Βασίλης Νικολαΐδης

Προγραμματισμός σε R

Η γλώσσα R, το οικοσύστημα και οι δυνατότητές της

ΒΑΣΙΛΗΣ ΝΙΚΟΛΑΪΔΗΣ
Αναπληρωτής Καθηγητής,
Τμήμα Λογιστικής και Χρηματοοικονομικής
Πανεπιστήμιο Πελοποννήσου

Προγραμματισμός σε R

Η γλώσσα R, το οικοσύστημα και οι δυνατότητές της

Προγραμματισμός σε R

Συγγραφή

Βασίλης Ν. Νικολαΐδης

Συντελεστές έκδοσης

Γλωσσική Επιμέλεια: Όλγα Σταυρουλοπούλου

Γραφιστική Επιμέλεια: Όλγα Σταυρουλοπούλου

Οπτικός Έλεγχος: Όλγα Σταυρουλοπούλου

Κεντρική Ομάδα Υποστήριξης

Γλωσσικός Έλεγχος: Γεωργία Τριανταφυλλίδου

Γραφιστικός Έλεγχος: Ηλίας Τσιώνης

Βιβλιοθηκονομική Επεξεργασία: 'Ελενα Αδαμοπούλου

Copyright © 2023, ΚΑΛΛΙΠΟΣ, ΑΝΟΙΚΤΕΣ ΑΚΑΔΗΜΑΪΚΕΣ ΕΚΔΟΣΕΙΣ

Το παρόν έργο αδειοδοτείται υπό τους όρους της άδειας Creative Commons Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Παρόμοια Διανομή 4.0. Για να δείτε ένα αντίγραφο της άδειας αυτής επισκεφτείτε τον ιστότοπο https://creativecommons.org/licenses/by-nc-sa/4.0/deed.el

Αν τυχόν κάποιο τμήμα του έργου διατίθεται με διαφορετικό καθεστώς αδειοδότησης, αυτό αναφέρεται ρητά και ειδικώς στην οικεία θέση.

ΚΑΛΛΙΠΟΣ

Εθνικό Μετσόβιο Πολυτεχνείο

Ηρώων Πολυτεχνείου 9, 15780 Ζωγράφου

www.kallipos.gr

ISBN: 978-618-5667-90-0

Βιβλιογραφική Αναφορά: Νικολαΐδης, Β. (2023). *Προγραμματισμός σε R* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. http://dx.doi.org/10.57713/kallipos-100

Πίνακας περιεχομένων

Πίνακας περιεχομένων ... 7

Πίνακας συντομεύσεων-ακρωνυμίων ... 12

Εισαγωγή ... 13

Κεφάλαιο 1: Ξεκινώντας με την R ... 15

1.1 Τι είναι η R ... 15

1.2 Εγκατάσταση της R ... 15

1.3 Εγκατάσταση του RStudio Desktop ... 16

1.4 Μία πρώτη επαφή με την R στο RStudio Desktop ... 17

1.4.1 Το παράθυρο Console ... 17

1.4.2 Άλλα παράθυρα και καρτέλες στο RStudio ... 19

1.4.2.1 Η καρτέλα Help ... 19

1.4.2.2 Η καρτέλα Packages ... 20

1.4.2.3 Οι καρτέλες Plots και Viewer ... 20

1.4.2.4 Η καρτέλα Files ... 21

1.4.2.5 Η καρτέλα Connections ... 21

1.4.2.6 Η καρτέλα History ... 21

1.4.2.7 Η καρτέλα Environment ... 21

1.5 Χρήση και διαχείριση πακέτων ... 22

1.5.1 Χρήση πακέτων ... 23

1.5.2 Πρόσβαση στην τεκμηρίωση των πακέτων ... 24

1.5.3 Εγκατάσταση και διαχείριση πρόσθετων πακέτων ... 24

1.5.4 Εγκατάσταση πακέτων από πηγές εκτός CRAN ... 25

1.6 Άλλα IDE, GUI, βοηθήματα και πηγές ... 26

1.7 Σχετικά με τα παραδείγματα του βιβλίου ... 27

Αναφορές Κεφαλαίου 1 ... 28

Κεφάλαιο 2: Τα βασικά στοιχεία της R ... 31

2.1 Χρήση βασικών συναρτήσεων και τελεστών ... 31

2.2 Μεταβλητές, αριθμητικές συναρτήσεις και τύποι αντικειμένων ... 34

2.2.1 Δημιουργία, χρήση, μετατροπή μεταβλητών και αριθμητικά δεδομένα ... 35

2.2.2 Το καθολικό περιβάλλον (Global Environment) ... 38

2.2.3 Ο ρόλος των περιβαλλόντων ... 39

2.2.4 Ειδικές τιμές ... 40

2.3 Συναρτήσεις κειμένου και ο βασικός τύπος character ... 41

2.3.1 Σύνθεση κειμένου ... 42

2.3.2 Έξοδος και εμφάνιση κειμένου ... 43

2.3.3 Ανάλυση και επεξεργασία κειμένου ... 45

2.3.3.1 Χρήσιμες συναρτήσεις επεξεργασίας κειμένου ... 45
2.3.3.2 Κανονικές εκφράσεις (regular expressions) ... 45
2.3.4 Είσοδος κειμένου ... 48
2.3.5 Υποστήριξη τοπικών γλωσσών ... 49
2.3.6 Κείμενο και εντολές ... 49
2.4 Εισαγωγή στα διανύσματα (vector) ... 50
2.4.1 Οι βασικές εντολές των διανυσμάτων ... 50
2.4.2 Δύο παραδείγματα χρήσης vector και η σημασία της ειδικής τιμής NA ... 53
2.4.2.1 Χειρισμός ειδικών τιμών (NA, NaN κλπ.) ... 56
2.5 Λογικές πράξεις, συγκρίσεις και ο βασικός τύπος logical ... 57
2.5.1 Συναρτήσεις σύγκρισης ... 57
2.5.2 Λογικές πράξεις ... 58
2.5.3 Τι χρησιμεύουν τα αντικείμενα τύπου logical ... 59
2.6 Τυχαίοι αριθμοί ... 60
2.7 Οι συναρτήσεις-βοηθήματα: str, summary και dput ... 63
2.8 Αποθήκευση του User Workspace ... 65
Αναφορές Κεφαλαίου 2 ... 66
Κεφάλαιο 3: Τα βασικά εργαλεία του προγραμματιστή R ... 67
3.1 Σενάρια (R-script) ... 67
3.1.1 Δημιουργία και εκτέλεση R script ... 68
3.1.2 Σχολιασμός του κώδικα ... 69
3.1.3 Βοηθήματα εντοπισμού λαθών (debugging) ... 69
3.1.4 Εργαλεία προφίλ (profiling) ... 70
3.1.5 Εργαλείο αναφοράς (report) ... 73
3.1.6 Φάκελοι αρχείων και ο τρέχων φάκελος εργασίας ... 75
3.1.7 Έργο (project) ... 76
3.1.8 Μερικές παρατηρήσεις σχετικές με τη συγγραφή σεναρίων ... 77
3.2 Προγραμματιστικές τεχνικές ... 79
3.2.1 Μπλοκ κώδικα ... 79
3.2.1.1 Επιστρεφόμενο αποτέλεσμα ενός μπλοκ κώδικα ... 80
3.2.1.2 Χρήση μπλοκ κώδικα για ορισμό εμβέλειας μεταβλητών ... 80
3.2.2 Έλεγχος ροής εκτέλεσης (control flow) ... 81
3.2.2.1 Η βασική δομή επιλογής (if και else) ... 82
3.2.2.2 Άλλες μέθοδοι επιλογής (switch, ifelse) ... 83
3.2.2.3 Επαναλήψεις (βρόχοι ή loop) ... 85
3.2.2.4 Τερματισμός εκτέλεσης ... 91
3.3 Στοιχεία διεπαφής χρήστη (user interface) ... 91
Αναφορές Κεφαλαίου 3 ... 93
Κεφάλαιο 4: Συνήθεις τύποι αντικειμένων ... 95

4.1 Συνήθεις ατομικοί τύποι αντικειμένων 95
4.1.1 Ο τύπος vector (διάνυσμα) 95
4.1.2 Οι τύποι factor και ordered (παράγοντας) 99
4.1.3 Πίνακες (matrix και array) 102
4.1.3.1 Ο τύπος matrix (πίνακας 2 διαστάσεων) 102
4.1.3.2 Ο τύπος array (πίνακας n διαστάσεων) 111
4.1.3.3 Η συνάρτηση outer 114
4.1.3.4 Η οικογένεια συναρτήσεων apply 116
4.2 Συνήθεις μη-ατομικοί τύποι αντικειμένων 121
4.2.1 Ο τύπος list (λίστα) 121
4.2.1.1 Ο τελεστής επιλογής $ 124
4.2.1.2 Επεξεργασία αντικειμένων τύπου list 124
4.2.1.3 Εφαρμογή συναρτήσεων σε list 125
4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων 126
4.2.1.5 Η λίστα επιλογών συνεδρίας 128
4.2.2 Ο τύπος environment (περιβάλλον) 129
4.2.2.1 Περιβάλλοντα και συναρτήσεις 131
4.2.2.2 Περιβάλλοντα και ο μηχανισμός αναφοράς 133
4.2.2.3 Περιβάλλοντα για δημιουργία δομών δεδομένων 135
4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων) 136
4.3 Άλλοι τύποι αντικειμένων 142
4.3.1 Οι τύποι tibble και data.table 142
4.3.1.1 Ο τύπος tibble 142
4.3.1.2 Ο τύπος data.table 144
4.3.2 Ο τύπος Matrix (αραιοί και πυκνοί πίνακες) 147
4.3.3 Οι τύποι ts και xts (χρονοσειρές) 150
4.3.4 Αντικείμενα τύπου language (expression, call, name και formula) 152
4.3.5 Ο τύπος function (συνάρτηση) 154
Αναφορές Κεφαλαίου 4 155
Κεφάλαιο 5: Συναρτήσεις και συναρτησιακός προγραμματισμός 157
5.1 Συναρτήσεις 157
5.1.1 Δημιουργία συναρτήσεων 157
5.1.2 Παράμετροι 160
5.1.3 Έλεγχος παραμέτρων 163
5.1.4 Έγερση και χειρισμός σφαλμάτων 164
5.1.5 Αλληλεπίδραση με το περιβάλλον 169
5.1.6 Συναρτήσεις ως ορίσματα και επιστρεφόμενες τιμές 171
5.1.7 Αναδρομή (recursion) 173
5.1.8 Παραδείγματα 174

5.2 Σωληνώσεις (pipe) 178
5.3 Συναρτησιακός προγραμματισμός 180
5.4 Η συναρτησιακή προσέγγιση στον πραγματικό κόσμο 181
Αναφορές Κεφαλαίου 5 186
Κεφάλαιο 6: Κλάσεις και αντικειμενοστραφής προγραμματισμός 187
6.1 Γιατί κλάσεις; 187
6.2 Αντικειμενοστραφής προγραμματισμός στην R 188
6.3 Κλάσεις S3 189
6.4 Κλάσεις S4 194
6.5 Κλάσεις αναφοράς 200
6.5.1 Κλάσεις RS 201
6.5.2 Κλάσεις R6 204
Αναφορές Κεφαλαίου 6 208
Κεφάλαιο 7: Συνεργασία με άλλες γλώσσες προγραμματισμού 209
7.1 Πολυγλωσσικές λύσεις 209
7.2 Tcl/Tk 209
7.3 Python 216
7.4 C++ 220
Αναφορές Κεφαλαίου 7 226
Κεφάλαιο 8: Δημιουργία πακέτων 227
8.1 Τι εξυπηρετούν τα πακέτα 227
8.2 Δομή φακέλου για δημιουργία πακέτου 227
8.3 Αρχείο DESCRIPTION 228
8.4 Χτίσιμο ενός πακέτου 228
8.5 Παραδείγματα δημιουργίας πακέτου 229
8.5.1 Από αντικείμενα που ήδη υπάρχουν 229
8.5.1.1 Αρχεία τεκμηρίωσης (.Rd) 230
8.5.1.2 Αρχείο NAMESPACE 230
8.5.1.3 Ολοκλήρωση της διαδικασίας 231
8.5.2 Από αρχεία R 231
8.5.3 Από το RStudio 232
Αναφορές Κεφαλαίου 8 233
Κεφάλαιο 9: Δεδομένα και περιεχόμενο 235
9.1 Εισαγωγή και εξαγωγή δεδομένων 235
9.1.1 Δεδομένα σε κείμενο 236
9.1.2 Αποθήκευση και ανάκληση αντικειμένων 238
9.1.3 Ανάγνωση δεδομένων web 239
9.1.4 Ένα εργαλείο για πολλές μορφές δεδομένων 239
9.2 Γραφικές παραστάσεις 239

9.2.1 Το υπόβαθρο: πακέτα ‘grDevices’ και ‘grid’ 240
9.2.2 Πακέτο ‘graphics’ 242
9.2.3 Πακέτο ‘lattice’ 247
9.2.4 Πακέτο ‘ggplot2’ 248
9.2.5 Διαδραστικά γραφήματα 250
9.3 Εφαρμογές web 251
9.3.1 Πακέτο ‘beakr’ 251
9.3.2 Πακέτο ‘shiny’ 253
9.4 Δυναμικά έγγραφα 254
Αναφορές Κεφαλαίου 9 258
Παραρτήματα 261
Π.1 Επίλυση προβλημάτων μετά την εγκατάσταση των R και RStudio 261
Π.1.1 Προβλήματα που σχετίζονται με τα Ελληνικά 261
Π.1.2 Αδυναμία εγκατάστασης πρόσθετων πακέτων επέκτασης 261
Π.1.3 Αλλαγή του προεπιλεγμένου φακέλου εργασίας 262
Π.2 Το iris και άλλα σύνολα δεδομένων 263
Αναφορές – Βιβλιογραφία 264

Πίνακας συντομεύσεων-ακρωνυμίων

ΛΣ	Λειτουργικό Σύστημα
CRAN	The Comprehensive R Archive Network
GUI	Graphical User Interface
IDE	Integrated Development Environment
SQL	Structured Query Language

Εισαγωγή

Το σύστημα R, είναι ένα ευρέως χρησιμοποιούμενο εργαλείο, ειδικά σε εφαρμογές στατιστικής, ανάλυσης δεδομένων και μηχανικής μάθησης. Η επεκτασιμότητά της R, την έχει κάνει κατάλληλη για εφαρμογές σε πολλούς τομείς και γνωστικά πεδία. Η R, μπορεί να χρησιμοποιηθεί με διαδραστικό τρόπο, χωρίς προγραμματισμό, αλλά είναι ταυτόχρονα και γλώσσα προγραμματισμού.

Σε αυτή τη δεύτερη φύση της R, επικεντρώνεται το βιβλίο αυτό, στοχεύει δηλαδή στην R ως γλώσσα προγραμματισμού. Παρουσιάζοντας την R σε φοιτητές Κοινωνικών Επιστημών, διαπιστώσαμε την ανάγκη για ένα βιβλίο που να είναι προσαρμοσμένο στις ιδιαιτερότητες της γλώσσας R, αλλά να περιέχει ταυτόχρονα και ύλη εισαγωγής στον προγραμματισμό. Ένα τέτοιο βιβλίο θα πρέπει να βοηθά τα πρώτα βήματα με τον προγραμματισμό, τις σχετικές έννοιες και τα εργαλεία, πάντα όμως προσαρμοσμένα στη χρήση της R και τοποθετημένα με μια σειρά που δεν θα προβληματίζει πολύ τον νέο χρήστη της γλώσσας αυτής. Αυτή την προσέγγιση προσπαθεί να ακολουθήσει το βιβλίο αυτό, αλλά ελπίζουμε να είναι χρήσιμο και σε χρήστες με υπόβαθρο στον προγραμματισμό ή και εμπειρία με την R. Οι ιδιαιτερότητες της γλώσσας R και του «οικοσυστήματός» της και ο σχεδιασμός της ως εργαλείο στατιστικής και γλώσσας προγραμματισμού ταυτόχρονα, κάνουν την R να διαφέρει αρκετά από άλλες γενικές γλώσσες. Επιπρόσθετα η R και τα πακέτα επέκτασής της δεν ακολουθούν πάντα κάποια ενιαία προγραμματιστική “οδηγία”, αλλά γίνεται χρήση διαφόρων μοτίβων προγραμματισμού, κάτι που πιθανότατα δυσκολεύει ακόμα περισσότερο την εμβάθυνση στη γλώσσα αυτή.

