ'.split('β');
> [ 'α', 'γ' ]
```

Άλλες χρήσιμες μέθοδοι του αντικείμενου String είναι:

```
myName.toLowerCase(); // "nikos"
myName.toUpperCase(); // "NIKOS"

myName.replace('kos','na') //"Nina"

myName.charAt(2); // "k"

myName.charCodeAt(2); // 107
```

Επίσης, η μέθοδος trim() αποκόπτει τα κενά στην αρχή και στο τέλος μιας συμβολοσειράς με αντίστοιχο τρόπο όπως η μέθοδος strip() στην Python:

```
"       καλή σας μέρα           ".trim();
> 'καλή σας μέρα'
```

8.3.4 Συμβολοσειρές-πρότυπα (template literals)

Ένας ειδικός τύπος συμβολοσειρών εισήχθη στην JavaScript στην έκδοση ES6. Είναι οι **συμβολοσειρές πρότυπα**, που έχουν το χαρακτηριστικό ότι μπορούν να περιλάβουν εκφράσεις JavaScript ${έκφραση}, οι οποίες αντικαθίστανται από την τιμή τους.

Οι συμβολοσειρές αυτές οριοθετούνται από το σύμβολο backquote (`).

Ας δούμε ένα παράδειγμα.

```
let a = 5;
let b = 10;
console.log(`a+b=${a+b} ενώ 2a+b=${2*a + b}.`);
>'a+b=15 ενώ 2a+b=20.'
```

Ασκήσεις

Άσκηση 1

Έστω

```
let input = "ΘεσΑΛΛΟνίκη";
```

Να γράψετε εντολές JavaScript που παράγουν σωστή συμβολοσειρά για το όνομα της πόλης.

Απάντηση

```
let temp = input.toLowerCase();
let startCharacter = temp[0].toUpperCase();
let result = startCharacter + temp.slice(1);
```

Άσκηση 2

Έστω

```
let input = "ATH675847583748sjt567654;Athens El.Venizelos";
```

Ζητείται να γράψετε κώδικα που εξάγει από την παραπάνω συμβολοσειρά την εξής (τους τρεις πρώτους χαρακτήρες και την υπόλοιπη συμβολοσειρά μετά το ;): ATH:Athens El.Venizelos

Απάντηση

```
console.log(input.slice(0,3)+ ":" + input.slice(input.indexOf(";")+1))
> 'ATH:Athens El.Venizelos'
```

8.4 Πρωτογενείς τύποι δεδομένων: null και undefined

Στην ενότητα αυτή θα δούμε δύο ακόμη πρωτογενείς τύπους δεδομένων, ειδικού σκοπού.

Ο πρώτος τύπος είναι το null, μια λέξη-κλειδί που υποδεικνύει την απουσία τιμής. Η χρήση του τελεστή typeof στο null επιστρέφει τη συμβολοσειρά “object”, υποδεικνύοντας ότι το null μπορεί να θεωρηθεί ως μια ειδική τιμή αντικείμενου που υποδηλώνει την απουσία αντικείμενου. Σύμφωνα με τον ορισμό της γλώσσας, το null δεν θεωρείται αντικείμενο αλλά το μοναδικό μέλος ενός ξεχωριστού τύπου δεδομένων που μπορεί να χρησιμοποιηθεί για να δείξει «μη τιμή» για αριθμούς και συμβολοσειρές, καθώς επίσης για αντικείμενα.

Έστω σε μια ιστοσελίδα χωρίς υπερσυνδέσμους (χωρίς στοιχεία <a>) ας δούμε τι θα συμβεί αν αναζητήσουμε στοιχεία αυτού του τύπου στο DOM:

```
let x = document.querySelector("a");
console.log(x);
> null
typeof x;
> "object"
```

Υπάρχει όμως ένας ακόμη τύπος δεδομένων που υποδηλώνει μη τιμή: ο τύπος undefined.

Ο τύπος αυτός αποδίδεται σε μεταβλητές που έχουν οριστεί χωρίς να λάβουν τιμή, σε συναρτήσεις που δεν επιστρέφουν δεδομένα (δεν έχουν εντολή return), σε ιδιότητες αντικειμένων που δεν υπάρχουν, σε τιμές ορισμάτων συναρτήσεων που δεν τους έχει δοθεί τιμή. Η τιμή αυτή επιστρέφει ως τύπος δεδομένων από τον τελεστή typeof. Ας δούμε μερικά παραδείγματα:

```
let x; // μεταβλητή χωρίς αρχική τιμή
typeof x;
> 'undefined'

const ob = {name: "Nikos"} // αντικείμενο
typeof ob.age; // ιδιότητα που δεν υπάρχει
> 'undefined'

function f(){} // συνάρτηση που δεν επιστρέφει τιμή
typeof f();
> 'undefined'
```

8.5 Πρωτογενείς τύποι δεδομένων: Boolean

Ο τύπος δεδομένων Boolean αντιπροσωπεύει λογικές μεταβλητές οι οποίες παίρνουν τιμή αληθές/ψευδές. Οι δεσμευμένες λέξεις true και false αντιστοιχούν στις τιμές αυτές αντίστοιχα.

Η τιμή αληθές/ψευδές είναι το αποτέλεσμα μιας σύγκρισης, η οποία εισάγεται με τον τελεστή σύγκρισης === (ισότητα τιμής και τύπου δεδομένων) ή == (ισότητα μόνο τιμής, όχι απαραίτητα και τύπου δεδομένων) ή και άλλων τελεστών σύγκρισης >, <, >=, <=, !== (έλεγχος ανισότητας τιμής και τύπου δεδομένων), != (έλεγχος ανισότητας τιμής).

Εκφράσεις που περιέχουν σύγκριση συνήθως χρησιμοποιούνται σε εντολές που απαιτούν τιμή αληθές/ψευδές, όπως η εντολή if ή η εντολή while.

Όμως, θα πρέπει να σημειωθεί ότι οποιαδήποτε έκφραση της JavaScript, ανάλογα με την τιμή που παίρνει, αντιστοιχεί σε αληθές/ψευδές. Ο κανόνας που ισχύει είναι ο εξής:

- Αν η έκφραση έχει την τιμή undefined, null, 0, -0, NaN, "" (κενή συμβολοσειρά), τότε είναι ισοδύναμη με false.
- Σε οποιαδήποτε άλλη περίπτωση είναι true.

Οι λογικοί τελεστές της JavaScript είναι:

- **&&** τελεστής AND (λογικό «και»)
- || τελεστής OR (λογικό «ή»)
- **!** τελεστής NOT (λογικό «όχι»)

8.5.1 Τιμή λογικών εκφράσεων

Ας δούμε τι επιστρέφουν οι λογικές εκφράσεις, κάτι που συχνά εκμεταλλεύονται οι προγραμματιστές JavaScript για να ελέγξουν τιμή σε μεταβλητές ή να δώσουν μια προκαθορισμένη τιμή σε μια μεταβλητή αν αυτή δεν έχει ήδη οριστεί.

Μια έκφραση λογικής OR θα λάβει την τιμή της πρώτης μεταβλητής που είναι true, επειδή η λογική OR ικανοποιείται σε αληθές αρκεί ένας όρος να είναι αληθής, ή την τελευταία τιμή αν κανένας όρος δεν είναι αληθής.

Παραδείγματα εκφράσεων OR:

- 0 || undefined || 5 : θα πάρει την τιμή 5 (true)
- 0 || 5 || 10 : θα πάρει την τιμή 5 (true)
- 0 || undefined || null : θα πάρει την τιμή null (false)

Αντίστοιχα, σε μια έκφραση με λογική AND θα επιστρέψει τον τελευταίο όρο αν όλοι είναι αληθείς και τον πρώτο ψευδή όρο αν υπάρχει ψευδής. Αυτό γιατί στη λογική AND αρκεί ένας όρος να είναι ψευδής για να πάρει η έκφραση την τιμή *ψευδής*.

Παραδείγματα εκφράσεων AND:

- 0 && 10 && 5 : θα πάρει την τιμή 0 (false)
- 8 && 10 && 5 : θα πάρει την τιμή 5 (true)
- 5 && 10 && null : θα πάρει την τιμή null (false)

8.6 Προγραμματιστικές δομές της JavaScript

Στη συνέχεια θα δούμε τις βασικές προγραμματιστικές δομές που υποστηρίζει η JavaScript και υλοποιούνται ως εντολές για έλεγχο της ροής του προγράμματος, όπως οι if και switch, εντολές που επιτρέπουν την επαναληπτική εκτέλεση ενός κώδικα, όπως οι for και while, και τέλος εντολές που επιτρέπουν τη μετάβαση σε άλλο τμήμα του κώδικα, όπως οι break, continue και throw.

- Θα πρέπει να σημειώσουμε ότι η JavaScript συντακτικά ακολουθεί το παράδειγμα πολλών άλλων γλωσσών προγραμματισμού, όπως η C, C++, Java, ως προς τη σύνταξη των εντολών αυτών, τη χρήση του ";" ως τερματικού εντολής και τη χρήση των άγκιστρων { ... } για τον ορισμό ενός μπλοκ εντολών.
- Θα πρέπει να σχολιάσουμε, όμως, ότι οι ομοιότητες μεταξύ JavaScript και Java σταματούν εδώ, αφού πρόκειται για δύο πολύ διαφορετικές τεχνολογίες και το όνομα JavaScript είναι ατυχές αφού παραπέμπει στην Java, με την οποία η JavaScript δεν έχει ιδιαίτερη σχέση.
- Αν ένα μπλοκ εντολών περιέχει μία μόνο εντολή, τότε τα άγκιστρα μπορούν να παραληφθούν. Ωστόσο, πολλοί προγραμματιστές βάζουν πάντα τα άγκιστρα, ακόμη και στην περίπτωση του μπλοκ με μια εντολή, για να γίνει πιο σαφής η εμβέλεια κάθε εντολής.
- Επίσης, μπορούμε να ορίσουμε ως κενή εντολή την εντολή ";".
- Ως προς τη στοίχιση των εντολών, αυτή δεν είναι υποχρεωτική, όπως γίνεται για παράδειγμα στην Python, όμως είναι πολύ καλή πρακτική ένα μπλοκ εντολών να στοιχίζεται μέσα στα άγκιστρα που το ορίζουν, κάτι που επεξεργαστές κώδικα όπως ο *VS Code* κάνουν αυτόματα.

8.6.1 Εντολή if-else

Η πρώτη δομή που θα δούμε ότι επιτρέπει την υπό συνθήκη εκτέλεση ενός τμήματος του κώδικα είναι η εντολή if που συναντάται σε όλες τις γλώσσες προγραμματισμού.

Στη γενική περίπτωση αυτή η δομή συντάσσεται ως εξής:

```
if (έκφραση){
    μπλοκ-εντολών-1
}
else {
    μπλοκ-εντολών-2
}
```

Το μπλοκ-εντολών-1 θα εκτελεστεί αν η έκφραση είναι αληθής, ενώ το μπλοκ-εντολών-2 αν είναι ψευδής.

Θα πρέπει να παρατηρήσουμε εδώ ότι η έκφραση θα πρέπει να παίρνει μια τιμή η οποία να αντιστοιχεί είτε σε αληθή είτε σε ψευδή τιμή, σύμφωνα με τους κανόνες αληθότητας/ψευδότητας που έχουν ήδη αναφερθεί. Υπενθυμίζουμε ότι οι τιμές undefined, null, 0, -0, NaN, "" (κενή συμβολοσειρά), false αντιστοιχούν στην τιμή **«ψευδής»**, ενώ οποιαδήποτε άλλη τιμή σε **«αληθής»**. Για παράδειγμα, ένας κενός πίνακας [ ] αντιστοιχεί στην τιμή «αληθής», το ίδιο και ένα κενό αντικείμενο { }, κάτι που διαφοροποιεί την JavaScript από την Python που η κενή λίστα [ ] αντιστοιχεί στην τιμή «ψευδής».

Επίσης, θα πρέπει να παρατηρήσουμε ότι το τμήμα else της παραπάνω δομής μπορεί να παραληφθεί:

```
if (έκφραση){
    μπλοκ-εντολών-1
}
```

Ακόμη, μπορούμε να κάνουμε διαδοχικούς ελέγχους εισάγοντας στο τμήμα else έναν νέο έλεγχο if:

```
if (συνθήκη1){
    μπλοκ-εντολών-1 // αν συνθήκη1 αληθής
} else if (συνθήκη2) {
    μπλοκ-εντολών-2 // αν όχι συνθήκη1 και συνθήκη2 αληθής
} else if (συνθήκη3) {
    μπλοκ-εντολών-3 // αν όχι συνθήκη1, ούτε συνθήκη 2 και συνθήκη3 αληθής
} else {
    μπλοκ-εντολών-4 // αν όχι συνθήκη1, ούτε συνθήκη2, ούτε συνθήκη3
}
```

Ας δούμε ένα παράδειγμα:

```
let forecast = 'snowing';

if (forecast === 'sunny') {
  console.log('Πάμε για μπάνιο.');
} else if (forecast === 'rainy') {
  console.log('Να πάρουμε ομπρέλα');
} else if (forecast === 'snowing') {
  console.log('Να πάμε για σκι');
} else {
  console.log('Μένουμε σπίτι');
}
> 'Να πάμε για σκι'
```

Στο παράδειγμα η μεταβλητή forecast ελέγχεται για την τιμή της και ανάλογα τυπώνεται ένα μήνυμα στην κονσόλα. Αν η τιμή της μεταβλητής είναι διαφορετική από τις τιμές που ελέγχονται διαδοχικά στη δομή αυτή, π.χ. forecast="cloudy", θα πάρουμε το μήνυμα που τυπώνεται από το σκέλος else που ακολουθεί τους ελέγχους.

8.6.2 Εντολή switch

Αν πρέπει να κάνουμε μια σειρά από ελέγχους οι οποίοι αφορούν την ίδια μεταβλητή, όπως στο προηγούμενο παράδειγμα, μια καλύτερη επιλογή από διαδοχικά if - else if είναι να χρησιμοποιηθεί η δομή switch.

Η εντολή αυτή συντάσσεται ως εξής:

```
switch(n) {
case 1:  // if n === 1
    block-εντολών-1
```

```
        break;
    case 2: // if n === 2
        block-εντολών-2
        break;
    case 3: // if n === 3
        block-εντολών-3
        break;
    default: // αν δεν ισχύει καμιά από τις περιπτώσεις
        block-εντολών-4
        break;
    }
```

Σύμφωνα με το πρότυπο αυτό, το προηγούμενο παράδειγμα θα διαμορφωθεί ως ακολούθως:

```
switch (forecast) {
  case 'sunny':
    console.log('Πάμε για μπάνιο.');
    break;
  case 'rainy':
    console.log('Να πάρουμε ομπρέλα.');
    break;
  case 'snowing':
    console.log('Να πάμε για σκι.');
    break;
  default:
    console.log('Μένουμε σπίτι');
}
```

Να σημειωθεί εδώ ότι οι έλεγχοι που γίνονται σε καθεμία περίπτωση (case) είναι τύπου “strict equality” "===".

8.6.3 Υπό συνθήκη εκχώρηση τιμής (τριαδικός τελεστής)

Ένας εναλλακτικός τρόπος να εκχωρηθεί διαφορετική τιμή σε μια μεταβλητή ανάλογα με την τιμή μιας έκφρασης είναι με χρήση του τριαδικού τελεστή a ? b : c.

Ο τελεστής αυτός, που λέγεται τριαδικός επειδή παίρνει τρεις παραμέτρους, είναι συντομογραφία μας εντολής if-else, όπου a είναι μια έκφραση που παίρνει τιμή αληθής/ψευδής, b η τιμή αν η έκφραση είναι αληθής, και c η τιμή αν η έκφραση είναι ψευδής.

(έκφραση) ? τιμή-αν-αληθής : τιμή-αν-ψευδής

Ας δούμε ένα παράδειγμα:

Έστω ότι η μεταβλητή result εκφράζει αν ένας φοιτητής πήρε προβιβάσιμο βαθμό ή όχι. Παίρνει την τιμή «επιτυχία» αν ο βαθμός είναι μεγαλύτερος ή ίσος του 5 και την τιμή «αποτυχία» αν ο βαθμός είναι μικρότερος του 5.

Η παρακάτω εντολή if-else αποδίδει τιμή στη μεταβλητή result, ανάλογα με την τιμή της vathmos:

```
let result;
if (vathmos < 5){
    result = "αποτυχία";
}
else {
    result = "επιτυχία";
}
```

Η παραπάνω λογική μπορεί να απλοποιηθεί όμως με χρήση του τριαδικού τελεστή ως εξής:

```
let result = (vathmos<5) ? "αποτυχία" : "επιτυχία";
```

8.6.4 Διαχείριση σφαλμάτων με try-catch

Η JavaScript διαθέτει μηχανισμό για διαχείριση σφαλμάτων ή *εξαιρέσεων (exceptions)* όπως λέγονται. Ο

μηχανισμός αυτός έχει πολλά κοινά με την εντολή if-else. Πρόκειται για την εντολή try-catch-finally.

Η σύνταξη της εντολής αυτής είναι:

```
try {
    // κώδικας στον οποίο γίνεται έλεγχος εξαίρεσης
}
catch (e) {
    // κώδικας που θα τρέξει αν συμβεί εξαίρεση
}
finally {
    // κώδικας που θα τρέξει σε κάθε περίπτωση
}
```

Η εντολή catch ακολουθείται από ένα μπλοκ κώδικα στον οποίο γίνεται έλεγχος για κάποια εξαίρεση (απρόβλεπτη κατάσταση σφάλματος). Αν αυτή συμβεί, τότε θα εκτελεστεί ο κώδικας που ακολουθεί την εντολή catch(e). Η παράμετρος e στο catch περιέχει στοιχεία για το σφάλμα, είτε κάποιο μήνυμα είτε κωδικό σφάλματος. Το τμήμα finally είναι προαιρετικό.

Ένα παράδειγμα:

```
try {
    function factorial(num){
        if (num === 0) return 1;
        else return num * factorial( num - 1 );
    }
    let n = Number(prompt("Δώστε θετικό αριθμό", ""));
    let f = factorial(n);
    alert(n + "! = " + f);
}
catch(ex) {
    alert(ex);
}
```

Στο παράδειγμα αυτό μέσα στο τμήμα try ορίζουμε συνάρτηση αναδρομικού υπολογισμού του παραγοντικού n! = n*(n-1)*(n-2) ... * 1

Στο τμήμα αυτό του κώδικα ζητείται από τον χρήστη να δώσει θετικό αριθμό και το πρόγραμμα του επιστρέφει μέσω alert την τιμή του παραγοντικού. Αν όμως ο χρήστης δεν δώσει θετικό αριθμό, ενεργοποιείται το τμήμα catch που στέλνει στον χρήστη μέσω alert το μήνυμα σφάλματος. Για παράδειγμα, αν ο χρήστης δώσει αρνητικό αριθμό, το μήνυμα είναι: "RangeError: Maximum call stack size exceeded".

Μια εντολή που συνήθως εμφανίζεται στη δομή try-catch είναι η εντολή throw.

Η εντολή αυτή, όταν εμφανίζεται σε ένα μπλοκ try, είναι εντολή εξόδου από το μπλοκ και μετάβασης στο μπλοκ catch. Μοιάζει δηλαδή με την εντολή break που θα δούμε στους βρόχους επανάληψης στη συνέχεια.

Η σύνταξη της throw είναι:

```
throw έκφραση-σφάλματος;
```

Η έκφραση σφάλματος είναι είτε ένα αντικείμενο τύπου Error ή μια συμβολοσειρά που σχετίζεται με τη συγκεκριμένη εξαίρεση, για παράδειγμα:

```
if (x < 0) throw new Error("το x πρέπει να είναι θετικό");
```

Στο παράδειγμα αυτό η throw επιστρέφει ένα αντικείμενο τύπου Error στο οποίο περνάμε μια συμβολοσειρά σφάλματος.

8.7 Δομές επανάληψης

Στην ενότητα αυτή θα δούμε τις προγραμματιστικές δομές που διαθέτει η JavaScript για επαναληπτική εκτέλεση ενός τμήματος του κώδικα. Οι εντολές που επιτρέπουν την επαναληπτική εκτέλεση ενός μπλοκ κώδικα είναι οι while, do/while, for, for/in, for/of. Η πιο τυπική χρήση μιας δομής επανάληψης είναι όταν ζητείται να διαπεράσουμε τα στοιχεία ενός πίνακα.

8.7.1 Εντολή while

Η πιο απλή εντολή επαναληπτικής εκτέλεσης κώδικα είναι η while, που συναντάται σε πολλές γλώσσες προγραμματισμού. Η while συντάσσεται ως εξής:

```
while (συνθήκη) {
    μπλοκ-εντολών
}
```

Ο διερμηνευτής της JS μόλις φτάσει στην εντολή αυτή υπολογίζει την τιμή της *συνθήκης*. Αν είναι ψευδής, παραλείπει το μπλοκ εντολών και προχωράει στην επόμενη εντολή του προγράμματος. Αν η συνθήκη είναι αληθής, εκτελεί το μπλοκ εντολών και επανέρχεται στην κορυφή στον εκ νέου έλεγχο της συνθήκης.

Είναι φανερό ότι αν κάτι δεν αλλάξει στο μπλοκ εντολών σε σχέση με τη συνθήκη και η συνθήκη παραμείνει αληθής, προκαλείται ατέρμων βρόχος επανάληψης.

Ας δούμε ένα παράδειγμα μιας εντολής while που επιτρέπει να τυπωθούν οι αριθμοί από 0 μέχρι 4.

```
let count = 0;
while(count < 5) {
    console.log(count++);
}
```

Το πρόγραμμα τυπώνει τους αριθμούς αρχίζοντας από το 0 και αυξάνοντας τον μετρητή count σε κάθε επανάληψη. Όταν τυπωθεί και ο αριθμός 4 και αυξηθεί ο μετρητής count σε 5, τότε η συνθήκη count < 5 είναι ψευδής και τερματίζει η επαναληπτική εκτέλεση του μπλοκ εντολών.

Μια παραλλαγή της δομής αυτής είναι η while(true)/break. Στην περίπτωση αυτή θέτουμε ως συνθήκη μια πάντα αληθή έκφραση και ελέγχουμε τη συνθήκη εξόδου μέσα στο μπλοκ εντολών, όταν δε η συνθήκη εξόδου ικανοποιηθεί, τότε εκτελούμε την εντολή break που μας στέλνει εκτός του βρόχου. Θυμίζουμε ότι έχουμε δει την εντολή αυτή στην περίπτωση της εντολής case. Παράδειγμα του προηγούμενου προγράμματος με αυτή τη δομή είναι:

```
let count = 0;
while(true) {
    console.log(count++);
    if (count >= 5) break;
}
```

Η break χρησιμοποιείται γενικότερα για να μεταφέρουμε τον έλεγχο εκτέλεσης του προγράμματος σε άλλο σημείο, όπως εκτός ενός βρόχου επανάληψης. Μάλιστα, μπορούμε να ορίσουμε το όνομα της εντολής στην οποία μεταφέρεται ο έλεγχος του προγράμματος προς μια εντολή που φέρει όνομα: όνομα: εντολή; και να συνοδεύσουμε την εντολή break με το όνομα του στόχου: break όνομα; (δομή αντίστοιχη της goto που δεν συναντάται πλέον συχνά στον προγραμματισμό). Γενικά όμως θεωρείται όχι καλή πρακτική η χρήση της break, πλην της περίπτωσης εξόδου από έναν βρόχο επανάληψης.

Επίσης, μια άλλη εντολή που σχετίζεται με επαναληπτικές δομές είναι η εντολή continue. Η εντολή αυτή όταν βρεθεί μέσα σε ένα μπλοκ εντολών ενός βρόχου κάνει τον διερμηνευτή του προγράμματος να παραλείψει τις υπόλοιπες εντολές του βρόχου και να μεταβεί στην επόμενη επανάληψη.

Ένα παράδειγμα χρήσης της break για έξοδο από έναν βρόχο είναι η αναζήτηση μιας τιμής target σε έναν πίνακα:

```
let i = 0;
while (i < ar.length){
    if (ar[i++] === target) break;
}
```

Θα πρέπει να σημειωθεί ότι οι εντολές break και continue μπορούν να χρησιμοποιηθούν σε όλες τις δομές επανάληψης που περιγράφονται στη συνέχεια.

8.7.2 Εντολή do/while

Η do/while είναι μια παραλλαγή της while που επίσης συναντάται σε πολλές γλώσσες προγραμματισμού

(εξαίρεση η Python που δεν διαθέτει δομή do/while). Η διαφορά από τη while είναι ότι ο έλεγχος της συνθήκης γίνεται στο τέλος του βρόχου επανάληψης, και συνεπώς το μπλοκ εντολών εκτελείται τουλάχιστον μια φορά, ακόμη και αν η συνθήκη είναι ψευδής.

Η σύνταξη της do/while είναι:

```
do {
    μπλοκ-εντολών
} while (συνθήκη);
```

Η χρήση της δομής αυτής είναι πιο σπάνια από της while. Βεβαίως, και εδώ ισχύει ο κίνδυνος του ατέρμονος βρόχου σε περίπτωση που η συνθήκη παραμείνει αληθής χωρίς να επηρεάζεται από το μπλοκ εντολών.

Το προηγούμενο παράδειγμα εκτύπωσης των αριθμών 0 μέχρι 4 με αυτή τη δομή είναι:

```
let count = 0;
do {
  console.log(count++);
} while (count < 5);
```

8.7.3 Εντολή for

Η δομή for, η οποία όπως θα δούμε στη συνέχεια εμφανίζεται σε διάφορες παραλλαγές, είναι λίγο πιο σύνθετη από τη while αλλά χρησιμοποιείται πολύ πιο συχνά στον προγραμματισμό. Είναι μια δομή που συναντάται σε όλες τις γλώσσες προγραμματισμού.

Η πιο συνήθης έκδοση της for απαιτεί την ύπαρξη μιας βοηθητικής μεταβλητής, του μετρητή. Η μεταβλητή αυτή αρχικοποιείται πριν αρχίσει η εκτέλεση των επαναλήψεων, ελέγχεται σύμφωνα με κάποια συνθήκη στην αρχή κάθε επανάληψης και στο τέλος κάθε επανάληψης μεταβάλλεται (συνήθως αυξάνεται). Αυτές οι τρεις ενέργειες κωδικοποιούνται σε μια εντολή for ως εξής:

```
for (αρχικοποίηση; έλεγχος; τροποποίηση) {
    μπλοκ-εντολών
}
```

Αν προσπαθήσουμε να επαναλάβουμε το παράδειγμα των προηγούμενων ενοτήτων, να τυπώσουμε δηλαδή τους αριθμούς από 0 .. 4, ακολουθεί η υλοποίησή του με χρήση της δομής for:

```
for (let count = 0; count < 5; count++) {
    console.log(count);
}
```

Η πιο συνήθης χρήση της δομής αυτής είναι στη διαπέραση ενός πίνακα, όπως στο παράδειγμα:

```
const fruits = ['Μπανάνα', 'Αχλάδι', 'Μήλο', 'Μάνγκο'];
for (let i = 0; i < fruits.length; i++) {
  console.log(`${i + 1}. ${fruits[i]} `);
}
```

Ο κώδικας αυτός θα διαπεράσει τον πίνακα των φρούτων και θα τα τυπώσει:

```
1. Μπανάνα
2. Αχλάδι
3. Μήλο
4. Μάνγκο
```

Με την ευκαιρία, να υπενθυμίσουμε ότι αν η μεταβλητή μετρητής οριστεί με τη λέξη let, έχει εμβέλεια μόνο μέσα στο μπλοκ for.

Τέλος, να αναφέρουμε ότι μπορούμε να παραλείψουμε κάποια από τις εκφράσεις της for ή και όλες τις εκφράσεις, όμως θα πρέπει να διατηρήσουμε τα σύμβολα “;”. Για παράδειγμα, μια δομή που είναι ισοδύναμη με while (true) θα μπορούσε να γραφτεί ως δομή for ως εξής:

```
for (;;){
    μπλοκ-εντολών
}
```

8.7.4 Εντολή for/of

Η δομή for/of εισήχθη στην JavaScript με την έκδοση ES6. Είναι δομή που θυμίζει την αντίστοιχη δομή for της Python.

Η δομή αυτή επιτρέπει να διαπεράσουμε μια ακολουθία στοιχείων. Μια ακολουθία είναι, για παράδειγμα, ένας πίνακας ή μια συμβολοσειρά.

Η σύνταξη της εντολής αυτής είναι:

```
for (let στοιχείο of ακολουθία) {
    μπλοκ εντολών
}
```

Ο βρόχος επαναλαμβάνεται για κάθε στοιχείο της ακολουθίας.

Ένα παράδειγμα:

```
const fruits = ['Μπανάνα', 'Αχλάδι', 'Μήλο', 'Μάνγκο'];
for (let fruit of fruits) {
  console.log(fruit);
}
```

Εδώ θα πάρουμε ένα προς ένα τα ονόματα των φρούτων.

Ας δούμε ένα ακόμη παράδειγμα διαπέρασης των χαρακτήρων μιας συμβολοσειράς στο πρόβλημα καταγραφής συχνότητας εμφάνισης χαρακτήρων.

```
let freq = {}; // αντικείμενο JavaScript
for (let letter of 'Ελλάδα') {
  freq[letter] = (freq[letter] || 0) + 1;
}
freq;
> { 'Ε': 1, 'λ': 2, 'ά': 1, 'δ': 1, 'α': 1 }
```

Μερικές παρατηρήσεις για τη λύση αυτή: Χρησιμοποιούμε ένα αντικείμενο, το freq (θα γίνουν οι επίσημες συστάσεις αντικειμένων σε επόμενο κεφάλαιο, για την ώρα θεωρήστε ότι είναι παρόμοιο με ένα λεξικό της Python). Για κάθε γράμμα, αν το γράμμα υπάρχει ήδη στο αντικείμενο freq, προστίθεται στην τιμή της ιδιότητας με τιμή τον χαρακτήρα +1, αν δεν υπάρχει, δημιουργείται η ιδιότητα και αρχικοποιείται σε 1. Αυτό γιατί η έκφραση: a || b παίρνει την τιμή του πρώτου στοιχείου της έκφρασης το οποίο είναι αληθές, αλλιώς του τελευταίου στοιχείου.

8.7.5 Εντολή for/in

Η δομή for/in μοιάζει με τη for/of που είδαμε στην προηγούμενη ενότητα, υπάρχει στην JavaScript από παλαιότερες εκδόσεις και έχει ευρύτερη χρήση από την προηγούμενη, αφού στην περίπτωση αυτή το στοιχείο μπορεί να είναι οι ιδιότητες ενός αντικείμενου, όχι απαραίτητα μιας ακολουθίας.

Η σύνταξη είναι:

```
for (let στοιχείο in αντικείμενο) {
    μπλοκ εντολών
}
```

Για τους πίνακες και τις συμβολοσειρές το *στοιχείο* παίρνει τις τιμές του δείκτη που, ως γνωστόν, παίρνει τιμές από 0 μέχρι length-1.

Συνεπώς, το παράδειγμα της προηγούμενης ενότητας παρουσίασης των αγαπημένων μας φρούτων τροποποιείται ως ακολούθως:

```
const fruits = ['Μπανάνα', 'Αχλάδι', 'Μήλο', 'Μάνγκο'];
for (let indx in fruits) {
  console.log(fruits[indx]);
}
```

Είναι αντιληπτό από το παράδειγμα ότι η δομή for/in δεν εκφράζει με την ίδια απλότητα τη διαπέραση των στοιχείων ενός πίνακα. Για τον λόγο αυτό, για την περίπτωση πινάκων ή συμβολοσειρών, που είναι και η πιο συνήθης περίπτωση χρήσης του βρόχου for, η νεότερη δομή for/of είναι προτιμότερη.

8.8 Ερωτήσεις αυτοαξιολόγησης

1. Τι θα επιστρέψει ο παρακάτω κώδικας:

```
let a;
console.log(a==null);
```

    1. true
    2. false
    3. “true”
    4. “error”
    5. Syntax Error

2. Έστω σε αρχείο HTML το οποίο δεν περιέχει υπερσυνδέσμους, ποια η τιμή της μεταβλητής a;
let a = document.querySelector("a");
Απάντηση: ……

3. Τι θα επιστρέψει ο παρακάτω κώδικας:

```
let a;
console.log(typeof a);
```

    1. 'null'
    2. 'undefined'
    3. null
    4. undefined
    5. Syntax Error

4. Τι θα επιστρέψει ο κώδικας:

```
let b = 5;
typeof(b>1);
```

    1. ‘boolean’
    2. true
    3. false
    4. ‘true’

5. Ποιο το αποτέλεσμα;

```
let a = 10;
let b = 9;
console.log(a>10 || b>=9 )
```

Απάντηση: _____

6. Ποιο το αποτέλεσμα;

```
let a = 8
let b = 2
console.log(a | b)
```

Απάντηση: _____

7. Ποιο το αποτέλεσμα;

```
let a = 8
console.log(a << 2)
```

Απάντηση: _____

8. Ποιο το αποτέλεσμα;

```
let a = -10/0;
console.log(a)
```

    1. NaN

2. Infinity
3. -Infinity
4. ZeroDivisionError

9. Ποιο το αποτέλεσμα: 10 == '10'
   Απάντηση: _____
10. Ποιο το αποτέλεσμα;

```
for(let i = 0; i<10; i++) {}
console.log(i);
```

1. 10
2. 11
3. ReferenceError

11. Ποιο το αποτέλεσμα;

```
console.log('x=', typeof x, x);
var x=1;
```

1. ReferenceError: x is not defined
2. undefined undefined
3. ‘undefined’ undefined

12. Ποιο το αποτέλεσμα;

```
let x =  10;
let x = 5;
console.log(x);
```

1. Duplicate declaration
2. 10
3. 5
4. 15

13. Ποια η τιμή του y;

```
const x=1;
let y = x++;
```

1. 1
2. 2
3. TypeError
4. -1

14. Ποιο το αποτέλεσμα; typeof (10+'10')
    1. TypeError
    2. ‘number’
    3. ‘string’
    4. integer
15. Ποιο το αποτέλεσμα;
    console.log(3 > Number(1+'1'))
    Απάντηση: _____
16. Ποιο το αποτέλεσμα;
    console.log(Number("καλημέρα".length+ "5"));
    Απάντηση: _____
17. Ποιο το αποτέλεσμα;

```
let myCountry="Greece"
console.log(myCountry.slice(4))
```

    Απάντηση: _____
18. Ποιο το αποτέλεσμα;

```
let myCountry="Greece"
console.log(myCountry.indexOf('rec'))
```

Απάντηση: _____

19. Ποιο το αποτέλεσμα;

```
console.log(`${3+2}${3*2}`);
```

Απάντηση: _____

20. Ποιο το αποτέλεσμα;

```
let x = 10;
if (0<x && x<5){console.log('a')}
else if (5<=x && x<=10){console.log('b')}
else console.log('c')
```

Απάντηση: _____

21. Ποιο το αποτέλεσμα;

```
let x = 1;
console.log((typeof x === 'string')?1:2);
```

Απάντηση: _____

22. Αν έχουμε να επιλέξουμε μεταξύ 5 διαφορετικών τιμών μιας μεταβλητής x, ποια η δομή που θα ήταν καλύτερο να χρησιμοποιήσουμε;
    1. if (x==v1) {} else if (x==v2){} ...
    2. try {x==v1} catch { ... }
    3. (x==v1) ? v2 : v3;
    4. switch(x){case v1: ...}
23. Ποιο το αποτέλεσμα;

```
for (var _i=0; _i<10; _i+=3){}
console.log(_i)
```

Απάντηση: _____

24. Ποιο το αποτέλεσμα;

```
let s = "";
for (let x=10; x>5 ;x--){s+=x}
console.log(s);
```

    1. ‘10 9 8 7 6 5’
    2. ‘10,9,8,7,6,5’
    3. ‘1098765’
    4. ‘10 9 8 7 6’
    5. ‘109876’

25. Συμπληρώστε τον κώδικα ώστε το αποτέλεσμα να είναι 123.

```
let s=[120,121,122]
for (let i of s){
    if (s.indexOf(i) == s.length-1)console.log( ......)}
```

Απάντηση: _____

26. Ποιο το αποτέλεσμα;

```
let ar=[5,10,15]
let s=''
for (let i in ar){s+=i}
console.log(s)
```

1. '51015'
2. '5 10 15'
3. '012'

27. Συμπληρώστε τον κώδικα ώστε το αποτέλεσμα να είναι “123”.

```
let s='';
let i=1;
while (true){
    s+=i
    ......
    i++}
console.log(s);
```

1. if(i>2)break;
2. if(i>3)break;
3. if(i>4)break;

28. Έστω ο παρακάτω κώδικας

```
let i=1;
while (i<10){
    i++;
    if (i%2 ===0 || i%3===0 || i%5===0)continue;
    console.log(i);
}
```

Ποιο το αποτέλεσμα;

1. Θα τυπώσει τις τιμές 1 έως 9.
2. Θα τυπώσει τις τιμές 1 7.
3. Θα τυπώσει την τιμή 7.
4. Θα τυπώσει τις τιμές 1 3 7.

29. Από τις δομές for, while και do while, ποια είναι αυτή που θα εκτελεστεί τουλάχιστον μια φορά;
    1. Καμιά από τις 3.
    2. Η δομή for.
    3. Η δομή while.
    4. Η δομή do while.

8.9 Βιβλιογραφία και Αναφορές

Και στο κεφάλαιο αυτό ισχύει η βιβλιογραφία του προηγούμενου κεφαλαίου. Το πρότυπο EcmaScript (ECMA-262) συντηρείται από τον οργανισμό ecma.

Στο διαδίκτυο υπάρχουν πολλές πηγές για εκμάθηση της JavaScript καθώς και για αναφορά στα στοιχεία της γλώσσας. Η MDN είναι μια καλή πηγή για εισαγωγικά και προχωρημένα μαθήματα, όπως και για την HTML και τη CSS. Επίσης, η w3schools περιέχει μαθήματα, ενώ μια πλήρη σειρά μαθημάτων για την JavaScript με παραδείγματα περιλαμβάνει η JavaScript.info, ξεκινώντας από τη γλώσσα και προχωρώντας στη διεπαφή της με τον φυλλομετρητή.

Η JavaScript περιλαμβάνεται σε βιβλία που ήδη αναφέρθηκαν που αφορούν τις τεχνολογίες του προγραμματισμού στον ιστό όπως η HTML και CSS. Αυτή την προσέγγιση στην ελληνική βιβλιογραφία ακολουθεί το βιβλίο των Δουληγέρη κ.ά. (2021), ενώ έχουν μεταφραστεί κάποια συγγράμματα και διατίθενται από τον Εύδοξο, όπως το βιβλίο των Kyrnin και Morrison (2021) και αυτό των Lemay, Coburn και Kyrnin (2016).

Μια άλλη προσέγγιση είναι αυτή εγχειριδίων που ξεκινούν με τη σύνταξη της γλώσσας JavaScript και περιλαμβάνουν τη διεπαφή με το DOM σε μεταγενέστερο κεφάλαιο. Στις πηγές αυτές συχνά περιλαμβάνονται και κεφάλαια που αφορούν τη λειτουργία της γλώσσας στο περιβάλλον node.js. Αυτή την προσέγγιση ακολουθεί το βιβλίο του Λιακέα (2021). Από τη διεθνή βιβλιογραφία παρόμοια προσέγγιση ακολουθεί το βιβλίο του Flanagan (2020), ενώ υπάρχουν και πολλά άλλα, όπως Frisbie (2020) κ.λπ.

Επιπλέον, οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Για αυτό το κεφάλαιο το σχετικό μάθημα είναι οι εισαγωγικές διαλέξεις του μαθήματος «Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, Javascript».

A. Ξενόγλωσσες

Flanagan, D. (2020). *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language* (7th ed.). O'Reilly Media, Inc.

Frisbie, M. (2020). *Professional JavaScript for Web Developers*. Wiley.

McFedries, P. (2019). *Web Design Playground - HTML & CSS The Interactive Way*. Manning.

B. Ελληνόγλωσσες

Αβούρης, Ν. (2018). Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, JavaScript. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr

Δουληγέρης, Χ., Μαυροπόδη, Ρ., Κοπανάκη, Ε., & Καραλής, Α. (2021). *Τεχνολογίες και Προγραμματισμός στον Παγκόσμιο Ιστό* (2η έκδ.). Εκδόσεις Νέων Τεχνολογιών. Κωδικός βιβλίου στον Εύδοξο: 102125023.

Kyrnin, J., & Meloni, J. (2021). *Sams Teach Yourself HTML5, CSS and Javascript* (3rd ed.). Εκδόσεις Γκιούρδας.

Lemay, L., Coburn, R., & Kyrnin, J. (2016). *Sams Teach Yourself HTML, CSS & JavaScript* (7th ed). Εκδόσεις Γκιούρδας. Κωδικός βιβλίου στον Εύδοξο: 59357307.

Λιακέας, Γ. (2021). *Η γλώσσα JavaScript* (3η έκδ.). Κλειδάριθμος. Κωδικός βιβλίου στον Εύδοξο: 102070465.

Κεφάλαιο 9. Τύποι δεδομένων αναφοράς: Πίνακες, Συναρτήσεις και Αντικείμενα στην JavaScript

Σύνοψη

*Στο κεφάλαιο αυτό, το τρίτο που έχει ως αντικείμενο τη γλώσσα προγραμματισμού JavaScript, θα δούμε διάφορους **τύπους δεδομένων αναφοράς (reference data types)** της JavaScript.*

*Αρχικά θα δούμε τον τύπο Array (πίνακας), ως πρώτο τύπο δεδομένων αναφοράς. Θα γίνει ιδιαίτερη αναφορά στις μεθόδους του τύπου Array και στους τρόπους ορισμού πινάκων. Οι πίνακες είναι ειδική κατηγορία αντικειμένων (objects). Στη συνέχεια θα εστιάσουμε στη συναρτησιακή λειτουργία της γλώσσας, που ορίζει τις συναρτήσεις ως αντικείμενα πρώτης τάξης. Επίσης, θα δούμε τους τρόπους που διαθέτουμε στην JS για δημιουργία **αντικειμένων** και προγραμματισμό με το αντικειμενοστραφές μοντέλο.*

Με αυτό το κεφάλαιο θα καλύψουμε σε μεγάλο βαθμό τον πυρήνα της γλώσσας. Αν πρόκειται να χρησιμοποιήσουμε την JavaScript μόνο στην πλευρά του φυλλομετρητή, τα τρία αυτά κεφάλαια αρκούν. Όμως, επειδή στο εγχειρίδιο αυτό προτείνεται η χρήση της JavaScript και στην πλευρά του εξυπηρετητή, χρειάζεται να γίνει αναφορά σε πιο σύνθετα στοιχεία της, όπως η ασύγχρονη λειτουργία και ο χειρισμός των συμβάντων. Αυτά θα καλυφθούν στο επόμενο κεφάλαιο. Επίσης, να σημειωθεί ότι η JavaScript θα είναι το βασικό μας εργαλείο όλων των επόμενων κεφαλαίων του βιβλίου.

Προαπαιτούμενη γνώση

Είναι απαραίτητη η εξοικείωση με τα κεφάλαια 7 και 8.

9.1 Τύποι δεδομένων αναφοράς

Ως τώρα έχουμε δει τους βασικούς πρωτογενείς τύπους δεδομένων (primitive data types). Στο κεφάλαιο αυτό θα δούμε κάποιους άλλους τύπους δεδομένων που ονομάζονται **τύποι δεδομένων αναφοράς (reference data types)**. Οι τύποι δεδομένων αναφοράς είναι τα αντικείμενα (objects), οι πίνακες (arrays), αλλά και οι συναρτήσεις (functions) που και αυτές, όπως θα δούμε, είναι αντικείμενα. Υπάρχουν κάποιες σημαντικές διαφορές μεταξύ των δεδομένων αναφοράς και των πρωτογενών δεδομένων: Τα πρωτογενή δεδομένα είναι μη τροποποιήσιμα (immutable), ενώ τα δεδομένα αναφοράς μπορούν να τροποποιηθούν. Τα δεδομένα αναφοράς έχουν ιδιότητες, κάτι που δεν έχουν τα πρωτογενή δεδομένα. Επίσης, η αντιγραφή ενός πρωτογενούς δεδομένου a σε μια νέα μεταβλητή (π.χ. let b = a;) δημιουργεί νέο αντίγραφο της τιμής του πρωτογενούς τύπου, ενώ στα δεδομένα αναφοράς δημιουργεί μια νέα αναφορά στο ήδη υπάρχον αντικείμενο. Αυτό, όπως θα δούμε, έχει ιδιαίτερη σημασία στις μεταβλητές που περνάμε στα ορίσματα συναρτήσεων.

Στο κεφάλαιο αυτό θα αρχίσουμε με τους **πίνακες (arrays)**. Στη συνέχεια θα δούμε τον τρόπο που η JavaScript χειρίζεται τις **συναρτήσεις (functions)** και τα **αντικείμενα (objects)**.

Οι συναρτήσεις, οι οποίες και αυτές είναι αντικείμενα πρώτης τάξης εκτός από τον ρόλο που παίζουν στη δόμηση ενός προγράμματος, χρησιμοποιούνται ως βασικό συστατικό του *συναρτησιακού μοντέλου προγραμματισμού (functional programming)*, το οποίο είναι ιδιαίτερα διαδεδομένο στην JavaScript.

Τέλος, τα αντικείμενα αποτελούν θεμελιώδες χαρακτηριστικό της αρχιτεκτονικής της JavaScript και χρησιμοποιούνται εκτενώς σε εφαρμογές που ακολουθούν το *αντικειμενοστραφές μοντέλο προγραμματισμού (object oriented programming)*.

9.2 Πίνακες

Ο πρώτος τύπος δεδομένων αυτής της κατηγορίας που θα δούμε είναι οι **πίνακες (Arrays)**. Ένας πίνακας είναι μια διαταγμένη ακολουθία στοιχείων. Κάθε στοιχείο βρίσκεται σε συγκεκριμένη θέση στην ακολουθία. Η θέση του στοιχείου ορίζεται από έναν δείκτη *(index)*, ένα ακέραιο αριθμό που παίρνει τιμές από 0 (το πρώτο στοιχείο) μέχρι length-1, όπου length το πλήθος στοιχείων του πίνακα. Οι πίνακες συναντώνται σε όλες σχεδόν τις γλώσσες προγραμματισμού, π.χ. στην Python ο αντίστοιχος τύπος δεδομένων είναι οι λίστες. Οι πίνακες της JavaScript είναι δυναμικοί τύποι δεδομένων, το μέγεθός τους δεν χρειάζεται να οριστεί κατά τη δημιουργία τους, καθώς μπορούν να προστεθούν ή να διαγραφούν στοιχεία (είναι τροποποιήσιμος τύπος δεδομένων).

Τα στοιχεία του πίνακα μπορεί να είναι διαφορετικών τύπων μεταξύ τους, μπορεί να είναι πρωτογενή δεδομένα ή αντικείμενα ή και άλλοι πίνακες.

Όπως θα δούμε στη συνέχεια, τα αντικείμενα τύπου Array κληρονομούν μεθόδους (ορίζονται ως ιδιότητες στο Array.prototype) με τις οποίες μπορούμε να διαχειριστούμε τα στοιχεία του πίνακα.

Οι πίνακες μπορεί να δημιουργηθούν είτε με ορισμό τιμής τους (Array literal) ή με κλήση της δημιουργού συνάρτησης νέων αντικειμένων new Array().

9.2.1 Δημιουργία πίνακα με ορισμό της τιμής του

Ας δούμε μερικά παραδείγματα δημιουργίας πινάκων με ορισμό των τιμών τους.

```
const shopping = ['ψωμί', 'γάλα', 'τυρί', 'μήλα'];
const sequence = [1, 1, 2, 3, 5, 8, 13];
const random = ['tree', 795, [0, 1, 2]];
const empty = [];
const sparse = [1, , ,];
```

Όπως βλέπουμε στο παράδειγμα, ο πίνακας shopping είναι πίνακας που περιέχει 4 συμβολοσειρές. Το μήκος του πίνακα shopping.length είναι 4. Ο πίνακας sequence είναι πίνακας ακέραιων με 7 στοιχεία (sequence.length = 7). Ο πίνακας random περιέχει τρία στοιχεία διαφορετικών τύπων μεταξύ τους, μια συμβολοσειρά, έναν ακέραιο και έναν πίνακα (random.length = 3). Ο πίνακας empty είναι πίνακας χωρίς στοιχεία, συνεπώς empty.length = 0. Τέλος, ο πίνακας sparse έχει τρία στοιχεία, εκ των οποίων το πρώτο έχει οριστεί, ενώ τα άλλα δύο όχι, παρ' όλα αυτά το μήκος του είναι sparse.length = 3.

9.2.2 Δημιουργία πίνακα με new Array()

Ένας εναλλακτικός τρόπος δημιουργίας πίνακα είναι με κλήση της δημιουργού συνάρτησης του αντικείμενου Array:

```
const ar = new Array();
```

Όπως θα δούμε σε επόμενο κεφάλαιο, αυτός είναι ο συνήθης τρόπος δημιουργίας αντικειμένων με κλήση της συνάρτησης-δημιουργού με τη λέξη **new**.

Ο πίνακας ar που δημιουργείται από την παραπάνω εντολή είναι απολύτως ισοδύναμος με την εντολή δημιουργίας πίνακα με ορισμό τιμής:

```
const ar = [];
```

Στη δημιουργό συνάρτηση Array() μπορούμε να περάσουμε ως όρισμα το μήκος του πίνακα που δημιουργούμε.

```
const ar = new Array(50);
ar.length;
> 50
```

Σε αυτή την περίπτωση δημιουργείται ένας κενός πίνακας με μήκος 50.

Εναλλακτικά, στα ορίσματα της συνάρτησης δημιουργού μπορούμε να περάσουμε τα στοιχεία του πίνακα:

```
const ar = new Array(10,20);
ar.length;
> 2
```

Στην περίπτωση αυτή δημιουργείται ένας πίνακας με δύο στοιχεία, άρα μήκους 2.

9.2.3 Αραιοί πίνακες

Μια ιδιαιτερότητα της JavaScript είναι ότι μπορούμε σε έναν πίνακα να εισαγάγουμε στοιχεία πέραν της τρέχουσας τιμής του δείκτη, και άρα του μήκους του πίνακα.

Ας δούμε ένα παράδειγμα:

```
const ar = [1,2,3];
undefined
ar[10] = 20;
20
console.log(ar)
(11) [1, 2, 3, empty × 7, 20]
ar.length
11
```

Στο παράδειγμα ορίζουμε έναν πίνακα τριών στοιχείων. Στη συνέχεια, στην 11η θέση του (δείκτης 10) βάζουμε ένα νέο στοιχείο. Τότε το μήκος του πίνακα γίνεται 11, δηλαδή λαμβάνει υπόψη τη θέση του τελευταίου στοιχείου, αν και ενδιάμεσα υπάρχουν 7 άδειες θέσεις. Ο πίνακας αυτός λέγεται αραιός (sparse).

Να σημειωθεί ότι σε άλλες γλώσσες προγραμματισμού αυτή η συμπεριφορά δεν θα ήταν επιτρεπτή, για παράδειγμα η εισαγωγή στοιχείου στη θέση 10 ενός πίνακα 3 στοιχείων στην Python θα έδινε IndexError.

9.2.4 Αρχικοποίηση πίνακα

Αν επιθυμούμε να αρχικοποιήσουμε έναν πίνακα μήκους length, μπορούμε, εκτός από τον κλασικό τρόπο με χρήση μιας δομής επανάληψης, να χρησιμοποιήσουμε τη μέθοδο fill(). Για παράδειγμα, αν θέλουμε να δώσουμε ως αρχική τιμή σε όλα τα στοιχεία ενός πίνακα την τιμή 5:

```
const ar = new Array(10);
ar.fill(5);
console.log(ar)
> [
  5, 5, 5, 5, 5,
  5, 5, 5, 5, 5
]
```

9.2.5 Τροποποίηση στοιχείων πίνακα

Οι πίνακες είναι τροποποιήσιμες δομές δεδομένων.

Με χρήση του συμβολισμού δείκτη πίνακας[δείκτης] μπορούμε να ορίσουμε ένα νέο στοιχείο ή να τροποποιήσουμε ένα στοιχείο του πίνακα που ήδη υπάρχει.

Επίσης, μπορούμε να διαγράψουμε ένα στοιχείο με τον τελεστή delete, όπως μπορούμε να διαγράψουμε τις ιδιότητες κάθε αντικείμενου της JavaScript.

Θα πρέπει να σημειωθεί ότι οι πράξεις αυτές δεν επηρεάζουν το μήκος του πίνακα, που παραμένει ίσο με την τιμή του μέγιστου δείκτη + 1.

Επίσης, το περιεχόμενο ενός πίνακα μπορεί να τροποποιηθεί με χρήση μεθόδων της κλάσης Array, όπως οι push(), pop(), shift(), unshift(), και θα περιγραφεί στην επόμενη ενότητα.

9.2.6 Μέθοδοι πινάκων

Ο τύπος δεδομένων Array διαθέτει πολύ ισχυρές μεθόδους για διαχείριση και τροποποίηση των στοιχείων ενός πίνακα. Στην ενότητα αυτή θα εστιάσουμε στις μεθόδους τροποποίησης και ταξινόμησης των στοιχείων, ενώ σε επόμενο κεφάλαιο, όταν εισαγάγουμε τις έννοιες του συναρτησιακού προγραμματισμού, θα δούμε μεθόδους για επαναληπτική εφαρμογή συναρτήσεων σε πίνακες.

Μέθοδοι στοίβας (stack)

Οι μέθοδοι push(στοιχείο) και pop() επιτρέπουν την εισαγωγή και διαγραφή στοιχείων στο τέλος ενός πίνακα, ώστε να δώσουν στον πίνακα συμπεριφορά στοίβας LIFO (last in first out).

Παρόμοια λειτουργία έχουν οι μέθοδοι shift() και unshift(στοιχείο) που αντίστοιχα διαγράφουν ή εισάγουν στοιχείο στον πίνακα, μόνο που αυτές το κάνουν στην αρχή του πίνακα και όχι στο τέλος, όπως οι push, pop.

Θα πρέπει να σημειωθεί ότι η push() επιστρέφει το νέο μήκος του πίνακα, ενώ η pop() επιστρέφει το στοιχείο που διαγράφτηκε. Ας δούμε μερικά παραδείγματα.

```
const myArray = ["Athens", "Patras", "Thessaloniki", "Volos"]
myArray.push('Larissa'); // επιστρέφει 5
myArray.push('Katerini', 'Kavala'); // επιστρέφει 7
myArray.pop(); // επιστρέφει "Kavala"
```

Αντίστοιχη λειτουργία έχουμε με τις μεθόδους shift(), unshift():

```
const myArray = ["Athens", "Patras", "Thessaloniki", "Volos"]
myArray.unshift('Larissa'); // επιστρέφει 5
myArray.unshift('Katerini', 'Kavala'); // επιστρέφει 7
myArray.shift(); // επιστρέφει "Katerini"
```

9.2.7 Μέθοδοι υποπινάκων

Η μέθοδος slice(), που την έχουμε δει και στις συμβολοσειρές, μας επιτρέπει να πάρουμε μια «φέτα» του πίνακα, ένα υποσύνολο των στοιχείων, και να δημιουργήσουμε έναν νέο πίνακα.

Ένα παράδειγμα:

```
const year = ["Ιανουάριος", "Φεβρουάριος", "Μάρτιος",
    "Απρίλιος", "Μάιος", "Ιούνιος", "Ιούλιος", "Αύγουστος",
    "Σεπτέμβριος", "Οκτώβριος", "Νοέμβριος", "Δεκέμβριος"];

const spring = year.slice(2,5);
console.log(spring);

> ["Μάρτιος", "Απρίλιος", "Μάιος"]
```

Η μέθοδος slice() επιστρέφει έναν νέο πίνακα. Παίρνει δύο ορίσματα, τον δείκτη του πρώτου στοιχείου του υποπίνακα και τον δείκτη του τελευταίου στοιχείου, το οποίο όμως δεν περιλαμβάνεται. Αν παραληφθεί το δεύτερο όρισμα, τότε επιστρέφει τα στοιχεία του πίνακα μέχρι το τέλος.

Μια μέθοδος που τροποποιεί τα στοιχεία ενός πίνακα, εισάγοντας νέα στοιχεία και διαγράφοντας στοιχεία που υπάρχουν, είναι η splice(). Θα πρέπει να σημειωθεί ότι η μέθοδος αυτή επιστρέφει τα στοιχεία που διαγράφτηκαν, ωστόσο τροποποιεί τον αρχικό πίνακα.

Η splice() συντάσσεται ως εξής:

```
myArray.splice(θέση, πλήθος-διαγραφή, στοιχεία-για-εισαγωγή);
```

Όπου η θέση ορίζει τον δείκτη του στοιχείου στο οποίο θα γίνει η τροποποίηση, η πλήθος-διαγραφή ορίζει το πλήθος στοιχείων που θα διαγραφούν από τη θέση αυτή και μετά, και τα στοιχεία-για-εισαγωγή είναι τα στοιχεία που θα εισαχθούν στο σημείο αυτό.

Να σημειωθεί ότι, αν παραληφθούν το δεύτερο και το τρίτο όρισμα, τότε θεωρούμε ότι τα στοιχεία για διαγραφή εκτείνονται μέχρι το τέλος του πίνακα, χωρίς στοιχεία για εισαγωγή.

Ένα πρώτο παράδειγμα είναι:

```
const ar = [1,2,3,4,5,6]
let x = ar.splice(3);
console.log(ar);
[1,2,3]
console.log(x);
[4,5,6]
```

Ένα ακόμη παράδειγμα με διαγραφή ορισμένου πλήθους στοιχείων:

```
const ar = [1,2,3,4,5,6]
let x = ar.splice(3,2); //διαγράφονται 2 στοιχεία
console.log(ar);
[1,2,3,6]
console.log(x);
[4,5]
```

Ακολουθεί ένα παράδειγμα με διαγραφή και ταυτόχρονη εισαγωγή στοιχείων:

```
const ar = [1,2,3,4,5,6]
let x = ar.splice(3,2,30,40); //διαγράφονται 2 στοιχεία και εισάγονται 2.
console.log(ar);
[1,2,3,30,40,6]
console.log(x);
[4,5]
```

Άσκηση

Έστω ο παρακάτω κώδικας, ποιο το αποτέλεσμα;

```
constidfruits = ["Μπανάνα", "Αχλάδι", "Μήλο", "Μάνγκο"];
fruits.splice(2, 1, "Λεμόνι", "Πεπόνι");
```

Απάντηση

Θα διαγραφεί ένα φρούτο στη θέση 2 (το "Μήλο") και θα εισαχθούν δύο στην ίδια θέση, άρα τελικά θα έχουμε:

```
["Μπανάνα", "Αχλάδι", "Λεμόνι", "Πεπόνι", "Μάνγκο"]
```

9.2.8 Μέθοδοι αναζήτησης

Οι μέθοδοι indexOf() και lastIndexOf() μας επιτρέπουν να αναζητήσουμε ένα στοιχείο σε πίνακα. Οι μέθοδοι αυτές εφαρμόζονται και στις συμβολοσειρές.

Μας επιστρέφουν τον δείκτη του στοιχείου που έχει τιμή ίση με το όρισμα, η μεν indexOf() αρχίζοντας την αναζήτηση από την αρχή του πίνακα, και η δε lastIndexOf() από το τέλος του πίνακα. Αν το στοιχείο δεν βρεθεί, επιστρέφουν την τιμή -1.

Ας δούμε ένα παράδειγμα:

```
const ar = [5,10,15,20,10,5];
ar.indexOf(15);
> 2
ar.lastIndexOf(10);
> 4
```

Θα πρέπει να προσέξουμε ότι η αναζήτηση εφαρμόζει τον τελεστή σύγκρισης “===” δηλαδή, αναζητάει στοιχεία που είναι ίσα ως προς τον τύπο και την τιμή με το όρισμα της συνάρτησης.

Να σημειωθεί ότι οι μέθοδοι αυτές παίρνουν ως δεύτερο προαιρετικό όρισμα έναν δείκτη από πού να αρχίσει η αναζήτηση.

9.2.9 Μέθοδοι ταξινόμησης

Η μέθοδος sort() ταξινομεί αλφαβητικά τα στοιχεία του πίνακα τροποποιώντας τον ίδιο τον πίνακα στη θέση του.

Ένα πρώτο παράδειγμα:

```
const students = ['Μαρία', 'Κώστας', 'Ανδρέας'];
students.sort();
> [ 'Ανδρέας', 'Κώστας', 'Μαρία' ]
```

Θα πρέπει να προσέξουμε ότι, αν δεν δώσουμε ως όρισμα μια συνάρτηση ταξινόμησης, η μέθοδος αυτή ταξινομεί αλφαβητικά τα στοιχεία, και αν αυτά δεν είναι συμβολοσειρές τα μετατρέπει σε συμβολοσειρές πριν την ταξινόμηση. Για παράδειγμα:

```
const ar = [5, 10, 15];
ar.sort();
ar;
> [ 10, 15, 5 ]
```

Ο λόγος για τον οποίο αυτός ο πίνακας ακέραιων ταξινομείται με αυτό τον τρόπο είναι γιατί τα στοιχεία του μετατρέπονται σε [“5”, “10”, “15”] τα οποία ταξινομούνται ως συμβολοσειρές με τον τρόπο που φαίνεται στο παράδειγμα.

Για να αντιμετωπίσουμε το πρόβλημα αυτό θα πρέπει να περάσουμε ως όρισμα μια συνάρτηση ταξινόμησης η οποία να δέχεται ως όρισμα δύο διαδοχικά στοιχεία και να μας επιστρέφει έναν θετικό αριθμό ή αρνητικό, αντίστοιχα, για να οριστεί η διάταξή τους. Το θέμα αυτό θα το δούμε πιο αναλυτικά όταν συζητήσουμε τις συναρτήσεις σε επόμενο κεφάλαιο, μπορούμε όμως να παραθέσουμε και εδώ τη λύση του προβλήματος ταξινόμησης ακέραιων.

```
const ar = [5, 10, 15];
ar.sort((a, b) => a - b); //αύξουσα σειρά
ar;
> [ 5, 10, 15 ]
```

Ενώ αντίστοιχα για φθίνουσα σειρά ταξινόμησης:

```
const ar = [5, 10, 15];
ar.sort((a, b) => b - a); // φθίνουσα σειρά
ar;
> [ 15, 10, 5 ]
```

Τέλος, θα πρέπει να αναφερθεί ότι η μέθοδος reverse() αντιστρέφει τη σειρά των στοιχείων τροποποιώντας το ίδιο το αρχείο.

9.2.10 Μέθοδοι μετατροπής πίνακα σε συμβολοσειρά

Όταν συζητήσαμε τις συμβολοσειρές, είδαμε τη μέθοδο split (*διαχωριστικό*), η οποία μετατρέπει μια συμβολοσειρά σε πίνακα στοιχείων με βάση ένα *διαχωριστικό*. Αν μάλιστα το διαχωριστικό είναι η κενή συμβολοσειρά, "" τότε μετατρέπει τη συμβολοσειρά σε έναν πίνακα που περιέχει ως στοιχεία τους χαρακτήρες της συμβολοσειράς.

Την αντίστροφη δουλειά κάνει η μέθοδος join() του τύπου Array. Η μέθοδος αυτή μετατρέπει τα στοιχεία σε συμβολοσειρές και στη συνέχεια τις συνενώνει σε μια συμβολοσειρά, θέτοντας ως διαχωριστικό τον χαρακτήρα ",". Όπως και στη split(), μπορούμε και στην join() να ορίσουμε ένα διαφορετικό διαχωριστικό μεταξύ των στοιχείων.

Παράδειγμα:

```
const ar = [5,10,15];
ar.join()
> "5,10,15"
```

Αν επιθυμούμε να συνενώσουμε τα στοιχεία χωρίς διαχωριστικό, τότε θα πρέπει να περάσουμε ως όρισμα στη μέθοδο join() την κενή συμβολοσειρά "".

Συνοψίζοντας, στην ενότητα αυτή είδαμε τον τύπο δεδομένων Array, που είναι ο πρώτος τύπος δεδομένων αναφοράς. Θα ακολουθήσει ο τύπος δεδομένων Object που είναι ο πιο βασικός τύπος δεδομένων αναφοράς. Υπάρχουν επίσης άλλες δομές που μοιάζουν με Array, όπως η λίστα κόμβων NodeList του DOM που επιστρέφει η συνάρτηση document.querySelectorAll(). Πολλές από τις μεθόδους που είδαμε στην ενότητα αυτή έχουν εφαρμογή και σε αυτή την περίπτωση.

Θα πρέπει να σημειωθεί ότι ο τύπος Array συνδέεται άμεσα με τις επαναληπτικές δομές for/in for/of, αλλά και με τις μεθόδους του συναρτησιακού προγραμματισμού που θα δούμε στην επόμενη ενότητα.

9.3 Συναρτήσεις

9.3.1 Εισαγωγή

Συνάρτηση είναι ένα επαναχρησιμοποιούμενο μπλοκ κώδικα, το οποίο ορίζεται μια φορά και μπορεί να κληθεί και να εκτελεστεί πολλές φορές. Η έννοια της συνάρτησης υπάρχει στις περισσότερες γλώσσες προγραμματισμού. Η JavaScript είναι ισχυρά συναρτησιακή γλώσσα, ορίζει τις συναρτήσεις ως αντικείμενα

(αναφέρονται ως *callable objects*). Οι συναρτήσεις που ορίζονται ως ιδιότητες αντικειμένων (objects) ονομάζονται *μέθοδοι* (methods).

Το όνομα μιας συνάρτησης θα πρέπει απαραίτητα να ξεκινάει από έναν αλφαβητικό χαρακτήρα (κεφαλαία ή πεζά) ή τον χαρακτήρα της κάτω παύλας (underscore) ή $. Μετά τη δήλωση συνάρτησης, αυτή μπορεί να κληθεί με χρήση του ονόματός της. Εξ ορισμού, οι τιμές των ορισμάτων εισόδου αποδίδονται στον κώδικα της συνάρτησης με αντιγραφή τους σε τοπικές μεταβλητές (κλήση μέσω τιμής – by value). Αν όμως το όρισμα εισόδου είναι κάποιος πίνακας ή άλλο αντικείμενο, αυτό αποδίδεται στην εσωτερική τοπική μεταβλητή μέσω αναφοράς (by reference). Τέλος, υπάρχει η δυνατότητα ορισμού ανώνυμων συναρτήσεων (αντίστοιχες των συναρτήσεων lambda της Python) ως ορίσματα άλλων συναρτήσεων.

Η σύνταξη του ορισμού της συνάρτησης έχει επεκταθεί στην ES6, με την εισαγωγή της σημειολογίας της συνάρτησης βέλους "=>" όπως θα δούμε στη συνέχεια.

9.3.2 Ορισμός συνάρτησης με χρήση της λέξης function

Υπάρχουν διάφοροι τρόποι ορισμού μιας συνάρτησης.

Ο πρώτος τρόπος που θα δούμε είναι με χρήση της λέξης-κλειδί function.

```
function onomaSynarthshs(orismata) {
  // σώμα συνάρτησης
}
```

Η εντολή return ορίζει τι επιστρέφει μια συνάρτηση. Αν μια συνάρτηση δεν περιέχει εντολή return στο τέλος εκτέλεσής της, τότε επιστρέφει την τιμή undefined.

Ένα παράδειγμα είναι η παρακάτω συνάρτηση που επιστρέφει έναν τυχαίο ακέραιο μεταξύ 1 και number.

```
function random(number) {
   // τυχαίος αριθμός μεταξύ 1 και number
   return Math.floor(Math.random()*number+1);
}

let x = random(5);  //κλήση συνάρτησης
```

Θα πρέπει να σημειωθεί ότι οι συναρτήσεις που δηλώνονται με αυτό τον τρόπο μπορούν να χρησιμοποιηθούν σε οποιοδήποτε τμήμα του κώδικα, ακόμη και πριν τον ορισμό τους.

Παραλλαγή αυτής της δήλωσης είναι η εκχώρηση της συνάρτησης σε μεταβλητή (literal notation).

```
const onomaSynarthshs  = function(orismata) {
  //κώδικας
  return δεδομένα;
}
```

Αυτός ο τρόπος ορισμού συνάρτησης οφείλει την ύπαρξή του στο γεγονός ότι οι συναρτήσεις είναι αντικείμενα.

Θα πρέπει να προσέξουμε ότι σε αυτή την περίπτωση ισχύουν όλα όσα ισχύουν για τον ορισμό μεταβλητών, για παράδειγμα, δεν μπορούν να χρησιμοποιηθούν πριν τον ορισμό τους.

Ξαναγράφουμε τη δήλωση της random(number):

```
const random = function(number) {
   // τυχαίος αριθμός μεταξύ 1 και number
   return Math.floor(Math.random()*number+1);
}
```

Μια πιο σπάνια περίπτωση είναι η συνάρτηση που ορίζεται μέσω εκχώρησής της σε μεταβλητή να έχει και αυτή όνομα, ώστε, για παράδειγμα, να χρησιμοποιηθεί σε περίπτωση αναδρομής.

Ένα παράδειγμα αναδρομικού υπολογισμού παραγοντικού είναι:

```
const f = function fact(x) {
  if (x <= 1) return 1;
  else return x * fact(x - 1);
};
```

```
f(5);
>120
```

9.3.3 Ορισμός συνάρτησης με χρήση βέλους =>

Από την έκδοση ES6 της JavaScript έχει εισαχθεί ένας ακόμη τρόπος ορισμού συναρτήσεων, με χρήση του συμβόλου =>. Οι συναρτήσεις που ορίζονται με αυτό τον τρόπο ονομάζονται **συναρτήσεις βέλους** (arrow functions).

```
const f = (parametroi) => {
  // σώμα συνάρτησης
}
```

Ο τρόπος αυτός ορισμού είναι πιο σύντομος, παραλείπει τη χρήση της λέξης function και, όπως θα δούμε στη συνέχεια, μπορεί σε ορισμένες περιπτώσεις να απλοποιηθεί περαιτέρω.

Όταν η συνάρτηση περιέχει στο σώμα της μόνο μια εντολή και αυτή είναι η εντολή return, τότε μπορεί να παραληφθούν η λέξη return και τα άγκιστρα ως εξής:

```
// πρώτος τρόπος
const mult =  (x,y) => {
  return x * y;
};

// απλοποιημένη μορφή
const mult =  (x,y) => x * y;
```

Θα πρέπει να δοθεί ιδιαίτερη προσοχή στην περίπτωση εκείνη που η συνάρτηση επιστρέφει την τιμή ενός αντικείμενου (οι τιμές των αντικειμένων, όπως θα δούμε στη συνέχεια, ορίζονται ως {key: value}). Στην περίπτωση αυτή είμαστε υποχρεωμένοι να βάλουμε το αντικείμενο σε παρενθέσεις, γιατί αλλιώς υπάρχει σύγχυση στη χρήση των άγκιστρων ως τερματικών χαρακτήρων της συνάρτησης αλλά και των αντικειμένων.

Ένα παράδειγμα, αν μια συνάρτηση λαμβάνει ως ορίσματα το όνομα και την ηλικία και επιστρέφει το αντικείμενο όπως {name:"Κώστας", age:20}, η συνάρτηση θα πρέπει να γραφεί ως εξής:

```
const f = (n, a) => ({name:n, age:a})
```

Αντίθετα, η έκφραση

```
const f = (n, a) => {name:n, age:a}
```

θα δώσει συντακτικό λάθος.

Οι συναρτήσεις βέλη χρησιμοποιούνται κυρίως ως ανώνυμες συναρτήσεις, για παράδειγμα στις περιπτώσεις που περνάμε μια συνάρτηση ως όρισμα μιας άλλης συνάρτησης (callback functions), ή για τις περιπτώσεις που ορίζουμε τη συνάρτηση ως χειριστή ενός συμβάντος.

9.3.4 Εμφώλευση συναρτήσεων

Επιτρέπεται να οριστεί μια συνάρτηση μέσα σε μια άλλη συνάρτηση. Η εμφωλευμένη συνάρτηση μπορεί να κληθεί μόνο μέσα στη συνάρτηση που έχει οριστεί, και έχει πρόσβαση στις μεταβλητές της συνάρτησης αυτής.

Για παράδειγμα, για τον υπολογισμό της υποτείνουσας μπορούμε να χρησιμοποιήσουμε την εξής συνάρτηση, μέσα στην οποία ορίζουμε μια συνάρτηση υπολογισμού του τετραγώνου:

```
function ypotinousa(a, b) {
  const sq = (x) => x*x;
  return Math.sqrt(sq(a) + sq(b));
}

ypotinousa(3,4)
> 5
```

Οι κανόνες εμβέλειας μεταβλητών συναρτήσεων είναι αντικείμενο της επόμενης ενότητας.

9.3.5 Εμβέλεια μεταβλητών

- Μεταβλητές που ορίζονται έξω από όλες τις συναρτήσεις ανήκουν στην περιοχή καθολικής εμβέλειας global scope.
- Μεταβλητές που ορίζονται στο επίπεδο αυτό είναι προσβάσιμες από όλο τον κώδικα.
- Κάθε συνάρτηση ορίζει τοπική εμβέλεια (local scope) και οι μεταβλητές που ορίζονται στο επίπεδο αυτό είναι προσβάσιμες μόνο μέσα στη συνάρτηση.

Έστω το παράδειγμα:

```
let x = 1;

function a() {
  let y = 2;
  b(x);
  b(y);
}

function b(value) {
  console.log( `Τιμή: ${value}`);
}
a();
> 'Τιμή: 1'
> 'Τιμή: 2'
```

Η συνάρτηση b() όταν καλείται μέσα από τη συνάρτηση a() έχει πρόσβαση και στις δύο μεταβλητές x, y, η πρώτη από τις οποίες έχει οριστεί στην περιοχή καθολικής εμβέλειας, ενώ η δεύτερη είναι τοπική μεταβλητή της a().

Ας δούμε ένα παράδειγμα κώδικα που παράγει ReferenceError λόγω εμβέλειας μεταβλητών.

