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.

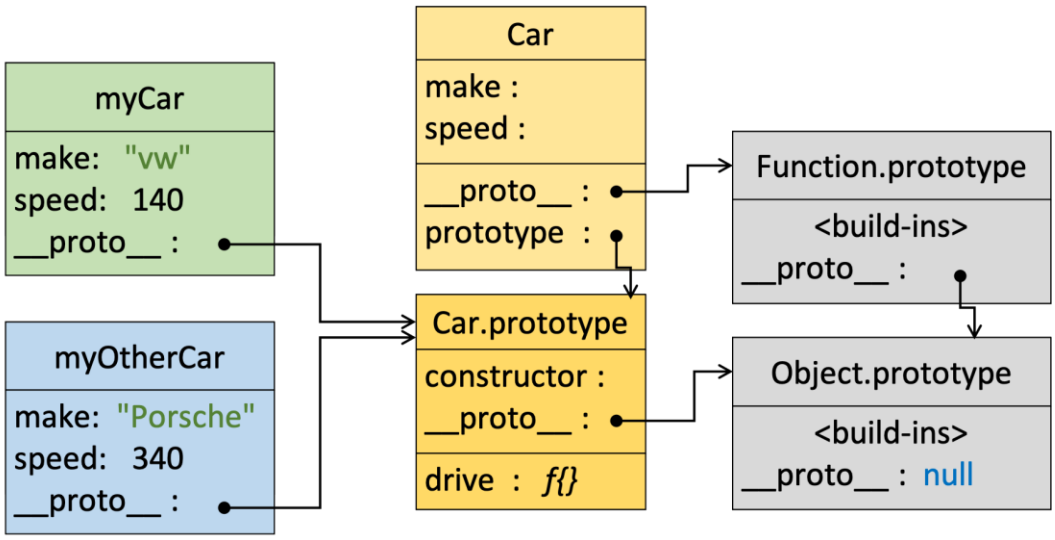
Να σημειωθεί εδώ ότι αυτός ο ορισμός παράγει αντικείμενα με τη μέθοδο στο πρωτότυπο (πρώτη μέθοδος της προηγούμενης ενότητας).

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.