Η ίδια η γλώσσα R, οι επεκτάσεις της και τα σχετικά με αυτή εργαλεία, συνεχώς βελτιώνονται και προσαρμόζονται σε νέες εξελίξεις και τάσεις. Είναι ανέφικτο για ένα βιβλίο να αποτυπώσει πλήρως, σε βάθος και διαχρονικά, θέματα που σχετίζονται με ένα τέτοιο ευρύ, δυναμικό και εξελισσόμενο «οικοσύστημα». Παρόλα αυτά, το βιβλίο θα προσπαθήσει να βοηθήσει τον αρχάριο στον προγραμματισμό και τον νέο χρήστη της R, αλλά και να συμπληρώσει άλλα εξειδικευμένα βιβλία δίνοντας μια γεύση από κάποιες χρήσιμες δυνατότητες της γλώσσας που ίσως παραβλέπουν ακόμα και οι πιο έμπειροι χρήστες της. Στο βιβλίο γίνεται μια παρουσίαση της γλώσσας ως ένα γενικό προγραμματιστικό εργαλείο, αρχικά χρησιμοποιώντας απλά, οικεία παραδείγματα γενικού προγραμματισμού και παρακάμπτοντας, σε κάποιο βαθμό, την παρουσίαση της ως επιστημονικό εργαλείο. Εξάλλου, συχνά η εκμάθηση αυτής της ευέλικτης γλώσσας γίνεται με στόχο τη χρήση της σε άλλα γνωστικά πεδία, δραστηριότητες ή μαθήματα (π.χ. στατιστικής, οικονομετρίας, βιοεπιστημών κλπ.), όπου εκεί προφανώς γίνεται και ουσιαστικότερη εμβάθυνση στις δυνατότητες που παρέχει για τον συγκεκριμένο τομέα. Παρουσιάζοντας την R ως μια γενική γλώσσα προγραμματισμού, οδηγούμαστε σε θέματα που ελπίζουμε να είναι χρήσιμα ασχέτως του πεδίου στο οποίο εφαρμόζεται.

Συνοψίζοντας, το πρώτο μέρος του βιβλίου (Κεφ. 1 έως 5), αφορά την αρχική επαφή με τον προγραμματισμό και την R, με αξιοποίηση (όπου χρειάζεται) και του περιβάλλοντος ανάπτυξης εφαρμογών RStudio. Η παρουσίαση της ύλης γίνεται μέσω παραδειγμάτων τα οποία συνοδεύονται από επεξήγηση και σχόλια σε κάθε γραμμή κώδικα. Στο δεύτερο μέρος (Κεφ. 6 έως 9), παρουσιάζονται πιο προχωρημένα θέματα που σχετίζονται με την ανάπτυξη σύνθετων εφαρμογών, πακέτων επέκτασης και λύσεων. Έτσι, ξεκινώντας από μηδενική βάση, γίνεται προοδευτική παρουσίαση όλο και πιο προχωρημένων θεμάτων με έμφαση στη χρήση παραδειγμάτων, που έχει ως τελικό στόχο την εισαγωγή σε πολλές δυνατότητες της γλώσσας R, πέραν της χρήσης της ως εργαλείο στατιστικής και ανάλυσης δεδομένων, τομείς οι οποίοι καλύπτονται εκτενώς από άλλα αξιόλογα βιβλία.

Το βιβλίο αυτό, δεν φιλοδοξεί να καλύψει όλες τις δυνατότητες της R. Η γλώσσα αυτή και τα εργαλεία που παράγει το οικοσύστημά της είναι ένα ευρύτατο, δυναμικό, εξελισσόμενο, πολύπλευρο θέμα το οποίο είναι αδύνατον να καλυφθεί από ένα βιβλίο. Επιπροσθέτως, πολλές δυνατότητες της R (ειδικά στον τομέα της στατιστικής) παρουσιάζονται με εκτενή τρόπο σε άλλα αξιόλογα και εξειδικευμένα βιβλία, οπότε δεν τους δόθηκε προτεραιότητα στο βιβλίο αυτό. Παρόλα αυτά, το βιβλίο, προσπαθεί να παρουσιάσει ένα ευρύ πεδίο θεμάτων που σχετίζονται με την R, με κοινό μοτίβο την προσέγγιση της R ως προγραμματιστικό εργαλείο, ως γλώσσα προγραμματισμού και όχι ως εργαλείο στατιστικής. Καθώς το βιβλίο αυτό γράφτηκε μέσα σε συγκεκριμένο χρονοδιάγραμμα, μοιραία θα περιέχει κάποια λάθη, ασάφειες ή και αβλεψίες. Επιπρόσθετα, κάποια από όσα αναφέρονται μπορεί να έχουν υποστεί αλλαγές από τον χρόνο συγγραφής του βιβλίου. Ακολουθώντας το πνεύμα του ανοικτού λογισμικού - όπως εξάλλου είναι και η ίδια R - η συνεισφορά του αναγνώστη με επισημάνσεις και προτάσεις βελτίωσης τυχών μελλοντικών εκδόσεων του είναι καλοδεχούμενη.

Κεφάλαιο 1: Ξεκινώντας με την R

Σύνοψη
Το κεφάλαιο αυτό περιγράφει κάποια πρώτα βήματα εξοικείωσης με την R για απολύτως αρχαρίους. Αναφέρονται θέματα όπως η εγκατάσταση της γλώσσας, τα σχετικά περιβάλλοντα ανάπτυξης εφαρμογών, τα αποθετήρια πακέτων επέκτασης και η διαχείριση των πακέτων, η εισαγωγή εντολών, η χρήση της τεκμηρίωσης κλπ.

Προαπαιτούμενη γνώση
Βασικές γνώσεις χρήσης υπολογιστών, εξοικείωση με προγράμματα υπολογιστικών φύλλων (προαιρετικό).

1.1 Τι είναι η R

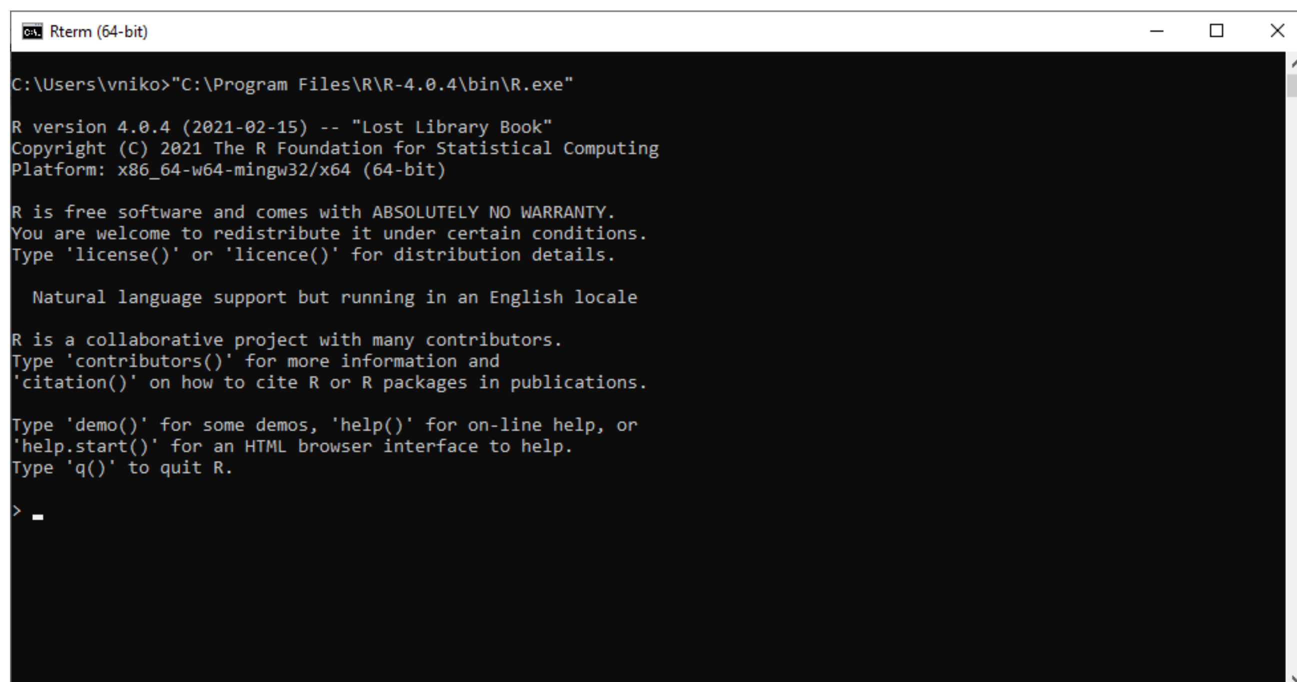
Το σύστημα R, είναι ένα ευρέως χρησιμοποιούμενο εργαλείο λογισμικού, δημοφιλές σε εφαρμογές στατιστικής, ανάλυσης δεδομένων και μηχανικής μάθησης. Αρχικά, δημιουργήθηκε ως ένα εργαλείο στατιστικής το οποίο ενσωματώνει μια γλώσσα προγραμματισμού για να υποβοηθά την επίλυση σχετικών προβλημάτων. Η γλώσσα R, δημιουργήθηκε λοιπόν για εφαρμογές σε συγκεκριμένο πεδίο (domain-specific language) και όχι ως μια γενική γλώσσα προγραμματισμού. Η επεκτασιμότητα όμως του συστήματος R, την έκανε κατάλληλη για χρήση και σε πολλούς άλλους τομείς, εφαρμογές και γνωστικά πεδία. Οι επεκτάσεις της R, προσθέτουν δυνατότητες στη γλώσσα, οι οποίες επιτρέπουν τη χρήση της ως γενική γλώσσα προγραμματισμού καθώς και την υλοποίηση λύσεων που ξεπερνούν τα πεδία εφαρμογής ενός κλασικού εργαλείου στατιστικής. Όπως αναφέρει η R Core Team, δηλαδή η ομάδα ανάπτυξης της γλώσσας: «πολλοί χρήστες θεωρούν την R ως ένα σύστημα στατιστικής. Εμείς προτιμούμε να το σκεπτόμαστε ως ένα περιβάλλον εντός του οποίου υλοποιούνται στατιστικές τεχνικές» [1]. Η R είναι ανοικτό λογισμικό, διατίθεται ελεύθερα, τρέχει σε διάφορα λειτουργικά συστήματα υπολογιστών και έχει υιοθετηθεί από μεγάλες εταιρείες και οργανισμούς. Υπάρχουν πολλά και αξιόλογα ελληνόφωνα συγγράμματα που αφορούν τη γλώσσα R και τις εφαρμογές της. Τα περισσότερα επικεντρώνονται στην εφαρμογή της γλώσσας σε πεδία που η γλώσσα αυτή είναι ιδιαίτερα δημοφιλής, δηλαδή τη στατιστική, την ανάλυση δεδομένων και γενικότερα συναφείς τομείς επίλυσης προβλημάτων δεδομένων για τα οποία η R είναι εξαρχής σχεδιασμένη και ιδιαίτερα κατάλληλη (βλ. ενδεικτικά [2] - [16]).

Η R, είναι διάλεκτος και εξέλιξη του συστήματος S και βασίζεται σε αυτό. Όπως αναφέρει ο John Chambers, δημιουργός του S, το εργαλείο σχεδιάστηκε ώστε να μπορούν οι χρήστες του να το αξιοποιήσουν με διαδραστικό τρόπο, χωρίς προγραμματισμό, δίνοντας απευθείας εντολές επεξεργασίας στο σύστημα (όπως επιτρέπουν και άλλα συστήματα στατιστικής π.χ. το SAS). Αλλά, προσθέτει ο Chambers, «στη συνέχεια, καθώς οι ανάγκες τους γίνονταν σαφέστερες και η πολυπλοκότητά τους αυξανόταν, θα έπρεπε να είναι σε θέση να γλιστρήσουν σταδιακά στον προγραμματισμό, όπου οι πτυχές της γλώσσας και του συστήματος γίνονται πιο σημαντικές» Η δυνατότητα μετάβασης των απλών χρηστών σε προγραμματιστές είναι βασικό μέρος του σχεδιασμού του εργαλείου [17] [18]. Η R, ως εξέλιξη της γλώσσας S, ακολουθεί κι αυτή την ίδια παραπάνω φιλοσοφία, είναι δηλαδή ένα εργαλείο που μπορεί να χρησιμοποιηθεί διαδραστικά, χωρίς προγραμματισμό, αλλά ταυτόχρονα είναι και μια γλώσσα προγραμματισμού.

1.2 Εγκατάσταση της R

Για να χρησιμοποιήσετε την R πρέπει να την εγκαταστήσετε στον υπολογιστή σας. Υπάρχουν τρόποι (και λόγοι) να χρησιμοποιηθεί μία εγκατάσταση της R που «τρέχει» σε κάποιον διακομιστή (server) και όχι τοπικά στον υπολογιστή, αλλά η τοπική εγκατάσταση είναι ο πλέον συνήθης και ευέλικτος τρόπος χρήσης της. Παρακάτω περιγράφεται σύντομα η εγκατάσταση της R σε έναν τυπικό υπολογιστή με λειτουργικό σύστημα Microsoft Windows. Η R, διατίθεται επίσης για Mac OS X και διάφορες διανομές του Linux. Είναι λογισμικό ανοιχτού κώδικα, και διατίθεται με ελεύθερη άδεια χρήσης (GPL-2 | GPL-3). Η βασική πηγή για την R είναι το αποθετήριο CRAN (Comprehensive R Archive Network) [19] το οποίο διαχειρίζεται η R Foundation for Statistical Computing, δηλαδή ο επίσημος φορέας ανάπτυξης και διαχείρισης της R. Η κεντρική ιστοσελίδα του

CRAN βρίσκεται στην παρακάτω διεύθυνση https://cran.r-project.org. Επιλέξτε “Download R for Windows”, μετά “base”, και τέλος “Download R”, και εκτελέστε το αρχείο ώστε να γίνει η εγκατάσταση[1].



Εικόνα 1.1 *: Η R σε παράθυρο εντολών.*

Μετά την ολοκλήρωση της εγκατάστασης θα βρείτε στην Έναρξη των Windows μια ομάδα R που περιέχει συντόμευση για την R ή πιο συγκεκριμένα για το πρόγραμμα RGui (R Graphical User Interface). επίσης, θα έχει δημιουργηθεί ένας φάκελος με όνομα R (εξ ορισμού στο φάκελο Έγγραφα του χρήστη), όπου η γλώσσα θα τοποθετεί τα πακέτα επέκτασης της R που ίσως θέλετε να προσθέσετε στο σύστημά σας[2]. Έχετε πλέον ότι χρειάζεται για να χρησιμοποιήσετε την R. Η ίδια η γλώσσα είναι απλώς ένας “διερμηνευτής” (interpreter) των εντολών σας, που περιμένει τις εντολές σας για να τις μεταφράσει και να τις εκτελέσει μία - μία. Αν γνωρίζετε αρκετά, μπορείτε να εντοπίσετε το κυρίως αρχείο της R (ονομάζεται R.exe) και να το τρέξετε σε Γραμμή Εντολών των Windows (βλ. Εικόνα 1.1). Κάθε εντολή που γράφετε εδώ, θα στέλνεται στην R και θα εκτελείται άμεσα.