```
function myFunction() {
  let userName = 'Nikos';
  console.log(userName); // ok
}
console.log(userName); // ReferenceError
```

Αν στον κώδικα αυτόν αντικαθιστούσαμε τη δήλωση της τοπικής μεταβλητής με τη λέξη-κλειδί var, και πάλι δεν θα είχαμε κάποια αλλαγή, αφού η μεταβλητή userName παραμένει τοπική μεταβλητή στη συνάρτηση myFunction().

Ένα ακόμη παράδειγμα:

Η μεταβλητή color στο παρακάτω παράδειγμα είναι καθολική μεταβλητή, αφού ορίζεται εκτός των συναρτήσεων. Αντίθετα, η anotherColor είναι τοπική στη συνάρτηση changeColor και είναι προσβάσιμη και στη συνάρτηση swapColors που ορίζεται εντός της changeColor.

```
let color = 'μπλε';
function changeColor() {
  let anotherColor = 'κόκκινο';
  function swapColors() {
    let tempColor = anotherColor;
    anotherColor = color;
    color = tempColor;
    // color, anotherColor, tempColor είναι προσβάσιμες εδώ
  }
  // μόνο color,anotherColor είναι προσβάσιμες
  swapColors();
}
// μόνο η color είναι προσβάσιμη εδώ
changeColor();
console.log(color); // 'κόκκινο'
```

Άσκηση

Έστω ο παρακάτω κώδικας:

```
function myBigFunction() {
  let myValue = 1;
  subFunction1();
  subFunction2();
}
function subFunction1() {
  console.log(myValue);
}
function subFunction2() {
  console.log(myValue);
}
myBigFunction();
```

Ποιο το αποτέλεσμα;

Απάντηση

Ο κώδικας αυτός θα προκαλέσει σφάλμα ReferenceError: myValue is not defined. Αυτό γιατί η μεταβλητή myValue ορίζεται ως τοπική μεταβλητή στο πλαίσιο της συνάρτησης myBigFunction(), άρα δεν είναι προσβάσιμη από τις άλλες δύο συναρτήσεις subFunction1() και subFunction2(), οι οποίες επίσης είναι ορισμένες στην καθολική περιοχή. Αν επιθυμούσαμε να έχουν πρόσβαση στη μεταβλητή αυτή θα έπρεπε να ορίσουμε τις συναρτήσεις μέσα στη myBigFunction() ως εξής:

```
function myBigFunction() {
  let myValue = 1;
  function subFunction1() {
    console.log(myValue);
  }
  function subFunction2() {
    console.log(myValue);
  }
  subFunction1();
  subFunction2();
}

myBigFunction();
```

Στην περίπτωση αυτή το αποτέλεσμα του κώδικα θα ήταν η εκτύπωση της τιμής 1 δύο φορές.

9.3.6 Προαιρετικά ορίσματα συναρτήσεων

Η JavaScript πριν την έκδοση ES6 δεν υποστήριζε άμεσα προαιρετικά ορίσματα συναρτήσεων.

Η κλήση μιας συνάρτηση απαιτεί τον ορισμό τιμών σε όλα τα ορίσματα. Μάλιστα, είναι αξιοσημείωτο ότι δεν προκαλείται εξαίρεση σε περίπτωση που δώσουμε περισσότερες τιμές από ό,τι τα ορίσματα.

```
function f(x,y,z){
    return x+y+z
}

f(1,2)
NaN
f(1,2,3,4,5)
6
```

Πριν την έκδοση ES6, αν επιθυμούσαμε σε κάποια ορίσματα να δώσουμε προκαθορισμένες τιμές και εφόσον ο χρήστης δεν τους έδινε τιμή κατά την κλήση της συνάρτησης, αυτό έπρεπε να γίνει με εσωτερικό έλεγχο που εντοπίζει τη μη χρήση του ορίσματος (που σε αυτή την περίπτωση έχει την τιμή undefined) και του αναθέτει μια προκαθορισμένη τιμή.

Να ένα παράδειγμα αυτής της προσέγγισης:

```
function f(optional) {
  if (typeof optional === 'undefined') optional = 1;
  return optional + 1;
}
f(3); // Επιστρέφεται 4
f(); // Επιστρέφεται 2
```

Επίσης, ένας εναλλακτικός τρόπος να κάνουμε τον έλεγχο τιμής της μεταβλητής είναι να αντικαταστήσουμε τη δεύτερη εντολή με την εξής ιδιωματική έκφραση:

```
  optional = optional || 1;
```

Ο τελεστής || επιστρέφει το πρώτο όρισμα αν είναι αληθές, διαφορετικά, το δεύτερο. Συνεπώς, αν η παράμετρος optional δεν έχει πάρει τιμή, θα πάρει την τιμή 1.

Στην έκδοση ES6 επιτρέπονται πλέον προαιρετικά ορίσματα με προκαθορισμένες τιμές, κάτι αντίστοιχο με τα keyword arguments στη γλώσσα Python.

Το παραπάνω παράδειγμα στη νεότερη έκδοση της γλώσσας θα μπορούσε να γραφτεί απλούστερα ως εξής:

```
function f(optional = 1) {
  return optional + 1;
}
f(3); // Επιστρέφεται 4
f(); // Επιστρέφεται 2
```

Να σημειωθεί επίσης ότι τα προαιρετικά ορίσματα πρέπει να ακολουθούν τα υποχρεωτικά.

9.3.7 Ο τελεστής ... στα ορίσματα συνάρτησης

Ο τελεστής ... (τελεστής ανάπτυξης, spread) είναι ένας τελεστής που χρησιμοποιείται για την ανάπτυξη ενός πίνακα ή μιας ακολουθιακής δομής, όπως μια συμβολοσειρά, στα επιμέρους στοιχεία από τα οποία απαρτίζεται.

Ο τελεστής αυτός είναι χρήσιμος σε διάφορες περιπτώσεις:

Αντίγραφο πίνακα

Στον ορισμό ενός πίνακα μπορούμε να μεταφέρουμε τα στοιχεία ενός πίνακα σε έναν άλλο, δημιουργώντας ένα αντίγραφό του.

Για παράδειγμα, το αποτέλεσμα του παρακάτω κώδικα:

```
const a = [1,2,3];
const b = [ ...a];
```

είναι ότι ο πίνακας b είναι ένας νέος πίνακας, πανομοιότυπο αντίγραφο του a (να σημειωθεί ότι, αν η δεύτερη εντολή αντικατασταθεί από την εντολή const b = a; δεν έχουμε το ίδιο αποτέλεσμα, αφού ο b είναι μια μεταβλητή που αναφέρεται στον ίδιο πίνακα όπως και η μεταβλητή a).

Πέρασμα των στοιχείων πίνακα ως ορίσματα συνάρτησης

Μια δεύτερη χρήση του *τελεστή ανάπτυξης* είναι στα ορίσματα συναρτήσεων, όπου θέτει τα στοιχεία ενός πίνακα ως ξεχωριστά γνωρίσματα όταν κάτι τέτοιο απαιτεί ο ορισμός της συνάρτησης.

Ας δούμε ένα παράδειγμα.

```
function f(x,y,z){
  return x+y+z;
}
const a = [5,6,7];
console.log(f(...a));
>18
```

Στην περίπτωση αυτή «σπάμε» τον πίνακα a στα επιμέρους στοιχεία του ώστε να τα περάσουμε ως ορίσματα στη συνάρτηση f().

Ορισμός συνάρτησης με απροσδιόριστο πλήθος ορισμάτων

Μια τρίτη περίπτωση χρήσης του τελεστή ανάπτυξης είναι κατά τον ορισμό συνάρτησης με μεταβλητό πλήθος ορισμάτων.

Στην περίπτωση αυτή ο τελεστής εφαρμόζεται σε μία μεταβλητή, την οποία μπορούμε στο σώμα της συνάρτησης να χειριστούμε ως πίνακα τιμών.

Ακολουθεί ένα παράδειγμα.

```
function max(...args) {
  let maxValue = args[0];
  for (let n of args) {
    if (n > maxValue) maxValue = n;
  }
  return maxValue;
}

max(10, 50, 60, 5, 8);
> 60
```

Στο παράδειγμα αυτό η μεταβλητή args εκπροσωπεί έναν πίνακα από τιμές μεταβλητού πλήθους.

9.3.8 Στατικές μεταβλητές συνάρτησης

Υπάρχουν περιπτώσεις που θα θέλαμε μια συνάρτηση να «θυμάται» τις προηγούμενες φορές που κλήθηκε. Αυτό μπορεί να έχει ενδιαφέρον σε ορισμένες περιπτώσεις. Σε κάποιες γλώσσες προγραμματισμού αυτό επιτυγχάνεται με τις λεγόμενες στατικές μεταβλητές. Ας υποθέσουμε ότι έχουμε μια συνάρτηση counter() η οποία κάθε φορά που την καλούμε επιστρέφει μια τιμή αυξημένη κατά 1 από την τελευταία φορά που την καλέσαμε. Ένας απλός τρόπος να το πετύχουμε είναι να ορίσουμε μια μεταβλητή του *αντικείμενου* που εκπροσωπεί τη συνάρτηση. Δεν θα πρέπει να ξεχνάμε ότι μια συνάρτηση είναι στην ουσία ένα object. Συνεπώς, μπορεί να έχει ιδιότητες, όπως όλα τα αντικείμενα.

```
function counter() {
 return ++counter.count;
}
counter.count = 0;

counter();  // επιστρέφει 1
counter(); // επιστρέφει 2
```

9.3.9 Οι συναρτήσεις ορίζουν μοναδικό χώρο ονομάτων

Υπάρχουν περιπτώσεις που θα θέλαμε ο κώδικάς μας να μην έχει συγκρούσεις ως προς τα ονόματα μεταβλητών με άλλα τμήματα κώδικα που πιθανόν φορτωθούν μαζί, για παράδειγμα στο πλαίσιο μιας ιστοσελίδας. Για να πετύχουμε αυτή την απομόνωση του χώρου ονομάτων μεταβλητών του κώδικά μας, μπορούμε να ενσωματώσουμε τον κώδικα σε μια συνάρτηση, την οποία να καλέσουμε μόλις το DOM φορτωθεί, και μέσα στη συνάρτηση αυτή να ορίσουμε τις μεταβλητές, τις συναρτήσεις, τα αντικείμενα κ.λπ.

```
function pageScript(){
  // εδώ ορίζουμε μεταβλητές, συναρτήσεις,
  // χειριστές γεγονότων κ.λπ.
}

document.addEventListener("DOMContentLoaded", pageScript);
```

9.3.10 Μέθοδοι επεξεργασίας πινάκων

Η επεξεργασία των στοιχείων ενός πίνακα είναι συχνή στην JavaScript. Είτε με σκοπό τον μετασχηματισμό τους είτε για να τα χρησιμοποιήσει σε μια ακολουθία πράξεων που τα εμπλέκουν, π.χ. να βρεθεί το άθροισμά τους.

Ο πιο κλασικός τρόπος επεξεργασίας των στοιχείων ενός πίνακα ή γενικότερα μιας ακολουθιακής δομής είναι η χρήση δομών επανάληψης, όπως η for. Όμως, η JavaScript, αναδεικνύοντας τον συναρτησιακό χαρακτήρα της, διαθέτει μεθόδους του αντικείμενου Array που επιτρέπουν πιο αποδοτικά και συνοπτικά να κάνουμε αυτή την επεξεργασία. Οι μέθοδοι αυτές έχουν το χαρακτηριστικό ότι παίρνουν ως όρισμα συναρτήσεις οι οποίες εφαρμόζονται στα στοιχεία του πίνακα.

Παραδείγματα αυτών των μεθόδων είναι:

- myArray.**forEach**(myFunction) Εφαρμόζει τη συνάρτηση σε κάθε στοιχείο του πίνακα.
- myArray.**map**(myFunction) Δημιουργεί νέο πίνακα με εφαρμογή της συνάρτησης σε κάθε στοιχείο του.
- myArray.**filter**(myFunction) Δημιουργεί νέο πίνακα με τα στοιχεία που ικανοποιούν τη συνάρτηση-φίλτρο.
- myArray.**reduce**(acumFun, αρχικήΤιμή) Παράγει μια τιμή μετά από διαδοχική εφαρμογή της συνάρτησης acumFun στα στοιχεία του πίνακα.

Ας δούμε στη συνέχεια τυπικά παραδείγματα χρήσης των μεθόδων αυτών.

Η μέθοδος forEach

Η μέθοδος αυτή εφαρμόζει το όρισμά της σε ένα προς ένα τα στοιχεία του πίνακα, δεν επιστρέφει κάποια τιμή.

```
const myAr = ["Χίος", "Μυτιλήνη", "Σάμος"]
myAr.forEach((el, i) => console.log(i, 'Η ' + el));
> 0 'Η Χίος'
> 1 'Η Μυτιλήνη'
> 2 'Η Σάμος'
```

Η συνάρτηση που περνάμε ως όρισμα στη forEach() είναι η συνάρτηση που εφαρμόζεται στο αντίστοιχο στοιχείο. Αν θέσουμε δεύτερο όρισμα, αυτό είναι ο δείκτης του στοιχείου, ενώ ένα τρίτο όρισμα αντιπροσωπεύει τον ίδιο τον πίνακα.

Να σημειωθεί ότι η μέθοδος αυτή δεν επηρεάζει τον ίδιο τον πίνακα στον οποίο εφαρμόζεται, ενώ δεν επιστρέφει τιμή.

Αν επιθυμούμε να τροποποιήσουμε τον ίδιο τον αρχικό πίνακα, μπορούμε να εκμεταλλευτούμε το γεγονός ότι το τρίτο όρισμα της συνάρτησης που περνάμε στη forEach() είναι ο ίδιος ο πίνακας.

Έτσι, για παράδειγμα, για να μετατρέψουμε τα ονόματα των νησιών σε κεφαλαία:

```
const myAr = ['Χίος', 'Μυτιλήνη', 'Σάμος'];
myAr.forEach((el, i, a) => {
  a[i] = a[i].toUpperCase();
});
console.log(myAr);
> [ 'ΧΙΟΣ', 'ΜΥΤΙΛΉΝΗ', 'ΣΆΜΟΣ' ]
```

Η μέθοδος map

Η μέθοδος αυτή επιστρέφει έναν νέο πίνακα, που προκύπτει από την εφαρμογή της συνάρτησης που θέτουμε ως όρισμά της σε κάθε στοιχείο του αρχικού πίνακα. Η μέθοδος map() είναι συνεπώς μια συνάρτηση μετασχηματισμού ενός πίνακα σε έναν νέο πίνακα.

```
const myAr = ['Χίος', 'Μυτιλήνη', 'Σάμος'];
const newAr = myAr.map((el, i) => i + ': η νήσος ' + el);
console.log(newAr);
> [ '0: η νήσος Χίος', '1: η νήσος Μυτιλήνη', '2: η νήσος Σάμος' ]
```

Όπως και στην προηγούμενη περίπτωση, η μέθοδος map() δεν επηρεάζει τον αρχικό πίνακα, όμως επιστρέφει έναν νέο πίνακα που προκύπτει από τον μετασχηματισμό.

Η συνάρτηση που περνάμε ως όρισμα στην map δέχεται τρία ορίσματα, το πρώτο είναι το εκάστοτε στοιχείο του πίνακα, το δεύτερο ο δείκτης του στοιχείου και το τρίτο ο ίδιος ο πίνακας.

Η μέθοδος filter

Η μέθοδος αυτή επιστρέφει επίσης έναν νέο πίνακα που προκύπτει από την εφαρμογή της συνάρτησης που θέτουμε ως όρισμά της, η οποία δρα ως φίλτρο σε κάθε στοιχείο του αρχικού πίνακα. Τα στοιχεία εκείνα για τα οποία η συνάρτηση αυτή επιστρέφει την τιμή true περιέχονται στον νέο πίνακα, ενώ εκείνα που επιστρέφουν false όχι. Η μέθοδος filter() είναι, συνεπώς, μια συνάρτηση μετασχηματισμού ενός πίνακα σε έναν νέο πίνακα.

```
const myAr = ['Χίος', 'Μυτιλήνη', 'Σάμος'];
const newAr = myAr.filter((el) => el.length > 5);
console.log(newAr);
> [ 'Μυτιλήνη' ]
```

Όπως και στην προηγούμενη περίπτωση, η μέθοδος filter() δεν επηρεάζει τον αρχικό πίνακα, όμως επιστρέφει έναν νέο πίνακα που προκύπτει από τον μετασχηματισμό.

Η συνάρτηση που περνάμε ως όρισμα στη filter() δέχεται ως όρισμα το εκάστοτε στοιχείο του πίνακα και πρέπει να επιστρέφει τιμή true/false.

Η μέθοδος reduce

Η μέθοδος αυτή διαφέρει από τις προηγούμενες, αφού επιστρέφει μία τιμή και όχι έναν πίνακα. Η τιμή αυτή προκύπτει από την εφαρμογή της συνάρτησης (πρώτο όρισμά της) διαδοχικά σε κάθε στοιχείο του αρχικού πίνακα.

Η συνάρτηση πρώτο όρισμα της reduce() παίρνει τα εξής ορίσματα:

- Ως πρώτο όρισμα έναν συσσωρευτή που διαδοχικά συσσωρεύει το αποτέλεσμα των προηγούμενων πράξεων,
- ως δεύτερο όρισμα το εκάστοτε στοιχείο του πίνακα,
- προαιρετικά μπορεί να πάρει ακόμη τον δείκτη στο εκάστοτε στοιχείο και τον ίδιο τον πίνακα.

Η μέθοδος αυτή παίρνει ως δεύτερο όρισμα την αρχική τιμή του συσσωρευτή.

Ακολουθεί ένα παράδειγμα.

```
const myAr = ['Χίος', 'Μυτιλήνη', 'Σάμος'];
const result = myAr.reduce((islands, el) => {
  return (islands += ' ' + el);
}, '');
console.log(result);
> "Χίος Μυτιλήνη Σάμος"
```

Όπως και στην προηγούμενη περίπτωση, η μέθοδος filter() δεν επηρεάζει τον αρχικό πίνακα, όμως επιστρέφει την τελική τιμή του συσσωρευτή.

Παραδείγματα

Έστω πίνακας που περιέχει ένα σύνολο αριθμητικών τιμών. Ως παράδειγμα, ας υποθέσουμε ότι έχουμε τον παρακάτω πίνακα:

```
const a = [5, 10, 18, 32, 20, 44];
```

Παράδειγμα 1: Υπολογισμός αθροίσματος στοιχείων πίνακα

Για τον υπολογισμό του αθροίσματος θα εφαρμόσουμε τη reduce() στον πίνακα, χρησιμοποιώντας έναν αθροιστή που αρχικά έχει την τιμή μηδέν και στον οποίο διαδοχικά αθροίζουμε τα στοιχεία.

```
const result = a.reduce((s, el) => s + el, 0);
console.log(result);
> 129
```

Παράδειγμα 2: Εύρεση ελάχιστης τιμής

Και στην περίπτωση αυτή θα εφαρμόσουμε τη reduce() στον πίνακα, χρησιμοποιώντας έναν συσσωρευτή που αρχικά έχει την τιμή Infinity. Στη συνέχεια, ελέγχουμε για κάθε στοιχείο αν είναι μικρότερο από τον συσσωρευτή, αν ναι, το στοιχείο παίρνει τη θέση του συσσωρευτή.

```
const result = a.reduce((s, el) => (el < s ? el : s), Infinity);
console.log(result);
> 5
```

Με αντίστοιχο τρόπο βρίσκουμε και τη μέγιστη τιμή του πίνακα.

Παράδειγμα 3: Υπολογισμός τυπικής απόκλισης συνόλου τιμών

Έστω πίνακας με ένα σύνολο αριθμητικών τιμών. Ζητείται να οριστεί συνάρτηση που υπολογίζει την τυπική απόκλιση.

Υπενθυμίζεται ότι ο τύπος της τυπικής απόκλισης είναι

$$\mathrm{SD} = \sqrt{\frac{\sum |x-\mu|^2}{N}}$$

όπου x είναι μια τιμή, μ η μέση τιμή και N το πλήθος των τιμών.

Καταρχήν, η *μέση τιμή* $\boldsymbol{\mu}$ των τιμών του a μπορεί να βρεθεί με επαναληπτική διαδικασία:

```
let mean = 0
for (let i of a) mean += i;
mean = mean/a.length
```

Ένας εναλλακτικός τρόπος υπολογισμού είναι με χρήση της μεθόδου reduce():

```
mean = a.reduce((s, i) => s + i, 0) / a.length;
```

Για να υπολογίσουμε τον όρο $\Sigma(|x-\mu|)^2$, μπορούμε επίσης να εφαρμόσουμε συναρτησιακή προσέγγιση με χρήση της map() για παραγωγή μιας ακολουθίας τετραγώνων:

```
console.log(a.map((x) => Math.pow(x - mean, 2)));
> [ 272.25, 132.25, 12.25, 110.25, 2.25, 506.25 ]
```

Στη συνέχεια, δε, με χρήση της reduce() να υπολογίσουμε το άθροισμα των τετραγώνων και την τετραγωνική ρίζα του αθροίσματος διά του πλήθους.

```
const s1 = a.map((x) => Math.pow(x - mean, 2));
const sd = Math.sqrt(s1.reduce( (a,b)=> a+b, 0)/a.length)
> 13.13709759929237
```

Συνοψίζοντας, μπορούμε να δημιουργήσουμε μια συνάρτηση ως εξής:

```
function standardDeviation(a) {
  const mean = a.reduce((s, i) => s + i, 0) / a.length;
  const s1 = a.map((x) => Math.pow(x - mean, 2));
  return Math.sqrt(s1.reduce((a, b) => a + b, 0) / a.length);
}
```

Παράδειγμα 4. Τυχαία αναδιάταξη στοιχείων πίνακα

Έστω ότι ζητείται να αναδιατάξουμε με τυχαίο τρόπο τα στοιχεία ενός πίνακα. Αναζητήσετε διαφορετικούς τρόπους και ελέγξετε την τυχαιότητα της λύσης.

Υποθέτουμε για λόγους απλότητας ότι έχουμε τον πίνακα:

```
const a = [1,2,3]
```

Ως πρώτη προσέγγιση στο πρόβλημα της αναδιάταξης των τιμών με τυχαίο τρόπο, υποθέτουμε τη χρήση της συνάρτησης:

```
function shuffle1(array) {
  array.sort(() => Math.random() - 0.5);
}
```

Θεωρώντας ότι η Math.random() επιστρέφει τιμές ομοιόμορφα κατανεμημένες στο διάστημα [0..1], η Math.random() - 0.5 θα έχει κατά τυχαίο τρόπο θετικό ή αρνητικό πρόσημο, άρα θα ταξινομεί δύο τυχαία στοιχεία σε αύξουσα ή φθίνουσα σειρά.

Εφαρμόζουμε τη συνάρτηση αυτή στον πίνακα a διαδοχικά για μεγάλο πλήθος επαναλήψεων και στη συνέχεια ελέγχουμε αν οι διαφορετικές διατάξεις έχουν παρόμοιες συχνότητες.

```
const count = {};
for (let _ = 0; _ < 10000; _++) {
  const a = [1, 2, 3];
  const result = shuffle1(a).join('');
  count[result] = result in count ? count[result] + 1 : 1;
}
console.log(count);
console.log(`SD=${standardDeviation(Object.values(count))}`);
```

Το αποτέλεσμα που προκύπτει είναι:

```
{'123': 3809, '132': 642, '213': 1183
 '231': 633, '312': 634, '321': 3099
}
'SD=1294.856062356825'
```

Είναι φανερό ότι κάποιες τιμές έχουν πολύ υψηλότερη συχνότητα εμφάνισης, άρα αποκαλύπτεται μια αδυναμία του συγκεκριμένου αλγόριθμου.

Μια δεύτερη προσπάθεια γίνεται με τον αλγόριθμο Fisher-Yates:

```
function shuffle2(a) {
  //  αλγόριθμος Fisher-Yates
  for (let i = a.length - 1; i > 0; i--) {
    let j = Math.floor(Math.random() * (i + 1)); // random index from 0 to
i
    [a[i], a[j]] = [a[j], a[i]];
  }
  return a;
}
```

Ο επαναληπτικός αυτός αλγόριθμος σε κάθε βήμα για i = από ν-1 μέχρι 1 κάνει αντιμετάθεση ενός τυχαίου στοιχείου που βρίσκεται σε τυχαία θέση από 0 μέχρι i με το στοιχείο i.

Σε αυτή την περίπτωση, ο ίδιος έλεγχος δίνει τα εξής αποτελέσματα:

```
{ '123': 1669, '132': 1643, '213': 1644,
  '231': 1623, '312': 1701, '321': 1720
}
'SD=34.179265969622904'
```

Παρατηρείται μια σαφής βελτίωση έναντι της προηγούμενης περίπτωσης.

9.4 Αντικείμενα και κλάσεις

Τα *αντικείμενα* είναι ο πιο βασικός τύπος δεδομένων της JavaScript.

Ένα αντικείμενο στην JavaScript είναι ένας σύνθετος τύπος δεδομένων που περιέχει ιδιότητες που έχουν

ως τιμή είτε πρωτογενή δεδομένα είτε άλλα αντικείμενα. Οι ιδιότητες ενός αντικείμενου δεν είναι ταξινομημένες και έχουν τη μορφή **ιδιότητα: τιμή**. Βεβαίως, η τιμή κάποιων ιδιοτήτων μπορεί να είναι συναρτήσεις (που σε αυτή την περίπτωση λέγονται ***μέθοδοι***). Αρχικά, τα αντικείμενα της JavaScript μοιάζουν με τα λεξικά της Python ή άλλες αντίστοιχες δομές, όπως πίνακες κατακερματισμού σε άλλες γλώσσες προγραμματισμού.

```
const car = {
  make: "volvo",
  speed: 140,
  engine: {
    size:1800,
    fuel: "diesel",
    pistons: ["piston1", "piston2"]},
  drive: function() {
    return `οδηγώ ${this.make}...`}}
}
```

Στο παράδειγμα αυτό το αντικείμενο car έχει 4 ιδιότητες, από αυτές η μία έχει ως τιμή ένα άλλο αντικείμενο και η άλλη μια μέθοδο.

Στις ιδιότητες ενός αντικείμενου μπορούμε να αναφερθούμε με δύο τρόπους. Με σημειογραφία τελείας:

```
console.log(car.make);
>'volvo'
```

Εναλλακτικά, μπορούμε να αναφερθούμε στην ιδιότητα με αγκύλες ως εξής:

```
console.log(car["make"]);
>'volvo'
```

Όμως, τα αντικείμενα της JavaScript δεν είναι απλά πίνακες αντιστοίχισης κλειδιών-τιμών. Κάθε αντικείμενο συνοδεύεται από μια πρόσθετη ιδιότητα που έχει την τιμή **prototype**, που είναι ένα αντικείμενο του οποίου κληρονομεί τις ιδιότητες. Αυτή είναι μια ιδιαιτερότητα της JavaScript που δεν συναντάται σε άλλες γλώσσες προγραμματισμού. Ο μηχανισμός αυτός λέγεται ***κληρονομικότητα μέσω πρωτοτύπου***.

Για τα ονόματα των ιδιοτήτων των αντικειμένων ισχύουν όσα ισχύουν για τις μεταβλητές της JavaScript.

Κάθε ιδιότητα ενός αντικείμενου, εκτός από *όνομα* και *τιμή*, έχει ακόμη τα εξής γνωρίσματα: *writable* (αν μπορεί να αλλάξει η τιμή), *enumerable* (αν αφορά ιδιότητα που θα εμφανίζεται σε βρόχο for), *configurable* (αν μπορεί να διαγραφεί). Να σημειωθεί ότι τα αντικείμενα της γλώσσας (Array, Number, Function κ.λπ.) δεν επιτρέπουν τη διαγραφή των ιδιοτήτων ή την τροποποίησή τους, κάτι που όμως επιτρέπεται για τα αντικείμενα των χρηστών.

Έτσι, στα αντικείμενα των χρηστών των οποίων οι ιδιότητες είναι enumerable μπορούμε να εφαρμόσουμε έναν βρόχο **for/in** ως εξής:

```
for (property in car) {
  console.log(property, car[property]);
}
> 'make' 'volvo'
> 'speed' 140
> 'engine' { size: 1800, fuel: 'diesel', pistons: [ 'piston1', 'piston2' ]
}
> 'drive' ƒ drive()'make'
```

Θα πρέπει να σημειώσουμε επίσης ότι τις ιδιότητες ενός αντικείμενου (μόνο τις enumerable) μπορούμε να τις ανακτήσουμε μέσω της μεθόδου Object.keys()

```
Objec.keys(car);
> [ 'make', 'speed', 'engine', 'drive' ]
```

Επίσης, θα πρέπει να αναφερθεί ότι και ένας πίνακας έχει ως ιδιότητες τους δείκτες 0,1,2…, άρα:

```
const ar = [10, 20, 30];
Object.keys(ar);
> [ '0', '1', '2' ]
```

9.4.1 Μετατροπή αντικειμένων σε JSON

Η μετατροπή ενός αντικείμενου JavaScript σε μια συμβολοσειρά από την οποία εν συνεχεία μπορεί να ανακτηθεί λέγεται **σειριοποίηση** (**serialization**). Η διαδικασία αυτή είναι χρήσιμη γιατί η μετατροπή του αντικείμενου σε ακολουθία χαρακτήρων μάς επιτρέπει να το μεταβιβάσουμε σε έναν παραλήπτη ή να το αποθηκεύσουμε. Με τη σύνταξη σε μορφή JSON (JavaScript Object Notation) μπορούμε να μετατρέψουμε τα αντικείμενα της JavaScript κατά τη σειριοποίησή τους. Η JSON είναι ένα πρότυπο ανταλλαγής δεδομένων με ευρεία χρήση, πέραν της JavaScript.

Η μετατροπή ενός αντικείμενου σε μορφή JSON γίνεται με τη μέθοδο JSON.stringify(obj), ενώ η αντίθετη μετατροπή γίνεται με τη μέθοδο JSON.parse(st).

Η μετατροπή αντικειμένων σε συμβολοσειρές δεν καλύπτει όμως όλες τις περιπτώσεις αντικειμένων της JavaScript, ώστε να μπορέσουμε να ανακτήσουμε την αρχική μορφή του αντικείμενου.

Τα αντικείμενα, πίνακες, συμβολοσειρές, αριθμοί, οι λογικές τιμές true/false και null μπορούν να μετατραπούν σε JSON και να ανακτηθούν.

Όμως, τιμές NaN, Infinity και -Infinity μετατρέπονται όλα σε null και δεν μπορεί έτσι να ανακτηθεί η αρχική τους τιμή.

Αντικείμενα τύπου Date μετατρέπονται σε συμβολοσειρές για ημερομηνία κατά το πρότυπο ISO (σύμφωνα με την Date.toJSON()), όμως η JSON.parse() δεν επαναφέρει την ημερομηνία από τη συμβολοσειρά αυτή. Τέλος, συναρτήσεις, κανονικές εκφράσεις και αντικείμενα σφάλματος (Error objects), καθώς και τιμές undefined δεν μπορούν να μετατραπούν σε JSON, ούτε βεβαίως να ανακτηθούν από JSON.

9.4.2 Δημιουργία κλάσεων και αντικειμένων

Ο πιο απλός τρόπος δημιουργίας αντικειμένων είναι με απευθείας ορισμό τους μέσα σε άγκιστρα που περιλαμβάνουν ακολουθία από ζευγάρια ιδιότητα: τιμή. Με τον τρόπο αυτό ορίστηκε το αντικείμενο car στην προηγούμενη ενότητα.

Για παράδειγμα, το πιο απλό αντικείμενο ορίζεται ως εξής:

```
const ob = {};
```

Το αντικείμενο αυτό φαίνεται να μην έχει ιδιότητες, όμως, όπως ήδη αναφέρθηκε, περιλαμβάνει την έξτρα ιδιότητα του πρωτοτύπου. Έτσι, αν στην κονσόλα ζητήσουμε να δούμε το περιεχόμενο αυτού του αντικείμενου, παρουσιάζεται η εξής εικόνα:

```
> ob
{}
__proto__: Object
```

Ένας δεύτερος τρόπος για να δημιουργήσουμε ένα αντικείμενο είναι με χρήση του τελεστή new που ακολουθείται από μια συνάρτηση. Ο τελεστής αυτός δημιουργεί ένα καινούργιο αντικείμενο. Η συνάρτηση που ακολουθεί λέγεται **δημιουργός** (**constructor**) και χρησιμεύει για να αρχικοποιήσει το καινούργιο αντικείμενο.

Υπάρχουν δημιουργοί για τα εγγενή αντικείμενα της γλώσσας, όπως οι δημιουργοί Object(), Array(), Date() κ.λπ. Όπως θα δούμε στη συνέχεια, μπορούμε και εμείς να ορίσουμε συναρτήσεις δημιουργούς αντικειμένων.

Για να ορίσουμε μια δική μας κλάση αντικειμένων, που έχουν παρόμοια δομή, πρέπει να ορίσουμε μια συνάρτηση η οποία θα δημιουργεί τα αντικείμενα αυτά.

Έστω ότι επιθυμούμε να δημιουργήσουμε μια κλάση αυτοκινήτων που αφορά αντικείμενα της μορφής:

```
const myCar = {
  make: "VW",
  speed: 140,
  drive: function(){
```

```
        return `οδηγώ ${this.make}...`}
}
```

Επιθυμούμε να ορίσουμε μια συνάρτηση δημιουργό Car() η οποία θα παράγει αυτοκίνητα, όπου να περνάμε τις τιμές των ιδιοτήτων ενός αντικείμενου (στιγμιότυπου της κλάσης):

```
const myCar = new Car("VW", 140);
const myOtherCar = new Car("Porshe", 350);
```

Η συνάρτηση αυτή ορίζεται ως εξής:

```
function Car(make, speed){
  this.make = make;
  this.speed = speed;
}
```

Παρατηρούμε τη χρήση της λέξης this για αναφορά στο εκάστοτε στιγμιότυπο της κλάσης. Ας χρησιμοποιήσουμε τώρα τη συνάρτηση αυτή για να κατασκευάσουμε ένα αντικείμενο myCar και ας εξερευνήσουμε τη δομή του νέου αντικείμενου:

```
const myCar = new Car("VW", 200);
> myCar
Car {make: "VW", speed: 200}
  make: "VW"
  speed: 200
  __proto__:
    constructor: ƒ Car(make, speed)
    __proto__: Object
```

Για να ελέγξουμε, μάλιστα, αν ένα αντικείμενο ανήκει σε ορισμένη κλάση, χρησιμοποιούμε τον τελεστή instanceof:

```
myCar instanceof Car;
> true
```

Αυτό είναι ένα πρώτο παράδειγμα χρήσης της συνάρτησης δημιουργού. Η συνάρτηση Car(), όπως παρατηρούμε, δεν έχει την ίδια συμπεριφορά με τις συναρτήσεις που έχουμε δει ως τώρα. Αν και δεν περιλαμβάνει εντολή return, μας επιστρέφει ένα νέο αντικείμενο όταν τη χρησιμοποιούμε με τον τελεστή new. Είναι ο τελεστής new που δίνει σε αυτή τη συνάρτηση ειδική χρήση, κάνει τη συνάρτηση Car() δημιουργό συνάρτηση αντικειμένων.

Επίσης, παρατηρούμε ότι το αντικείμενο που δημιουργήσαμε, εκτός από τις δύο ιδιότητες που ορίσαμε στον δημιουργό του, έχει μια ακόμη ιδιότητα, την __proto__ που δηλώνει το *πρωτότυπο* του αντικείμενου, κληρονομεί από το αντικείμενο Object και έχει ως δημιουργό του τη συνάρτηση Car(), δηλαδή τη δημιουργό συνάρτηση της κλάσης μας. Το «πρωτότυπο» αυτό δημιουργήθηκε λοιπόν από τη συνάρτηση δημιουργό μας. Θα πρέπει να σημειωθεί βεβαίως ότι η __proto__ δεν είναι πραγματικά ιδιότητα αλλά αναφορά στο πρωτότυπο του δημιουργού, δεν μπορούμε να την τροποποιήσουμε και το αντικείμενο, αν ερωτηθεί σχετικά, δεν την αναγνωρίζει:

```
myCar.hasOwnProperty("make");
> true
myCar.hasOwnProperty("__proto__");
> false
```

9.4.3 Ορισμός μεθόδων

Ένα ακόμη θέμα που πρέπει να δούμε είναι πώς ορίζουμε τις μεθόδους μιας κλάσης αντικειμένων. Έστω ότι θέλουμε τα αντικείμενά μας να έχουν τη μέθοδο drive().

Αυτή μπορούμε να την ορίσουμε στο πρωτότυπο της κλάσης ώστε να την κληρονομήσουν όλα τα στιγμιότυπά της:

```
Car.prototype.drive: function() {
    return `οδηγώ ${this.make}...`;
  }

myCar.drive()
> 'οδηγώ VW...'
```

Αν στην κονσόλα ζητήσουμε τη δομή του αντικείμενου myCar, αυτή τώρα είναι:

```
> myCar
Car {make: "VW", speed: 200}
  make: "VW"
  speed: 200
  __proto__:
    drive: ƒ ()
    __proto__: Object
      constructor: ƒ Car(make, speed)
```

Όπως βλέπουμε, έχει προστεθεί η ιδιότητα drive στο πρωτότυπο, η τιμή της οποίας είναι η μέθοδος που ορίσαμε.

Στο μέλλον μπορούμε να προσθέτουμε ιδιότητες ή μεθόδους στο πρωτότυπο της κλάσης Car, και αυτές θα κληρονομούνται από όλα τα στιγμιότυπα.

Ας δούμε έναν εναλλακτικό τρόπο ορισμού της μεθόδου drive() μέσα στη δημιουργό συνάρτηση της κλάσης Car:

```
function Car(make, speed) {
  this.make = make;
  this.speed = speed;
  this.drive = function () {
    return `οδηγώ ${this.make}...`;
  };
}
```

Σε αυτή την περίπτωση το αντικείμενο έχει την εξής δομή:

```
myCar
> Car {make: "VW", speed: 200, drive: ƒ}
  drive: ƒ ()
  make: "VW"
  speed: 200
  __proto__:
    constructor: ƒ Car(make, speed)
    __proto__: Object
```

Υπάρχει μια μικρή διαφορά μεταξύ των δύο τρόπων ορισμού μιας μεθόδου που είδαμε ως τώρα. Με τον δεύτερο τρόπο ορισμού της μεθόδου drive(), η μέθοδος είναι η τιμή της ιδιότητας drive του αντικείμενου. Πρακτικά, αυτό σημαίνει ότι κάθε φορά που δημιουργούμε ένα νέο αντικείμενο με τον δημιουργό Car() δημιουργούμε ένα νέο αντίγραφο της συνάρτησης drive. Άσκοπη δαπάνη μνήμης. Ενώ στην πρώτη προσέγγιση είχαμε μόνο ένα αντίγραφο της μεθόδου στο αντικείμενο Car.prototype από το οποίο κληρονομούν τη μέθοδο όλα τα στιγμιότυπα τύπου Car. Εκεί όμως έχουμε καθυστέρηση κάθε φορά που καλούμε τη μέθοδο, την αναζητάμε πρώτα στο στιγμιότυπο και στη συνέχεια στο πρωτότυπο. Ό,τι χάνουμε σε χώρο το κερδίζουμε σε χρόνο.

9.4.4 Ορισμός κλάσεων με τη λέξη-κλειδί class

Μια από τις αλλαγές που έφερε η ES6 είναι η εισαγωγή μιας νέας σύνταξης για ορισμό κλάσεων που περιλαμβάνει τη λέξη-κλειδί class. Η σύνταξη αυτή δεν αλλάζει την ουσία του μηχανισμού δημιουργίας αντικειμένων που αναφέρθηκε, η οποία στηρίζεται στα πρωτότυπα, όμως κάνει πιο απλή τη σύνταξη και την JavaScript να μοιάζει πιο πολύ με άλλες γλώσσες προγραμματισμού.

```
class Car{
  constructor(make, speed){
    this.make = make;
    this.speed = speed;
  }
  drive(){
    return `οδηγώ ${this.make}...`;
  }
}

const myCar = new Car("VW", 140);

myCar.drive();
> "οδηγώ VW..."
```

Παρατηρούμε ότι η σύνταξη της class έχει κάποιες ιδιαιτερότητες: Περιλαμβάνει χρήση της λέξης class ακολουθούμενης από το όνομα της κλάσης και τον ορισμό της μέσα σε άγκιστρα. Μέσα στο σώμα περιλαμβάνουμε τη δημιουργό συνάρτηση με τη λέξη constructor() και στη συνέχει ορισμό των μεθόδων χωρίς τη χρήση της λέξης κλειδί function. Δεν βάζουμε κόμμα ανάμεσα στις μεθόδους ή στον constructor και τις μεθόδους.

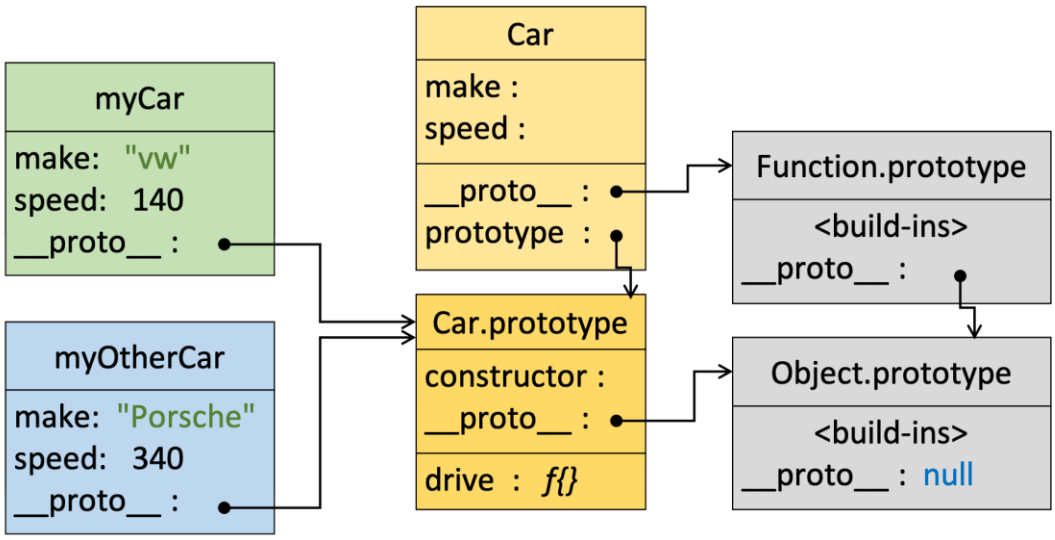
Να σημειωθεί εδώ ότι αυτός ο ορισμός παράγει αντικείμενα με τη μέθοδο στο πρωτότυπο (πρώτη μέθοδος της προηγούμενης ενότητας).

Έτσι, αν δημιουργήσουμε ένα αντικείμενο με χρήση αυτής της κλάσης, το περιεχόμενό του είναι:

```
const myCar = new Car("VW", 140);
> myCar
Car {make: "VW", speed: 140}
  make: "VW"
  speed: 140
  __proto__:
    constructor: class Car
    drive: ƒ drive()
    __proto__: Object
```

Παρατηρούμε ότι η μέθοδος drive() ανήκει στο αντικείμενο της ιδιότητας __proto__, δηλαδή στο πρωτότυπο αντικείμενο της κλάσης.

Αν θέλαμε να αποδώσουμε σχηματικά τη σχέση μεταξύ του δημιουργού και του πρωτοτύπου αυτή φαίνεται στην **Εικόνα *9.1*** για την περίπτωση της κλάσης Car.



Εικόνα 9.1 *Δημιουργός, πρωτότυπο για την κλάση Car.*

Ο δημιουργός της κλάσης, δηλαδή η συνάρτηση Car, έχει την ιδιότητα prototype η οποία έχει τιμή το πρωτότυπο της συγκεκριμένης κλάσης. Να σημειωθεί ότι όλες οι συναρτήσεις έχουν ιδιότητα prototype. Το αντικείμενο αυτό έχει ως δημιουργό του την κλάση Car. Κάθε στιγμιότυπο που δημιουργούμε με την κλάση Car κληρονομεί το πρωτότυπο αυτό, οι ιδιότητες του οποίου μπορεί να είναι νέες μέθοδοι, που δημιουργούνται ως εξής:

```
Car.prototype.newMethod = function() { ... }
```

Το ίδιο αποτέλεσμα μπορούμε να έχουμε με έμμεση αναφορά στο πρωτότυπο αυτό, για παράδειγμα αν δημιουργήσουμε τη μέθοδο ως

```
myCar.__proto__.newMethod = function() { ... }
```

κάνουμε έμμεση αναφορά στο πρωτότυπο Car.prototype.

Τέλος, να αναφερθεί ότι η συνάρτηση δημιουργός Car(), όπως και όλες οι συναρτήσεις, κληρονομεί το πρωτότυπο Function.prototype, ενώ το αντικείμενο Car.prototype κληρονομεί το πρωτότυπο Object.prototype, το οποίο, ως πρωτότυπο του αρχέγονου αντικείμενου, δεν κληρονομεί από κανένα αντικείμενο και για αυτό η ιδιότητά του __proto__ έχει την τιμή null.

9.4.5 Κληρονομικότητα κλάσεων

Αν επιθυμούμε να ορίσουμε μια υποκλάση μιας κλάσης, αυτό γίνεται με χρήση του τελεστή extends.

Ας δούμε ένα παράδειγμα. Έστω μια κλάση Person, η οποία ορίζεται ως ακολούθως:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}
```

Ας υποθέσουμε ότι θέλουμε να δημιουργήσουμε μια υποκλάση της Person που την εξειδικεύει για την περίπτωση δασκάλων. Έστω λοιπόν η κλάση Teacher που ορίζεται ως εξής:

```
class Teacher extends Person {
  constructor(name, age, school) {
    super(name, age);
    this.school = school;
  }
}
```

Παρατηρούμε ότι η κλάση Teacher ορίζεται ως επέκταση της κλάσης Person. Ο δημιουργός αντικειμένων της νέας αυτής κλάσης κάνει αναφορά στην αρχική κλάση για τις ιδιότητες name, age που είναι ιδιότητες όλων των αντικειμένων τύπου Person. Αυτό γίνεται με κλήση της μεθόδου super() που εκπροσωπεί τον δημιουργό της υπερκλάσης. Αν θέλαμε να αναφερθούμε σε επιμέρους μεθόδους της υπερκλάσης, θα μπορούσαμε να το κάνουμε με σημειογραφία τελείας: super.μέθοδος().

Για να δημιουργήσουμε ένα αντικείμενο της κλάσης Teacher, αυτό γίνεται ως εξής:

```
const t = new Teacher('Κώστας', 40, '1ο Λύκειο');

>t
Teacher {name: "Κώστας", age: 40, school: "1ο Λύκειο"}
  age: 40
  name: "Κώστας"
  school: "1ο Λύκειο"
  __proto__: Person
    constructor: class Teacher
    __proto__: Object
```

Στο παραπάνω απόσπασμα οι ιδιότητες name, age έχουν δημιουργηθεί από τον δημιουργό super() της υπερκλάσης, ενώ η ιδιότητα school προστέθηκε από τον δημιουργό της Teacher.

9.5 Ερωτήσεις αυτοαξιολόγησης

1. Ποιο το αποτέλεσμα;

```
let c = new Array(2);
c[3]=5;
typeof c[2]
```

   1. ‘number’
   2. ‘string’
   3. ‘undefined’
   4. 0

2. Ποιο το αποτέλεσμα;

```
let a = new Array("10");
console.log(a.length)
```

   1. 2
   2. 1
   3. 10

3. Ποιο το αποτέλεσμα;

```
let a = new Array(10);
a[1] = 8;
a[2] = 18;
console.log(a.length)
```

   1. 10
   2. 2
   3. 3
   4. undefined

4. Ποιο το αποτέλεσμα;

```
let a = [10,20,30];
console.log(a.push(40));
```

   Απάντηση: _____

5. Ποιο το αποτέλεσμα;

```
let a = [8,15,40,66];
console.log(a.pop());
```

   Απάντηση: _____

6. Ποιο το αποτέλεσμα;

```
let a = [6, 66, 36];
a.shift();
a.unshift(56);
console.log(a);
```

   1. [6, 66, 36, 56]
   2. [56, 6, 66, 36]
   3. [56, 66, 36]
   4. [66, 36, 56]

7. Ποιο το αποτέλεσμα;

```
console.log("1,2,3".split(",").join("-"))
```

   Απάντηση: *_____*

8. Ποιο το αποτέλεσμα;

```
let a = [5,10,15,20,25];
a.splice(1,4,30);
console.log(a);
```

1. [5,10,15,20,25,30]
2. [5,4,30,10,15,20,25]
3. [5,4,30]
4. [5,30]

9. Ποιο το αποτέλεσμα του παρακάτω κώδικα;

```
let f =  function(x){
  return x**2
}
console.log(f(5))
```

Απάντηση: _____

10. Ποιο το αποτέλεσμα;

```
let f = function(x){
  x += 10;
}
let x = 5;
f(x);
console.log(x);
```

Απάντηση: _____

11. Ποιο το αποτέλεσμα;

```
let f =  function(x){
  x += 10
}
console.log(f(5));
```

1. 15
2. 5
3. 10
4. undefined

12. Ποια η χρήση της συνάρτησης:

```
function r(n){
  return Math.floor(Math.random()*n+1);
}
```

1. Επιστρέφει ένα τυχαίο αριθμό μεταξύ 0 και 1.
2. Επιστρέφει ένα τυχαίο ακέραιο αριθμό μεταξύ 0 και n.
3. Επιστρέφει ένα τυχαίο ακέραιο αριθμό μεταξύ 0 και n+1.
4. Επιστρέφει ένα τυχαίο ακέραιο αριθμό μεταξύ 1 και n+1.
5. Επιστρέφει ένα τυχαίο ακέραιο αριθμό μεταξύ 1 και n.

13. Συμπληρώστε τον παρακάτω κώδικα ώστε το αποτέλεσμα να είναι 10.

```
let f = (x)=>x+x**3;
console.log(f(....));
```

Απάντηση: _____

14. Έστω ο παρακάτω κώδικας:

```
<button onclick="f(........)">button</button>
<script>
  function f(element){
```

```
    element.style.fontSize='3em';}
</script>
```

Να συμπληρωθεί ο κώδικας ώστε όταν επιλέγεται το πλήκτρο button το μέγεθος χαρακτήρων του να τριπλασιάζεται.
Απάντηση: _____

15. Έστω ο παρακάτω κώδικας:

```
<html><body>
  <button onclick="f()"">button</button>
  <script>
      "use strict"
      function f(){
          console.log(this.innerHeight);}
    </script>
</body></html>
```

Ποιο το αποτέλεσμα;

1. Όταν πατηθεί το πλήκτρο θα τυπωθεί το ύψος του τρέχοντος παράθυρου στην κονσόλα.
2. Όταν πατηθεί το πλήκτρο θα τυπωθεί το ύψος του στοιχείου button στην κονσόλα.
3. Όταν πατηθεί το πλήκτρο θα πάρουμε σφάλμα: το this είναι undefined.
4. Όταν πατηθεί το πλήκτρο θα πάρουμε σφάλμα: το στοιχείο button δεν έχει ιδιότητα innerHeight.

16. Ποιο το αποτέλεσμα του παρακάτω κώδικα;

```
function f() {
  let v = 1;
  f1(); }
function f1() {
  console.log(v); }
f()
```

1. Τυπώνει την τιμή 1 στην κονσόλα.
2. Θα πάρουμε ReferenceError αφού η v δεν έχει οριστεί.

17. Ποιο το αποτέλεσμα του παρακάτω κώδικα;

```
let v = 1;
function f() {
  f1(); }
function f1() {
  console.log(v); }
f()
```

1. Τυπώνει την τιμή 1 στην κονσόλα.
2. Θα πάρουμε ReferenceError αφού η v δεν έχει οριστεί.

18. Ποιο το αποτέλεσμα του παρακάτω κώδικα;

```
let v = 1;
function f() {
  let v =2
  f1(); }
function f1() {
  console.log(v); }
f()
```

1. Τυπώνει την τιμή 1 στην κονσόλα.
2. Θα πάρουμε ReferenceError αφού η v δεν έχει οριστεί.
3. Τυπώνει την τιμή 2 στην κονσόλα.

19. Ποιο το αποτέλεσμα του παρακάτω κώδικα:

```
let i = 3
function f(){
  for(let i = 0; i<10; i++) { }
  console.log(i)}
f()
```

    1. Τυπώνει την τιμή 9 στην κονσόλα.
    2. Τυπώνει την τιμή 10 στην κονσόλα.
    3. Τυπώνει την τιμή 3 στην κονσόλα.
    4. Θα πάρουμε ReferenceError για τη μεταβλητή i

20. Ποιο το αποτέλεσμα;

```
function f( x=1, y=2) {
return x+y;
}
f(3);
```

    Απάντηση: _____

21. Ποιο το αποτέλεσμα;

```
function f( x, y=5) {
return x*y;
}
f(3);
```

    1. 15
    2. 8
    3. NaN

22. Ποιο το αποτέλεσμα;

```
function f( x, y=5) {
return x*y;
}
f(3,2);
```

    1. 15
    2. 10
    3. 6
    4. NaN

23. Ποιο το αποτέλεσμα;

```
function f( x, y=5) {
return x+y;
}
f();
```

    1. 5
    2. 10
    3. 0
    4. NaN

24. Να συμπληρώσετε τον παρακάτω κώδικα που υπολογίζει το τετράγωνο του αριθμού που δίνει ο χρήστης.

```
<button onclick="f()"> button </button>
<script>
    let f = ()=>{
        let x = prompt("x=")
        if (.........) alert('error')
        else alert (`the square of ${x} is ${x*x}`)
```

```
    }
</script>
```

Να σημειώσετε όλα όσα ταιριάζουν:

1. isNaN(x)
2. !parseInt(x)
3. !parseFloat(x)
4. parseInt(x)
5. parseFloat(x)
6. !x

25. Έστω ο παρακάτω κώδικας:

```
let x = prompt("x=");
let y = prompt("y=");
if (!parseFloat(x/y))alert('error')
else alert (`x/y = ${x/y}`);
```

Ποια η διαφορά αν αντικατασταθεί η συνάρτηση !parseFloat με τη συνάρτηση isNaN;

1. Καμιά διαφορά.
2. Με τη συνάρτηση isNaN θα ελέγχεται αν ο χρήστης έδωσε μη αριθμητικά δεδομένα.
3. Με τη συνάρτηση isNaN θα παίρνουμε error αν ο χρήστης έδωσε y=0
4. Με τη συνάρτηση isNaN θα παίρνουμε πάντα error.

26. Ποιο το αποτέλεσμα;

```
let x = "10"
console.log(parseInt(x, 2))
```

Απάντηση: _____

27. Ποιο το αποτέλεσμα;

```
f = (x,y) =>{
  if (isFinite(x/y))console.log (`x/y = ${x/y}`)
  else console.log('error');}
f(10,0);
```

1. x/y = 10
2. x/y = 0
3. error
4. x/y = Infinite

28. Να συμπληρώσετε τον παρακάτω κώδικα ώστε μετά την εκτέλεσή του ο πίνακας a2 να έχει την τιμή [10,100,1000]

```
let a = [1,2,3]
let a2 = a.map((x) => *_____*
```

Απάντηση: _____

29. Ποιο το αποτέλεσμα;

```
let a = [1,2,3,4,5,6]
let b = a.filter((x) => x>2)
console.log(b.length)
```

Απάντηση: _____

30. Ποιο το αποτέλεσμα;

```
let a = [1,2,3]
let result=0
let b = a.reduce((result, x) => result += x*x)
console.log(b)
```

Απάντηση: _____

9.6 Βιβλιογραφία και Αναφορές

Και στο κεφάλαιο αυτό ισχύει η βιβλιογραφία των προηγούμενων δύο κεφαλαίων. Το πρότυπο EcmaScript (ECMA-262) συντηρείται από τον οργανισμό ecma.
Στο διαδίκτυο υπάρχουν πολλές πηγές για εκμάθηση της JavaScript καθώς και για αναφορά στα στοιχεία της γλώσσας. Η MDN είναι μια καλή πηγή για εισαγωγικά και προχωρημένα μαθήματα, όπως και για την HTML και CSS. Επίσης, η w3schools περιέχει μαθήματα, ενώ μια πλήρη σειρά μαθημάτων για την JavaScript με παραδείγματα περιλαμβάνει η JavaScript.info, ξεκινώντας από τη γλώσσα και προχωρώντας στη διεπαφή της με τον φυλλομετρητή.

Η JavaScript περιλαμβάνεται σε βιβλία που ήδη αναφέρθηκαν που αφορούν τις τεχνολογίες του προγραμματισμού στον ιστό όπως η HTML και CSS. Αυτή την προσέγγιση στην ελληνική βιβλιογραφία ακολουθεί το βιβλίο των Δουληγέρη κ.ά. (2021), ενώ έχουν μεταφραστεί κάποια συγγράμματα και διατίθενται από τον Εύδοξο, όπως το βιβλίο των Kyrnin και Morrison (2021) και αυτό των Lemay, Coburn και Kyrnin (2016).

Μια άλλη προσέγγιση είναι αυτή εγχειριδίων που ξεκινούν με τη σύνταξη της γλώσσας JavaScript και περιλαμβάνουν τη διεπαφή με το DOM σε μεταγενέστερο κεφάλαιο. Στις πηγές αυτές συχνά περιλαμβάνονται και κεφάλαια που αφορούν τη λειτουργία της γλώσσας στο περιβάλλον node.js. Αυτή την προσέγγιση ακολουθεί το βιβλίο του Λιακέα (2021). Από τη διεθνή βιβλιογραφία παρόμοια προσέγγιση ακολουθεί το βιβλίο του Flanagan (2020), ενώ υπάρχουν και πολλά άλλα, όπως του Frisbie (2020) κ.λπ.

Επιπλέον, οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Για αυτό το κεφάλαιο το σχετικό μάθημα είναι διαλέξεις του μαθήματος «Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, Javascript» καθώς και του μαθήματος «Προχωρημένα θέματα ανάπτυξης ιστοσελίδων».

Α. Ξενόγλωσσες

Flanagan, D. (2020). *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language* (7th ed.). O'Reilly Media, Inc.

Frisbie, M. (2020). *Professional JavaScript for Web Developers*. Wiley.

McFedries, P. (2019). *Web Design Playground - HTML & CSS The Interactive Way*. Manning.

Β. Ελληνόγλωσσες

Αβούρης, Ν. (2018). Εισαγωγή στην ανάπτυξη ιστοσελίδων με HTML5, CSS3, JavaScript. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr

Δουληγέρης, Χ., Μαυροπόδη, Ρ., Κοπανάκη, Ε., & Καραλής, Α. (2021). *Τεχνολογίες και Προγραμματισμός στον Παγκόσμιο Ιστό* (2η έκδ.). Εκδόσεις Νέων Τεχνολογιών. Κωδικός βιβλίου στον Εύδοξο: 102125023.

Kyrnin, J., & Meloni, J. (2021). *Sams Teach Yourself HTML5, CSS and Javascript* (3rd ed.). Εκδόσεις Γκιούρδας.

Lemay, L., Coburn, R., & Kyrnin, J. (2016). *Sams Teach Yourself HTML, CSS & JavaScript* (7th ed). Εκδόσεις Γκιούρδας. Κωδικός Βιβλίου στον Εύδοξο: 59357307.

Λιακέας, Γ. (2021). *Η γλώσσα JavaScript* (3η έκδ.). Κλειδάριθμος. Κωδικός βιβλίου στον Εύδοξο: 102070465.

Κεφάλαιο 10. Ασύγχρονη JavaScript – διαχείριση συμβάντων

Σύνοψη

Στο κεφάλαιο αυτό θα παρουσιάσουμε τους μηχανισμούς που διαθέτει η JavaScript για ασύγχρονη εκτέλεση κώδικα και στη συνέχεια θα εστιάσουμε στον μηχανισμό διαχείρισης συμβάντων (events).

Η ασύγχρονη εκτέλεση κώδικα είναι απαίτηση που συναντάμε συχνά στον φυλλομετρητή (διαχείριση συμβάντων και ασύγχρονη πρόσβαση σε πόρους του διαδικτύου) αλλά και στο περιβάλλον του εξυπηρετητή, της Node.js, όπως θα δούμε σε επόμενα κεφάλαια.

- *Στην **ακολουθιακή (σύγχρονη)** εκτέλεση ενός προγράμματος η μια εντολή εκτελείται μετά την άλλη.*
- *Στην **ασύγχρονη εκτέλεση** ένα τμήμα του προγράμματος, που πρέπει να περιμένει, τοποθετείται σε μια άλλη ουρά εκτέλεσης, χωρίς να μπλοκάρει τη ροή εκτέλεσης.*

Η JavaScript έχει δύο μηχανισμούς ασύγχρονης εκτέλεσης:

- *Την **κλήση συνάρτησης επιστροφής** (callback function),*
- *Τον **μηχανισμό Promise**, όπως θα δούμε στη συνέχεια. Ο μηχανισμός αυτός έχει επιπλέον υλοποιηθεί ως μηχανισμός **async/await**, όπως θα δούμε στο επόμενο κεφάλαιο.*

Στο κεφάλαιο αυτό θα ολοκληρώσουμε την επισκόπηση της JavaScript. Θα δούμε ότι οι λειτουργίες της γλώσσας αποτελούν υπόβαθρο για τα επόμενα κεφάλαια, στα οποία θα ασχοληθούμε με το πώς μπορούμε να εξυπηρετήσουμε αιτήματα στην πλευρά του εξυπηρετητή με την JavaScript.

Προαπαιτούμενη γνώση

Είναι απαραίτητη η εξοικείωση με τα κεφάλαια 7, 8 και 9.

10.1 Ασύγχρονη εκτέλεση κώδικα

Η JavaScript είναι μονο-νηματική γλώσσα εκ κατασκευής. Δηλαδή, στην τυπική λειτουργία της εκτελεί τις εντολές τη μια μετά την άλλη ακολουθιακά. Η λειτουργία αυτή λέγεται **σύγχρονη λειτουργία**.

Όμως, η JavaScript έχει συχνά ανάγκη για διαχείριση ασύγχρονων λειτουργιών, όπως το να εκτελεστεί ένα τμήμα του κώδικα μετά παρέλευση κάποιου χρόνου είτε μετά από ένα συμβάν το οποίο θα προκληθεί από κάποιον έξω προς το περιβάλλον της γλώσσας, π.χ. από τον χρήστη, από το δίκτυο ή από άλλες διεργασίες του λειτουργικού συστήματος.

Δεν θα πρέπει να ξεχνάμε πως η τυπική λειτουργία ενός κώδικα JavaScript στον φυλλομετρητή, όπως ήδη περιγράψαμε, είναι ότι, αφού περάσει αρχικά από τη φάση φορτώματος του κώδικα και της αρχικής εκτέλεσης, εισέρχεται και παραμένει σε κατάσταση αναμονής συμβάντων (**ασύγχρονη λειτουργία**).

Παραδείγματα ασύγχρονης εκτέλεσης κώδικα που θα συζητήσουμε στη συνέχεια ή που έχουμε ήδη δει είναι:

- Με κλήση των μεθόδων του αντικείμενου window **setTimeout(f, t)**, η οποία επιτρέπει την εκτέλεση μιας συνάρτησης f μετά από παρέλευση ορισμένου χρόνου t, ή της **setInterval(f, t)** η οποία επιτρέπει την εκτέλεση της συνάρτησης f επαναληπτικά κάθε t ms.
- Με τον ορισμό **χειριστών συμβάντων**, π.χ. button.addEventListener(event, handler), όπως έχουμε ήδη δει σε προηγούμενα παραδείγματα.
- Με τη χρήση της διεπαφής **fetch()** που χρησιμεύει για ανάκτηση δεδομένων από εξυπηρετητή, με χρήση του μηχανισμού ασύγχρονης λειτουργίας *Promise*.
- Με την κλήση του **requestAnimationFrame** για επαναληπτική εκτέλεση κώδικα ώστε να επιτύχουμε κίνηση γραφικών στοιχείων (animations).

10.1.1 Συναρτήσεις setTimeout() και setInterval()

Κλήση συνάρτησης επιστροφής με καθυστέρηση

Μπορούμε να καλέσουμε μια συνάρτηση επιστροφής μετά από κάποια καθυστέρηση με κλήση της μεθόδου setTimeout() του global object **window**.

```
setTimeout(συνάρτηση επιστροφής, delayInMsec);
```

Για παράδειγμα, στον παρακάτω κώδικα η συνάρτηση console.log() που περνάμε ως όρισμα στην setTimeout θα εκτελεστεί μετά παρέλευση 2000 msec, με αποτέλεσμα να εμφανιστούν πρώτα τα μηνύματα: «Πριν», «Μετά», και μετά με καθυστέρηση 2΄΄ τυπώνεται το μήνυμα «Με καθυστέρηση 2sec».

```
console.log('Πριν');
setTimeout( () => {
    console.log('Με καθυστέρηση 2sec');
}, 2000)
console.log('Μετά');
```

Στο παράδειγμα η συνάρτηση επιστροφής ορίστηκε ως ανώνυμη συνάρτηση, ως πρώτο όρισμα της setTimeout(), ενώ το δεύτερο όρισμα ορίζει την καθυστέρηση σε msec.

Εναλλακτικά, μπορούμε να περάσουμε μια επώνυμη συνάρτηση ως όρισμα της setTimeout. Στην περίπτωση αυτή τα ορίσματα της συνάρτησης, αν υπάρχουν, θα ακολουθούν το όρισμα *delay*. Για παράδειγμα, εναλλακτικά ο παραπάνω κώδικας μπορεί να γραφτεί ως:

```
console.log('Πριν');
setTimeout( console.log, 2000, 'Με καθυστέρηση 2sec');
console.log('Μετά');
```

Άσκηση

Ποιο το αποτέλεσμα του παρακάτω κώδικα;

```
console.log('αρχή');
setTimeout( () => {console.log('timeout1')}, 2000)
setTimeout( () => {console.log('timeout2')}, 1000)
console.log('τέλος');
```

Απάντηση

```
αρχή
τέλος
timeout2
timeout1
```

Επαναληπτική κλήση συνάρτησης με κάποιο διάλειμμα

Η δεύτερη μέθοδος του global object η οποία επιτρέπει την κλήση συνάρτησης με καθυστέρηση είναι η setInterval().

Η μέθοδος αυτή παίρνει δύο ορίσματα, το πρώτο είναι η συνάρτηση επιστροφής η οποία καλείται επαναληπτικά και το δεύτερο είναι το διάστημα σε ms, ανάμεσα σε δύο διαδοχικές κλήσεις. Η συνάρτηση αυτή είναι χρήσιμη για επαναληπτικές διαδικασίες, και για τον λόγο αυτό χρησιμοποιείται σε κίνηση αντικειμένων (animations).

Η setInterval() επιστρέφει μια μοναδική ταυτότητα του interval και αυτή η ταυτότητα μπορεί να χρησιμοποιηθεί για τη διαγραφή του με κλήση της clearInterval().

```
const id = setInterval(func, interval);
clearInterval(id);
```

Άσκηση

Ποιο το αποτέλεσμα του παρακάτω κώδικα;

```
let counter = 0;
const id = setInterval(() => {
    console.log(++counter, "Γεια ");
}, 50);
```

```
setTimeout(clearInterval, 200, id);
```

Απάντηση

```
1 'Γεια '
2 'Γεια '
3 'Γεια '
4 'Γεια '
```

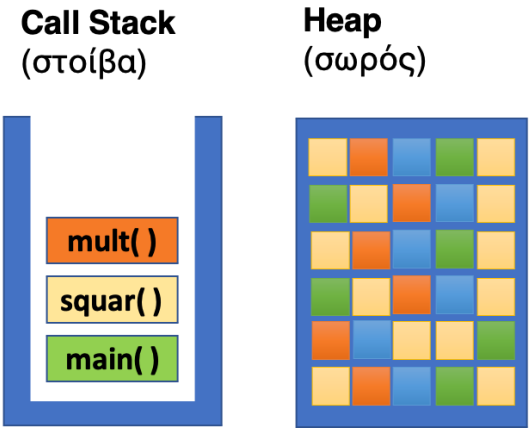
Πριν προχωρήσουμε σε πιο λεπτομερή περιγραφή των μηχανισμών ασύγχρονης λειτουργίας που διαθέτει η γλώσσα, θα πρέπει να περιγράψουμε τον μηχανισμό που ονομάζεται βρόχος ελέγχου συμβάντων (event loop).

10.1.2 Ο βρόχος ελέγχου συμβάντων

Ο **βρόχος ελέγχου συμβάντων** (event loop) είναι ο μηχανισμός της JavaScript ο οποίος επιτρέπει την επιλογή εργασιών από την **ουρά κλήσεων συναρτήσεων επιστροφής (callback queue)** μόνο όταν η **στοίβα (call stack)** είναι κενή (δεν υπάρχουν εργασίες για εκτέλεση).

Η σύγχρονη εκτέλεση κώδικα στηρίζεται σε δύο δομές της μνήμης του υπολογιστή μας (όπως και σε πολλές άλλες γλώσσες προγραμματισμού): Στον **σωρό (heap)** και στη **στοίβα κλήσεων (call stack)**.

Ο **σωρός** είναι ο χώρος της μνήμης όπου αποθηκεύονται τα δεδομένα που χειρίζεται το πρόγραμμά μας. Εκεί υπάρχουν ο κοινός χώρος διευθύνσεων (global object), τα αντικείμενα και οι άλλοι τύποι δεδομένων στα οποία αναφέρεται το πρόγραμμά μας. Η **στοίβα κλήσεων** είναι μια δομή στην οποία τοποθετούνται η μια μετά την άλλη οι συναρτήσεις με τη σειρά που καλούνται. Όταν το πρόγραμμα καλεί μια νέα συνάρτηση, ένα νέο πλαίσιο κώδικα (frame) τοποθετείται στην κορυφή της στοίβας κλήσεων. Στο κάθε πλαίσιο υπάρχουν οι τοπικές μεταβλητές της συνάρτησης. Η στοίβα είναι δομή LIFO (last-in first-out). Η διαδοχική κλήση συναρτήσεων (π.χ. όταν μια συνάρτηση καλεί μια άλλη συνάρτηση) συσσωρεύει διαδοχικά πλαίσια, όπως φαίνεται στην **Εικόνα *10.1***:



Εικόνα 10.1 *Δομές μνήμης κατά την εκτέλεση ενός προγράμματος JavaScript: στοίβα κλήσεων και σωρός. Η main() έχει καλέσει τη squar και αυτή με τη σειρά της τη mult(), η οποία εκτελείται τώρα. Όταν τελειώσει η εκτέλεσή της, το πλαίσιο της mult() θα αφαιρεθεί από τη στοίβα κλήσεων και ο έλεγχος θα περάσει στη squar().*

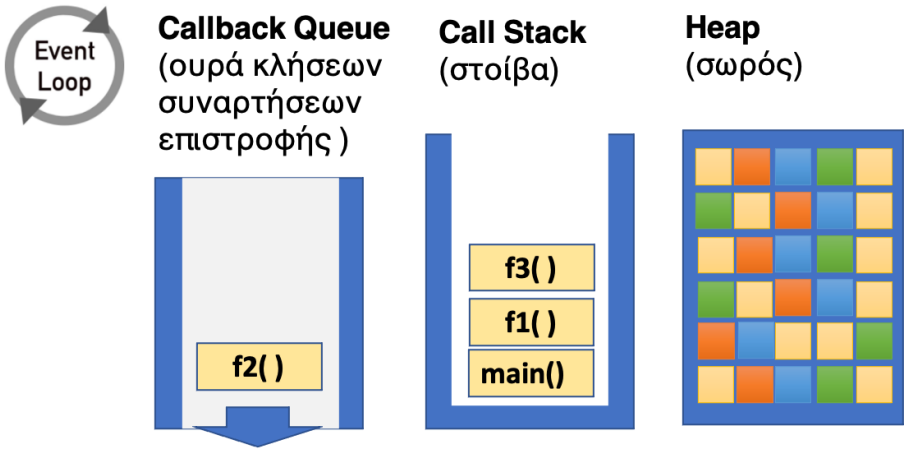
Ας υποθέσουμε ότι εκτελείται ένα πρόγραμμα χωρίς ασύγχρονες λειτουργίες όπως το παρακάτω:

```
function mult(a,b) {
    return a*b
};

function squar(n) {
    return mult(n,n)
};
```

```
console.log(squar(5));
```

Στην ***Εικόνα 10.2*** φαίνεται ότι στη στοίβα κλήσεων έχει φορτωθεί αρχικά το πλαίσιο main(), το οποίο κάλεσε τη συνάρτηση squar() και η οποία με τη σειρά της κάλεσε τη συνάρτηση mult() που εκτελείται κατά τη συγκεκριμένη στιγμή. Όταν η mult() ολοκληρώσει την εκτέλεσή της, θα ελευθερώσει τη στοίβα κλήσεων και θα περάσει τον έλεγχο στη συνάρτηση squar() μέχρι να αδειάσει η στοίβα κλήσεων.

Στη συνέχεια, θα εξετάσουμε πώς διαφοροποιείται αυτός ο μηχανισμός όταν υπάρχουν ασύγχρονα τμήματα κώδικα.



Εικόνα 10.2 *Δομές μνήμης κατά την εκτέλεση ενός προγράμματος JavaScipt με εκτέλεση ασύγχρονων τμημάτων κώδικα: χρήση ουράς κλήσεων συναρτήσεων επιστροφής.*

Ας παρακολουθήσουμε την εκτέλεση του παρακάτω κώδικα βήμα προς βήμα.

```
function f2() {console.log("f2") };

function f3() {console.log("f3") };

function f1() {
    console.log("f1");
    setTimeout(f2, 0);
    f3();
};

f1();
```

Στην περίπτωση αυτή ενεργοποιείται ο βρόχος ελέγχου συμβάντων της JavaScript, αφού στον μηχανισμό έχει προστεθεί ένα ακόμη στοιχείο που είναι η **ουρά κλήσεων συναρτήσεων επιστροφής**. Πρόκειται για μια ουρά FIFO (first-in first-out) στην οποία τοποθετούνται τμήματα του κώδικα τα οποία έχουν κληθεί να εκτελεστούν με ασύγχρονο τρόπο ως συνέπεια κάποιου συμβάντος. Στην ***Εικόνα 10.2*** φαίνεται ότι η συνάρτηση f2() τη στιγμή αυτή είναι σε αναμονή στην ουρά κλήσεων συναρτήσεων επιστροφής. Το ερώτημα είναι πότε θα εκτελεστεί η συνάρτηση αυτή σε σχέση με τα τμήματα του κώδικα που βρίσκονται στη στοίβα.

Ας παρακολουθήσουμε την εκτέλεση του κώδικα. Όταν φορτωθεί το πρόγραμμα (πλαίσιο main()), η πρώτη εντολή είναι η κλήση της συνάρτησης f1() και στη συνέχεια καλείται μέσα από τη συνάρτηση αυτή η setTimeout(f2,0), η οποία άμεσα τοποθετεί τη συνάρτηση f2 στην ουρά κλήσεων συναρτήσεων επιστροφής (callback queue), αφού ο χρόνος αναμονής της είναι 0 ms. Όμως, ο μηχανισμός ασύγχρονης εκτέλεσης της JavaScript δεν επιτρέπει την εκτέλεση κώδικα που βρίσκεται στην ουρά αυτή ενόσω η στοίβα κλήσεων είναι γεμάτη. Στη συνέχεια καλείται η f3() και τοποθετείται στη στοίβα κλήσεων. Τέλος, μετά την ολοκλήρωση εκτέλεσής της, η στοίβα επιτέλους αδειάζει. Τότε μόνο η f(2) μεταφέρεται από την ουρά των κλήσεων συναρτήσεων επιστροφής στη στοίβα και εκτελείται. Συνεπώς, η σειρά εμφάνισης των αποτελεσμάτων του κώδικα αυτού είναι f1, f3, f2.

Στη συνέχεια θα δούμε μια άλλη περίπτωση ορισμού συνάρτησης περιστροφής: τον ορισμό χειριστή συμβάντων.

10.2 Ορισμός χειριστή συμβάντων

Όπως έχει ήδη αναφερθεί, η JavaScript στο περιβάλλον του φυλλομετρητή ακολουθεί το μοντέλο του **προγραμματισμού συμβάντων**, όπως και άλλες γλώσσες προγραμματισμού γραφικών διεπαφών χρήστη.

Ο κώδικας JavaScript αφού φορτωθεί περιμένει. Το περιβάλλον του φυλλομετρητή γεννάει συμβάντα, ως αποτέλεσμα, για παράδειγμα, ενεργειών του χρήστη. Το πρόγραμμά μας έχει καταχωρίσει «χειριστές», συναρτήσεις δηλαδή που περιμένουν αυτά τα συμβάντα. Οι χειριστές καταχωρίζονται σε έναν πίνακα **συμβάντων**. Όταν προκύψει το συμβάν, η JavaScript ανατρέχει στον πίνακα συμβάντων και ανασύρει τον χειριστή που θα πρέπει να ενεργοποιηθεί για το αντίστοιχο συμβάν. Θέτει τον χειριστή στην ουρά κλήσεων συναρτήσεων επιστροφής και ο χειριστής ενεργοποιείται όταν αδειάσει η στοίβα κλήσεων. Ο μηχανισμός ορισμού συμβάντων είναι αρκετά σύνθετος (στηρίζεται στο αντικείμενο **Event** και τις ιδιότητες και μεθόδους του) και θα συζητηθεί πιο εκτενώς στη συνέχεια. Εδώ κάνουμε επισκόπηση των διαφόρων κατηγοριών συμβάντων και διαφόρων τρόπων ορισμού χειριστών τους.

Έχει ενδιαφέρον να παρατηρήσουμε ότι αποτέλεσμα του παραπάνω μηχανισμού (βρόχος χειρισμού συμβάντων) είναι ότι ένα πρόγραμμα JavaScript ποτέ δεν σταματάει την εκτέλεσή του λόγω εξαίρεσης ή σφάλματος. Αν προκύψει ένα σφάλμα σε έναν χειριστή, γεννιέται ένα αντικείμενο **Error** το οποίο παράγει διαγνωστικά μηνύματα, όμως ο χειριστής συμβάντων συνεχίζει να λειτουργεί και τα επόμενα συμβάντα θα προκαλέσουν ασύγχρονη κλήση άλλων χειριστών.

10.2.1 Κατηγορίες συμβάντων

Τα πιο σημαντικά συμβάντα προκύπτουν από ενέργειες του χρήστη, χρήση του ποντικιού ή άλλης δεικτικής συσκευής ή χειρισμοί σε οθόνη αφής, πληκτρολογήσεις κ.λπ. Όμως, υπάρχουν πολλές κατηγορίες συμβάντων, ενώ οι διάφορες εκδόσεις του DOM level 2.0, 3.0 κ.λπ. έχουν οδηγήσει σε διαφοροποιήσεις του μοντέλου συμβάντων μεταξύ των φυλλομετρητών.

Μια πρώτη κατηγοριοποίηση των συμβάντων διακρίνει:

1. **Συμβάντα εξαρτώμενα από συσκευή**. Αυτά τα συμβάντα εξαρτώνται από το είδος της συσκευής, δηλαδή αν πρόκειται για ποντίκι, πληκτρολόγιο ή οθόνη αφής. Τέτοια συμβάντα είναι: mousedown, mousemove, mouseup, touchstart, touchmove, touchend, keydown, keyup κ.λπ.
2. **Συμβάντα ανεξάρτητα από συσκευή**. Το πιο κλασικό παράδειγμα είναι το συμβάν click το οποίο δηλώνει ότι ένας υπερσύνδεσμος ή ένα πλήκτρο έχουν πατηθεί, ενέργεια που μπορεί να γίνει με ποντίκι, πληκτρολόγιο ή το δάχτυλο. Άλλο παράδειγμα είναι το συμβάν input, το οποίο είναι ισοδύναμο του keydown, όμως, εκτός από είσοδο από το πληκτρολόγιο σε ένα στοιχείο εισαγωγής κειμένου, υποστηρίζει και επικόλληση κειμένου. Επίσης, τα συμβάντα pointerdown, pointermove αντιστοιχούν στα αντίστοιχα συμβάντα με ποντίκι ή αφή.
3. **Συμβάντα διεπαφής**. Τα συμβάντα αυτά περιγράφουν την κατάσταση των στοιχείων της διεπαφής. Παραδείγματα είναι τα συμβάντα focus, change (αλλαγή περιεχομένου ενός στοιχείου μιας φόρμας), submit όταν επιλέγεται το πλήκτρο υποβολής μιας φόρμας.
4. **Συμβάντα αλλαγής κατάστασης**. Τα συμβάντα αυτά δεν προκύπτουν κατευθείαν από ενέργειες του χρήστη, αλλά σχετίζονται με την κατάσταση του φυλλομετρητή ή προκύπτουν από το δίκτυο. Παράδειγμα τέτοιων συμβάντων είναι το συμβάν load του αντικείμενου Window, καθώς και το συμβάν DOMContentLoaded του αντικείμενου Document, τα οποία είδαμε στην ενότητα περιγραφής του φορτώματος της ιστοσελίδας στον φυλλομετρητή (7.6). Άλλο παράδειγμα είναι το συμβάν online καθώς και offline που αφορούν σύνδεση και αποσύνδεση του φυλλομετρητή στο διαδίκτυο.
5. **Συμβάντα ειδικών API**. Τέλος, μια άλλη κατηγορία συμβάντων προκύπτουν από προγραμματιστικές διεπαφές, όπως των στοιχείων <video> και <audio> της HTML, του XMLHttpRequest API κ.λπ.

10.2.2 Καταχώριση χειριστών συμβάντων

Υπάρχουν δύο διαφορετικοί τρόποι για να οριστεί ένας χειριστής συμβάντων: είτε ως τιμή στην αντίστοιχη ιδιότητα του σχετικού αντικείμενου (π.χ. του στοιχείου του DOM) είτε, η πιο πρόσφατη προσέγγιση, μέσω της μεθόδου addEventListener() του αντίστοιχου αντικείμενου, στην οποία περνάμε τον χειριστή και τον τύπο του

συμβάντος ως ορίσματά της.

Χειριστής συμβάντων ως ιδιότητα του αντικείμενου

Κατά σύμβαση, οι ιδιότητες των αντικειμένων οι οποίες αντιστοιχούν σε συμβάντα έχουν όνομα "on"+συμβάν. Για παράδειγμα, onclick, onchange, onload. Θέλει προσοχή το γεγονός ότι οι ιδιότητες αυτές δεν ακολουθούν τη συνηθισμένη σύμβαση **camelCase** που έχουμε ως τώρα δει στην JavaScript.

Παραδείγματος χάρη, αν επιθυμούμε να φορτώνεται ένα τμήμα του κώδικα όταν ολοκληρωθεί το φόρτωμα της ιστοσελίδας (όταν δηλαδή προκύψει το συμβάν load στο αντικείμενο window), τότε:

```
function pageScript() {
    // ο κώδικας
}
window.onload = pageScript;
```

ή η πιο συνηθισμένη έκδοση:

```
window.onload = function() {
    // ο κώδικας
}
```

Επίσης, ο ορισμός της ιδιότητας αυτής μπορεί να γίνει μέσα στον κώδικα HTML (όχι καλή πρακτική) ως γνώρισμα του αντίστοιχου στοιχείου HTML:

```
<button onclick= "console.log('ευχαριστώ')">πατήστε</button>
```

Χειριστής συμβάντων με μέθοδο addEventListener()

Ένας εναλλακτικός τρόπος ορισμού ενός χειριστή συμβάντων, που είναι και ο προτεινόμενος τρόπος, είναι με χρήση της μεθόδου addEventListener() του αντικείμενου στο οποίο επισυνάπτουμε τον χειριστή. Έστω ότι επιθυμούμε να ορίσουμε χειριστή για το συμβάν επιλογής ενός πλήκτρου <**button**>.

```
<button>πατήστε</button>
<script>
    function f(event) {
        console.log('πλήκτρο πατήθηκε οκ');
    }
    // ορισμός συνάρτησης επιστροφής, (χειριστής συμβάντος "click")
    document.querySelector('button').addEventListener('click', f);
</script>
```

Εναλλακτικός τρόπος ορισμού της συνάρτησης στα ορίσματα της μεθόδου addEventListener() ως ανώνυμης συνάρτησης βέλους:

```
document.querySelector('button').addEventListener(
    'click',
    () => { console.log('πλήκτρο πατήθηκε οκ') }
);
```

Θα επανέλθουμε στο τέλος του κεφαλαίου με τη συζήτηση του μηχανισμού διαχείρισης συμβάντων της JavaScript.

Το παράδειγμα πελάτη-κούριερ

Ένα κάπως πιο σύνθετο παράδειγμα ορισμού χειριστών συμβάντων που έχει ως συνέπεια την επικοινωνία μεταξύ των αντικειμένων δύο κλάσεων που περιμένουν ένα συμβάν για αλλαγή της κατάστασής τους είναι το παράδειγμα προσομοίωσης της καθυστέρησης ενός κούριερ μιας μεταφορικής εταιρείας. Ορίζουμε δύο κλάσεις – στο παράδειγμα αυτό πρώτη την κλάση Pelatis:

```
class Pelatis {
    //Ο πελάτης που περιμένει και μετράει τα δευτερόλεπτα
```

```
    constructor(delState, t) {
        this.delivered = delState;
        this.timer = t;
        this.setting = setInterval(()=>this.checkDelivery(), 1000);
    }

    checkDelivery = function () {
        if (this.delivered == 'έφτασε') {
            clearInterval(this.setting);
            display.innerHTML += 'ΠΕΛΑΤΗΣ: Επιτέλους... έκαναν ...' +
this.timer + 'sec.<br>';
            return;
        }

        display.innerHTML += 'ΠΕΛΑΤΗΣ: περιμένω...' + ++this.timer +
"<br>";
    }
}
```

Στη συνέχεια ορίζουμε την κλάση Courier:

```
class Courier{
    // Ο κούριερ που παραδίδει δέμα μετά από delay msec, στον Pelatis
    constructor(pelatis, delay){
        this.pelatis = pelatis;
        this.delay = delay;
        this.setting = setTimeout(
            () => {
                this.pelatis.delivered = 'έφτασε';
                display.innerHTML += 'ΚΟΥΡΙΕΡ: εγώ το παρέδωσα...<br>';
            },
            this.delay);
    }
}
```

Τέλος, αρχικοποιούμε δύο αντικείμενα των κλάσεων ως εξής:

```
p = new Pelatis('όχι', 0);
new Courier(p, 5000);
```

Το αποτέλεσμα που θα προκύψει είναι:

```
ΠΕΛΑΤΗΣ: περιμένω...1
ΠΕΛΑΤΗΣ: περιμένω...2
ΠΕΛΑΤΗΣ: περιμένω...3
ΠΕΛΑΤΗΣ: περιμένω...4
ΠΕΛΑΤΗΣ: περιμένω...5
ΚΟΥΡΙΕΡ: εγώ το παρέδωσα...
ΠΕΛΑΤΗΣ: Επιτέλους... έκαναν ...5sec.
```

Ως τώρα έχουμε δει διάφορα παραδείγματα ορισμού συναρτήσεων επιστροφής που καλούνται είτε μετά παρέλευση ορισμένου χρόνου είτε όταν προκύψει ένα συμβάν που περιμένουν.

Ας δούμε όμως στη συνέχεια προβλήματα που παρουσιάζονται κατά τον προγραμματισμό με χρήση αυτού του μοντέλου ασύγχρονης εκτέλεσης κώδικα.

10.3 Προγραμματισμός με κλήση συναρτήσεων επιστροφής

Έχουμε δει ως τώρα παραδείγματα από συναρτήσεις που καλούνται ασύγχρονα όταν προκύψει κάποιο συμβάν. Ένα πρόβλημα που εμφανίζεται σε αυτό το προγραμματιστικό μοντέλο είναι η δυσκολία που έχουμε να χρησιμοποιήσουμε το αποτέλεσμα που παράγει η συνάρτηση επιστροφής. Το αποτέλεσμα αυτό δεν είναι

διαθέσιμο κατά την κλήση της συνάρτησης, συνεπώς δεν μπορούμε να το εκχωρήσουμε σε μια μεταβλητή, όπως γίνεται με την ακολουθιακή (σύγχρονη) κλήση μιας συνάρτησης.

Ας δούμε ένα παράδειγμα. Έστω η συνάρτηση double(), η οποία μετά την παρέλευση κάποιου χρόνου (2 δευτερόλεπτα) καλεί τη συνάρτηση f() η οποία εκτελεί έναν υπολογισμό και επιστρέφει το αποτέλεσμα.

```
function f(v) {
    return v * 2;
}

function double(value) {
    setTimeout(f(value), 2000);
}

x = double(5);
console.log(x);
```

Το αποτέλεσμα που θα μας επιστρέψει ο παραπάνω κώδικας είναι undefined αφού η double() κατά την κλήση της δεν έχει επιστρέψει το αποτέλεσμά της, αφού εκτελείται ασύγχρονα. Ποιος είναι λοιπόν ο τρόπος για να πάρουμε το αποτέλεσμα της double;

Ένας τρόπος για να λύσουμε το πρόβλημα αυτό είναι να περάσουμε ως όρισμα στην double() μια συνάρτηση που θα αναλάβει να διαχειριστεί το αποτέλεσμα της ασύγχρονης συνάρτησης όταν αυτό είναι διαθέσιμο.

Έστω ότι αυτή είναι η συνάρτηση με όνομα callback(). Ξαναγράφουμε συνεπώς τη συνάρτηση double() που εκτελεί το ασύγχρονο έργο ως εξής:

```
function double(value, callback) {
    setTimeout(
        () => {
            callback(f(value));
        },
        2000);
}
```

Παρατηρούμε ότι αυτή τη φορά περάσαμε στην double() δύο ορίσματα: την αρχική τιμή και τη συνάρτηση callback() που θα πάρει το αποτέλεσμα της f().

Όταν καλέσουμε την double(), θα πρέπει να φροντίσουμε να περάσουμε ως δεύτερο όρισμα μια κατάλληλη συνάρτηση η οποία θα παραλάβει το αποτέλεσμα της f(). Για παράδειγμα:

```
double(5, (x) => {
  console.log(`αποτέλεσμα = ${x}`);
});
```

Αυτή τη φορά όταν τρέξουμε τον κώδικα θα παρατηρήσουμε ότι τίποτα δεν θα συμβεί αρχικά, όμως μετά από περίπου 2 δευτερόλεπτα θα δούμε να εμφανίζεται το μήνυμα:

```
'αποτέλεσμα = 10'
```

Αυτός είναι ένας συνηθισμένος τρόπος χειρισμού αποτελεσμάτων ασύγχρονων συναρτήσεων.

Ένα επόμενο ερώτημα που ανακύπτει είναι πώς να χειριστούμε την περίπτωση που η ασύγχρονη συνάρτηση δεν επιστρέφει το προσδοκώμενο αποτέλεσμα γιατί, για παράδειγμα, ο υπολογισμός απέτυχε (π.χ. ένας εξωτερικός πόρος, όπως ο εξυπηρετητής, δεν είναι διαθέσιμος). Για να καλύψουμε και αυτό το πρόβλημα, θα πρέπει να προβλέψουμε να περάσουμε στην double() μια ακόμη συνάρτηση που θα χειριστεί το σφάλμα, αν αυτό προκύψει.

Μια νεότερη έκδοση της double() που προβλέπει και αυτό το ενδεχόμενο είναι η εξής:

```
function double(value, success, failure) {
    setTimeout(() => {
        const result = f(value);
        if (typeof result !== 'number') {
            failure();
```

```
            } else {
                success(result);
            }
        }, 2000);
    }
```

Η κλήση στη συνάρτηση αυτή τη φορά έχει την εξής μορφή:

```
    double(
        5,
        (x) => {
            console.log(`αποτέλεσμα = ${x}`);
        },
        () => console.log('Σφάλμα: δεν επέστρεψε αποτέλεσμα')
    );
```

Σε αυτό το παράδειγμα έχουμε φροντίσει να περάσουμε ως ορίσματα στην double() δύο συναρτήσεις, που η πρώτη χειρίζεται το αποτέλεσμα που επιστρέφει η ασύγχρονη συνάρτηση και η δεύτερη παράγει ένα μήνυμα σφάλματος αν η ασύγχρονη συνάρτηση αποτύχει στον υπολογισμό.

Τα πράγματα όμως μπορεί να γίνουν ακόμη πιο σύνθετα αν πρέπει να περάσουμε σε μια συνάρτηση τα αποτελέσματα της ασύγχρονης κλήσης που και αυτή καλείται ασύγχρονα κ.ο.κ. Το πρόβλημα αυτό έχει γίνει γνωστό ως η κόλαση των ασύγχρονων κλήσεων, *callback hell*. Μια πρόταση για λύση του προβλήματος αυτού είναι ο μηχανισμός Promise που περιγράφεται στη συνέχεια.

10.4 Ο μηχανισμός Promise

Ο μηχανισμός **Υποσχέσεων (Promise)** εισήχθη στην JavaScript στην έκδοση ES6 για να απλοποιήσει την ασύγχρονη λειτουργία της γλώσσας, να λύσει το πρόβλημα διαδοχικών callbacks και για να επιτρέψει καλύτερη διαχείριση σφαλμάτων που προκύπτουν κατά την ασύγχρονη εκτέλεση κώδικα.

Μια υπόσχεση Promise είναι ένα αντικείμενο που εκπροσωπεί το αποτέλεσμα μιας ασύγχρονης εργασίας.

Promises χρησιμοποιούνται από την JavaScript και στον εξυπηρετητή και στον φυλλομετρητή. Παράδειγμα το Fetch API και το Service Workers API, ενώ ο πιο πρόσφατος μηχανισμός **async/await**, που θα δούμε στο επόμενο κεφάλαιο, βασίζεται επίσης στα promises.

Το αποτέλεσμα της υπόσχεσης δεν μπορούμε να το πάρουμε, για παράδειγμα, εκχωρώντας την υπόσχεση σε μια μεταβλητή, όπως κάνουμε με τη σύγχρονη εκτέλεση κώδικα.

Για να δημιουργήσουμε μια υπόσχεση, καλούμε τη συνάρτηση-δημιουργό Promise(), στην οποία περνάμε μια συνάρτηση executor() που θα εκτελέσει την ασύγχρονη εργασία. Φροντίζουμε στη συνάρτηση αυτή να περάσουμε ως ορίσματα δύο συναρτήσεις επιστροφής, τις resolve() και reject(), οι οποίες θα πρέπει να κληθούν αν η υπόσχεση ικανοποιηθεί ή όχι αντίστοιχα.

```
let p = new Promise(executor(resolve, reject));
```

10.4.1 Καταστάσεις του αντικείμενου Promise

Η εντολή new Promise(executor(resolve, reject)) επιστρέφει ένα αντικείμενο Promise που έχει τις ιδιότητες state και result.

Η ιδιότητα state ορίζει την κατάσταση του αντικείμενου και παίρνει τις τιμές:

- "pending" Ένδειξη ότι η ασύγχρονη εργασία είναι σε εξέλιξη.
- "fulfilled" Ένδειξη ότι η ασύγχρονη εργασία έχει ολοκληρωθεί επιτυχώς.
- "rejected" Ένδειξη ότι η ασύγχρονη εργασία απέτυχε να ολοκληρωθεί.

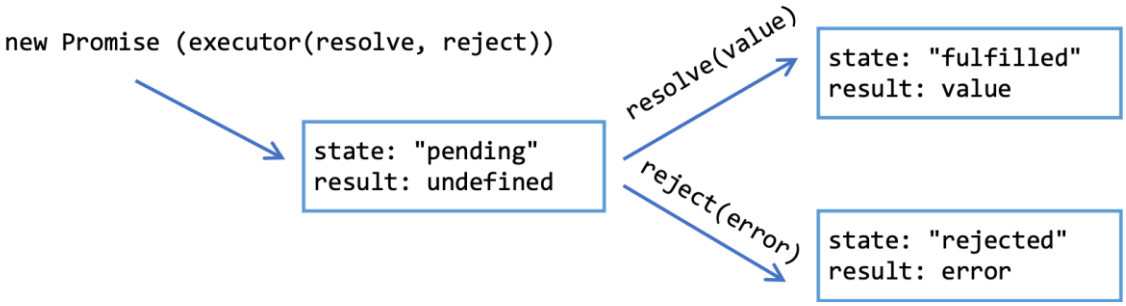
Η ιδιότητα result παίρνει αρχικά την τιμή undefined, ενώ στη συνέχεια παίρνει την τιμή του αποτελέσματος αν η υπόσχεση ικανοποιηθεί ή την τιμή σφάλματος αν όχι.

Θα πρέπει να σημειωθεί ότι, όταν αλλάξει η κατάσταση του αντικείμενου Promise, η resolve() αν ξανακαλεστεί δεν αλλάζει πλέον την κατάσταση.

Η κατάσταση μιας υπόσχεσης μπορεί να αλλάξει σε "fulfilled" ή "rejected", αλλά και να μείνει για πάντα "pending". Ο κώδικάς μας θα πρέπει να προβλέψει όλα αυτά τα ενδεχόμενα. Έχει ενδιαφέρον ότι η κατάσταση

μιας υπόσχεσης είναι ιδιωτική ιδιότητα, στην οποία δεν έχουμε πρόσβαση από την JavaScript και την οποία δεν μπορούμε να αλλάξουμε προγραμματιστικά, εκτός από τη συνάρτηση executor(). Αυτό για να αποφεύγουμε τη σύγχρονη διαχείριση της υπόσχεσης μέσω ελέγχου ή τροποποίησης της κατάστασής της.

Η συνάρτηση εκτέλεσης executor() που περνάμε στον δημιουργό της Promise καλείται αυτόματα από τον δημιουργό του αντικείμενου Promise. Ο ρόλος της συνάρτησης αυτής είναι αφενός να αρχίζει την ασύγχρονη εκτέλεση του κώδικα, αφετέρου να ελέγχει τις αλλαγές κατάστασης. Η αλλαγή της κατάστασης γίνεται με κλήση των δύο συναρτήσεων, παραμέτρων της executor(), τις οποίες ονομάζουμε συνήθως resolve, reject. Όταν κληθεί η resolve() θα αλλάξει η κατάσταση της υπόσχεσης σε "fulfilled", ενώ όταν κληθεί η reject() θα αλλάξει η κατάσταση σε "rejected".



Εικόνα 10.3 *Διάγραμμα καταστάσεων ενός αντικείμενου Promise (υπόσχεση).*

Όταν κληθεί η συνάρτηση resolve(), ως όρισμα περνάμε τα αποτελέσματα ώστε αυτά να αποτελέσουν την τιμή της ιδιότητας result της υπόσχεσης, ενώ αν κληθεί η συνάρτηση reject() σε αυτή περνάμε το αντικείμενο Error ή το μήνυμα σφάλματος, το οποίο στην περίπτωση αυτή αποτελεί την τιμή της ιδιότητας result της υπόσχεσης, όπως φαίνεται στην **Εικόνα *10.3***.

Ας δημιουργήσουμε μια υπόσχεση με συνάρτηση εκτέλεσης που άμεσα καλεί τη συνάρτηση επιστροφής resolve(). Όταν προσπαθήσουμε να εκτυπώσουμε την υπόσχεση στην κονσόλα της JavaScript θα πάρουμε το εξής αποτέλεσμα:

```
let p1 = new Promise((resolve, reject) => resolve(5));
setTimeout(() => console.log(p1), 0);
> Promise {<fulfilled>: 5}
    __proto__: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: 5
```

Αν όμως ορίσουμε ότι η resolve() καλείται με καθυστέρηση 100ms, θα πάρουμε το μήνυμα ότι η κατάσταση της υπόσχεσης είναι <pending>

```
let p1 = new Promise((resolve, reject) =>
    setTimeout(resolve 100 5);
setTimeout(() => console.log(p1), 0);
> Promise {<pending>}
```

Θα πρέπει να σημειωθεί ότι η reject(error) δεν παράγει κατάσταση σφάλματος που μπορεί να ελεγχθεί από τη δομή try/except, η οποία αφορά σφάλματα και σύγχρονη εκτέλεση κώδικα της JavaScript.

Ένα παράδειγμα:

```
try {
    Promise.reject(new Error('Σφάλμα!'));
} catch(e) {
    console.log('διαπιστώθηκε σφάλμα:', e);
}
```

Στον κώδικα αυτόν το τμήμα catch() δεν θα τυπώσει το μήνυμα, ενώ θα πάρουμε το μήνυμα: > Uncaught (in promise) Error: Σφάλμα!

Διαπιστώνουμε εδώ ότι τα σφάλματα στον ασύγχρονο κώδικά μας (στην υπόσχεση) δεν μπορούμε να τα διαχειριστούμε με τον συνήθη τρόπο που έχουμε διαθέσιμο για τον σύγχρονο κώδικα.

Η σύνδεση ανάμεσα στον σύγχρονο και στον ασύγχρονο κώδικα γίνεται με τις μεθόδους καταναλωτές του αντικείμενου Promise, που περιγράφονται στη συνέχεια.

10.4.2 Καταναλωτές υποσχέσεων

Ένας συνηθισμένος τρόπος να πάρουμε τα αποτελέσματα μιας υπόσχεσης είναι να ορίσουμε έναν ή περισσότερους *καταναλωτές* της υπόσχεσης, οι οποίοι υλοποιούνται με τις μεθόδους then(), catch(), finally() του αντικείμενου Promise.

```
p = new Promise(f)
p.then(callbackSuccess, callbackFailure);
```

Σε έναν καταναλωτή then() μπορούμε να περάσουμε δύο συναρτήσεις, την callbackSuccess() για την επεξεργασία του αποτελέσματος της υπόσχεσης και την callbackFailure() για διαχείριση του σφάλματος σε περίπτωση αποτυχίας. Επιτρέπεται σε έναν καταναλωτή then() να περάσουμε μόνο την πρώτη συνάρτηση ως όρισμα, χωρίς να ορίζουμε συνάρτηση διαχείρισης σφάλματος:

```
p.then(callbackSuccess);
```

Ακολουθεί, τέλος, ένα παράδειγμα όπου ορίζουμε μόνο τη συνάρτηση διαχείρισης σφάλματος ως εξής:

```
p.then(null, callbackFailure);
```

Εναλλακτικά, την αποτυχία εκπλήρωσης της υπόσχεσης μπορούμε να την καταναλώσουμε μέσω ενός ειδικού καταναλωτή .catch(errorHandling) ενώ ο καταναλωτής .finally(callback) εκτελείται σε κάθε περίπτωση.

Οι συναρτήσεις αυτές θα ενεργοποιηθούν όταν η υπόσχεση βρεθεί στην αντίστοιχη κατάσταση ("fulfilled" ή "rejected"). Επειδή κάθε υπόσχεση μεταβαίνει προς κάποια από τις δύο αυτές καταστάσεις μία φορά μόνο, είναι προφανές ότι η εκτέλεση των συναρτήσεων callbackSuccess() και callbackFailure() ενός καταναλωτή είναι αμοιβαία αποκλειόμενες.

Θα πρέπει να σημειωθεί ότι σε ένα στιγμιότυπο του αντικείμενο Promise μπορεί να κληθεί ένας από τους παραπάνω καταναλωτές, ο οποίος με τη σειρά του παράγει μια υπόσχεση, συνεπώς, μπορούμε να δημιουργήσουμε μια αλυσίδα καταναλωτών με σημειολογία τελείας p.then().then() .... Να σημειωθεί επίσης ότι, ακόμη και αν στη μέθοδο then() δεν έχει οριστεί συνάρτηση χειριστής, η υπόσχεση περνάει στο επόμενο then της αλυσίδας.

Παράδειγμα χρήσης καταναλωτών

```
const p = new Promise(promisedFunc);

function promisedFunc (resolve,reject) {
    //συναρτήσεις που ενεργοποιούνται σε περίπτωση επιτυχούς ή όχι
ολοκλήρωσης
    if (Math.random()>0.5){ //προσομοίωση τυχαιότητας λειτουργίας
        resolve('Επιτυχής ολοκλήρωση')
    }
    else {
        reject('Σφάλμα')
    }
}

// χρήση της Promise
function callBackSuccess(result){ //συνάρτηση resolve
    console.log(result)
}

function callBackFailure(err){ //συνάρτηση reject
```

```
    console.log(err)
}

p.then(callBackSuccess, callBackFailure) //κλήση Promise
```

Στο παράδειγμα αυτό ορίζουμε μια συνάρτηση promisedFunc() που δέχεται δύο συναρτήσεις ως ορίσματα, τις resolve() και reject(), οι οποίες καλούνται στο σώμα της συνάρτησης διαβιβάζοντάς τους το αποτέλεσμα της υπόσχεσης ή το μήνυμα σφάλματος αντίστοιχα, αφού ενεργοποιούνται στην περίπτωση επιτυχούς ή μη επιτυχούς εκπλήρωσης της υπόσχεσης.

Στη συνέχεια καλούμε τον καταναλωτή then στον οποίο περνάμε δύο συναρτήσεις οι οποίες χειρίζονται το αποτέλεσμα της υπόσχεσης ή το σφάλμα αντίστοιχα, τυπώνοντας το σχετικό μήνυμα. Όταν τρέξουμε κατ' επανάληψη τον παραπάνω κώδικα, θα πάρουμε κάποιες φορές το μήνυμα:

```
> 'Σφάλμα'
```

ή σε άλλες περιπτώσεις το:

```
> 'Επιτυχής ολοκλήρωση'
```

10.4.3 Παράδειγμα αλυσίδας καταναλωτών υποσχέσεων

Όπως είδαμε, η κατανάλωση μιας υπόσχεσης γίνεται κυρίως με την κλήση της μεθόδου .then(f) και η μέθοδος then(), όταν εφαρμόζεται σε ένα αντικείμενο Promise, επιστρέφει Promise, συνεπώς μπορούμε να δημιουργήσουμε αλυσίδα από καταναλωτές υποσχέσεων.

Ας δούμε ένα παράδειγμα αλυσίδας καταναλωτών υποσχέσεων:

```
//αρχή προγράμματος
console.log('αρχή προγράμματος');
new Promise(function (resolve, reject) {
  setTimeout(() => resolve(1), 1000); // η υπόσχεση υλοποιείται σε 1"
})
  .then(function (result) {
    // καλείται ο 1ος καταναλωτής then()
    console.log('καταναλωτής-1: ', result);
    return result * 2; // επιστρέφει την τιμή 2
  })
  .then(function (result) {
    // καλείται ο 2ος καταναλωτής
    console.log('καταναλωτής-2: ', result);
    return result * 2; // επιστρέφει την τιμή 4
  })
  .then(function (result) {
    // καλείται ο 3ος καταναλωτής
    console.log('καταναλωτής-3: ', result);
    return result * 2;
  });
console.log('τέλος προγράμματος');
```

Το αποτέλεσμα από την εκτέλεση του προγράμματος αυτού θα είναι:

```
'αρχή προγράμματος'
'τέλος προγράμματος'
'καταναλωτής-1: ' 1
'καταναλωτής-2: ' 2
'καταναλωτής-3: ' 4
```

Άσκηση 1

Ποιο το αποτέλεσμα του παρακάτω κώδικα;

```
let promise = new Promise((resolve, reject) =>  {
    setTimeout(() => resolve(10), 2000);
    setTimeout(() => resolve(20), 1000);
});

promise.then((result) => {
    console.log(result);
});
```

Απάντηση

Το αποτέλεσμα είναι:

```
20
```

Αυτό γιατί, όταν στην ασύγχρονη συνάρτηση της Promise η resolve() καλείται την πρώτη φορά, αλλάζει η κατάσταση του αντικείμενου, η οποία δεν μπορεί να ξαναλλάξει με επόμενες κλήσεις της resolve().

Άσκηση 2

Στην άσκηση αυτή ζητείται να υλοποιήσετε τη συνάρτηση setTimeout() που είδαμε σε προηγούμενη ενότητα, με χρήση υποσχέσεων. Η συνάρτηση setTimeout(f, d) υπενθυμίζεται ότι εκτελεί τη συνάρτηση f() ασύγχρονα μετά παρέλευση d millisecond. Ζητείται να υλοποιηθεί η ίδια συμπεριφορά μέσω μιας υπόσχεσης Promise, που περιλαμβάνει κλήση της συνάρτησης delay(d), το αποτέλεσμα της οποίας καταναλώνει στη συνέχεια μια συνάρτηση .then(f).

Απάντηση

```
let delay = (ms) => new Promise((result) => setTimeout(result, ms));
```

Μπορούμε να καταναλώσουμε τη συνάρτηση με κλήση της then(), όπως φαίνεται στο παράδειγμα:

```
delay(3000).then(() => console.log("εκτελείται μετά από 3 sec"));
```

10.4.4 Promises στον βρόχο ελέγχου συμβάντων

Είδαμε σε προηγούμενη ενότητα τη λειτουργία του βρόχου ελέγχου συμβάντων που περιλαμβάνει την **ουρά των κλήσεων συναρτήσεων αναμονής** και τη λειτουργία του **σωρού** και της **στοίβας κλήσεων συναρτήσεων**.

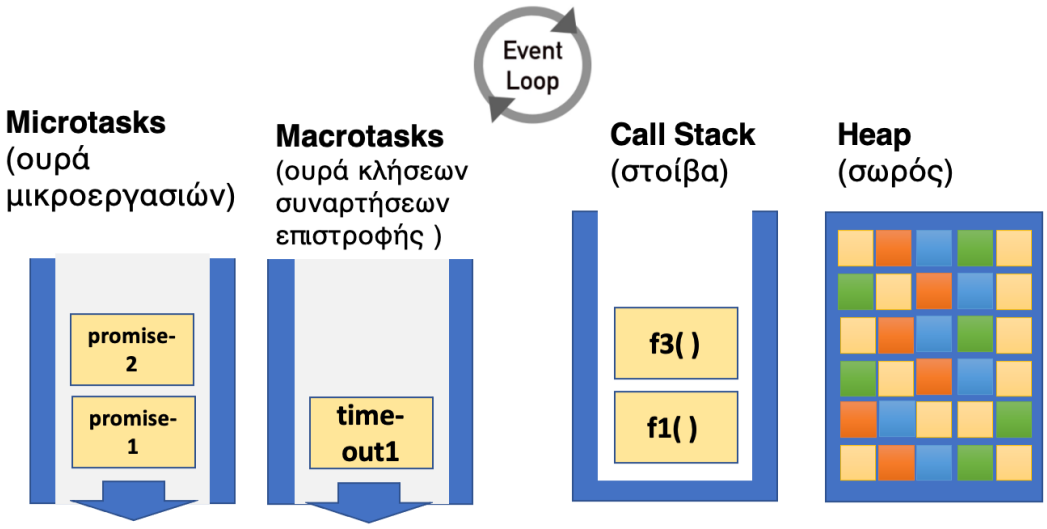
Το ερώτημα που προκύπτει είναι πώς η JavaScript χειρίζεται την ασύγχρονη κλήση συναρτήσεων που γίνεται μέσω του μηχανισμού των Promises.

Έστω, για παράδειγμα, ο παρακάτω κώδικας:

```
console.log('αρχή προγράμματος');
setTimeout(() => console.log('πρώτο timeout'), 0);
requestAnimationFrame(() => console.log('πρώτο animationFrame'));
Promise.resolve()
    .then(() => console.log('promise1'))
    .then(() => console.log('promise2'));
console.log('τέλος προγράμματος');
```

Το ερώτημα είναι με τι σειρά θα εκτελεστούν οι εντολές και ποιο θα είναι το αποτέλεσμα που θα δούμε στην κονσόλα της JS.

Η νέα αρχιτεκτονική του βρόχου συμβάντων φαίνεται στη συνέχεια.



Εικόνα 10.4 *Δομές μνήμης κατά την εκτέλεση κώδικα JavaScript που περιλαμβάνει ασύγχρονα τμήματα κώδικα και χειρισμό υποσχέσεων.*

Οι Promises εισέρχονται σε μια **ουρά μικροεργασιών**, η οποία εξαντλείται σε κάθε κύκλο του βρόχου ελέγχου συμβάντων.

Κατά συνέπεια, η σειρά εκτέλεσης των Promises θα προηγηθεί της κλήσης συνάρτησης επιστροφής από τη μέθοδο setTimeout(), καθώς και της κλήσης του πρώτου animation frame, ενώ η εκτέλεση του τμήματος του προγράμματος που δεν περιλαμβάνει ασύγχρονη εκτέλεση κώδικα θα προηγηθεί όλων:

```
αρχή προγράμματος
τέλος προγράμματος
promise1
promise2
πρώτο animationFrame
πρώτο timeout
```

Θα πρέπει να διευκρινιστεί ότι σε κάθε κύκλο του βρόχου ασύγχρονης εκτέλεσης εργασιών εκτελούνται διάφορα βήματα, που σχετίζονται με την απεικόνιση πληροφορίας (resize, scroll, animation frames κ.λπ.) και εξαντλούνται τα microtasks (promises), ενώ σε διαφορετικούς φυλλομετρητές η σειρά εκτέλεσης των εργασιών αυτών πιθανόν να μεταβάλλεται.

10.5 Η διεπαφή fetch

Από την εποχή του Internet Explorer 5 το 1998 εισήχθη η δυνατότητα ασύγχρονων κλήσεων προς εξυπηρετητές του διαδικτύου από τον φυλλομετρητή με χρήση κλήσεων του **XMLHttpRequest (AJAX)**. Η σύνταξη της σχετικής κλήσης του αντικείμενου XMLHttpRequest ήταν αρκετά σύνθετη.

Η βιβλιοθήκη jQuery, αργότερα, προσέφερε πιο απλή σύνταξη jQuery.ajax(), jQuery.get().

Σήμερα η πιο διαδεδομένη διεπαφή που χρησιμοποιείται για ανάκτηση δεδομένων από εξυπηρετητές είναι η Fetch, που περιλαμβάνει κλήση της μεθόδου fetch(url) και έχει ενσωματωθεί στο αντικείμενο global window, για παράδειγμα, fetch("/file.json"), γεννάει ένα αίτημα GET της HTTP και διαχειρίζεται με ασύγχρονο τρόπο την υποδοχή της απάντησης (μήνυμα HTTP response).

Η μέθοδος fetch(url) επιστρέφει Promise, άρα το αποτέλεσμα μπορούμε να το χειριστούμε με αλυσίδα από κλήσεις μεθόδων then(). Θα πρέπει να σημειωθεί ότι εναλλακτικά θα μπορούσαμε να χειριστούμε το αντικείμενο Promise με χρήση του μηχανισμού async/await, όπως θα δούμε στο επόμενο κεφάλαιο.

10.5.1 Παράδειγμα χρήσης της fetch

Έστω ότι επιθυμούμε την ανάκτηση δεδομένων τύπου JSON από μια διεύθυνση url.

```
function loadJson(url) {
    return fetch(url)
        .then(response => {
            if (response.status == 200) {
                return response.json();
            } else {
                throw new Error(response.status);
            }
        })
        .then(data => console.log("απόκριση:", data))
}

loadJson(url)
.catch(alert); // Error: 404
```

Σ' αυτό το παράδειγμα, στον καταναλωτή then() ελέγχουμε την κατάσταση της απόκρισης μέσω του γνωρίσματος status της απόκρισης που έχουμε λάβει. Αν αυτή είναι 200 ('OK'), τότε καλούμε τη μέθοδο json() του αντικείμενου response. Αυτή η μέθοδος εκτελεί αντίστοιχη λειτουργία με της μεθόδου JSON.parse() που είδαμε σε προηγούμενη ενότητα (9.5.1), δηλαδή μετατρέπει μια συμβολοσειρά σε ένα αντικείμενο JavaScript. Συνεπώς, επιστρέφει αντικείμενο JavaScript που προκύπτει από τη συντακτική ανάλυση του σώματος της απόκρισης. Το αντικείμενο αυτό αποτελεί την απόκριση του promise και μπορεί να καταναλωθεί (π.χ. να εκτυπωθεί στην κονσόλα) από την επόμενη μέθοδο then() της αλυσίδας καταναλωτών.

Ιδιότητες του αντικείμενου που επιστρέφει η fetch

Το αντικείμενο που επιστρέφει η fetch ανήκει στην κλάση Response. Το αντικείμενο αυτό έχει ιδιότητες και μεθόδους που μας επιτρέπουν να χειριστούμε τα αποτελέσματα της κλήσης. Ήδη είδαμε τη μέθοδο json(). Εδώ θα δούμε μερικές ακόμη.

- **headers**. Μέσω του αντικείμενου αυτού έχουμε πρόσβαση στην κεφαλίδα της απόκρισης του HTTP (response header). Ένα παράδειγμα ανάκτησης των γνωρισμάτων Content-Type και Date:

  ```
  fetch('./file.json').then(response => {
      console.log(response.headers.get('Content-Type'))
      console.log(response.headers.get('Date')) })
  ```

- **status**. Ιδιότητα που παίρνει τιμή έναν ακέραιο αριθμό που ορίζει το HTTP response status. Τα 101, 204, 205 ή 304 σημαίνουν null body status, τα 200 μέχρι 299 ότι είναι OK (success), τα 301, 302, 303, 307 ή 308 redirect. Ένα παράδειγμα ανάκτησης του status:

  ```
  fetch('./file.json')
  .then(response => console.log(response.status))
  ```

- **statusText**. Ιδιότητα που παίρνει ως τιμή συμβολοσειρά με το status message της απόκρισης. Για παράδειγμα, έχει την τιμή "OK" αν είναι επιτυχής η απόκριση. Παράδειγμα:

  ```
  fetch('./file.json')
  .then(response => console.log(response.statusText))
  ```

- **url**. Ιδιότητα που έχει ως τιμή τη διεύθυνση URL του αντικείμενου. Παράδειγμα:

  ```
  fetch('./file.json')
  .then(response => console.log(response.url))
  ```

- **Body content**. Η απόκριση περιέχει ένα αντικείμενο body, στο οποίο έχουμε πρόσβαση μέσω διαφόρων μεθόδων: Η μέθοδος text() επιστρέφει το body ως συμβολοσειρά, η json() το επιστρέφει ως αντικείμενο JSON-parsed object, blob() ως Blob object (binary large object), η formData() ως FormData object κ.λπ.

Ανάκτηση διαθέσιμων δεδομένων από διεπαφές REST

Η fetch() μπορεί να έχει πολλές χρήσεις, μεταξύ άλλων να χρησιμοποιηθεί για να ανακτηθούν πληροφορίες από ένα σημείο επαφής (endpoint) μιας διεπαφής REST (Representation State Transfer) συνήθως μέσω αιτήματος GET.

Υπάρχουν πολλοί φορείς που διαθέτουν δημόσια δεδομένα και πληροφορίες, κάποιοι χωρίς να χρειάζεται εγγραφή. Βλέπε σχετικά το Open Data Initiative (opendatainitiative.github.io).

Παραδείγματα τέτοιων διεπαφών είναι:

- https://restcountries.com/ (γεωγραφικά στοιχεία χωρών)
- http://www.7timer.info/doc.php?lang=en#api (πρόβλεψη καιρού)
- https://www.exchangerate-api.com/ (τιμές συναλλάγματος)
- https://covid19api.com/ (υγειονομική κρίση Covid-19)

10.5.2 Παράδειγμα: Οι καλοί γείτονες

Στην ενότητα αυτή θα δούμε ένα παράδειγμα χρήσης της διεπαφής fetch για την ανάκτηση δεδομένων από την προγραμματιστική διεπαφή https://restcountries.com/.

Η διεπαφή αυτή έχει διάφορα σημεία επαφής, μέσω των οποίων μπορούμε να ανακτήσουμε τα στοιχεία μιας χώρας (πληθυσμό, έκταση, πρωτεύουσα, γλώσσα, άλλες χώρες με τις οποίες συνορεύει κ.λπ.) είτε δίνοντας το όνομα της χώρας είτε δίνοντας έναν κωδικό δύο γραμμάτων της χώρας (π.χ. “GR” για την Ελλάδα) είτε κωδικό τριών γραμμάτων κ.λπ.

Ως άσκηση ζητείται να κάνουμε τα εξής:

- Ανακτούμε τα στοιχεία της χώρας από το αντίστοιχο σημείο επαφής (π.χ. για τη χώρα Italy).
- Αφού μελετήσουμε τη δομή των δεδομένων JSON που επιστρέφει η διεπαφή, εξετάζουμε τους γείτονες της χώρας.
- Αναζητούμε τη σχετική πληροφορία για τις χώρες που συνορεύει (Array: borders). Παρατηρούμε εδώ ότι για τη χώρα αυτή ο πίνακας περιέχει τις χώρες με τις οποίες συνορεύει η Ιταλία, κωδικοποιημένες με κωδικό τριών χαρακτήρων: borders = ['AUT', 'FRA', 'SMR', 'SVN', 'CHE', 'VAT'].
- Για καθεμία από τις χώρες αυτές θα πρέπει να βρούμε τα στοιχεία της με βάση τον κώδικά της και από αυτά να διαλέξουμε το όνομά της. Άρα, να ανακτήσουμε τα στοιχεία καθεμίας από τις χώρες αυτές μέσω του σημείου επαφής για κωδικούς τριών χαρακτήρων.

Για την επίλυση του προβλήματος αυτού θα πρέπει να αναζητήσουμε τρόπους εκτέλεσης παράλληλων ασύγχρονων κλήσεων σε διεπαφή δεδομένων.

Για τον σκοπό αυτό μπορούμε να χρησιμοποιήσουμε τη μέθοδο .all(set-of-promises) του αντικείμενου Promise.

Η μέθοδος all() του αντικείμενου Promise δέχεται ως όρισμα ένα σύνολο από υποσχέσεις και επιστρέφει μια υπόσχεση η οποία εκπληρώνεται όταν όλες οι υποσχέσεις έχουν εκπληρωθεί.

Η Promise.all() είναι χρήσιμη για περιπτώσεις που έχουμε ένα σύνολο υποσχέσεων που θα θέλαμε όλες να εκπληρωθούν πριν προχωρήσουμε στην παρουσίαση των αποτελεσμάτων.

Θα πρέπει εδώ, για πληρότητα της παρουσίασης, να γίνει αναφορά και σε άλλες αντίστοιχες μεθόδους του αντικείμενου Promise:

- Promise.all() Σταματάει στην πρώτη άρνηση υπόσχεσης και επιστρέφει error, ενώ επιστρέφει υπόσχεση με τα αποτελέσματα αν όλες εκπληρωθούν.
- Promise.race() Σταματάει στην πρώτη που ολοκληρώνεται είτε ως εκπληρωμένη υπόσχεση είτε ως σφάλμα και την επιστρέφει.
- Promise.any() Σταματάει στην πρώτη που επιστρέφει εκπληρωμένη υπόσχεση και την επιστρέφει.
- Promise.allSettled() Επιστρέφει όλες τις υποσχέσεις, {status: 'fulfilled', value: result} για όσες έχουν εκπληρωθεί και {status: 'rejected', reason: error} για όσες όχι.

Λύση της άσκησης

Αρχικά ορίζουμε μια συνάρτηση loadCountryNameFromCode η οποία παίρνει ως όρισμα τον κωδικό τριών χαρακτήρων μιας χώρας και επιστρέφει μια Promise που περιλαμβάνει το όνομα της χώρας από τα δεδομένα της αντίστοιχης υπόσχεσης. Είναι σημαντικό ότι η συνάρτηση αυτή θα πρέπει να επιστρέφει υπόσχεση ώστε να δημιουργήσουμε έναν πίνακα υποσχέσεων τον οποίο να περάσουμε ως όρισμα στη μέθοδο .all().

```
const urlCountry = 'https://restcountries.com/v3.1/name/';
const urlCode = 'https://restcountries.com/v3.1/alpha/';

function loadCountryNameFromCode(code) {
// επιστρέφει Promise του αποτελέσματος – του ονόματος της χώρας
    return new Promise((resolve, reject) => {
        fetch(urlCode + code)
        .then((resp) => {
            if (resp.status == 200) {
                return resp.json();
            } else reject(new Error(response.status));
        })
        .then((data) => {
            resolve(data.name.common); // όνομα από κωδικό "Greece" από
"GR"
        });
    });
}
```

Το κυρίως πρόγραμμά μας θα είναι η συνάρτηση findBorders(country):

```
function findBorders(country) {
    fetch(urlCountry + country)
    .then((response) => {
        if (response.status === 200) {
            return response.json();
        } else throw new Error(response.status);
    })
    .then((data) => {
        if (data[0].borders.length > 0) { // οι κωδικοί των γειτόνων
            const theCountries = [];
            data[0].borders.forEach((item) => {
                theCountries.push(loadCountryNameFromCode(item));
            });
            Promise.all(theCountries) //  array από Promises
            .then((allCountriesNames) => {
                console.log(
    `Η χώρα ${country} συνορεύει με τις εξής χώρες: ${allCountriesNames}`
                ); // render the result
            })
            .catch((error) => {    console.log(error);    });
    }});
}
```

Η συνάρτηση αυτή αρχικά ανακτά μέσω fetch() τα στοιχεία της χώρας από το σημείο επαφής urlCountry.

Εφόσον ο κωδικός του μηνύματος απόκρισης είναι 200 (OK), κάνουμε συντακτική ανάλυση του αποτελέσματος στον πρώτο καταναλωτή της υπόσχεσης then() και εξάγουμε το αποτέλεσμα με τη μορφή αντικείμενου JavaScript.

Στη συνέχεια, ο επόμενος αλυσιδωτά καταναλωτής then() ελέγχει αν το πρώτο από τα στοιχεία που έχουν επιστραφεί (μπορεί πολλές χώρες να ικανοποιούν τη συμβολοσειρά αναζήτησης country) έχει γνώρισμα borders με πλήθος στοιχείων μεγαλύτερο του 0. Αυτό γιατί μπορεί μια χώρα να μην έχει γείτονες (π.χ. μια νησιωτική χώρα όπως η Ιαπωνία).

Στη συνέχεια γεμίζουμε τον πίνακα theCountries με τα αποτελέσματα της κλήσης της συνάρτησης loadCountryNameFromCode(). Η συνάρτηση αυτή, όπως αναφέρθηκε προηγουμένως, επιστρέφει ένα Promise.

Συνεπώς, διαδοχικά γεμίζουμε τον πίνακα αυτό με ασύγχρονες υποσχέσεις ανάκτησης των στοιχείων όλων των χωρών του πίνακα borders.

Τέλος, περνάμε τον πίνακα αυτό ως όρισμα στην all:

```
Promise.all(theCountries)
```

Το αποτέλεσμα της εντολής αυτής είναι, όπως αναφέρθηκε, υπόσχεση, άρα μπορούμε να την καταναλώσουμε με then().

Στον καταναλωτή αυτόν περνάμε το αποτέλεσμα, που είναι ένας πίνακας που περιέχει τα ονόματα των χωρών, δηλαδή το ζητούμενο της άσκησης.

10.6 Διαχείριση συμβάντων

Όπως έχουμε ήδη συζητήσει, η JavaScript είναι γλώσσα που ακολουθεί το παράδειγμα προγραμματισμού συμβάντων. Είδαμε σε προηγούμενη ενότητα τις διάφορες κατηγορίες συμβάντων που μπορεί να προκύψουν, από συσκευές, ανεξάρτητα συσκευής, συμβάντα της διεπαφής ή συμβάντα από APIs.

Επίσης, έχουμε συζητήσει τρόπους να ορίσουμε έναν χειριστή συμβάντων, είτε ως ιδιότητα του αντίστοιχου στοιχείου, element.onclick = χειριστής, είτε με κλήση της μεθόδου element.addEventListener(συμβάν, χειριστής).

Όταν προκύψει το συμβάν στο συγκεκριμένο στοιχείο το οποίο έχει οριστεί χειριστής του, η συνάρτηση-χειριστής καλείται.

10.6.1 Κλήση χειριστή συμβάντος

Κατά την κλήση του χειριστή συμβάντος περνάμε ως όρισμα το αντικείμενο Event. Το αντικείμενο αυτό έχει τις εξής ιδιότητες:

- event.type, τον τύπο του συμβάντος.
- event.target, το αντικείμενο στο οποίο έχει γίνει το συμβάν.
- event.currentTarget, για συμβάντα τα οποία μεταδίδονται, όπως θα δούμε στη συνέχεια, η ιδιότητα αυτή περιγράφει το αντικείμενο στο οποίο έχει συνδεθεί ο χειριστής του συμβάντος, που μπορεί να είναι διαφορετικό από το event.target.
- event.timeStamp, η χρονική στιγμή κατά την οποία έγινε το συμβάν, μετρημένη σε ms από την αρχή φορτώματος της ιστοσελίδας.
- event.isTrusted, ένδειξη αν το συμβάν προέκυψε από τον φυλλομετρητή (true) ή από τον κώδικα JavaScript.

Επιπροσθέτως, το αντικείμενο Event μπορεί να έχει και άλλες ιδιότητες ανάλογα με τον τύπο του. Για παράδειγμα, το συμβάν click ή εν γένει συμβάντα από το ποντίκι ή, πιο γενικά, από τη δεικτική συσκευή έχουν ως ιδιότητες τα event.clientX και event.clientY, με τις συντεταγμένες του σημείου στο παράθυρο. Αν το συμβάν αφορά το πληκτρολόγιο, έχει ως ιδιότητα τον κωδικό του χαρακτήρα που πληκτρολογήθηκε ως event.keyCode κ.λπ.

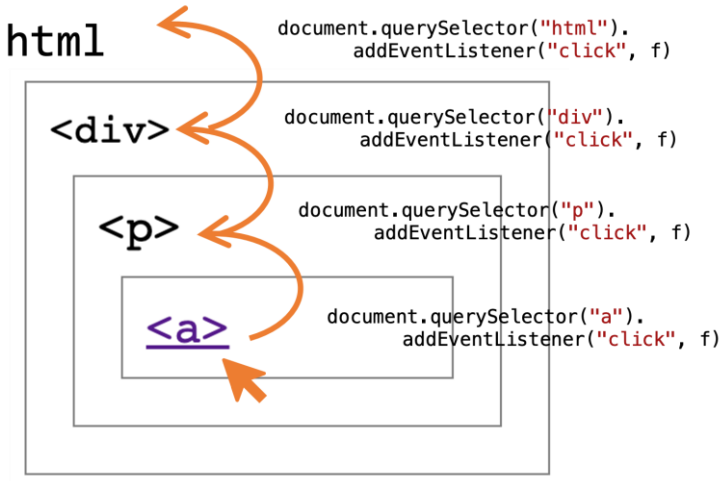
Θα πρέπει να λάβουμε υπόψη μας το γεγονός ότι ένας χειριστής ορίζεται ως τιμή στην ιδιότητα ενός αντικείμενου, συνεπώς ο τελεστής this σε έναν χειριστή αναφέρεται στο αντικείμενο στο οποίο αυτός έχει οριστεί.

10.6.2 Ο μηχανισμός φυσαλίδας

Ένα ενδιαφέρον θέμα είναι η διαχείριση συμβάντων που σχετίζονται με στοιχεία του DOM τα οποία ανήκουν σε μια ιεραρχία. Αν ο χρήστης επιλέξει ένα στοιχείο, επιλέγει και το στοιχείο στο οποίο αυτό ανήκει και εκείνο στο οποίο ανήκει ο πατέρας του κ.ο.κ. Το ερώτημα που γεννάται είναι, αν έχουν οριστεί χειριστές του συμβάντος στα ανώτερα επίπεδα της ιεραρχίας, αυτοί θα αντιδράσουν επίσης στο συμβάν;

Η JavaScript έχει έναν σύνθετο μηχανισμό διάδοσης συμβάντων (ο **μηχανισμός φυσαλίδας**, event bubbling), που περιγράφεται στη συνέχεια, για να αντιμετωπίσει το θέμα αυτό.

Ας υποθέσουμε το παράδειγμα στην **Εικόνα *10.5***:



Εικόνα 10.5 *Παράδειγμα ιεραρχίας στοιχείων και αντίστοιχων χειριστών συμβάντων.*

Ας υποθέσουμε ότι σε ένα έγγραφο HTML έχουμε την ιεραρχία που φαίνεται και στην ***Εικόνα 10.5***:

```
<html>
    <body>
        <div>
            <p>
                <a> </a>
            </p>
        </div>
    </body>
</html>
```

Ας υποθέσουμε επίσης ότι έχουν οριστεί οι παρακάτω χειριστές για το έγγραφο αυτό:

```
function f(event){
    console.log(`φυσαλίδα συμβάντος: ${event.target} this=
${this.tagName}`);
}

document.querySelector("a").addEventListener("click", f)
document.querySelector("p").addEventListener("click", f)
document.querySelector("div").addEventListener("click", f)
document.querySelector("html").addEventListener("click", f)
```

Για τα τέσσερα αυτά στοιχεία της ιεραρχίας έχει οριστεί η συνάρτηση f(), η οποία καλείται αν ο χρήστης κάνει κλικ με τη δεικτική συσκευή στο αντίστοιχο στοιχείο. Επίσης, να σημειώσουμε ότι έχουμε ορίσει ο *χειριστής* να μας γνωρίζει ότι έχει κληθεί τυπώνοντας σχετικό μήνυμα στο οποίο μας δίνει την τιμή της ιδιότητας event.target και της ιδιότητας this.tagName.

Ας εξετάσουμε στη συνέχεια τι θα συμβεί αν ο χρήστης επιλέξει το στοιχείο **<a>**.

Στην περίπτωση αυτή στην κονσόλα θα δούμε τα εξής μηνύματα:

```
> φυσαλίδα συμβάντος: http://localhost/event2.html# this= A
> φυσαλίδα συμβάντος: http://localhost/event2.html# this= P
> φυσαλίδα συμβάντος: http://localhost/event2.html# this= DIV
> φυσαλίδα συμβάντος: http://localhost/event2.html# this= HTML
```

Παρατηρούμε ότι το συμβάν του χρήστη έχει προκαλέσει τέσσερις διαδοχικές κλήσεις του χειριστή, που έχουν προκύψει από τα στοιχεία **<a>**, **<p>**, ** **, **<html>** με αυτή τη σειρά, καθώς το συμβάν διαδόθηκε στην ιεραρχία μέχρι το ανώτερο επίπεδο και ενεργοποιήθηκαν όλοι οι χειριστές της ιεραρχίας αυτής. Τα μηνύματα αυτά έκαναν όλα αναφορά στην ίδια τιμή του event.target (τον υπερσύνδεσμο που επέλεξε ο χρήστης), αλλά σε διαφορετική κάθε φορά τιμή του this.tagName, που σχετίζεται με στοιχείο στο οποίο έχει επισυναφθεί ο

αντίστοιχος χειριστής.

Αν, αντίθετα, κάνουμε κλικ στην περιοχή του στοιχείου <**div**> έξω από τα υπόλοιπα εσωτερικά στοιχεία, θα πάρουμε μόνο τα εξής δύο μηνύματα:

```
> φυσαλίδα συμβάντος: [object HTMLDivElement] this= DIV
> φυσαλίδα συμβάντος: [object HTMLDivElement] this= HTML
```

Συνέπεια του φαινομένου αυτού είναι ότι μπορούμε να ορίσουμε έναν χειριστή για έναν υποδοχέα που περιέχει πολλά στοιχεία και να διαγνώσουμε ποιο συγκεκριμένο στοιχείο είναι το αντικείμενο που έχει επιλέξει ο χρήστης ελέγχοντας την τιμή του event.target.

Θα πρέπει να σημειώσουμε ότι κάποια συμβάντα δεν διαδίδονται με τον μηχανισμό φυσαλίδας, αυτά περιλαμβάνουν τα συμβάντα “focus”, “blur” και “scroll”.

Επίσης, το φαινόμενο διάδοσης ενός συμβάντος είναι ακόμη πιο σύνθετο από τον μηχανισμό φυσαλίδας που ήδη περιγράφηκε. Η πλήρης περιγραφή του μηχανισμού αναλύεται στη συνέχεια.

Φάση σύλληψης συμβάντος

Όταν γίνει ένα συμβάν, ξεκινάει ένας έλεγχος για χειριστές του συγκεκριμένου συμβάντος από τη ρίζα του δένδρου προς τα κάτω, μέχρι να φτάσουμε στο κατώτερο επίπεδο στο οποίο υπάρχει χειριστής. Η φάση αυτή περιγράφεται ως **φάση σύλληψης του συμβάντος** (event capturing), προηγείται της κλήσης του χειριστή συμβάντος και δεν προκαλεί κλήση των χειριστών που θα βρεθούν στην πορεία αυτή.

Όταν ο έλεγχος φτάσει στο κατώτερο επίπεδο που βρίσκεται αντίστοιχος χειριστής, τότε αυτός ενεργοποιείται και αρχίζει η δεύτερη φάση της διάδοσης προς τα πάνω με τον μηχανισμό φυσαλίδας που ήδη αναφέρθηκε.

Το ερώτημα είναι ποιος ο λόγος ύπαρξης της φάσης σύλληψης του συμβάντος. Η χρήση αυτής της φάσης του μηχανισμού είναι περιορισμένη, αλλά μπορεί να φανεί χρήσιμη για να εμποδιστεί ένα συμβάν να διαδοθεί προς τα κάτω. Ένα παράδειγμα είναι, όταν θέλουμε να σύρουμε ένα αντικείμενο στην επιφάνεια εργασίας, δεν επιθυμούμε να προκληθούν συμβάντα που σχετίζονται με το πέρασμα της συσκευής από υποκείμενα στοιχεία της διεπαφής. Πώς όμως θα οριστεί η συμπεριφορά αυτή; Αυτό γίνεται κατά τον ορισμό του χειριστή που θα αναλάβει αυτό το έργο. Ο ορισμός του χειριστή αυτού έχει την εξής σύνταξη:

```
element.addEventListener(συμβάν, χειριστής, {capture: true})
```

Στον χειριστή που έχει οριστεί με τη σύνταξη αυτή μπορεί να κληθεί η μέθοδος κατάργησης του συμβάντος event.stopPropagation() η οποία σταματάει τη διάδοση του συμβάντος προς τα κάτω και συνεπώς την κλήση του χειριστή.

Θα πρέπει να σημειωθεί ότι η μέθοδος αυτή σταματάει τη διάδοση ενός συμβάντος και προς τα πάνω αν βρεθεί σε έναν χειριστή του συμβάντος κατά τη φάση της διάδοσης με τον μηχανισμό φυσαλίδας.

Αποτροπή προκαθορισμένης συμπεριφοράς

Τα συμβάντα προκαλούν προκαθορισμένη συμπεριφορά σε μια ιστοσελίδα.

Για παράδειγμα, όταν ο χρήστης επιλέξει έναν υπερσύνδεσμο ο φυλλομετρητής ξεκινάει κλήση για φόρτωμα της ιστοσελίδας στόχου, όταν ο χρήστης πληκτρολογήσει έναν χαρακτήρα σε ένα πλαίσιο κειμένου, ο φυλλομετρητής θα εμφανίσει τον χαρακτήρα στο πλαίσιο, όταν σύρει το δάχτυλό του σε μια οθόνη αφής θα προκληθεί κύλιση της οθόνης ή μετάβαση σε προηγούμενη σελίδα, όταν επιλέξει το πλήκτρο «Υποβολή» σε μια φόρμα, θα σταλθεί το περιεχόμενο της φόρμας στον ορισμένο αποδέκτη.

Ένας χειριστής ενός τέτοιου συμβάντος μπορεί να αποτρέψει τον φυλλομετρητή από το να ξεκινήσει την προκαθορισμένη ενέργεια με κλήση της μεθόδου event.preventDefault().

Μια συνηθισμένη χρήση της μεθόδου αυτής είναι σε μια φόρμα, αν ορίσουμε έναν χειριστή του συμβάντος “submit” με σκοπό να κάνουμε έλεγχο εγκυρότητας των δεδομένων που έχει εισαγάγει ο χρήστης. Αν ο έλεγχος αυτός διαπιστώσει σφάλματα στα στοιχεία της φόρμας, τότε καλείται η μέθοδος event.preventDefault(), ώστε να αποτραπεί η υποβολή της φόρμας και να δοθεί η ευκαιρία μέσω κατάλληλων υποδείξεων στον χρήστη να διορθώσει τα στοιχεία πριν υποβληθούν. Για τη χρήση της preventDefault() μπορείτε να ανατρέξετε και στο παράδειγμα της ενότητας **6.7.1 Έλεγχος εγκυρότητας με την Bootstrap.**

10.7 Ερωτήσεις αυτοαξιολόγησης

1. Η JavaScript έχει διάφορους μηχανισμούς για ασύγχρονη εκτέλεση ενός τμήματος του προγράμματος, σημειώστε όλους όσους ισχύουν.
    1. Ο μηχανισμός κλήσεων συναρτήσεων επιστροφής (call back functions).
    2. Ο μηχανισμός της στοίβας κλήσεων (call stack).
    3. Η μέθοδος fetch.
    4. Ο μηχανισμός υποσχέσεων Promises.
2. Η fetch με ποιον μηχανισμό ασύγχρονης εκτέλεσης προγράμματος υλοποιείται;
    1. Με τον μηχανισμό Promise.
    2. Με τον μηχανισμό κλήσης συνάρτησης επιστροφής callback.
    3. Και με τον μηχανισμό Promise και με τον μηχανισμό callback.
3. Ποιος ο ρόλος της στοίβας (call stack) κατά την εκτέλεση ενός προγράμματος JavaScript;
    1. Είναι ένας μηχανισμός για τον έλεγχο της κλήσης και εκτέλεσης συναρτήσεων.
    2. Είναι ένας χώρος αποθήκευσης δεδομένων των συναρτήσεων.
    3. Είναι ένας χώρος ελέγχου εκτέλεσης συναρτήσεων επιστροφής (callbacks).
4. Η JavaScript είναι πολυ-νηματική (multi-threaded) γλώσσα προγραμματισμού.
    – Σωστό/Λάθος
5. Οι συναρτήσεις που εκτελούνται με ασύγχρονο τρόπο τίθενται:
    1. στην ουρά κλήσεων συναρτήσεων επιστροφής (callback queue).
    2. στη στοίβα κλήσεων (call stack).
    3. στον σωρό (heap).
6. Προβλέψτε το αποτέλεσμα εκτέλεσης του παρακάτω κώδικα:

```
console.log('1');
setTimeout(() => {console.log('2')}, 10);
setTimeout(() => {console.log('3')}, 0);
console.log('4');
```

    1. σειρά 1, 2, 3, 4
    2. σειρά 1, 4, 2, 3
    3. σειρά 1, 4, 3, 2
    4. σειρά 1, 3, 2, 4
7. Έστω ο παρακάτω κώδικας:

```
let counter = 0;
const i = setInterval(() => {console.log(++counter)}, 100);
setTimeout(clearInterval, 300, i);
```

    Πόσες διαφορετικές τιμές θα τυπωθούν κατά την εκτέλεσή του;
    Απάντηση: _____
8. Ο μηχανισμός Promise έχει εισαχθεί στην JavaScript από το 2015 (έκδοση ES6).
    – Σωστό/Λάθος
9. Καταναλωτές ενός Promise μπορεί να είναι οι εξής μέθοδοι:
    1. try()
    2. catch()
    3. then()
    4. finally()
10. Το όρισμα φ της κλάσης Promise(φ) είναι:
    1. μια συνάρτηση που εκτελεί την ασύγχρονη λειτουργία.
    2. δύο συναρτήσεις που η μια καλείται σε περίπτωση επιτυχούς ολοκλήρωσης και η άλλη σε περίπτωση αποτυχίας.
    3. δύο μεταβλητές status και result οι οποίες στην αρχή έχουν την τιμή “pending” και “undifined”.
11. Η ουρά μικροεργασιών (microtask queue) διαχειρίζεται τους καταναλωτές υποσχέσεων που βρίσκονται σε αναμονή.
    – Σωστό/Λάθος
12. Ποια η διαφορά μεταξύ ουράς μικροεργασιών και ουράς μακροεργασιών; Σημειώστε όλα όσα ισχύουν.

1. Η ουρά των μικροεργασιών σε κάθε κύκλο του βρόχου ασύγχρονης εκτέλεσης αδειάζει, ενώ από την ουρά μακροεργασιών εκτελείται το πρώτο στοιχείο της ουράς.
2. Η ουρά μικροεργασιών είναι πιο μικρή από την ουρά μακροεργασιών.
3. Η ουρά μακροεργασιών τροφοδοτείται από συναρτήσεις και τμήματα του κώδικα που προκύπτουν από κλήσεις επιστροφής (call back), ενώ η ουρά μικροεργασιών από καταναλωτές υποσχέσεων (promise).

13. Ποιο το αποτέλεσμα του παρακάτω προγράμματος;

```
new Promise((res, rej) => {
    setTimeout(() => res(10), 2000);
    setTimeout(() => res(20), 3000);
    setTimeout(() => rej(30), 1000);
    setTimeout(() => rej(50), 4000);
})
.then((r) => {console.log("result:"+r);})
.catch((e) => {console.log("error:"+e);});
```

1. Διαδοχικά θα τυπωθούν error:30, result:10, result:20, error:50
2. Διαδοχικά θα τυπωθούν result:10, result:20
3. Θα τυπωθεί result:10
4. Θα τυπωθεί error:30
5. Διαδοχικά θα τυπωθούν error:30, error:50

14. Ποιο το αποτέλεσμα εκτέλεσης του προγράμματος;

```
console.log('αρχή προγράμματος');
setTimeout(() => {console.log('setTimeout');}, 0);
const p = new Promise((resolve, reject) => resolve());
p.then(setTimeout(() => {console.log('promise1');}, 100));
p.then(setTimeout(() => {console.log('promise2');}, 100));
console.log('τέλος προγράμματος');
```

1. Διαδοχικά θα τυπωθούν: ‘αρχή προγράμματος’, ‘setTimeout’, ‘promise1’, ‘promise2’, ‘τέλος προγράμματος’.
2. Διαδοχικά θα τυπωθούν: ‘αρχή προγράμματος’, ‘τέλος προγράμματος’, ‘setTimeout’, ‘promise1’, ‘promise2’.
3. Διαδοχικά θα τυπωθούν: ‘αρχή προγράμματος’, ‘τέλος προγράμματος’, ‘promise1’, ‘promise2’, ‘setTimeout’.

15. Τι επιστρέφει η μέθοδος then(φ,ψ) του αντικείμενου Promise;
1. Το αποτέλεσμα της επεξεργασίας του αποτελέσματος της υπόσχεσης.
2. Ένα αντικείμενο τύπου Promise.
3. true αν είναι επιτυχή τα αποτελέσματα της υπόσχεσης ή false αν είναι ανεπιτυχή.