Αν και ο παραπάνω τρόπος χρήσης της R από τη Γραμμή Εντολών είναι πλήρης σε δυνατότητες, δεν προσφέρει ιδιαίτερα βοηθήματα στον προγραμματιστή. Έτσι, χρησιμοποιούνται συνήθως άλλα προγράμματα που συνεργάζονται με την R (ή και άλλες γλώσσες προγραμματισμού), που είναι σχεδιασμένα να διευκολύνουν τον προγραμματιστή ενώ παρέχουν και εργαλεία για την καλύτερη αξιοποίηση της γλώσσας. Ένα τέτοιο πρόγραμμα είναι το RGui που εγκαθίσταται μαζί με την R για Windows (και εκτελείται όταν επιλέγετε το εικονίδιο της R), αλλά πολύ περισσότερες δυνατότητές παρέχει το λογισμικό RStudio που αναφέρεται στην επόμενη ενότητα.

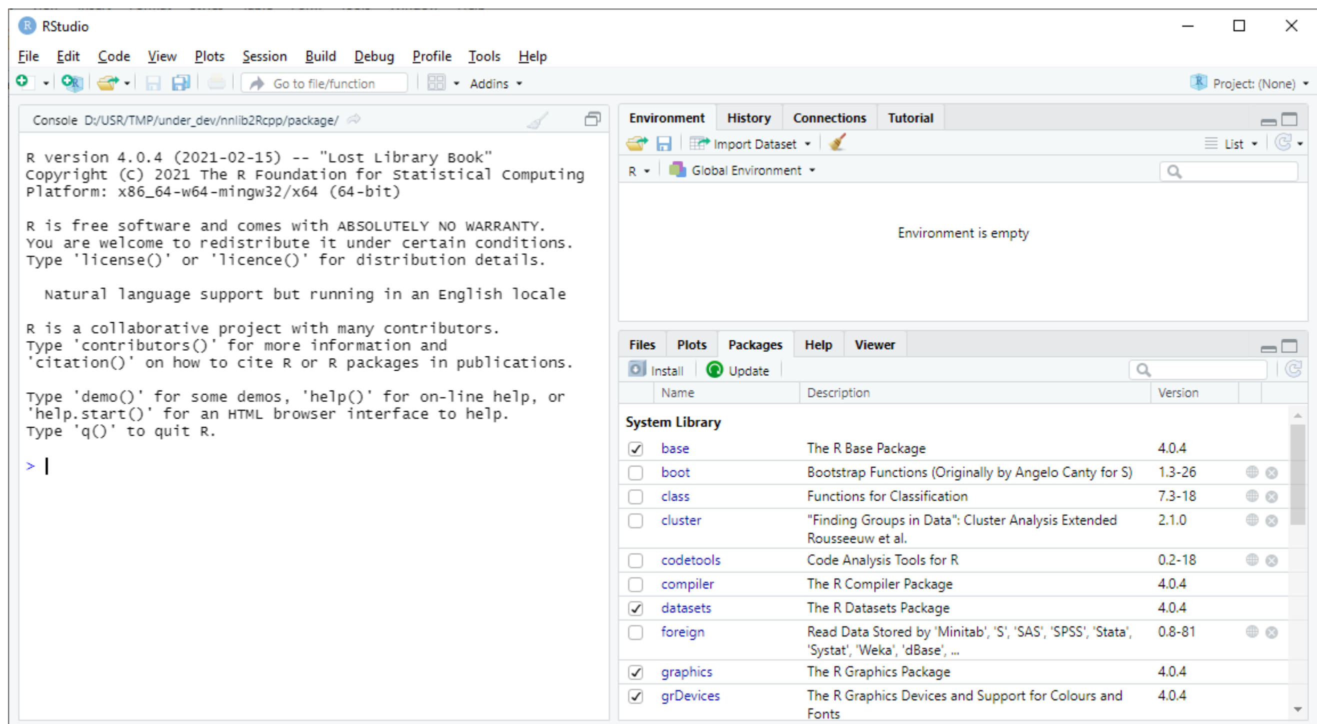
1.3 Εγκατάσταση του RStudio Desktop

Η εργασία των προγραμματιστών διευκολύνεται από μια κατηγορία προγραμμάτων που στόχο έχουν να παρέχουν απαραίτητα εργαλεία και βοηθήματα για τη συγγραφή κώδικα. Τα προγράμματα αυτά ονομάζονται IDE (Integrated Development Environment). Πολλά IDE υποστηρίζουν την R (μεταξύ αυτών τα Eclipse/StatET, Microsoft Visual Studio και IntelliJ), αλλά το RStudio Desktop [20] (βλ. Εικόνα 1.2) είναι ένα IDE που δημιουργήθηκε ακριβώς για την R. Αν και η εγκατάσταση του RStudio είναι προαιρετική, διευκολύνει σημαντικά τη χρήση της. Είναι ιδιαίτερα δημοφιλές στους χρήστες της R και προέρχεται από μία δραστήρια

[1] Σε κάποια συστήματα ίσως χρειαστούν δικαιώματα διαχειριστή. Σε αυτή την περίπτωση, εκτελέστε με δεξί-κλικ στο αρχείο εγκατάστασης και την επιλογή «Εκτέλεση ως Διαχειριστής» (Run as Administrator).

2 βλ. και σχετικό Παράρτημα Π.1 Επίλυση προβλημάτων μετά την εγκατάσταση των R και RStudio.

κοινότητα η οποία έχει δημιουργήσει (και συνεχίζει να δημιουργεί) ενδιαφέροντα εργαλεία λογισμικού καθώς και πακέτα επέκτασης.



Εικόνα 1.2 *Το RStudio Desktop.*

Περισσότερα για το RStudio υπάρχουν στον διαδικτυακό του τόπο, στη διεύθυνση https://rstudio.com όπου διατίθεται δωρεάν και η έκδοση RStudio Desktop. Έτσι, μετά την εγκατάσταση της R συνιστούμε να εγκαταστήσετε το RStudio Desktop (διατίθεται για Windows, Linux και Mac OS X) καθώς τα βήματα που περιγράφονται σε αυτό το βιβλίο θα βασίζονται συχνά στη χρήση του περιβάλλοντος αυτού.

Πέραν της R, το RStudio υποστηρίζει τη δημιουργία κώδικα σε διάφορες γλώσσες προγραμματισμού (π.χ. C++, Python, Javascript, SQL κ.α.) αλλά και τη συγγραφή διαφόρων τύπων εγγράφων και άλλων τμημάτων που μπορεί να αποτελούν μέρος μιας της υλοποίησης κάποιας σύνθετης λύσης λογισμικού[3].

1.4 Μία πρώτη επαφή με την R στο RStudio Desktop

Όπως αναφέρθηκε παραπάνω, στο βιβλίο αυτό θα γίνεται κυρίως χρήση της R μέσω του RStudio Desktop (θα το αναφέρουμε από εδώ και πέρα απλά ως RStudio). Η χρήση του RStudio αν και δεν είναι απαραίτητη, βελτιώνει σημαντικά την εμπειρία με την R. Ξεκινώντας το RStudio εμφανίζει ένα περιβάλλον παρεμφερές με αυτό που φαίνεται στην Εικόνα 1.2. Οι επόμενες παράγραφοι σκοπεύουν να σας εξοικειώσουν με κάποιες από τις καρτέλες που περιέχει το περιβάλλον αυτό.

1.4.1 Το παράθυρο Console

Η R λειτουργεί ως διερμηνευτής (δηλαδή διερμηνέας) που μεταφράζει άμεσα κάθε εντολή R σε χαμηλότερου επιπέδου εντολές που θα εκτελεστούν από το λειτουργικό σύστημα και τον υπολογιστή. Ο χρήστης ξεκινά μια συνεδρία (session ή σύνοδο) με την R κατά την οποία η R περιμένει κείμενο με εντολές που περιγράφουν τι πρέπει να κάνει. Ένα τέτοιο κείμενο στην R ονομάζεται «έκφραση» (expression). Για να το μεταφράσει και να το εκτελέσει, η έκφραση πρέπει να έχει σωστή σύνταξη (βάσει των κανόνων της γλώσσας) και να είναι πλήρης. Κάθε εντολή, μεταφράζεται σε εκτελέσιμο κώδικα, εκτελείται και ακολούθως η R προχωρά στην επεξεργασία της επόμενης (ή περιμένει να της δοθεί μία αν δεν υπάρχει). Αυτό συνεχίζεται μέχρι να τελειώσει η συνεδρία,

[3] Ένα έργο ανάπτυξης λογισμικού (που αποτελείται συνήθως από πολλά μέρη) αποκαλείται συχνά και «λύση» (solution).

κάτι που θα συμβεί όταν τερματίσετε την R[4]. Έτσι μπορείτε να δίνετε απευθείας εντολές στην R και (αφού πατήσετε Enter) να έχετε άμεσα την απόκριση από τη γλώσσα. Εντοπίστε το παράθυρο με τον τίτλο "Console" (ίσως να γράφει επίσης τη διαδρομή του τρέχοντα φακέλου εργασίας μετά το "Console"). Στο παράθυρο αυτό, δοκιμάστε τα παρακάτω:

Δοκιμάστε:	**Σχόλιο**
`2+2`	Γράψτε το στο Console και πατήστε Enter. Η απάντηση της R είναι (προφανώς) 4.

Τι έγινε:

1. Αρχικά η R περιμένει την επόμενη εντολή σας εμφανίζοντας το «σύμβολο προτροπής» (prompt) δηλαδή το >. Με το σύμβολο αυτό η R σας προτρέπει να γράψετε κάτι.
2. Γράφετε 2+2 και πατάτε Enter. Το σύμβολο + είναι εδώ ένας «τελεστής», δηλαδή ένας χαρακτήρας που παρέχει κάποια λειτουργία, συγκεκριμένα καλεί τη συνάρτηση της πρόσθεσης.
3. Η R απαντά εμφανίζοντας στο Console το παρακάτω: [1] 4, δηλαδή έτρεξε την εντολή σας και το 1ο (και μοναδικό) μέρος του αποτελέσματός από την εκτέλεση της εντολής (αυτό δείχνει το [1]) είναι το 4. Εφόσον αυτό το αποτέλεσμα δεν αποθηκεύτηκε σε κάποια μεταβλητή εμφανίζεται ως έξοδος, κάτι που ονομάζεται «αυτόματη εκτύπωση» (auto-print).
4. Μετά η R περιμένει την επόμενη εντολή σας (εμφανίζοντας πάλι το >).

Δοκιμάστε:	**Σχόλιο**
`1/4`	Γράψτε το στο Console και πατήστε Enter.

Τι έγινε:

1. Αρχικά η R περιμένει την επόμενη εντολή σας εμφανίζοντας το «σύμβολο προτροπής» >.
2. Γράφετε 1/4 και πατάτε Enter. Το / είναι ο τελεστής της διαίρεσης.
3. Η R απαντά: [1] 0.25, δηλαδή έτρεξε την εντολή σας και το μοναδικό αποτέλεσμα της είναι το 0.25 (παρατηρήστε πως η R χρησιμοποιεί την τελεία για διαχωριστικό των δεκαδικών ψηφίων ακόμα και αν το λειτουργικό σύστημα στον υπολογιστή σας είναι ρυθμισμένο διαφορετικά)[5].
4. Μετά η R περιμένει την επόμενη εντολή σας (εμφανίζοντας πάλι το >).

Δοκιμάστε:	**Σχόλιο**
`1/`	Γράψτε το στο Console και πατήστε Enter.

Τι έγινε:

1. Αρχικά η R περιμένει την επόμενη εντολή σας εμφανίζοντας το «σύμβολο προτροπής» >.
2. Γράφετε 1/ και πατάτε Enter.
3. Η εντολή σας είναι ημιτελής (ένα δια κάτι, αλλά τι;) και έτσι η R δεν μπορεί να απαντήσει.
4. Η R σας προτρέπει να προσθέσετε ό,τι λείπει από την εντολή που ξεκινήσατε εμφανίζοντας το σύμβολο +. Εδώ το + το εμφάνισε η R και δεν είναι παρά μια ένδειξη ότι η έκφραση που γράψατε είναι ημιτελής.
5. Τώρα υπάρχουν δύο επιλογές: είτε να ολοκληρώσετε την εντολή σας (π.χ. γράφοντας 4 και Enter) οπότε η R θα υπολογίσει το 1/4 όπως παραπάνω ή να πατήσετε το πλήκτρο Esc και να ακυρώσετε την ημιτελή εντολή.
6. Μετά η R περιμένει την επόμενη εντολή σας (εμφανίζοντας πάλι το >).

Το τελευταίο παράδειγμα ακολουθεί διαφορετικό μοτίβο. Παραδοσιακά, τα βιβλία εκμάθησης γλωσσών προγραμματισμού ξεκινούν πάντα με την εντολή που εμφανίζει το κείμενο Hello World!. Ας δοκιμάσουμε λοιπόν κάτι αντίστοιχο:

Δοκιμάστε:	**Σχόλιο**
`cat("Hello world!")`	Γράψτε το στο Console και πατήστε Enter.

[4] Η συνεδρία προφανώς τερματίζεται και κλείνοντας το RStudio. Επιπρόσθετα, στο RStudio υπάρχει δυνατότητα τερματισμού και επανεκκίνησης της R από τις επιλογές μενού Session/Terminate R και Session/Restart R αντίστοιχα.
[5] Υπάρχει σχετική ρύθμιση και στην R (βλ. Π.1.1 Προβλήματα που σχετίζονται με τα Ελληνικά).

Όπως παραπάνω, η εντολή σας εκτελείται και εμφανίζει απλώς το κείμενο:

```
Hello world!
```

Οι λειτουργίες της R παρέχονται σχεδόν εξ ολοκλήρου μέσα από συναρτήσεις[6]. Η cat είναι μια τέτοια συνάρτηση και η λειτουργία της είναι να εμφανίζει κείμενο. Οι συναρτήσεις συνήθως καλούνται με το όνομά τους ακολουθούμενο από τις τιμές για τις παραμέτρους τους (τα ορίσματα τους) μέσα σε παρένθεση. Εδώ π.χ. δόθηκε ως παράμετρος της cat το κείμενο που θέλουμε να εμφανιστεί. Αυτός ο κλασσικός τρόπος κλήσης και εκτέλεσης συναρτήσεων θα σας είναι ήδη οικείος αν έχετε εργαστεί με συναρτήσεις σε προγράμματα υπολογιστικών φύλλων (όπως το Microsoft Excel, Libreoffice Calc κ.α.), όμως στην R υπάρχουν και άλλοι χρήσιμοι τρόποι να συνταχθεί η κλήση συναρτήσεων. Μια άλλη βασική διαφορά με τα προγράμματα υπολογιστικών φύλλων είναι πως εδώ τα κεφαλαία θεωρούνται διαφορετικά από τα μικρά γράμματα στα ονόματα (πακέτων, συναρτήσεων, μεταβλητών, αντικειμένων κλπ.), δηλαδή η R είναι μια case-sensitive γλώσσα**.** Έτσι το παρακάτω:

Δοκιμάστε:	**Σχόλιο**
`cAt("Hello world!")`	Γράψτε το στο Console και πατήστε Enter. Αφήστε το A κεφαλαίο.

θα επιστρέψει μήνυμα λάθους, εφόσον κάποια συνάρτηση με όνομα cAt (με κεφαλαίο A) δεν μπορεί να βρεθεί.

Συνοψίζοντας, το Console απλώς δέχεται εντολές και (αν χρειάζεται) εμφανίζει τα πιθανά αποτελέσματα ως κείμενο. Μπορείτε οποιαδήποτε στιγμή να καθαρίσετε όσα εμφανίζονται στο Console πατώντας Ctrl+L (δηλαδή με πατημένο το πλήκτρο Ctrl πιέζετε το πλήκτρο L). Αυτό δεν έχει κανένα άλλο αποτέλεσμα πέραν του οπτικού (έχετε πλέον ένα Console καθαρό από κείμενο).