16. Ποια η απόκριση του παρακάτω κώδικα;

```
p = new Promise((res, rej) => {
if (Math.random() > 0.5) res(1);
else rej(0);});
p.then(
(r) => {console.log('r1:' + r++)},
(e) => {console.log('e1:' + e++)});
p.then(
(r) => {console.log('r2:' + r++)},
(e) => {console.log('e2:' + e++)});
```

1. Διαδοχικά r1:1, e1:0, r2:2, e2:1
2. Διαδοχικά r1:1, r2:1, e1:0, e2:0
3. Είτε r1:1, r2:1 είτε e1:0, e2:0
4. Είτε r1:1, r2:2 είτε e1:0, e2:1

17. Ποια η απόκριση του παρακάτω κώδικα;

```
p = new Promise((resolve, reject) => {
if (Math.random() > 0.5) resolve(1);
else reject(0);
});
p.then((r) => {
if (r) {
    console.log('r1:' + r++);
    return r;
}
})
.then((r) => {
    console.log('r2:' + r++);
})
.catch((e) => {
    console.log('e1:' + e++);
});
```

1. Διαδοχικά r1:1, e1:0, r2:2
2. Διαδοχικά r1:1, r2:2, e1:0
3. Είτε r1:1, r2:1 είτε e1:0

18. Η δυνατότητα ασύγχρονης κλήσης μια ιστοσελίδας στον εξυπηρετητή εισήχθη για πρώτη φορά το 2015 με την έκδοση ES6 της JavaScript.
    – Σωστό/Λάθος
19. Η μέθοδος fetch() ανήκει στο αντικείμενο…
    Απάντηση: _____
20. Η μέθοδος fetch() επιστρέφει ένα αντικείμενο:
    1. JSON
    2. HTTP response
    3. Promise
21. Στον παρακάτω κώδικα ποια η απόκριση σε περίπτωση επιτυχούς κλήσης της fetch;
    fetch("/api.json").then((r) => ({console.log(r.statusText)}))
    Απάντηση: _____
22. Αν η μέθοδος fetch(url) επιστρέψει στο body δεδομένα σε μορφή JSON, η επεξεργασία τους με την ασύγχρονη μέθοδο .json() είναι αντίστοιχη με ποια από τις μεθόδους που έχουμε δει στη σύγχρονη επεξεργασία δεδομένων;
    1. JSON.parse(body)
    2. JSON.stringify(body)
    3. JSON.process(body)
23. Ποιο το αποτέλεσμα του παρακάτω κώδικα;

```
const thePromises = [];
for (let i = 0; i < 3; i++) {
thePromises.push(
    new Promise((resolve, reject) => {
    const res = Math.floor(Math.random() * 10 + 1);
    if (Math.random() > 0.1) {
        setTimeout(() => resolve('OK'), res * 10);
    } else {
        reject('error...');
    }
    })
);
}
Promise.all(thePromises)
.then((res) => {console.log(res)})
.catch((er) => {console.log(er)});
```

1. Είτε [‘OK’, ‘OK’, ‘OK’] είτε [‘error…’, ‘error…’, ‘error…’]

2. Ένα array 3 στοιχείων που περιλαμβάνει στοιχεία είτε ‘OK’ είτε ‘error…’
3. Είτε [‘OK’, ‘OK’, ‘OK’] είτε ‘error…’

24. Η διαφορά της Promise.any() από την Promise.race() είναι:
    1. Η Promise.any() επιστρέφει μια τυχαία υπόσχεση που έχει ολοκληρωθεί, ενώ η Promise.race() την πρώτη που έχει ολοκληρωθεί.
    2. Η Promise.any() επιστρέφει την πρώτη υπόσχεση που έχει ολοκληρωθεί επιτυχώς, ενώ η Promise.race() την πρώτη που έχει ολοκληρωθεί ανεξάρτητα αποτελέσματος.
    3. Η Promise.any() επιστρέφει στοιχεία για όλες τις υποσχέσεις, ενώ η Promise.race() μόνο για αυτή που τερμάτισε πρώτη.
25. Ποια είναι η μέθοδος του αντικείμενου Promise η οποία δέχεται ένα σύνολο από υποσχέσεις και επιστρέφει μια υπόσχεση με όλα τα αποτελέσματα, είτε αυτά είναι επιτυχή είτε όχι; Promise([p1,p2,...]). _____()
    Απάντηση: _____

10.8 Βιβλιογραφία και Αναφορές

Το πρότυπο EcmaScript (ECMA-262) συντηρείται από τον οργανισμό ecma.

Η βιβλιογραφία του κεφαλαίου αυτού μόνο εν μέρει καλύπτεται από τη βιβλιογραφία των προηγούμενων κεφαλαίων.

Οι πηγές για εκμάθηση της JavaScript στο διαδίκτυο καλύπτουν και τα θέματα αυτού του κεφαλαίου. Η MDN είναι μια καλή πηγή για εισαγωγικά και προχωρημένα μαθήματα, όπως και για την HTML και CSS. Επίσης, η w3schools περιέχει μαθήματα, ενώ η JavaScript.info περιλαμβάνει μια πλήρη σειρά μαθημάτων για την JavaScript με παραδείγματα, ξεκινώντας από τη γλώσσα και προχωρώντας στη διεπαφή της με τον φυλλομετρητή.

Τα θέματα του κεφαλαίου αυτού καλύπτονται επίσης από τα εγχειρίδια της JavaScript. Στις πηγές αυτές συχνά περιλαμβάνονται και κεφάλαια που αφορούν τη λειτουργία της γλώσσας στο περιβάλλον node.js. Αυτή την προσέγγιση ακολουθεί το βιβλίο του Λιακέα (2021). Από τη διεθνή βιβλιογραφία παρόμοια προσέγγιση ακολουθούν πολλά εγχειρίδια, όπως του Flanagan (2020), του Frisbie (2020) κ.λπ.

Επιπλέον, οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Για αυτό το κεφάλαιο το σχετικό μάθημα είναι οι διαλέξεις του μαθήματος «Προχωρημένα θέματα ανάπτυξης ιστοσελίδων».

Α. Ξενόγλωσσες

Flanagan, D. (2020). *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language* (7th ed.). O'Reilly Media, Inc.

Frisbie, M. (2020). *Professional JavaScript for Web Developers*. Wiley.

Β. Ελληνόγλωσσες

Αβούρης, Ν., & Σιντόρης, Χ. (2020). Προχωρημένα θέματα ανάπτυξης ιστοσελίδων. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr

Λιακέας, Γ. (2021). *Η γλώσσα JavaScript* (3η έκδ.). Κλειδάριθμος. Κωδικός βιβλίου στον Εύδοξο: 102070465.

Κεφάλαιο 11. Προγραμματισμός στην πλευρά του εξυπηρετητή

Σύνοψη

Στο κεφάλαιο θα χρησιμοποιήσουμε την JavaScript ως γλώσσα προγραμματισμού εφαρμογών από την πλευρά του εξυπηρετητή στο προγραμματιστικό περιβάλλον της Node.js. Η Node.js παρέχει ένα οικοσύστημα για την υποστήριξη της ανάπτυξης εφαρμογών και συνοδεύεται από εργαλεία όπως το npm για τη διαχείριση πακέτων. Θα παρουσιαστούν τα δύο κύρια συστήματα βιβλιοθηκών στην JavaScript, το CommonJS και το ES6 Modules (ESM), καθώς και το πώς μπορούμε να γράψουμε και να χρησιμοποιήσουμε βιβλιοθήκες ESM. Θα χρησιμοποιήσουμε κάποιες βασικές βιβλιοθήκες της Node.js, την fs και την http, στα παραδείγματά μας σε τρεις εκδοχές, σύγχρονα, ασύγχρονα με συναρτήσεις επιστροφής και ασύγχρονα με τη διεπαφή των Promises. Με την ευκαιρία αυτή θα χρησιμοποιήσουμε τη διεπαφή async/await σε κώδικα που χρησιμοποιεί τα Promises.

Επιπλέον, θα παρουσιαστούν μερικά πολύ βασικά εργαλεία και βοηθήματα για την ανάπτυξη εφαρμογών, όπως το nodemon και το Visual Studio Code, για την αποσφαλμάτωση των προγραμμάτων μας, καθώς και τη δημοσίευση των εφαρμογών διαδικτύου μας στο Heroku.

Προαπαιτούμενη γνώση

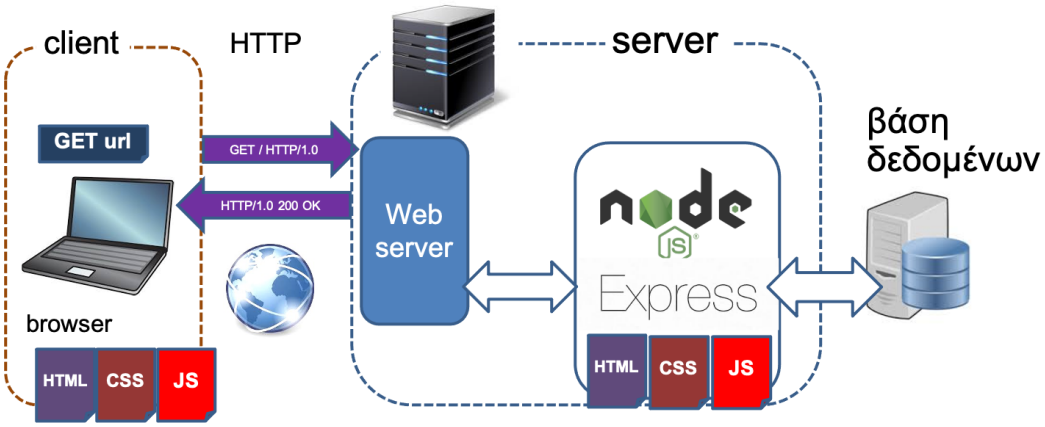
Για τη μελέτη αυτού του κεφαλαίου είναι απαραίτητη η εξοικείωση με τη γλώσσα JavaScript, όπως έχει παρουσιαστεί στα προηγούμενα κεφάλαια (7, 8, 9 και 10). Απαραίτητη είναι και η μελέτη του κεφαλαίου 1, όπου παρουσιάζεται η αρχιτεκτονική του διαδικτύου και το πρωτόκολλο HTTP.

11.1 Εισαγωγή

Έχοντας φτάσει στο κεφάλαιο αυτό, έχουμε αποκτήσει μια πολύ καλή γνώση των βασικών συστατικών μιας διαδικτυακής εφαρμογής από την πλευρά του φυλλομετρητή, δηλαδή του *πελάτη*, για να χρησιμοποιήσουμε την ορολογία του μοντέλου πελάτη-εξυπηρετητή. Όπως είδαμε, το περιεχόμενο προβάλλεται στον φυλλομετρητή χρησιμοποιώντας τις τεχνολογίες HTML, CSS και JavaScript. Το περιεχόμενο αυτό είναι μία ή περισσότερες *στατικές* σελίδες, δηλαδή σελίδες που είναι γραμμένες από πριν και αποθηκευμένες σε κάποιον υπολογιστή, τον εξυπηρετητή. Ο εξυπηρετητής τις παρέχει σε όποιον πελάτη τις ζητήσει, χωρίς μεταβολές και χωρίς να παρέμβει στο περιεχόμενο. Αυτή είναι και η αρχική ιδέα πίσω από τον παγκόσμιο ιστό, δηλαδή αρχεία HTML που βρίσκονται σε διαφορετικούς εξυπηρετητές και τα οποία είναι συνδεμένα μέσω υπερσυνδέσμων.

Στις περισσότερες περιπτώσεις είναι πιο χρήσιμο η εφαρμογή μας να επιστρέφει περιεχόμενο που να είναι προσαρμοσμένο στον χρήστη, να είναι δηλαδή αποτέλεσμα κάποιας επεξεργασίας. Σε αυτό και στα επόμενα κεφάλαια θα παρουσιαστούν οι διαδικτυακές εφαρμογές από την πλευρά του εξυπηρετητή. Η επεξεργασία πραγματοποιείται στον εξυπηρετητή χρησιμοποιώντας κάποια γλώσσα προγραμματισμού. Στο βιβλίο αυτό η γλώσσα με την οποία γράφουμε εφαρμογές στην πλευρά του εξυπηρετητή θα συνεχίσει να είναι η JavaScript, την οποία έχουμε ήδη δει εκτεταμένα στα προηγούμενα κεφάλαια. Άλλες δημοφιλείς τεχνολογίες για τη συγγραφή προγραμμάτων στην πλευρά του εξυπηρετητή είναι η PHP, η Python, η Java, C# κ.λπ. Στην πραγματικότητα η JavaScript είναι η μοναδική γλώσσα με την οποία μπορούμε να γράψουμε προγράμματα τόσο στην πλευρά του φυλλομετρητή όσο και του εξυπηρετητή.

Στην ***Εικόνα 11.1*** φαίνεται η αρχιτεκτονική πελάτη-εξυπηρετητή όπου παράγονται δυναμικά οι ιστοσελίδες, όπως είδαμε και στην ενότητα 1.5. Ο εξυπηρετητής στο αίτημα του πελάτη (GET) δεν επιστρέφει μια στατική σελίδα HTML, αλλά αναθέτει στη Node.js να ετοιμάσει ένα έγγραφο HTML, ενδεχομένως διαβάζοντας πληροφορίες που υπάρχουν σε κάποια βάση δεδομένων, να ενσωματώσει τυχόν CSS και JavaScript και έπειτα να το επιστρέψει στον πελάτη.



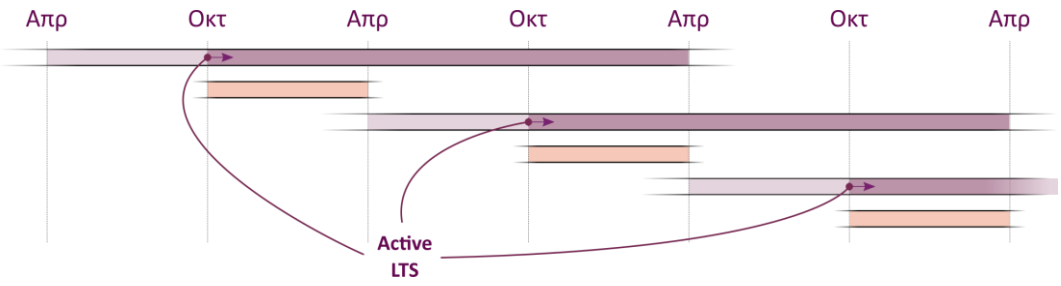
Εικόνα 11.1 *Αρχιτεκτονική μιας τυπικής εφαρμογής που θα αναπτύξουμε στο πλαίσιο του βιβλίου.*

Βέβαια, αν και σημαντική, η εξοικείωση με μια γλώσσα προγραμματισμού δεν είναι πάντα ο πιο καθοριστικός παράγοντας για την επιλογή της στην ανάπτυξη εφαρμογών εξυπηρετητή. Μεγάλη σημασία έχει και το ευρύτερο οικοσύστημα, τα εργαλεία που συνοδεύουν και συμπληρώνουν τη γλώσσα για την ανάπτυξη στον εξυπηρετητή. Όσον αφορά την JavaScript, το κύριο εργαλείο μας είναι η πλατφόρμα Node.js. Η Node.js είναι ένα περιβάλλον με το οποίο μπορούμε να εκτελέσουμε κώδικα γραμμένο σε JavaScript. Μέχρι τώρα έχουμε δει πως ο κώδικας JavaScript εκτελείται από τον φυλλομετρητή από μια μηχανή εκτέλεσης κώδικα JavaScript (Ενότητα 7.1.4). Η Node.js χρησιμοποιεί τη μηχανή JavaScript του φυλλομετρητή Chrome, τη μηχανή V8, με κάποιες αλλαγές που θα δούμε στην πορεία.

11.2 Λήψη και εγκατάσταση της Node.js

Για την παρακολούθηση αυτού του κεφαλαίου είναι απαραίτητη η λήψη και εγκατάσταση της Node.js στον υπολογιστή, μια σχετικά απλή διαδικασία που μπορούμε να ολοκληρώσουμε ακολουθώντας τις οδηγίες στον ιστότοπο της Node.js.

Η Node.js διατίθεται σε διάφορες εκδόσεις, που δημοσιεύονται 2 φορές το χρόνο, κάθε Απρίλιο (με ζυγό αριθμό έκδοσης: 2, 4, 6, 8…) και κάθε Οκτώβριο (με μονό αριθμό έκδοσης 1, 3, 5, 7…). Επιλέγουμε κατά προτίμηση την έκδοση του πιο πρόσφατου Απρίλη, δηλαδή την έκδοση με τον μεγαλύτερο άρτιο αριθμό. Οι εκδόσεις αυτές ονομάζονται LTS (Long-term support - υποστήριξη μεγάλης διάρκειας) και συντηρούνται για διάστημα 3 ετών όσον αφορά κρίσιμα θέματα ασφάλειας. Από τον Απρίλιο του 2022 η τρέχουσα έκδοση LTS είναι η 18. Για την ακρίβεια, η σύσταση είναι να επιλέγουμε την πιο πρόσφατη LTS αφού έχουν περάσει έξι μήνες από τη διάθεσή της (**Εικόνα *11.2***).



Εικόνα 11.2 *Οι εκδόσεις της Node.js που διατίθενται κάθε Απρίλιο υποστηρίζονται για τρία χρόνια (LTS) και έχουν ζυγό αριθμό. Αυτές που διατίθενται κάθε Οκτώβριο έχουν μονό αριθμό.*

Για τη χρήση της Node.js απαιτείται η εξοικείωση με βασικές λειτουργίες σε περιβάλλον γραμμής εντολών, που στα Windows είναι προσβάσιμο από την εφαρμογή PowerShell ή τη γραμμή εντολών και στους υπολογιστές Mac και Linux από το Terminal ή τερματικό.

Μαζί με τη Node.js θα εγκατασταθεί και το εργαλείο **npm** (node package manager) που βοηθά στη δημιουργία και τη διαχείριση του προγράμματός μας, το οποίο θα είναι ένα «πακέτο», καθώς και στην

εγκατάσταση πακέτων που έχουν δημιουργήσει άλλοι.

11.2.1 Μια απλή εφαρμογή σε Node.js

Έχοντας εγκαταστήσει τη Node.js, μπορούμε να γράψουμε και να εκτελέσουμε το πρώτο πρόγραμμά μας και να το αποθηκεύσουμε σε ένα αρχείο με όνομα, για παράδειγμα, index.js:

```
console.log('Καλή σας μέρα!`);
```

Για να τρέξει το πρόγραμμα αυτό αρκεί να εκτελέσουμε στο τερματικό την εντολή

```
node index.js
```

Προϋπόθεση βέβαια είναι το τερματικό μας να είναι στον φάκελο που έχουμε αποθηκεύσει το αρχείο index.js, κάτι που μπορούμε να εξασφαλίσουμε αν ανοίξουμε το τερματικό που είναι ενσωματωμένο στο Visual Studio Code.

11.2.2 Διαφορές με την JavaScript στον φυλλομετρητή

Η JavaScript που εκτελείται από τη Node.js είναι η ίδια με αυτή του φυλλομετρητή, ειδικά αν ο φυλλομετρητής μας είναι ο Goole Chrome ή χρησιμοποιεί τη V8, όπως ο Microsoft Edge, ο Opera κ.ά.

Ωστόσο, είναι σαφές πως το περιβάλλον εκτέλεσης της Node.js είναι τελείως διαφορετικό από τον φυλλομετρητή, όπου συνήθως χρησιμοποιούμε την JavaScript για να αλληλεπιδράσουμε με το DOM και για να χρησιμοποιήσουμε διάφορες ενσωματωμένες λειτουργίες όπως το Cookies API κ.λπ. Το DOM και τα διάφορα API προφανώς δεν είναι διαθέσιμα όταν εκτελείται η JavaScript έξω από τον φυλλομετρητή. Επιπλέον, δεν έχουμε στη διάθεσή μας ούτε τα αντίστοιχα αντικείμενα του DOM, όπως το document και το window.

Από την άλλη πλευρά, όπως θα δούμε (Ενότητα 11.3.1), η Node.js περιλαμβάνει μια σειρά από βιβλιοθήκες με τις οποίες μπορούμε να κάνουμε πράγματα που δεν είναι δυνατό να γίνουν στον φυλλομετρητή, όπως να επεξεργαστούμε αρχεία στον δίσκο (βιβλιοθήκη fs) ή να δημιουργήσουμε έναν εξυπηρετητή (βιβλιοθήκη http).

Επιπλέον, δεν γνωρίζουμε από πριν ποιον φυλλομετρητή θα χρησιμοποιήσει ο επισκέπτης της ιστοσελίδας μας, ούτε σε ποια έκδοση θα είναι αυτός. Αυτό σημαίνει πως δεν ξέρουμε ποια έκδοση ή παραλλαγή της JavaScript μπορεί να εκτελέσει ο φυλλομετρητής και ενδεχομένως χρειάζεται να πάρουμε ειδικά μέτρα για να το αντιμετωπίσουμε. Τη Node.js όμως την εγκαθιστούμε σε ένα σημείο, στον εξυπηρετητή, και έχουμε απόλυτη επίγνωση για την έκδοσή της και τις δυνατότητές της.

Ένα τέταρτο σημείο στο οποίο διαφέρουν, αν και αυτό τείνει να εξαλειφθεί, είναι το σύστημα φόρτωσης εξωτερικών βιβλιοθηκών (modules), που στον φυλλομετρητή είναι το ECMAScript 6, ενώ η Node.js υποστηρίζει δύο συστήματα, το παλιότερο CommonJS αλλά και το ES6 (Ενότητα 11.3).

Τέλος, υπάρχουν και κάποιες διαφορές στον βρόχο ελέγχου συμβάντων (Ενότητα 10.1.2).

11.2.3 Ένας απλός εξυπηρετητής σε Node.js

Αν και η Node.js έχει πολλά χρήσιμα χαρακτηριστικά που την κάνουν ένα ενδιαφέρον περιβάλλον προγραμματισμού, ο λόγος που κυρίως μας ενδιαφέρει είναι η χρήση της ως γλώσσα από την πλευρά του εξυπηρετητή. Όπως είδαμε στην ενότητα 1.3.1, η Node.js έχει έναν ενσωματωμένο εξυπηρετητή που μπορούμε να χρησιμοποιήσουμε άμεσα:

```
// Αρχείο app.js
const  http = require('http');

http.createServer(function (req, res) {
  res.write('Καλή σας μέρα!');
  res.end();
}).listen(8080); // ο εξυπηρετητής αποκρίνεται στη θύρα 8080
```

Εκτελούμε στο τερματικό ή τη γραμμή εντολών **$ node app.js** και, αν δεν υπάρξει κάποιο σφάλμα, μπορούμε να πλοηγηθούμε με τον φυλλομετρητή στη διεύθυνση http://127.0.0.1:8080, όπου θα δούμε το μήνυμα «Καλή σας μέρα!».

Στην πρώτη γραμμή του προγράμματός μας, με τη συνάρτηση **require()** φορτώνουμε τη βιβλιοθήκη (module) **http** που μας διαθέτει έναν απλό εξυπηρετητή για το πρωτόκολλο HTTP.

11.3 Βιβλιοθήκες και πακέτα στη Node.js

Έχουμε λοιπόν κατασκευάσει έναν απλό εξυπηρετητή και μπορούμε να τον χρησιμοποιήσουμε για να παρουσιάσουμε, αντί για το απλό μήνυμα «Καλή σας μέρα!», ολόκληρες σελίδες HTML, που παράγονται δυναμικά. Ένα τέτοιο εγχείρημα θα πάρει γρήγορα μεγάλο μέγεθος και θα χρειαστεί να χωρίσουμε το πρόγραμμα σε αυτοτελείς μονάδες ώστε να τις συντηρούμε πιο εύκολα. Στην JavaScript οι μονάδες αυτές ονομάζονται βιβλιοθήκες (modules) και συνήθως μια βιβλιοθήκη περιέχει κώδικα που εξυπηρετεί έναν συγκεκριμένο σκοπό. Μια βιβλιοθήκη είναι αποθηκευμένη σε ένα αρχείο.

11.3.1 Ενσωματωμένες βιβλιοθήκες

Ο μηχανισμός των modules μάς επιτρέπει να χρησιμοποιήσουμε βιβλιοθήκες με κώδικα τρίτων, επιταχύνοντας και διευκολύνοντας κατά πολύ την ανάπτυξη της εφαρμογής μας. Η Node.js έχει ενσωματωμένα έναν αριθμό από modules που μας βοηθούν να επιτελέσουμε μερικές πολύ χρήσιμες εργασίες. Για παράδειγμα, το module **http** που χρησιμοποιήσαμε στο προηγούμενο παράδειγμα είναι αυτό που μας παρέχει έναν απλό εξυπηρετητή HTTP. Μερικά ακόμη ενσωματωμένα modules είναι (η πλήρης λίστα στην τεκμηρίωση της Node.js):

- **https**: Υλοποιεί έναν εξυπηρετητή στο πρωτόκολλο HTTPS.
- **http2**: Υλοποιεί έναν εξυπηρετητή στο πρωτόκολλο HTTP2.
- **url**: Εργαλεία για το διάβασμα και τον χειρισμό URL.
- **process**: Πληροφορίες και εργαλεία για την τρέχουσα διεργασία της Node.js.
- **fs**: Δυνατότητες πρόσβασης σε αρχεία.
- **zlib**: Εργαλεία για τη συμπίεση και αποσυμπίεση αρχείων.
- **path**: Εργαλεία για τον χειρισμό διαδρομών και ονομάτων αρχείων στον δίσκο.
- **crypt**: Παρέχει κρυπτογραφικές λειτουργίες.
- **module**: Εργαλεία για την αλληλεπίδραση με τις βιβλιοθήκες.

Οι ενσωματωμένες βιβλιοθήκες δεν χρειάζονται εγκατάσταση, αλλά είναι διαθέσιμες απευθείας.

Για να δούμε τις διαθέσιμες ενσωματωμένες βιβλιοθήκες, μπορούμε να χρησιμοποιήσουμε τη βιβλιοθήκη module:

```
// Αρχείο modules.mjs
import * as module from 'module';

console.log(module.builtinModules)
```

Το αποτέλεσμα θα είναι η πλήρης λίστα των ενσωματωμένων βιβλιοθηκών.

11.3.2 Τοπικές βιβλιοθήκες

Τα ενσωματωμένα modules είναι πολύ χρήσιμα, αλλά πολλές φορές θα χρειαστεί να γράψουμε δικά μας, είτε για να τα χρησιμοποιήσουμε στην εφαρμογή μας είτε για να τα διαθέσουμε σε τρίτους.

11.3.3 Βιβλιοθήκες ECMAScript6 και CommonJS

Η Node.js υποστηρίζει δύο συστήματα για module που δεν είναι τελείως συμβατά μεταξύ τους. Ο λόγος είναι πως όταν κατασκευάστηκε η Node.js, η JavaScript δεν είχε ενσωματωμένο μηχανισμό στη γλώσσα που να υποστηρίζει module. Η Node.js χρησιμοποίησε λοιπόν τον μηχανισμό που βασίζεται στο πρότυπο CommonJS. Αυτός ο μηχανισμός είναι που φορτώνει ένα module με τη συνάρτηση `require()`.

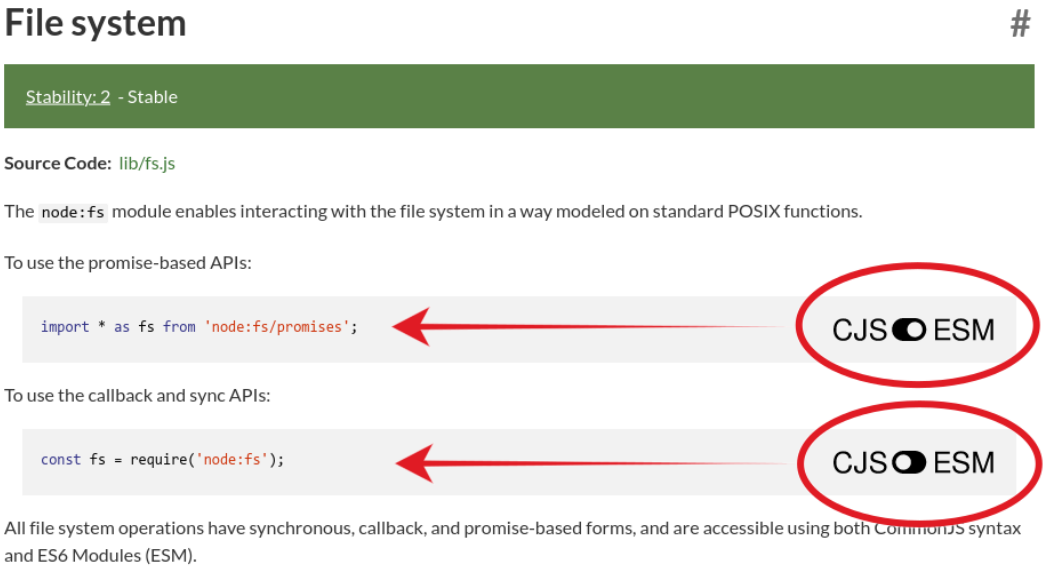
Το 2015, όμως, με την έκδοση ECMAScript 6 (ES6), η JavaScript απέκτησε ενσωματωμένο σύστημα module. Αυτός είναι ο μηχανισμός που ονομάζεται συνήθως ES6 modules και χρησιμοποιεί την οδηγία import για το φόρτωμα. Ο μηχανισμός είναι ο ίδιος που μπορούμε να χρησιμοποιήσουμε για να φορτώσουμε module στον φυλλομετρητή. Επιπλέον, με τη σύνταξη import της ES6 μπορούμε να φορτώσουμε βιβλιοθήκες που είναι γραμμένες σύμφωνα με το CommonJS. Αντίθετα, με τη σύνταξη `require()` του CommonJS δεν μπορούμε να φορτώσουμε βιβλιοθήκες γραμμένες σύμφωνα με το ES6.

Η Node.js χρησιμοποιεί σαν προκαθορισμένο σύστημα το CommonJS. Αν θέλουμε να χρησιμοποιήσουμε το σύστημα ES6, τότε θα πρέπει να αποθηκεύσουμε το αρχείο μας με κατάληξη **.mjs**, για παράδειγμα **index.mjs**:

```
// αρχείο με κατάληξη .mjs
import http from 'http';

http.createServer(function (req, res) {
  res.write('Καλή σας μέρα!');
  res.end();
}).listen(8080); // ο εξυπηρετητής αποκρίνεται στη θύρα 8080
```

Εναλλακτικά, μπορούμε να ορίσουμε πως θέλουμε στο πρότζεκτ μας να χρησιμοποιήσουμε το πρότυπο ES6 δηλώνοντάς το στο αρχείο package.json (Ενότητα 11.3.4).

Στην τεκμηρίωση που παρέχει η Node.js για τις ενσωματωμένες βιβλιοθήκες της μπορούμε να δούμε παραδείγματα και για τους δύο τρόπους (**Εικόνα *11.3***):



Εικόνα 11.3 *Στιγμιότυπο από την τεκμηρίωση των ενσωματωμένων βιβλιοθηκών της Node.js. Με τον διακόπτη μπορούμε να επιλέξουμε τον κώδικα για την εισαγωγή του module στο πρόγραμμά μας.*

Ο μηχανισμός ES6 παρουσιάζεται πιο αναλυτικά στη συνέχεια (Ενότητα 11.7). Είναι όμως αρκετά απλός ώστε να συνεχίσουμε με τη βασική χρήση της Node.js χωρίς να μπούμε ακόμα σε λεπτομέρειες.

Μια απλή τοπική βιβλιοθήκη

Ο παρακάτω κώδικας περιγράφει ένα ES6 module που περιέχει μια λίστα για ψώνια και τη συνάρτηση isInList() που μας απαντάει αν κάποιο προϊόν είναι ή όχι στη λίστα μας:

```
// Αρχείο groceryList.mjs
const items = ['μακαρόνια', 'ελαιόλαδο', 'σκόρδο']

class GroceryList {
    isInList(item) {
        return items.includes(item)
    }
};

export { GroceryList }
```

Για να χρησιμοποιήσουμε αυτό το module θα πρέπει να το κάνουμε import, για παράδειγμα:

```
// Αρχείο app.mjs
import {GroceryList}  from './groceryList.mjs';

const gList = new GroceryList();
console.log(gList.isInList("σκόρδο"));
```

Αν εκτελέσουμε το πρόγραμμά μας με **node app.mjs**, θα δούμε το αποτέλεσμα true στο τερματικό μας.

11.3.4 Πακέτα και το εργαλείο npm

Το τρίτο και ίσως πιο χρήσιμο είδος βιβλιοθήκης στη Node.js είναι τα πακέτα. Τα πακέτα δεν είναι τίποτε άλλο από μια ομάδα αρχείων σε έναν φάκελο. Στον αρχικό φάκελο κάθε πακέτου υπάρχει ένα αρχείο με όνομα **package.json** που περιέχει βασικές πληροφορίες και οδηγίες που αφορούν το πακέτο αυτό.

Ένα πακέτο που το φορτώνουμε με την import ή με τη require είναι και module. Είναι σημαντικό να τονίσουμε πως από εδώ και στο εξής κάθε πρόγραμμά μας σε Node.js θα είναι και ένα πακέτο.

Δημιουργία πακέτου

Ο πιο απλός τρόπος δημιουργίας αυτού του αρχείου είναι να χρησιμοποιήσουμε το εργαλείο **npm** (node package manager) που εγκαθίσταται μαζί με τη Node.js (Ενότητα 11.2). Στο τερματικό, στον φάκελο που θέλουμε να δημιουργήσουμε το πακέτο μας, γράφουμε:

```
npm init -y
```

Με τις παραμέτρους init και -y θα δημιουργηθεί ένα βασικό αρχείο package.json και θα συμπληρωθεί με κάποιες πληροφορίες.

Το αρχείο αυτό θα περιέχει τα εξής:

```
{
  "name": "node",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Το πιο σημαντικό πεδίο είναι το main, που προσδιορίζει ποιο θα είναι το αρχείο που περιέχει το σημείο εισόδου του πακέτου μας, τι θα φορτωθεί όταν κάποιος φορτώσει το πρόγραμμά μας σαν πακέτο στο δικό του πρόγραμμα. Στην πορεία, το package.json θα εμπλουτιστεί και με άλλα πεδία και τιμές που θα μας βοηθήσουν στην ανάπτυξη.

Ένα τέτοιο πεδίο είναι το **type**. Με το type μπορούμε να προσδιορίσουμε ποιο σύστημα module προτιμούμε. Αν δεν γράψουμε τίποτα, όπως στο παράδειγμα πιο πάνω, τότε το σύστημά μας θα είναι το CommonJS και θα πρέπει να έχουμε στα αρχεία μας κατάληξη .mjs αν θέλουμε να χρησιμοποιήσουμε βιβλιοθήκες τύπου ES6. Μπορούμε όμως να δώσουμε την τιμή module:

```
{
  ...
  "type": "module",
  ...
}
```

Με την τιμή “module” η Node.js θα χρησιμοποιήσει το σύστημα ES6 για να φορτώσει τα modules και μπορούμε

να τα ονομάσουμε και με τη συνήθη κατάληξη .js.

Αν δεν υπάρχει η ιδιότητα type ή αν έχει τιμή "commonjs", τότε θα θεωρηθεί πως ακολουθούμε το CommonJS.

11.3.5 Εγκατάσταση πακέτων από το npmjs.com

Ένα από τα πιο ελκυστικά χαρακτηριστικά της Node.js είναι η μεγάλη δημόσια βιβλιοθήκη πακέτων, που είναι διαθέσιμη από την ιστοσελίδα https://www.npmjs.com. Ενδεικτικά, στο αποθετήριο αυτό υπάρχουν περίπου 2.000.000 πακέτα (Μάιος 2022), από τα περίπου 1.600.000 που υπήρχαν τον Μάιο του 2021 και προστίθενται πάνω από 1.000 νέα πακέτα κάθε μέρα.

Αφού εντοπίσουμε ένα πακέτο που μας ενδιαφέρει να χρησιμοποιήσουμε στο δικό μας πακέτο, χρησιμοποιούμε το npm για να το εγκαταστήσουμε, για παράδειγμα η παρακάτω εντολή στο τερματικό θα εγκαταστήσει στον τρέχοντα φάκελο το πακέτο yodasay:

```
npm install yodasay
```

Το πακέτο θα αποθηκευτεί στον υποφάκελο node_modules στον τρέχοντα φάκελο. Καθώς το yodasay έχει εξαρτήσεις και σε άλλα πακέτα, το npm αναλαμβάνει να αποθηκεύσει και αυτά. Στο τέλος, στον υποφάκελο node_modules θα βρίσκεται το yodasay καθώς και όλα τα πακέτα που χρειάζεται για να λειτουργήσει, συνολικά 6 πακέτα.

Επιπλέον, στο αρχείο package.json θα προστεθεί η ιδιότητα dependencies που γράφει πως το δικό μας πακέτο εξαρτάται από το yodasay, στην έκδοση τουλάχιστον 1.1.9:

```
{
  ...
  "dependencies": {
    "yodasay": "^1.1.9"
  },
  ...
}
```

Από εδώ προκύπτει μια πολύ χρήσιμη λειτουργία, δηλαδή να μπορούμε να κατεβάσουμε όλες τις εξαρτήσεις μας (τα πακέτα που χρειάζεται το δικό μας πρόγραμμα) ακόμη και αν δεν υπάρχει ο φάκελος node_modules. Με την εντολή

```
npm install
```

θα εγκατασταθούν όλες οι εξαρτήσεις που περιγράφονται στο package.json.

Το yodasay είναι ένα πακέτο που ζωγραφίζει στο τερματικό τον Yoda, χαρακτήρα των ταινιών Star Wars, που λέει ό,τι του ζητήσουμε, για παράδειγμα:

```
//αρχείο με όνομα my_yoda.mjs
import yodasay from 'yodasay';

console.log(yodasay.say({
  text: 'Use yodasay, you will.'
}));
```

Αν εκτελέσουμε στο τερματικό **node my_yoda.mjs** θα εμφανίσει το εξής:

```
 **
< Use yodasay, you will. >
 ------------------------
      \
       \
          .--.
  \`--._,'.::.`._.--'/
    .   ` **::** '   .
      -:.`'..`'.:-
```

```
\ `--' /
  ----
```

Άσκηση

Δημιουργήστε ένα νέο πακέτο και χρησιμοποιήστε το πακέτο cowsay για να γράψετε μια μικρή εφαρμογή. Χρησιμοποιήστε το cowsay για να εμφανίσετε στην οθόνη τη λίστα με τα ψώνια των προηγούμενων παραδειγμάτων.

Απεγκατάσταση πακέτων

Μπορούμε να απεγκαταστήσουμε ένα πακέτο απλά σβήνοντάς το από τον υποφάκελο node_modules. Άλλος ένας τρόπος, που σβήνει και όλες τις εξαρτήσεις του πακέτου που απομακρύνουμε, είναι, π.χ., να τρέξουμε npm uninstall --no-save yodasay.

Και στις δύο περιπτώσεις στο package.json θα συνεχίσει να υπάρχει αναφορά στο πακέτο. Στην επόμενη εκτέλεση της npm install θα γίνει πάλι εγκατάσταση.

Για να απομακρύνουμε τελείως ένα πακέτο και οτιδήποτε άλλο εγκαταστάθηκε από το npm για λογαριασμό του, απλά εκτελούμε:

```
npm uninstall package
```

11.3.6 Αρίθμηση εκδόσεων κατά SemVer

Θα παρατηρήσουμε πως η αρίθμηση των εκδόσεων αποτελείται από τρεις ακέραιους αριθμούς και αυτό ισχύει τόσο για τα πακέτα που κατεβάζουμε με το npm όσο και για την έκδοση που έδωσε το npm στο δικό μας πακέτο όταν τρέξαμε την εντολή npm init -y.

Εικόνα 11.4 *Η αρίθμηση κατά SemVer χρησιμοποιεί ακεραίους σε τρεις θέσεις για να σηματοδοτήσει το είδος των αλλαγών μεταξύ εκδόσεων.*

Ο λόγος είναι πως η Node.js ακολουθεί τη σύμβαση Semantic Versioning ή SemVer (**Εικόνα *11.4***). Στο SemVer οι τρεις ακέραιοι σηματοδοτούν αλλαγές ανάλογα με το εύρος τους:

- **Μεγάλη** (Major) αλλαγή: Ο πρώτος ακέραιος αυξάνει κατά ένα όταν η νέα μας έκδοση εισάγει μια αλλαγή που κάνει το πακέτο μας μη συμβατό με προηγούμενες εκδόσεις.
- **Μικρή** (Minor) αλλαγή: Ο δεύτερος ακέραιος αυξάνει κατά ένα όταν η νέα μας έκδοση εισάγει μια νέα λειτουργία, αλλά το πακέτο μας είναι συμβατό με προηγούμενες εκδόσεις, δηλαδή κάποιος που χρησιμοποιούσε την προηγούμενη έκδοση, μπορεί να χρησιμοποιήσει και τη νέα χωρίς να χρειαστεί να κάνει αλλαγές στον δικό του κώδικα.
- **Διόρθωση** (Patch): Ο τρίτος ακέραιος αυξάνει όταν η νέα έκδοση του πακέτου μας δεν εισάγει αλλαγή στη λειτουργικότητα, αλλά διορθώνει ένα σφάλμα της προηγούμενης έκδοσης.

Το SemVer είναι μια αρκετά δημοφιλής και απλή σύμβαση και η υιοθέτησή της έχει κάποια πλεονεκτήματα. Μπορούμε να καταλάβουμε με μια ματιά τι είδους αλλαγή συνέβη από τη μια έκδοση στην άλλη και να προσδιορίσουμε ποιες εκδόσεις πακέτων αποδεχόμαστε.

Αν δούμε ξανά την εγγραφή στο package.json για την εξάρτηση στο yodasay, θα παρατηρήσουμε πως γράφει

```
"yodasay": "^1.1.9"
```

Το σύμβολο ^ απλά λέει πως αποδεχόμαστε οποιαδήποτε έκδοση του yodasay από την 1.1.9 και πάνω, αρκεί να μην αυξηθεί ο πρώτος ακέραιος, άρα δεν επιτρέπουμε εκδόσεις με μεγάλες αλλαγές.

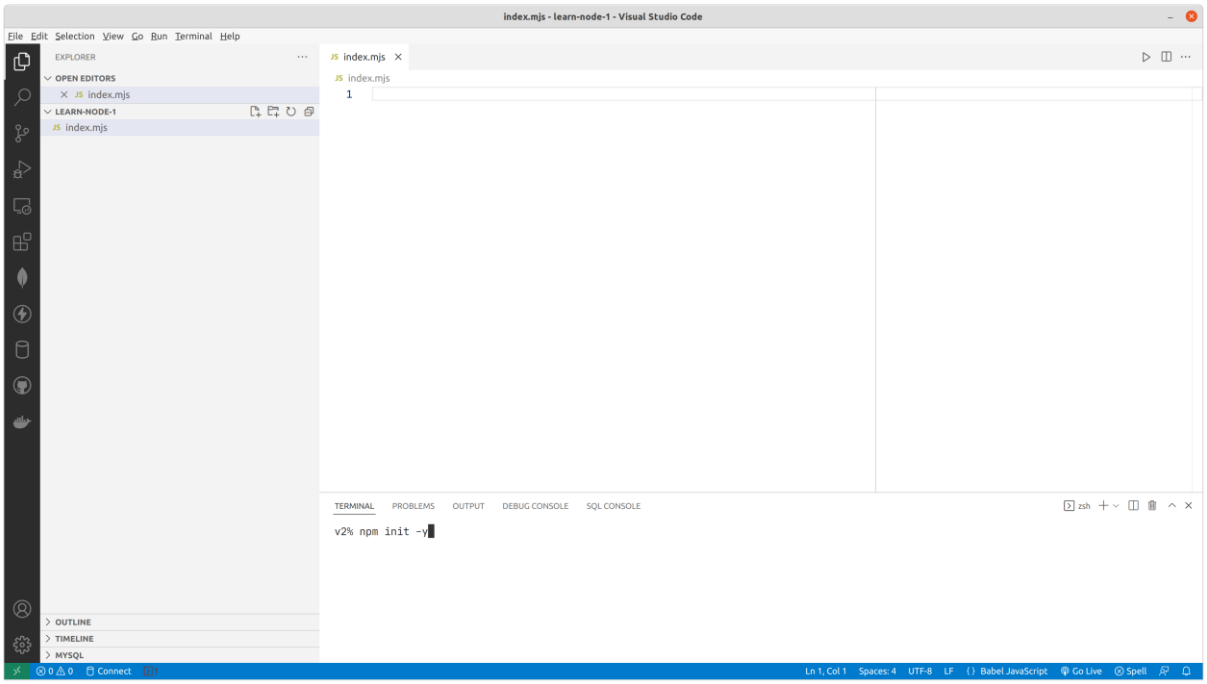
Το εργαλείο npm semver calculator μάς βοηθά αν θέλουμε να διατυπώσουμε πιο λεπτομερείς απαιτήσεις όσον αφορά την έκδοση των πακέτων που αποδεχόμαστε.

11.4 Βασικές εργασίες με αρχεία

Πριν χρησιμοποιήσουμε τη Node.js για τη δημιουργία εφαρμογών διαδικτύου, είναι σκόπιμο να εξοικειωθούμε μαζί της με πιο απλά παραδείγματα, χρησιμοποιώντας την ενσωματωμένη βιβλιοθήκη **fs**.

Η βιβλιοθήκη **fs** μας επιτρέπει να κάνουμε βασικές εργασίες με αρχεία, να ανοίξουμε και να διαβάσουμε ένα αρχείο, να γράψουμε νέο περιεχόμενο κ.λπ.

Αν και δεν είναι αυστηρά απαραίτητο, ωστόσο ακόμη και για απλά παραδείγματα ενδείκνυται να χρησιμοποιήσουμε το npm για να δημιουργήσουμε ένα πακέτο. Σε έναν νέο φάκελο λοιπόν, που μπορούμε να ονομάσουμε learn-node-1, ανοίγουμε το Visual Studio Code. Από το VS Code επιλέγουμε File->Open Folder και ανοίγουμε τον φάκελο που θα περιέχει το πακέτο μας. Στη συνέχεια, στο τερματικό του VS Code πληκτρολογούμε **npm init -y** (**Εικόνα 11.5**) και δημιουργείται το αρχείο package.json.



Εικόνα 11.5 *Το τερματικό είναι προσβάσιμο από το μενού Terminal->New Terminal.*

11.4.1 Εγγραφή σε αρχείο

Η βιβλιοθήκη fs παρέχει πολλές χρήσιμες λειτουργίες για τον χειρισμό αρχείων. Μπορούμε να την ενσωματώσουμε είτε με το σύστημα CommonJS

```
const fs = require('fs')
```

είτε με το σύστημα ES6 modules, που θα προτιμήσουμε σε αυτό το βιβλίο. Η βιβλιοθήκη fs μπορεί να λειτουργήσει με τρεις τρόπους:

- **Σύγχρονα**, όπου η κάθε εντολή εκτελείται μόνο αφού τελειώσει η προηγούμενη. Σε αυτή την περίπτωση, αν για παράδειγμα έχουμε να ανοίξουμε ένα πολύ μεγάλο αρχείο θα έχουμε καθυστερήσεις.

Όλες οι υπόλοιπες εντολές, που δεν εξαρτώνται από το αποτέλεσμα του ανοίγματος του αρχείου, θα περιμένουν να τελειώσει αυτό για να εκτελεστούν.

- **Ασύγχρονα**, όπου μπορούμε να εκτελέσουμε στο παρασκήνιο την ενέργεια που μπορεί να μπλοκάρει την εκτέλεση του προγράμματός μας, ώστε να εκτελεστεί *ασύγχρονα*. Η βιβλιοθήκη fs μπορεί να λειτουργήσει και με τους δύο μηχανισμούς που είδαμε στο προηγούμενο κεφάλαιο πως παρέχει η JavaScript για την ασύγχρονη εκτέλεση κώδικα:
  - τη **συνάρτηση επιστροφής** και
  - τις **υποσχέσεις**.

Σύγχρονα

Με την πιο απλή σε χρήση, σύγχρονη εκδοχή, όταν η Node.js ζητήσει πρόσβαση στο αρχείο από το λειτουργικό σύστημα, θα περιμένει μέχρι να τερματίσει αυτή η ενέργεια. Μόνο όταν τελειώσει η πρόσβαση και η Node.js πάρει το αποτέλεσμα από το λειτουργικό σύστημα θα συνεχίσει η εκτέλεση των επόμενων εντολών.

```
//αρχείο write_sync.mjs
import *  as fs from 'fs'

console.log("Αρχή")
try {
    const fd = fs.openSync('shopping-list.txt', 'a+')
    fs.writeSync(fd, "Μπανάνες\n");
    fs.closeSync(fd);
}
catch (err) {
    if (err.code == 'EACCES')
        console.error("Δεν έχω πρόσβαση")
    else
        throw err
}
//Ανάγνωση των περιεχομένων του αρχείου
try {
    const data = fs.readFileSync('shopping-list.txt', 'utf-8');
    console.log('Δεδομένα: ', data.trim());
}
catch(err) {
    throw err
}
console.log("Τέλος")
```

Στο παράδειγμα αυτό χρησιμοποιείται η readFileSync(), που διαβάζει το περιεχόμενο ολόκληρου του αρχείου σύγχρονα. Αν το αρχείο είναι μεγάλο, μπορεί, εκτός από το μπλοκάρισμα του event loop, να προκύψει και πρόβλημα με τη διαθέσιμη μνήμη, καθώς η συνάρτηση αποπειράται να φορτώσει όλα τα περιεχόμενα του αρχείου στη μνήμη.

Με συναρτήσεις επιστροφής

Με τη χρήση συναρτήσεων επιστροφής μπορούμε να εκτελέσουμε τις ενέργειές μας ασύγχρονα, χωρίς να μπλοκάρουμε το event loop. Όταν ολοκληρωθούν, θα κληθεί η συνάρτηση επιστροφής. Το ίδιο προηγούμενο παράδειγμα μπορούμε να το γράψουμε με συναρτήσεις επιστροφής.

```
//αρχείο write_cb.mjs
import *  as fs from 'fs'

console.log("Αρχή")
fs.open('shopping-list.txt', 'a+', (err, fd) => {
    if (err)
        throw err
    else {
```

```
            fs.write(fd, "Μπανάνες\n", (err, written, string) => {
                if (err)
                    throw err
                else
                    fs.readFile('shopping-list.txt', 'utf-8', (err, data) => {
                        if (err)
                            throw err
                        else
                            console.log('Δεδομένα: ', data.trim());
                        });
            })
        }
    })
    console.log("Τέλος")
```

Αυτό το μικρό πρόγραμμα χρησιμοποιεί συναρτήσεις επιστροφής που είδαμε σε προηγούμενη ενότητα (Ενότητα 10.1.2). Η συνάρτηση fs.open(), λοιπόν, καλείται με τρία ορίσματα, **όνομα αρχείου**, **flag** και μια **συνάρτηση επιστροφής**. Με το δεύτερο όρισμα, το **flag**, ορίζουμε τι μπορεί να γίνει με το αρχείο, για παράδειγμα:

- **a+**: Ανοίγει το αρχείο για συμπλήρωση και ανάγνωση. Το + σημαίνει πως αν δεν υπάρχει το αρχείο θα δημιουργηθεί.

Το fs δίνει πολλές άλλες επιλογές για το flag, για παράδειγμα:

- **r**: Ανοίγει για ανάγνωση, αν όμως δεν υπάρχει θα προκύψει εξαίρεση.
- **w+**: Ανοίγει για ανάγνωση και εγγραφή. Το αρχείο, αν δεν υπάρχει, θα δημιουργηθεί. Αν υπάρχει, τότε θα σβηστούν τα περιεχόμενά του.

Η τρίτη παράμετρος, η συνάρτηση επιστροφής, θα κληθεί από τη fs.open(). Η `open()` θα περάσει δύο παραμέτρους στη συνάρτηση επιστροφής. Η πρώτη θα έχει κάποια τιμή αν υπήρξε σφάλμα κατά το άνοιγμα, η δεύτερη είναι ένας **δείκτης προς το αρχείο** που άνοιξε η `open()`, που συνήθως ονομάζεται file descriptor.

Στο σώμα της συνάρτησης επιστροφής ελέγχουμε αν έχει προκύψει κάποιο σφάλμα και, στο παράδειγμα, το χειριζόμαστε υποτυπωδώς, καθώς απλά ρίχνουμε ξανά το σφάλμα με τη throw. Για παράδειγμα, θα μπορούσαμε να μην έχουμε δικαιώματα εγγραφής στον δίσκο ή στο συγκεκριμένο αρχείο.

Αν όλα πήγαν καλά και το αντικείμενο err που πέρασε η `open()` στη συνάρτηση επιστροφής έχει τιμή null, θα συνεχίσει η εκτέλεσή της. Σε αυτή την περίπτωση θα κληθεί η συνάρτηση fs.write().

Η `write()` καλείται και αυτή με τρία ορίσματα, τον **δείκτη στο αρχείο**, το **αλφαριθμητικό** που θέλουμε να γράψουμε και μια **συνάρτηση επιστροφής**. Στο απλό μας παράδειγμα χρησιμοποιούμε τη συνάρτηση επιστροφής για την περίπτωση που υπάρξει κάποιο σφάλμα κατά την εγγραφή, κατ' αναλογία με αυτό που κάναμε στη συνάρτηση επιστροφής της open(). Οι δύο επιπλέον παράμετροι που ορίζουμε στη συνάρτηση επιστροφής (**written** και **string**) περιέχουν το πλήθος των bytes που χρειάστηκαν για το αλφαριθμητικό που γράφτηκε.

Με Promises

Η τρίτη εκδοχή του παραδείγματός μας χρησιμοποιεί το Promise API, που είδαμε πιο αναλυτικά στην προηγούμενη ενότητα, όπως φαίνεται από τη γραμμή του import. Στη συνέχεια, καλούμε την writeFile(), που αυτή τη φορά επιστρέφει ένα promise, και χρησιμοποιούμε μια αλυσίδα καταναλωτών (Ενότητα 10.4.2). Στο επόμενο βήμα διαβάζουμε από το αρχείο με τη readFile() που και αυτή επιστρέφει ένα promise και, όταν επιλυθεί και αυτή, τυπώνουμε το αποτέλεσμα, που περιέχεται στη μεταβλητή data.

```
//αρχείο write_promise.mjs
import *  as fs from 'fs/promises'

console.log("Αρχή")
fs.writeFile('shopping-list.txt', "Αχλάδια\n", { flag: 'a+' })
    .then(() => fs.readFile('shopping-list.txt', 'utf-8'))
    .then((data) => {
        console.log("Δεδομένα: ", data)
    })
```

```
    .catch(err => {
        console.error("Παρουσιάστηκε σφάλμα: ", err)
    });

console.log("Τέλος")
```

Με async/await

Η εκδοχή με τις συναρτήσεις επιστροφής οδηγεί σε κώδικα που μπορεί να είναι δυσνόητος και, μάλιστα, όπως είδαμε, αυτή η κατάσταση είναι τόσο συχνή που έχει όνομα, η «κόλαση των callback». Η εισαγωγή του Promise API είναι μια μεγάλη βελτίωση στη συγγραφή ασύγχρονου κώδικα, ωστόσο πολλοί χρήστες (προγραμματιστές) δεν τη βρίσκουν ιδιαίτερα ευχάριστη.

Υπάρχει λοιπόν μια εναλλακτική σύνταξη για το Promise API, που είναι πιο εύχρηστη, η σύνταξη "async/await". Ας δούμε πρώτα το παράδειγμα εγγραφής και ανάγνωσης σε αρχείο, πριν την εξηγήσουμε.

```
//αρχείο write_async_await.mjs
import *  as fs from 'fs/promises`

try {
    await fs.writeFile('./shopping-list.txt', 'Μήλα\n', { flag: 'a+' })
    const data = await fs.readFile('shopping-list.txt', 'utf-8')
    console.log("Δεδομένα: ", data)
}
catch (err) {
    throw err
}

console.log("Τέλος")
```

Είναι αμέσως εμφανές πως ο κώδικάς μας είναι πιο ευανάγνωστος με τη σύνταξη async/await. Με τη λέξη-κλειδί **await** μπροστά από μια εντολή δηλώνουμε πως αυτή θα εκτελεστεί ασύγχρονα. Αυτό σημαίνει πως η readFile() θα εκτελεστεί μόνο αφού τελειώσει η writeFile(). Ωστόσο, σε αντίθεση με τη σύγχρονη εκτέλεση που είδαμε στο πρώτο παράδειγμα, το event loop δεν θα μπλοκαριστεί για όσο χρόνο χρειάζεται να εκτελεστεί η writeFile(). Η μεταβλητή **data** θα πάρει τιμή μόνο όταν ολοκληρωθεί (fulfill, στην ορολογία του Promise API) η ασύγχρονη readFile(). Όλο το μπλοκ κώδικα είναι μέσα σε μια δομή **try/catch** (Ενότητα 8.6.4), που κάνει και τη διαχείριση σφαλμάτων αρκετά πιο εύκολη.

11.5 Async/await

Η σύνταξη async/await είναι ένας εναλλακτικός τρόπος χρήσης του Promise API και κάνει τη συγγραφή ασύγχρονου κώδικα αισθητά πιο απλή. Ο βασικός μηχανισμός όμως συνεχίζει να είναι αυτός που παρέχει το Promise API (Ενότητα 10.4).

11.5.1 Η λέξη-κλειδί async

Η λέξη-κλειδί **async** μπροστά από μια συνάρτηση δηλώνει πως αυτή επιστρέφει ένα promise. Για να δούμε τη χρήση της θα ξαναγράψουμε το παράδειγμα χρήσης καταναλωτών από την ενότητα 10.4.2:

```
const p = new Promise((resolve, reject) => {
    //συναρτήσεις που ενεργοποιούνται σε περίπτωση επιτυχούς ή όχι
ολοκλήρωσης
    if (Math.random() > 0.5) { //προσομοίωση τυχαιότητας λειτουργίας
        resolve('Επιτυχής ολοκλήρωση')
    }
    else {
        reject('Σφάλμα')
    }
})
```

```
async function f() {
    return p
}

f().then(result => console.log(result),
    error => console.error(error))
```

Η συνάρτησή μας f() έχει σημειωθεί με τη λέξη-κλειδί **async**, καθώς επιστρέφει ένα promise.

Θα μπορούσαμε ωστόσο να μην επιστρέψουμε ένα promise, αλλά κατευθείαν μια τιμή:

```
async function f() {
    if (Math.random() > 0.5) { //προσομοίωση τυχαιότητας λειτουργίας
        return ('Επιτυχής ολοκλήρωση')
    }
    else {
        throw Error("Σφάλμα")
    }
}

f().then(result => console.log(result))
     .catch(error => console.error(error.message))
```

Σε αυτή την περίπτωση, λοιπόν, η συνάρτησή μας δεν επιστρέφει ρητά ένα promise. Η async, παρ' όλ' αυτά, αναλαμβάνει να τοποθετήσει το αποτέλεσμα σε ένα promise και μας επιστρέφει αυτό.

11.5.2 Η λέξη-κλειδί await

Η async, ωστόσο, χρησιμοποιείται σε συνδυασμό με τη λέξη-κλειδί **await**. Με την **await** η JavaScript περιμένει να ολοκληρωθεί η υπόσχεση ώστε να συνεχίσει την εκτέλεση. Πλέον, το παράδειγμά μας μπορεί να γραφτεί ως εξής:

```
async function f() {
    if (Math.random() > 0.5) { //προσομοίωση τυχαιότητας λειτουργίας
        return ('Επιτυχής ολοκλήρωση')
    }
    else {
        throw Error("Σφάλμα")
    }
}

try {
    // καλείται η f() και περιμένουμε μέχρι να επιλυθεί η υπόσχεση:
    const result = await f()
    console.log(result)
}
catch(error) {
    console.log(error.message)
}
```

Όταν εκτελείται η γραμμή **const** result = **await** f(), η JavaScript δεν εκτελεί την επόμενη γραμμή, αλλά περιμένει μέχρι να επιλυθεί η υπόσχεση που επιστρέφει η f(). Η αναμονή αυτή όμως δεν μπλοκάρει τον βρόχο ελέγχου συμβάντων, το event loop.

Η σύνταξη αυτή είναι σαφώς πιο εύκολη στην ανάγνωση από τη σύνταξη του Promise API, όπως γίνεται εμφανές αν γράψουμε ένα κάπως πιο σύνθετο παράδειγμα, όπως αυτό της ενότητας 10.4.3, χρησιμοποιώντας async/await:

```
console.log('αρχή προγράμματος');

// Καθώς η setTimeout δεν είναι συνάρτηση async (δεν επιστρέφει promise),
```

```
// για να πάρουμε το αποτέλεσμα μετά από 1000 ms είτε θα καλέσουμε μια
συνάρτηση
// επιστροφής είτε θα χρησιμοποιήσουμε promise:
function sleep() {
  return new Promise((resolve) => {
    setTimeout(() => resolve(10), 1000)
  })
}

// Τώρα μπορούμε να περιμένουμε μέχρι να ολοκληρωθεί η υπόσχεση
let result = await sleep();

// Υλοποιούμε τρεις φορές τη συνάρτηση με ελαφρώς διαφορετική σύνταξη
// Πρακτικά, οι τρεις συναρτήσεις κάνουν το ίδιο: διπλασιάζουν την τιμή
// εισόδου
async function function1(aValue) {
  console.log('async-1: ', aValue);
  return aValue * 2;
}

const function2 = async aValue => {
  console.log('async-2: ', aValue);
  return aValue * 2;
}

const function3 = async function (aValue) {
  console.log('async-3: ', aValue);
  return aValue * 2;
}

let res1 = await function1(result);
let res2 = await function2(res1);
await function3(res2);
console.log('τέλος προγράμματος');
```

Όπως βλέπουμε, η εκτέλεση των ασύγχρονων συναρτήσεων είναι εύκολα κατανοητή. Οι τρεις ασύγχρονες συναρτήσεις κάνουν τον ίδιο υπολογισμό, διπλασιάζουν την είσοδό τους, με ελαφρώς διαφορετική σύνταξη. Το αποτέλεσμα από την εκτέλεση του προγράμματος θα είναι:

```
αρχή προγράμματος
await-1:  10
await-2:  20
await-3:  40
τέλος προγράμματος
```

Η χρήση της await στον κώδικα

Σε παλιότερες εκδόσεις της Node.js αλλά και σε παλιότερους φυλλομετρητές η χρήση της await δεν επιτρεπόταν αν δεν ήταν μέσα σε μια συνάρτηση async. Για παράδειγμα, ο παρακάτω κώδικας δεν μπορούσε να εκτελεστεί:

```
let aValue = await someAsyncFunction();
let bValue = await anotherAsyncFunction(aValue);
```

Αυτός ο περιορισμός ξεπερνιόταν χρησιμοποιώντας μια σύνταξη όπου η συνάρτηση εκτελείται την ώρα που ορίζεται. Η σύνταξη αυτή ονομάζεται Immediately Invoked Function Expression ή άμεσα καλούμενη συναρτησιακή έκφραση:

```
(async () => {
    let aValue = await someAsyncFunction();
```

```
    let bValue = await anotherAsyncFunction(aValue);
})();
```

Στη Node.js επιτρέπεται πλέον η χρήση της await στο ανώτερο επίπεδο, έξω από συνάρτηση async, ακριβώς όπως στα προηγούμενα παραδείγματα. Για να γίνει αυτό, όμως, θα πρέπει ο κώδικάς μας να είναι σε μορφή ECMAScript6, δηλαδή

- να είναι αποθηκευμένος σε αρχείο με κατάληξη **.mjs** ή
- να έχει οριστεί ο τύπος του πακέτου μας στο αρχείο package.json σε module, όπως περιγράφεται στην ενότητα 11.3.4.

Top-level await στον φυλλομετρητή

Αντίστοιχα, στον φυλλομετρητή η χρήση των await στο ανώτερο επίπεδο επιτρέπεται αν φορτώσουμε τον κώδικά μας σαν ECMAScript6 module, χρησιμοποιώντας το γνώρισμα **type="module**, όπως στο παρακάτω παράδειγμα που τυπώνει τον πληθυσμό της Ελλάδας στην κονσόλα του φυλλομετρητή.

```
<script type="module">
    let response = await
fetch("https://restcountries.com/v3.1/name/greece")
    let responseJson = await response.json()

    console.log(responseJson[0].population)
</script>
```

11.5.3 Χειρισμός σφαλμάτων

Συνοψίζοντας, η χρήση της λέξης **async** μπροστά από μια συνάρτηση έχει σαν αποτέλεσμα η συνάρτηση να επιστρέφει πάντα μια υπόσχεση. Με τη λέξη **await** μπροστά από μια υπόσχεση, η JavaScript θα περιμένει μέχρι να υλοποιηθεί αυτή.

Αν κατά την υλοποίηση της υπόσχεσης συμβεί κάποιο σφάλμα, μπορούμε να το χειριστούμε με τον συνήθη τρόπο του Promise API. Όπως είδαμε, το async/await είναι απλά μια εναλλακτική σύνταξη για να γράφουμε ασύγχρονο κώδικα με το Promise API. Συνεπώς, ο χειρισμός σφαλμάτων μπορεί να γίνει με τον ειδικό καταναλωτή .catch(), όπως ακριβώς και στις υποσχέσεις.

Είναι όμως συνήθως πιο βολικό να χειριστούμε το σφάλμα χρησιμοποιώντας τη δομή **try/catch** (+@#sec:try_catch). Για παράδειγμα, έστω μια συνάρτηση async που υπό προϋποθέσεις μπορεί να εγείρει ένα νέο σφάλμα:

```
async function myDivision(a, b) {
  if (b==0)
      throw new Error("Δεν μπορεί να γίνει διαίρεση με το 0")
  return a/b
}
```

Ο χειρισμός μπορεί να γίνει με την try/catch:

```
try {
  console.log(await myDivision(2, 1))
}
catch(error) {
  console.log("Συνέβη σφάλμα", error.message)
}
```

ή με τον καταναλωτή catch():

```
myDivision(2, 0)
    .then( result => console.log(result))
    .catch( error => console.log(error))
```

11.6 Η Node.js ως εξυπηρετητής διαδικτύου

Μέχρι τώρα έχουμε συζητήσει μερικά βασικά χαρακτηριστικά του περιβάλλοντος που παρέχει η Node.js. Έχουμε δει τα πακέτα και τις βιβλιοθήκες, τον ασύγχρονο προγραμματισμό με το παράδειγμα της βασικής βιβλιοθήκης **fs** και χρησιμοποιήσαμε την ευκαιρία για να συμπληρώσουμε την εικόνα του Promise API με τη βολικότερη σύνταξη async/await.

Στη συνέχεια θα εστιάσουμε στη χρήση της Node.js ως περιβάλλον ανάπτυξης εφαρμογών διαδικτύου από την πλευρά του εξυπηρετητή.

Στην αρχή του κεφαλαίου είδαμε ήδη έναν απλό εξυπηρετητή Node.js, που μπορούμε να ξεκινήσουμε με node <όνομα αρχείου.mjs> από το τερματικό:

```
// αρχείο με κατάληξη .mjs
import http from 'http';

http.createServer(function (req, res) {
  res.write('Καλή σας μέρα!');
  res.end();
}).listen(8080); // ο εξυπηρετητής αποκρίνεται στη θύρα 8080
```

Κάθε φορά που θα φτάνει ένα αίτημα στον εξυπηρετητή μας (π.χ. αν ανοίξουμε τον φυλλομετρητή και φορτώσουμε τη σελίδα http://127.0.0.1:8080) θα εκτελείται η συνάρτηση που περάσαμε σαν όρισμα στην createServer(). Η συνάρτηση αυτή είναι η συνάρτηση χειρισμού αιτημάτων (request handler). Η Node.js, όταν φτάσει ένα αίτημα στον εξυπηρετητή μας, καλεί τον χειριστή αιτημάτων με δύο ορίσματα, το αντικείμενο του αιτήματος HTTP και το αντικείμενο της απάντησης HTTP. Συνηθίζεται να ονομάζουμε τις παραμέτρους αυτές req και res ή request και response αντίστοιχα.

Κάθε αίτημα λοιπόν συνοδεύεται από το **request object**. Με βάση τις πληροφορίες που περιέχονται στο αίτημα, ο εξυπηρετητής μπορεί να αποφασίσει τι θα επιστρέψει σαν απάντηση. Όπως έχουμε δει (Ενότητα 1.4.3), ένα αίτημα χαρακτηρίζεται από τη μέθοδο, τη διαδρομή του πόρου που ζητήθηκε, την έκδοση του πρωτοκόλλου, τις κεφαλίδες και το σώμα του μηνύματος.

Μπορούμε να ανακτήσουμε όλες αυτές τις πληροφορίες μέσω του αντικείμενου αιτήματος:

```
// αρχείο server.mjs
import http from 'http';

http.createServer(function (req, res) {
    console.log("Μέθοδος: ", req.method)
    console.log("Διαδρομή: ", req.url)
    console.log("Πρωτόκολλο: ", req.httpVersion)
    console.log("Κεφαλίδες: ", req.headers)
    console.log("Σώμα: ", req.body)
    res.write('Καλή σας μέρα!');
    res.end();
}).listen(8080); // ο εξυπηρετητής αποκρίνεται στη θύρα 8080
```

Συνήθως, τα αιτήματα με τη μέθοδο **GET** συνοδεύονται και από μια λίστα παραμέτρων. Για παράδειγμα, ένα αίτημα προς τη μηχανή αναζήτησης Google έχει τη μορφή https://www.google.com/search?q=καιρός, όπου η παράμετρος q έχει την τιμή καιρός. Αν έχουμε περισσότερες παραμέτρους, αυτές χωρίζονται με τον χαρακτήρα **&**:

```
var1=value1&var2=value2&var3=value3
```

Ο εξυπηρετητής διαμορφώνει και την απάντηση που θα δώσει στο αίτημα με βάση τις παραμέτρους του αιτήματος. Η βιβλιοθήκη http δεν παρέχει βοηθήματα για την ανάγνωση των παραμέτρων, αλλά μπορούμε να καταφύγουμε στην επίσης ενσωματωμένη στη Node.js βιβλιοθήκη url. Εδώ μπορεί να δημιουργηθεί μια μικρή σύγχυση, καθώς η βιβλιοθήκη url περιλαμβάνει το αντικείμενο URL, το οποίο μας ενδιαφέρει. Το αντικείμενο αυτό είναι διαθέσιμο στο global object, οπότε δεν χρειάζεται να το φορτώσουμε με import.

Αρχικά, κατασκευάζουμε ένα νέο στιγμιότυπο του URL με βάση τη διαδρομή του αιτήματός μας, που αποτελείται από το πρωτόκολλο του αιτήματος, που χάριν απλότητας το θέτουμε σε http, το όνομα του

εξυπηρετητή μας (στον οποίο απευθύνθηκε το αίτημα) και τη διαδρομή του αιτήματος:

```
const myURL = new URL("http://" + req.headers.host + req.url)
```

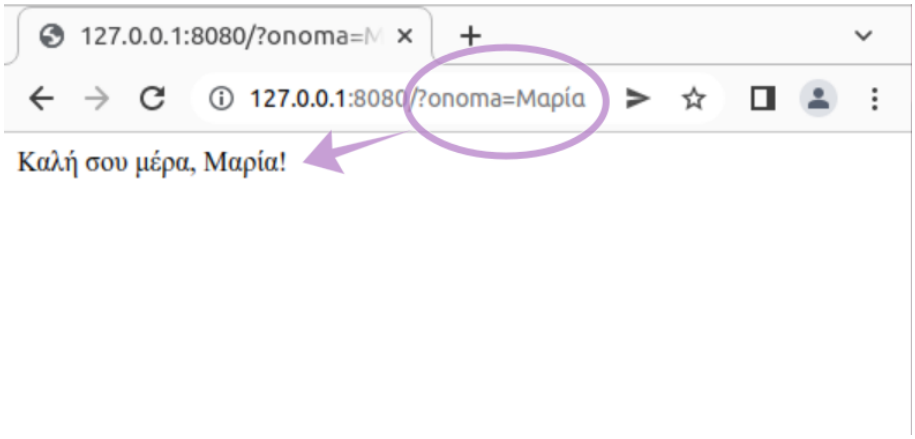
Έχοντας κατασκευάσει έτσι ένα αντικείμενο URL, μπορούμε να ανακτήσουμε τις τιμές των παραμέτρων που μας ενδιαφέρουν με την κλάση URL.searchParams και τη συνάρτησή της get():

```
const name = myURL.searchParams.get('onoma')
```

Έτσι, αν το αίτημα ήταν στο URL http://127.0.0.1:8080/?onoma=Μαρία, η μεταβλητή name θα έχει την τιμή «Μαρία». Πριν στείλουμε την απάντησή μας στον φυλλομετρητή, λόγω των ελληνικών, θα πρέπει να ειδοποιήσουμε τον φυλλομετρητή πως η απάντηση που στέλνουμε είναι κωδικοποιημένη σε UTF-8, ορίζοντας την κατάλληλη κεφαλίδα στην απάντηση που θα στείλουμε.

```
// αρχείο server.mjs
import http from 'http';

http.createServer(function (req, res) {
    const myURL = new URL("http://" + req.headers.host + req.url)
    const name = myURL.searchParams.get('onoma')

    res.setHeader('Content-Type',  'text/html; charset=utf-8')
    res.write(`Καλή σου μέρα, ${name}!`);
    res.end();
}).listen(8080); // ο εξυπηρετητής αποκρίνεται στη θύρα 8080
```

Η πολύ απλοϊκή εφαρμογή μας θα εμφανίσει στον φυλλομετρητή ένα μήνυμα καλωσορίσματος (**Εικόνα *11.6***)



Εικόνα 11.6 *Η εφαρμογή παράγει διαφορετική απόκριση με βάση την τιμή της παραμέτρου onoma.*

11.7 ES6 export και import

Όπως είδαμε πριν (Ενότητα 11.3.3), η Node.js υποστηρίζει το πιο πρόσφατο πρότυπο βιβλιοθηκών ECMAScript6, γνωστό και ως ES6. Με αυτό το πρότυπο μπορούμε να μοιράσουμε τον κώδικά μας σε λογικές ενότητες, πρακτικά σε διαφορετικά αρχεία, χρησιμοποιώντας τον ίδιο μηχανισμό που έχουμε και στην πλευρά του φυλλομετρητή. Έχουμε ήδη δει σε προηγούμενα παραδείγματα πώς μπορούμε να φορτώσουμε βιβλιοθήκες με τον μηχανισμό ES6.

Σε αυτή την ενότητα θα δούμε το πρότυπο με μεγαλύτερη λεπτομέρεια έτσι ώστε να μπορούμε να γράφουμε δικές μας βιβλιοθήκες και να φορτώνουμε βιβλιοθήκες τρίτων με μεγαλύτερη ευελιξία. Η χρήση των βιβλιοθηκών ES6 είναι απλή, αλλά ο μηχανισμός παρέχει κάμποσους εναλλακτικούς τρόπους για να ορίσουμε και να φορτώσουμε βιβλιοθήκες, που καλό είναι να τους έχουμε υπόψη.

11.7.1 Ονοματισμένα export

Για να ορίσουμε δικές μας βιβλιοθήκες χρησιμοποιούμε τη δήλωση export, μπροστά από τα στοιχεία που θέλουμε να κάνουμε διαθέσιμα για φόρτωμα. Αυτά τα στοιχεία μπορεί να είναι μεταβλητές (**var**, **let** ή **const**), συναρτήσεις ή κλάσεις:

```
//αρχείο my-module.mjs
export let myVariable = "Καλημέρα"

export function myFunction() {
    console.log("Δουλεύω...")
}

export class MyClass {
    constructor() {
        console.log("Κατασκευάζω...")
    }
}
```

Εναλλακτικά, και μερικές φορές ίσως είναι πιο βολικό, μπορούμε να έχουμε μια μοναδική δήλωση export που να περιέχει σε άγκιστρα τα στοιχεία που θέλουμε να διαθέσουμε:

```
export {myVariable, myFunction, myClass}
```

Έχουμε λοιπόν **ονοματίσει** ποια στοιχεία θέλουμε να κάνουμε διαθέσιμα για φόρτωμα σε άλλα αρχεία. Για να τα φορτώσουμε χρησιμοποιούμε τη δήλωση import, ονοματίζοντας και πάλι τα στοιχεία που χρειαζόμαστε για να φορτώσουμε:

```
//αρχείο use-my-module.mjs
//το my-module-mjs βρίσκεται στον ίδιο φάκελο ("./")
import {myVariable, myFunction, MyClass}  from './my-module.mjs';

console.log(myVariable)  //τυπώνει "Καλημέρα"
myFunction() //τυπώνει "Δουλεύω..."
const myObj = new MyClass()
```

11.7.2 Μετονομασία των στοιχείων

Συχνά θα χρειαστεί να φορτώσουμε βιβλιοθήκες που παρέχουν τρίτοι, όπου ενδεχομένως χρησιμοποιούνται ονόματα για τα στοιχεία που εξάγουν ίδια με τα δικά μας ονόματα. Ο μηχανισμός ES6 μάς δίνει τη δυνατότητα μετονομασίας των στοιχείων που φορτώνουμε με τη λέξη-κλειδί as:

```
import {myVariable as myOtherVariable, myFunction as anotherFunction}
from './my-module.mjs';

myVariable="Καληνύχτα"
console.log(myOtherVariable) //τυπώνει "Καλημέρα"
anotherFunction()  //τυπώνει "Δουλεύω..."
```

Η ίδια λέξη-κλειδί as μπορεί να χρησιμοποιηθεί και στην πλευρά του export:

```
export {myVariable as myOtherVariable, myFunction, myClass}
```

11.7.3 Default exports

Ο λόγος όμως που χρησιμοποιούμε κάποιο σύστημα βιβλιοθηκών εξαρχής είναι πως επιδιώκουμε τα προγράμματά μας να είναι αρθρωτά. Να αποτελούνται δηλαδή από καλά ορισμένα επιμέρους τμήματα, καθένα από τα οποία να επιτελεί μια εργασία. Αν ένα τμήμα ή βιβλιοθήκη επιτελεί περισσότερες εργασίες, ίσως είναι καλή ιδέα να τη διασπάσουμε σε επιμέρους αρχεία.

Στο ES6 μπορούμε όμως να ακολουθήσουμε έναν ενδιάμεσο δρόμο και να ορίσουμε *ένα* στοιχείο που

να φορτώνεται χωρίς να ονοματιστεί, το λεγόμενο **default export**. Θα μπορούμε να έχουμε πολλά export, αλλά μόνο ένα θα είναι το **default**:

```
//αρχείο my-module.mjs
export let myVariable = "Καλημέρα"

export default function myDefaultFunction() {
    console.log("Δουλεύω πάντα...")
}

export class MyClass { ... }
```

ή εναλλακτικά:

```
export {myVariable, myFunction as default, myClass}
```

Πλέον, πέρα από τα *ονοματισμένα* import, μπορούμε να φορτώσουμε και το default ή μόνο αυτό:

```
import myDefaultFunction from './my-module.mjs'

...

myDefaultFunction() //τυπώνει "Δουλεύω πάντα..."
```

ή

```
import { default as anotherFunction } from './my-module.mjs'

...

anotherFunction() //τυπώνει "Δουλεύω πάντα..."
```

Τέλος, μπορούμε να συνδυάσουμε default και ονοματισμένα import, αρκεί το default να είναι πρώτο:

```
import myDefaultFunction, {myVariable, myFunction, MyClass}  from './my-
module.mjs';
```

11.7.4 Εισαγωγή της βιβλιοθήκης ως αντικειμένου

Ο πιο πρακτικός τρόπος για να φορτώσουμε τη βιβλιοθήκη είναι να τη φορτώσουμε ως αντικείμενο και είναι πολύ απλός:

```
// Τα στοιχεία που κάνει export το my-module θα είναι διαθέσιμα
// στο αντικείμενο με όνομα MyModule
import * as MyModule from './my-module.mjs'

console.log(MyModule.myOtherVariable) //τυπώνει "Καλημέρα"
MyModule.myFunction()  //τυπώνει "Δουλεύω..."
```

Με τον τρόπο αυτό, έχουμε δημιουργήσει ένα νέο αντικείμενο, με όνομα MyModule, μέλη του οποίου είναι όλα τα export του my-module.mjs. Και πάλι, μπορούμε να συνδυάσουμε την εισαγωγή ως αντικείμενο με την εισαγωγή default, αρκεί το default να είναι πρώτο:

```
import myDefaultFunction, * as MyModule from './my-module.mjs';
```

11.7.5 Φόρτωση και εκτέλεση βιβλιοθήκης

Πιο σπάνια, μπορεί να χρειαστεί να φορτώσουμε μια βιβλιοθήκη όχι για να εισαγάγουμε κάποια στοιχεία από αυτή, αλλά για να εκτελεστεί κάποιος κώδικας που περιέχει. Ένα τέτοιο παράδειγμα υπάρχει πιο κάτω στο κεφάλαιο, στη χρήση του πακέτου dotenv (11.8.4). Σε αυτή την περίπτωση δεν χρειάζεται να προσδιορίσουμε ποια στοιχεία θέλουμε να εισαγάγουμε, απλά φορτώνουμε τη βιβλιοθήκη. Έστω μια απλή βιβλιοθήκη:

```
//αρχείο my-module.mjs
export let myVariable = "Καλημέρα"

export function myFunction() {
    console.log("Δουλεύω...")
}

export class MyClass {
    constructor() {
        console.log("Κατασκευάζω...")
    }
}
console.log("Εκτέλεση κώδικα στο my-module.mjs")
```

Φορτώνοντας λοιπόν τη βιβλιοθήκη μας

```
import './my-module.mjs'
//Θα εμφανιστεί το μήνυμα "Εκτέλεση κώδικα στο my-module.mjs"
//τα exports δεν θα είναι διαθέσιμα.
```

θα εκτελεστεί ο κώδικας που αυτή περιέχει, αλλά τα export δεν θα μας είναι διαθέσιμα.

11.7.6 Δυναμικά import

Από την έκδοση EcmaScript 2020 του προτύπου μπορούμε να χρησιμοποιήσουμε την έκφραση import(), με σύνταξη που μοιάζει με συνάρτηση, για να φορτώσουμε βιβλιοθήκες. Έτσι, μπορούμε να χρησιμοποιήσουμε την import() μέσα σε μια συνάρτηση ή σαν μέρος ενός λογικού ελέγχου συνθήκης με **if** κ.ο.κ.

Με την έκφραση import() επιστρέφεται ένα Promise, το οποίο ολοκληρώνεται (fullfil) όταν φορτωθεί η βιβλιοθήκη. Μπορούμε έτσι, για παράδειγμα, να γράψουμε προγράμματα που φορτώνουν βιβλιοθήκες ανάλογα με την τιμή κάποιας παραμέτρου από το περιβάλλον:

```
const PLATFORM = process.env.PLATFORM // μπορεί να είναι Windows, Mac,
Linux κ.λπ.

if (PLATFORM=="Windows") {
    await import('./module_with_windows_functionality.mjs')
    doStuff()
    ...
} else {
    ...
}
```

11.7.7 Ιδιότητες και περιορισμοί

Συνοπτικά, ο μηχανισμός των βιβλιοθηκών ECMAScript είναι κατανοητός και απλός. Πέρα από τα παραπάνω παραδείγματα, μερικά άλλα σημαντικά σημεία είναι πως, με εξαίρεση τα δυναμικά import –με import()–, όλα τα export και import πρέπει να είναι στο ανώτερο επίπεδο στον κώδικά μας, π.χ. δεν μπορεί να γίνει export σε μια συνάρτηση που ανήκει σε μια κλάση, ούτε να χρησιμοποιηθεί η import μέσα σε μια συνάρτηση – η import() όμως ναι.

Επιπλέον, δεν έχει σημασία σε ποια γραμμή γίνεται το import, θα γίνει πάντα hoist (Ενότητα 7.3.4), δηλαδή θα μετακινηθεί αυτόματα πάνω και η βιβλιοθήκη που κάνουμε import θα είναι διαθέσιμη από την πρώτη γραμμή του κώδικά μας.

Τέλος, ένα σημαντικό χαρακτηριστικό του μηχανισμού είναι πως τα στοιχεία που κάνουμε import είναι μόνο για ανάγνωση και δεν μπορούμε να μεταβάλουμε τις τιμές τους.

Άσκηση
Γράψτε μια εφαρμογή διαδικτύου όπου η λίστα με τα ψώνια να εμφανίζεται, αντί για την κονσόλα, σαν απάντηση σε ένα αίτημα HTTP.

11.8 Βοηθητικά εργαλεία για την ανάπτυξη εφαρμογών Node.js

Στο κλείσιμο αυτού του κεφαλαίου θα αναφερθούμε σε βοηθητικά εργαλεία που είναι χρήσιμα για την ανάπτυξη εφαρμογών Node.js, τα οποία ξεπερνούν τα όρια του τετριμμένου παραδείγματος. Όσο αυξάνεται η πολυπλοκότητα της εφαρμογής μας χρειάζεται να χρησιμοποιήσουμε τεχνικές και εργαλεία που κάνουν τη ζωή μας ως προγραμματιστές πιο εύκολη. Επίσης, θα περιγράψουμε πώς μπορούμε με απλά βήματα να δημοσιεύσουμε την εφαρμογή μας ώστε να είναι διαθέσιμη στο κοινό, μέσα από την υπηρεσία Heroku.

11.8.1 Dependencies και DevDependencies

Όπως είδαμε στην ενότητα 11.3.5, το εργαλείο npm μας βοηθά να προσδιορίσουμε και να εγκαταστήσουμε τις εξαρτήσεις της εφαρμογής μας. Μερικές από τις εξαρτήσεις, όμως, δεν χρειάζονται για τη λειτουργία της εφαρμογής, αλλά είναι πολύ χρήσιμες για τον προγραμματιστή κατά την ανάπτυξή της. Τέτοιο παράδειγμα είναι το πακέτο nodemon που παρουσιάζεται αμέσως μετά.

Το npm μας δίνει τη δυνατότητα να έχουμε εξαρτήσεις δύο ειδών, τις απλές, που λειτουργούν όπως ήδη γνωρίζουμε, και τις εξαρτήσεις ανάπτυξης (DevDependencies). Οι εξαρτήσεις ανάπτυξης βρίσκονται και αυτές στο package.json, αλλά στο αντικείμενο "devDependencies".

Οι εξαρτήσεις ανάπτυξης εγκαθίστανται με την εντολή

```
npm install <package-name> --save-dev
```

Με την εντολή npm install θα εγκατασταθούν όλες οι εξαρτήσεις, τόσο οι απλές όσο και οι ανάπτυξης, π.χ. αν διαγράψουμε τον φάκελο node_modules/ και θέλουμε να τον ανασυνθέσουμε.

Ωστόσο, και εδώ είναι η κύρια διαφορά, μπορούμε να ζητήσουμε να εγκατασταθεί το πακέτο μας όχι για ανάπτυξη, αλλά για παραγωγική χρήση. Τότε, με την εντολή

```
npm install --production
```

θα εγκατασταθούν μόνο οι απλές εξαρτήσεις και όχι οι εξαρτήσεις ανάπτυξης.

11.8.2 Το πακέτο nodemon

Στο βασικό παράδειγμα που είδαμε, κάθε φορά που κάνουμε μια αλλαγή στον εξυπηρετητή μας θα πρέπει να τον σταματήσουμε (πατώντας Ctrl-C ή Cmd-C στο παράθυρο του τερματικού) και να ξεκινήσουμε από την αρχή τον εξυπηρετητή όταν κάνουμε τροποποιήσεις στον κώδικα.

Το πακέτο nodemon επανεκκινεί την εφαρμογή μας αυτόματα όταν αποθηκεύσουμε το αρχείο με τον κώδικά μας. Είναι απαραίτητα δύο βήματα, πρώτα να εγκαταστήσουμε το nodemon και έπειτα να ορίσουμε ένα σκριπτ με το οποίο θα ξεκινάει το nodemon. Τα δύο πρώτα βήματα χρειάζεται να τα κάνουμε μόνο μια φορά:

1. Εγκαθιστούμε το πακέτο τοπικά εκτελώντας στο τερματικό (έχουμε αρχικοποιήσει βέβαια το πακέτο μας με npm init -y):

   ```
   npm install nodemon --save-dev
   ```

2. Στο αρχείο package.json του πακέτου μας προσθέτουμε στην ιδιότητα scripts το όνομα του σκριπτ, π.χ. “start”, και την εντολή που θέλουμε να εκτελείται, στην προκειμένη περίπτωση nodemon <όνομα αρχείου>:

   ```
   "scripts": {
       "test": "echo \"Error: no test specified\" && exit 1",
       "start": "nodemon server.mjs"
   },
   ```

3. Κάθε φορά που θέλουμε να δουλέψουμε με το πρότζεκτ μας αρκεί να εκτελέσουμε το σκριπτ μας από το τερματικό εκτελώντας npm run <όνομα σκριπτ>:

   ```
   npm run start
   ```

Για την ακρίβεια, η επανεκκίνηση με το nodemon γίνεται σε κάθε αλλαγή σε αρχεία στον φάκελό μας που έχουν κατάληξη js, mjs, και json. Αυτή η συμπεριφορά μπορεί να αλλάξει ρυθμίζοντας κατάλληλα το nodemon,

π.χ. γράφοντας στο package.json:

```
...
"nodemonConfig": {
  "ignore": ["*.json"]
},
...
```

11.8.3 Ορισμός και χρήση μεταβλητών περιβάλλοντος

Πολύ συχνά είναι σκόπιμο να μη γράφουμε κάποιες τιμές στον κώδικά μας, αλλά να τις παίρνουμε από το περιβάλλον της εφαρμογής μας. Οι μεταβλητές περιβάλλοντος είναι μεταβλητές που δεν ανήκουν στην εφαρμογή μας, αλλά υπάρχουν στο λειτουργικό σύστημα στο οποίο αυτή εκτελείται. Συνήθως αυτές οι μεταβλητές γράφονται με κεφαλαία γράμματα. Η χρήση των μεταβλητών περιβάλλοντος μας επιτρέπει να διαχωρίσουμε την παραμετροποίηση της εφαρμογής μας από την ίδια την εφαρμογή, να αποθηκεύσουμε δηλαδή τις τιμές της παραμετροποίησης ξεχωριστά από την ίδια την εφαρμογή.

Μια τέτοια περίπτωση είναι ο αριθμός της θύρας (port) στην οποία θα δέχεται μηνύματα ο εξυπηρετητής μας. Στα προηγούμενα παραδείγματα την έχουμε ορίσει με 8080. Συχνά όμως αυτή θα είναι μια παράμετρος η οποία εξαρτάται από το λειτουργικό σύστημα στο οποίο εκτελείται η εφαρμογή. Θα μπορούσε, για παράδειγμα, να υπάρχει ήδη μια εφαρμογή που να περιμένει μηνύματα στη θύρα 8080 και, συνεπώς, να πρέπει να αφήσουμε το σύστημα να προσδιορίσει τη θύρα. Αυτό μπορεί να γίνει μέσω μιας μεταβλητής περιβάλλοντος.

Οι μεταβλητές περιβάλλοντος είναι διαθέσιμες στη Node.js μέσα από την ενσωματωμένη βιβλιοθήκη process, που έχει πληροφορίες για τη διεργασία που εκτελεί τη Node.js στον υπολογιστή.

Συνηθίζεται, για παράδειγμα, να χρησιμοποιούμε τη θύρα που ορίζεται στη μεταβλητή περιβάλλοντος **PORT** σαν θύρα στην οποία ο εξυπηρετητής μας περιμένει για συνδέσεις. Η τιμή αυτής της μεταβλητής θα είναι διαθέσιμη στην εφαρμογή μας μέσω της ιδιότητας process.env.PORT. Για να εξασφαλίσουμε πως θα έχουμε πάντα μια τιμή για τη θύρα μας, ανεξάρτητα από το αν υπάρχει ή όχι η μεταβλητή περιβάλλοντος, μπορούμε να γράψουμε:

```
const serverListeningPort = process.env.PORT || 8080

http.createServer(function (req, res) {
    ...
}.listen(serverListeningPort)
```

11.8.4 Μεταβλητές περιβάλλοντος μέσω dotenv

Το πακέτο dotenv επιτρέπει να έχουμε «μεταβλητές περιβάλλοντος» σε εξωτερικά αρχεία. Με άλλα λόγια, μας επιτρέπει να ορίσουμε εμείς κάποιες μεταβλητές, οι οποίες θα είναι διαθέσιμες στην εφαρμογή μας σαν να ήταν μεταβλητές περιβάλλοντος.

1. Εγκαθιστούμε το πακέτο τοπικά εκτελώντας στο τερματικό (έχουμε αρχικοποιήσει βέβαια το πακέτο μας με npm init -y):

   ```
   npm install dotenv
   ```

2. Σε αρχείο με όνομα **.env** (dot-env) γράφουμε τα ζεύγη μεταβλητών-τιμών που θέλουμε να είναι διαθέσιμα στο περιβάλλον της εφαρμογής μας, π.χ.:

   ```
   HOST = localhost
   PORT = 8081
   ```

3. Για να φορτώσουμε τις μεταβλητές στην εφαρμογή μας αρκεί να γράψουμε:

   ```
   import 'dotenv/config'
   //Πλέον οι τιμές που έχουν οριστεί στο .env είναι διαθέσιμες:
   console.log(HOST)
   console.log(PORT)
   ```

Στην περίπτωση που η μεταβλητή περιβάλλοντος που έχουμε ορίσει στο αρχείο .env υπάρχει ήδη στο

περιβάλλον του λειτουργικού συστήματος, θα χρησιμοποιηθεί αυτή του συστήματος.

11.8.5 Φιλοξενία της εφαρμογής στο Heroku

Μια εφαρμογή διαδικτύου δεν έχει πολύ νόημα αν δεν είναι προσβάσιμη στο διαδίκτυο. Για όσο διαρκεί η ανάπτυξη της εφαρμογής μας μπορεί να επαρκεί να χρησιμοποιούμε τον υπολογιστή για την ανάπτυξη και τη φιλοξενία της. Όταν θελήσουμε όμως να τη δείξουμε σε άλλους ή να τη διαθέσουμε στο ευρύτερο κοινό, αυτό δεν θα είναι εύκολα δυνατό από τον υπολογιστή μας. Μπορούμε να χρησιμοποιήσουμε διάφορες υπηρεσίες, από τις οποίες αρκετά δημοφιλής είναι αυτή που παρέχει το Heroku.

Προαπαιτούμενα

Για να χρησιμοποιήσουμε την υπηρεσία του Heroku χρειάζεται να ικανοποιήσουμε μερικά προαπαιτούμενα. Συγκεκριμένα, χρειάζονται

- Ένας λογαριασμός (δωρεάν) στο Heroku.
- Να εγκαταστήσουμε τα εργαλεία Git (όχι Github).
- Να εγκαταστήσουμε το εργαλείο Heroku CLI (command line interface).

Θα πρέπει να σημειωθεί ότι η πλατφόρμα αυτή κατήργησε τη δωρεάν χρήση προς όλους, αφήνοντας όμως τη δυνατότητα δωρεάν χρήσης της υπηρεσίας για μη κερδοσκοπικούς ή εκπαιδευτικούς σκοπούς. Συνεπώς, θα πρέπει να κάνουμε αίτηση για δωρεάν υπηρεσία ως μη κερδοσκοπική ή εκπαιδευτική δραστηριότητα ώστε να μπορέσουμε να τη χρησιμοποιήσουμε.

Με τη χρήση του Heroku, αυτόματα θα έχουμε δύο σημεία στα οποία θα υπάρχει η εφαρμογή μας: Ο υπολογιστής στον οποίο γίνεται η ανάπτυξή της, η διόρθωση σφαλμάτων, η προσθήκη λειτουργιών κ.λπ., και ο χώρος μας στο Heroku, όπου θα ανεβάζουμε ή θα «σπρώχνουμε» (push) τις αλλαγές στη δημόσια έκδοση της εφαρμογής μας. Η διαδικασία αυτή γίνεται στο Heroku μέσα από το δημοφιλές εργαλείο τήρησης εκδόσεων git. Αν και το git δείχνει τις δυνατότητές του σε εφαρμογές που αναπτύσσονται από πολλούς προγραμματιστές ταυτόχρονα, ωστόσο, ακόμα και στην περίπτωση που ο προγραμματιστής είναι ένας είναι χρήσιμο εργαλείο για τη διατήρηση του ιστορικού ανάπτυξης της εφαρμογής, καθώς διατηρεί το σωρευτικό ιστορικό των αλλαγών που πραγματοποιούμε στον κώδικα.

Προετοιμασία της εφαρμογής μας

Πριν τη φιλοξενία της εφαρμογής μας από το Heroku χρειάζεται να έχουμε αρχικοποιήσει το πακέτο μας εκτελώντας npm init -y και να δηλώσουμε στο package.json ποια έκδοση της Node.js θέλουμε να χρησιμοποιήσει το Heroku όταν εκτελεί την εφαρμογή μας:

- Με την εντολή node -v στο τερματικό βρίσκουμε την έκδοση που έχουμε και
- δηλώνουμε στο packages.json την έκδοσή μας:

```
"engines": {
    "node": "16.x"
},
```

Χρειάζεται επίσης να προσδιορίσουμε ποιο είναι το σημείο εισόδου στην εφαρμογή μας. Για να το δηλώσουμε αυτό δημιουργούμε ένα αρχείο με όνομα Procfile.

- Στο αρχείο Procfile γράφουμε (αν, π.χ,. η εφαρμογή ξεκινάει από το αρχείο server.mjs):

  ```
  web: node server.mjs
  ```

- Πριν ανεβάσουμε την εφαρμογή, μπορούμε να δοκιμάσουμε τοπικά χρησιμοποιώντας το Heroku CLI:

  ```
  heroku local web
  ```

- Αν θέλουμε να δοκιμάσουμε σε διαφορετικό PORT, π.χ. 8888, γράφουμε στο αρχείο με όνομα .env (δείτε στην ενότητα 11.8.4 για το πακέτο dotenv).

  ```
  PORT=8888
  ```

- Αν δεν το έχουμε κάνει ήδη, αρχικοποιούμε το git στον φάκελο του πρότζεκτ μας με

  ```
  git init
  ```

- Αν όλα πήγαν καλά, τότε κάνουμε login και ζητάμε από το Heroku να μας δώσει τον χώρο που θα ανεβάσουμε το πρότζεκτ, είτε από τη σελίδα του dashboard στο Heroku είτε απευθείας από τη γραμμή εντολών

```
heroku login
...
heroku create --region=eu
```

Η τελευταία εντολή θα μας δώσει δύο πληροφορίες:
- τον σύνδεσμο στον οποίο θα φιλοξενηθεί το πρότζεκτ μας, δηλαδή το URL με το οποίο θα ανοίγουμε την εφαρμογή στον φυλλομετρητή,
- το **git remote** στο οποίο θα ανεβάζουμε (push) τις νέες μας εκδόσεις. Το git remote είναι ο χώρος στον οποίο θα φυλάσσονται τα αρχεία μας στο Heroku. Κάθε φορά που θα είμαστε έτοιμοι για να δημοσιεύσουμε μια νέα έκδοση θα την ανεβάζουμε στο git remote.

Πολλά από τα αρχεία που συνοδεύουν την εφαρμογή μας δεν χρειάζεται να δημοσιεύονται στο git remote. Ένα παράδειγμα είναι τα περιεχόμενα του υποφακέλου node_modules/ του πρότζεκτ μας. Όπως είδαμε (Ενότητα 11.3.5), μπορούμε όποτε θέλουμε να εγκαταστήσουμε τα πακέτα που περιέχονται εκεί, συνεπώς δεν χρειάζεται να τα φυλάμε στο ιστορικό του git.

- Για να το πετύχουμε αυτό δημιουργούμε ένα αρχείο με όνομα .gitignore, που κάθε γραμμή του ορίζει φακέλους ή αρχεία που θα αγνοούνται από το ιστορικό του git:

```
# Να αγνοούνται τα node modules
/node_modules
# διάφορα αρχεία
npm-debug.log
# αρχείο με πληροφορίες φακέλου στο MacOS
.DS_Store
# το αρχείο dotenv (.env)
*.env
```

Δημοσίευση στο Heroku

Τα προηγούμενα βήματα χρειάζεται να γίνουν μόνο μια φορά για κάθε πρότζεκτ μας. Στη συνέχεια, όταν θέλουμε να δημοσιεύσουμε την εφαρμογή με όλες τις αλλαγές και τροποποιήσεις, αρκεί να επαναλάβουμε τα παρακάτω βήματα.

- Ενημερώνουμε το ιστορικό του git:
  - Αν έχουμε προσθέσει νέα αρχεία, με την εντολή git add . για να επισημάνουμε τα νέα αρχεία που έχουμε δημιουργήσει. Έτσι, αυτά θα συμπεριληφθούν στην επόμενη καταχώριση στο ιστορικό του git. Θα συμπεριληφθούν όλα τα αρχεία που δεν αναγράφονται στο .gitignore,
  - git commit -m «Περιγραφή των αλλαγών» θα κάνει μια νέα καταχώριση στο ιστορικό του git. Θα καταχωριστούν οι αλλαγές στα αρχεία μας μέχρι τώρα σαν μια νέα έκδοση. Η πιο πρόσφατη έκδοση ονομάζεται πάντα "HEAD".
- Ανεβάζουμε (push) την πιο πρόσφατη έκδοσή μας στο Heroku. Με την εντολή git push heroku master ανεβαίνει πάντα η έκδοση "HEAD", δηλαδή αυτή που καταχωρίστηκε την τελευταία φορά που εκτελέσαμε git commit.

Άσκηση

Δημοσιεύστε στο Heroku τη μικρή εφαρμογή διαδικτύου που φτιάξατε στην προηγούμενη άσκηση.

11.8.6 Αποσφαλμάτωση με το Visual Studio Code

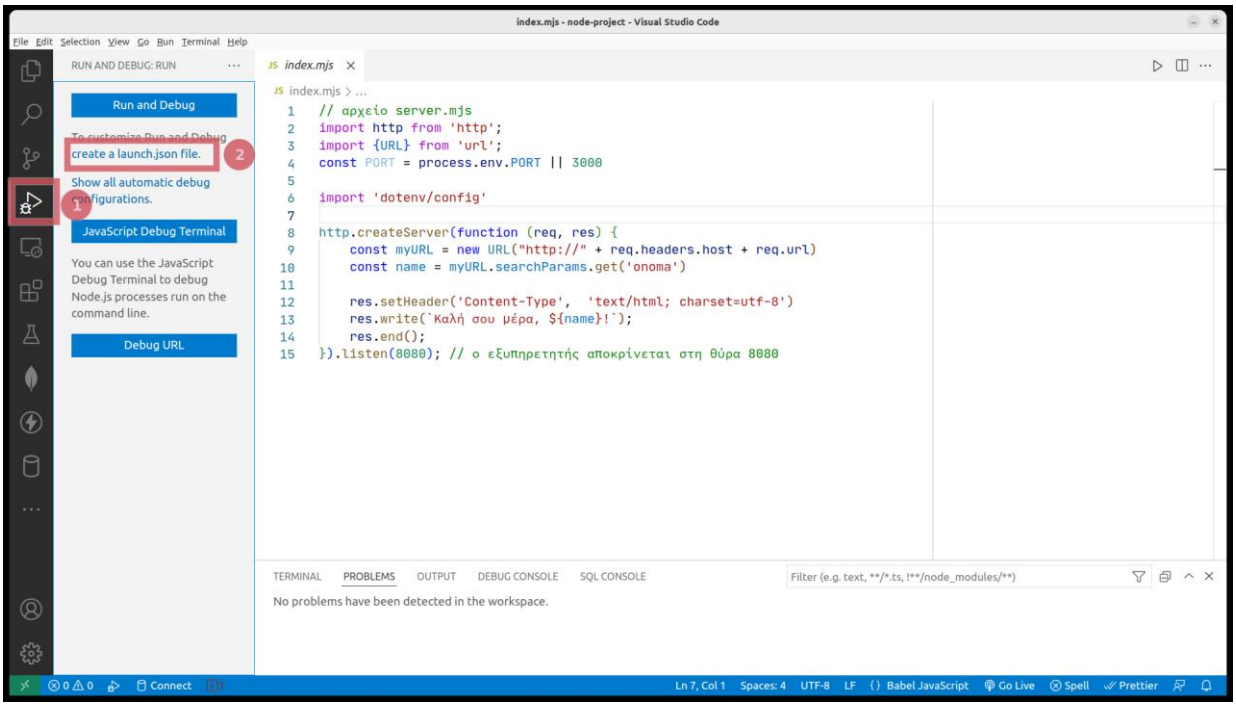
Η εύρεση και αντιμετώπιση σφαλμάτων είναι συχνά χρονοβόρα και επίπονη διαδικασία. Το εργαλείο αποσφαλμάτωσης (debugger) που είναι ενσωματωμένο στο Visual Studio Code είναι εύχρηστο και πολύ χρήσιμο για την εύρεση και την αντιμετώπιση σφαλμάτων.

Ρύθμιση του εργαλείου αποσφαλμάτωσης

Για να ενεργοποιηθεί το εργαλείο αποσφαλμάτωσης χρειάζεται να γίνουν δύο ενέργειες. Αρχικά, να προστεθεί μια ρύθμιση στο αρχείο .vscode/launch.json. Αν το αρχείο αυτό δεν υπάρχει ήδη στο πρότζεκτ μας, μπορεί να

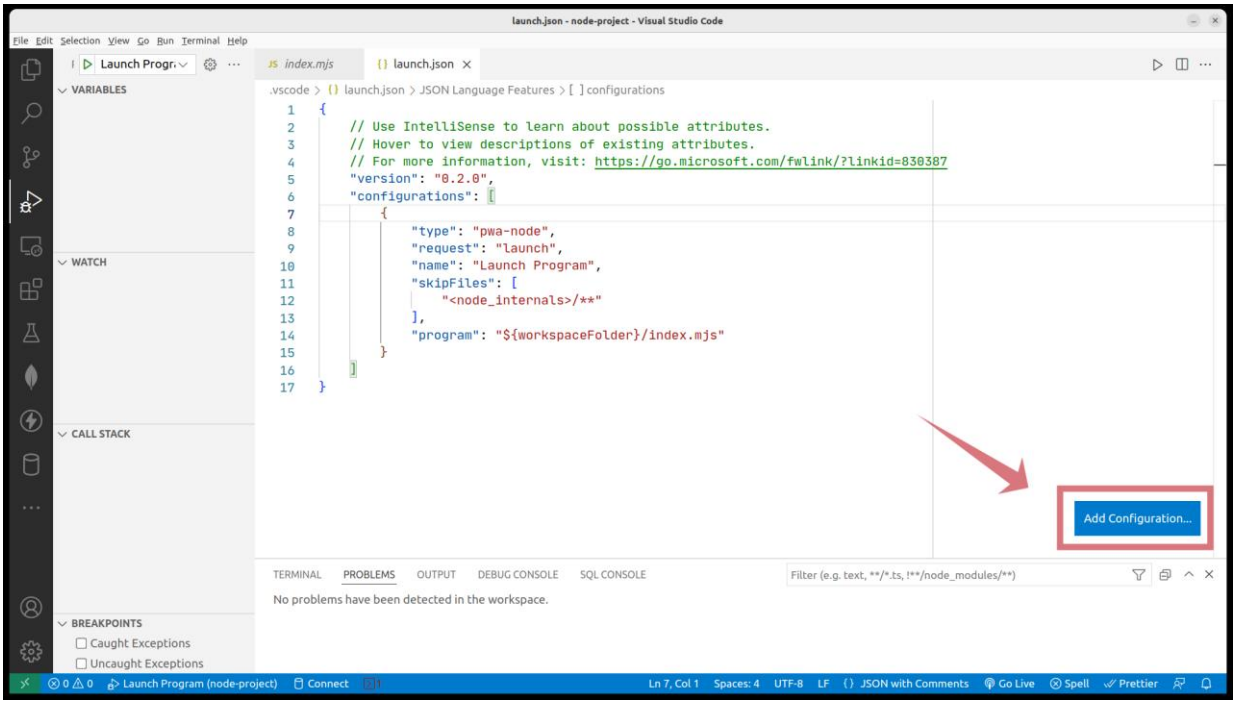
δημιουργηθεί πολύ εύκολα, για παράδειγμα:

- Στο Visual Studio Code επιλέγουμε το εργαλείο “Run and Debug” (ή πατάμε Ctrl+Shift+D (σε Mac Cmd+Shift+D) από την αριστερή στήλη εργαλείων,
- επιλέγουμε τον σύνδεσμο create a launch.json file.



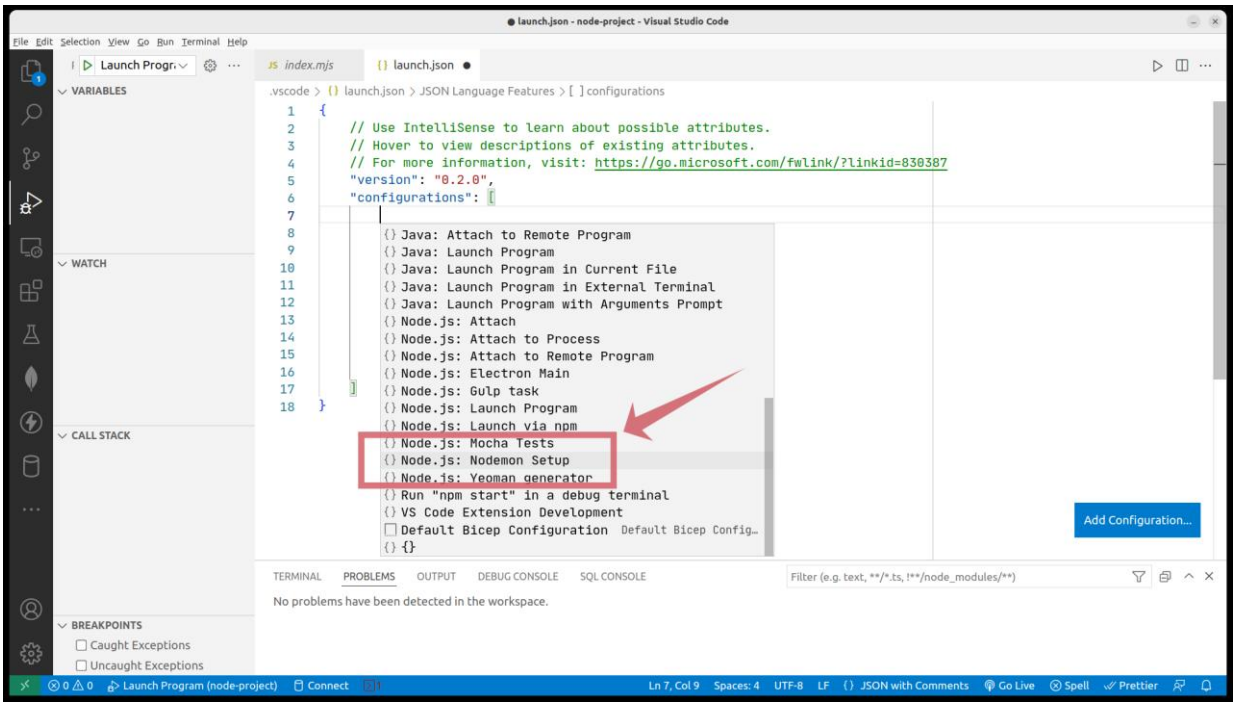
Εικόνα 11.7 *Βήματα για τη δημιουργία του αρχείου .vscode/launch.json, αν δεν υπάρχει ήδη.*

Στη συνέχεια θα εμφανιστεί το αρχείο launch.json που θα δημιουργήσει αυτόματα το Visual Studio Code. Επιλέγουμε “Add Configuration…” (**Εικόνα *11.8***).



Εικόνα 11.8 *Δημιουργούμε ένα νέο launch configuration.*

Έπειτα, επιλέγουμε ένα από τα έτοιμα launch configuration που μας δίνει το εργαλείο (**Εικόνα *11.9***). Μια καλή επιλογή είναι να διαλέξουμε το **nodemon**, όπως φαίνεται στην εικόνα. Αυτό βέβαια χρειάζεται να υπάρχει το nodemon εγκατεστημένο είτε στο σύστημά μας είτε σαν εξάρτηση στο πακέτο μας (Ενότητα 11.8.2).



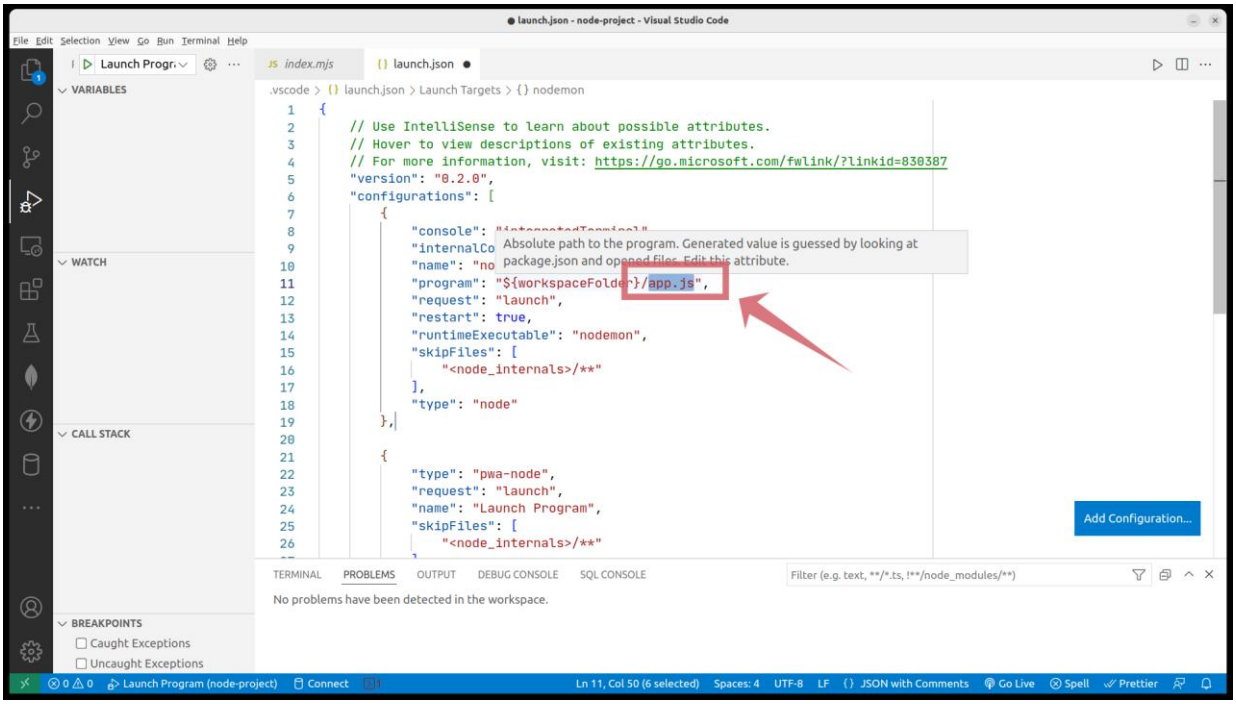
Εικόνα 11.9 *Επιλογή του προτιμώμενου launch configuration.*

Αν προσέξετε τις επιλογές που έχει κάνει το Visual Studio Code, θα δείτε ότι θεωρεί πως το αρχείο που τρέχει την εφαρμογή μας είναι το **app.js**. Προφανώς, αυτό θα πρέπει να το αλλάξουμε, ώστε να ανταποκρίνεται στο όνομα του δικού μας προγράμματος, π.χ. index.mjs ή server.mjs ή όπως αλλιώς ονομάζεται το αρχείο μας. Θα μπορούσαμε εδώ να δώσουμε την τιμή:

```
"program": "${workspaceFolder}/${relativeFile}",
```

Έτσι, όταν εκσφαλματώσουμε θα τρέξει το τρέχον επιλεγμένο αρχείο στο Visual Studio Code.

Μια άλλη ιδιότητα που ίσως θέλουμε να αλλάξουμε είναι η name, που τώρα έχει τιμή “nodemon”. Μπορούμε να την ονομάσουμε «Αποσφαλμάτωση με nodemon» και να αποθηκεύσουμε το αρχείο.



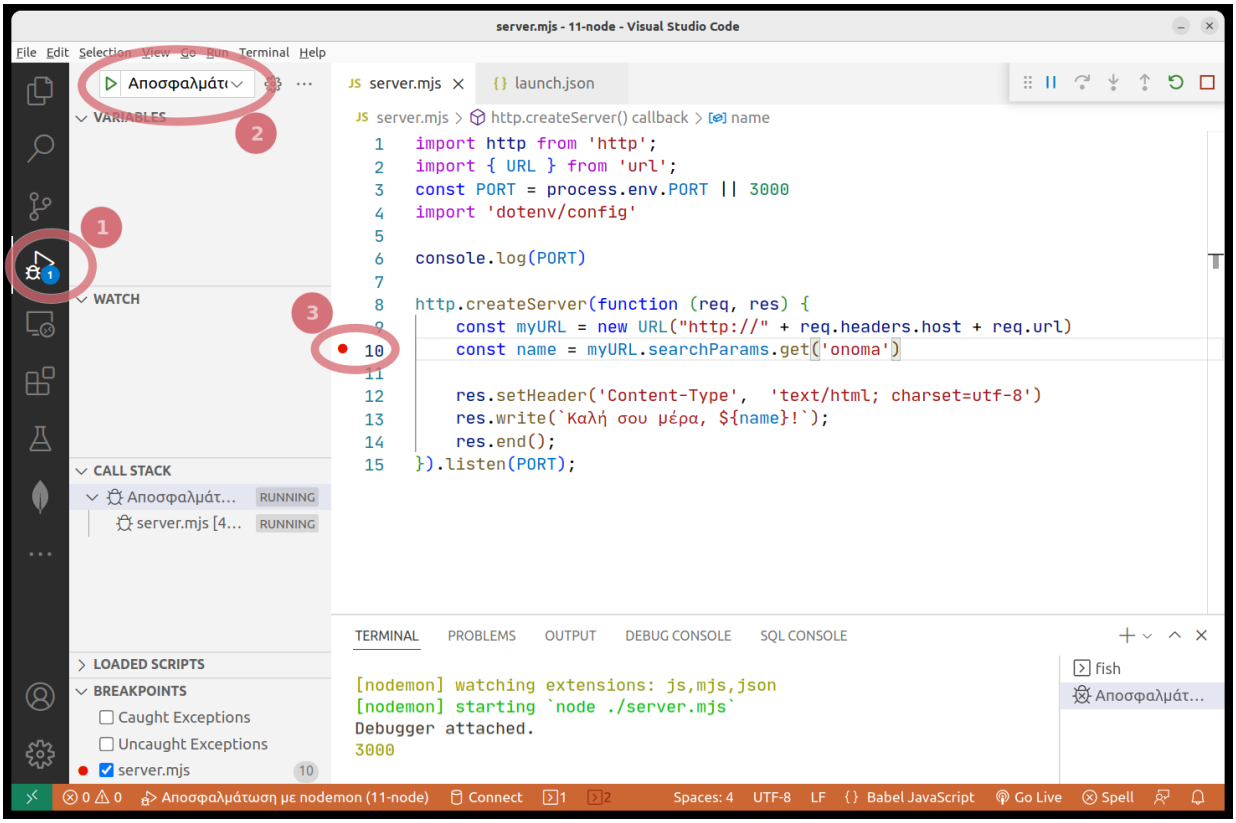
Εικόνα 11.10 *Προσαρμογή των επιμέρους ρυθμίσεων του launch configuration. Προσοχή στο όνομα του αρχείου που θέλουμε να εκσφαλματώσουμε.*

Με το launch configuration έτοιμο, έχουμε πλέον στη διάθεσή μας την επιλογή «Αποσφαλμάτωση με nodemon».

Χρήση του εργαλείου αποσφαλμάτωσης

Πλέον, μπορούμε να εκτελέσουμε τον κώδικά μας βήμα βήμα και να παρατηρήσουμε τις τιμές των μεταβλητών την ώρα της εκτέλεσης.

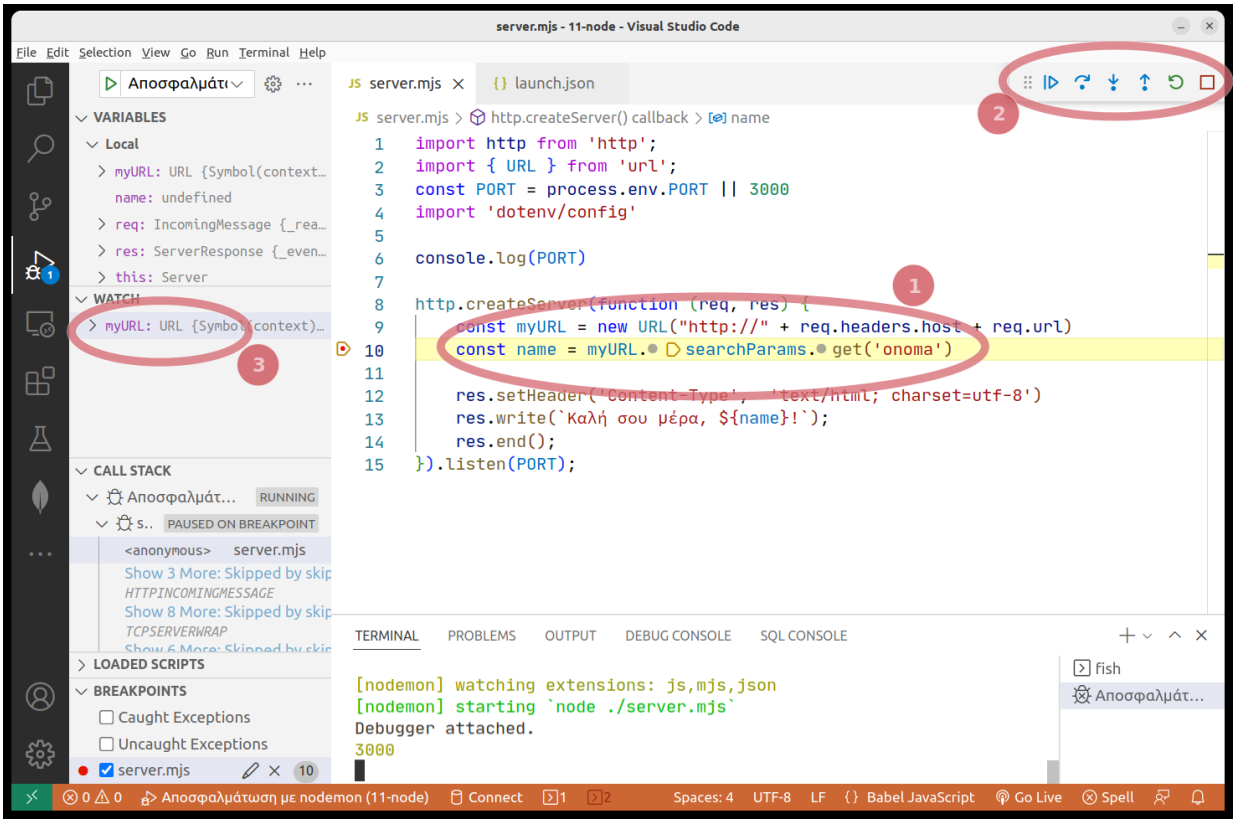
- Ανοίγουμε στο Visual Studio Code το αρχείο που αποτελεί την είσοδο στην εφαρμογή μας, π.χ. το αρχείο server.mjs.
- Μεταβαίνουμε στο εργαλείο αποσφαλμάτωσης με Ctrl+Shift+D (βήμα 1 στην **Εικόνα *11.11***).
- Επιλέγουμε το launch configuration που μόλις δημιουργήσαμε. Στο παράδειγμά μας αυτό είναι το «Αποσφαλμάτωση με nodemon» (βήμα 2 στην εικόνα 114).
- Ξεκινάμε την αποσφαλμάτωση πατώντας το πράσινο τρίγωνο (βήμα 2) ή πατώντας F5.



Εικόνα 11.11 *Το εργαλείο αποσφαλμάτωσης του Visual Studio Code.*

Η εφαρμογή έχει ξεκινήσει και, καθώς είναι εφαρμογή εξυπηρετητή, περιμένει κάποιον πελάτη για να στείλει την απάντησή της. Στο βήμα 3 στην **Εικόνα *11.11*** έχουμε ενεργοποιήσει ένα **breakpoint** με κλικ στη γραμμή 10. Αυτό σημαίνει πως το πρόγραμμα θα σταματήσει να εκτελείται ακριβώς πριν τη γραμμή 10 και θα μπορούμε να δούμε, για παράδειγμα, τις τιμές των μεταβλητών σε αυτό το σημείο.

Μόλις ανοίξουμε τη σελίδα μας στον φυλλομετρητή (http://127.0.0.1:3000), θα εκτελεστεί ο κώδικας μέχρι να φτάσει στο πρώτο breakpoint. Στο παράδειγμα της εικόνας βλέπουμε πως η εκτέλεση έχει σταματήσει προσωρινά στη γραμμή 10 (σημείο 1 στην εικόνα). Έχουμε στη διάθεσή μας διάφορα εργαλεία, τα πιο σημαντικά είναι το εργαλείο με το οποίο ελέγχουμε την εκτέλεση (σημείο 2) και το “Watch” όπου μπορούμε να προσθέσουμε τις μεταβλητές που μας ενδιαφέρουν οι τιμές τους (3).



Εικόνα 11.12 *Η εκτέλεση του κώδικα έχει σταματήσει προσωρινά στο breakpoint.*

11.9 Ερωτήσεις

1. Στη Node.js, παρ' όλο που δεν υπάρχει φυλλομετρητής, συνεχίζουμε να έχουμε πρόσβαση στις ιδιότητες των αντικειμένων **document** και **window**.
   – Σωστό/Λάθος
2. Οι παρακάτω προτάσεις περιέχουν ισχυρισμούς για τις διαφορές και ομοιότητες μεταξύ της JavaScript στη Node.js και της JavasScript στον φυλλομετρητή. Επιλέξετε όσους από τους ισχυρισμούς αληθεύουν.
   1. Ο φυλλομετρητής με τη χρήση της JavaScript μάς δίνει πρόσβαση στα τοπικά αρχεία του υπολογιστή μας.
   2. Ο βρόχος εκτέλεσης συμβάντων είναι ο ίδιος στη Node.js και στον φυλλομετρητή, αφού και στις δύο περιπτώσεις χρησιμοποιείται η μηχανή V8.
   3. Με την JavaScript στον φυλλομετρητή έχουμε δύο μηχανισμούς να φορτώσουμε βιβλιοθήκες, τον CommonJS και τον ES6.
   4. Στη Node.js, όπως και στον φυλλομετρητή, η JavaScript εκτελείται σε ένα νήμα [x].
   5. Τόσο στη Node.js όσο και στον φυλλομετρητή μπορούμε να γράψουμε ασύγχρονο κώδικα με συναρτήσεις επιστροφής ("callback") καθώς και με υποσχέσεις (το Promise API).
   6. Η Node.js δεν μας δίνει πρόσβαση στο Document Object Model (DOM) [x].
3. Για να χρησιμοποιήσουμε τις ενσωματωμένες (core) βιβλιοθήκες της Node.js δεν χρειάζεται να κάνουμε κάτι ιδιαίτερο πέρα από να τις φορτώσουμε.
   – Σωστό/Λάθος
4. Είναι προτιμότερο να χρησιμοποιούμε τη σύγχρονη εκδοχή της βιβλιοθήκης fs.
   – Σωστό/Λάθος
5. Οι ενσωματωμένες (core) βιβλιοθήκες της Node.js παρέχονται από τη μηχανή V8 και γι' αυτό είναι διαθέσιμες και στον φυλλομετρητή.
   – Σωστό/Λάθος
6. Για ποιο λόγο χρησιμοποιούμε το try/catch όταν χειριζόμαστε αρχεία;

1. Για να χειριστούμε σφάλματα που πιθανά προκύψουν κατά τον χειρισμό των αρχείων.
2. Είναι υποχρεωτική η χρήση του μπλοκ try/catch όταν χειριζόμαστε αρχεία.
3. Το try/catch δεν είναι δομή που μπορούμε να χρησιμοποιήσουμε με την JavaScript.

7. Με τις συναρτήσεις επιστροφής ο κώδικας δεν εκτελείται αναγκαστικά με τη σειρά που είναι γραμμένος.
   – Σωστό/Λάθος
8. Με τις υποσχέσεις (Promise API) μπορούμε να κατασκευάσουμε αλυσίδες όπου κάθε βήμα εκτελείται μόνο αν έχει ολοκληρωθεί το προηγούμενο.
   – Σωστό/Λάθος
9. Η Node.js μπορεί να χρησιμοποιήσει modules τόσο με τον μηχανισμό CommonJS όσο και με τον ES6.
   – Σωστό/Λάθος
10. Με τον μηχανισμό CommonJS χρησιμοποιούμε την **import** για να φορτώσουμε βιβλιοθήκες, ενώ με τον μηχανισμό ES6 χρησιμοποιούμε τη **require**.
    – Σωστό/Λάθος
11. Ένας τρόπος που έχουμε για να χρησιμοποιήσουμε το σύστημα ES6 για να φορτώσουμε βιβλιοθήκες είναι να αποθηκεύσουμε τα αρχεία μας με κατάληξη .mjs
    – Σωστό/Λάθος
12. Ένα πακέτο είναι ένας φάκελος που περιέχει ένα αρχείο JSON με όνομα **package.json**.
    – Σωστό/Λάθος
13. Στο αρχείο **package.json**, αν δώσουμε στο πεδίο “type” την τιμή “module”, τότε θα χρησιμοποιηθεί ο μηχανισμός ES6 και αναγκαστικά θα πρέπει να αποθηκεύσουμε το αρχείο μας με κατάληξη **.mjs**
    – Σωστό/Λάθος
14. Όλες οι εξαρτήσεις του πακέτου μας που περιγράφονται στο αρχείο **package.json** κατεβαίνουν και αποθηκεύονται στον υποφάκελο **node_modules**.
    – Σωστό/Λάθος
15. Αν διαγράψουμε τα περιεχόμενα του υποφακέλου **node_modules**, μπορούμε να τα κατεβάσουμε ξανά με την εντολή npm install.
    – Σωστό/Λάθος
16. Οι εξαρτήσεις ανάπτυξης (dev dependencies) δεν επηρεάζονται αν διαγράψουμε τον υποφάκελο **node_modules**, καθώς αποθηκεύονται σε άλλον φάκελο.
    – Σωστό/Λάθος
17. Επιλέξετε όσα από τα παρακάτω είναι συστατικά ενός αιτήματος HTTP:
    1. μέθοδος HTTP (method)
    2. μονοπάτι του πόρου (path)
    3. κωδικοποίηση χαρακτήρων (encoding)
    4. αριθμός έκδοσης του πρωτοκόλλου (protocol version)
    5. κεφαλίδες αιτήματος (request headers)
    6. σώμα του αιτήματος (request body)
    7. μηχανισμός module (commonjs/ES6)
18. Στη Node.js, με κάθε αίτημα HTTP που λαμβάνει η εφαρμογή μας, έχουμε διαθέσιμο το αντικείμενο **request** που περιέχει τις πληροφορίες του αιτήματος. Το αντικείμενο **response** με το οποίο θα στείλουμε την απάντηση δεν είναι διαθέσιμο και θα πρέπει να το κατασκευάσουμε.
    – Σωστό/Λάθος
19. Αν σε ένα αίτημα με τη μέθοδο GET έχουμε τιμές για περισσότερες από μια παραμέτρους, τότε το διαχωριστικό για να τις ξεχωρίζουμε είναι ο χαρακτήρας
    1. ;
    2. ,
    3. &
    4. +
20. Αν σε μια φόρμα δεν οριστεί η μέθοδος αποστολής, τότε η φόρμα
    1. δεν θα αποσταλεί.
    2. θα αποσταλεί με τη μέθοδο GET.
    3. θα αποσταλεί με τη μέθοδο POST.
21. Για να εγκαταστήσουμε ένα πακέτο με το npm έτσι ώστε να είναι ορατό σε όλο το σύστημά μας και ανεξάρτητα από το συγκεκριμένο πακέτο που αναπτύσσουμε, μπορούμε να γράψουμε

1. npm install <όνομα πακέτου>
2. npm install -g <όνομα πακέτου>
3. npm i <όνομα πακέτου>
4. npm g <όνομα πακέτου>

11.10 Βιβλιογραφία και Αναφορές

Ο αναγνώστης μπορεί να βρει λεπτομερείς οδηγίες χρήσης καθώς και σε βάθος περιγραφή των χαρακτηριστκών της Node.js στην επίσημη τεκμηρίωσή της. Πιο προσβάσιμοι οδηγοί για τη χρήση της Node.js υπάρχουν στη σελίδα nodejs.dev, που συντηρείται και ενημερώνεται από την κοινότητα.

Η ιστοσελίδα MDN αποτελεί και για αυτό το κεφάλαιο πολύτιμη πηγή. Έτσι, μπορεί κανείς να μελετήσει αναλυτικά τη χρήση του συστήματος βιβλιοθηκών ES6 Modules καθώς και τη χρήση του async/await.

Στον ιστότοπο του mathesis οι συγγραφείς έχουν δημοσιεύσει ένα διαδικτυακό μάθημα με τίτλο «**Ανάπτυξη διαδικτυακών εφαρμογών με Node.js**», στο περιεχόμενο του οποίου έχει εν μέρει βασιστεί και το υλικό που παρουσιάζεται σε αυτό και στα επόμενα κεφάλαια. https://mathesis.cup.gr/courses/course-v1:ComputerScience+CS3.3+22D/about

Α. Ξενόγλωσσες

Doglio, F. (2018). *REST API Development with Node.js*. Apress.

Herron, D. (2020). *Node.js Web Development* (5th ed.). Packt.

Mardan, A. (2018). *Practical Node.js, Building Real-World Scalable Web Apps* (2nd ed.). Apress.

Συλλογικό (2022). *Node.js API Reference Documentation*. Node.js. Ανακτήθηκε στις 14 Σεπτεμβρίου 2022 από https://nodejs.org/en/docs/

Β. Ελληνόγλωσσες

Σιντόρης, Χ., & Αβούρης, Ν. (2022). Ανάπτυξη διαδικτυακών εφαρμογών με Node.js. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr.

Κεφάλαιο 12. Ανάπτυξη εφαρμογών με το πακέτο Express.js

Σύνοψη

Στο κεφάλαιο αυτό θα εξετάσουμε το Express.js, ένα πακέτο της Node.js το οποίο παρέχει ένα απλό και ισχυρό πλαίσιο για την ανάπτυξη εφαρμογών διαδικτύου. Θα παρουσιάσουμε τον κύκλο επεξεργασίας ενός αιτήματος πελάτη προς τον εξυπηρετητή και τον τρόπο λειτουργίας της γραμμής επεξεργασίας (middleware pipeline) του Express.js, καθώς και τη δρομολόγηση και την πρόσβαση στα αντικείμενα αιτήματος και απόκρισης. Στη συνέχεια θα χρησιμοποιήσουμε τη μηχανή template Handlebars για την προετοιμασία της απάντησης.

Προαπαιτούμενη γνώση

Για τη μελέτη αυτού του κεφαλαίου είναι απαραίτητη η εξοικείωση με τη Node.js, που παρουσιάστηκε στο κεφάλαιο 11 καθώς και με την JavaScript (κεφάλαια 7, 8, 9 και 10). Επίσης, για την κατασκευή templates χρησιμοποιείται, σε βασικό επίπεδο μόνο, και η γλώσσα HTML που παρουσιάζεται στα κεφάλαια 2 και 3.

12.1 Εισαγωγή

Το Express.js είναι μια βιβλιοθήκη της Node.js που μας βοηθά να αναπτύξουμε εφαρμογές Node.js στην πλευρά του εξυπηρετητή. Αν και θα μπορούσαμε να χρησιμοποιήσουμε απευθείας τη Node.js για να γράψουμε τις εφαρμογές μας, ωστόσο οτιδήποτε ξεπερνά τα όρια ενός βασικού παραδείγματος γρήγορα μπορεί να μας οδηγήσει να γράφουμε κώδικα για εργασίες που είναι κοινές στις περισσότερες εφαρμογές διαδικτύου και για τις οποίες έχουν δοθεί λύσεις. Αυτές τις λύσεις μπορούμε να τις χρησιμοποιήσουμε έτοιμες ή, τουλάχιστον, μέσα σε ένα πλαίσιο που θα βοηθήσει στην ανάπτυξη και τη συντήρηση ή επέκταση της εφαρμογής μας.

Μερικές τέτοιες βασικές εργασίες με τις οποίες διευκολυνόμαστε είναι ενδεικτικά:

1. HTTP Request parsing, Cookie parsing
2. Session management
3. Request routing
4. Send HTTP Response
5. Error handling

12.2 Μια απλή εφαρμογή με το Express.js

Το Express.js είναι ένα module της Node.js και εγκαθίσταται με τον συνήθη τρόπο. Για να κατασκευάσουμε μια εφαρμογή με το Express.js, αρχικοποιούμε, αν δεν το έχουμε ήδη κάνει, το πρότζεκτ μας:

```
npm init -y
```

Στη συνέχεια εγκαθιστούμε το πακέτο express και τις εξαρτήσεις του:

```
npm install express
```

Σε ένα αρχείο με όνομα, για παράδειγμα, index.mjs γράφουμε μια υποτυπώδη εφαρμογή διαδικτύου:

```
import express  from 'express'
const app = express()

app.get('/', (req, res) => res.send('Γεια σου express!'))
app.listen(3000, () => console.log('Η εφαρμογή τρέχει'))
```

Ξεκινάμε την εφαρμογή με τον τρόπο που το κάνουμε για όλες τις εφαρμογές διαδικτύου στη Node.js, γράφοντας δηλαδή στο τερματικό node index.mjs ή προσθέτοντας το κατάλληλο σκριπτ στο package.json ή με το πακέτο nodemon, όπως είδαμε στο προηγούμενο κεφάλαιο. Η εφαρμογή μας τρέχει και μπορούμε να τη δούμε στον φυλλομετρητή στη σελίδα http://localhost:3000.

12.3 Ο κύκλος επεξεργασίας του αιτήματος

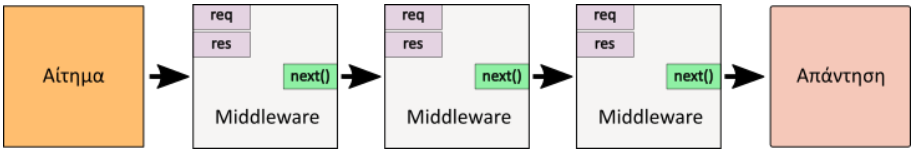
Το Express.js χρησιμοποιεί εκτεταμένα το middleware pattern. Με την προσέγγιση αυτή, κάθε αίτημα που δέχεται η εφαρμογή περνάει από διαδοχικά υποτμήματα (middleware), το ένα μετά το άλλο, που επεξεργάζονται το αίτημα μέχρι να σχηματιστεί και να αποσταλεί η απόκριση στον πελάτη. Μια εφαρμογή Express.js αποτελείται, λοιπόν, από επιμέρους υποτμήματα (τα middleware), τα οποία συντίθενται σε μια αλυσίδα επεξεργασίας (**Εικόνα *12.1***), που στο Express.js ονομάζεται "pipeline". Η αλυσίδα επεξεργασίας είναι μια απλή αλλά πολύ ισχυρή ιδέα. Καθώς πολλές εργασίες στις εφαρμογές διαδικτύου είναι κοινές σε μεγάλο πλήθος εφαρμογών, μπορούμε να χρησιμοποιήσουμε ένα μεγάλο πλήθος από έτοιμα πακέτα που παρέχουν εξειδικευμένα middleware.

12.3.1 Τα middleware του Express.js

Στη αλυσίδα επεξεργασίας κάθε υποτμήμα της (middleware) δέχεται δύο αντικείμενα, το **αντικείμενο αιτήματος** (request object), που περιέχει πληροφορίες για το αίτημα του πελάτη και το **αντικείμενο απόκρισης** (response object), που περιέχει την απόκριση που θα στείλει στο τέλος ο εξυπηρετητής στον πελάτη. Ένα middleware είναι μια απλή συνάρτηση της JavaScript που μπορεί να εκτελέσει οποιαδήποτε επεξεργασία, ακριβώς όπως μια συνήθης συνάρτηση. Μπορεί να διαβάσει και να επεξεργαστεί τα αντικείμενα του αιτήματος και της απόκρισης και, επιπλέον, επειδή είναι μέρος της αλυσίδας επεξεργασίας, όταν το υποτμήμα τελειώσει με την επεξεργασία του, πρέπει να κάνει ένα από τα δύο:

1. να δώσει τον έλεγχο στο επόμενο υποτμήμα της αλυσίδας επεξεργασίας ή
2. να τερματίσει την αλυσίδα επεξεργασίας στέλνοντας μια απόκριση στο αίτημα του πελάτη.

Αν δεν γίνει κάτι από τα δύο, π.χ. λόγω κάποιου σφάλματος, το αίτημα του πελάτη θα μείνει αναπάντητο και θα εκκρεμεί.



Εικόνα 12.1 *Η αλυσίδα επεξεργασίας του Express.js.*

Γενικότερα, ένα υποτμήμα της αλυσίδας επεξεργασίας

1. επεξεργάζεται και μεταβάλλει τα αντικείμενα request και response και διεκπεραιώνει κάποιες ενέργειες, για παράδειγμα,
    1. καταγραφή συμβάντων (logging),
    2. αυθεντικοποίηση (authentication),
    3. επεξεργασία του αιτήματος κ.λπ.
2. τερματίζει τη λειτουργία του με έναν από τους δύο τρόπους:
    1. προωθεί το request και το response object στο επόμενο υποτμήμα,
    2. αποκρίνεται στο αίτημα του πελάτη, δηλαδή τερματίζει την αλυσίδα επεξεργασίας.

Ένα middleware είναι μια συνάρτηση με τρία ορίσματα (`req`, `res`, `next`), π.χ.

```
const myMiddleware = (req, res, next) => {
    console.log('my middleware')
    next()
}
```

Οι παράμετροι ονομάζονται συμβατικά req, res, και next. Τα req και res είναι τα αντικείμενα του αιτήματος και της απάντησης, όπως ακριβώς και στη Node.js (Ενότητα 11.6). Η παράμετρος next είναι το επόμενο υποτμήμα στην αλυσίδα επεξεργασίας. Στο παράδειγμα, η `next()` είναι η συνάρτηση που το myMiddleware θα καλέσει όταν διεκπεραιώσει τον ρόλο του στην αλυσίδα επεξεργασίας.

12.3.2 Καθορισμός της αλυσίδας επεξεργασίας

Μπορούμε να καθορίσουμε τα ενδιάμεσα βήματα της αλυσίδας επεξεργασίας με τη συνάρτηση

app.use(middleware). Η σειρά με την οποία καλείται η app.use() καθορίζει και τη σειρά των τμημάτων στην αλυσίδα επεξεργασίας.

Στο παρακάτω παράδειγμα έχουμε κατασκευάσει δύο απλοϊκά υποτμήματα, που το μόνο που κάνουν είναι καταγράφουν ότι καλέστηκαν. Στη συνέχεια, με την app.use() προσδιορίζουμε τη σειρά με την οποία θα κληθούν στην αλυσίδα επεξεργασίας, πρώτο το myMiddleware2, έπειτα το myMiddleware1 και τέλος το myMiddleware3.

```
import express  from 'express'
const app = express()

const myMiddleware1 = (req, res, next) => {
    console.log('MW-1')
    next()
}

const myMiddleware2 = (req, res, next) => {
    console.log('MW-2')
    next()
}

const myMiddleware3 = (req, res, next) => {
    console.log('MW-3')
    res.send("Ok!")
}

app.use(myMiddleware2)
app.use(myMiddleware1)
app.use(myMiddleware3)
```

Η χρήση της app.use() στο πάνω παράδειγμα σημαίνει και ότι αυτά τα υποτμήματα θα εκτελεστούν σε **κάθε** αίτημα HTTP. Στο τερματικό θα εμφανιστούν τα μηνύματα με αυτή τη σειρά, ανεξάρτητα από τι ζήτησε ο χρήστης στο URL:

```
MW-2
MW-1
MW-3
```

Την ίδια σειρά επεξεργασίας μπορούμε εναλλακτικά να ορίσουμε περνώντας μια σειρά από υποτμήματα σαν παραμέτρους στην app.use():

```
app.use(myMiddleware2, myMiddleware1, myMiddleware3)
```

ή χρησιμοποιώντας, για ευκολία, έναν πίνακα:

```
const pipeline = [myMiddleware2, myMiddleware1, myMiddleware3]
app.use(pipeline)
```

Στο υποτμήμα myMiddleware3 τερματίζουμε την αλυσίδα επεξεργασίας με κλήση στη συνάρτηση res.send(). Η send() είναι συνάρτηση του αντικείμενου απόκρισης res η οποία αποστέλλει την απάντηση στον πελάτη και τερματίζει την τρέχουσα αλυσίδα επεξεργασίας. Αν δηλαδή μετακινήσουμε τη myMiddleware3 πιο πριν στην αλυσίδα επεξεργασίας, τα επόμενα υποτμήματα δεν θα εκτελεστούν, καθώς η αλυσίδα θα τερματίσει με την κλήση στη res.send().

Είναι φανερό πως θα ήταν πολύ χρήσιμη δυνατότητα η αλυσίδα επεξεργασίας να μπορεί να είναι διαφορετική για αιτήματα του χρήστη που γίνονται σε διαφορετική διαδρομή ή με διαφορετική μέθοδο. Η διαδικασία αυτή ονομάζεται δρομολόγηση (routing) και είναι πολύ ευέλικτη με το Express.js.

12.3.3 Δρομολόγηση

Η δρομολόγηση καθορίζει τον τρόπο που το Express.js θα αποκριθεί σε αιτήματα πελατών που γίνονται σε

διαφορετικές διαδρομές ή και με διαφορετικές μεθόδους HTTP. Μια διαδρομή (route) καθορίζεται, λοιπόν, από:

1. τη μέθοδο HTTP που χρησιμοποιήθηκε.
2. το μονοπάτι του αιτήματος που έλαβε η εφαρμογή μας.

Η δρομολόγηση μπορεί να γίνει με το αντικείμενο εφαρμογής του Express.js. Αυτό συνήθως ονομάζεται "app" και δημιουργείται με κλήση στον δημιουργό του Express.js: **const** app = express(). Η δρομολόγηση μπορεί όμως να γίνει και με την κλάση Router, που έχει σχεδόν τις ίδιες δυνατότητες δρομολόγησης με το αντικείμενο app και συνήθως προτιμάται, καθώς με τη Router έχουμε τη δυνατότητα να ορίσουμε πολλούς δρομολογητές και να έχουμε πιο σπονδυλωτό κώδικα. Ας δούμε αρχικά τη δρομολόγηση με το αντικείμενο εφαρμογής του Express, που είναι και η πιο απλή περίπτωση.

12.3.4 Προσάρτηση σε συγκεκριμένο μονοπάτι

Είναι πολύ χρήσιμο η εφαρμογή μας να αποκρίνεται με διαφορετικό τρόπο σε διαφορετικά μονοπάτια (path), π.χ. ένα αίτημα στο μονοπάτι "/login" να οδηγεί σε μια σελίδα όπου ο χρήστης θα δώσει τα στοιχεία ταυτοποίησης ή ένα αίτημα στο μονοπάτι "/getUserProfile/maria" να επιστρέψει το προφίλ του χρήστη "maria" κ.ο.κ., δηλαδή η αλυσίδα επεξεργασίας να διαφοροποιείται ανάλογα με το μονοπάτι.

Για να προσαρτήσουμε υποτμήματα σε συγκεκριμένα μονοπάτια, μπορούμε να περάσουμε το μονοπάτι σαν πρώτο όρισμα στην `app.use()`. Στο παρακάτω παράδειγμα έχουμε ορίσει δύο μονοπάτια, το "/path1" και το "/".

```
app.use("/path1", myMiddleware1)
app.use("/path2", myMiddleware2)
app.use(myMiddleware3)
```

Με τον τρόπο αυτό ορίσαμε δύο διαφορετικές αλυσίδες επεξεργασίας για τα δύο μονοπάτια:

1. myMiddleware1 → myMiddleware3, για αιτήματα στο μονοπάτι "/path1",
2. myMiddleware2 → myMiddleware3, για αιτήματα στο μονοπάτι "/path2".

12.3.5 Προσάρτηση σε συγκεκριμένη μέθοδο

Εκτός από το μονοπάτι, μπορούμε να προσδιορίσουμε σαν σημείο προσάρτησης και τη μέθοδο HTTP με την οποία γίνεται το αίτημα του πελάτη. Αν ξαναγράψουμε το προηγούμενο παράδειγμα και αντικαταστήσουμε την `use()` με την `get()`, τότε η αλυσίδα επεξεργασίας θα ισχύει μόνο για αιτήματα που έχουν φτάσει στον εξυπηρετητή με τη μέθοδο GET.