1.4.2 Άλλα παράθυρα και καρτέλες στο RStudio

Η προεπιλεγμένη διάταξη παραθύρων του RStudio εμφανίζει δύο ακόμα παράθυρα (panes)[7]. Στα δύο αυτά παράθυρα του RStudio εμφανίζονται διάφορες καρτέλες (tab). Στο πάνω μέρος κάθε καρτέλας υπάρχουν εικονίδια για χρήσιμες λειτουργίες που αφορούν τη συγκεκριμένη καρτέλα. Παρακάτω περιγράφονται μερικές από τις καρτέλες αυτές:

1.4.2.1 Η καρτέλα Help

Η καρτέλα **Help** (συνήθως βρίσκεται στο παράθυρο κάτω δεξιά) εμφανίζει τεκμηρίωση, δηλαδή περιεχόμενο βοήθειας. Αν δοκιμάσατε το παράδειγμα με την cat στην προηγούμενη παράγραφο, ίσως αναρωτηθήκατε, τι σημαίνει cat στην R (δεν σημαίνει "γάτα") ή ίσως προσέξατε πως το κείμενο «Hello world!» εμφανίστηκε χωρίς να προηγείται η ένδειξη [1], όπως έγινε στις αριθμητικές πράξεις παραπάνω. Σε αυτή την περίπτωση, ας ζητήσουμε βοήθεια[8]:

Δοκιμάστε:	**Σχόλιο**
`help("cat")`	Γράψτε το στο Console και πατήστε Enter.

Εναλλακτικά θα μπορούσατε να γράψετε help(cat) ή απλώς ?cat. Σε κάθε περίπτωση, η καρτέλα Help του RStudio θα εμφανίσει τη βοήθεια για τη συνάρτηση cat. Καθώς χρησιμοποιείτε την R, θα εκτιμήσετε την πληθώρα τεκμηρίωσης που παρέχει αυτή και τα πακέτα επέκτασής της, με κείμενα βοήθειας που ακολουθούν ένα κοινό μοτίβο το οποίο διευκολύνει αρκετά την αναζήτηση πληροφορίας (αργότερα στο βιβλίο αυτό περιγράφεται και η δημιουργία τεκμηρίωσης για τις δικές σας συναρτήσεις). Ήδη στη βοήθεια για την cat υπάρχει πληροφορία όπως:

---

[6] Ο αριθμός των δεσμευμένων λέξεων της γλώσσας R είναι ιδιαίτερα μικρός, βλ. help(Reserved).

[7] Η διάταξη αυτή μπορεί να διαμορφωθεί διαφορετικά από το Pane Layout των επιλογών του RStudio (θα τις βρείτε στο μενού Tools/Global Options).

[8] Στο βιβλίο αυτό γίνεται συχνά παραπομπή στην τεκμηρίωση που συνοδεύει την R (παρέχεται από τη συνάρτηση help ή το σύμβολο ?). Όπου συμβαίνει αυτό, η αναφορά αφορά την έκδοση 4.1 της R (διαθέσιμη κατά τη περίοδο συγγραφής του βιβλίου) και την επίσημη πηγή της [1]).

(α) ότι είναι μια συνάρτηση που περιλαμβάνεται στο πακέτο ‘base’ (βλ. πάνω αριστερά).

(β) ότι το όνομα της συνάρτησης είναι συντομογραφία του concatenate and print (συγχώνεψε [πολλά αντικείμενα] σε ένα κείμενο και εμφάνισε το αποτέλεσμα).

(γ) ότι η συγκεκριμένη συνάρτηση δεν είναι generic[9], δηλαδή ένα γενικό όνομα που μπορεί να αφορά πολλές συναρτήσεις, ενώ η επιλογή της μίας από αυτές που τελικά θα εκτελεστεί εξαρτάται από τον τύπο των αντικειμένων στις παραμέτρους. Αν μια συνάρτηση είναι generic αυτό συνήθως καταγράφεται στα πρώτα στοιχεία της περιγραφής της στην τεκμηρίωση.

(δ) ότι χρησιμοποιείται με τον τρόπο που περιγράφεται στην παράγραφο Usage. Εκεί καταγράφονται οι παράμετροι που δέχεται, οι προεπιλεγμένες τιμές τους κλπ.

(ε) ότι επιστρέφει αυτό που περιγράφεται στην παράγραφο Value, όπου π.χ. για την cat αναφέρει πως δεν επιστρέφει τίποτα (ή πιο σωστά, επιστρέφει invisible NULL).

(στ) ποιες άλλες συναρτήσεις σχετίζονται με τη συνάρτηση αυτή (στην παράγραφο See Also) όπου αναφέρει π.χ. τη συνάρτηση print.

(ζ) παραδείγματα χρήσης της συνάρτησης (στην παράγραφο Examples). Τα παραδείγματα μπορούν να εκτελεστούν άμεσα με τη συνάρτηση example, εδώ δηλαδή με την εντολή example(cat).

Όλα αυτά μπορεί να μη σημαίνουν πολλά για τον νέο χρήστη της R αλλά σύντομα θα εξηγήσουμε γιατί είναι χρήσιμα. Παρεμπιπτόντως, για όσους ίσως αναρωτήθηκαν τον λόγο που η R δεν πρόσθεσε την ένδειξη [1] μπροστά από το κείμενο «Hello world!» στο προηγούμενο παράδειγμα, η απάντηση υπάρχει στην τεκμηρίωση της: όπως αναφέρει το κείμενο βοήθειας της cat στην παράγραφο Value, η συγκεκριμένη συνάρτηση ουσιαστικά δεν επιστρέφει καμία τιμή άρα - κατά κάποιο τρόπο - δεν υπάρχει αποτέλεσμα[10]. Αντίθετα, οι αριθμητικές συναρτήσεις στα παραδείγματα που προηγήθηκαν επέστρεφαν κάτι, συγκεκριμένα το αποτέλεσμα της πράξης.

1.4.2.2 Η καρτέλα Packages

Η καρτέλα **Packages** (παράθυρο κάτω δεξιά) αφορά τα πακέτα επέκτασης της R που είναι εγκατεστημένα στο σύστημά σας. Στην ορολογία της R, η συλλογή αυτή από εγκατεστημένα πακέτα ονομάζεται «βιβλιοθήκη» (library). Κάθε πακέτο είναι μια συλλογή από συναρτήσεις, δεδομένα, μεταγλωττισμένο κώδικα, τεκμηρίωση κλπ. Για να χρησιμοποιήσει η R τα περιεχόμενα ενός πακέτου από τη βιβλιοθήκη, πρέπει αυτό να συνδεθεί με την R (δηλαδή να “φορτωθεί” και ενεργοποιηθεί). Περισσότερα για τη διαχείριση και χρήση πακέτων θα βρείτε στη σχετική ενότητα του βιβλίου (βλ.§1.5 Χρήση και διαχείριση πακέτων). Για την ώρα απλώς παρατηρήστε πως στην κατηγορία System Library (δηλαδή στη βιβλιοθήκη από πακέτα που εγκαθίστανται αυτόματα μαζί με την R) υπάρχει το πακέτο ΄base΄ και είναι ήδη συνδεδεμένο με την R (δηλαδή “φορτωμένο”), αφού υπάρχει ένα checkmark (τικ) στο σχετικό κουτάκι δίπλα του. Γενικά, από την καρτέλα αυτή μπορείτε να εγκαταστήσετε, να ενημερώσετε, να συνδέσετε, να αποσυνδέσετε ή να απεγκαταστήσετε (αφαιρώντας οριστικά από τη βιβλιοθήκη) πακέτα επέκτασης της R. Παρεμπιπτόντως, το πακέτο ΄base΄ το έχετε ήδη χρησιμοποιήσει καθώς περιέχει την cat αλλά και τις αριθμητικές συναρτήσεις που χρειάστηκαν στα παραπάνω παραδείγματα. Μπορείτε επίσης να δείτε τα κείμενα βοήθειας που αφορούν τα περιεχόμενα κάθε πακέτου (δηλαδή το βασικό μέρος της τεκμηρίωσης του) με κλικ στο όνομα του πακέτου. Για παράδειγμα, επιλέγοντας το πακέτο ‘base’ με το ποντίκι θα εμφανιστεί λίστα με όλα τα αντικείμενα που περιέχει και για τα οποία υπάρχει τεκμηρίωση. Π.χ. ανατρέχοντας στη λίστα αυτή μπορείτε να εντοπίσετε την cat και να εμφανίσετε την βοήθεια που παρουσιάστηκε προηγουμένως.

1.4.2.3 Οι καρτέλες Plots και Viewer

Οι καρτέλες **Plots** και **Viewer** (παράθυρο κάτω δεξιά) χρησιμεύουν για την εμφάνιση περιεχομένου που δημιουργεί ο κώδικάς σας, είτε γραφημάτων (στην καρτέλα Plots) είτε περιεχομένου Web (ιστοσελίδας, στην καρτέλα Viewer). Για παράδειγμα, δοκιμάστε τις παρακάτω συναρτήσεις γραφικών παραστάσεων:

---

[9] Generic σημαίνει γενικό, κοινό, γενόσημο. Πολλές συνήθεις συναρτήσεις της R (όπως οι sum, mean, plot, summary) είναι generic. Για το ρόλο των generic συναρτήσεων, βλ. Κεφάλαιο 6 και ειδικότερα §6.3 Κλάσεις S3.

[10] Αυτό δεν είναι απολύτως ακριβές, οι συναρτήσεις στην R *πάντα* επιστρέφουν κάτι. Όπως αναφέρει και η τεκμηρίωση της cat, η συνάρτηση επιστρέφει ένα “αόρατο NULL” (το NULL είναι ειδική τιμή που συμβολίζει το “τίποτα/κενό”, βλ. §2.2.4 Ειδικές τιμές).

Δοκιμάστε:	**Σχόλιο**
`plot(iris[1:4])`	Η plot δημιουργεί γραφήματα διασποράς.
`boxplot(Petal.Length~Species,data=iris)`	Η boxplot δημιουργεί θηκογράμματα.

Το παράδειγμα δημιουργεί δύο γραφήματα στην καρτέλα Plots. Στο πάνω μέρος της καρτέλας υπάρχουν εργαλεία με τα οποία μπορείτε να περιηγηθείτε στα γραφήματα, να τα διαγράψετε, να τα εμφανίσετε σε μεγαλύτερο παράθυρο, να τα μετατρέψετε σε αρχείο εικόνας κλπ. Παρατήρηση: τα γραφήματα στο παραπάνω παράδειγμά εμφανίζουν δεδομένα από το σύνολο δεδομένων iris που περιέχεται στο πακέτο ‘datasets’ (για περισσότερα βλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων), ενώ οι δύο συναρτήσεις που χρησιμοποιούνται (plot και boxplot) περιέχονται στο πακέτο ‘graphics’. Και τα δύο αυτά πακέτα είναι ενσωματωμένα στην εγκατάσταση της R και συνδέονται μαζί της αυτόματα όταν ξεκινάει η R (κάτι που μπορείτε να ελέγξετε από την καρτέλα Packages όπως αναφέρθηκε παραπάνω).

1.4.2.4 Η καρτέλα Files

Η καρτέλα **Files** (παράθυρο κάτω δεξιά) δείχνει αρχεία και φακέλους στον υπολογιστή σας. Ξεκινά από τον τρέχοντα φάκελο εργασίας (working directory), εκεί δηλαδή που αυτή τη στιγμή η R θα αναζητήσει ή θα δημιουργήσει αρχεία καθώς εκτελεί τις σχετικές εντολές σας. Μπορείτε να αλλάξετε φάκελο και να περιηγηθείτε σε αυτόν ή να ανοίξετε, να αντιγράψετε, να μετονομάσετε, να διαγράψετε, να μετακινήσετε και γενικά να χειριστείτε αρχεία και φακέλους, όπως κάνετε συνήθως κατά τον χειρισμό αρχείων στο λειτουργικό σύστημα. επίσης, μπορείτε από εδώ να ορίσετε άλλον τρέχοντα φάκελο εργασίας, με τη σχετική επιλογή στο πάνω μέρος της καρτέλας.

1.4.2.5 Η καρτέλα Connections

Η καρτέλα Connections (παράθυρο πάνω δεξιά) αφορά τη διαχείριση συνδέσεων με εξωτερικές πηγές δεδομένων (όπως βάσεις δεδομένων).

1.4.2.6 Η καρτέλα History

Η καρτέλα **History** (παράθυρο πάνω δεξιά) εμφανίζει το ιστορικό των εντολών που έχετε ήδη εκτελέσει στην Console ώστε να μπορείτε να τις ξαναχρησιμοποιήσετε εύκολα. Τα εικονίδια της καρτέλας επιτρέπουν την αποθήκευση του ιστορικού σε αρχείο, την ανάκληση ιστορικού από αρχείο, την αντιγραφή μιας ή περισσότερων εντολών στην Console (για να τις στείλετε ξανά προς εκτέλεση) ή σε κάποιο αρχείο σεναρίου R που επεξεργάζεστε (περισσότερα για τα αρχεία αυτά παρακάτω). Τέλος μπορείτε να διαγράψετε επιλεγμένες ή όλες τις εντολές που υπάρχουν στο ιστορικό. Χρήσιμο είναι να προσέξετε πως όταν γράφετε στην Console πατώντας τα πλήκτρα άνω και κάτω βέλος του πληκτρολογίου, μπορείτε να ανακαλέσετε (και να ξαναχρησιμοποιήσετε) εντολές από το ιστορικό.

1.4.2.7 Η καρτέλα Environment

Κλείνοντας την περιήγηση στις καρτέλες του RStudio, σημαντικότατη είναι η καρτέλα **Environment** (παράθυρο πάνω δεξιά). Εδώ εμφανίζονται τα ονόματα για αντικείμενα (δεδομένα, συναρτήσεις κλπ.) που υπάρχουν αυτή τη στιγμή στην τρέχουσα συνεδρία (session) της R, δηλαδή τα αντικείμενα που έχουν ανατεθεί σε μεταβλητές και είναι άμεσα προσβάσιμα από την R. Για τα αντικείμενα αυτά εμφανίζονται οι ιδιότητες και οι τρέχουσες τιμές τους και είναι κατηγοριοποιημένα σε αυτά που βρίσκονται στο Global Environment (καθολικό περιβάλλον) και σε αυτά που βρίσκονται σε πακέτα που είναι αυτή τη στιγμή συνδεδεμένα (“φορτωμένα”). Το Global Environment είναι το περιβάλλον μέσα στο οποίο δημιουργούνται συνήθως αντικείμενα από τον κώδικα και τις εντολές του χρήστη (βλ.§2.2.2 Το καθολικό περιβάλλον (Global Environment)). Εκεί θα βρείτε τις μεταβλητές που έχετε δημιουργήσει εσείς (ή κάποιο σενάριο - δηλαδή πρόγραμμα R - που έχετε εκτελέσει).

Δοκιμάστε:	**Σχόλιο**
`a<-5`	Αναθέτει (αποθηκεύει) τον αριθμό 5 σε μεταβλητή με όνομα a.

Παρατηρήστε πως η νέα σας μεταβλητή a εμφανίζεται πλέον στην καρτέλα Environment. Πάνω-δεξιά στην καρτέλα θα βρείτε ένα κουμπί επιλογής του τρόπου εμφάνισης των αντικειμένων, δηλαδή αν θα εμφανίζονται ως μία λίστα (List) ή ως ένα πλέγμα (Grid). Σε κατάσταση List (κάτω από την ένδειξη Values (τιμές) βλέπετε κάτι σαν το παρακάτω:

a	5

Δηλαδή, το όνομα και η τρέχουσα τιμή της μεταβλητής. Σε κατάσταση Grid εμφανίζονται περισσότερα στοιχεία για τις μεταβλητές. Έτσι σε κατάσταση Grid εμφανίζονται για την a τα παρακάτω στοιχεία:

Name	Type	Length	Size	Value
a	Numeric	1	56b	5

Τα παραπάνω καταγράφουν πως στο Global Environment (που περιέχει καθολικές μεταβλητές) υπάρχει μια μεταβλητή a, η οποία αυτή τη στιγμή αφορά ένα αριθμητικό (numeric) αντικείμενο με μήκος (length) ίσο με 1, δηλαδή περιέχει μόνο ένα στοιχείο και η τιμή (Value) του στοιχείου αυτού είναι 5.

Από την καρτέλα Environment μπορείτε να διαγράψετε όλα τα περιεχόμενα του Global Environment ή να επιλέξετε (σε κατάσταση Grid) οτιδήποτε δεν σας είναι πλέον απαραίτητο και να το διαγράψετε. Μπορείτε να κάνετε κλικ (σε κατάσταση List) πάνω στη μεταβλητή που σας ενδιαφέρει και - εφόσον ο τύπος της το υποστηρίζει - να την εμφανίσετε (αυτό εκτελεί τη συνάρτηση View στο αντικείμενο αυτό και αν η μεταβλητή είναι πίνακας εμφανίζει τα περιεχόμενα του, αν η μεταβλητή αφορά συνάρτηση εμφανίζει τον ορισμό της συνάρτησης αυτής, κλπ.). Επιπρόσθετα στην καρτέλα Environment μπορείτε να εμφανίσετε αντικείμενα που βρίσκονται σε άλλα περιβάλλοντα (εκτός του Global Environment), όπως τα περιβάλλοντα που δημιουργούν τα πακέτα όταν συνδέονται με την R. Για να γίνει αυτό, από το σχετικό μενού (πάνω αριστερά) της καρτέλας, κάνετε κλικ στο Global Environment και εμφανίζεται μενού με τα συνδεδεμένα πακέτα. Από εδώ μπορείτε, για παράδειγμα, να επιλέξετε το πακέτο ‘base’ (package:base) και να εμφανίσετε το περιβάλλον του, το οποίο περιέχει διάφορα αντικείμενα, μεταξύ αυτών τις συναρτήσεις (functions) που περιέχει. Έτσι, μπορείτε να εντοπίσετε και να κάνετε κλικ στην cat για να δείτε πώς έχει οριστεί η συνάρτηση αυτή. επίσης, στην καρτέλα Environment μπορείτε να εμφανίσετε και εσωτερικές μεταβλητές συναρτήσεων όταν κάνετε εκσφαλμάτωση (debug) αλλά και μεταβλητές από κώδικα σε γλώσσα Python (αν τη χρησιμοποιείτε παράλληλα με την R, βλ.§7.3 Python). Τέλος, στην καρτέλα Environment υπάρχει το μενού Import Dataset που υποβοηθά τη δημιουργία μεταβλητών με εισαγωγή δεδομένων από αρχεία (κείμενου, Excel, SPSS, SAS, ή Stata). Αν απαιτούνται συγκεκριμένα πακέτα για να γίνει η εισαγωγή των δεδομένων αυτών, το RStudio θα καθοδηγήσει την εγκατάστασή τους, ενώ τα δεδομένα που θα εισαχθούν αποθηκεύονται σε μεταβλητές[11] του Global Environment.

1.5 Χρήση και διαχείριση πακέτων

Στις προηγούμενες ενότητες έγινε μια πρώτη αναφορά στα πακέτα επέκτασης (packages) της R και χρήση κάποιων συναρτήσεων από τα προ-εγκατεστημένα πακέτα, αυτά δηλαδή που εγκαθίστανται αυτόματα μαζί με τη γλώσσα. Τα πακέτα είναι σημαντικότατο κομμάτι του «οικοσυστήματος» της R καθώς παρέχουν τη λειτουργικότητα και δυνατότητες της γλώσσας αυτής. Κάθε πακέτο περιέχει συναρτήσεις, δεδομένα, μεταγλωττισμένο κώδικα, τεκμηρίωση κλπ. Πρακτικά, όλες οι λειτουργίες (ακόμα και βασικές) στην R γίνονται μέσω συναρτήσεων που παρέχονται από κάποιο πακέτο, οπότε τα πακέτα προσφέρουν ένα ευέλικτο τρόπο επέκτασης της γλώσσας. Τα πακέτα που είναι εγκατεστημένα στο σύστημά σας αναφέρονται ως βιβλιοθήκη[12] (Library). Συνήθως, υπάρχουν δύο μέρη της βιβλιοθήκης αυτής, το System Library (βασικά πακέτα που έρχονται με την R και εγκαθίστανται στο φάκελο εγκατάστασης της) και το User Library το οποίο περιλαμβάνει πακέτα που έχετε προσθέσει εσείς (και τοποθετούνται εξ ορισμού μέσα στο home folder του χρήστη[13]. Σε πολλά

[11] Τα σχετικά αντικείμενα είναι συνήθως τύπου data.frame, βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων).

[12] Σε αρκετές άλλες γλώσσες προγραμματισμού «βιβλιοθήκη» ονομάζεται το δικό τους αντίστοιχο των πακέτων.

[13] Στα Microsoft Windows ο φάκελος αυτός είναι συνήθως τα “Εγγραφα” μέσα στον οποίο υπάρχει η διαδρομή “\R\win-library” όπου τοποθετούνται τα πακέτα του χρήστη (σε κάποιες εκδόσεις της R η διαδρομή ίσως είναι άλλη, π.χ. ο υποκατάλογος AppData\Local\R στον φάκελο του χρήστη). Για να ορίσετε διαφορετικό φάκελο στον οποίο θα τοποθετούνται τα πακέτα, βλ. οδηγίες σχετικά με τη μεταβλητή περιβάλλοντος R_LIBS_USER στο Παράρτημα Π.1.2

παραδείγματα του βιβλίου αυτού θα χρειαστεί να εγκαταστήσετε και πρόσθετα πακέτα στο σύστημά σας. Υπενθυμίζεται πως στο RStudio η καρτέλα Packages εμφανίζει τα πακέτα επέκτασης που υπάρχουν στη βιβλιοθήκη σας (βλ.§1.4.2.2 Η καρτέλα Packages). Επιπρόσθετα, στο πακέτο 'utils' υπάρχει η συνάρτηση installed.packages() που επιστρέφει πίνακα με ονόματα και άλλα στοιχεία όλων των εγκατεστημένων πακέτων.

1.5.1 Χρήση πακέτων

Τα πακέτα εγκαθίστανται μια φορά στην R, αλλά παραμένουν ανενεργά. Μπορείτε όμως να χρησιμοποιήσετε απευθείας κάποιο αντικείμενο από ένα εγκατεστημένο πακέτο, αν προσδιορίσετε το πακέτο με το όνομά του και μετά με χρήση διπλής άνω-κάτω τελείας το όνομα του αντικειμένου[14]. Το επόμενο παράδειγμα καλεί τη συνάρτηση xyplot που υπάρχει στο πακέτο 'lattice' [21][15] για να εμφανίσει ένα σημείο στη θέση 10,50 σε γράφημα διασποράς.

Δοκιμάστε:	**Σχόλιο**
`lattice::xyplot(50 ~ 10)`	Καλεί την συνάρτηση xyplot του πακέτου lattice.

Κάθε πακέτο ορίζει ένα περιβάλλον με τα αντικείμενα που περιέχει και έναν χώρο ονομάτων με τα ονόματα των αντικειμένων αυτών στα οποία μπορεί να υπάρχει πρόσβαση από χρήστες του πακέτου. Έτσι το παραπάνω σημαίνει «χρησιμοποίησε την xyplot που αναφέρεται στον χώρο ονομάτων (namespace) που εξάγει το περιβάλλον του lattice»[16]. Όμως, αυτός ο τρόπος (με τον τελεστή ::) πρόσβασης στα αντικείμενα ενός πακέτου χρησιμοποιείται σπάνια, κυρίως όταν πρέπει να αποσαφηνιστεί το πακέτο που περιέχει το αντικείμενο που θέλουμε να χρησιμοποιηθεί. Αν και φορτώνει (load) το πακέτο στην R, δεν το συνδέει με αυτή, δηλαδή δεν προσαρτά τα αντικείμενα του πακέτου ώστε να είναι άμεσα διαθέσιμα σε εντολές της R. Ο συνήθης τρόπος χρήσης των περιεχομένων των πακέτων γίνεται συνδέοντας (προσαρτώντας/attach) ολόκληρα τα πακέτα στην τρέχουσα συνεδρία. Αυτή η μέθοδος όχι μόνο φορτώνει τα πακέτα στη μνήμη αλλά δίνει και άμεση πρόσβαση σε όλα τα ονόματα των αντικειμένων του πακέτου για την υπόλοιπη διάρκεια της συνεδρίας (μέχρι να «κλείσετε» την R). Κάποια βασικά πακέτα από το System Library συνδέονται αυτόματα όταν ξεκινά η R (όπως π.χ. τα πακέτα 'base', 'methods', 'datasets' και 'graphics'[17]), τα υπόλοιπα όμως πακέτα που ίσως είναι απαραίτητα πρέπει να συνδεθούν από τον χρήστη. Για να συνδέσετε ένα πακέτο από τη βιβλιοθήκη, η συνήθης συνάρτηση είναι η **library**[18]. Για παράδειγμα, για να συνδέσετε το πακέτο 'lattice', εκτελείτε την εντολή library(lattice). Το όνομα του πακέτου μπορεί να δοθεί και ως κείμενο μέσα σε εισαγωγικά, π.χ. library("lattice") ή library('lattice'). Αν η εντολή αποτύχει σημαίνει πως το πακέτο αυτό δεν υπάρχει στη βιβλιοθήκη σας και θα πρέπει να το εγκαταστήσετε (βλ. παρακάτω).

Δοκιμάστε:	**Σχόλιο**
`library(lattice)`	Συνδέει (φορτώνει και προσαρτά) το πακέτο lattice.
`xyplot(50 ~ 10)`	Η xyplot του πακέτου lattice είναι πλέον απευθείας διαθέσιμη.

Αν είχατε δοκιμάσει μόνο τη 2η γραμμή του παραπάνω παραδείγματος με τη συνάρτηση xyplot χωρίς κάποια στιγμή πριν να έχετε συνδέσει στη συνεδρία σας το πακέτο 'lattice' στο οποίο περιέχεται η xyplot, αυτό θα οδηγούσε σε μήνυμα λάθους. Η σύνδεση πάντως ενός πακέτου χρειάζεται να γίνει μόνο μία φορά κατά τη συνεδρία με την R, αφού μετά τη σύνδεση ενός πακέτου όλο το περιβάλλον του παραμένει διαθέσιμο για το υπόλοιπο της διάρκειας της συνεδρίας αυτής. Στη σπάνια περίπτωση που χρειάζεται να αποσυνδεθεί πλήρως ένα συνδεδεμένο πακέτο εν μέσω μιας συνεδρίας, αυτό γίνεται με χρήση της συνάρτησης **detach** που αποσυνδέει αντικείμενα. Π.χ. για να αποσυνδεθεί το πακέτο 'lattice', η πλήρης εντολή είναι:

```
detach(package:lattice, unload=TRUE)
```

Αδυναμία εγκατάστασης πρόσθετων πακέτων επέκτασης.

[14] Γενικά ο τελεστής :: δίνει πρόσβαση σε αντικείμενα εντός ενός άλλου χώρου ονομάτων (namespace), όπως π.χ. συναρτήσεις εντός ενός πακέτου (βλ. τη σχετική τεκμηρίωση της R γράφοντας help(Syntax)).

[15] βλ. και §9.2.3 Πακέτο 'lattice'.

[16] Για τον τρόπο που ορίζεται το namespace κατά τη δημιουργία των πακέτων, βλ.§8.5.1.2 Αρχείο NAMESPACE.

[17] Τα πακέτα που συνδέονται αυτόματα μπορεί να διαφέρουν ανάλογα με τις ρυθμίσεις της εγκατάστασης της R.

[18] Η ίδια η συνάρτηση library περιέχεται στο προ-εγκατεστημένο πακέτο base.

(το όνομα του πακέτου μπορεί να δοθεί και ως κείμενο μέσα σε εισαγωγικά). Τέλος, η συνάρτηση **require** μπορεί να χρησιμοποιηθεί αντί της library για τη σύνδεση πακέτων. Η require ελέγχει αν το πακέτο είναι εγκατεστημένο στο σύστημα και επιστρέφει τη σχετική απάντηση. Εφόσον το πακέτο είναι εγκατεστημένο προχωρά στη σύνδεσή του.

1.5.2 Πρόσβαση στην τεκμηρίωση των πακέτων

Τα πακέτα συνήθως περιέχουν σημαντικό περιεχόμενο βοήθειας και άλλο υλικό που υποβοηθά και υποστηρίζει τη χρήση τους. Το RStudio δίνει εύκολη πρόσβαση σε αυτό το υλικό (όπως αναφέρεται και στην §1.4.2.1 Η καρτέλα Help), αλλά υπάρχουν και αρκετές σχετικές συναρτήσεις, βασικότερη όλων η **help** που συνήθως χρησιμοποιείται για να εμφανίσει το κείμενο τεκμηρίωσης κάποιας συνάρτησης από τα συνδεδεμένα πακέτα.

Δοκιμάστε:	**Σχόλιο**
`help.start()`	Εμφανίζει την τεκμηρίωση (βοήθεια) για την ίδια την R.
`help("cat")`	Βοήθεια για τη συνάρτηση cat σε κάποιο συνδεδεμένο πακέτο.
`help(cat)`	Ίδιο με το παραπάνω.
`?cat`	Ίδιο με το παραπάνω.
`?base::cat`	Βοήθεια για την cat του πακέτου ‘base’ (ίδιο με το παραπάνω).
`example(cat)`	Εκτελεί τα παραδείγματα στην τεκμηρίωση για την cat.
`help.search("cat")`	Αναζητά περιεχόμενο βοήθειας που περιέχει το κείμενο “cat”.
`help(package="base")`	Βοήθεια για το εγκατεστημένο πακέτο ‘base’.

Πέραν όμως του απλού help τα πακέτα περιέχουν συχνά εκτεταμένη τεκμηρίωση (ακόμα και ολόκληρες σχετικές επιστημονικές δημοσιεύσεις), τις επονομαζόμενες βινιέτες με τη συνάρτηση **vignette**:

Δοκιμάστε:	**Σχόλιο**
`vignette()`	Διαθέσιμες βινιέτες στα εγκατεστημένα πακέτα.
`vignette(all=FALSE)`	Διαθέσιμες βινιέτες στα συνδεδεμένα πακέτα.
`vignette('Sweave',package='utils')`	Η βινιέτα με θέμα Sweave του πακέτου ‘utils’

Μπορείτε επίσης να δείτε κάποιες από τις παρουσιάσεις (**demo**) που περιέχονται στα πακέτα της R:

Δοκιμάστε:	**Σχόλιο**
`demo()`	Διαθέσιμα demo.
`demo(image, package="graphics")`	Παρουσίαση με όνομα image από το πακέτο graphics.
`demo(Hershey, package="graphics")`	Παρουσίαση με όνομα Hershey από το πακέτο graphics.

Τέλος, πολλά πακέτα έχουν ορίσει τον τρόπο με τον οποίο θέλουν να γίνεται αναφορά τους σε δημοσιεύσεις που τα χρησιμοποιούν, μέσω της συνάρτησης **citation** που επιστρέφει το αποτέλεσμα ως κείμενο αλλά και σε μορφή bibtex.

Δοκιμάστε:	**Σχόλιο**
`citation(lattice)`	Βιβλιογραφική αναφορά για το πακέτο ΄lattice’.