```
app.get("/path1", myMiddleware2)
app.get("/path2", myMiddleware1)
app.get(myMiddleware3)
```

Το Express.js διαθέτει συναρτήσεις για όλες τις διαθέσιμες μεθόδους HTTP, αν και οι get(), post(), put(), delete() είναι ίσως οι πιο συχνά χρησιμοποιούμενες. Γενικά λοιπόν, ο πιο απλός τρόπος δρομολόγησης είναι απευθείας στο αντικείμενο του Express.js:

```
app.METHOD(path, callback)
```

όπου

1. METHOD και path είναι η διαδρομή του αιτήματος
    1. METHOD είναι μια από τις μεθόδους HTTP (π.χ. GET, POST κ.λπ.),
    2. path είναι το μονοπάτι για το οποίο θα ισχύει ο δρομολογητής,
2. callback είναι η συνάρτηση που θα εκτελεστεί για τον συνδυασμό METHOD+path. Η συνάρτηση είναι πρακτικά η αρχή της αλυσίδας επεξεργασίας για την εκάστοτε διαδρομή METHOD+path.

12.3.6 Δρομολόγηση με τη μέθοδο app.route()

Δρομολόγηση με το αντικείμενο εφαρμογής του Express.js μπορεί να γίνει και με τη συνάρτηση `route()` του αντικείμενου εφαρμογής. Με τον τρόπο αυτό μπορούμε να ορίσουμε τον χειρισμό σε ένα συγκεκριμένο

μονοπάτι με ευανάγνωστο τρόπο, καθώς με τη route() ορίζουμε πρώτα το μονοπάτι και έπειτα όσες μεθόδους HTTP θέλουμε να προδιαγράψουμε για αυτό:

```
app.route('/recipe')
    .all((req, res) => {
        // το 'all' εκτελείται σε κάθε περίπτωση
    })
    .get((req, res) => {
        res.send('Επιστρέφει μια συνταγή')
    })
    .post((req, res) => {
        res.send('Προσθήκη της συνταγής')
    })
    .put((req, res) => {
        res.send('Ενημέρωση της συνταγής')
    })
```

12.3.7 Δρομολόγηση με την κλάση Router

Η δρομολόγηση με το αντικείμενο Router λειτουργεί όπως και με το αντικείμενο εφαρμογής του Express.js. Η διαφορά είναι πως το αντικείμενο Router υποστηρίζει μόνο τον μηχανισμό middleware και δρομολόγησης, ενώ το αντικείμενο εφαρμογής παρέχει περισσότερες λειτουργίες. Η δρομολόγηση με το αντικείμενο Router βοηθά στην οργάνωση όταν η εφαρμογή γίνεται πιο σύνθετη.

```
import express  from 'express'
const app = express()
const router = express.Router()

router.get('/random', function (req, res) {
  res.send('Επιστρέφει μια συνταγή')
})

router.get('/about', function (req, res) {
  res.send('Σχετικά με την εφαρμογή συνταγών')
})

app.use('/recipe', router)
```

Όπως βλέπουμε, η διαδρομή ορίζεται σε δύο βήματα. Πρώτα ορίζουμε τον δρομολογητή (router) και στη συνέχεια τον προσαρτάμε κάτω από ένα μονοπάτι. Στο τέλος, το παράδειγμά μας θα επιστρέφει μια τυχαία συνταγή σε αιτήματα που φτάνουν στο μονοπάτι “/recipe/random” και διάφορες πληροφορίες στο “/recipe/about”.

Σημειώστε πως θεωρούνται ίδιες οι διαδρομές

1. με ή χωρίς τελικό '/', π.χ. αίτημα στο /recipe είναι το ίδιο με /recipe/,
2. με μικρά ή κεφαλαία, π.χ. /recipe το ίδιο με /Recipe.

Η συμπεριφορά αυτή αλλάζει με τις αντίστοιχες ιδιότητες του αντικείμενου ρύθμισης του Router, strict και caseSensitive:

```
const options = {
    caseSensitive: true,
    strict: true
}
const router = express.Router(options)
```

12.4 Τα αντικείμενα αιτήματος και απόκρισης

Έχουμε δει μέχρι τώρα πώς να δρομολογήσουμε ένα αίτημα στην κατάλληλη αλυσίδα επεξεργασίας. Για να μπορέσουμε να επεξεργαστούμε κατάλληλα το αίτημα στα υποτμήματα της αλυσίδας επεξεργασίας,

χρειαζόμαστε πληροφορίες σχετικά με αυτό, που μας τις δίνει το αντικείμενο αιτήματος (request). Αντίστοιχα, μπορούμε να χρησιμοποιήσουμε το αντικείμενο απόκρισης (response) για να προδιαγράψουμε την απόκρισή μας στο αίτημα του πελάτη.

12.4.1 Το αντικείμενο αιτήματος

Το API του αντικείμενου Request μάς δίνει πρόσβαση στα δεδομένα του αιτήματος (Request API). Ίσως η πιο χρήσιμη ιδιότητα του αντικείμενου είναι η **request.query**, ένας πίνακας που περιέχει τις παραμέτρους που έστειλε ο πελάτης στο αίτημά του μέσω του query string.

Μπορούμε να δούμε τη βασική χρήση με ένα απλό παράδειγμα για την υποθετική εφαρμογή μας TastyRecipes. Έστω πως θέλουμε η εφαρμογή να επιστρέφει συνταγές με βάση χαρακτηριστικά που προσδιορίζει ο χρήστης στο αίτημά του, όπως π.χ. δυσκολία της συνταγής ή βαθμολογία:

```
import express from 'express'
const app = express()

app.get('/recipe/', function (req, res, next) {
    for (const key in req.query) {
        console.log(key, ":", req.query[key])
        next()
    }
    res.send("got a recipe.")
})

app.listen(3000, () => console.log('Ready'))
```

Ένα αίτημα όπως το http://localhost:3000/recipe?difficulty=easy&rating=5 θα τυπώσει στο τερματικό του εξυπηρετητή τα ονόματα των παραμέτρων (key) καθώς και την τιμή της καθεμίας (res.query[key]):

```
Ready
difficulty : easy
rating : 5
```

12.4.2 Ονοματισμένες παράμετροι

Ένας εναλλακτικός τρόπος να περάσουμε παραμέτρους με το URL του αιτήματος είναι να χρησιμοποιήσουμε τον μηχανισμό των ονοματισμένων παραμέτρων (named parameters). Οι τιμές των παραμέτρων θα είναι διαθέσιμες στο αντικείμενο request.parameters. Μπορούμε να ξαναγράψουμε το πιο πάνω παράδειγμα ώστε να αποκρίνεται σε αιτήματα σε URL της μορφής /recipe/<difficulty>/<rating>:

```
app.get('/recipe/difficulty/:difficulty/rating/:rating', function (req,
res, next) {
    console.log(req.params)
    res.send("got a recipe.")
})
```

Η κλήση στο URL http://localhost:3000/recipe/difficulty/easy/rating/5 θα έχει σαν αποτέλεσμα:

```
{ difficulty: 'easy', rating: '5' }
```

12.4.3 Πρόσβαση στα δεδομένα POST

Αιτήματα που έχουν φτάσει με τη μέθοδο POST, όμως, δεν μεταφέρουν την πληροφορία με το query string, αλλά περιέχεται στο σώμα του αιτήματος. Για να έχουμε πρόσβαση στα δεδομένα που έχουν σταλεί με τη μέθοδο POST, π.χ. από την υποβολή μιας φόρμας, χρειάζεται να χρησιμοποιήσουμε το middleware urlencoded, που είναι ενσωματωμένο στη βασική εγκατάσταση του Express.js:

```
app.use(express.urlencoded({ extended: true }))
```

```
app.post('/', (req, res) => {
    console.log("request body:", req.body)
    res.send("post ok")
})
```

12.4.4 Το αντικείμενο απόκρισης

Όταν τελειώσει η επεξεργασία του αιτήματος, κάποιο από τα υποτμήματα της αλυσίδας επεξεργασίας θα πρέπει να στείλει την απόκριση στον πελάτη. Η αποστολή της απόκρισης μπορεί να γίνει με διάφορες συναρτήσεις του αντικείμενου απόκρισης (response).

1. Response.send(message) Υπολογίζει το μέγεθος της απάντησης (Content-Length), τη στέλνει και σταματά την αλυσίδα επεξεργασίας.
2. Response.end() Στέλνει κενή απάντηση, χωρίς δεδομένα, και σταματά την αλυσίδα επεξεργασίας.
3. Response.status() Ορίζει την κεφαλίδα κατάστασης της απάντησης, π.χ. res.status(404).end()

Για να απαντήσουμε με κώδικα HTML, ο πιο απλός αλλά και κουραστικός τρόπος είναι να στείλουμε τον κώδικά μας με τη Response.send(), π.χ.:

```
app.get('/', (req, res) => {
    res.send('<html><head><meta charset="utf-8">.....')
})
```

Η πρακτική αυτή όμως είναι εμφανώς δύσχρηστη και για αυτό τον λόγο χρησιμοποιούμε μηχανές template (Ενότητα 12.5) για την προετοιμασία και αποστολή της απόκρισής μας, όπως θα δούμε στη συνέχεια. Για να αποστείλουμε την απάντησή μας μέσω μιας μηχανής template χρησιμοποιούμε τη συνάρτηση Response.render().

Τέλος, πολύ χρήσιμη είναι η δυνατότητα να αποστείλουμε απάντηση σε μορφή JSON, π.χ αν χρησιμοποιούμε το Express.js για να υλοποιήσουμε ένα REST API που θα είναι προσβάσιμο όχι από χρήστες αλλά από άλλες εφαρμογές. Για να στείλουμε την απάντηση σε μορφή JSON χρησιμοποιούμε τη συνάρτηση res.json(message)

12.5 Μηχανές template

Στα απλοϊκά παραδείγματα που έχουμε δει μέχρι τώρα η απόκριση της εφαρμογής μας στον χρήστη γίνεται με τη συνάρτηση Response.send(). Με τις μηχανές για template έχουμε απλές γλώσσες για τη δυναμική δημιουργία HTML.

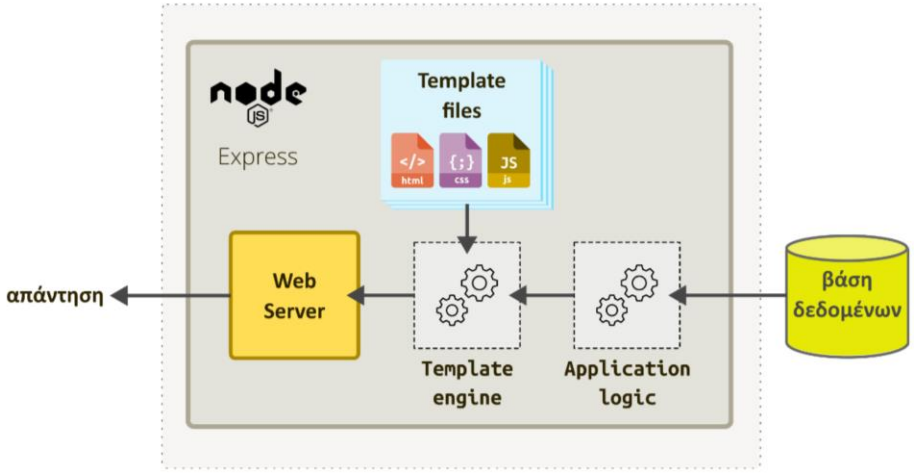
Μια μηχανή template συνδυάζει κάποια δεδομένα JavaScript ή JSON με κάποιο template για να παραγάγει HTML, το οποίο και τελικά θα επιστραφεί στον πελάτη.

Η επίσημη μηχανή για templates στο Express.js είναι η Pug. Αξίζει να σημειωθεί πως το παλιό της όνομα είναι Jade, και υπάρχουν ακόμη αναφορές σε αυτό το όνομα. Μια από τις πιο δημοφιλείς μηχανές, όμως, είναι η Handlebars.js, και αυτή είναι που θα δούμε στη συνέχεια καθώς είναι πιο απλή.

12.5.1 Βασική χρήση της Handlebars

Η χρήση μιας μηχανής template μάς επιτρέπει να διαχωρίζουμε τη λογική της εφαρμογής από την παρουσίαση της απόκρισης στον χρήστη. Η απόκρισή μας προετοιμάζεται στο κυρίως μέρος της εφαρμογής και, πριν σταλεί στον πελάτη, η μηχανή template αναλαμβάνει να την εμπλουτίσει με την κατάλληλη HTML.

Στην πραγματικότητα, η μηχανή template αναλαμβάνει να συνθέσει και να αποστείλει στον πελάτη μια σελίδα HTML που να περιέχει το αποτέλεσμα της επεξεργασίας του αιτήματός του (**Εικόνα *12.2***). Έχουμε δει στα προηγούμενα κεφάλαια πως η παρουσίαση πληροφορίας με τις τεχνολογίες HTML, CSS και JavaScript μπορεί γρήγορα να γίνει σύνθετη υπόθεση. Η μηχανή template Handlebars και γενικότερα όλες οι μηχανές template, σε διαφορετικό βαθμό η καθεμία, παρέχουν αρκετά εργαλεία ώστε ο κώδικας που συνθέτει την απόκρισή μας, εκτός από αποτελεσματικός, να είναι σπονδυλωτός και να μπορεί να συντηρηθεί και να επεκταθεί εύκολα. Τέτοια εργαλεία περιλαμβάνουν τη δυνατότητα να χρησιμοποιήσουμε διαφορετικά layout, π.χ. ένα layout για σελίδες που θα προβάλλονται σε αυθεντικοποιημένους χρήστες, άλλο για τους απλούς επισκέπτες, άλλο ίσως για τον διαχειριστή της εφαρμογής κ.ο.κ., ή τη δυνατότητα να συνθέσουμε τη σελίδα μας από διάφορα επιμέρους αποσπάσματα template κ.λπ.



Εικόνα 12.2 *Η μηχανή template συνθέτει την τελική απάντηση πριν αποσταλεί στον πελάτη.*

Στη συνέχεια θα δούμε τις βασικές δυνατότητες που παρέχει η μηχανή Handlebars, μέσα από ένα υποθετικό πρότζεκτ.

Η βασική δομή των αρχείων του πρότζεκτ μας θα είναι:

```
|-- index.mjs
|-- views/
|   |-- home.hbs
|   |-- layouts/
|   |   |-- main.hbs
|   |-- partials/
|   |   |-- header.hbs
|   |   |-- footer.hbs
```

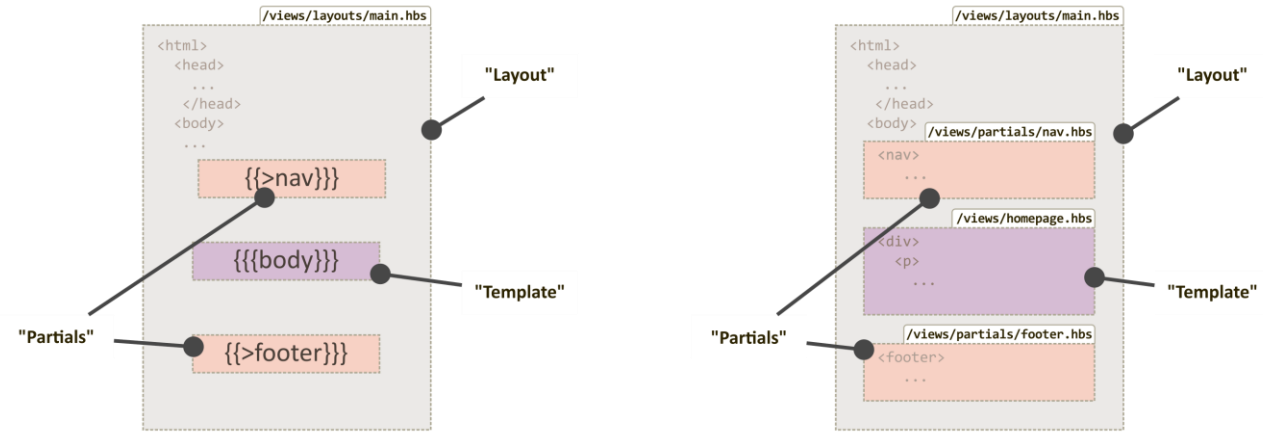
όπου

1. '/views' είναι η θέση των template,
2. '/layouts/main.hbs' το κύριο layout, μέσα στο οποίο προβάλλονται τα template,
3. '/partials/*' επιμέρους αποσπάσματα template, για να φορτώνονται από άλλα template.

Για τη μηχανή Handlebars χρησιμοποιούμε το πακέτο express-handlebars (npm install express-handlebars). Στη βασική μας εφαρμογή (index.mjs) ορίζουμε ότι θα χρησιμοποιήσουμε τη Handlebars και ότι τα templates μας θα είναι σε αρχεία με κατάληξη '.hbs', αντί για τη προκαθορισμένη κατάληξη .handlebars:

```
import express from 'express';
import { engine } from 'express-handlebars';

const app = express()

app.engine('.hbs', engine({ extname: '.hbs' }));
app.set('view engine', '.hbs');
```

Η συνάρτηση `app.engine()` ορίζει ποια θα είναι η μηχανή template της εφαρμογής μας, δηλαδή εκείνη που θα καλείται με τη μέθοδο `res.render()` όταν θέλουμε να στείλουμε την απάντησή μας. Η συνάρτηση `app.set()` είναι μια βοηθητική συνάρτηση του Express.js με την οποία μπορούμε να προσδιορίσουμε την τιμή διαφόρων παραμέτρων, στην προκειμένη περίπτωση την κατάληξη των αρχείων της μηχανής template που χρησιμοποιούμε.



Εικόνα 12.3 *Στην ορολογία του Handlebars ένα template «τοποθετείται» μέσα σε ένα "view". Το template μπορεί να συνοδεύεται από επιμέρους "partial" τμήματα.*

Το template μας μπορεί να χωριστεί σε πολλά τμήματα. Το default layout '/views/layouts/main.hbs', περιέχει το βασικό layout μέσα στο οποίο θα προβληθεί το template, δηλαδή τον κώδικα HTML που θα πλαισιώνει όλες τις σελίδες της εφαρμογής μας (**Εικόνα *12.3***):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Tasty Recipes</title>
</head>
<body>
    {{{body}}}
</body>
</html>
```

Το εκάστοτε template θα προβληθεί στη θέση {{{body}}}. Για παράδειγμα, το template '/views/home.hbs' μπορεί να είναι πολύ απλό:

```
<main><p> Οι νόστιμες συνταγές: {{greeting}} </p></main>
```

Αλλά συνήθως χρειάζεται να μεταφέρουμε κάποια δεδομένα που θέλουμε να προβάλει η μηχανή template στην HTML. Μπορούμε να εμφανίσουμε τιμές μεταβλητών απλά ονοματίζοντάς τες σε διπλά άγκιστρα, όπως φαίνεται πιο πάνω, και να περάσουμε τις τιμές σαν παράμετρο στη συνάρτηση res.render():

```
app.get('/', (req, res) => {
  res.render('home', { greeting: 'Καλώς ήλθατε!'});
});
```

Η res.render() λοιπόν θα χρησιμοποιήσει το template './views/home.hbs', καθώς το πρώτο όρισμά της είναι το home. Επίσης, θα περάσει στο template αυτό το αντικείμενο του δεύτερου ορίσματος, το { greeting: 'Καλώς ήλθατε!'}. Το αποτέλεσμα θα προστεθεί στο default layout (που είναι το './views/layouts/main.hbs') και θα επιστραφεί τελικά στον πελάτη.

Σημειώστε εδώ πως το express-handlebars έχει προκαθορισμένη συμπεριφορά να αναζητά τα template στον φάκελο με όνομα /views και να χρησιμοποιεί το layout /views/layouts/main.hbs για να προβάλλει όλα τα άλλα template, όπως π.χ. το home.hbs. Δεν χρειάστηκε λοιπόν να τα προσδιορίσουμε αυτά στον κώδικά μας, αν και μπορούμε αν θέλουμε να αλλάξουμε αυτή τη συμπεριφορά.

Εκτός από το main.hbs, στον φάκελο './views/layouts/' μπορούμε να έχουμε και άλλα layout, που να αποτελούν τον βασικό σκελετό κάποιων από τις σελίδες μας:

```
|-- views
|   |-- layouts
|   |   |-- main.hbs
|   |   |-- other-layout.hbs
```

Για παράδειγμα, στον παρακάτω κώδικα το αίτημα στη διαδρομή /other-layout θα απαντηθεί κάνοντας render το template './views/home.hbs' στο layout './views/layouts/other-layout.hbs':

```
app.get('/other-layout', (req, res) => {
    res.render('home', { layout: "other-layout" })
})
```

Partials

Γρήγορα όμως θα χρειαστεί σε ένα πρότζεκτ να κατασκευάσουμε σύνθετες σελίδες, που αποτελούνται από επιμέρους template. Αυτό είναι πολύ συχνό, για παράδειγμα, σε ιστότοπους που το υποσέλιδό τους είναι πάντα το ίδιο, αυτό που συχνά ονομάζεται footer. Αντίστοιχα χρήσιμο είναι να έχουμε σε ξεχωριστό αρχείο τη γραμμή πλοήγησης ή την κεφαλίδα (header) με τον τίτλο και το λογότυπο του ιστότοπού μας.

Ο μηχανισμός των partials επιτρέπει να φορτώσουμε templates μέσα από άλλα templates. Στη δομή αρχείων του παραδείγματός μας έχουμε ορίσει το partial '/views/partials/footer.hbs', που μπορεί να περιέχει το εξής:

```
<footer><p> Το υποσέλιδο </p></footer>
```

Με αυτό τον τρόπο μπορούμε να επαναχρησιμοποιήσουμε τον κώδικα του footer.hbs σε πολλά διαφορετικά layout. Στο παράδειγμά μας έχουμε ήδη δύο layout, το main.hbs και το other-layout.hbs. Για να φορτώσουμε το partial στο layout μας, π.χ. στο main.hbs, γράφουμε:

```
...
{{> footer}}
</body>
```

12.5.2 Το αντικείμενο res.locals

Πολύ βολικό είναι το αντικείμενο res.locals. Ό,τι καταχωρίσουμε σε αυτό το αντικείμενο θα είναι διαθέσιμο στη μηχανή template χωρίς άλλες ενέργειες. Για παράδειγμα:

```
app.get("/", (req, res, next) => {
    res.locals.greeting = "Καλημέρα";
    next()
    }, (req, res) => {
        // δεν χρειάζεται να περάσουμε ρητά τη μεταβλητή "greeting"
        res.render('home')
    })
```

Τώρα στο template “home” μπορούμε απλά να γράψουμε

```
<main><p> Οι νόστιμες συνταγές: {{greeting}} </p></main>
```

Στατικά αρχεία

Πώς όμως θα διαθέσουμε στους πελάτες του εξυπηρετητή αρχεία που δεν χρειάζονται επεξεργασία; Τέτοια αρχεία είναι, για παράδειγμα, εικόνες, τα αρχεία CSS ή αρχεία με JavaScript που εκτελείται στον φυλλομετρητή. Αυτά ονομάζονται **στατικά αρχεία** και για να τα διαθέσουμε χρησιμοποιούμε άλλο ένα από τα ενσωματωμένα middleware του Express.js.

Το middleware static είναι αυτό που μας δίνει πρόσβαση σε στατικά αρχεία. Για να το χρησιμοποιήσουμε ορίζουμε έναν φάκελο που περιέχει στατικά αρχεία (ή και περισσότερους):

```
// Τα στατικά αρχεία μας θα βρίσκονται στον φάκελο /public
app.use(express.static('public'))
```

Όλα τα αρχεία που βρίσκονται στον φάκελο “public” θα είναι προσβάσιμα απευθείας, π.χ.

```
|-- public
|   |-- images
|       |-- mypic.png
|   |-- css
|       |-- mystyle.css
...
```

Τα αρχεία της παραπάνω δομής είναι προσβάσιμα από URL της μορφής:

1. http://localhost:3000/images/mypic.png
2. http://localhost:3000/images/mystyle.css

Προσέξτε πως στο URL έχουμε παραλείψει το μονοπάτι όπου φιλοξενούνται τα αρχεία, δηλαδή το public/.

12.6 Βοηθητικά εργαλεία της Handlebars

Η μηχανή Handlebars παρέχει και κάποια βοηθήματα (helpers) που μας δίνουν κάποια ευελιξία στη συγγραφή των template, όπως π.χ.:

1. #if:

```
{{#if greeting}}
    {{greeting}}
{{else}}
    Αντίο
{{/if}}
```

1. #unless: το αντίθετο του #if
2. #each: απαρίθμηση των στοιχείων ενός πίνακα

```
{{#each people}}
    <li>{{this}}</li>
{{/each}}
```

12.7 Handlebars - Expressions

Οι εκφράσεις περικλείονται από διπλά άγκιστρα. Η έκφραση {{greeting}}, όπως είδαμε πιο πάνω, θα αντικατασταθεί με την τιμή της μεταβλητής greeting. Αν, αντί για μεταβλητές, έχουμε αντικείμενα, μπορούμε να έχουμε πρόσβαση στις ιδιότητές τους με dot-notation, π.χ. για το αντικείμενο address που θα μπορούσαμε να περάσουμε στο template:

```
res.render( 'home', {   address: {  street: "Ypatias 1", city: "Patra" }
});
```

θα έχουμε διαθέσιμες τις τιμές του αντικείμενου address στη Handlebars:

```
<p>{{address.street}} {{address.city}}</p>
```

και

```
{{#with address}}
<p> {{street}} {{city}}</p>
{{/with}}
```

ή

```
{{#each address}}
<p> {{@key}}: {{this}}</p>
{{/with}}
```

12.8 Ερωτήσεις

1. Το Express.js είναι
    1. μια εξελιγμένη έκδοση της Node.js.
    2. ένα πακέτο της Node.js.
    3. μια αρχιτεκτονική που μπορούμε να χρησιμοποιήσουμε για να γράψουμε καλά δομημένα την εφαρμογή μας.
    4. ένας εναλλακτικός τρόπος να γράψουμε ασύγχρονες εφαρμογές στη Node.js.
2. Στη 2η γραμμή του παρακάτω αποσπάσματος κώδικα

```
import express  from 'express'
const app = express()
```

    1. η μεταβλητή app είναι το αντικείμενο εφαρμογής (application object) του Express.js.
    2. θα προκληθεί σφάλμα, καθώς η αρχικοποίηση ενός νέου στιγμιότυπου απαιτεί τη λέξη-κλειδί new.
    3. θα προκληθεί σφάλμα, καθώς δεν επιτρέπεται να χρησιμοποιηθεί η λέξη-κλειδί const με αυτό τον τρόπο.
    4. θα έπρεπε να γράψουμε const app = express.express()
3. Σε έναν χειριστή αιτήματος στο Express.js η εντολή res.send()
    1. είναι ισοδύναμη με την ακολουθία στη Node.js: res.write();res.end()
    2. θα προκαλέσει σφάλμα.
    3. είναι ισοδύναμη με την ακολουθία στη Node.js: res.writeHead();res.end()
4. Ένα υποτμήμα της αλυσίδας επεξεργασίας είναι
    1. μια συνάρτηση με ορίσματα τα αντικείμενα αιτήματος και απόκρισης.
    2. μια κλάση τύπου Express, που περιέχει μια συνάρτηση με ορίσματα τα αντικείμενα αιτήματος και απόκρισης.
    3. μια συνάρτηση που είναι μέλος αντικείμενου εφαρμογής (application object) και που έχει ορίσματα τα αντικείμενα αιτήματος και απόκρισης.
5. Επιλέξετε όσα ισχύουν: Ένα υποτμήμα της αλυσίδας επεξεργασίας τερματίζει τη λειτουργία του με τους εξής τρόπους:
    1. Προωθεί το request και το response object στο επόμενο υποτμήμα με τη next()
    2. Επιστρέφει τον έλεγχο ροής στο αντικείμενο εφαρμογής του Express.js με τη return.
    3. Αποκρίνεται στο αίτημα του πελάτη.
    4. Επιστρέφει τον έλεγχο ροής στο αντικείμενο εφαρμογής του Express.js με κλήση σε συνάρτηση επιστροφής.
6. Μια διαδρομή (route) καθορίζεται από:
    1. τη μέθοδο HTTP (HTTP method).
    2. το μονοπάτι του αιτήματος (path).
    3. τη θύρα του εξυπηρετητή (server port).
    4. την αλυσίδα επεξεργασίας που θα χρησιμοποιηθεί.
    5. τον μηχανισμό βιβλιοθηκών που χρησιμοποιούμε (commonjs/ES6).
7. Έστω η παρακάτω αλυσίδα επεξεργασίας

```
app.get("/path1", MW1)
app.use(MW2)
app.get("/", MW3)
```

    Επιλέξετε όσα ισχύουν:
    1. με αίτημα στο μονοπάτι “/path1”, η αλυσίδα επεξεργασίας είναι MW1->MW2->MW3
    2. με αίτημα στο μονοπάτι “/path1”, η αλυσίδα επεξεργασίας είναι MW1->MW2
    3. με αίτημα στο μονοπάτι “/path1”, η αλυσίδα επεξεργασίας είναι MW1->MW3
    4. με αίτημα στο μονοπάτι “/”, η αλυσίδα επεξεργασίας είναι MW3
    5. με αίτημα στο μονοπάτι “/”, η αλυσίδα επεξεργασίας είναι MW2->MW3
8. Επιλέξτε όσα από τα παρακάτω αποτελούν έγκυρους τρόπους να δηλωθεί η αλυσίδα επεξεργασίας:
    1. app.use(path, callback)
    2. app.get(path, callback)

3. app.get(callback).post(callback)
4. app.route(path).all(callback)
5. app.route(path).get(callback).post(callback)

9. Η ιδιότητα request.query του αντικείμενου του αιτήματος HTTP περιέχει της παραμέτρους του αιτήματος HTTP όταν αυτό γίνεται με τη μέθοδο GET είτε με την POST.
   – Σωστό/Λάθος
10. Η ιδιότητα request.parameters του αντικείμενου του αιτήματος HTTP περιέχει τις παραμέτρους του αιτήματος HTTP όταν αυτό γίνεται με τη μέθοδο GET είτε με την POST.
   – Σωστό/Λάθος
11. Με ποιον από τους παρακάτω τρόπους μπορούμε να αποκτήσουμε πρόσβαση στα δεδομένα του αιτήματος στο παρακάτω τμήμα;

```
app.get('/name/:name/age:age', function (req, res, next) {
    --> Επιλέξτε τι θα μπει εδώ για να τυπωθούν οι παράμετροι και οι τιμες
τους <--
    res.send("ok")
})
```

1. console.log(req.params)
2. console.log(req.body)
3. console.log(req.query)

12. Με τη response.status(404) ορίζουμε την κεφαλίδα απόκρισης και τη στέλνουμε στον πελάτη.
   – Σωστό/Λάθος
13. Επιλέξετε με ποιους από τους παρακάτω τρόπους μπορούμε να στείλουμε μια απάντηση στον πελάτη και να τερματίσουμε την αλυσίδα επεξεργασίας.
   1. res.send()
   2. res.end()
   3. res.status()
   4. res.render()
   5. res.json()
14. Το αντικείμενο **Router** χρησιμοποιείται όπως και οποιοδήποτε middleware.
   – Σωστό/Λάθος
15. Όταν η εφαρμογή είναι σύνθετη, προτιμάμε την οργάνωση των διαδρομών με (επιλέξτε το σωστό)
   1. app.METHOD(path, callback, όπου method οποιοδήποτε από τα ρήματα του HTTP (get, post κ.λπ.).
   2. app.route(path).METHOD(callback).
   3. app.route(path, router), όπου router ένα αντικείμενο τύπου Router.
16. Η προκαθορισμένη συμπεριφορά στη δρομολόγηση με το Express.js είναι:
   1. Όπως και στη Node.js.
   2. Έχει σημασία αν τα γράμματα του μονοπατιού είναι πεζά η κεφαλαία, αλλά δεν μας ενδιαφέρει αν υπάρχει ή όχι τελικό "/".
   3. Έχει σημασία αν υπάρχει ή όχι τελικό "/", αλλά δεν διακρίνουμε μικρά ή κεφαλαία γράμματα.
   4. Δεν έχει σημασία το τελικό "/", ούτε γίνεται διάκριση πεζών και κεφαλαίων γραμμάτων.
17. Όταν λέμε στατικά αρχεία εννοούμε αρχεία τα οποία το Express.js επιστρέφει όπως είναι, χωρίς επεξεργασία από κάποιο ειδικό υποτμήμα.
   – Σωστό/Λάθος
18. Σε μια εφαρμογή Express.js μπορούμε να ορίσουμε μόνο έναν φάκελο που θα περιέχει τα στατικά μας αρχεία.
   – Σωστό/Λάθος
19. Τα στατικά μας αρχεία μπορούν να βρίσκονται μόνο σε φάκελο με όνομα static ή public.
   – Σωστό/Λάθος
20. Έστω η εφαρμογή μας "http://localhost/" και ότι έχουμε γράψει:

```
import express  from 'express'
const app = express()
app.use(express.static('/public'))
```

Τότε, για να έχουμε πρόσβαση στο αρχείο css, που βρίσκεται π.χ. στο /public/mystyle.css, θα κάνουμε αίτημα στο

1. http://localhost/static/public/mystyle.css
2. http://localhost/public/mystyle.css
3. http://localhost/mystyle.css

21. Έστω η εφαρμογή μας “http://localhost/” και ότι έχουμε γράψει:

```
import express  from 'express'
const app = express()
app.use(express.static('/public'))
```

Τότε, για να φορτώσουμε από την HTML το αρχείο css, που βρίσκεται π.χ. στο /public/mystyle.css, θα γράψουμε στην περιοχή <head>

1. <link rel="stylesheet" href="/static/public/mystyle.css">
2. <link rel="stylesheet" href="/public/mystyle.css">
3. <link rel="stylesheet" href="/mystyle.css">

22. Με την παρακάτω εντολή μπορούμε να χρησιμοποιήσουμε τη μηχανή template για να κατασκευάσει και να στείλει την απάντησή μας στον πελάτη:
    1. res.engine()
    2. res.send()
    3. res.render()
    4. res.end()
    5. res.template()
23. Για να προσδιορίσουμε ποια μηχανή template,
    1. καλούμε τη μέθοδο app.enable()
    2. καλούμε τη μέθοδο app.set()
    3. καλούμε τη μέθοδο app.engine()
    4. δεν χρειάζεται να κάνουμε κάτι ιδιαίτερο, καθώς το Express.js υποστηρίζει εγγενώς τη χρήση μηχανών template.
24. Στο layout μας, για να υποδείξουμε το σημείο που θέλουμε να προβληθεί το template, χρησιμοποιούμε την παρακάτω σύνταξη:
    1. `{{{body}}}`
    2. `{{{template}}}`
    3. `{{{view}}}`
    4. `{{{hbs}}}`
25. Το προκαθορισμένο layout είναι αποθηκευμένο στο αρχείο
    1. '/views/layouts/main.hbs'
    2. '/views/main.hbs'
    3. '/views/layout.hbs'
    4. '/views/templates/main.hbs'
26. Στο handlebars, το template μας περικλείεται από ένα layout.
    – Σωστό/Λάθος
27. Το layout μέσα στο οποίο περιέχεται το template πρέπει να είναι υποχρεωτικά αποθηκευμένο σε αρχείο με όνομα /views/layouts/main.hbs
    – Σωστό/Λάθος
28. Το βοήθημα `{{#if}}` θα επιστρέψει true αν
    1. στην έκφραση {{#if property value}} ισχύει property==value
    2. στην έκφραση {{#if property==value}} ισχύει property==value
    3. στην έκφραση {{#if property}} το property είναι διάφορο των false, undefined, null, 0 ή []
29. Το βοήθημα `{{#each}}` μπορεί να διατρέξει τα στοιχεία ενός πίνακα και να τυπώσει τη θέση, το όνομα και την τιμή κάθε στοιχείου με:
    1. `{{#each object}} {{@index}}: {{@key}}: {{this}}  {{/each}}`
    2. `{{#each object}} {{index}}:  {{key}}:  {{value}}  {{/each}}`
    3. `{{#each object}} {{@index}}: {{@key}}: {{@value}} {{/each}}`

12.9 Βιβλιογραφία και Αναφορές

Όπως και στο κεφάλαιο για τον προγραμματισμό στο περιβάλλον της Node.js (κεφ. 11), έτσι και εδώ το περιεχόμενο αφορά την παρουσίαση και χρήση μιας τεχνολογίας. Συνεπώς η κύρια πηγή πληροφοριών και η πρώτη στάση για περαιτέρω εμβάθυνση στο θέμα είναι και εδώ η επίσημη ιστοσελίδα του Express.js. Εκεί μπορεί κανείς να βρει την αναλυτική τεκμηρίωση του API, οδηγούς για χρήστες που αποκτούν την πρώτη επαφή, αλλά και άρθρα που αφορούν εξειδικευμένα θέματα και που ενδιαφέρουν τους προχωρημένους χρήστες.

Η κύρια πηγή και για τη μηχανή Handlebars είναι ο οδηγός που βρίσκεται στην επίσημη ιστοσελίδα. Όπως και για το Express.js, ο οδηγός συνοδεύεται και από την τεκμηρίωση της βιβλιοθήκης της Handlebars.

A. Ξενόγλωσσες

Συλλογικό (2022). *Express.js4.x API Reference*. Express.js. Ανακτήθηκε στις 14 Σεπτεμβρίου 2022 από https://expressjs.com/en/4x/api.html

Συλλογικό (2022). *Node.js API Reference Documentation*. Node.js. Ανακτήθηκε στις 14 Σεπτεμβρίου 2022 από https://nodejs.org/en/docs/

B. Ελληνόγλωσσες

Σιντόρης, Χ., & Αβούρης, Ν. (2022). Ανάπτυξη διαδικτυακών εφαρμογών με Node.js. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr.

Κεφάλαιο 13. Σύνδεση του εξυπηρετητή με βάση δεδομένων

Σύνοψη

Στο κεφάλαιο αυτό θα εξετάσουμε τη διασύνδεση του εξυπηρετητή node.js με μόνιμη αποθήκευση πληροφορίας, δηλαδή με μια βάση δεδομένων.

Όπως έχουμε αναφέρει στο εισαγωγικό κεφάλαιο, οι εφαρμογές του διαδικτύου στηρίζονται σε αρχιτεκτονική τριών επιπέδων, που περιλαμβάνει τον πελάτη, τον εξυπηρετητή και τη βάση δεδομένων. Ο χρήστης μέσω της διαδικτυακής εφαρμογής συνήθως χρειάζεται να έχει πρόσβαση σε βάσεις δεδομένων, στις οποίες αποθηκεύονται χρήσιμες πληροφορίες που αφορούν, για παράδειγμα, θέσεις σε αεροπλάνα, θέατρα, δρομολόγια, δωμάτια ξενοδοχείων κ.ο.κ. Η παροχή προς τον χρήστη των πληροφοριών αυτών γίνεται από τον εξυπηρετητή που συνδέεται με την αντίστοιχη βάση δεδομένων. Ο εξυπηρετητής ανακτά τις κατάλληλες πληροφορίες και τις προωθεί στον χρήστη ανταποκρινόμενος στα σχετικά αιτήματα ή προωθεί προς τη βάση δεδομένων δεδομένα που εισάγει ο χρήστης.

Μέσω μιας απλής εφαρμογής θα δούμε στη συνέχεια πώς γίνεται η σύνδεση του εξυπηρετητή με τη βάση δεδομένων.

Θα συζητήσουμε διάφορες τεχνολογίες βάσεων δεδομένων καθώς και επιλογές που διατίθενται για φιλοξενία της βάσης δεδομένων στο νέφος. Δίνεται ιδιαίτερη βαρύτητα στο σχεσιακό μοντέλο δεδομένων και ιδιαίτερα στην PostgreSQL.

Προαπαιτούμενη γνώση

Για τη μελέτη αυτού του κεφαλαίου είναι απαραίτητη η εξοικείωση με τη Node.js, που παρουσιάστηκε στο κεφάλαιο 11, καθώς και με την JavaScript (κεφάλαια 7, 8, 9 και 10).

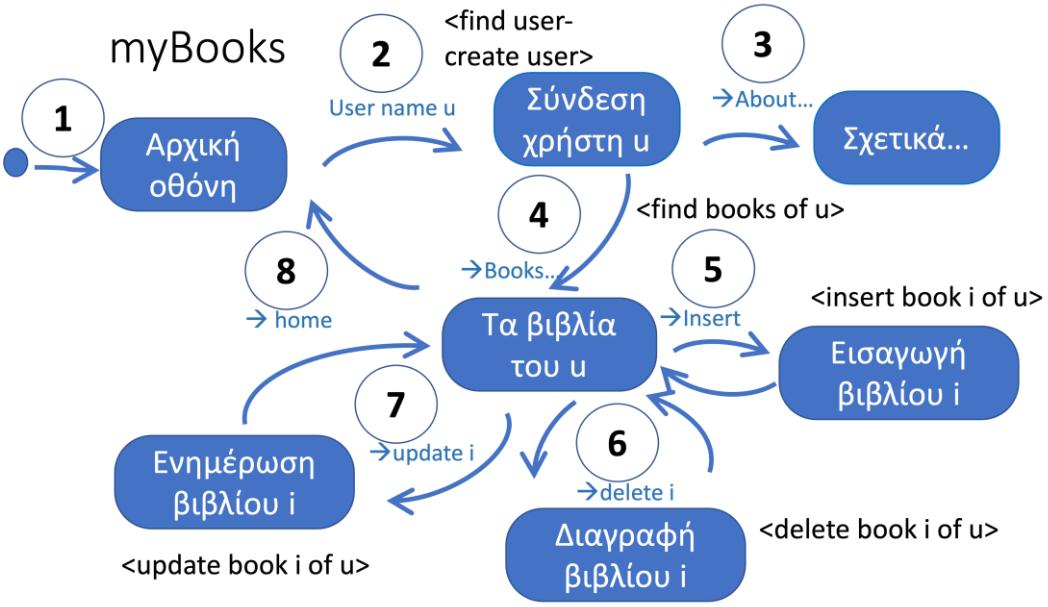
13.1 Σχεδίαση μιας διαδικτυακής εφαρμογής – δεδομένα

Θα ξεκινήσουμε από τα πρώτα βήματα σχεδίασης μιας διαδικτυακής εφαρμογής, εστιάζοντας στα δεδομένα που αυτή αφορά και την οργάνωσή τους σε μια βάση δεδομένων.

Η εφαρμογή που θα εξετάσουμε στο κεφάλαιο αυτό αφορά τον εξής μικρόκοσμο: *«Μια παρέα από φίλους που τους αρέσει το διάβασμα επιθυμούν να χρησιμοποιήσουν μια εφαρμογή που τους επιτρέπει να εισάγουν τα βιβλία που έχουν διαβάσει, καθώς και τα σχόλια που κάνουν σε αυτά»*. Η απλή αυτή διαδικτυακή εφαρμογή θα επιτρέπει σε κάθε χρήστη της να διαχειρίζεται τα βιβλία του, δηλαδή να εισάγει βιβλία, να τροποποιεί τα βιβλία που έχει εισαγάγει, να διαγράφει βιβλία (δηλαδή να κάνει τις πράξεις CRUD –Create, Read, Update, Delete– στα δεδομένα).

Για να σχεδιάσουμε την εφαρμογή αυτή θα πρέπει να πάρουμε αποφάσεις που αφορούν αφενός την εμφάνισή της, αφετέρου την αλληλεπίδραση του χρήστη με αυτή, η οποία μπορεί να περιγραφεί με έναν χάρτη μετάβασης καταστάσεων. Πρέπει επίσης να ορίσουμε τα δεδομένα που θα αποθηκεύονται σε μόνιμο μέσο αποθήκευσης, καθώς και την οργάνωσή τους, δηλαδή να ορίσουμε το **μοντέλο δεδομένων** της εφαρμογής. Στο κεφάλαιο αυτό θα εστιάσουμε κυρίως στο μοντέλο δεδομένων και στους τρόπους σύνδεσης της εφαρμογής με τη μόνιμη αποθήκευση των δεδομένων αυτών.

Μια πιθανή λειτουργία της εφαρμογής φαίνεται στο διάγραμμα μετάβασης καταστάσεων στην **Εικόνα *13.1***.

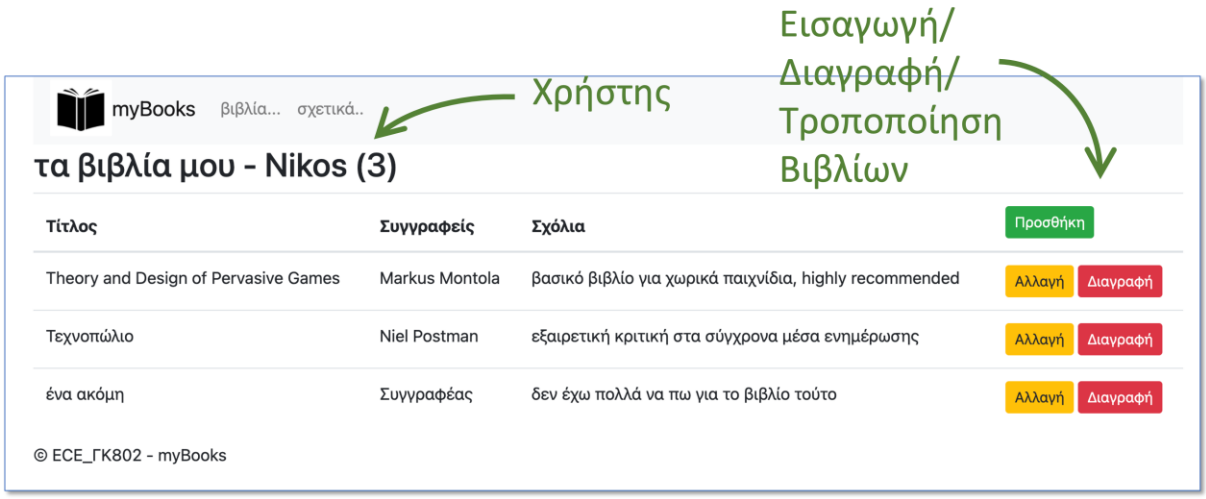


Εικόνα 13.1 *Διάγραμμα μετάβασης καταστάσεων της εφαρμογής myBooks.*

Κάθε κόμβος περιγράφει μια σελίδα (μια κατάσταση της εφαρμογής που περιμένει ενέργειες του χρήστη). Κάθε βέλος περιγράφει μια μετάβαση σε μια άλλη κατάσταση ως συνέπεια κάποιας ενέργειας του χρήστη. Οι μεταβάσεις αριθμούνται και παρέχεται μια σύντομη περιγραφή της ενέργειας που προκαλεί τη μετάβαση. Δίνεται επίσης η αντίστοιχη πράξη επί των δεδομένων της εφαρμογής. Ας κάνουμε μια σύντομη περιγραφή του διαγράμματος, και συνεπώς της λειτουργίας της εφαρμογής. (1) Ο χρήστης συνδέεται με την εφαρμογή και βρίσκεται μπροστά στην οθόνη που του ζητάει το όνομά του. Όταν δώσει το όνομα (μετάβαση 2), βρίσκεται σε μια οθόνη που δείχνει το όνομα του χρήστη που είναι συνδεδεμένος. Από εδώ έχει δύο επιλογές: να μεταβεί στην οθόνη "σχετικά" (3) ή να επιλέξει τη λίστα με τα "βιβλία" του (4). Στη δεύτερη περίπτωση μεταβαίνει σε μια οθόνη που περιέχει λίστα με τα βιβλία του χρήστη που έχει συνδεθεί. Η οθόνη αυτή επιτρέπει την προσθήκη νέου βιβλίου (5), τη διαγραφή ενός βιβλίου (6) ή την τροποποίηση ενός βιβλίου που έχει επιλεγεί (7). Μετά από την ολοκλήρωση κάθε τέτοιας ενέργειας, ο χρήστης επιστρέφει στην οθόνη με τη λίστα των βιβλίων. Θα πρέπει να σημειωθεί ότι με μετάβαση στην αρχική οθόνη (8) ο χρήστης μπορεί να επιλέξει άλλο όνομα χρήστη, οπότε η εφαρμογή εμφανίζει τα βιβλία του νέου αυτού χρήστη.

Επίσης, θα πρέπει να σημειωθεί ότι για λόγους απλότητας έχει παραληφθεί η επιβεβαίωση της ταυτότητας του χρήστη με χρήση μυστικού κωδικού, κάτι που θα συζητήσουμε στο επόμενο κεφάλαιο. Τέλος, θα πρέπει να παρατηρήσουμε ότι η εφαρμογή θα πρέπει να «θυμάται» ποιος είναι ο χρήστης που έχει συνδεθεί ανεξάρτητα σε ποια οθόνη είναι ο χρήστης, μια απαίτηση που για να ικανοποιηθεί θα πρέπει να χρησιμοποιήσουμε τη βιβλιοθήκη express-session, όπως περιγράφεται με περισσότερη λεπτομέρεια στο επόμενο κεφάλαιο.

Μια ιδέα για τη σχεδίαση της βασικής οθόνης με τη λίστα των βιβλίων ενός χρήστη φαίνεται στην **Εικόνα *13.2***. Όπως φαίνεται στην εικόνα, έχει συνδεθεί ο χρήστης με όνομα 'Nikos', ο οποίος έχει εισαγάγει τρία βιβλία στο σύστημα και του παρέχεται η δυνατότητα εισαγωγής νέου βιβλίου (πλήκτρο Προσθήκη) ή διαγραφής ή τροποποίησης του καθενός από τα βιβλία που έχουν ήδη εισαχθεί.

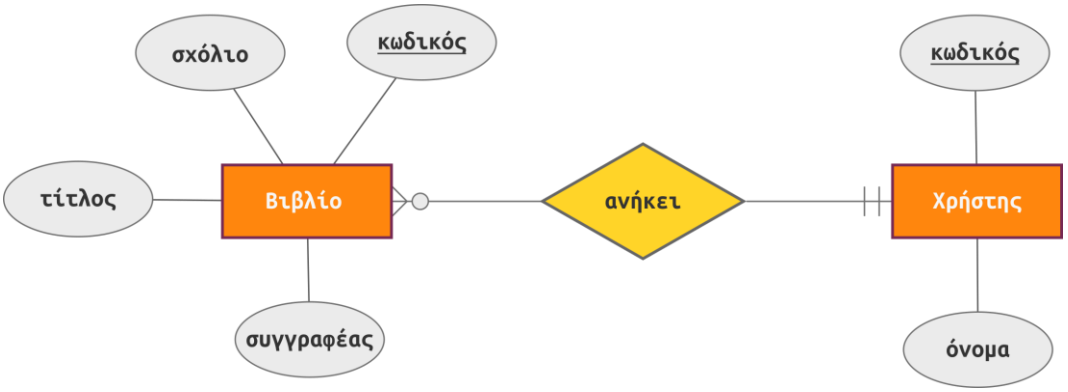


Εικόνα 13.2 *Η βασική οθόνη της εφαρμογής myBooks.*

Με βάση τη σχεδίαση που ήδη αναφέρθηκε, προκύπτουν απαιτήσεις για αποθήκευση δεδομένων της εφαρμογής αυτής στη βάση δεδομένων, καθώς και οι πράξεις που πρέπει να γίνουν σε αυτά τα δεδομένα, που έχουν ήδη σημειωθεί στο διάγραμμα καταστάσεων στην **Εικόνα *13.1***.

Η φάση αυτή της σχεδίασης ονομάζεται ανάλυση δεδομένων και οδηγεί στη δημιουργία ενός μοντέλου δεδομένων της εφαρμογής και στην προδιαγραφή της διεπαφής με τη βάση δεδομένων. Τη διεπαφή αυτή πρέπει να υλοποιήσουμε στη συνέχεια στον εξυπηρετητή.

Ένας τρόπος για να εκφράσουμε το μοντέλο δεδομένων της εφαρμογής είναι μέσω ενός διαγράμματος οντοτήτων-συσχετίσεων (Entity-Relation Diagram, ERD). Το διάγραμμα αυτό αποτυπώνει τις βασικές οντότητες του μικρόκοσμου που εξυπηρετεί η εφαρμογή, και τις συσχετίσεις μεταξύ τους. Είναι φανερό ότι στο απλό αυτό πρόβλημα οι βασικές οντότητες είναι το **Βιβλίο** και ο **Χρήστης**, για την καθεμία από τις οποίες ορίζουμε τι γνωρίσματα θέλουμε να αποθηκεύσουμε στη βάση δεδομένων. Επίσης, ορίζουμε πώς συνδέονται οι δύο αυτές οντότητες, καθώς και τον λόγο πληθικότητας της συσχέτισης, δηλαδή πόσα στοιχεία τής κάθε οντότητας σχετίζονται με στοιχεία της άλλης. Για την περίπτωσή μας ορίζουμε ότι η σχέση ***Βιβλίο-ανήκει-σε-Χρήστη*** είναι σχέση τύπου ένα προς πολλά (1:N), δηλαδή κάθε βιβλίο ανήκει σε έναν χρήστη, ενώ ένας χρήστης μπορεί να έχει από κανένα μέχρι πολλά βιβλία. Θα πρέπει να σημειωθεί ότι, αν άλλαζε αυτή η παραδοχή, θα μπορούσαμε να είχαμε σχέση βιβλίου-χρήστη τύπου πολλά προς πολλά (M:N). Σε αυτή την περίπτωση, αν το ίδιο βιβλίο τυχαίνει να το έχουν περισσότεροι από έναν χρήστες, τότε θα υπήρχε στη βάση δεδομένων μόνο μία φορά και θα μπορούσε πιθανόν κάθε χρήστης να δει ποιοι άλλοι έχουν το ίδιο βιβλίο και έχουν καταθέσει σχόλια για αυτό. Όμως, αυτή η σχεδίαση κρίθηκε πιο σύνθετη για το παράδειγμά μας.

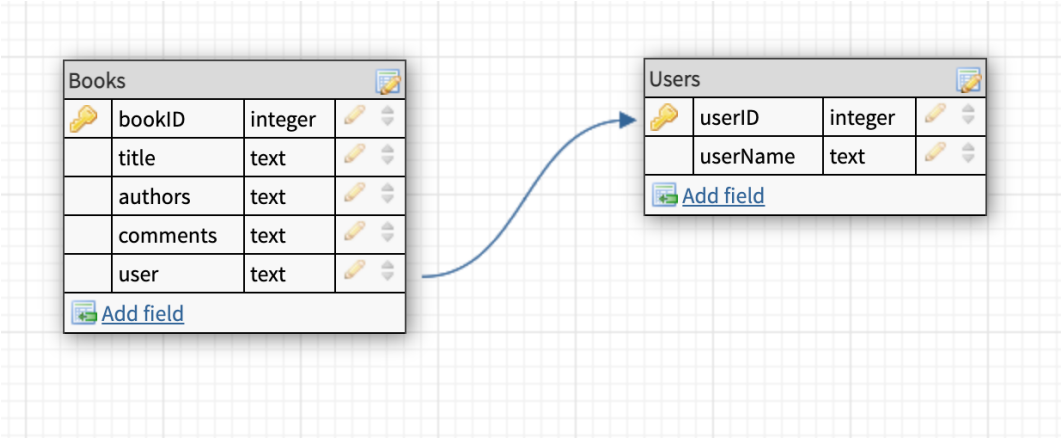


Εικόνα 13.3 *Το μοντέλο δεδομένων της εφαρμογής myBooks με μορφή διαγράμματος οντοτήτων-συσχετίσεων (ERD).*

Στη συνέχεια, το μοντέλο δεδομένων αυτό θα πρέπει να μετασχηματιστεί σε ένα **Σχήμα (Schema)** μιας βάσης δεδομένων. Ο πιο συνηθισμένος τρόπος οργάνωσης δεδομένων είναι μέσω διασυνδεδεμένων *πινάκων* που

ονομάζονται **σχέσεις (relations)**. Το μοντέλο δεδομένων που στηρίζεται σ' αυτό τον τρόπο οργάνωσης λέγεται **σχεσιακό μοντέλο (relational model)** και το πρότυπο γλώσσας αλληλεπίδρασης με αυτά τα δεδομένα είναι το **SQL (Structured Query Language)**.

Το μοντέλο δεδομένων στην **Εικόνα *13.3*** μετασχηματίζεται στο παρακάτω σχεσιακό σχήμα (**Εικόνα *13.4***), που περιλαμβάνει δύο πίνακες, οι οποίοι συνδέονται μέσω του μηχανισμού ξένου κλειδιού: Το γνώρισμα user του πίνακα *Books* αναφέρεται στο γνώρισμα userID του πίνακα *Users*, που υποδεικνύεται στην εικόνα από το σχετικό βέλος. Στο παράδειγμα αυτό λέμε ότι το γνώρισμα user είναι *ξένο κλειδί*.



Εικόνα 13.4 *Σχεσιακό σχήμα της βάσης δεδομένων myBooks (έχει παραχθεί με την εφαρμογή app.diagrams.net).*

Στο σχήμα αυτό ορίζονται οι τύποι δεδομένων για τα γνωρίσματα κάθε πίνακα, τα γνωρίσματα που είναι πρωτεύοντα κλειδιά κάθε πίνακα (το γνώρισμα bookID είναι το πρωτεύον κλειδί του πίνακα Books και το γνώρισμα userID είναι το πρωτεύον κλειδί του πίνακα Users), καθώς και το ξένο κλειδί user του πίνακα Books.

Εκτός από την οργάνωση των δεδομένων, πρέπει να ορίσουμε τις πράξεις που προβλέπεται να γίνουν στα δεδομένα αυτά. Αν επιστρέψουμε στο διάγραμμα καταστάσεων της εφαρμογής (**Εικόνα *13.1***), θα δούμε πως είχαμε καταγράψει τις εξής πράξεις στα δεδομένα:

- αναζήτηση χρήστη με όνομα 'userName',
- δημιουργία νέου χρήστη με όνομα 'userName',
- αναζήτηση όλων των βιβλίων του χρήστη 'user',
- αναζήτηση όλων των στοιχείων του βιβλίου με κωδικό 'bookID',
- διαγραφή του βιβλίου με κωδικό 'bookID',
- ενημέρωση των στοιχείων του βιβλίου με κωδικό 'bookID',
- δημιουργία νέου βιβλίου με κωδικό 'bookID'.

Πριν προχωρήσουμε στην υλοποίηση της διεπαφής της εφαρμογής μας με τη βάση δεδομένων, θα συζητήσουμε την αρχιτεκτονική του κώδικα.

13.2 Η αρχιτεκτονική MVC (model-view-controller)

Μια διαδικτυακή εφαρμογή, ακόμη και εκείνες με τις πιο απλές λειτουργίες, όπως είναι σαφές από τα παραδείγματα που έχουμε δει ως τώρα, είναι ένα σύνθετο προϊόν λογισμικού που απαιτεί οργάνωση του κώδικα. Μια αρχιτεκτονική οργάνωσης του κώδικα που προτείνεται είναι το MVC (model-view-controller), που προτάθηκε από τον Trygve Reenska από το 1978. Η οργάνωση αυτή προβλέπει τρία διακριτά τμήματα του κώδικα, το τμήμα *Model* (Μοντέλο), το οποίο χειρίζεται τη λογική των δεδομένων, το τμήμα *View* (Προβολή), το οποίο εμφανίζει τις πληροφορίες από το μοντέλο στον χρήστη και το τμήμα *Controler* (Ελεγκτής), που ελέγχει τη ροή δεδομένων σε ένα αντικείμενο μοντέλου και ενημερώνει την προβολή κάθε φορά που αλλάζουν τα δεδομένα.

Σε μια διαδικτυακή εφαρμογή ο *Ελεγκτής* είναι το τμήμα της εφαρμογής που είναι υπεύθυνο για τη δρομολόγηση αιτημάτων και για τη βασική λογική της εφαρμογής, το τμήμα *Προβολής* αφορά τα αρχεία κώδικα HTML, CSS, JavaScript που προορίζονται για τον χρήστη καθώς και τα αρχεία της template engine που χρησιμοποιείται, όπως τα αρχεία Handlebars που είδαμε στο προηγούμενο κεφάλαιο, ενώ το τμήμα *Μοντέλο* είναι υπεύθυνο για τη διεπαφή με τη βάση δεδομένων. Οι διεπαφές μεταξύ των τμημάτων αυτών πρέπει να

είναι σαφώς ορισμένες. Στη συνέχεια θα περιγράψουμε την οργάνωση του κώδικα της εφαρμογής myBooks.

13.2.1 Ο Ελεγκτής της εφαρμογής

Ο Ελεγκτής της εφαρμογής υλοποιείται στο αρχείο index.mjs.

Με την εντολή

```
import * as model from './model/model.mjs'
```

εισάγουμε το Μοντέλο στον Ελεγκτή, δηλαδή τις συναρτήσεις της διεπαφής με τη βάση δεδομένων, που θα δούμε στη συνέχεια.

Στον Ελεγκτή δημιουργούμε τον εξυπηρετητή Express, διαμορφώνουμε παραμέτρους του, όπως τη μηχανή template (εδώ η Handlebars), προσθέτουμε το middleware *express-session*, το οποίο φροντίζει για τη διατήρηση της ταυτότητας του συνδεδεμένου χρήστη σε όλα τα σημεία δρομολόγησης, επίσης δημιουργούμε το middleware *redirectHome*, το οποίο φροντίζει για μη συνδεδεμένους χρήστες να ανακατευθύνονται στην αρχική σελίδα αν κάνουν αίτημα δρομολόγησης σε οποιοδήποτε μονοπάτι.

Τέλος, ο Ελεγκτής εξυπηρετεί τα διάφορα σημεία δρομολόγησης της εφαρμογής. Αυτά είναι:

- Το μονοπάτι "/" που αφορά την αρχική σελίδα, που καλείται είτε με μέθοδο GET όταν συνδέεται ο χρήστης με αυτή, είτε με μέθοδο POST όταν υποβάλλει τη φόρμα με το όνομα του χρήστη.
- Το μονοπάτι "**/about**" που αφορά τη σελίδα που περιέχει πληροφορίες για την εφαρμογή.
- Το μονοπάτι "**/books**" που αφορά την κεντρική σελίδα που περιέχει τη λίστα των βιβλίων του χρήστη.
- Το μονοπάτι "**/create**" που αφορά τη σελίδα με τη φόρμα δημιουργίας νέου βιβλίου καθώς και την υποβολή της φόρμας με τη μέθοδο POST.
- Το παραμετρικό μονοπάτι "**/edit/:bookID**" που αφορά τη φόρμα ενημέρωσης των στοιχείων του βιβλίου με κωδικό bookID, που εξυπηρετείται για αιτήματα GET για αποστολή της φόρμας με τα υπάρχοντα στοιχεία, και αιτήματα POST για υποβολή των ενημερωμένων στοιχείων.
- Τέλος, το παραμετρικό μονοπάτι "**/delete/:bookID**" με αντίστοιχη λειτουργία για τη διαγραφή ενός βιβλίου.

Η δομή του Ελεγκτή φαίνεται στη συνέχεια.

```
import express from "express";
import exphbs from "express-handlebars";
import path from "path";
import session from "express-session";

import * as model from './model/model.js';  // Το τμήμα model

// Δημιουργία εξυπηρετητή Express
const app = express();

// Διαμόρφωση του εξυπηρετητή
app.engine('hbs', exphbs.engine({ extname: 'hbs', defaultLayout: 'main',
layoutsDir: __dirname + '/views/layouts' }));
app.set('view engine', 'hbs');
app.use(express.static("public"));
app.use(express.urlencoded({ extended: false }));

// Προσθήκη του express-session middleware
app.use(session({
  name: process.env.SESS_NAME,
  secret: process.env.SESSION_SECRET, // κλειδί για κρυπτογράφηση του
cookie
  ...
}));

const redirectHome = (req, res, next) => {
  console.log('redirect...', req.session)
  if (!req.session.userID) {
```

```
    res.redirect('/');
  } else {
    next();
  }
};

// Εκκίνηση του εξυπηρετητή
const PORT = process.env.PORT || 3003
app.listen(PORT, () => {
  console.log(`Συνδεθείτε στη σελίδα: http://localhost:${PORT}`);
});

// GET /
app.get("/", (req, res) => { ... });

// POST /
app.post("/", (req, res) => { ... });

// GET /about
app.get("/about", (req, res) => { ... });

// GET /books
app.get("/books", redirectHome, (req, res) => { ... });

// GET /create
app.get("/create", redirectHome, (req, res) => { ... });

// POST /create
app.post("/create", redirectHome, (req, res) => { ... });

// GET /edit/:bookID
app.get("/edit/:bookID", redirectHome, (req, res) => { ... });

// POST /edit/:bookID
app.post("/edit/:id", redirectHome, (req, res) => { ... });

// GET /delete/:bookID
app.get("/delete/:id", redirectHome, (req, res) => { ... });

// POST /delete/:bookID
app.post("/delete/:id", (req, res) => { ... });
```

Οι παραπάνω μέθοδοι δρομολόγησης περιέχουν τη λογική της εφαρμογής, η οποία αφορά την εξυπηρέτηση του αντίστοιχου κάθε φορά αιτήματος, με κλήση της διεπαφής της βάσης δεδομένων. Ας δούμε ως παράδειγμα τη μέθοδο δρομολόγησης του μονοπατιού /books.

```
app.get("/books", redirectHome, (req, res) => {
  const userID = req.session.userID;
  const userName = req.session.userName;
  model.getMyBooks(userID, (err, rows) => {
    if (err) {
      return console.error(err.message);
    }
    res.render("books", { user: userName, data: rows });
  });
});
```

Το αίτημα δρομολόγησης αυτό προκύπτει όταν ο χρήστης επιλέξει από το μενού να δει τα βιβλία του. Παρατηρούμε ότι πρώτα καλείται η redirectHome(), δηλαδή η συνάρτηση middleware που ελέγχει αν υπάρχει

συνδεδεμένος χρήστης και αν όχι, δεν επιτρέπεται να προχωρήσει η δρομολόγηση και ο χρήστης ανακατευθύνεται στην αρχική σελίδα, όπου μπορεί να συνδεθεί δίνοντας το όνομά του.

Στη συνέχεια ανακτούμε από το αντικείμενο session του αντικείμενου req τα στοιχεία του χρήστη, userID, και userName. Έπειτα καλούμε τη συνάρτηση getMyBooks(userID, callback) του Μοντέλου.

Η συνάρτηση αυτή αποτελεί τμήμα της διεπαφής της βάσης δεδομένων. Όπως όλες οι συναρτήσεις της διεπαφής αυτής, εκτελείται με ασύγχρονο τρόπο, με ορισμό συνάρτησης επιστροφής callback που θα εκτελεστεί όταν η αλληλεπίδραση με τη βάση δεδομένων ολοκληρωθεί. Η συνάρτηση επιστροφής δέχεται δύο αμοιβαία αποκλειόμενα ορίσματα, όπως έχουμε συζητήσει σε προηγούμενο κεφάλαιο – το πρώτο, το err, θα έχει τιμή αν η πρόσβαση στη βάση δεδομένων αποτύχει, το δεύτερο, το rows, θα έχει τιμή αν η πρόσβαση στη βάση δεδομένων είναι επιτυχής, και είναι ένας πίνακας με αντικείμενα, τα αποτελέσματα της αναζήτησης.

Στην περίπτωση που δεν έχει προκύψει σφάλμα, η συνάρτηση επιστροφής καλεί τη μέθοδο `render()` που επιτρέπει τη χρήση ενός template στο οποίο περνάμε τα αποτελέσματα της αναζήτησης, δηλαδή τον πίνακα rows, ώστε ο χρήστης να δει τα βιβλία του.

Με παρόμοιο τρόπο ορίζονται και τα άλλα σημεία δρομολόγησης, τα οποία καλούν συναρτήσεις της διεπαφής με τη βάση δεδομένων. Στην επόμενη ενότητα θα δούμε πώς υλοποιείται το *Μοντέλο*, που περιέχει όλες τις συναρτήσεις της διεπαφής με τη βάση δεδομένων.

13.2.2 Το Μοντέλο της εφαρμογής myBooks

Το αρχείο model.mjs που βρίσκεται στον φάκελο /model υλοποιεί το *Μοντέλο* της εφαρμογής μας. Οι συναρτήσεις που περιλαμβάνονται στη διεπαφή με τη μόνιμη αποθήκευση δεδομένων είναι οι εξής:

```
getMyBooks = (userID, callback) => {
/// διάβασε τα βιβλία του χρήστη με κωδικό userID
}

newBook = (book, callback) => {
/// εισαγωγή ενός νέου βιβλίου στη βάση δεδομένων, το αντικείμενο book
περιέχει τα στοιχεία του βιβλίου
}

findBook = (bookID, callback) => {
/// ανάκτησε το βιβλίο με κωδικό bookID
}

updateBook = (book, callback) => {
/// ενημέρωση του βιβλίου book στη βάση δεδομένων
}

deleteBook = (bookID, callback) => {
/// διαγραφή του βιβλίου book από τη βάση δεδομένων
}

insertUser = (user, callback) => {
// εισαγωγή του χρήστη user και επιστροφή του εισαχθέντος χρήστη
}

findUser = (userID=null, userName=null, callback) => {
// εύρεση των στοιχείων του χρήστη με όνομα userName, αν δεν υπάρχει
δημιουργία νέου χρήστη ή εύρεση των στοιχείων του χρήστη με κωδικό userID
}
```

Όπως φαίνεται, όλες οι συναρτήσεις της διεπαφής λειτουργούν ασύγχρονα και για αυτό περιλαμβάνουν στα ορίσματά τους μια συνάρτηση επιστροφής callback, η οποία έχει οριστεί στον *Ελεγκτή*, όπως είδαμε στην προηγούμενη ενότητα. Η συνάρτηση αυτή θα χρησιμοποιηθεί για να περάσουμε το αποτέλεσμα της πράξης στη βάση δεδομένων ή το μήνυμα σφάλματος στον Ελεγκτή.

Στις συναρτήσεις αυτές υπάρχουν ως ορίσματα είτε αντικείμενα, όπως για παράδειγμα το αντικείμενο book, που τυπικά θα έχει ως περιεχόμενο τα στοιχεία ενός βιβλίου:

```
{
    "title": "Ποιήματα",
    "author": "Κωνσταντίνος Καβάφης",
    "comment": "",
    "user": "1",
    "bookID": "10"
}
```

είτε μεταβλητές, όπως ο κωδικός ενός βιβλίου bookID ή ο κωδικός ενός χρήστη userID.

Υπάρχουν σημεία της διεπαφής, όπως η συνάρτηση `findUser()`, που η λογική τους είναι πιο σύνθετη. Η συνάρτηση αυτή δέχεται ως παράμετρο είτε τον κωδικό χρήστη είτε το όνομα του χρήστη. Στην πρώτη περίπτωση επιστρέφει τα στοιχεία του χρήστη, π.χ. τα στοιχεία του χρήστη με κωδικό “2” είναι:

```
{ "userID": "2", "userName": "Νίκος" }
```

Στην περίπτωση που καλείται με ορισμένο το όνομα του χρήστη, η συνάρτηση αναζητά τον χρήστη με το συγκεκριμένο όνομα και επιστρέφει τα στοιχεία του, ενώ, αν ο χρήστης δεν βρεθεί, εισάγει τον νέο χρήστη στη μόνιμη αποθήκευση και τότε επιστρέφει τα στοιχεία του.

Ο λόγος που χρειάζεται να επιστραφούν τα στοιχεία του χρήστη είναι γιατί η εφαρμογή δεν γνωρίζει τον κωδικό του χρήστη, κάτι που χρειάζεται για τη μονοσήμαντη αναγνώρισή του. Συνήθως, ο κωδικός χρήστη είναι ένας αριθμός που παράγεται αυτόματα από τη βάση δεδομένων όταν γίνεται η πρώτη εγγραφή του χρήστη στην εφαρμογή και αποθηκεύονται τα στοιχεία του στον πίνακα (π.χ. με τη δήλωση AUTO_INCREMENT κατά τη δημιουργία ενός πίνακα σε μια βάση δεδομένων SQL).

Στη συνέχεια παρουσιάζεται ως παράδειγμα η υλοποίηση μιας από τις συναρτήσεις της διεπαφής με τη βάση δεδομένων. Για να φανεί η σημασία της αρχιτεκτονικής MVC, θα υλοποιηθούν δύο εκδόσεις του μοντέλου, η πρώτη με χρήση αρχείων ως μόνιμου μέσου αποθήκευσης και η δεύτερη με χρήση μιας σχεσιακής βάσης δεδομένων – δύο διαφορετικές υλοποιήσεις που χρησιμοποιούν όμως την ίδια διεπαφή.

13.3 Υλοποίηση με μόνιμη αποθήκευση σε αρχεία JSON

Η πρώτη λύση αφορά την αποθήκευση των δεδομένων σε αρχεία JSON. Επειδή έχουμε δύο πίνακες να αποθηκεύσουμε, θα χρειαστούμε δύο αρχεία, το αρχείο books.json που περιέχει τον πίνακα των βιβλίων και το αρχείο users.json που περιέχει τον πίνακα των χρηστών. Τυπικό περιεχόμενο των αρχείων αυτών είναι:

```
// αρχείο users.json
[
    {
        "userID": "1",
        "userName": "Κατερίνα"
    },
    {
        "userID": "2",
        "userName": "Νίκος"
    },
    {
        "userID": "3",
        "userName": "Γιώργος"
    }
]
// αρχείο books.json
[
    {
        "title": "Άπαντα",
        "author": "Σεφέρη",
        "comment": "πολλά καλά ποιήματα",
        "bookID": "6",
        "user": "1"
    },
```

```
    {
        "title": "Η Αλίκη στη χώρα των θαυμάτων",
        "author": "Lewis Carroll",
        "comment": "δεν είναι απλό παραμύθι...",
        "bookID": "7",
        "user": "1"
    }
]
```

Η αλληλεπίδραση με το σύστημα αρχείων του δίσκου μας γίνεται, όπως είδαμε σε προηγούμενο κεφάλαιο, με την ενσωματωμένη στη βιβλιοθήκη "fs" της Node.js.

Αρχικά εισάγουμε στο αρχείο model.mjs τη βιβλιοθήκη, καθώς και βοηθητικές βιβλιοθήκες για διαχείριση των μονοπατιών του συστήματος αρχείων:

```
import fs from "fs";
import path from "path";
import { fileURLToPath } from "url";
const __dirname = path.dirname(fileURLToPath(import.meta.url));
```

Στη συνέχεια ορίζουμε τις συναρτήσεις της διεπαφής και, τέλος, εξάγουμε τις συναρτήσεις με την εντολή:

```
export {getMyBooks, newBook, findBook, updateBook, deleteBook, insertUser,
findUser};
```

Θα πρέπει να σημειωθεί ότι έχει χρησιμοποιηθεί το σύστημα ES6 για εγκατάσταση modules. Άρα ή θα χρησιμοποιήσουμε την κατάληξη .mjs στα αρχεία μας ή θα ορίσουμε στο αρχείο package.json: "type": "module".

Ας δούμε τώρα ως παράδειγμα την υλοποίηση της συνάρτησης getMyBooks:

```
const getMyBooks = (userID, callback) => {
    // διάβασε το αρχείο books.json
    const db_name = path.join(__dirname, "../data", "books.json");
    fs.readFile(db_name, {encoding:"utf-8"}, (err, data) => {
        if (err) {
            callback(err, null);
        }
        else {
            const theBooks = JSON.parse(data).filter((item) => item.user
=== userID)
            callback(null, theBooks)
        }
    })
}
```

Η συνάρτηση δέχεται ως όρισμα τον κωδικό του συνδεδεμένου χρήστη userID και αναζητά στο αρχείο των βιβλίων books.json τα βιβλία του χρήστη αυτού. Χρησιμοποιείται η συνάρτηση fs.readFile, η οποία είναι ασύγχρονη συνάρτηση που ανοίγει το αρχείο και αν είναι επιτυχής μεταφέρει ως data τα δεδομένα που ανάκτησε στη συνάρτηση ανάκλησης. Τα δεδομένα αυτά είναι, ως γνωστόν, χαρακτήρες. Για μετατροπή τους σε δομή δεδομένων της JavaScript καλούμε τη μέθοδο **JSON**.parse(), η οποία επιστρέφει έναν πίνακα από αντικείμενα τύπου book. Στη συνέχεια, με εφαρμογή της filter(), επιλέγουμε τα βιβλία τα οποία έχουν τιμή στο κλειδί user αυτή του userID, δηλαδή επιλέγουμε, από όλα τα βιβλία του πίνακα, τα βιβλία του χρήστη που μας ενδιαφέρει.

Τα δεδομένα που ανακτήσαμε επιστρέφουμε με κλήση της συνάρτησης επιστροφής callback, η οποία όπως είδαμε ορίστηκε στον Ελεγκτή και η οποία μεταφέρει τα δεδομένα στο template για να αποσταλούν στον χρήστη.

13.4 Υλοποίηση με μόνιμη αποθήκευση σε βάση δεδομένων PostgreSQL

Η προηγούμενη λύση έχει διδακτικό χαρακτήρα, αλλά δεν μπορεί να ικανοποιήσει τις απαιτήσεις μιας

εφαρμογής του διαδικτύου, η οποία δυνητικά θα πρέπει να υποστηρίζει μεγάλο αριθμό χρηστών και να διαχειρίζεται μεγάλο όγκο δεδομένων, αφού η λύση του αρχείου json θα γίνει πολύ αργή όταν ο όγκος των δεδομένων αυξηθεί και τα αιτήματα για πρόσβαση στο αρχείο επίσης αυξηθούν. Θα πρέπει λοιπόν για τη μόνιμη αποθήκευση των δεδομένων της εφαρμογής μας να αναζητήσουμε λύση σε μια βάση δεδομένων που ακολουθεί το μοντέλο *πελάτη-εξυπηρετητή*, η οποία συνήθως είναι εγκαταστημένη σε άλλη υπολογιστική υποδομή (database server), διαφορετική από αυτή της εφαρμογής μας.

Υπάρχουν πολλές λύσεις που μπορεί κανείς να χρησιμοποιήσει για τον σκοπό αυτό. Σύμφωνα με το StackOverflow Developers Survey, οι πιο δημοφιλείς σήμερα τεχνολογίες βάσεων δεδομένων που ικανοποιούν το παραπάνω κριτήριο και είναι συνεπώς κατάλληλες να χρησιμοποιηθούν για εφαρμογές διαδικτύου είναι η MySQL και η PostgreSQL. Θα πρέπει να σημειωθεί ότι και τα δύο αυτά συστήματα διαχείρισης βάσεων δεδομένων ακολουθούν το σχεσιακό μοντέλο δεδομένων και, συνεπώς, χρησιμοποιούν το πρότυπο SQL ως γλώσσα αλληλεπίδρασης με τα δεδομένα. Επίσης, και τα δυο είναι συστήματα ανοικτού κώδικα και ελεύθερα διαθέσιμα. Να σημειωθεί ακόμη ότι στη λίστα με τις πιο δημοφιλείς τεχνολογίες βάσεων δεδομένων ακολουθεί η ενσωματωμένη βάση δεδομένων SQlite. Η τεχνολογία αυτή είναι ιδιαίτερα απλή και χρησιμοποιείται για αρχικό πειραματισμό με σχεσιακές βάσεις δεδομένων, αφού ολόκληρη η βάση δεδομένων αποθηκεύεται σε ένα αρχείο του δίσκου. Όμως, δεν είναι κατάλληλη για παραγωγικές εφαρμογές διαδικτύου, αφού δεν ακολουθεί το μοντέλο πελάτη-εξυπηρετητή, δεν διαχειρίζεται πρόσβαση πολλαπλών χρηστών κ.λπ. Στη συνέχεια, στον πίνακα εμφανίζεται η βάση δεδομένων MongoDB, η οποία επίσης χρησιμοποιείται για εφαρμογές διαδικτύου, δεν ακολουθεί όμως το σχεσιακό μοντέλο, και την οποία θα συζητήσουμε σε επόμενη ενότητα. Ακολουθούν άλλες βάσεις δεδομένων, οι περισσότερες σχεσιακού τύπου (Microsoft SQL Server, MariaDB, Oracle κ.λπ.).

Στην ενότητα αυτή θα υλοποιήσουμε την εφαρμογή myBooks με χρήση της βάσης δεδομένων **PostgreSQL**, μιας από τις πιο δημοφιλείς και διαδεδομένες βάσεις δεδομένων στο διαδίκτυο, η οποία έχει το πρόσθετο πλεονέκτημα ότι υποστηρίζεται εγγενώς από την πλατφόρμα νέφους Heroku που θα χρησιμοποιήσουμε για δημοσιοποίηση της εφαρμογής μας.

13.4.1 Εγκατάσταση της PostgreSQL

Πρώτα θα πρέπει να εγκαταστήσουμε έναν εξυπηρετητή PostgreSQL στο περιβάλλον ανάπτυξης της εφαρμογής μας. Η εγκατάσταση γίνεται σύμφωνα με τις οδηγίες της PostgreSQL. Θα πρέπει να δοθεί ιδιαίτερη προσοχή στη διαμόρφωση του περιβάλλοντος του υπολογιστή για πρόσβαση στα εργαλεία γραμμής εντολών της PostgreSQL, τα οποία θα φανούν χρήσιμα για τη μεταφορά της βάσης δεδομένων στο Heroku. Ιδιαίτερα, μαζί με τον εξυπηρετητή PostgreSQL περιλαμβάνεται το εργαλείο psql. Όταν η εγκατάσταση ολοκληρωθεί επιτυχώς, μπορείτε να ελέγξετε την επιτυχή διαμόρφωση του περιβάλλοντος ανάπτυξης αν δώσετε την εντολή which psql στο τερματικό σας (σε συστήματα Mac ή Linux) και λάβετε ως απάντηση το μονοπάτι του εργαλείου αυτού. Οδηγίες μπορείτε να βρείτε στις σελίδες της PostgreSQL.

Μια εναλλακτική επιλογή για εγκατάσταση του εξυπηρετητή είναι με χρήση Docker image που μπορείτε να εγκαταστήσετε με την εντολή docker pull postgres, αφού βεβαίως έχει προηγούμενα εγκατασταθεί η εφαρμογή Docker στον υπολογιστή σας.

Το επόμενο βήμα είναι η εγκατάσταση ενός εργαλείου διαχείρισης της βάσης δεδομένων, αν δεν επιθυμούμε η διαχείριση να γίνει αποκλειστικά από τη γραμμή εντολών psql. Ένα πλήρες και ιδιαίτερα δημοφιλές περιβάλλον διαχείρισης είναι το περιβάλλον pgAdmin. Αφού εγκατασταθεί το περιβάλλον και συνδεθεί με τον τοπικό εξυπηρετητή postgres, μπορούμε να δημιουργήσουμε μια νέα βάση δεδομένων, να ορίσουμε τους πίνακες, τη δομή τους, τους περιορισμούς πρωτεύοντος κλειδιού και ξένου κλειδιού, άλλα χαρακτηριστικά τους, καθώς και να εισαγάγουμε δεδομένα στους πίνακες αυτούς. Οι ενέργειες αυτές μπορούν να γίνουν είτε μέσω γραφικού περιβάλλοντος, π.χ. επιλογή create database, επιλογή schemas/public/Tables/Create …/table κ.λπ., δηλαδή με επιλογές από τα μενού του περιβάλλοντος, είτε με εντολές SQL. Θα πρέπει να δοθεί ιδιαίτερη προσοχή γιατί το PgAdmin περιλαμβάνει πολλές επιλογές και δεν γίνεται διάκριση μεταξύ των κύριων και των δευτερευόντων επιλογών.

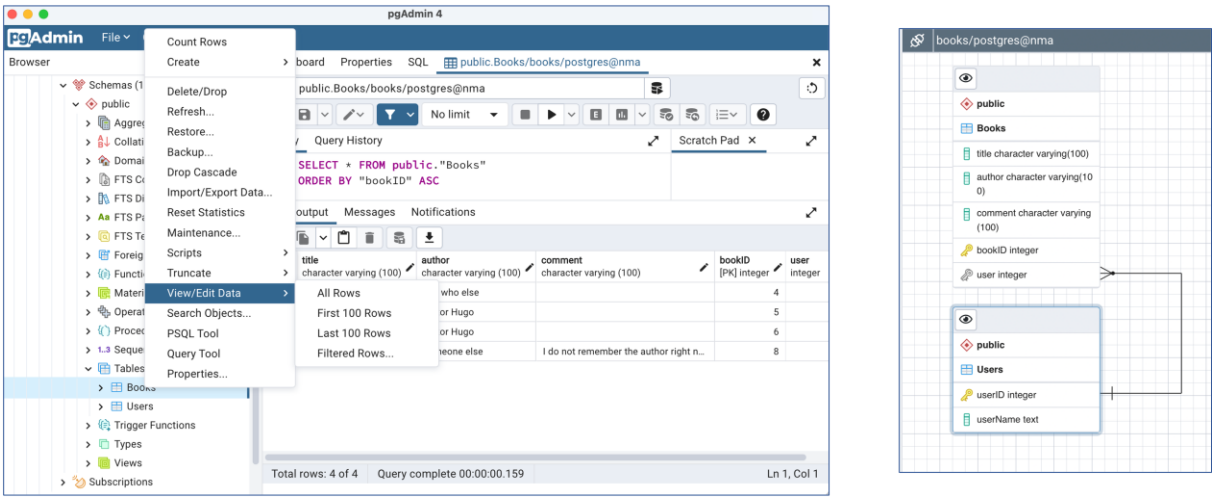
Ένα παράδειγμα δημιουργίας πίνακα με εντολές SQL ακολουθεί:

```
CREATE TABLE "Books" (
    "title" character varying(100) NOT NULL,
    "author" character varying(100),
    "comment" character varying(100),
    "bookID" integer NOT NULL,
    "user" integer
```

```
);

CREATE TABLE  "Users" (
    "userID" integer NOT NULL,
    "userName" text
);

ALTER TABLE "Users" ALTER COLUMN "userID" ADD GENERATED ALWAYS AS IDENTITY
(
    SEQUENCE NAME public."Users_userID_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);

ALTER TABLE "Books" ALTER COLUMN "bookID" ADD GENERATED ALWAYS AS IDENTITY
(
    SEQUENCE NAME public."books_bookID_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);

ALTER TABLE ONLY "Users"
    ADD CONSTRAINT pk_1 PRIMARY KEY ("userID");

ALTER TABLE ONLY "Books"
    ADD CONSTRAINT "pk_bookID" PRIMARY KEY ("bookID");

ALTER TABLE ONLY "Books"
    ADD CONSTRAINT fk_1 FOREIGN KEY ("user") REFERENCES
public."Users"("userID") NOT VALID;
```

Στον κώδικα αυτόν ορίζονται οι πίνακες “Books” και “Users” και στη συνέχεια διάφοροι περιορισμοί, όπως η αυτόματη δημιουργία τιμών των κωδικών τους, οι περιορισμοί πρωτεύοντος κλειδιού και ο περιορισμός ξένου κλειδιού. Ο παραπάνω κώδικας παρήχθη αυτόματα από το περιβάλλον της pgAdmin μετά τη δημιουργία της βάσης δεδομένων, όμως θα μπορούσαν να είχαν δοθεί οι εντολές από το τερματικό της SQL που περιλαμβάνεται στο περιβάλλον. Επίσης, ιδιαίτερη προσοχή θα πρέπει να δοθεί ότι τα ονόματα των πινάκων, γνωρισμάτων πινάκων κ.λπ. περιβάλλονται από διπλά εισαγωγικά (double quotes).

Ένα στιγμιότυπο του περιβάλλοντος φαίνεται στην **Εικόνα *13.5***.



Εικόνα 13.5 *Στιγμιότυπο από το περιβάλλον διαχείρισης pgAdmin4 κατά τη δημιουργία της βάσης δεδομένων books στο RDBMS PostgreSQL, αριστερά η επιλογή του πίνακα books, δεξιά η γραφική αναπαράσταση των πινάκων.*

Από το περιβάλλον του εξυπηρετητή θα πρέπει να καταγράψουμε τα στοιχεία σύνδεσης με τη βάση, όπως το όνομα του χρήστη, μυστικό κωδικό, το όνομα της βάσης, τη θύρα του εξυπηρετητή κ.λπ., τα οποία θα χρειαστούμε για τη σύνδεση της εφαρμογής μας με τη βάση δεδομένων.

13.4.2 Σύνδεση με την PostgreSQL

Αφού έχει ολοκληρωθεί η εγκατάσταση του εξυπηρετητή βάσης δεδομένων στο τοπικό περιβάλλον ανάπτυξης και έχει δημιουργηθεί η βάση δεδομένων της εφαρμογής μας, μπορούμε να επιστρέψουμε στον κώδικα και να δημιουργήσουμε μια νέα έκδοση του Μοντέλου.

Θα πρέπει πρώτα να εγκαταστήσουμε τη σχετική βιβλιοθήκη της Node.js

```
npm install pg
```

Επίσης, είναι καλή ιδέα να μελετήσουμε την τεκμηρίωση της διεπαφής της βιβλιοθήκης αυτής.

Το νέο αρχείο model.mjs θα πρέπει να περιλαμβάνει αρχικά σύνδεση με τη βάση δεδομένων.

Όπως θα δούμε, στην τεκμηρίωση της pg παρέχονται δύο επιλογές σύνδεσης, είτε μέσω δεξαμενής συνδέσεων είτε με δημιουργία νέας σύνδεσης. Η **δεξαμενή συνδέσεων (connection pool)** είναι μια προσωρινή μνήμη των συνδέσεων βάσης δεδομένων που επιτρέπει οι συνδέσεις να επαναχρησιμοποιούνται αντί να δημιουργούνται κάθε φορά που ζητείται μια σύνδεση. Ο δημιουργός αντικείμενου δεξαμενής συνδέσεων Pool() χρησιμοποιείται για τον σκοπό αυτό. Ενώ ο δημιουργός Client() για τη δημιουργία μιας νέας σύνδεσης.

Ένα τυπικό παράδειγμα δημιουργίας δεξαμενής συνδέσεων είναι το εξής:

```
const pool = new pg.Pool({  // τοπική σύνδεση
    user: ///username,
    host: 'localhost',
    database: 'books',
    password: /// password,
    port: 5432,
})
```

Όπως έχει αναφερθεί στο προηγούμενο κεφάλαιο, δεν είναι καλή πρακτική να περιλάβουμε ευαίσθητα δεδομένα όπως το όνομα χρήστη ή τον κωδικό πρόσβασης στον κώδικα της εφαρμογής μας. Μπορούμε με χρήση της βιβλιοθήκης dotenv να δώσουμε πρόσβαση στο πρόγραμμά μας στο βοηθητικό αρχείο .env, στο οποίο αποθηκεύουμε μεταβλητές περιβάλλοντος καθώς και τις μεταβλητές περιβάλλοντος του υπολογιστή μας. Με χρήση αυτής της προσέγγισης ο παρακάτω κώδικας είναι ισοδύναμος με το προηγούμενο παράδειγμα:

```
import dotenv from "dotenv";
dotenv.config();
```

```
const pool = new pg.Pool(); //οι παράμετροι ορίζονται ως μεταβλητές
περιβάλλοντος (βλέπε χρήση dotenv)
```

Για να είναι ισοδύναμη η σύνδεση αυτή, θα πρέπει να έχει δημιουργηθεί ένα αρχείο .env με το εξής περιεχόμενο:

```
PGHOST=localhost
PGUSER=<username>
PGDATABASE=books
PGPASSWORD= <password>
PGPORT=5432
```

Όταν αργότερα χρειαστεί να συνδέσουμε την εφαρμογή μας με έναν εξυπηρετητή PostgreSQL που βρίσκεται στο διαδίκτυο, μια τυπική πρακτική είναι ο πάροχος της υπηρεσίας να μας δώσει μια συμβολοσειρά που λέγεται **connection URL**. Η συμβολοσειρά αυτή εισάγεται ως μεταβλητή περιβάλλοντος στο αρχείο .env, για παράδειγμα με το όνομα:

```
DATABASE_URL=postgres://paxdhrppercbkb:440e6775c721...
```

Σε αυτή την περίπτωση η σύνδεση μπορεί να γίνει ως εξής:

```
const pool = new pg.Pool({
    connectionString: process.env.DATABASE_URL, //μεταβλητή περιβάλλοντος
    ssl: {
        rejectUnauthorized: false
    }
  });
```

Στη συνέχεια θα ορίσουμε μια ασύγχρονη συνάρτηση που δημιουργεί και επιστρέφει μια σύνδεση από τη δεξαμενή συνδέσεων που έχουμε δημιουργήσει. Το προγραμματιστικό μοντέλο που θα χρησιμοποιήσουμε στην περίπτωση αυτή είναι αυτό των υποσχέσεων με τη σύνταξη async/await που είδαμε στο προηγούμενο κεφάλαιο (11.5). Η σύνταξη αυτή είναι πιο απλή σε σύγκριση με τις συναρτήσεις επιστροφής και θυμίζει αυτή της σειριακής εκτέλεσης κώδικα.

Η συνάρτηση connect() καλεί τη μέθοδο connect() του αντικείμενου pool που έχουμε δημιουργήσει και επιστρέφει τη νέα σύνδεση. Ο έλεγχος σφάλματος της ασύγχρονης αυτής λειτουργίας γίνεται με τον τρόπο που έχουμε δει στη σειριακή εκτέλεση κώδικα, δηλαδή με τη δομή try/catch. Στην περίπτωση σφάλματος τυπώνεται το σχετικό μήνυμα και η συνάρτηση επιστρέφει null.

```
async function connect() {
    try {
        const client = await pool.connect();
        return client
    }
    catch(e) {
        console.error(`Failed to connect ${e}`)
    }
}
```

13.4.3 Υλοποίηση της διεπαφής με τη βάση δεδομένων PostgreSQL

Αφού έχουμε υλοποιήσει τη σύνδεση με τη βάση δεδομένων, στη συνέχεια προχωράμε στην υλοποίηση των συναρτήσεων της διεπαφής.

Θα δούμε εκ νέου την υλοποίηση της συνάρτησης getMyBooks που συζητήθηκε στην προηγούμενη ενότητα χρήσης αρχείου JSON. Η συνάρτηση αυτή τώρα πρέπει να ανακτήσει τα βιβλία του χρήστη με κωδικό userID από τη βάση δεδομένων books που δημιουργήσαμε στον εξυπηρετητή PostgreSQL.

```
async function getMyBooks(userID, callback) {
    // ανάκτηση όλων των βιβλίων του χρήστη από τη βάση δεδομένων
    const sql = `SELECT * FROM "Books" WHERE "user" = $1 ORDER BY
"title";`
```

```
    const params  = [userID];
    try {
        const client = await connect();
        const res = await client.query(sql, params)
        await client.release()
        callback(null, res.rows) // επιστρέφει array
    }
    catch (err) {
        callback(err, null);
    }
}
```

Φροντίζουμε να υλοποιήσουμε τη λειτουργικότητα της διεπαφής που ορίστηκε στην προηγούμενη ενότητα, δηλαδή τη χρήση της callback για επιστροφή μιας δομής array με τα αντικείμενα τύπου book που ανακτήθηκαν από τη βάση δεδομένων.

Δημιουργούμε αρχικά μια σύνδεση με τη συνάρτηση connect() η οποία, όπως είδαμε, δημιουργεί μια σύνδεση με χρήση της δεξαμενής συνδέσεων pool.

Στη συνέχεια καλούμε τη μέθοδο `query()` στην οποία περνάμε ως συμβολοσειρά την εντολή SQL προς τη βάση δεδομένων, καθώς και έναν πίνακα με τις παραμέτρους που εισάγουμε στη συμβολοσειρά sql. Να σημειωθεί εδώ ότι θα μπορούσαμε να χρησιμοποιήσουμε πρότυπη συμβολοσειρά με εισαγωγή παραμέτρων όπως, για παράδειγμα, `SELECT * FROM "Books" WHERE "user" = '${userID}' ORDER BY "title";`. Όμως, η χρήση παραμέτρων θεωρείται πιο ασφαλής για την αποφυγή επιθέσεων τύπου SQL injection.

Η ασύγχρονη κλήση της συνάρτησης query() επιστρέφει το αποτέλεσμα της εκτέλεσης της εντολής SQL στο αντικείμενο res, το οποίο έχει την ιδιότητα rows. Αυτή επιστρέφει έναν πίνακα με αντικείμενα, καθένα από τα οποία έχει τα ονόματα των στηλών του πίνακα ως κλειδιά και τις τιμές του πίνακα ως αντίστοιχες τιμές. Με αυτό τον τρόπο επιστρέφουμε στη συνάρτηση ανάκλησης callback πίνακα αντικειμένων, που ήταν το ζητούμενο.

Στο σημείο αυτό είναι σκόπιμο να κάνουμε μια σύντομη αναφορά στο πρότυπο SQL που χρησιμοποιήθηκε για να εκφράσουμε την αναζήτηση στη βάση δεδομένων (εντολή SELECT). Η SQL είναι το πρότυπο αλληλεπίδρασης με βάσεις δεδομένων στις οποίες τα δεδομένα είναι οργανωμένα σε συσχετισμένους πίνακες (σχεσιακό μοντέλο). Το πρότυπο περιλαμβάνει εντολές για τη δημιουργία πινάκων και εντολές για τη διαχείριση της πληροφορίας (εισαγωγή/διαγραφή/τροποποίηση εγγραφών στους πίνακες), καθώς και εντολές ερωτήσεων για την ανάγνωση εγγραφών από τους πίνακες. Αν και είναι πρότυπο, διαφορετικά συστήματα έχουν αποκλίσεις στη σύνταξη. Για παράδειγμα, η εντολή δημιουργίας του πίνακα Users που είδαμε έχει την εξής σύνταξη σε διαφορετικές βάσεις δεδομένων (σύμφωνα με το εργαλείο εξαγωγής κώδικα SQL του εργαλείου DBDesigner):

```
-- sqlite
CREATE TABLE Users (
    userID integer PRIMARY KEY AUTOINCREMENT,
    userName text
);
-- mysql
 CREATE TABLE `Users` (
    `userID` INT NOT NULL AUTO_INCREMENT,
    `userName` TEXT NOT NULL UNIQUE,
    PRIMARY KEY (`userID`)
);
--postgreSQL
CREATE TABLE "public.Users" (
    "userID" serial NOT NULL,
    "userName" TEXT NOT NULL UNIQUE,
    CONSTRAINT "Users_pk" PRIMARY KEY ("userID")
) WITH (
  OIDS=FALSE
);
```

Χαρακτηριστικά παραδείγματα εντολών SQL δίνονται στη συνέχεια, χρησιμοποιώντας τις συναρτήσεις του

Μοντέλου της εφαρμογής μας.

```
// συνάρτηση newBook, εισαγωγή νέας εγγραφής στον πίνακα Book
sql = `INSERT INTO "Books" ("title", "author", "comment", "user")
        VALUES ('${book.title}', '${book.author}', '${book.comment}',
'${book.user}');`;
// συνάρτηση findBook, αναζήτηση βιβλίου με κωδικό bookID
sql = `SELECT * FROM "Books" WHERE "bookID" = '${bookID}'`;
// συνάρτηση updateBook, ενημέρωση εγγραφής βιβλίου
sql = `UPDATE "Books"
        SET "title" = '${book.title}', author = '${book.author}', comment
= '${book.comment}'
        WHERE ("bookID" = '${book.bookID}');`;
// συνάρτηση deleteBook, διαγραφή βιβλίου με κωδικό bookID
sql = `DELETE FROM "Books"
        WHERE  "bookID" = '${bookID}';`;
// συνάρτηση insertUser, εισαγωγή νέου χρήστη
const sql = `INSERT INTO "Users"("userName") VALUES ('${userName}')
RETURNING "userID";`
// συνάρτηση findUser, αναζήτηση χρήστη με όνομα userName
sql = `SELECT * FROM "Users" WHERE "userName" = '${userName}';`;
```

Θα πρέπει να σημειωθεί ότι στα παραπάνω παραδείγματα δημιουργήσαμε την εντολή SQL με συμβολοσειρές-πρότυπα (template literals, Ενότητα 8.3.4), τεχνική που όπως αναφέρθηκε μπορεί να προκαλέσει κίνδυνο στην ασφάλεια του συστήματος λόγω SQL injection (ο χρήστης δίνει συμβολοσειρά σε πεδίο φόρμας που του ζητείται, στην οποία περιέχεται κώδικας SQL, ο οποίος όταν εκτελεστεί προκαλεί βλάβη). Εδώ χρησιμοποιήθηκε για να βελτιώσει την αναγνωσιμότητα του κώδικα. Η προτεινόμενη εκτέλεση εντολών SQL είναι με πέρασμα παραμέτρων, όπως είδαμε στο παράδειγμα της συνάρτησης getMyBooks πιο πάνω.

13.5 ORM (Object Relational Mapper)

Μια επιλογή που χρησιμοποιείται συχνά για σύνδεση της εφαρμογής με μια βάση δεδομένων, όπως η PostgreSQL, είναι η χρήση ενός συστήματος που επιτρέπει την αντιστοίχιση των αντικειμένων (δεδομένων) της εφαρμογής με τους πίνακες της βάσης δεδομένων. Το σύστημα αυτό ονομάζεται γενικά **Object-Relational Mapper (ORM)**. Αν χρησιμοποιήσουμε ένα τέτοιο σύστημα, δημιουργούμε ένα μοντέλο δεδομένων το οποίο αντιστοιχεί στους πίνακες και εν συνεχεία κάνουμε πράξεις στο μοντέλο, χωρίς να είμαστε υποχρεωμένοι να δίνουμε εντολές SQL προς τη βάση, όπως κάναμε στις προηγούμενες ενότητες. Ένα πλεονέκτημα αυτής της προσέγγισης είναι επίσης ότι το ORM «κρύβει» τις ιδιαιτερότητες κάθε συστήματος βάσεων δεδομένων και ο ίδιος κώδικας μπορεί να χρησιμοποιηθεί για σύνδεση σε διαφορετικά συστήματα βάσεων δεδομένων.

Ένα δημοφιλές σύστημα αντιστοίχισης δεδομένων-σχεσιακών πινάκων είναι το Sequelize, ενώ για τη μη σχεσιακή βάση δεδομένων MongoDB το αντίστοιχο σύστημα είναι το Mongoose, το οποίο θα δούμε σε επόμενη ενότητα.

Ένα παράδειγμα ορισμού ενός μοντέλου για έναν πίνακα Book(title, author) είναι το εξής:

```
const Book = sequelize.define(
'Book', {
    title: {
        type: DataTypes.TEXT,
        primaryKey: true,
        unique: true},
    author: {
        type: DataTypes.TEXT,
        allowNull: false},
});
```

Ενώ ένα παράδειγμα πράξης στο μοντέλο αυτό είναι:

```
Book.findAll( { where: {author: "Θερβάντες"} });
```

Η πράξη αυτή αντιστοιχεί στην εντολή SQL

```
SELECT * FROM Book WHERE author="Θερβάντες";
```

13.6 Φιλοξενία της εφαρμογής και της βάσης δεδομένων στο Heroku

Στις προηγούμενες ενότητες είδαμε πώς υλοποιήσαμε το τμήμα *Μοντέλου* της εφαρμογής μας, στην τελευταία μάλιστα περίπτωση με σύνδεση σε τοπικό εξυπηρετητή PostgreSQL. Τώρα θα δούμε ποια είναι τα βήματα που θα χρειαστούν για φιλοξενία της εφαρμογής αλλά και της βάσης δεδομένων στην πλατφόρμα Heroku. Είναι κατανοητό ότι το τελευταίο αυτό βήμα είναι βασικό στοιχείο της ροής εργασιών ανάπτυξης μιας διαδικτυακής εφαρμογής, αφού πρέπει η εφαρμογή να αποκτήσει δημόσια πρόσβαση, και αυτό διευκολύνεται με χρήση υπηρεσιών όπως η Heroku ή άλλων αντίστοιχων πλατφορμών φιλοξενίας διαδικτυακών εφαρμογών.

Σε προηγούμενη ενότητα (11.8.5) έγινε περιγραφή των βημάτων που απαιτούνται ώστε η εφαρμογή μας να ανέβει στην πλατφόρμα Heroku. Θα πρέπει να δημιουργήσουμε δωρεάν λογαριασμό, να εγκαταστήσουμε και να αρχικοποιήσουμε το git στο πρότζεκτ μας, να συνδέσουμε το περιβάλλον με τη νέα εφαρμογή στο Heroku και, τέλος, να ανεβάσουμε το πρότζεκτ σε ένα Heroku dyno. Θα πρέπει να σημειωθεί ότι η πλατφόρμα αυτή κατήργησε τα δωρεάν dynos εκτός από εκπαιδευτική ή μη κερδοσκοπική χρήση. Συνεπώς, θα πρέπει να κάνουμε αίτηση για δωρεάν υπηρεσία ως μη κερδοσκοπική ή εκπαιδευτική δραστηριότητα ώστε να τη χρησιμοποιήσουμε.

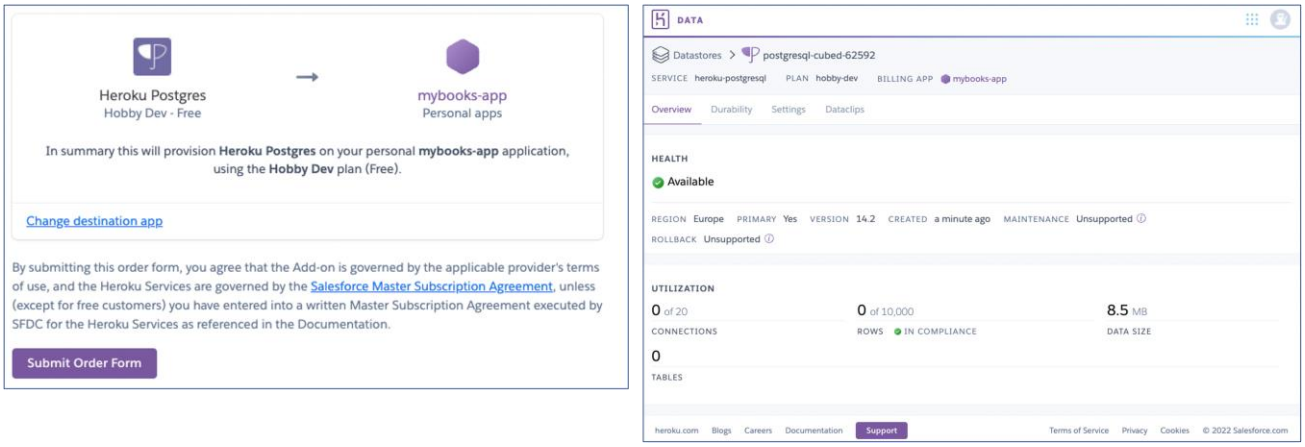
Αφού έχουν γίνει όλα αυτά τα βήματα, θα δούμε εδώ πώς θα συνδέσουμε την εφαρμογή με τη βάση δεδομένων μας.

Αφού συνδεθούμε στο περιβάλλον Heroku, επιλέγουμε να κάνουμε αίτημα για υπηρεσία δεδομένων (data). Οι διαθέσιμες υπηρεσίες σήμερα περιλαμβάνουν: Heroku Postgres, Heroku Redis και Heroku Apache Kafka. Μια σύντομη περιγραφή καθεμίας από τις υπηρεσίες αυτές:

- Η Heroku Postgres είναι η υπηρεσία του Heroku για παροχή ενός εξυπηρετητή PostgreSQL με δυνατότητες για συνεχή εφεδρεία των δεδομένων. Παρέχονται διάφορες επιλογές πλάνων χρήσης της υπηρεσίας αυτής, hobby, standard, premium και enterprise. Για εκπαιδευτικούς σκοπούς αρκεί το πλάνο hobby dev, το οποίο διατίθεται δωρεάν. Το πλάνο αυτό έχει σήμερα περιορισμούς, όπως 1 GB χώρο δεδομένων, μέγιστο 20 ταυτόχρονων συνδέσεων και 10.000 εγγραφές. Αυτό το πλάνο θα χρησιμοποιήσουμε στο παράδειγμά μας.
- Η Heroku Redis είναι μια υπηρεσία που στηρίζεται στη Redis, μια βάση δεδομένων τύπου κλειδί-τιμή που παραμένει στην κεντρική μνήμη, η οποία χρησιμοποιείται συχνά ως κρυφή μνήμη, για παράδειγμα για αποθήκευση πολλαπλών συνεδριών (συνδέσεων) ενός εξυπηρετητή.
- Το Apache Kafka είναι μια πλατφόρμα δεδομένων ροής ανοικτού κώδικα που λειτουργεί όπως μια παραδοσιακή ουρά μηνυμάτων publish-subscribe, η οποία μας επιτρέπει να δημοσιεύουμε και να εγγραφόμαστε σε ροές μηνυμάτων.

Οι τρεις αυτές υπηρεσίες εξυπηρετούν διαφορετικές ανάγκες. Η Heroku Postgres είναι αυτή που εξυπηρετεί την ανάγκη μόνιμης αποθήκευσης δομημένων δεδομένων σχεσιακού τύπου, όπως η βάση δεδομένων που σχεδιάσαμε για την εφαρμογή myBooks.

Καταθέτουμε το αίτημα συνδρομής στην υπηρεσία, κατά το οποίο θα μας ζητηθεί να ορίσουμε την εφαρμογή Heroku στην οποία θα παρέχεται η υπηρεσία. Στην ***Εικόνα 13.6*** φαίνεται το αίτημα που αφορά τη σύνδεση της Heroku Postgres (hoby dev plan) με την εφαρμογή mybooks-app (το όνομα που διαλέξαμε για την εφαρμογή μας στο Heroku). Στη συνέχεια, όταν το αίτημα γίνει δεκτό, μπορούμε να επισκεφθούμε την οθόνη που μας παρέχει τις πληροφορίες για την υπηρεσία με την οποία συνδέσαμε την εφαρμογή μας, καθώς και χρήσιμες πληροφορίες όπως το DATABASE_URL, δηλαδή τον σύνδεσμο μέσω του οποίου θα συνδεθούμε με τη βάση δεδομένων που θα δημιουργήσουμε στον εξυπηρετητή Heroku Postgres. Η οθόνη με τα στοιχεία της υπηρεσίας Heroku Postgres φαίνεται επίσης στην ***Εικόνα 13.6***.



Εικόνα 13.6 *Σύνδεση στη Heroku Postgres: (α) το αίτημα σύνδεσης, (β) η οθόνη με τα στοιχεία της νέας σύνδεσης.*

Το επόμενο βήμα είναι να δημιουργήσουμε τη βάση δεδομένων της εφαρμογής μας στον εξυπηρετητή Heroku Postgres και να συνδέσουμε την εφαρμογή μας με αυτόν.

Για τη δημιουργία της βάσης δεδομένων, χρησιμοποιούμε τα εργαλεία γραμμής εντολών της τοπικής postgres στο περιβάλλον ανάπτυξης μέσω του Heroku.

Για σύνδεση με τον εξυπηρετητή μέσω της γραμμής εντολών δίνουμε την εντολή:

```
heroku pg:psql
```

Στο περιβάλλον αυτό μπορούμε μέσω εντολών SQL να δημιουργήσουμε τους πίνακες της βάσης δεδομένων μας “Books” και “Users”, καθώς και να εισαγάγουμε δεδομένα σε αυτούς. Η διαδικασία αυτή μπορεί να επιταχυνθεί αν χρησιμοποιήσουμε το αρχείο SQL που παράγεται από την τοπική εγκατάσταση της βάσης δεδομένων, όπως περιγράφηκε στην προηγούμενη παράγραφο.

Μπορούμε να βεβαιωθούμε για τη δημιουργία της βάσης και την εισαγωγή δεδομένων με χρήση της SQL:

```
mybooks-app::DATABASE=> select * from "Users";
 userID | userName
--------+-----------
      1 | Nikos
      2 | Ευριπίδης
(2 rows)
```

Τέλος, η σύνδεση της εφαρμογής μας με τη νέα βάση δεδομένων γίνεται, όπως περιγράφηκε στην προηγούμενη ενότητα, με χρήση της συμβολοσειράς DATABASE_URL που δημιούργησε η Heroku Postgres.

Τη νέα έκδοση της εφαρμογής θα πρέπει να ανεβάσουμε στο Heroku μέσω git. Έχοντας ολοκληρώσει τη διαδικασία, μπορούμε με χρήση του URL της εφαρμογής μας, που είναι της μορφής https://<όνομα-εφαρμογής>.herokuapp.com, να συνδεθούμε στην εφαρμογή. Για την εφαρμογή μας αυτό το URL είναι το https://mybooks-app.herokuapp.com/. Τώρα μπορούμε πλέον να κάνουμε πράξεις στα δεδομένα της βάσης δεδομένων Heroku Postgres μέσω της εφαρμογής, όπως μπορούμε να διαπιστώσουμε από έλεγχο του περιεχομένου της βάσης δεδομένων από τη γραμμή εντολών του pg:psql.

13.7 Σύνδεση με βάση δεδομένων MongoDB

Στις προηγούμενες ενότητες συνδέσαμε την εφαρμογή μας με μια σχεσιακή βάση δεδομένων, η οποία οργανώνει τα δεδομένα μας ως δύο συσχετιζόμενους πίνακες, Books και Users. Η αλληλεπίδραση με τα δεδομένα αυτά γίνεται χρησιμοποιώντας το πρότυπο SQL. Αυτός ο τρόπος οργάνωσης και αλληλεπίδρασης με μόνιμη αποθήκευση δεδομένων είναι ο πιο διαδεδομένος. Είναι χαρακτηριστικό ότι το σχεσιακό μοντέλο ορίστηκε κατά τη δεκαετία του 1970, δηλαδή έχει ζωή πάνω από μισό αιώνα, παρ’ όλα αυτά παραμένει κυρίαρχο μέχρι σήμερα.

Στην πορεία του χρόνου έχουν προταθεί άλλοι τρόποι οργάνωσης των δεδομένων και άλλες γλώσσες αλληλεπίδρασης με δεδομένα, κάτω από τον γενικό τίτλο μη SQL βάσεις δεδομένων. Μια βάση δεδομένων

αυτής της κατηγορίας που ξεχωρίζει και έχει επίσης ευρεία χρήση σε εφαρμογές διαδικτύου είναι η MongoDB.

Ο όρος mongoDB προέρχεται από την αγγλική λέξη "Humongous DB", σημαίνει huge-monstrous, δηλαδή τερατωδώς μεγάλη, πελώρια, υποδεικνύοντας ότι η τεχνολογία αυτή απευθύνεται σε βάσεις δεδομένων με μεγάλο όγκο δεδομένων και εγγραφών, όπως συμβαίνει συχνά στο διαδίκτυο. Η τεχνολογία αυτή είναι ελεύθερα διαθέσιμη από τη mongodb.com, η οποία προσφέρει επίσης μια πλατφόρμα φιλοξενίας εξυπηρετητών, τη MongoDB Atlas, η οποία για μικρές σε μέγεθος βάσεις δεδομένων προσφέρεται δωρεάν, συνεπώς αποτελεί καλή επιλογή για πειραματισμό με αυτό τον τύπο βάσης δεδομένων για εκπαιδευτικούς σκοπούς.

Σε μια βάση δεδομένων mongoDB η βασική οντότητα δεν είναι ο πίνακας όπως στις σχεσιακές βάσεις δεδομένων αλλά το **έγγραφο (document)**, μια δομή δεδομένων που παρουσιάζει ομοιότητες με αντικείμενα JavaScript. Οι εντολές διαχείρισης της βάσης δεδομένων για την οποία δεν ορίζεται εξαρχής το σχήμα, όπως στη SQL, είναι στο παρακάτω παράδειγμα:

```
db.Users.insertOne({
      "_id": 1,
      "name": "Nikos"
    });

db.Books.find(
    {user: 1},
    {title: 1, author: 1,  comment: 1 });

db.Books.update(
    {_id: 1},
    { $set: {title: "Τα ποιήματά μου"}});
```

Στο παράδειγμα αυτό έχουμε μια εντολή εισαγωγής ενός νέου χρήστη στη συλλογή εγγράφων Users, μια εντολή αναζήτησης των βιβλίων του χρήστη με κωδικό 1 (να επιστραφούν τα γνωρίσματα title, author και comment), καθώς και μια εντολή τροποποίησης του βιβλίου με κωδικό 1, με νέο τίτλο: «Τα ποιήματά μου».

Η σύνδεση της εφαρμογής μας με μια βάση δεδομένων MongoDB μπορεί να γίνει ακολουθώντας τα εξής βήματα:

1. Δημιουργία λογαριασμού στην πλατφόρμα MongoDB Atlas και δημιουργία μιας βάσης δεδομένων σε ένα starter cluster (προσοχή, χρησιμοποιήστε το MO Sandbox το οποίο έχει όρια 512MB). Από την επιλογή δημιουργίας χρήστη δημιουργούμε έναν χρήστη που θα έχει πρόσβαση στη βάση δεδομένων μας, ενώ από την επιλογή Network access επιλέγουμε IP white list / allow access from anywhere. Με αυτό τον τρόπο θα είναι δυνατή η σύνδεση στη βάση δεδομένων από τον υπολογιστή στον οποίο έχει εγκατασταθεί η εφαρμογή μας.
2. Στη συνέχεια επιλέγουμε σύνδεση στη βάση μας με το εργαλείο της γραμμής εντολών mongo shell, ακολουθώντας τις οδηγίες εγκατάστασης του εργαλείου αυτού για το λειτουργικό μας σύστημα, που επιτρέπει να συνδεθούμε στον εξυπηρετητή της mongodb από τη γραμμή εντολών μας. Χρησιμοποιούμε τη συμβολοσειρά σύνδεσης που παίρνουμε από τον εξυπηρετητή mongo, η οποία είναι συνήθως μια συμβολοσειρά που αρχίζει με "mongodb+srv://cluster0...". Θα πρέπει να σημειώσουμε ότι, αντίθετα από την περίπτωση σχεσιακών βάσεων δεδομένων, η mongo δεν απαιτεί δημιουργία πινάκων πρώτα, αλλά οι πίνακες δημιουργούνται αυτόματα και δομούνται σύμφωνα με τις εντολές εισαγωγής δεδομένων στη βάση δεδομένων.
3. Η σύνδεση της εφαρμογής με τη βάση δεδομένων που δημιουργήσαμε γίνεται με χρήση της βιβλιοθήκης mongoose. Ακολουθεί ένα παράδειγμα σύνδεσης μέσω της mongoose με μια βάση δεδομένων στον τοπικό υπολογιστή:

```
import mongoose from 'mongoose';

mongoose.connect('mongodb://localhost:27017/testDB', {
    useNewUrlParser: true,
    useUnifiedTopology: true});

const db = mongoose.connection
db.on('error', console.error.bind(console, 'connection error:'));
```

```
db.once('open', function() {
    // μήνυμα επιτυχούς σύνδεσης
});
```

Θα πρέπει να σημειωθεί ότι η mongoose είναι βιβλιοθήκη που επιτρέπει τη δημιουργία ενός σχήματος της βάσης δεδομένων, όπως γινόταν με τις σχεσιακές βάσεις δεδομένων, και στη συνέχεια την επικοινωνία με το σχήμα αυτό με χρήση δομών δεδομένων της JavaScript, αποκρύπτοντας τις σύνθετες εντολές της mongoDB. Η βιβλιοθήκη αυτή, όπως έχει ήδη αναφερθεί, ανήκει στην κατηγορία των βιβλιοθηκών αντιστοίχισης των δομών της γλώσσας προγραμματισμού στο μοντέλο δεδομένων της βάσης, γνωστών ως ORM (object-relational model) για τις σχεσιακές βάσεις δεδομένων, και ODM (object-document model) για την περίπτωση της mongoDB.

Έστω το παρακάτω παράδειγμα δημιουργίας του σχήματος βάσης δεδομένων και εν συνεχεία ενός μοντέλου το οποίο χρησιμεύει για δημιουργία αντικειμένων που εισάγονται στη βάση δεδομένων.

```
import mongoose from 'mongoose';

// σύνδεση με τη βάση δεδομένων
mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true },
(err, res) => {
    if (err) { console.log(err) }
});

const db = mongoose.connection;

// ορισμός σχήματος
const bookSchema = new mongoose.Schema({
    bookID: String,
    title: String,
    authors: String,
    comments: String,
    user: String
});

const userSchema = new mongoose.Schema({
    userID: String,
    userName: String
});

// δημιουργία μοντέλου από το σχήμα
const Book = mongoose.model('Book', bookSchema, 'Books');
const User = mongoose.model('User', userSchema, 'Users');

// έστω νέο βιβλίο
const book1 = new Book({
    title: 'Άπαντα',
    authors: "Καβάφης",
    comments: "", user: "1"
});

// save model to database
book1.save(function (err, book1) {
    if (err) return console.error(err);
    console.log(book1.title + " το βιβλίο αποθηκεύτηκε");
});
```

13.8 Ερωτήσεις αυτοαξιολόγησης

1. Το **σχεσιακό μοντέλο** ορίζει ως τρόπο οργάνωσης δεδομένων:
    1. σύνολο από αντικείμενα με ζευγάρια κλειδί-τιμή.
    2. συλλογές από έγγραφα.
    3. διασυνδεδεμένους πίνακες.
    4. ιεραρχικές δομές γράφων.
    5. κανένα από τα παραπάνω.
2. Η γλώσσα **SQL** επιτρέπει τις εξής ενέργειες:
    1. τη διαχείριση (εισαγωγή/ τροποποίηση/ διαγραφή) δεδομένων.
    2. τον ορισμό και διαχείριση των δομών των δεδομένων (δημιουργία/διαγραφή/τροποποίηση πινάκων δεδομένων).
    3. κανένα από τα παραπάνω.
    4. και τα δύο παραπάνω.
3. Ένα **Object-relational mapper (ORM)** είναι:
    1. μια βάση δεδομένων που χειρίζεται αντικείμενα.
    2. μια διεπαφή που επιτρέπει την αντιστοίχιση αντικειμένων σε πίνακες του σχεσιακού μοντέλου.
    3. μια βιβλιοθήκη που χειρίζεται πίνακες με χάρτες.
4. Σε έναν πίνακα ποιος είναι ο ρόλος του primary key;
    1. Είναι το πρώτο γνώρισμα του πίνακα.
    2. Είναι το πιο σημαντικό γνώρισμα του πίνακα.
    3. Είναι το γνώρισμα του πίνακα που για κάθε εγγραφή παίρνει διαφορετική τιμή.
    4. Είναι ένα γνώρισμα ή ένα σύνολο από γνωρίσματα που για κάθε εγγραφή παίρνουν διαφορετική τιμή.
5. Ποια είναι η εντολή SQL για δημιουργία ενός στοιχείου μιας βάσης δεδομένων;
    1. CREATE TABLE
    2. CREATE DATABASE
    3. CREATE RECORD
    4. BUILD RECORD
6. Σε έναν πίνακα με τα στοιχεία των μαθητών STUDENTS(name, year, city), που περιλαμβάνει όνομα, έτος γέννησης και πόλη γέννησης, πώς θα μάθουμε τα ονόματα των μαθητών που είναι από την Πάτρα;
    1. SELECT * FROM STUDENTS WHERE name = "Patra";
    2. SELECT name FROM STUDENTS WHERE year = "Patra";
    3. SELECT * FROM STUDENTS WHERE city = "Patra";
    4. SELECT name FROM STUDENTS WHERE city = "Patra";
    5. UPDATE city FROM STUDENTS where name = "Patra";
7. Μια βάση δεδομένων SQLite αποθηκεύεται σε:
    1. ένα και μοναδικό αρχείο που περιέχει τη βάση δεδομένων.
    2. σε έναν εξυπηρετητή SQLite server, αρκεί να γνωρίζουμε το όνομα της βάσης δεδομένων.
    3. είτε σε ένα αρχείο είτε σε έναν εξυπηρετητή, ανάλογα με την επιλογή μας.
8. Ποια από τις παρακάτω προτάσεις δεν είναι αληθής;
    1. Η PostgreSQL ακολουθεί το μοντέλο του πελάτη-εξυπηρετητή ενώ η SQLite όχι.
    2. Και οι δύο SQLite και PostgreSQL είναι σχεσιακές βάσεις δεδομένων.
    3. Το σύστημα διαχείρισης ασφάλειας και πρόσβασης της PostgreSQL είναι πολύ πιο σύνθετο από αυτό της SQLite.
    4. Η PostgreSQL χρησιμοποιείται για εμπορικές εφαρμογές, ενώ η SQLite κυρίως για εκπαιδευτικούς σκοπούς.
    5. Η SQLite και PostgreSQL είναι από τις πιο δημοφιλείς βάσεις δεδομένων.
9. Η διαδικασία ανεβάσματος μιας εφαρμογής στο Heroku περιλαμβάνει την εξής εντολή του heroku CLI: heroku git:remote -a application-name. Ποιο το νόημα αυτής της εντολής;
    1. Είναι η εντολή που μεταφέρει τα αρχεία από το τοπικό αποθετήριο στην εφαρμογή Heroku.
    2. Είναι η εντολή που ορίζει ότι κάθε φορά που εκτελούμε την εντολή git push heroku master θα ανεβάζει τα αρχεία από το τοπικό αποθετήριο στην εφαρμογή του Heroku.
    3. Είναι η εντολή που ορίζει αυτόματη σύνδεση της εφαρμογής στον τοπικό υπολογιστή και του εξυπηρετητή Heroku ώστε να γίνεται αυτόματα ο συγχρονισμός τους.

10. Η μεταφορά μιας εφαρμογής με σύνδεση στην PostgreSQL στο Heroku περιλαμβάνει τα εξής βήματα: (σημειώστε το βήμα που δεν απαιτείται)
    1. Δημιουργία πρότζεκτ git στον τοπικό υπολογιστή.
    2. Δημιουργία λογαριασμού στο Heroku.
    3. Δημιουργία εφαρμογής στο Heroku.
    4. Ανέβασμα της εφαρμογής από τον τοπικό υπολογιστή στο Heroku.
    5. Σύνδεση του Heroku Postgres data storage με την εφαρμογή στον τοπικό υπολογιστή.
    6. Σύνδεση του Heroku Postgres data storage με την εφαρμογή στο Heroku.
    7. Τροποποίηση των παραμέτρων της σύνδεσης της PostgreSQL ώστε να συνδεθεί με το Heroku data storage.

13.9 Βιβλιογραφία και Αναφορές

Ο αναγνώστης που επιθυμεί εμβάθυνση στα θέματα του κεφαλαίου αυτού θα πρέπει να μελετήσει το αντικείμενο των βάσεων δεδομένων, και ιδιαίτερα των σχεσιακών βάσεων δεδομένων και της SQL. Υπάρχει πλούσια βιβλιογραφία στο αντικείμενο αυτό. Βλέπε στην ελληνική βιβλιογραφία: Βερύκιος και Βασιλακόπουλος (2022), Γιαννακουδάκης (2014) και άλλα. Επίσης, έχουν μεταφραστεί στα ελληνικά αξιόλογα διεθνή διδακτικά εγχειρίδια, όπως αυτά των Date (1998), Elmasri και Navathe (2016) κ.λπ.

Ηλεκτρονικές πηγές για σχεσιακές βάσεις δεδομένων και SQL υπάρχουν πολλές, ενδεικτικά:

- sqlzoo
- w3schools
- tutorialspoint

Επίσης, επιμέρους συστήματα βάσεων δεδομένων περιλαμβάνουν εισαγωγικά μαθήματα και οδηγούς, όπως η τεκμηρίωση της mongoDB, της postgresql, της sqlite κ.λπ.

Ακόμη, θα πρέπει να σημειωθεί ότι αρκετά βιβλία-οδηγοί ανάπτυξης διαδικτυακών εφαρμογών αφιερώνουν κεφάλαια στη διασύνδεση του εξυπηρετητή με τη βάση δεδομένων. Για παράδειγμα το βιβλίο του Mardan (2018) αφιερώνει το κεφάλαιο 5 στη διασύνδεση ενός εξυπηρετητή node.js με μια βάση δεδομένων mongoDB.

Επιπλέον, οι συγγραφείς αυτού του βιβλίου έχουν δημιουργήσει *ανοιχτά διαδικτυακά μαθήματα* στην πλατφόρμα mathesis, υλικό από τα οποία έχει χρησιμοποιηθεί και σε αυτό το σύγγραμμα. Για αυτό το κεφάλαιο το σχετικό μάθημα είναι διαλέξεις του μαθήματος «Ανάπτυξη διαδικτυακών εφαρμογών με Node.js».

Α. Ξενόγλωσσες

Mardan, A. (2018). *Practical Node.js, Building Real-World Scalable Web Apps* (2nd ed.). Apress.

Β. Ελληνόγλωσσες

Βερύκιος, Β., & Βασιλακόπουλος, Μ. (2022). *Συστήματα Βάσεων Δεδομένων* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. https://repository.kallipos.gr/handle/11419/8413

Γιαννακουδάκης, Ε. Ι. (2014). *Βάσεις Δεδομένων, Θεωρία και Πράξη*. Εκδόσεις Μπένου, Αθήνα. Κωδικός βιβλίου στον Εύδοξο: 112694420.

Date, C. J. (1998). *Εισαγωγή στα συστήματα βάσεων δεδομένων* (Α΄ τόμος) (6η έκδ.). Εκδόσεις Κλειδάριθμος. Κωδικός βιβλίου στον Εύδοξο: 94644156

Elmasri, R., & Navathe, S. B. (2016). *Θεμελιώδεις Αρχές Συστημάτων Βάσεων Δεδομένων* (7η έκδ.). Εκδόσεις Δίαυλος. Κωδικός βιβλίου στον Εύδοξο: 50662846.

Σιντόρης, Χ., & Αβούρης, Ν. (2022). *Ανάπτυξη διαδικτυακών εφαρμογών με Node.js*. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr

Κεφάλαιο 14. Πρόσβαση χρηστών και αυθεντικοποίηση

Σύνοψη

Σε αυτό το κεφάλαιο θα συζητήσουμε κάποια από τα θέματα ασφάλειας που σχετίζονται με την πρόσβαση των χρηστών σε προστατευμένες περιοχές, την αυθεντικοποίησή τους, καθώς και τη φύλαξη των διαπιστευτηρίων τους με κρυπτογραφικό κατατεμαχισμό. Στη συνέχεια, θα παρουσιάσουμε τη διαχείριση συνεδριών με χρήση cookies, καθώς και θα συζητηθεί η ασφαλής μεταφορά δεδομένων μεταξύ πελάτη και εξυπηρετητή μέσω του πρωτοκόλλου HTTPS.

Προαπαιτούμενη γνώση

Τα θέματα που θίγονται στο κεφάλαιο αυτό αφορούν την πλευρά του εξυπηρετητή και, συνεπώς, τα παραδείγματα που χρησιμοποιούνται για τη διαχείριση των συνεδριών είναι υλοποιημένα σε Express.js (κεφάλαιο 12) σε περιβάλλον Node.js (κεφάλαιο 11). Είναι απαραίτητη λοιπόν η εξοικείωση με την JavaScript (κεφάλαια 7, 8, 9 και 10).

14.1 Εισαγωγή

Στο κεφάλαιο αυτό θα εξετάσουμε πώς μπορούμε να προστατέψουμε την πρόσβαση σε περιοχές του ιστοτόπου μας. Η προστασία της πρόσβασης προϋποθέτει τον έλεγχο της ταυτότητας του χρήστη, τη διακρίβωση δηλαδή από το σύστημα πως ο χρήστης είναι αυτός που ισχυρίζεται πως είναι, μια διαδικασία γνωστή και ως αυθεντικοποίηση (authentication). Η διακρίβωση της ταυτότητας, η αυθεντικοποίηση, γίνεται τυπικά με την υποβολή από την πλευρά του χρήστη ενός ονόματος χρήστη και ενός συνθηματικού. Το όνομα χρήστη είναι ένα μοναδικό αναγνωριστικό, με το οποίο μπορεί το σύστημα να ξεχωρίσει τους χρήστες μεταξύ τους. Το συνθηματικό είναι η απόδειξη πως ο χρήστης είναι όντως αυτός που ισχυρίζεται. Εφόσον διακριβωθεί η ταυτότητα του χρήστη, αυτός μπορεί να αποκτήσει (ή όχι) πρόσβαση σε συγκεκριμένες προστατευμένες περιοχές της εφαρμογής (authorization).

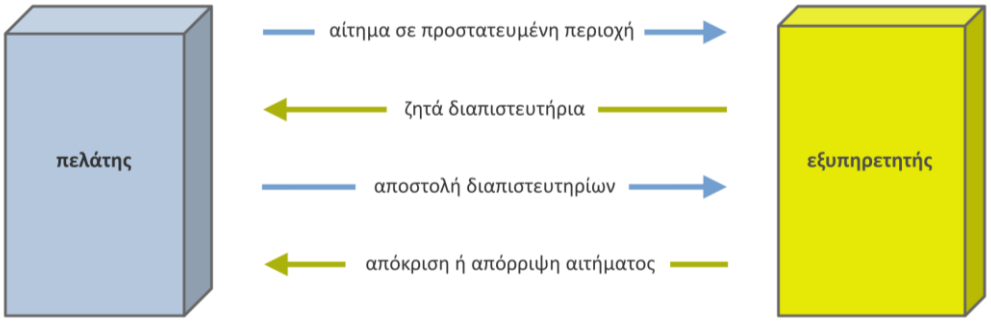
Στη συνέχεια του κεφαλαίου θα επεκτείνουμε το παράδειγμά μας, αυτό της εφαρμογής διαχείρισης των βιβλίων μιας παρέας, από το προηγούμενο κεφάλαιο (Ενότητα 13). Εκεί έχουν ήδη χρησιμοποιηθεί κάποιες από τις τεχνικές που θα δούμε εδώ, χωρίς όμως να εξηγηθούν με λεπτομέρεια. Πριν όμως καταπιαστούμε με την εφαρμογή, θα εξετάσουμε με κάποια λεπτομέρεια θέματα όπως η προστασία των διαπιστευτηρίων.

14.2 Αυθεντικοποίηση

Η ιδέα της αυθεντικοποίησης (authentication) χρηστών είναι απλή: Πρέπει πρώτα να αναγνωρίσει το σύστημα τον χρήστη ώστε να του επιτρέψει την πρόσβαση σε κάποιες από τις προστατευμένες περιοχές του. Η αναγνώριση του χρήστη μπορεί να γίνει με διάφορους τρόπους ή και με συνδυασμό τους:

- με κάτι που ο χρήστης αποδεικνύει πως γνωρίζει, όπως ένα συνθηματικό (pin) κ.λπ.,
- με κάτι που ο χρήστης αποδεικνύει πως κατέχει, όπως ένα ψηφιακό πιστοποιητικό, ένα κινητό τηλέφωνο κ.λπ.,
- χρησιμοποιώντας βιομετρικά χαρακτηριστικά όπως το δακτυλικό αποτύπωμα ή κάποια χαρακτηριστικά της συμπεριφοράς του κ.λπ.

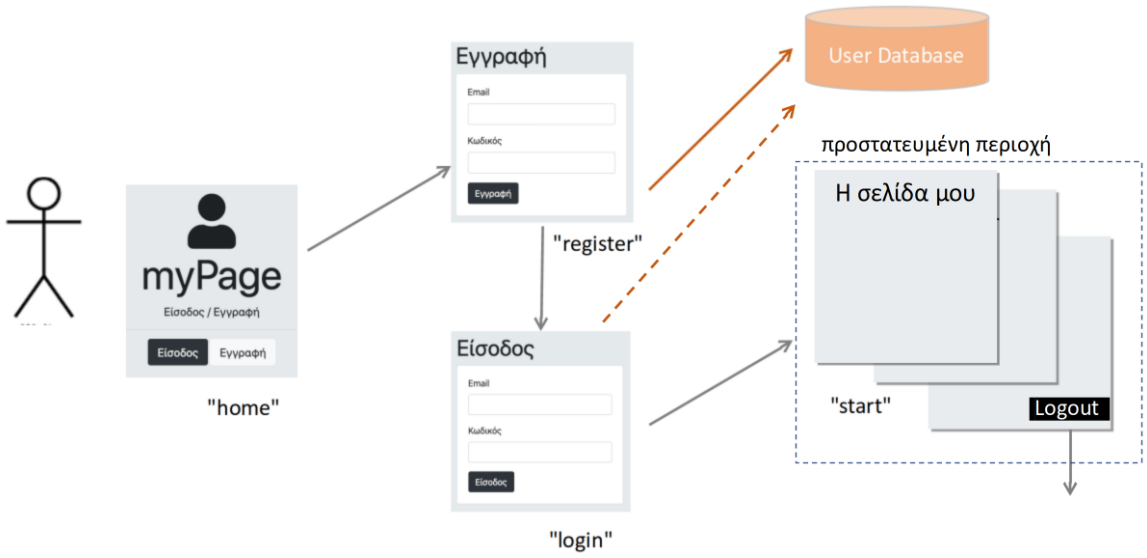
Στην πιο συνήθη μορφή της η διαδικασία της αυθεντικοποίησης περιλαμβάνει την υποβολή από τον χρήστη των διαπιστευτηρίων του (***Εικόνα 14.1***).



Εικόνα 14.1 *Αυθεντικοποίηση χρήστη χρησιμοποιώντας κάποιο είδος διαπιστευτηρίων.*

14.2.1 Απαιτήσεις από το σύστημα διαχείρισης χρηστών

Στην ***Εικόνα 14.2*** φαίνεται εποπτικά μια τυπική ροή στη διαδικασία αυθεντικοποίησης. Ο χρήστης που ζητάει, για παράδειγμα, πρόσβαση σε μια προστατευμένη περιοχή παραπέμπεται σε μια φόρμα εισόδου. Στη φόρμα αυτή υποβάλλει τα στοιχεία του, το μέιλ του και το συνθηματικό. Το σύστημα, για να διαπιστώσει αν ο χρήστης είναι αυτός που ισχυρίζεται (ο ιδιοκτήτης του συγκεκριμένου μέιλ), συγκρίνει το συνθηματικό που έδωσε ο χρήστης με αυτό που έχει αποθηκεύσει στη βάση δεδομένων για το συγκεκριμένο μέιλ. Αυτή η αποθήκευση έγινε κάποια στιγμή στο παρελθόν, κατά την εγγραφή του χρήστη. Εφόσον διαπιστωθεί πως το συνθηματικό που έδωσε ο χρήστης ταιριάζει με το συνθηματικό που είναι αποθηκευμένο στη βάση δεδομένων, τότε επιτρέπεται η πρόσβαση στην προστατευμένη περιοχή.



Εικόνα 14.2 *Κάποια από τα περιεχόμενα της ιστοσελίδας μπορεί να βρίσκονται σε προστατευμένη περιοχή. Μόνο χρήστες που έχουν αυθεντικοποιηθεί μπορούν να έχουν πρόσβαση.*

Συνοπτικά, οι βασικές απαιτήσεις από το σύστημά μας είναι:

1. Να μην επιτρέπει σε **μη αυθεντικοποιημένους** χρήστες να έχουν πρόσβαση στο **προστατευμένο τμήμα** της εφαρμογής.
2. Να παρέχει μηχανισμούς ώστε, για ένα εύλογο διάστημα, να επιτρέπεται στους χρήστες που έχουν αναγνωριστεί να επιστρέφουν στην προστατευμένη περιοχή χωρίς να χρειάζεται να περάσουν εκ νέου από διαδικασία αυθεντικοποίησης.
3. Να φυλάσσει τα **διαπιστευτήρια** (credentials) των χρηστών, όπως π.χ. το συνθηματικό, σε τέτοια μορφή έτσι ώστε αυτοί να είναι προστατευμένοι *ακόμη και αν τα διαπιστευτήρια πέσουν στα χέρια τρίτων*.

14.2.2 Φύλαξη των διαπιστευτηρίων

Τα διαπιστευτήρια του χρήστη χρειάζεται να είναι διαθέσιμα στο σύστημα ώστε να μπορούν κάθε φορά να συγκριθούν με αυτά που παρέχει ο χρήστης που ζητά πρόσβαση.

Ο πιο απλός τρόπος είναι η αποθήκευση του ζεύγους **όνομα χρήστη / συνθηματικό** στη βάση δεδομένων του συστήματος.

Χάριν παραδείγματος, θα μπορούσε σε ένα απλοϊκό –και καθόλου ασφαλές– σύστημα η αποθήκευση των διαπιστευτηρίων να γίνεται σε ένα αρχείο JSON. Σε ένα τέτοιο σύστημα τόσο το αναγνωριστικό μέιλ του χρήστη όσο και το συνθηματικό είναι αποθηκευμένα σε απλό, **μη κρυπτογραφημένο κείμενο** (cleartext).

```
{ "_id" : "1", "email" : "a@a.com", "password" : "έναμυστικό"}
```

Ο κίνδυνος που έχει μια τέτοια προσέγγιση είναι πολύ μεγάλος, καθώς στην περίπτωση που το αρχείο JSON πέσει στα χέρια τρίτων, τότε αυτοί θα έχουν στη διάθεσή τους τα στοιχεία πρόσβασης των χρηστών της εφαρμογής μας. Ακόμη χειρότερα, καθώς οι χρήστες συνηθίζουν να επαναχρησιμοποιούν συνθηματικά, πολύ πιθανόν ο συνδυασμός "a@a.com" και "έναμυστικό" να έχει χρησιμοποιηθεί και σε άλλες υπηρεσίες από αυτό τον χρήστη. Έτσι, οι χρήστες είναι ευάλωτοι σε επιθέσεις με πολύ σοβαρές συνέπειες.

Είναι σημαντικό λοιπόν να προστατευτούν τα διαπιστευτήρια έτσι ώστε ακόμη και αν γίνουν γνωστά σε τρίτους να μην αποτελούν κίνδυνο για τους χρήστες. Ευτυχώς, η προστασία αυτή είναι απλή διαδικασία και συνιστάται να γίνεται ***πάντα***, ακόμη και σε εφαρμογές που κατασκευάζουμε μόνο και μόνο για να μάθουμε ή για να δοκιμάσουμε κάτι ή για επίδειξη κάποιας πρωτότυπης λειτουργίας.

14.2.3 Προστασία διαπιστευτηρίων

Η προστασία των διαπιστευτηρίων μπορεί να γίνει χρησιμοποιώντας μια μονόδρομη κρυπτογραφική συνάρτηση κατατεμαχισμού (one-way cryptographic hashing function). Η συνάρτηση κατατεμαχισμού έχει μερικά ενδιαφέροντα χαρακτηριστικά που την κάνουν πιο χρήσιμη από την κρυπτογράφηση, όπως περιγράφεται στη συνέχεια. Στην κρυπτογράφηση (***Εικόνα 14.3***) το κρυπτογραφημένο μήνυμα έχει μεταβλητό μέγεθος, που εξαρτάται από το μέγεθος που έχει το αρχικό, μη κρυπτογραφημένο μήνυμα. Επιπλέον, όποιος κατέχει το κλειδί αποκρυπτογράφησης μπορεί να ανακτήσει το αρχικό μήνυμα.

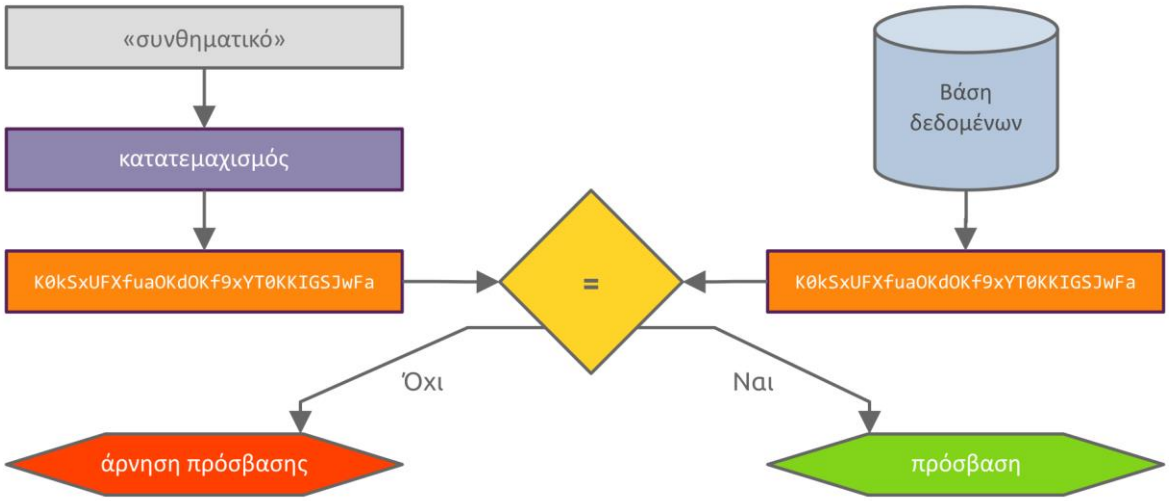
Εικόνα 14.3 *Η κρυπτογράφηση ενός μηνύματος παράγει κείμενο ανάλογου μεγέθους με το αρχικό.*

Αντίθετα, ο μονόδρομος κατατεμαχισμός δεν παράγει κρυπτογραφημένο μήνυμα, αλλά μια **περίληψη** (**digest** ή **hash**), η οποία έχει πάντα σταθερό μέγεθος, ανεξάρτητα από το μέγεθος του αρχικού μηνύματος. Επιπλέον, όντας μονόδρομη, η συνάρτηση κατατεμαχισμού παράγει περίληψη τέτοια ώστε να είναι αδύνατη η ανάκτηση του αρχικού μηνύματος.

Εικόνα 14.4 *Ο μονόδρομος κατατεμαχισμός (one-way hash) παράγει αλφαριθμητικό σταθερού μήκους.*

Με τη χρήση της συνάρτησης κατατεμαχισμού δεν χρειάζεται πια να φυλάξουμε το συνθηματικό του χρήστη στη βάση δεδομένων. Κατά την εγγραφή του χρήστη υπολογίζουμε την περίληψή του (hash) και αποθηκεύουμε αυτή. Σε κάθε μελλοντική διαδικασία αυθεντικοποίησης ο χρήστης μάς παρέχει το συνθηματικό του, του

οποίου, σε κάθε αυθεντικοποίηση, υπολογίζουμε εκ νέου την περίληψη και τη συγκρίνουμε με αυτή που έχουμε στη βάση δεδομένων. Έτσι, μπορούμε πλέον να υπολογίσουμε και να αποθηκεύσουμε την περίληψη και όχι το ίδιο το συνθηματικό (***Εικόνα 14.5***).



Εικόνα 14.5 *Η αυθεντικοποίηση γίνεται συγκρίνοντας τις περιλήψεις.*

Υπάρχουν πολλές διαθέσιμες συναρτήσεις κατατεμαχισμού, που μπορεί να χρησιμοποιούνται για διαφορετικούς σκοπούς. Για τον σκοπό της αυθεντικοποίησης επιθυμούμε η συνάρτηση να έχει ορισμένα χαρακτηριστικά. Η συνάρτηση κατατεμαχισμού, H(x), επιθυμούμε να παράγει μια περίληψη ενός μηνύματος οποιουδήποτε μεγέθους:

- εύκολα και γρήγορα. Γρήγορα όσον αφορά την ανθρώπινη κλίμακα, π.χ. ελάχιστα δευτερόλεπτα ή και λιγότερο, αλλά αργά όσον αφορά την κλίμακα του υπολογιστή, ώστε να επιβραδύνονται τυχαίες αναζητήσεις (brute force),
- η περίληψη να είναι **ίδιου** μεγέθους για οποιοδήποτε μέγεθος κειμένου,
- για το ίδιο κείμενο να παράγεται **πάντα** η **ίδια περίληψη**,
- αν έχουμε μια συγκεκριμένη περίληψη **d**, να είναι **υπολογιστικά ανέφικτο** να βρούμε τέτοιο μήνυμα **m** που να παράγει την ίδια περίληψη, δηλαδή **H(m) = d** (*preimage resistance*),
- για οποιοδήποτε μήνυμα **m** να είναι **υπολογιστικά ανέφικτο** να βρούμε διαφορετικό μήνυμα **m'**, για το οποίο να παράγεται η ίδια περίληψη **d**, δηλαδή **H(m) = H(m') = d** (*2nd preimage resistance*),
- να είναι **υπολογιστικά ανέφικτο** να βρεθεί ζεύγος **(m, m')** για το οποίο να ισχύει: **H(m) = H(m')** (*collision resistance*).