1.5.3 Εγκατάσταση και διαχείριση πρόσθετων πακέτων

Πρόσθετα πακέτα υπάρχουν σε διάφορα δημόσια αποθετήρια (repositories) στο Internet, με βασικότερο, κεντρικότερο και δημοφιλέστερο όλων το αποθετήριο CRAN. Το CRAN (από το οποίο γίνεται η διανομή και της ίδιας της R) είναι το αποθετήριο που διαχειρίζεται ο κεντρικός φορέας της R (το R Foundation for Statistical Computing). Η R είναι εξ ορισμού ρυθμισμένη ώστε να ανατρέχει στο αποθετήριο CRAN όταν αναζητά πρόσθετα πακέτα επέκτασης. Τα πακέτα που παρέχει το CRAN[19] έχουν υποστεί σειρά από ελέγχους πριν διατεθούν, ενώ ο μεγάλος αριθμός χρηστών του λειτουργεί ως ένα περιβάλλον ελέγχου που επιταχύνει τον

[19] Λίστα με πακέτα στο CRAN: http://cran.r-project.org/web/packages/available_packages_by_name.html (5/2021). Τα ονόματα και άλλα στοιχεία των διαθέσιμων πακέτων μπορούν να ανακτηθούν με τη συνάρτηση **available.packages**.

εντοπισμό και τη διόρθωση πιθανών λαθών σε αυτά. Το CRAN έχει και πληθώρα mirrors, δηλαδή εναλλακτικούς διακομιστές (servers) που παρέχουν το ίδιο περιεχόμενο με τον κεντρικό.

Για να εγκαταστήσετε ένα πακέτο από το CRAN το μόνο που χρειάζεται να γνωρίζετε είναι το όνομά του, το οποίο θα δώσετε στη συνάρτηση **install.packages**, π.χ.

```
install.packages("apackage")
```

όπου apackage το υποτιθέμενο όνομα κάποιου πακέτου που θέλετε να εγκαταστήσετε. Η εγκατάσταση θα συμπεριλάβει και πιθανές «εξαρτήσεις» (dependencies), δηλαδή άλλα πακέτα που ίσως είναι απαραίτητα για να λειτουργήσει το πακέτο που εγκαθιστάτε. Τα εγκατεστημένα πακέτα παραμένουν διαθέσιμα στο σύστημα και συνήθως δεν χρειάζεται να εγκατασταθούν ξανά στο μέλλον.

Όταν ο κώδικας R που χρησιμοποιεί κάποιο πακέτο πρέπει να τρέχει και σε άλλα συστήματα στα οποία δεν υπάρχει απευθείας πρόσβαση για διαχείριση των πακέτων της βιβλιοθήκης τους, μπορούν να συνδυαστούν οι προαναφερθείσες εντολές για να εξασφαλιστεί η εγκατάσταση των απαιτούμενων πακέτων. Έτσι, (προτρέχοντας λίγο) ένας καλύτερος τρόπος να γίνει σύνδεση κάποιου πακέτου ‘apackage’ σε κώδικα R που θα χρησιμοποιηθεί από τρίτους (π.χ. σε σενάρια R) είναι με τον παρακάτω συνδυασμό εντολών, ο οποίος αφού πρώτα ελέγξει αν το apackage υπάρχει ήδη στη βιβλιοθήκη το συνδέει (με την συνάρτηση require). Αν δεν υπάρχει, το εγκαθιστά και ακολούθως το συνδέει (μέσω των συναρτήσεων install.packages και library):

```
if(!require(apackage)){
  install.packages("apackage"); library(apackage)}
```

Τα εγκατεστημένα πακέτα παραμένουν διαθέσιμα στο σύστημα σας, αλλά δεν ενημερώνονται αυτόματα. Έτσι ίσως χρειαστεί κάποια στιγμή να τα ενημερώσετε σε νέες εκδόσεις τους, κάτι που γίνεται με τη συνάρτηση update.packages, π.χ. με την επιλογή update.packages(ask="graphics") εμφανίζεται σε παράθυρο μία λίστα πακέτων προς ενημέρωση και μπορείτε να επιλέξετε ποια θέλετε να ενημερωθούν. Για την απεγκατάσταση ενός πακέτου υπάρχει η συνάρτηση remove.packages[20]. Οι διάφορες συναρτήσεις διαχείρισης πακέτων (όπως οι προαναφερθείσες) συνήθως έχουν και μια παράμετρο libs (ή libs.loc) που επιτρέπει να οριστεί συγκεκριμένος φάκελος στον υπολογιστή ως πρόσθετη βιβλιοθήκη, δηλαδή σημείο εναπόθεσης πακέτων (π.χ. η εντολή install.packages("apackage", lib="d:/myRlibs") εγκαθιστά το υποτιθέμενο πακέτο με όνομα ‘apackage’ στο φάκελο d:/myRlibs.

Επιπροσθέτως, τα παραπάνω βήματα εγκατάστασης και ενημέρωσης πακέτων μπορούν να γίνουν εύκολα (και χωρίς εντολές) μέσα από το RStudio από το μενού Tools/Install Packages και Tools/Check for Package Updates, ή από την καρτέλα Packages (βλ.§1.4.2.2 Η καρτέλα Packages) η οποία επιπροσθέτως επιτρέπει τη σύνδεση, αποσύνδεση καθώς και απεγκατάσταση (διαγραφή) πακέτων.

1.5.4 Εγκατάσταση πακέτων από πηγές εκτός CRAN

Με τον μεγάλο αριθμό πακέτων που διαθέτει, το CRAN καλύπτει πλήρως τις ανάγκες των περισσοτέρων χρηστών της R. Παρόλα αυτά, υπάρχουν και άλλες πηγές, εκτός του CRAN, από τις οποίες μπορούν να εγκατασταθούν πακέτα σε ένα σύστημα R, όπως ενδεικτικά[21]:

• Πακέτα μπορεί να διατεθούν και να εγκατασταθούν από αρχεία σε μορφή συμπιεσμένου φακέλου (zip, tar). Τα αρχεία αυτά ακολουθούν συγκεκριμένη δομή και έχουν δημιουργηθεί μέσω μιας σαφώς ορισμένης διαδικασίας (για περισσότερα σχετικά με τη δημιουργία πακέτων βλ. Κεφ.8).

• Πακέτα μπορούν να εγκατασταθούν από διάφορα άλλα αποθετήρια πακέτων. Τα αποθετήρια πακέτων είναι απλώς διαδικτυακά σημεία στα οποία τοποθετούνται πακέτα. Ο καθένας μπορεί να δημιουργήσει ένα αποθετήριο, δημόσιο ή και ιδιωτικό (π.χ. σε εταιρικό περιβάλλον). Στο CRAN υπάρχουν και σχετικά πακέτα για τον σκοπό αυτό, όπως τα πακέτα ‘minicran’ [22] και ‘drat’ [23]. Συνήθεις τρόποι για να γίνεται εγκατάσταση και ενημέρωση πακέτων από ένα τέτοιο αποθετήριο είναι: (α) να ρυθμιστεί η R ώστε να χρησιμοποιεί το αποθετήριο όταν καλείται η install.packages, (β) μέσω των ρυθμίσεων στο RStudio (επιλογή μενού Tools/Global Options) στην κατηγορία Packages και την καρτέλα Management όπου μπορεί να γίνει ορισμός CRAN mirror ή πρόσθεση άλλων δευτερευόντων αποθετηρίων (secondary repositories) και (γ) μέσω της παραμέτρου repos της install.packages π.χ. install.packages("apackage", repos="https://aserver.uop.gr"),

[20] Ένας πιο γρήγορος αλλά λιγότερο ασφαλής τρόπος είναι απλώς να διαγράψετε τα αρχεία του πακέτου από τον σχετικό φάκελο βιβλιοθήκης, αλλά απαιτεί να έχουν τερματιστεί όλες οι συνεδρίες της R και του RStudio.

[21] Περισσότερα για το θέμα αυτό, βλ. https://cran.r-project.org/doc/manuals/r-patched/R-admin.html#Installing-packages

όπου apackage ένα υποθετικό όνομα πακέτου που πρέπει να εγκατασταθεί από κάποιο υποθετικό αποθετήριο στη διεύθυνση https://aserver.uop.gr.

• Πακέτα μπορούν να εγκατασταθούν απευθείας από το GitHub (https://github.com/) που είναι ίσως η πλέον γνωστή πλατφόρμα συνεργασίας και διαμοιρασμού περιεχομένου για όλες τις γλώσσες. Εκεί υπάρχουν και οι τελευταίες εκδόσεις για πολλά πακέτα της R αφού συχνά οι ομάδες ανάπτυξής πακέτων χρησιμοποιούν τη συγκεκριμένη πλατφόρμα για συνεργασία και διαχείριση των εκδόσεων των σχετικών αρχείων. Το πακέτο ‘devtools’ [24] περιέχει (μεταξύ άλλων) υποστήριξη για την εγκατάσταση πακέτων από το GitHub, με τη συνάρτηση install_github().

• Πακέτα μπορεί να εγκατασταθούν μέσω του r-universe (https://r-universe.dev/) που χρησιμοποιεί και οργανώνει περιεχόμενο σχετικό με την R από το GitHub (βλ. παραπάνω). Έτσι, αναδεικνύει υλικό (πακέτα, άρθρα κλπ.) που αφορούν την R, ενώ παρέχει και εργαλεία για το υλικό αυτό (π.χ. δυνατότητα δημιουργίας και ελέγχου των πακέτων παρόμοια με αυτά του CRAN). Τα πακέτα είναι χωρισμένα σε θεματικά «σύμπαντα» (universe) καθένα εκ των οποίων παρέχει ένα αποθετήριο πακέτων που μπορούν να χρησιμοποιηθούν στην install.packages (σε κάθε universe υπάρχουν και οδηγίες καθώς και ο απαραίτητος κώδικας για την εγκατάσταση των σχετικών πακέτων).

• Πακέτα μπορούν να εγκατασταθούν από το αποθετήριο Bioconductor (http://www.bioconductor.org/), κυρίως για πακέτα υπολογιστικής βιολογίας και βιοπληροφορικής και διαθέτει πακέτα ανοικτού κώδικα, ενώ ακολουθεί διαδικασίες υποβολής και ελέγχου αυτών πριν τα διαθέσει (όπως δηλαδή γίνεται και στο CRAN).

• Πακέτα μπορούν να εγκατασταθούν από το Microsoft R Application Network (MRAN) (https://mran.microsoft.com/). Η Microsoft παρέχει μια δική της διανομή R (Microsoft R Open) ενώ στο MRAN υπάρχουν οι τρέχουσες αλλά και παλαιότερες εκδόσεις των πακέτων του CRAN, ώστε να μπορούν να αναζητηθούν και πακέτα που ενδεχομένως έχουν καταργηθεί από το τελευταίο.

1.6 Άλλα IDE, GUI, βοηθήματα και πηγές

Πριν επικεντρωθούμε περαιτέρω στη χρήση της R σε συνδυασμό με το RStudio καλό είναι να αναφερθεί πως πολλά άλλα εργαλεία υποστηρίζουν την R. Την περίοδο που γράφεται το βιβλίο αυτό, διαθέσιμα περιβάλλοντα ανάπτυξης κώδικα R περιλαμβάνουν τα: Microsoft Visual Studio, IntelliJ, StatET (βασισμένο στο Eclipse IDE), Emacs (με το Emacs Speaks Statistics Project) κ.α.

Υπάρχουν επίσης αρκετά γραφικά περιβάλλοντα χρήσης (Graphical User Interface ή GUI) της R, μεταξύ αυτών το δημοφιλές R-Commander [25] που διατίθεται στο πακέτο επέκτασης ‘rcmd’ της R από το CRAN και βοηθά στην εφαρμογή μεθόδων ανάλυσης δεδομένων και στη δημιουργία αναφορών με αυτόματη δημιουργία των σχετικών εντολών R. Άλλα GUI περιλαμβάνουν τα Bluesky και R AnalyticFlow. Για περισσότερα τέτοια εργαλεία βλ. [26].

Τέλος, αξίζει να αναφερθούν τα διάφορα πακέτα (διαθέσιμα στο CRAN ή άλλες πηγές) που στόχο έχουν να διευκολύνουν ή και να αυτοματοποιήσουν την εκτέλεση εργασιών με την R, όπως το πακέτο ‘DataExplorer’ [27] που δημιουργεί αυτόματες αναφορές σχετικά με τα δεδομένα σε κάποιο αντικείμενο R, το πακέτο ‘modelStudio’ [28] που δημιουργεί διαδραστικά διαγράμματα επεξήγησης κάποιου μοντέλου που έχει δημιουργηθεί από τα δεδομένα ή τη συλλογή πακέτων ‘easystats’ που στοχεύει στη διευκόλυνση εφαρμογής στατιστικών μεθόδων και επεξήγησης των αποτελεσμάτων.

Όσον αφορά την τεκμηρίωση, την πληροφόρηση και την απάντηση ερωτημάτων σχετικά με την R, η γλώσσα αυτή, είναι διαδεδομένο εργαλείο ανοιχτού λογισμικού οπότε έχει μεγάλη υποστήριξη από την κοινότητα των χρηστών της. Πέραν των σχετικών με την R βιβλίων και περιοδικών που ασχολούνται πλήρως ή μερικώς με αυτή (όπως τα The R Journal, Journal of Statistical Software και Journal of Open Source Software), στο Διαδίκτυο υπάρχει πληθώρα υλικού που περιλαμβάνει site, blog, ομάδες σε κοινωνικά δίκτυα, σεμινάρια (webminar), e-book, βίντεο κ.α. Έτσι, δεν υπάρχει έλλειψη από πηγές που αξίζει να συμβουλευτεί κανείς είτε είναι νεοεισερχόμενος στον κόσμο της R είτε έμπειρος χρήστης της. Μεταξύ αυτών, το r-project [19] (κεντρικό site του R Foundation for Statistical Computing, επίσημου φορέα ανάπτυξης και διαχείρισης της R, όπου υπάρχουν βιβλία και άλλη τεκμηρίωση για τη γλώσσα), το προαναφερθέν R-universe[22] (που επιτρέπει την αναζήτηση σε πακέτα, άρθρα και δημοσιεύσεις), to site του RStudio [20] (επιλογή Resources), το Big Book of R [29] (online, στο οποίο καταγράφονται πηγές και βιβλιογραφία σχετικές με την R), το R-bloggers [30] (site με συνεχή ροή αναρτήσεων σχετικά με την R), τα code-project [31] και stack-overflow (tag R) [32] (όπου

[22] βλ.§1.5.4 Εγκατάσταση πακέτων από πηγές εκτός CRAN.

χρήστες παραθέτουν κώδικα ή/και απαντούν σε ερωτήματα άλλων χρηστών). Τέλος, όλος ο κώδικας της R και των πακέτων της, ο πηγαίος κώδικας για κάθε συνάρτηση, τύπο ή κλάση που χρησιμοποιείται είναι διαθέσιμος είτε άμεσα (μέσα στην R) είτε έμμεσα (μέσα από το αποθετήριο που τα διέθεσε) και άρα μπορεί να χρησιμοποιηθεί ως παράδειγμα.

1.7 Σχετικά με τα παραδείγματα του βιβλίου

Ήδη παρουσιάστηκαν παραπάνω κάποιες πρώτες εντολές (εκφράσεις) κώδικα R τις οποίες μπορείτε να δοκιμάσετε. Στο βιβλίο αυτό, όσα παραδείγματα μπορούν να εκτελεστούν απευθείας στο Console της R (βλ. προηγούμενη §1.4.1 Το παράθυρο Console) θα δίνονται υπό μορφή πίνακα όπως ο παρακάτω, με τον κώδικα και την επεξήγησή του.

Δοκιμάστε:	**Σχόλιο**
`Εντολή`	Επεξήγηση της λειτουργίας της εντολής.

Σε κάποιες περιπτώσεις όμως, οι εντολές αυτές θα εξαρτώνται από άλλα βήματα που έχουν προηγηθεί στην ίδια ενότητα, οπότε καλό είναι να καταχωρείτε τις εντολές αυτές με τη σειρά που δίνονται. Καθώς η θεματολογία θα οδηγεί σε πιο πολύπλοκες υλοποιήσεις, ο κώδικας και λοιπό περιεχόμενο των παραδειγμάτων θα δίνεται υπό άλλες μορφές. Π.χ. στην περίπτωση των σεναρίων R (βλ.§3.1 Σενάρια (R-script), θα δίνεται και θα επεξηγείται ολόκληρος ο κώδικας του σεναρίου, όπως για παράδειγμα:

```
Εντολή 1
Εντολή 2
Εντολή 3
```

Στην περίπτωση αυτή, καταλληλότερος τρόπος να καταχωριστούν και να δοκιμαστούν τα παραδείγματα, είναι μέσω κάποιου κατάλληλου βοηθήματος, όπως του επεξεργαστή Source στο RStudio (βλ.§3.1.1 Δημιουργία και εκτέλεση R script).

Με παρόμοιο τρόπο θα παρουσιάζονται και αποτελέσματα που εμφανίζονται στο Console της R μετά την εκτέλεση κάποιου κώδικα. Αν πρόκειται για μια εντολή που μπορεί να δοθεί απευθείας στο Console, θα εμφανίζεται το «σύμβολο προτροπής» (prompt) της R, δηλαδή ο χαρακτήρας '>' (ο οποίος δεν είναι μέρος της εντολής), ακολουθούμενο από την εντολή και το αποτέλεσμα της, όπως π.χ. στο παρακάτω παράδειγμα μιας εντολής που χρησιμοποιεί τη συνάρτηση cat:

```
> cat(1:10)
1 2 3 4 5 6 7 8 9 10
```

Η εντολή που δόθηκε στην R για το παράδειγμα αυτό είναι cat(1:10). Τονίζεται ότι αν θέλετε να δοκιμάσετε το παράδειγμα, δεν γράφετε το prompt '>' (αυτό εμφανίζεται αυτόματα από την R). Η επόμενη μία ή περισσότερες γραμμές είναι το αποτέλεσμα που εμφανίστηκε στο Console μετά την εκτέλεση της εντολής.

Τέλος, αν αντιμετωπίζετε προβλήματα με την εκτέλεση των παραδειγμάτων, αυτό ίσως οφείλεται στις ρυθμίσεις γλώσσας του συστήματός σας. Ειδικά για τις περιπτώσεις παραδειγμάτων που περιέχουν ελληνικούς χαρακτήρες τέτοια προβλήματα επιλύονται συνήθως εκτελώντας την παρακάτω εντολή (βλ. και Παράρτημα Π.1.1 Προβλήματα που σχετίζονται με τα Ελληνικά):

```
Sys.setlocale(category = "LC_ALL", locale = "Greek")
```

Αναφορές Κεφαλαίου 1

[1] R Core Team (2021). R: A Language and Environment for Statistical Computing. Vienna, Austria. https://www.R-project.org/

[2] Crawley, M. J. (2014). *Εισαγωγή στη στατιστική ανάλυση με το R*. Αθήνα: Πασχαλίδης - Broken Hill. ISBN 978-996-371-625-8.

[3] Ανδρουλάκης, Γ., Witte, S. R., Witte, S. J., & Κουνέτας, Κ. (2019). *Στατιστική: Ανάλυση δεδομένων με χρήση της R*. Εκδόσεις Κριτική. ISBN 978-960-586-309-8.

[4] Ιωαννίδης, Δ., & Αθανασιάδης, Ι. (2017). *Στατιστική και μηχανική μάθηση με την R: Θεωρία και εφαρμογές (1η Έκδοση)*. Εκδόσεις Τζιόλα. ISBN 978-960-418-642-6.

[5] Ντζούφρας, Ι., & Καρλής, Δ. (2015). *Εισαγωγή στον προγραμματισμό και στη στατιστική ανάλυση με R* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. ISBN 978-960-603-449-7. https://hdl.handle.net/11419/2601

[6] Κολυβά-Μαχαίρα, Φ., Μπόρα-Σέντα, Ε., & Μπράτσας, Χ. (2018). *Στατιστική (2η Έκδοση)*. Εκδόσεις Ζήτη. ISBN 978-960-456-511-5.

[7] Νικολάου, Χ. (2019). *Ανάλυση Δεδομένων με την R*. Εκδόσεις Δίσιγμα. ISBN 978-618-5242-56-5.

[8] Κουνετάς, Κ., & Χατζησταμούλου, Ν. (2015). *Εισαγωγή στην επιχειρησιακή έρευνα και στον γραμμικό προγραμματισμό. Λύσεις προβλημάτων με το πρόγραμμα R* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. ISBN 978-960-603-301-8. https://hdl.handle.net/11419/5699

[9] Βερύκιος, Β., Καγκλής, Β., & Σταυρόπουλος, Η. (2015). *Η επιστήμη των δεδομένων μέσα από τη γλώσσα R* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. ISBN 978-960-603-394-0. https://hdl.handle.net/11419/2965

[10] Σταυρακούδης, Α. (2012). *Εισαγωγή στις υπολογιστικές μεθόδους για τις οικονομικές και επιχειρησιακές σπουδές.* Αθήνα: Κλειδάριθμος. ISBN 978-960-461-511-7.

[11] Κουτσουπιάς, Ν. Δ. (2018). *Πολυμεταβλητή ανάλυση δεδομένων με τη γλώσσα R*. Θεσσαλονίκη: Πανεπιστήμιο Μακεδονίας.

[12] Φουσκάκης, Δ. (2013). *Ανάλυση Δεδομένων με Χρήση της R*. Εκδόσεις Τσότρας.

[13] Θεμιστοκλέους, Χ. (2017). *Πειραματική μεθοδολογία και στατιστική στη γλωσσολογία, με τη χρήση R*. Εκδοτικός Όμιλος Ίων. ISBN 978-960-508-187-4.

[14] Πετράκος, Γ. (2016). *Εφαρμογές της Θεωρίας πιθανοτήτων με τη χρήση της R*. Εκδόσεις Τσότρας.

[15] Arratia, A. (2014). *Computational Finance*. Atlantis Press. ISBN 978-94-6239-069-0. http://doi.org/10.2991/978-94-6239-070-6

[16] Ang, C. S. (2015). *Analyzing Financial Data and Implementing Financial Models Using R*. Springer. ISBN 978-3-319-14074-2. http://doi.org/10.1007/978-3-319-14075-9

[17] Chambers, J. M. (1998). *Programming with Data*. New York: Springer-Verlag,The Green Book. ISBN 978-0-387-98503-9.

[18] Peng, R. D. (2015). *R Programming for Data Science*. Lean Publishing. https://bookdown.org/rdpeng/rprogdatascience/

[19] The R Foundation, «The R Project for Statistical Computing». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://www.r-project.org/

[20] The RStudio Team, «RStudio». Ηλεκτρονικό. Προσπελάστηκε Φεβρουάριος, 2022, από https://www.rstudio.com/

[21] Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. New York: Springer.

[22] De Vries, A. (2020). MiniCRAN: Create a Mini Version of CRAN Containing Only Selected Packages. https://CRAN.R-project.org/package=miniCRAN

[23] Eddelbuettel, D. (2021). Drat: 'Drat' R Archive Template. https://CRAN.R-project.org/package=drat

[24] Wickham, H., Hester, J., Chang, W., & Bryan, J. (2021). Devtools: Tools to Make Developing R Packages Easier. https://CRAN.R-project.org/package=devtools και https://devtools.r-lib.org/

[25] Fox, J. (2017). *Using the R Commander: A Point-and-Click Interface of R*. Boca Raton FL: Chapman

and Hall/CRC Press.

[26] Muenchen, R. A. (2022, February). R Graphical User Interface Comparison. Ηλεκτρονικό. https://r4stats.com/articles/software-reviews/r-gui-comparison/

[27] Cui, B. (2020). DataExplorer: Automate Data Exploration and Treatment. https://CRAN.R-project.org/package=DataExplorer

[28] Baniecki, H., & Przemyslaw, B. (2019, November). ModelStudio: Interactive Studio with Explanations for ML Predictive Models. *Journal of Open Source Software, τόμ. 4, αρ. 43*, p. 1798.

[29] Baruffa, O. et al., «*Big Book of R*». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://www.bigbookofr.com

[30] R-bloggers, «R-bloggers». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://www.r-bloggers.com/

[31] CodeProject, «The Code Project». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://www.codeproject.com/tags/r

[32] Stack Overflow, «The Stack Overflow». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://stackoverflow.com/questions/tagged/r

Κεφάλαιο 2: Τα βασικά στοιχεία της R

Σύνοψη
Το κεφάλαιο αυτό παρουσιάζει τα πρώτα βασικά στοιχεία της R. Τα θέματα παρουσιάζονται με αυτόνομα μικρά παραδείγματα που μπορούν να δοκιμαστούν απευθείας στη γλώσσα και περιλαμβάνουν χρήση βασικών αριθμητικών και λογικών πράξεων, εισαγωγή στη χρήση μεταβλητών, ειδικές τιμές, βασικούς τύπους και δομές (αριθμητικές, λογικές, κείμενο, σειρές) και σχετικές βασικές συναρτήσεις, εισαγωγή στα διανύσματα κλπ.

Προαπαιτούμενη γνώση
Εξοικείωση με το περιβάλλον της R και του RStudio (Κεφάλαιο 1).

2.1 Χρήση βασικών συναρτήσεων και τελεστών

Ακόμα και αν δεν γνωρίζετε πολλά σχετικά με τον προγραμματισμό υπολογιστών, ίσως έχετε χρησιμοποιήσει προγράμματα υπολογιστικών φύλλων (όπως τα Microsoft Excel, Libreoffice Calc κ.α.). Εκεί πιθανότατα χρειάστηκε να αξιοποιήσετε τις συναρτήσεις που αυτά παρέχουν για υπολογισμούς και άλλες λειτουργίες και έτσι σας είναι οικεία η χρήση τύπων και έτοιμων συναρτήσεων όπως π.χ. το sum που υπολογίζει το άθροισμα ή το log που επιστέφει τον λογάριθμο ενός αριθμού σε κάποια βάση. Ας χρησιμοποιήσουμε λοιπόν οικίες έτοιμες αριθμητικές/μαθηματικές συναρτήσεις[23] για εξάσκηση στη χρήση συναρτήσεων στην R, αλλά και να αναδείξουμε σημεία στα οποία η χρήση συναρτήσεων στην R διαφέρει από αυτές στα προγράμματα υπολογιστικών φύλλων. Εξάλλου στην R, όλες σχεδόν οι λειτουργίες παρέχονται από συναρτήσεις[24] που υπάρχουν σε κάποιο πακέτο και η σύνταξη τους γίνεται με παρόμοιο τρόπο, είτε αφορούν μαθηματικούς υπολογισμούς είτε κάτι άλλο. Στο προγραμματισμό, τη χρήση μιας συνάρτησης την ονομάζουμε και «κλήση» ή «εκτέλεση» της συνάρτησης (άρα την «καλούμε» ή την «εκτελούμε»), ενώ το αποτέλεσμα που προκύπτει από την εκτέλεση της συνάρτησης το ονομάζουμε τιμή που «επιστρέφει» η συνάρτηση.

Στην R, όπως και στα προγράμματα υπολογιστικών φύλλων, οι συναρτήσεις καλούνται με το όνομά τους ακολουθούμενες, μέσα σε παρένθεση, από τις τιμές των παραμέτρων τους (τα ορίσματά τους), τα οποία είναι χωρισμένα με κάποιο διαχωριστικό (που ποικίλει ανάλογα με τις ρυθμίσεις του υπολογιστή, αλλά συνήθως είναι το ερωτηματικό ή το κόμμα). Π.χ. σε προγράμματα όπως το Excel ή το Calc θα γράφατε σε κάποιο «κελί»:

```
=sum(1,2,2,5)
```

ή αν το διαχωριστικό[25] για λίστες είναι το ερωτηματικό, θα γράφατε:

```
=sum(1;2;2;5)
```

για να υπολογιστεί το άθροισμα των αριθμών 1,2,2 και 5, που επιστρέφει ως αποτέλεσμα το 10. Στην R κάνετε κάτι αντίστοιχο, γράφοντας π.χ. στο Console:

Δοκιμάστε:	**Σχόλιο**
`sum(1,2,2,5)`	Επιστρέφει 10, το άθροισμα των αριθμών 1,2,2 και 5.

Παρατηρήστε πως πριν την απάντηση η R αναγράφει την ένδειξη [1], που δείχνει πως αυτό είναι το 1ο (και εδώ μοναδικό) στοιχείο της απάντησης στην εντολή σας. Στην R, τα ορίσματα μίας συνάρτησης είναι πάντα διαχωρισμένα με κόμμα. Το ερωτηματικό (;) χρησιμοποιείται για άλλο σκοπό, δηλαδή, για να δείξει το σημείο όπου τελειώνει κάθε εντολή όταν πολλές εντολές είναι γραμμένες στην ίδια γραμμή. Αντίστοιχα, αν θέλαμε να υπολογίσουμε τον λογάριθμο του 1000 με βάση το 10 (δηλαδή να επιστραφεί το 3, αφού 10^3=1000) σε προγράμματα όπως το Excel ή το Calc θα γράφαμε σε κάποιο «κελί»:

```
=log(1000;10)
```

Στην R κάνετε κάτι αντίστοιχο, γράφοντας στο Console:

Δοκιμάστε:	**Σχόλιο**
`log(1000,10)`	Επιστρέφει 3, τον λογάριθμο του 1000 με βάση το 10.

[23] Οι αριθμητικές/μαθηματικές συναρτήσεις της ενότητας αυτής παρέχονται από το προ-εγκατεστημένο πακέτο ‘base’.
[24] Όπως τονίζει ο Chambers, «οτιδήποτε συμβαίνει στην R είναι αποτέλεσμα κάποιας συνάρτησης» [62].
[25] Το διαχωριστικό επιλέγεται στις ρυθμίσεις του λειτουργικού συστήματος του υπολογιστή σας.

Η σειρά με την οποία δίνονται τα ορίσματα έχει σημασία. Εξ ορισμού, τόσο η συνάρτηση log στα προγράμματα υπολογιστικών φύλλων, όσο και η παρεμφερής και συνώνυμή της συνάρτηση log στην R, θεωρούν πως το πρώτο όρισμα είναι ο αριθμός για τον οποίο θέλουμε να υπολογίσουμε τον λογάριθμο και ο δεύτερος η βάση. Στην R όμως, υπάρχει τρόπος να ορίσουμε συγκεκριμένα την παράμετρο στην οποία θέλουμε να πάει η κάθε τιμή που δίνουμε ως όρισμα. Έτσι, επειδή η συνάρτηση log στην R έχει οριστεί να δέχεται δύο παραμέτρους, η πρώτη με όνομα x (τον αριθμό) και η δεύτερη με όνομα base (η βάση), τα παρακάτω είναι διαφορετικοί αλλά ισοδύναμοι τρόποι να γίνει ο ίδιος υπολογισμός:

Δοκιμάστε:	**Σχόλιο**
`log(1000,10)`	Επιστρέφει 3, τον λογάριθμο του 1000 με βάση το 10.
`log(x=1000,base=10)`	Ίδιο με το προηγούμενο.
`log(base=10,x=1000)`	Ίδιο με το προηγούμενο.
`log(10,x=1000)`	Ίδιο με το προηγούμενο.
`log(1000,base=10)`	Ίδιο με το προηγούμενο.