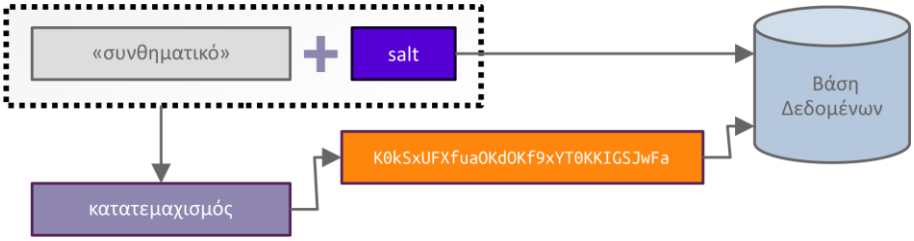
Δεν έχουν όλες οι συναρτήσεις κατατεμαχισμού αυτά τα χαρακτηριστικά και, συνεπώς, δεν είναι όλες κατάλληλες για χρήση στη διαδικασία της αυθεντικοποίησης. Για παράδειγμα, ο αποδεδειγμένα μη ασφαλής αλγόριθμος κατατεμαχισμού MD5 χρησιμοποιούνταν ακόμη και μέχρι πρόσφατα (2019) σε συστήματα που με κάποιον τρόπο αποθηκεύουν διαπιστευτήρια χρηστών. Ο αλγόριθμος SHA-1 είναι επίσης ένας μη ασφαλής αλγόριθμος, για τον οποίο οι συστάσεις μη χρήσης είναι πιο πρόσφατες. Οι μη ασφαλείς συναρτήσεις κατατεμαχισμού συνεχίζουν να είναι χρήσιμες, όπως για παράδειγμα για να διαπιστώσουμε αν δύο έγγραφα ή αρχεία είναι ίδια, αλλά είναι ακατάλληλες για τη φύλαξη των διαπιστευτηρίων.

Ακόμη όμως και με τη χρήση κατατεμαχισμού, ο κίνδυνος να αποκαλυφθεί το συνθηματικό σε κάποια επίθεση παραμένει, για διάφορους λόγους. Μια από τις σοβαρότερες αδυναμίες είναι πως πολλές φορές οι χρήστες χρησιμοποιούν συνηθισμένα συνθηματικά, όπως 'password', 'secret', '123456' κ.ά. Είναι διαθέσιμες αρκετές τέτοιες συλλογές συνθηματικών. Έτσι, μπορεί κάποιος να υπολογίσει τις περιλήψεις για συνήθη συνθηματικά, να τις συγκρίνει με αυτές που είναι αποθηκευμένες στη βάση δεδομένων όπου φυλάμε τα στοιχεία πρόσβασης και να αποκαλύψει έτσι κάποια τουλάχιστον από τα διαπιστευτήρια των χρηστών.

Άλλη μια σοβαρή αδυναμία είναι πως, επειδή η εκτέλεση της συνάρτησης κατατεμαχισμού έχει κάποιο υπολογιστικό κόστος, είναι πολλές φορές πιο εύκολο να συγκριθεί η περίληψη με περιλήψεις που έχουν υπολογιστεί από πριν και είναι αποθηκευμένες σε έναν μεγάλο πίνακα. Αυτού του είδους η επίθεση είναι γνωστή ως rainbow table attack.

14.2.4 Κατατεμαχισμός με «αλάτισμα»

Υπάρχει όμως μια εύκολη λύση που προφυλάσσει και από τα παραπάνω σενάρια. Η προσθήκη ενός τυχαίου αλφαριθμητικού, το λεγόμενο «αλάτι» (**salt**), κατά την παραγωγή της περίληψης είναι εύκολη και ισχυρή άμυνα σε τέτοιες επιθέσεις. Στη βάση δεδομένων όπου φυλάσσονται τα διαπιστευτήρια αποθηκεύονται πλέον δύο πληροφορίες, το αλάτι και η τελική περίληψη. Κάθε φορά που ο χρήστης εισάγει τον κωδικό του στο σύστημα (**Εικόνα *14.6***) το σύστημα ανασύρει το αλάτι από τη βάση δεδομένων και το προσθέτει στο συνθηματικό που έδωσε ο χρήστης. Σε αυτό τον συνδυασμό **συνθηματικό** + **αλάτι** υπολογίζεται η περίληψη και συγκρίνεται με την περίληψη που είχε δημιουργηθεί όταν έγινε η αρχική εγγραφή του χρήστη.



Εικόνα 14.6 *Κατατεμαχισμός με αλάτι. Η περίληψη θα προκύψει από συνθηματικό + αλάτι, όπου το αλάτι είναι γνωστό στο σύστημα και διαφορετικό για κάθε συνθηματικό.*

Το αλάτι είναι ένα τυχαίο αλφαριθμητικό που παράγει το σύστημα κατά την εγγραφή του χρήστη στο σύστημα. Είναι διαφορετικό για κάθε αποθηκευμένο συνθηματικό. Ακόμη και αν δύο ή περισσότεροι χρήστες έχουν επιλέξει το ίδιο συνθηματικό, η τελική περίληψη θα είναι διαφορετική.

Να σημειωθεί πως το αλάτισμα δεν προστατεύει όταν το συνθηματικό είναι συνηθισμένο, όταν δηλαδή ο επιτιθέμενος χρησιμοποιεί ένα «λεξικό» συχνά χρησιμοποιούμενων συνθηματικών και τα δοκιμάζει ένα ένα μέχρι να βρει κάποιο που παράγει την ίδια περίληψη. Αυτό γίνεται επειδή το αλάτι αποθηκεύεται μαζί με την περίληψη στη βάση δεδομένων όπου φυλάσσονται τα διαπιστευτήρια του χρήστη. Συνεπώς, αν κάποιος αποκτήσει πρόσβαση στην περίληψη, ξέρει και το αλάτι.

14.2.5 Η συνάρτηση κατατεμαχισμού bcrypt

Στη Node.js η ενσωματωμένη βιβλιοθήκη crypto παρέχει μια σειρά από συναρτήσεις κρυπτογράφησης με διάφορες εφαρμογές, μεταξύ των οποίων και συναρτήσεις κατατεμαχισμού. Από τις διαθέσιμες συναρτήσεις κατατεμαχισμού στην crypto, κατάλληλη και εύκολη στη χρήση είναι η συνάρτηση που υλοποιεί τον αλγόριθμο PBKDF. Ωστόσο, στη συνέχεια αυτού του κεφαλαίου θα χρησιμοποιήσουμε τη συνάρτηση bcrypt, που μπορούμε να βρούμε στο πακέτο bcrypt.

Ο αλγόριθμος που υλοποιεί η bcrypt θεωρείται σήμερα ένας ασφαλής και εύχρηστος τρόπος κατατεμαχισμού. Σχεδιάστηκε από τους Niels Provos και David Mazières που βασίστηκαν στον Blowfish: b από τον Blowfish και crypt από τη συνάρτηση hashing που χρησιμοποιείται από το σύστημα κωδικών χρηστών του UNIX.

Ο κατατεμαχισμός με την bcrypt εκτελείται σε «κύκλους αλατίσματος» (salting rounds), όπου κάθε κύκλος αυξάνει τα βήματα υπολογισμού. Σήμερα θεωρείται επαρκής η τιμή saltRounds = 10 (δηλαδή 2^{10} κύκλοι αλατίσματος). Κάθε αύξηση της τιμής saltRounds διπλασιάζει περίπου τον χρόνο υπολογισμού της περίληψης. Η bcrypt είναι έτσι σχεδιασμένη ώστε να μπορεί να προσαρμοστεί σε μελλοντικές συνθήκες όπου η διαθέσιμη υπολογιστική ισχύς θα είναι πιθανότατα μεγαλύτερη.

Στο παρακάτω παράδειγμα εκτελούμε την bcrypt() με διαφορετικά κόστη, χαρακτηριστικά από κόστος 10 μέχρι 13, και τυπώνουμε τον χρόνο εκτέλεσης σε μια σχετικά σύγχρονη μηχανή (Intel Core i7-8550U):

```
import bcrypt from "bcrypt"
```

```
const myPassword = "πολύδύσκολονατομαντέψω"

for (let cost = 10; cost <= 13; cost++) {
    console.time(`cost: ${cost}, time`)
    bcrypt.hashSync(myPassword, cost)
    console.timeEnd(`cost: ${cost}, time`)
}
```

Το αποτέλεσμα που παίρνουμε είναι:

```
$2b$10$f4djbO6HfIvcyjzdyOr30OZE/7shjfEJAhIwMtteoeN5ks7DsDeQC
cost: 10, time: 63.737ms
$2b$11$t/j2IvYymLxXyPPVZzUycusp/FA3Su2zcn4pwrfRuaWlox8.vckPG
cost: 11, time: 131.044ms
$2b$12$83VbGJJCJrRptWfM9Q4s.eTjc8Fn4qZY2H2ImFeZAyXaDiEQGuqDe
cost: 12, time: 232.214ms
$2b$13$sYVHrq/fpG.mZ4GUwpsw2uThCRWbFxXwBrGPiRdHv18HEV0VDRVbO
cost: 13, time: 443.192ms
```

Παρατηρούμε πως ο χρόνος υπολογισμού περίπου διπλασιάζεται. Η περίληψη που δημιουργείται με την bcrypt [έχει τη μορφή](#)

```
$[algorithm]$[cost]$[salt][hash]
```

Για παράδειγμα:

```
$2b$10$nOUIs5kJ7naTuTFkBy1veuK0kSxUFXfuaOKdOKf9xYT0K...
 |  |  |                     |
 |  |  |               περίληψη  (24 Byte) = K0kSxUFXfuaOKdOKf9xYT0K...
 |  |  αλάτι (16 Byte) = nOUIs5kJ7naTuTFkBy1veu
 |  cost => 10 = 2^10 γύροι
 algorithm  => 2b = BCrypt
```

Με άλλα λόγια, η bcrypt αποθηκεύει το αλάτι μαζί με την περίληψη και συνεπώς δεν χρειάζεται να το αποθηκεύσουμε ξεχωριστά. Τα χαρακτηριστικά ασφαλείας της bcrypt και η ευκολία χρήσης της την καθιστούν μια δημοφιλή συνάρτηση κατατεμαχισμού και αυτή θα χρησιμοποιήσουμε στα παραδείγματά μας.

Φυσικά, εκτός από τη σύγχρονη εκδοχή με την `bcrypt.hashSync()` που είδαμε στο προηγούμενο παράδειγμα, μπορούμε να χρησιμοποιήσουμε την bcrypt με συνάρτηση επιστροφής:

```
bcrypt.hash("πολύδύσκολονατομαντέψω", saltRounds, function (err, hash) {
    console.log(`περίληψη: ${hash}`)
});
```

ή με το Promise API:

```
bcrypt.hash("πολύδύσκολονατομαντέψω", saltRounds).then(function(hash) {
    console.log(`περίληψη: ${hash}`)
});
```

ή με async/await:

```
const hash = await bcrypt.hash("πολύδύσκολονατομαντέψω", saltRounds)
console.log(`περίληψη: ${hash}`)
```

Το πακέτο bcrypt παρέχει για διευκόλυνσή μας και τη συνάρτηση compare(), με την οποία μπορούμε να συγκρίνουμε δύο περιλήψεις. Αυτή είναι βολική για χρήση στον κώδικα όταν θέλουμε να συγκρίνουμε το συνθηματικό που μας έδωσε ο χρήστης με αυτό που είναι αποθηκευμένο στη βάση μας:

```
const hash = .... // Η περίληψη που ανασύραμε από τη βάση μας

bcrypt.compare("πολύδύσκολονατομαντέψω", hash).then(function(match) {
    // αν το "πολύδύσκολονατομαντέψω" παράγει περίληψη ίδια με την "hash",
```

```
τότε
    // η match θα είναι true, αλλιώς θα είναι false.
}
```

Καθώς η compare επιστρέφει μια υπόσχεση, μπορούμε να τη χρησιμοποιήσουμε και με τη σύνταξη async/await:

```
async function authenticate(username, passwordGiven) {
    // ... βρες το username στη βάση δεδομένων και φέρε την αποθηκευμένη
περίληψη ως passwordHash ...

    const match = await bcrypt.compare(passwordGiven, passwordHash);
    if (match) {
        // επιτυχής αυθεντικοποίηση
    }
    // άλλες ενέργειες ...
}
```

14.3 Διαχείριση συνεδρίας

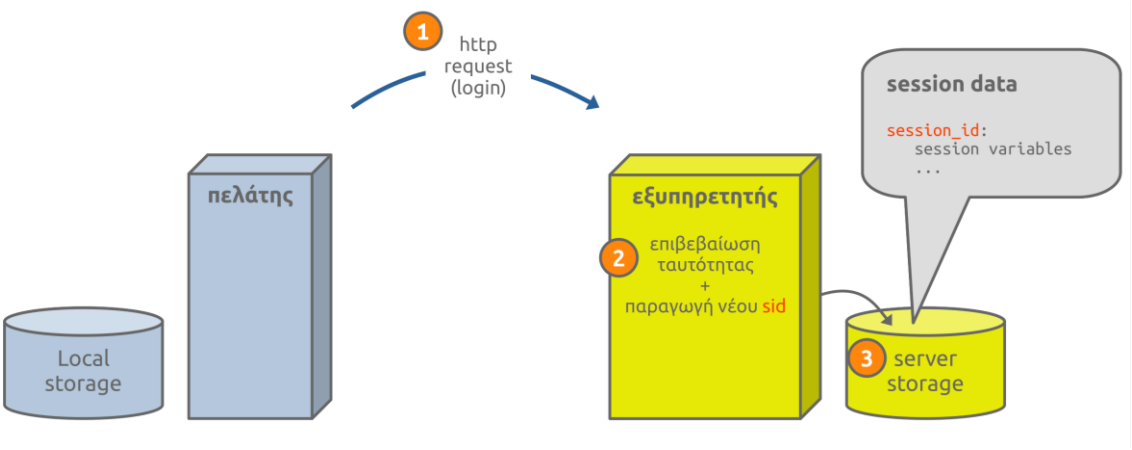
Είδαμε λοιπόν εργαλεία και τεχνικές που μπορούμε να χρησιμοποιήσουμε για να αυθεντικοποιήσουμε τους χρήστες μας. Το επόμενο πρόβλημα, που προκύπτει μάλιστα άμεσα, είναι ότι είμαστε υποχρεωμένοι να ζητάμε τα διαπιστευτήρια *κάθε φορά* που οι χρήστες ζητούν να έχουν πρόσβαση σε μια προστατευμένη περιοχή, κάτι το οποίο προφανώς δεν μπορεί να είναι αποδεκτό. Αντίθετα, και όπως γνωρίζουμε από την καθημερινή μας εμπειρία, αφού δώσουμε τα στοιχεία πρόσβασής μας σε ένα σύστημα και μας αυθεντικοποιήσει, για κάποιο διάστημα έχουμε πρόσβαση στις προστατευμένες περιοχές χωρίς να χρειάζεται να γίνει ξανά αυθεντικοποίηση. Στο διάστημα αυτό έχουμε μια ενεργή **συνεδρία** (**session**) με το σύστημα. Η κατάσταση αυτή, η συνεδρία, ισχύει μέχρις ότου να «αποσυνδεθούμε» (sign out, log out, sign off) ή να παρέλθει ένα προκαθορισμένο διάστημα απραξίας.

Το πρωτόκολλο HTTP, όπως είδαμε στην ενότητα 1.4, δεν «θυμάται», δηλαδή ο εξυπηρετητής δεν διακρίνει αν διαδοχικά αιτήματα του πελάτη ανήκουν στην ίδια συνεδρία. Κάθε αίτημα HTTP είναι ξεχωριστό. Για να ξεπεράσουμε αυτό το πρόβλημα χρειάζεται με κάποιο τρόπο ο εξυπηρετητής να μπορεί να αναγνωρίζει αν ένα αίτημα του πελάτη ανήκει σε μια ενεργή συνεδρία. Η πιο συνηθισμένη τεχνική για να το επιτύχουμε αυτό είναι με τη χρήση ενός cookie.

Το cookie (Ενότητα 1.4.8) δεν είναι τίποτε άλλο παρά ένα ψήγμα πληροφορίας που αποθηκεύεται στον φυλλομετρητή και το οποίο ο φυλλομετρητής το στέλνει στον εξυπηρετητή σε **κάθε** αίτημα που κάνει προς αυτόν. Ο εξυπηρετητής χρησιμοποιεί την πληροφορία αυτή για να αναγνωρίσει πως το αίτημα ανήκει στον συγκεκριμένο πελάτη. Μπορούμε να δούμε τον τρόπο με τον οποίο τα cookies χρησιμοποιούνται για τη διαχείριση συνεδρίας με ένα παράδειγμα.

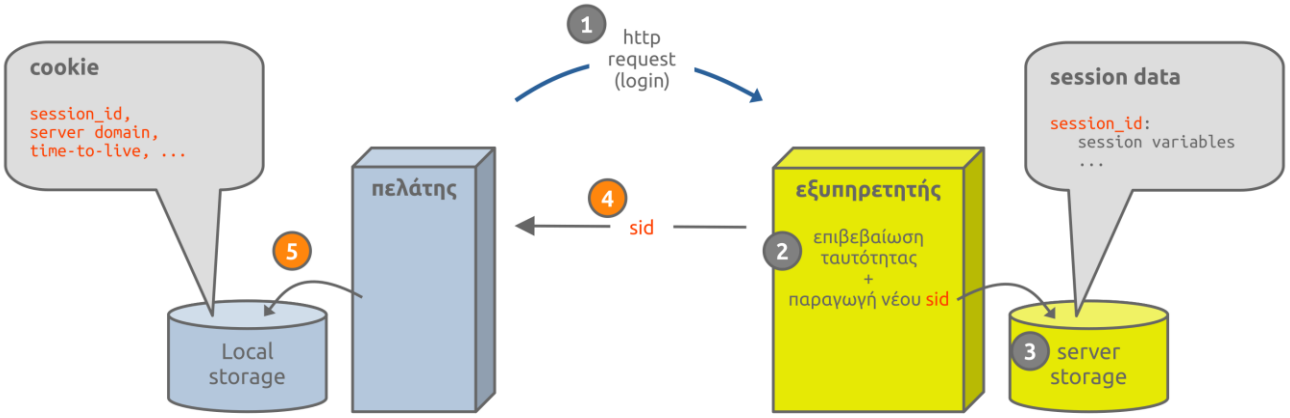
14.3.1 Μια τυπική συνεδρία

Στην **Εικόνα *14.7*** ο πελάτης έχει δώσει τα σωστά στοιχεία πρόσβασης (1) και έχει αυθεντικοποιηθεί επιτυχώς από τον εξυπηρετητή. Ο εξυπηρετητής, πριν απαντήσει στον πελάτη πως τον αυθεντικοποίησε, ξεκινά μια νέα **συνεδρία**, στα πλαίσια της οποίας θα λαμβάνονται από εδώ και πέρα τα αιτήματα που στέλνει ο πελάτης. Για να ξεχωρίσει τη συνεδρία αυτή από άλλες, παράλληλες συνεδρίες, που πιθανόν να έχουν ξεκινήσει με άλλους χρήστες, ο εξυπηρετητής παράγει ένα τυχαίο, μοναδικό αναγνωριστικό για τη συγκεκριμένη συνεδρία, το **session ID** ή **sid** (2). Αφού παραγάγει το αναγνωριστικό sid, ο εξυπηρετητής το φυλάει σε ένα αποθετήριο, το *session store* (3). Αυτό μπορεί να έχει πολλές μορφές, για παράδειγμα, κάθε sid να είναι ένα διαφορετικό αρχείο στον δίσκο του εξυπηρετητή ή να αποθηκεύεται σε μια βάση δεδομένων. Ανεξάρτητα από το ποια μορφή έχει το αποθετήριο, τα sid θα φυλάσσονται εκεί για όσο διαρκεί η συνεδρία.



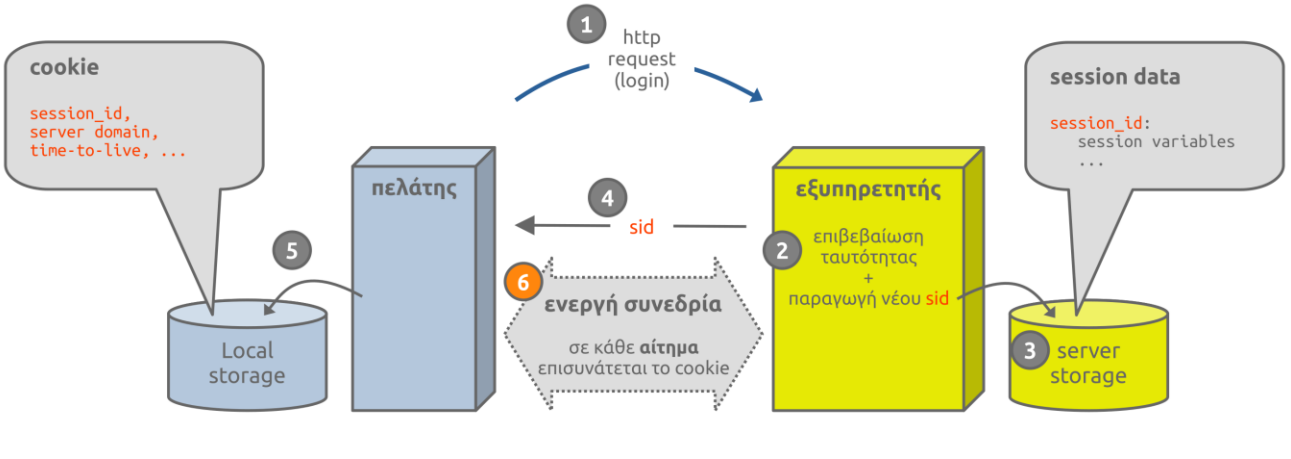
Εικόνα 14.7 *Μετά την επιτυχή αυθεντικοποίηση, παράγεται ένα μοναδικό "session ID" (sid) (2) και φυλάσσεται για περιορισμένο χρόνο στον εξυπηρετητή (3). Σε ένα sid μπορεί να αντιστοιχούν μεταβλητές συνεδρίας.*

Στη συνέχεια, ο εξυπηρετητής απαντά στον πελάτη πως τον αυθεντικοποίησε και, μαζί με την απάντηση, του στέλνει και ένα cookie (**Εικόνα *14.8***). Το cookie περιέχει το αναγνωριστικό session ID (4) που έχει αποθηκεύσει ο εξυπηρετητής. Επίσης, περιέχει μια τιμή για τον χρόνο ζωής του και την πληροφορία για το domain στο οποίο ισχύει το cookie, σε ποιες δηλαδή διευθύνσεις εξυπηρετητών θα στέλνει ο φυλλομετρητής το cookie. Η συνεδρία έχει ξεκινήσει. Ο φυλλομετρητής αποθηκεύει το cookie τοπικά και το φυλάσσει για το διάστημα του χρόνου ζωής του (5). Κάθε cookie αντιστοιχεί σε συγκεκριμένο domain (ή ομάδα domain). Για όσο έχει ισχύ το cookie, ο φυλλομετρητής το επισυνάπτει **σε κάθε αίτημα** που κάνει στον εξυπηρετητή. Με την προϋπόθεση βέβαια πως ο εξυπηρετητής βρίσκεται στα επιτρεπτά domain του cookie και πως το cookie δεν έχει λήξει.



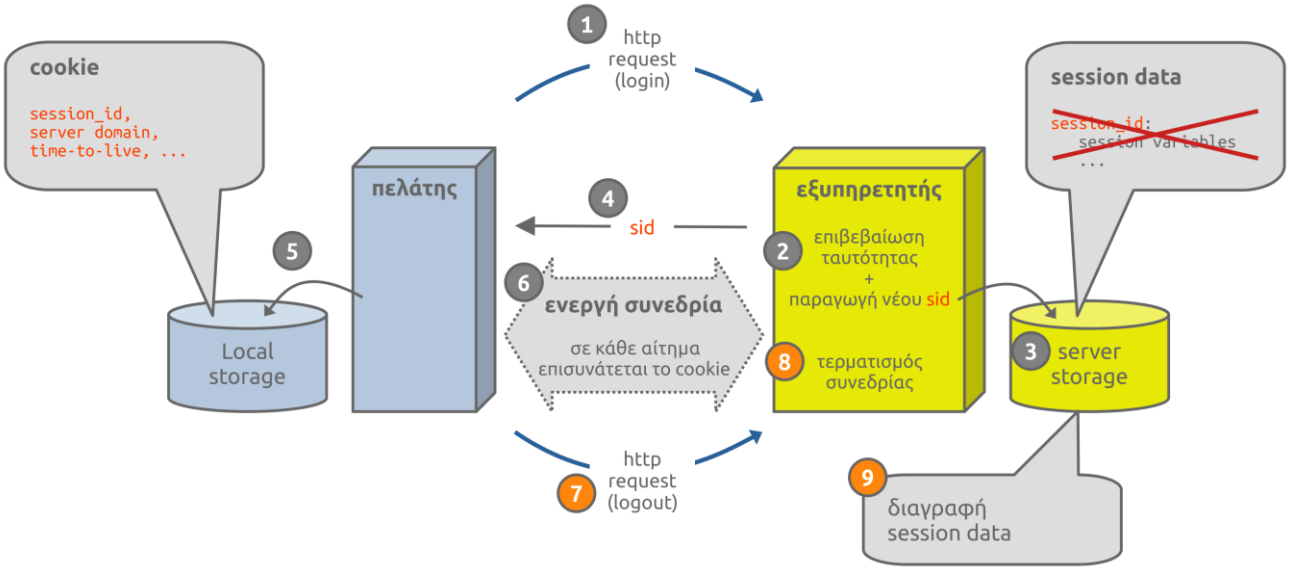
Εικόνα 14.8 *Το session ID επιστρέφεται στον πελάτη (4). Ο πελάτης το αποθηκεύει σε ένα cookie. Το cookie περιέχει, εκτός από το session ID, και πληροφορίες σχετικά με το domain στο οποίο ισχύει, τον χρόνο ισχύος του κ.ά. (5).*

Από εδώ και στο εξής, λοιπόν, όλα τα αιτήματα του πελάτη πραγματοποιούνται μέσα σε μια ενεργή συνεδρία (**Εικόνα *14.9***) (6). Σε **κάθε** νέο αίτημα προς τη διεύθυνση URL του εξυπηρετητή (στην πραγματικότητα σε κάθε αίτημα σε διεύθυνση που να ανήκει σε domain στο οποίο έχει ισχύ το cookie), ο πελάτης επισυνάπτει το cookie, δηλαδή την τιμή του session ID. Όταν το αίτημα φτάνει στον εξυπηρετητή, αυτός συγκρίνει την τιμή του session ID που του έστειλε ο πελάτης με αυτό που βρίσκεται στο αποθετήριο. Αν επιβεβαιώσει ότι το session ID είναι έγκυρο, δηλαδή υπάρχει και δεν έχει λήξει, τότε θεωρεί πως το αίτημα ανήκει στην ίδια συνεδρία.



Εικόνα 14.9 *Στο εξής, σε κάθε αίτημα του φυλλομετρητή στέλνεται και το cookie (6). Ο εξυπηρετητής θεωρεί πως τα αιτήματα που γίνονται σε κοντινά χρονικά διαστήματα (για παράδειγμα, 15 λεπτά) ανήκουν στην ίδια συνεδρία.*

Αν έχει περάσει μεγάλο διάστημα από την προηγούμενη επικοινωνία, ο εξυπηρετητής θα θεωρήσει πως η συνεδρία δεν είναι πια ενεργή (8), θα διαγράψει το session ID από το αποθετήριο (9) και θα ζητήσει από τον πελάτη να αυθεντικοποιηθεί ξανά. Το ίδιο θα συμβεί και αν ο πελάτης εσκεμμένα ζητήσει να αποσυνδεθεί (7) (**Εικόνα *14.10***).



Εικόνα 14.10 *Στην αποσύνδεση (logout) (7) ή όταν το διάστημα μεταξύ δύο διαδοχικών αιτημάτων ξεπεράσει κάποιο όριο, το session ID διαγράφεται από τον εξυπηρετητή (9) και πλέον το cookie δεν θα αναγνωρίζεται.*

Ένα σημείο που αξίζει να προσέξουμε είναι πως ο εξυπηρετητής στο αποθετήριο, μαζί με το session ID, μπορεί να αποθηκεύσει και διάφορες μεταβλητές με τιμές που είναι χρήσιμες κατά τη διάρκεια της συνεδρίας. Αυτές οι μεταβλητές ονομάζονται μεταβλητές συνεδρίας (session variables). Ανάλογα με την τεχνική λύση που θα επιλέξουμε για τη διαχείριση της συνεδρίας, οι μεταβλητές αυτές μπορεί να αποθηκεύονται στο αποθετήριο του εξυπηρετητή (3), όπως είδαμε στο παράδειγμα, ή στο cookie που στέλνεται στον πελάτη. Καθώς η δεύτερη λύση συνεπάγεται και διάφορα θέματα, όπως η περιορισμένη χωρητικότητα του cookie ή το ότι μπορεί να αποκαλυφθούν ευαίσθητες πληροφορίες, θα προτιμήσουμε την πρώτη λύση, η οποία είναι και αυτή που παρέχει το δημοφιλές πακέτο express-session. Η δεύτερη προσέγγιση ακολουθείται από το πακέτο cookie-session, με το οποίο μπορούμε να αποθηκεύσουμε τις μεταβλητές συνεδρίας στο cookie.

Μια ακόμη προσέγγιση, η οποία είναι κατάλληλη για υπηρεσίες που αποτελούνται από διαφορετικές εφαρμογές και πιθανόν βρίσκονται και σε διαφορετικούς εξυπηρετητές, είναι τα JSON Web Token (JWT). Εδώ

όλη η πληροφορία συνεδρίας αποθηκεύεται σε ένα αντικείμενο JSON, το οποίο ο πελάτης επισυνάπτει σε κάθε αίτημά του, που υλοποιείται, για παράδειγμα, στο πακέτο jsonwebtoken.

Ερώτηση

Συζητήστε τον λόγο για τον οποίο ορίζουμε τον χρόνο ζωής των cookies. Πόσος θεωρείτε ότι είναι ένα εύλογος χρόνος ζωής;

Απάντηση

Οι συνεδρίες υπάρχουν έτσι ώστε να μη χρειάζεται αυθεντικοποίηση κάθε φορά που ο χρήστης φορτώνει μια προστατευμένη σελίδα. Είναι μια σκόπιμη αποδυνάμωση της ασφάλειας του ιστοτόπου. Τη δεχόμαστε γιατί το αντάλλαγμα είναι η άνεση του χρήστη. Αν δεν είχαμε συνεδρίες, ο χρήστης θα έπρεπε να αυθεντικοποιείται ξανά κάθε φορά που προσπελάζει μια προστατευμένη περιοχή. Πρακτικά, αυτό θα σήμαινε με κάθε φόρτωμα σελίδας. Σκεφτείτε το παράδειγμα μιας εφαρμογής e-banking. Συνηθίζεται σε αυτές τις εφαρμογές η συνεδρία να διαρκεί 10 λεπτά. Σε άλλες εφαρμογές, π.χ. webmail, αυτό το διάστημα είναι αρκετά μεγαλύτερο.

Ιδανικά, θα θέλαμε η συνεδρία να είναι όσο το δυνατόν πιο σύντομη και να διαρκεί ακριβώς τόσο όσο χρειάζεται για να ολοκληρωθεί η εργασία του χρήστη. Πρακτικά όμως, η διάρκεια επιλέγεται συνήθως στη χρυσή τομή μεταξύ ασφάλειας και ενόχλησης του χρήστη, που κυμαίνεται ανάλογα με το είδος της εφαρμογής. Οι τράπεζες μπορούν να ζητήσουν από τους χρήστες τους να αυθεντικοποιούνται συχνά, χωρίς αυτό να προκαλεί δυσφορία στους χρήστες.

14.3.2 Διαχείριση συνεδρίας με το express-session

Το express-session χρησιμοποιείται όπως οποιοδήποτε άλλο middleware του Express.js. Με την `app.use()` το τοποθετούμε στην αλυσίδα επεξεργασίας του Express.js. Επειδή θα θέλουμε το express-session να ισχύει για όλα τα αιτήματα προς την εφαρμογή μας, θα πρέπει να είναι από τα πρώτα τμήματα της αλυσίδας επεξεργασίας του Express.js. Συνήθως η δήλωσή του αμέσως μετά το middleware “static” είναι μια καλή πρακτική. Πριν όμως χρειάζεται να το αρχικοποιήσουμε με κάποιες βασικές ρυθμίσεις. Στο παρακάτω παράδειγμα βλέπουμε πώς αυτό γίνεται παρέχοντας ένα αντικείμενο κατά την αρχικοποίηση.

Στο αντικείμενο αυτό είναι απαραίτητο να οριστεί η τιμή της ιδιότητας secret, η οποία πρέπει να είναι ένα μεγάλο τυχαίο αλφαριθμητικό. Η secret χρησιμοποιείται για να αναγνωρίσει ο εξυπηρετητής πως το session ID που δέχεται από τον πελάτη έχει όντως παραχθεί από αυτό τον εξυπηρετητή (και δεν είναι μια κακόβουλη απόπειρα να αποκτηθεί πρόσβαση σε μια συνεδρία που ήδη έχει ξεκινήσει), συνεπώς είναι σημαντικό η τιμή του να είναι καλά φυλαγμένη. Για την παραγωγή της τιμής του secret μπορούμε να χρησιμοποιήσουμε, για παράδειγμα, τη βιβλιοθήκη crypto: crypto.randomBytes(32).toString('hex') και να τη φυλάξουμε σε ένα αρχείο dotenv (Ενότητα 11.8.4).

Η ιδιότητα **cookie.maxAge** ορίζει για πόσα msec θα ισχύει το cookie (1200*1000 = 20 λεπτά). Ο χρόνος αυτός μετράει αντίστροφα από τη στιγμή της πιο πρόσφατης απάντησης του εξυπηρετητή. Με κάθε νέα απάντηση στο maxAge η αντίστροφη μέτρηση ξεκινά από την αρχή.

Οι άλλες δύο ιδιότητες του παραδείγματός μας, resave και saveUninitialized, έχουν προκαθορισμένη ιδιότητα true, ωστόσο συστήνεται να γίνουν false. Αυτό εξαρτάται βέβαια από τα χαρακτηριστικά της εφαρμογής και το είδος του αποθετηρίου (session storage) που θα διαλέξουμε. Με saveUninitialized: false ορίζουμε πως δεν θέλουμε να αποθηκευτεί μια συνεδρία στο αποθετήριο συνεδριών, αν δεν έχει αλλάξει. Προκειμένου να αποθηκευτεί η συνεδρία στο αποθετήριο, δεν αρκεί, δηλαδή, απλά να ξεκινήσει η συνεδρία, αλλά θα πρέπει να έχει αλλάξει η κατάστασή της (π.χ. να τεθεί μια μεταβλητή συνεδρίας).

```
import expSession from 'express-session'

const sessionConf = {
    // χρησιμοποιούμε το dotenv για να διαβάσουμε από το περιβάλλον την
τιμή της
    // ή παρέχουμε ένα μεγάλο τυχαίο αλφαριθμητικό
    secret: process.env.secret || "έναμεγάλοτυχαίοαλφαριθμητικό",
    cookie: {maxAge: 1200*1000}
    resave: false,
    saveUninitialized: false
```

```
};

app.use(expSession(sessionConf))
```

14.3.3 Αποθετήριο συνεδρίας

Όπως είδαμε, οι πληροφορίες για την κάθε συνεδρία, όπως το session ID που στέλνουμε στον πελάτη και που το φυλάσσει σαν μορφή cookie, καθώς και οι μεταβλητές συνεδρίας, αποθηκεύονται στον εξυπηρετητή από το express-session. Το προεπιλεγμένο αποθετήριο, για το οποίο δεν χρειάζεται να κάνουμε κάτι, είναι στη μνήμη (RAM) του εξυπηρετητή. Ωστόσο, αυτό το αποθετήριο έχει **εσκεμμένα** προβληματική υλοποίηση. Είναι ίσως αρκετό για σύντομη δοκιμαστική χρήση της εφαρμογής, αλλά σε πραγματικές συνθήκες θα εμφανιστούν προβλήματα. Το πρόβλημα είναι πως δεν διαγράφονται από το αποθετήριο οι συνεδρίες που έχουν λήξει και σταδιακά θα εξαντληθεί η διαθέσιμη μνήμη RAM.

Συνεπώς, πρέπει να καταφύγουμε σε άλλο αποθετήριο για τις συνεδρίες. Το express-session συνεργάζεται με μεγάλο αριθμό αποθετηρίων και η επιλογή εξαρτάται από τα ιδιαίτερα χαρακτηριστικά της εφαρμογής μας. Σε ένα πραγματικό σύστημα που δέχεται μεγάλη κίνησης πιθανώς να θέλουμε οι συνεδρίες να αποθηκεύονται σε μνήμη που να εκτείνεται ανάλογα με τις τρέχουσες ανάγκες, να είναι γρήγορη και να χρησιμοποιεί δευτερεύουσες μονάδες αποθήκευσης.

Μια εύκολη εναλλακτική, που μπορεί να είναι επαρκής για μικρά πρότζεκτ, είναι το πακέτο memorystore που προσφέρει μια πιο πλήρη υλοποίηση της αποθήκευσης στη μνήμη RAM του εξυπηρετητή.

```
import expSession from 'express-session'
import createMemoryStore from 'memorystore';

// Δημιουργία ενός memory store constructor
const MemoryStore = createMemoryStore(expSession);

const sessionConf = {
    secret: process.env.secret || "έναμεγάλοτυχαίοαλφαριθμητικό",
    cookie: {maxAge: 1200*1000},
    // Το αποθετήριο θα είναι το MemoryStore. Κάθε 86400 sec (24 ώρες)
    // θα διαγράφονται αυτόματα οι συνεδρίες που έχουν λήξει
    store: new MemoryStore({ checkPeriod: 86400 * 1000 }),
    resave: false,
    saveUninitialized: false
};

app.use(expSession(sessionConf))
```

14.3.4 Μεταβλητές συνεδρίας

Στην αρχή της αλυσίδας επεξεργασίας της εφαρμογής μας βρίσκεται το express-session, το τμήμα που αναλαμβάνει να διαβάσει το session ID από το cookie που έχει στείλει ο πελάτης μαζί με το αίτημα και να ελέγξει αν είναι έγκυρο. Πριν χρησιμοποιήσουμε τον μηχανισμό για να προστατέψουμε την πρόσβαση σε κάποιες περιοχές της εφαρμογής μας, ας δούμε ένα πιο απλό παράδειγμα συνεδρίας. Στο παράδειγμα αυτό θα χρησιμοποιήσουμε τις μεταβλητές της συνεδρίας μας για να διατηρήσουμε πληροφορία μεταξύ διαφορετικών αιτημάτων πελάτη-εξυπηρετητή.

Στο πολύ απλό παράδειγμά μας έχουμε δύο διαδρομές, την ping/ και την pong/. Σε κάθε αίτημα προς αυτές τις δύο διαδρομές πρώτα τυπώνεται η τιμή της μεταβλητής req.session.mostRecentHit και της μεταβλητής req.session.hits. Έπειτα η τιμή της mostRecentHit ορίζεται σε ping ή pong, ανάλογα και η τιμή της hits αυξάνει κατά ένα.

```
import express from 'express'
import expSession from 'express-session'

const app = new express()
```

```
const sessionConf = {
    secret: process.env.secret || "έναμεγάλοτυχαίοαλφαριθμητικό",
    cookie: { maxAge: 10 * 1000 }, // μόλις 10 δευτερόλεπτα!
    resave: false,
    saveUninitialized: false
};

app.use(expSession(sessionConf))

app.get("/ping", (req, res) => {
    console.log("Ping: προηγούμενο αίτημα ήταν στο",
req.session.mostRecentHit, req.session.hits)
    req.session.mostRecentHit = "ping"
    req.session.hits++
    res.send("<a href='/pong/'>pong<a>")
})

app.get("/pong", (req, res) => {
    console.log("Pong: προηγούμενο αίτημα ήταν στο",
req.session.mostRecentHit, req.session.hits)
    req.session.mostRecentHit = "pong"
    req.session.hits++
    res.send("<a href='/ping/'>ping<a><br><a href='/pong?close=true'>close
session</a>")
})

app.listen(3000, () => console.log('Η εφαρμογή τρέχει'))
```

Την πρώτη φορά που τρέχει η εφαρμογή οι μεταβλητές δεν έχουν τιμή, οπότε στην κονσόλα τυπώνεται "undefined". Στη συνέχεια, καθώς επιλέγουμε μια από τις δύο διαθέσιμες διαδρομές στην κονσόλα εμφανίζεται κάτι σαν την παρακάτω έξοδο:

```
Η εφαρμογή τρέχει
Ping: προηγούμενο αίτημα ήταν στο undefined undefined
Pong: προηγούμενο αίτημα ήταν στο ping null
Ping: προηγούμενο αίτημα ήταν στο pong 1
Pong: προηγούμενο αίτημα ήταν στο ping 2
Ping: προηγούμενο αίτημα ήταν στο pong 3
...
```

Βλέπουμε πως μπορούμε να αποθηκεύσουμε στο αντικείμενο session οποιεσδήποτε μεταβλητές θέλουμε μέσω της ιδιότητας **req.session{js}**. Οι μεταβλητές αυτές αποθηκεύονται στο αποθετήριο της συνεδρίας (session storage) στην πλευρά του εξυπηρετητή. Με κάθε αίτημα που δέχεται ο εξυπηρετητής, η Express.js ανατρέχει στο αποθετήριο, ανασύρει τις μεταβλητές και μας τις κάνει διαθέσιμες στο αίτημα req.

Το αντικείμενο της συνεδρίας req.session έχει και μερικές χρήσιμες μεθόδους, όπως η destroy(callback), που διαγράφει τη συνεδρία από το αποθετήριο ή η regenerate(callback) που επανεκκινεί τη συνεδρία. Οι δύο συναρτήσεις καλούν τη συνάρτηση επιστροφής callback όταν ολοκληρωθούν. Το αναγνωριστικό της συνεδρίας είναι διαθέσιμο στη μεταβλητή req.sessionID.

Για παράδειγμα, μπορούμε να ορίσουμε ένα ακόμη τμήμα της αλυσίδας επεξεργασίας πριν τους χειριστές των διαδρομών μας:

```
...

app.use(expSession(sessionConf))

app.use((req, res, next) => {
    console.log("session ID:", req.sessionID)
    if (req.query['close']) {
        console.log("επανεκκίνηση συνεδρίας")
```

```
            req.session.regenerate((callback) => next())
        }
        else
            next()
})

app.get("/ping", (req, res) => {
...
    res.send("<a href='/pong/'>pong<a><br><a href='/pong?close=true'>close
session</a>")
})
...
```

Παρατηρούμε τώρα πως όταν πατήσουμε τον σύνδεσμο “close” ή όταν περάσουν πάνω από 10 δευτερόλεπτα αποκτούμε νέο session ID.

```
Η εφαρμογή τρέχει
session ID: jKTV_PVtoljwJjxhGLNRyoI8ZiB-l-iG
Ping: προηγούμενο αίτημα ήταν στο undefined undefined
session ID: jKTV_PVtoljwJjxhGLNRyoI8ZiB-l-iG
Pong: προηγούμενο αίτημα ήταν στο ping null
session ID: jKTV_PVtoljwJjxhGLNRyoI8ZiB-l-iG
Ping: προηγούμενο αίτημα ήταν στο pong 1
session ID: jKTV_PVtoljwJjxhGLNRyoI8ZiB-l-iG
επανεκκίνηση συνεδρίας
Pong: προηγούμενο αίτημα ήταν στο undefined undefined
session ID: O9CB92kCjtIRn_ugP_UrPNwxQv5po0By
Ping: προηγούμενο αίτημα ήταν στο pong null
...
```

14.4 Προστασία περιοχής με το Express.js

Με τον μηχανισμό των συνεδριών και την αυθεντικοποίηση του χρήστη έχουμε στη διάθεσή μας τα βασικά στοιχεία που είναι απαραίτητα για να προστατέψουμε περιοχές της εφαρμογής μας. Η ιδέα είναι εξαιρετικά απλή. Σε κάθε αίτημα στον εξυπηρετητή που αφορά προστατευμένη περιοχή, πριν ικανοποιηθεί το αίτημα, γίνεται έλεγχος αν ο χρήστης είναι αυθεντικοποιημένος (Εικόνα 126). Για να κάνουμε τον έλεγχο θα χρησιμοποιήσουμε και πάλι τον μηχανισμό της αλυσίδας επεξεργασίας του Express.js. Το πρώτο υποτμήμα της αλυσίδας επεξεργασίας ελέγχει αν ο χρήστης έχει δικαίωμα πρόσβασης, π.χ. αν είναι αυθεντικοποιημένος.

```
app.get('/sensitive-area',
    (req, res, next) => {
        if (req.session.authenticatedUsername)
            next()
        else
            res.redirect('/login-form')
    },
    (req, res) => {
        console.log('Επιτυχής πρόσβαση στην προστατευμένη περιοχή')
    });
```

Στο παραπάνω απόσπασμα ελέγχουμε αν έχει τεθεί η μεταβλητή req.session.authenticatedUsername και αν ναι, τότε συνεχίζει η εκτέλεση της αλυσίδας επεξεργασίας με τη next(). Αν όχι, η αλυσίδα επεξεργασίας διακόπτεται και γίνεται ανακατεύθυνση στη διαδρομή /login-form, όπου ο χρήστης θα μπορεί να εισαγάγει τα διαπιστευτήριά του.

Η φόρμα στη διαδρομή /login-form, όταν πατηθεί το κουμπί «Σύνδεση», θα πραγματοποιήσει ένα αίτημα GET στη διαδρομή /do-login και θα δώσει στις μεταβλητές username και password τις τιμές που θα έχει εισαγάγει ο χρήστης στα αντίστοιχα πεδία.

```
<form action="/do-login" method="POST">
    <fieldset> Δώστε τα στοιχεία πρόσβασης<br>
        <input name="username" type="text" placeholder="όνομα χρήστη"><br>
        <input name="password" type="password"
placeholder="συνθηματικό"><br>
        <button type="submit">Σύνδεση</button>
    </fieldset>
</form>
```

Ας το δούμε συνολικά σε μια μίνι εφαρμογή:

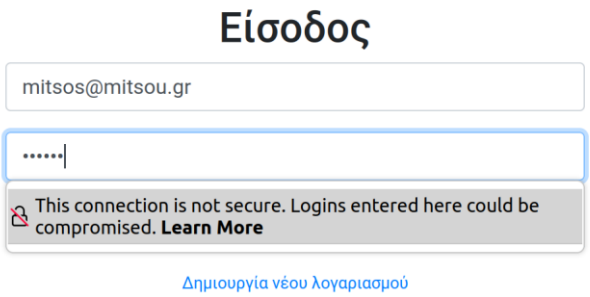
```
app.get('/login-form',
    (req, res) => {
        // Επιστρέφουμε τη φόρμα
        res.send('<form action="/do-login" method="POST">....')
    };)

app.post('/do-login', (req, res) => {
        const usernameGiven = req.params.username;
        const passwordGiven = req.params.password;
        // ανασύρουμε από τη βάση δεδομένων την περίληψη του συνθηματικού
που
        // έχουμε αποθηκεύσει για τον χρήστη givenUsername ως
"passwordHash"
        ...
        bcrypt.compare(passwordGiven, passwordHash, (result) => {
            if (result) {
                // επιτυχία, αποθηκεύομε το username στις μεταβλητές
συνεδρίας
                // ώστε να ξέρουμε πως έγινε η σύνδεση
                req.session.authenticatedUsername = usernameGiven
                redirect('/sensitive-area')
            }
            else {
                // αποτυχία, ανακατεύθυνση στη φόρμα εισόδου για νέα
δοκιμή
                redirect('/login-form')
            }
        });
    })

app.get('/sensitive-area',
    checkAuthenticated,
    (req, res) => {
        console.log('Επιτυχής πρόσβαση στην προστατευμένη περιοχή')
    });

const checkAuthenticated = (req, res, next) => {
        if (req.session.authenticatedUsername)
            next()
        else
            res.redirect('/login-form')
    }
```

Στο πιο ολοκληρωμένο αυτό παράδειγμα, υποθέτοντας πως θα έχουμε περισσότερες από μία προστατευμένες περιοχές, μεταφέρουμε τον έλεγχο σε μια νέα συνάρτηση την οποία μπορούμε να επαναχρησιμοποιήσουμε σε αλυσίδες επεξεργασίας, την checkAuthenticated. Αν δεν έχει τιμή η μεταβλητή συνεδρίας authenticatedUsername, τότε ανακατευθύνουμε στη φόρμα εισόδου που βρίσκεται στη διαδρομή /login-form. Η φόρμα υποβάλλει στη διαδρομή /do-login τα στοιχεία πρόσβασης που δίνει ο χρήστης. Εκεί ανασύρεται από

τη βάση δεδομένων η αποθηκευμένη περίληψη του συνθηματικού και συγκρίνεται με την bcrypt.compare() με αυτό που έδωσε ο χρήστης. Αν είναι ίδιες, τότε έχουμε επιτυχή αυθεντικοποίηση και θέτουμε τη μεταβλητή req.session.authenicatedUsername σαν ένδειξη ότι για τη διάρκεια της συνεδρίας ο χρήστης είναι αυθεντικοποιημένος. Αυτή είναι και η μεταβλητή που ελέγχουμε στη συνάρτηση checkAuthenticated().

14.5 Ασφαλής μεταφορά

Η ασφαλής αποθήκευση των διαπιστευτηρίων δεν έχει μεγάλη αξία αν η μεταφορά τους γίνεται μέσω HTTP. Με το HTTP η επικοινωνία μεταξύ πελάτη και εξυπηρετητή είναι μη κρυπτογραφημένη. Έτσι, θα μπορούσε κάποιος κακόβουλος χρήστης να παρακολουθήσει και να υποκλέψει όλη την επικοινωνία του πελάτη με τον φυλλομετρητή, η οποία περιέχει, μεταξύ άλλων, και το συνθηματικό που εισάγει ο χρήστης στη φόρμα αυθεντικοποίησης. Οι φυλλομετρητές προειδοποιούν τον χρήστη, με μια ένδειξη στην μπάρα διεύθυνσης, αλλά και πιο εμφατικά, όπως ο Firefox, που εμφανίζει την προειδοποίηση ακριβώς πάνω στο στοιχείο που θα μεταφερθεί μη κρυπτογραφημένο (**Εικόνα *14.11***).

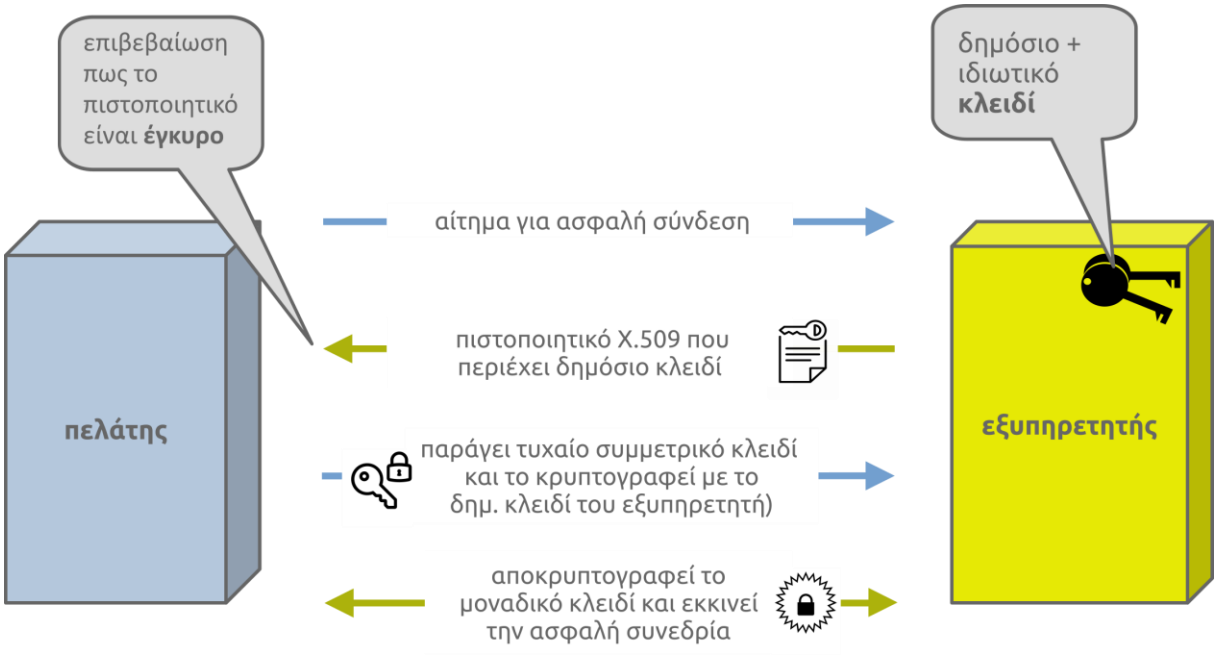


Εικόνα 14.11 *Ο φυλλομετρητής Firefox προειδοποιεί έντονα όταν πρόκειται να υποβάλουμε ευαίσθητα δεδομένα με μια φόρμα με το πρωτόκολλο HTTP.*

Το πρόβλημα αυτό λύνεται σήμερα σχετικά γρήγορα με τη χρήση του HTTPS, την κρυπτογραφημένη εκδοχή του HTTP. Η χρήση του ασφαλούς, κρυπτογραφημένου πρωτοκόλλου επικοινωνίας HTTPS είναι τα τελευταία χρόνια μια εύκολη υπόθεση. Στο HTTPS η κρυπτογράφηση γίνεται με το κρυπτογραφικό πρωτόκολλο TLS 1.3, που αντικατέστησε το παλιότερα χρησιμοποιούμενο SSL. Μάλιστα, ο εξυπηρετητής μπορεί με την κεφαλίδα απόκρισης HTTP Strict-Transport-Security να απαιτήσει από τους φυλλομετρητές να πραγματοποιούν μαζί του συνδέσεις μόνο με HTTPS.

Για να χρησιμοποιήσουμε το HTTPS είναι απαραίτητη η προμήθεια ενός πιστοποιητικού κρυπτογράφησης, το οποίο θα χρησιμοποιηθεί από τον εξυπηρετητή. Το πιστοποιητικό κρυπτογράφησης συνοδεύεται από δύο κρυπτογραφικά κλειδιά, ένα ιδιωτικό και ένα δημόσιο. Με το δημόσιο κλειδί του ζευγαριού κλειδιών μπορούμε να κρυπτογραφήσουμε μηνύματα. Τα μηνύματα αυτά μπορούμε να τα αποκρυπτογραφήσουμε μόνο με το ιδιωτικό κλειδί του ζευγαριού. Με αυτό τον τρόπο, ο εξυπηρετητής κρατάει κρυφό το ιδιωτικό κλειδί και κάνει διαθέσιμο το δημόσιο (**Εικόνα *14.12***). Οποιοσδήποτε πελάτης επιθυμεί να επικοινωνήσει με τον εξυπηρετητή λαμβάνει πρώτα το πιστοποιητικό και το δημόσιο κλειδί. Ο πελάτης εξετάζει το πιστοποιητικό αν είναι έγκυρο, δηλαδή πως δεν έχει λήξει και πως έχει εκδοθεί από έμπιστη αρχή πιστοποίησης. Σημειώστε πως οι φυλλομετρητές έχουν ενσωματωμένες λίστες με αρχές πιστοποίησης που θεωρούνται έμπιστες.

Στη συνέχεια, εφόσον θεωρηθεί έγκυρο το πιστοποιητικό, ο πελάτης χρησιμοποιεί το δημόσιο κλειδί για να στείλει τα μηνύματά του και να ξεκινήσει την κρυπτογραφημένη επικοινωνία. Η επικοινωνία αυτή στην πραγματικότητα γίνεται με ένα προσωρινό, τρίτο κλειδί, το οποίο είναι έγκυρο μόνο για τη συνεδρία αυτή. Σημειώστε πως η έννοια «συνεδρία» εδώ δεν έχει σχέση με τη «συνεδρία» που είδαμε στην προηγούμενη ενότητα, αλλά αφορά μόνο το πρωτόκολλο HTTPS, την κρυπτογραφημένη επικοινωνία με το προσωρινό κλειδί.



Εικόνα 14.12 *Η χρήση HTTPS απαιτεί ο εξυπηρετητής να διαθέτει ψηφιακό πιστοποιητικό και ζεύγος κλειδιών με το οποίο να κρυπτογραφείται η επικοινωνία.*

14.5.1 Επίπεδα επικύρωσης πιστοποιητικών

Συνολικά, η ασφαλής μεταφορά με HTTPS/TLS εξασφαλίζει:

- **Αυθεντικοποίηση**. Κάθε μέρος μπορεί να επιβεβαιώσει την ταυτότητα του άλλου, μια εξασφάλιση δηλαδή πως επικοινωνούμε με αυτό που θέλουμε να επικοινωνήσουμε και όχι με κάποιον που τον υποδύεται.
- **Κρυπτογράφηση**. Η επικοινωνία είναι προστατευμένη ώστε κακόβουλοι τρίτοι να μην μπορούν να υποκλέψουν το περιεχόμενό της.
- **Ακεραιότητα**. Τα δεδομένα που μεταδίδονται δεν αλλοιώνονται από κακόβουλους τρίτους κατά τη μετάδοση.

Η κρυπτογράφηση και η ακεραιότητα αφορούν κυρίως χαρακτηριστικά του πρωτοκόλλου HTTPS/TLS και των αλγορίθμων που χρησιμοποιούνται και μπορούμε να θεωρήσουμε πως με τη χρήση του HTTPS/TLS τα εξασφαλίζουμε. Όσον όμως αφορά το πρώτο σημείο, την επιβεβαίωση της ταυτότητας του άλλου, υπάρχουν τρία διαφορετικά επίπεδα επιβεβαίωσης. Αυτά αντιστοιχούν στο πόσο εξονυχιστικός ήταν ο έλεγχος της ταυτότητας του κατόχου του πιστοποιητικού όταν αυτό εκδόθηκε.

Τα τρία επίπεδα, από τον λιγότερο στον περισσότερο εξονυχιστικό έλεγχο, είναι:

- **Επικύρωση του domain (Domain Validation)** (DV). Επικύρωση πως ο κάτοχος του πιστοποιητικού είναι όντως αυτός που διαχειρίζεται το domain. Για παράδειγμα, επικύρωση πως το domain www.microsoft.com όντως ανήκει στον κάτοχο του domain και όχι σε κάποιον τρίτο που υποδύεται τον κάτοχο του www.microsoft.com.
- **Επικύρωση του οργανισμού (Oganization Validation)** (OV). Επικύρωση πως ο κάτοχος του πιστοποιητικού είναι αυτός που διαχειρίζεται το domain και επιπλέον πως υπάρχει η νομική οντότητα του κατόχου, δηλαδή ότι υπάρχει οντότητα Microsoft που διαχειρίζεται το domain www.microsoft.com.
- **Εκτεταμένη επικύρωση (Extended Validation)** (EV). Επικυρώνει τη νομική οντότητα του κατόχου του πιστοποιητικού με ανθρώπινο έλεγχο, δηλαδή, κάποιος υπάλληλος πιστοποιεί την ύπαρξη του κατόχου.

Τα πιστοποιητικά επιπέδου OV και ακόμη περισσότερο EV παρέχουν αυξημένη βεβαιότητα για την ταυτότητα του ιστοτόπου. Οι φυλλομετρητές εμπιστεύονται αυτά τα πιστοποιητικά, που εκδίδονται από αναγνωρισμένες αρχές πιστοποίησης, εταιρείες δηλαδή που εξειδικεύονται στην έκδοση και την πώληση τέτοιων πιστοποιητικών. Οι εταιρείες αυτές εκδίδουν επίσης και πιστοποιητικά επιπέδου DV, και αυτά με κάποιο μη αμελητέο κόστος.

Στο άλλο άκρο, για να αποφύγουμε το κόστος, μπορούμε να εφοδιάσουμε τον εξυπηρετητή με πιστοποιητικά που έχουμε εκδώσει οι ίδιοι, εντελώς δωρεάν. Με αυτό τον τρόπο εξασφαλίζουμε μεν την ακεραιότητα και την κρυπτογράφηση, ωστόσο, οι φυλλομετρητές, επειδή δεν περιλαμβανόμαστε στις αναγνωρισμένες αρχές πιστοποίησης, δεν τα εμπιστεύονται.

Η συνιστώμενη, δωρεάν λύση είναι πιστοποιητικά που εκδίδονται από τον μη κερδοσκοπικό οργανισμό Let's Encrypt. Τα πιστοποιητικά αυτά οι φυλλομετρητές τα θεωρούν έμπιστα και, παρ' όλο που είναι επιπέδου DV, αρκούν για να έχουμε κρυπτογραφημένη επικοινωνία μεταξύ πελάτη και εξυπηρετητή και εξασφάλιση ότι κανείς τρίτος δεν παρεμβαίνει για να αλλοιώσει την επικοινωνία. Έχουν μικρή διάρκεια, τριών μηνών, ωστόσο η διαδικασία ανανέωσης μπορεί να γίνει αυτόματα από εργαλεία που παρέχει ο οργανισμός Let's Encrypt.

14.5.2 Ασφαλής μεταφορά με Let's Encrypt στο Heroku

Η χρήση της υπηρεσίας Let's Encrypt είναι σχετικά εύκολη υπόθεση. Στο Heroku, την πλατφόρμα που χρησιμοποιούμε σε αυτό το βιβλίο για να φιλοξενήσουμε μερικά από τα παραδείγματά μας στο διαδίκτυο, παρέχει κάποια επιπλέον υποστήριξη. Αν χρησιμοποιούμε τις δωρεάν υπηρεσίες του Heroku και η εφαρμογή μας φιλοξενείται σε domain που heroku.app, τότε παρέχεται αυτόματα πιστοποιητικό Let's Encrypt.

14.6 Ερωτήσεις

1. Ο συνιστώμενος χώρος αποθήκευσης των διαπιστευτηρίων των χρηστών μας είναι η βάση δεδομένων του εξυπηρετητή μας. Εκεί δεν διατρέχουμε τον κίνδυνο να διαρρεύσουν τα διαπιστευτήρια ή να παραβιαστεί η βάση δεδομένων.
   - Σωστό/Λάθος
2. Η αυθεντικοποίηση είναι η διαδικασία με την οποία το σύστημα αναγνωρίζει τον χρήστη για να του δώσει πρόσβαση σε κάποιες από τις προστατευμένες περιοχές του.
   - Σωστό/Λάθος
3. Όταν, για τον σκοπό της φύλαξης ενός συνθηματικού, ένα αρχικό μήνυμα περνάει μέσα από τη διαδικασία του μονόδρομου κατατεμαχισμού, η περίληψη που παράγεται έχει τα εξής χαρακτηριστικά: (επιλέξτε όσα ισχύουν)
   1. Το μέγεθος της περίληψης εξαρτάται από το μέγεθος του αρχικού μηνύματος.
   2. Αν γνωρίζουμε την περίληψη, μπορούμε να ανασυνθέσουμε το αρχικό μήνυμα.
   3. Αν γνωρίζουμε την περίληψη και τον αλγόριθμο κατατεμαχισμού, μπορούμε να ανασυνθέσουμε το αρχικό μήνυμα.
   4. Ακόμη και αν γνωρίζουμε την περίληψη και τον αλγόριθμό κατατεμαχισμού, είναι αδύνατο να ανασυνθέσουμε το αρχικό μήνυμα.
4. Το αλάτι (salt) στη διαδικασία παραγωγής περιλήψεων είναι ένα αλφαριθμητικό
   1. που προστίθεται στην τελική περίληψη και είναι άγνωστο στον χρήστη.
   2. που προστίθεται στο αρχικό μήνυμα πριν την παραγωγή της περίληψης και είναι άγνωστο στον χρήστη.
   3. που προστίθεται στο αρχικό μήνυμα πριν την παραγωγή της περίληψης και είναι γνωστό στον χρήστη.
5. Το σύστημα χρησιμοποιεί το ίδιο αλάτι για όλες τις περιλήψεις που δημιουργεί κατά τη διάρκεια λειτουργίας του.
   - Σωστό/Λάθος

14.7 Βιβλιογραφία και Αναφορές

Στο περιορισμένο χώρο αυτού του κεφαλαίου καλύπτονται τα απολύτως απαραίτητα θέματα που αφορούν την ασφάλεια και την αυθεντικοποίηση των χρηστών. Οργανισμοί όπως ο Open Web Application Security Project (OWASP) παρέχουν εκτεταμένες οδηγίες και καλές πρακτικές για την ασφάλιση των διαδικτυακών μας εφαρμογών. Στη σειρά των "cheatsheets", δηλαδή σύντομων και περιεκτικών οδηγιών του, ο οργανισμός παρέχει εισαγωγική τεκμηρίωση για πλήθος θεμάτων, όπως, για παράδειγμα, για τον κατατεμαχισμό ή τη διαχείριση συνεδρίας.

Καθώς και εδώ, όπως και στα προηγούμενα κεφάλαια που παρουσιάστηκαν συγκεκριμένες τεχνολογίες, ο πρώτος σταθμός περαιτέρω εμβάθυνσης είναι η τεκμηρίωση των λογισμικών που χρησιμοποιούμε, για παράδειγμα, η τεκμηρίωση του πακέτου bcrypt και του πακέτου express-session.

Τέλος, παρόμοια θέματα, καλύπτονται και στους προχωρημένους οδηγούς του Express.js και της σελίδας mdn.

Ξενόγλωσσες

Herron, D. (2020). *Node.js Web Development* (5th ed.). Packt.

Hoffman, A. (2020). *Web Application Security*. O'Reilly Media, Inc.

Συλλογικό (2022). *API Reference Documentation*. Node.js. Ανακτήθηκε στις 14 Σεπτεμβρίου 2022 από https://nodejs.org/en/docs/

Παράρτημα Απαντήσεις στις ερωτήσεις

- Κεφάλαιο 1
  1. Λάθος
  2. 3
  3. Λάθος
  4. 1
  5. 1, 2, 3, 4 και 5
  6. 1
  7. 2, 3 και 4
  8. 1
  9. 4
  10. 2
  11. Σωστό
  12. 3
  13. 2
  14. www.host.com:5050
  15. https://
- Κεφάλαιο 2
  1. Λάθος
  2. 2,3,5
  3. 3
  4. Λάθος
  5. 4
  6. Λάθος
  7. 2
  8. Όχι
  9. Λάθος
  10. 2
  11. 2
  12. 2
  13. Όχι
  14. Σωστό
  15. Σωστό
  16. 1,2,3,4
  17. 2
  18. 1
  19. 2
  20. 1,2
  21. 3
  22. Σωστό
  23. 3
  24. Λάθος
  25. 2,4,5
  26. Λάθος
  27. Λάθος
  28. 3
  29. 2
  30. 2
  31. Λάθος
  32. 2
  33. Λάθος
  34. 3
  35. 1

36. 1
37. 2
38. 3
39. 2
40. 3
41. 2
42. 4
43. 3
44. 1,3,4

- Κεφάλαιο 3
    1. figs/myfig.jpg
    2. 1
    3. 2
    4. 3,4
    5. 4
    6. pix3.png
    7. loop
    8. autoplay
    9. 4
    10. sandbox
    11. 2
    12. object
    13. 1
    14. 2
    15. 1
- Κεφάλαιο 4
    1. 1
    2. 1
    3. Λάθος
    4. 1
    5. 2
    6. 2
    7. 2 και 3
    8. 4
    9. 2 και 3
    10. 1, 2, 3 και 4
    11. 2, 3 και 4
    12. 4
    13. 3 και 4
    14. 1
    15. 3
    16. 78
    17. 40
    18. 1
    19. 3
    20. 36
    21. 1 και 3
    22. Σωστό
    23. 2
    24. 3
    25. Λάθος
    26. 2
- Κεφάλαιο 5
    1. Σωστό
    2. 3

3. 1
4. 2
5. 1
6. 1
7. 1
8. 2
9. 2
10. Λάθος
11. Λάθος
12. 3
13. Λάθος
14. 3
15. 3
16. 4
17. 3
18. Λάθος
19. 1
20. 3
21. 1
22. 4
23. 4
24. 2
25. Λάθος
26. Λάθος
27. 2
28. 4
29. 1
30. Λάθος
31. 1
32. 1
33. Λάθος
34. Σωστό

- Κεφάλαιο 6
  1. Σωστό
  2. 2
  3. Λάθος
  4. 2 και 3
  5. 1
  6. 2
  7. 1
  8. 3
  9. 3
  10. 4
  11. 1
  12. 3
  13. 4
  14. Σωστό
  15. 4
  16. Λάθος
- Κεφάλαιο 7
  1. Λάθος
  2. V8
  3. 2
  4. Λάθος
  5. 2

6. defer
7. 2
8. 1
9. γάτες
10. 4
11. 2
12. 2
13. 3
14. 3
15. 1, 2, 3, 5
16. 1, 2, 3, 4
17. Σωστό
18. 10

- Κεφάλαιο 8
  1. 1
  2. null
  3. 2
  4. 1
  5. true
  6. 10
  7. 32
  8. 3
  9. true
  10. 3
  11. 3
  12. 1
  13. 3
  14. 3
  15. false
  16. 85
  17. ce
  18. -1
  19. ‘56’
  20. b
  21. 2
  22. 4
  23. 12
  24. 5
  25. i+1
  26. 3
  27. 1
  28. 3
  29. 4
- Κεφάλαιο 9
  1. 3
  2. 2
  3. 1
  4. 4
  5. 66
  6. 3
  7. 1-2-3
  8. 4
  9. 25
  10. 5
  11. 4

12. 5
13. 2
14. this
15. 3
16. 2
17. 1
18. 1
19. 3
20. 5
21. 1
22. 3
23. 4
24. 1,2,3
25. 1
26. 2
27. 3
28. 10**x
29. 4
30. 14

- Κεφάλαιο 10
    1. 4
    2. 1
    3. 1
    4. Λάθος
    5. 1
    6. 3
    7. 3
    8. Σωστό
    9. 2, 3, 4
    10. 1
    11. Σωστό
    12. 1, 3
    13. 4
    14. 2
    15. 2
    16. 3
    17. 2
    18. Λάθος
    19. window
    20. 3
    21. OK
    22. 1
    23. 3
    24. 2
    25. allSettled
- Κεφάλαιο 11
    1. Λάθος
    2. 4 και 6
    3. Σωστό
    4. Λάθος
    5. Λάθος
    6. 1
    7. Σωστό
    8. Σωστό
    9. Σωστό

10. Λάθος
11. Σωστό
12. Σωστό
13. Λάθος
14. Σωστό
15. Σωστό
16. Λάθος
17. 1, 2, 4, 5, 6
18. Λάθος
19. 3
20. 2
21. 2

- Κεφάλαιο 12
    1. 2
    2. 1
    3. 1
    4. 1
    5. 1, 2 και 3
    6. 1 και 2
    7. 2 και 5
    8. 1, 2, 4 και 5
    9. Λάθος
    10. Λάθος
    11. 1
    12. Λάθος
    13. 1, 2, 4 και 5
    14. Σωστό
    15. 3
    16. 4
    17. Σωστό
    18. Λάθος
    19. Λάθος
    20. 3
    21. 3
    22. 3
    23. 3
    24. 1
    25. 1
    26. 1
    27. Λάθος
    28. 3
    29. 1
- Κεφάλαιο 13
    1. 3
    2. 4
    3. 2
    4. 4
    5. 1
    6. 4
    7. 1
    8. 4
    9. 2
    10. 5
- Κεφάλαιο 14
    1. Λάθος

2. Σωστό
3. 4
4. 2
5. Λάθος

Το παρόν σύγγραμμα αποτελεί διδακτικό βοήθημα ενός εισαγωγικού μαθήματος στον προγραμματισμό διαδικτυακών εφαρμογών. Επιχειρείται εισαγωγή στην αρχιτεκτονική του Διαδικτύου, ξεκινώντας από το μοντέλο πελάτη/εξυπηρετητή και το πρωτόκολλο HTTP. Καλύπτονται οι βασικές τεχνολογίες του Ιστού στην πλευρά του φυλλομετρητή, HTML, CSS, JavaScript, καθώς και η βιβλιοθήκη Bootstrap που επεκτείνει τη CSS. Επίσης, περιγράφεται ο εξυπηρετητής που στηρίζεται στο περιβάλλον Node.js της JavaScript, και βιβλιοθήκες όπως η Express.js, και καλύπτονται θέματα σύνδεσης με τις βάσεις δεδομένων και αυθεντικοποίησης χρηστών.

Το παρόν σύγγραμμα δημιουργήθηκε στο πλαίσιο του Έργου ΚΑΛΛΙΠΟΣ+	
Χρηματοδότης	Υπουργείο Παιδείας και Θρησκευμάτων, Προγράμματα ΠΔΕ, ΕΠΑ 2020-2025
Φορέας υλοποίησης	ΕΛΚΕ ΕΜΠ
Φορέας λειτουργίας	ΣΕΑΒ/Παράρτημα ΕΜΠ/Μονάδα Εκδόσεων
Διάρκεια 2ης Φάσης	2020-2023
Σκοπός	Η δημιουργία ακαδημαϊκών ψηφιακών συγγραμμάτων ανοικτής πρόσβασης (περισσότερων από 700) • Προπτυχιακών και μεταπτυχιακών εγχειριδίων • Μονογραφιών • Μεταφράσεων ανοικτών textbooks • Βιβλιογραφικών Οδηγών
Επιστημονικά Υπεύθυνος	Νικόλαος Μήτρου, Καθηγητής ΣΗΜΜΥ ΕΜΠ
ISBN: 978-618-228-104-8	**DOI:** http://dx.doi.org/10.57713/kallipos-337

Το παρόν σύγγραμμα χρηματοδοτήθηκε από το Πρόγραμμα Δημοσίων Επενδύσεων του Υπουργείου Παιδείας.