Η ανάθεση τιμής σε μια συγκεκριμένη παράμετρο (όπως π.χ. το base=10 στο τελευταίο παράδειγμα παραπάνω) γίνεται ανεξαρτήτως σειράς, ενώ αν έχουν μείνει ορίσματα χωρίς συγκεκριμένη παράμετρο (όπως το 1000 στο ίδιο παράδειγμα) θα ανατεθούν στις εναπομένουσες παράμετρους της συνάρτησης με τη σειρά που έχουν οριστεί (εδώ έχει μείνει μόνο μια «ορφανή» παράμετρος, το x και σε αυτό θα ανατεθεί το 1000). Προσοχή, η ανάθεση τιμής σε παραμέτρους πρέπει να γίνεται με τον τελεστή = και όχι με < - ή - >. Αν π.χ. γραφεί log(1000, base<-10) το συντακτικό της R το δέχεται ως εντολή ανάθεσης του 10 σε κάποια μεταβλητή base (και όχι ορισμό τιμής στην παράμετρο base της συνάρτησης log). Τι γίνεται όμως αν κάποια παράμετρος δεν οριστεί;

Δοκιμάστε:	**Σχόλιο**
`log(1000)`	Επιστρέφει 6.907755, το φυσικό λογάριθμο του 1000.

Η R έχει έναν μηχανισμό για χρήση προεπιλεγμένων (default) τιμών στις παράμετρούς που δεν έχουν οριστεί από τον χρήστη. Οι δημιουργοί των συναρτήσεων[26] μπορεί να έχουν επιλέξει να χρησιμοποιήσουν τον μηχανισμό αυτό. Αν ανατρέξετε στην τεκμηρίωση[27] της συνάρτησης log (κάτι που πάντα συνιστάται για όποια νέα συνάρτηση δοκιμάζετε) θα δείτε πως ο ορισμός της log είναι log(x, base = exp(1)). Αυτό σημαίνει πως η log έχει φτιαχτεί έτσι ώστε να απαιτεί πάντα κάποια τιμή για την παράμετρο x (το x δεν έχει προεπιλεγμένη τιμή). Αντίθετα, για την παράμετρο base υπάρχει προεπιλεγμένη τιμή ίση με exp(1). Άρα, αν δεν δοθεί άλλη συγκεκριμένη τιμή από τον χρήστη, η παράμετρος base θα πάρει την τιμή exp(1), όπου το exp είναι μια άλλη συνάρτηση που επιστρέφει τη γνωστή σταθερά ‘e’ (περίπου ίση με 2.718282) υψωμένη σε κάποια δύναμη. Εδώ το exp(1) επιστρέφει το e^1 (άρα το ίδιο το e) όπως μπορείτε να επιβεβαιώσετε γράφοντας exp(1) στο Console. Έτσι, η εντολή log(1000) θα καλέσει τη συνάρτηση log για να υπολογίσει τον λογάριθμο του αριθμού x=1000 με base=2.718282 και έτσι θα επιστρέψει τον επονομαζόμενο φυσικό λογάριθμο του 1000. Στα προγράμματα υπολογιστικών φύλλων (όπως το Excel) υπάρχει διαφορετική συνάρτηση για τον φυσικό λογάριθμο (η ln), ενώ δεν υπάρχει συνάρτηση ln στα πακέτα που συνδέονται αυτόματα με την R. Άρα, αν και στα παραπάνω παραδείγματα τα ονόματα παρεμφερών συναρτήσεων ήταν ίδια τόσο στην R όσο και στα προγράμματα υπολογιστικών φύλλων, αυτό προφανώς δεν είναι απαραίτητο (εξάλλου τα πακέτα της R παρέχουν πολύ μεγαλύτερο αριθμό συναρτήσεων). Τυπικό παράδειγμα είναι η συνάρτηση για τον υπολογισμό του αριθμητικού μέσου όρου που ονομάζεται average στα υπολογιστικά φύλλα (π.χ. στο Excel), ενώ η αντίστοιχη συνάρτηση που είναι ενσωματωμένη στην R ονομάζεται mean και λειτουργεί με διαφορετικό τρόπο[28]. Τέλος, (όπως έχει αναφερθεί και παραπάνω) η R είναι case-sensitive, δηλαδή δεν θεωρεί τα κεφαλαία γράμματα ίδια με τα μικρά στα ονόματα συναρτήσεων κλπ.

Όσο αφορά τις αριθμητικές πράξεις, οι τελεστές είναι σύμβολα (όπως το ‘+’ ή το ‘*’) που έχουν ειδικό νόημα και παρέχουν συνήθως κάποια τέτοια λειτουργία. Στην R είναι και αυτά συναρτήσεις, απλώς επιτρέπουν

---

[26] Αυτός ο μηχανισμός μπορεί να χρησιμοποιηθεί και στις συναρτήσεις που ορίζει ο χρήστης (βλ. §5.1.2 Παράμετροι)

[27] Για την τεκμηρίωση της συνάρτησης log, γράφετε help(log) στο Console.

[28] Η mean περιέχεται στο προ-εγκατεστημένο πακέτο ‘base’. Ο ρόλος της συνάρτησης mean είναι να υπολογίζει τον αριθμητικό μέσο όρο των στοιχείων στο πρώτο όρισμά της, x. Επειδή είναι generic συνάρτηση, ο τύπος του x ορίζει και τη μέθοδο που θα εφαρμοστεί για να γίνει ο υπολογισμός. Για τον τρόπο λειτουργίας των generic συναρτήσεων, βλ. Κεφάλαιο 6 και ειδικότερα §6.3 Κλάσεις S3.

τον κλασικό, οικείο τρόπο γραφής, χωρίς παρενθέσεις, με το σύμβολο του τελεστή τοποθετημένο ανάμεσα (infix) στα ορίσματά του (π.χ. η εντολή 3+4 είναι ταυτόσημη της '+'(3,4)). Οι συναρτήσεις που έχουν οριστεί στην R για αριθμητικές πράξεις είναι ονομασμένες με ήδη γνωστά σύμβολα όπως τα +,-,*,/,^ κλπ. Επειδή δεν είναι υποχρεωτικές οι παρενθέσεις, όταν συνδυάζονται τέτοιες πράξεις υπάρχουν καθορισμένοι κανόνες προτεραιότητας της εκτέλεσης τους (π.χ. οι πολλαπλασιασμοί προηγούνται των προσθέσεων). Δείτε λίστα και πληροφορίες για τις αριθμητικές συναρτήσεις-τελεστές του πακέτου base γράφοντας help(Arithmetic) στο Console, ενώ για να δείτε την προτεραιότητα των τελεστών (αριθμητικών και μη) γράψτε help(Syntax).

Συνδυάζοντας τα παραπάνω ας χρησιμοποιήσουμε την R σαν αριθμομηχανή. Δοκιμάστε στο Console:

Δοκιμάστε:	**Σχόλιο**
`2+3`	Επιστρέφει τον αριθμό 5.
`2+3*4`	Επιστρέφει 14 λόγω προτεραιότητας εκτέλεσης πράξεων (το * προηγείται του +).
`(2+3)*4`	Επιστρέφει 20 (αφού επιβάλαμε προτεραιότητα με παρενθέσεις).
`4^3`	Επιστρέφει το 4^3 δηλαδή 64.
`1.2e5`	Επιστρέφει το 1.2 x 10^5 δηλαδή 120000.
`1.2e+5`	Ίδιο με το παραπάνω.
`4^-3`	Επιστρέφει το 4^{-3} δηλαδή 0.015625 (= 1/4/4/4).
`14%/%5`	Επιστρέφει 2, κάνοντας ακέραια διαίρεση: 14/5 = 2 με υπόλοιπο (modulus) 4.
`14%%5`	Επιστρέφει 4, το υπόλοιπο (modulus) της ακέραιας διαίρεση: 14/5.
`sqrt(9)+10`	Επιστρέφει 13, την τετραγωνική ρίζα του 9 στο οποίο προσθέτει 10.
`sqrt(7+sqrt(81))`	Επιστρέφει 4, την τετραγωνική ρίζα του 16.[29]
`pi`	π ή 3.141593. Το pi είναι μία σταθερά, ορισμένη στο πακέτο ‘base’.
`sin(2*pi/4)`	Επιστρέφει 1, το ημίτονο 2π/4 ακτινίων.
`exp(2)`	Επιστρέφει το e^2.
`base::exp(2)`	Ίδιο με το παραπάνω, καλεί την exp από συγκεκριμένο πακέτο (‘base’).
`"+"(2,3)`	Επιστρέφει 5, ή 2+3. Οι τελεστές είναι συναρτήσεις άρα μπορούν να γράφουν έτσι.
`10-1;3*5`	Επιστρέφει 9 και μετά 15. Το ; χωρίζει εντολές γραμμένες στην ίδια γραμμή.
`2+3.5`	Επιστρέφει 5.5, η τελεία (.) είναι διαχωριστικό δεκαδικών ψηφίων.
`2+3,5`	Λάθος, η R δεν δέχεται το κόμμα ως διαχωριστικό δεκαδικών ψηφίων.

Υπάρχουν περιπτώσεις που οι αριθμητικές πράξεις, αν και ολοκληρώνονται χωρίς κάποιο σφάλμα κατά την εκτέλεση τους, αδυνατούν να επιστρέψουν ένα συγκεκριμένο αριθμητικό αποτέλεσμα. Σε τέτοιες περιπτώσεις συχνά επιστρέφουν κάποια ειδική τιμή, όπως Inf (άπειρο) ή NaN (μη αριθμητικό αποτέλεσμα). Οι ειδικές αυτές τιμές περιγράφονται στην §2.2.4 Ειδικές τιμές.

Σχετικά με την εμφάνιση των αποτελεσμάτων, στην R υπάρχει ένας μηχανισμός μέσω του οποίου ρυθμίζονται διάφορες παράμετροι οι οποίες χρησιμοποιούνται κατά την εκτέλεση εντολών στην τρέχουσα συνεδρία. Η συνάρτηση **options** επιτρέπει την αντικατάσταση των προεπιλεγμένων τιμών των επιλογών (βλ. και §4.2.1.5 Η λίστα επιλογών συνεδρίας). Μία τέτοια επιλογή είναι και η digits, η οποία ρυθμίζει τον αριθμό των χαρακτήρων που θα χρησιμοποιούνται για το δεκαδικό μέρος όταν εμφανίζονται αριθμητικές τιμές:

Δοκιμάστε:	**Σχόλιο**
`sqrt(3)`	Επιστρέφει την τετρ. ρίζα του 3, εμφανίζοντάς τη με προεπιλεγμένο αριθμό ψηφίων.
`options(digits=4)`	Επιλεγεί να εμφανίζονται 4 χαρακτήρες στο δεκαδικό μέρος των αριθμών.
`sqrt(3)`	Επιστρέφει την τετρ. ρίζα του 3, εμφανίζει με 4 χαρακτήρες (3 ψηφία), δηλαδή 1.732.

Ας προχωρήσουμε τώρα σε μια ακόμα πιο βασική διαφορά με τα προγράμματα υπολογιστικών φύλλων. Σε αυτά, οι τιμές που δίνονται ή υπολογίζονται διατηρούνται μέσα στα κελιά των φύλλων. Στην R όμως τίποτα δεν διατηρείται αν δεν υπάρχει σχετική εντολή. Άρα, στα παραπάνω παραδείγματα τα αποτελέσματα των υπολογισμών που επεστράφησαν από τις συναρτήσεις, επειδή δεν υπήρχε εντολή να αποθηκευτούν κάπου, απλά εμφανίστηκαν στο Console (όπως σε μια αριθμομηχανή) και χάθηκαν. Για να διατηρηθεί οτιδήποτε, πρέπει να χρησιμοποιηθεί μία μεταβλητή, κάτι που μας οδηγεί στην επόμενη ενότητα.

---

[29] Παρατηρήστε πως όπως γίνεται και σε άλλες χρήσεις παρενθέσεων, στην περίπτωση που καλούνται συναρτήσεις (εδώ π.χ. η sqrt) τα περιεχόμενα στις εσωτερικές παρενθέσεις απομιμούνται πρώτα. Έτσι πρώτα υπολογίζεται το εσωτερικό sqrt(81) επιστρέφοντας 9, ακολούθως προστίθεται σε αυτό το 7 και έτσι προκύπτει το 16 για το οποίο καλείται η εξωτερική sqrt επιστρέφοντας 4.

2.2 Μεταβλητές, αριθμητικές συναρτήσεις και τύποι αντικειμένων

Οτιδήποτε προκύπτει ως αποτέλεσμα μιας συνάρτησης (άρα οτιδήποτε δημιουργείται στην R) είναι ένα αντικείμενο[30] και υπάρχει μόνο όσο είναι χρήσιμο, π.χ. μόνο όσο εκτελείται μια εντολή που το χρησιμοποιεί. Για να διατηρηθεί στη μνήμη του υπολογιστή ένα αντικείμενο (το αποτέλεσμα ενός υπολογισμού ή οτιδήποτε άλλο) στην R, πρέπει αυτό να καταχωρηθεί (ανατεθεί) σε μια μεταβλητή.

Τι εννοούμε 'αντικείμενο' όμως; Στην R (και στις περισσότερες συνήθεις γλώσσες προγραμματισμού γενικότερα) υπάρχουν (ή και μπορούν να οριστούν) διάφοροι τύποι αντικειμένων. Η γλώσσα δημιουργεί ένα 'αντικείμενο' όταν πρέπει να αποθηκεύσει μια συγκεκριμένη περίπτωση (στιγμιότυπο ή instantiation) ενός τύπου αντικειμένου. Για παράδειγμα, ο αριθμός 4 είναι συγκεκριμένη περίπτωση του αριθμητικού τύπου αντικειμένων[31]. Οι τύποι αντικειμένων, είναι πρότυπα με τα οποία η γλώσσα ξέρει πώς θα χειρίζεται αντικείμενα αυτού του τύπου, πώς θα τα αποθηκεύει στη μνήμη του υπολογιστή και τι υποστηρίζουν. Ο τύπος του αντικειμένου, καθορίζει τι μπορεί να γίνει με αυτό, π.χ. το γεγονός πως στο 4 επιτρέπονται αριθμητικές πράξεις το καθορίζει ο (αριθμητικός) τύπος στον οποίο ανήκει το αντικείμενο αυτό. Στην περίπτωση του αριθμού 4, την επιλογή του τύπου του αντικειμένου που θα δημιουργηθεί για να το αποθηκεύσει, την έκανε η R αυτόματα βασισμένη στο ότι κατά τη λεκτική ανάλυση κάποιας εντολής τα δεδομένα (η σταθερά 4) αναγνωρίστηκε ως αριθμός. Τον τύπο του αντικειμένου τον ονομάζουμε και κλάση του αντικειμένου (object class). Αργότερα θα δούμε άλλους τύπους[32] καθώς και πώς ορίζουμε νέους τύπους, δηλαδή κλάσεις αντικειμένων[33].

Οι μεταβλητές μπορούν να περιέχουν διάφορους τύπους αντικειμένων. Πολλές συνήθεις γλώσσες προγραμματισμού[34] απαιτούν όταν ορίζεται μια μεταβλητή, να έχει οριστεί στον κώδικα και ο τύπος του αντικειμένου που θα αποθηκεύει. Σε αυτή την περίπτωση, οι μεταβλητές προκατασκευάζουν τον χώρο αποθήκευσης του αντικειμένου. Στην R ο χειρισμός των μεταβλητών είναι πιο απλός. Τα αντικείμενα δημιουργούνται δυναμικά (κατά τον χρόνο εκτέλεσης του κώδικα, ως αποτέλεσμα κλήσης συναρτήσεων) και ο τύπος τους προκύπτει κατά τη δημιουργία τους. Τα αντικείμενα επίσης διαγράφονται αυτόματα όταν πάψουν να είναι αναγκαία (μια διαδικασία που ονομάζεται «αποκομιδή απορριμμάτων» (garbage collection). Αν όμως σχετιστούν με ένα όνομα, δηλαδή με μια μεταβλητή, παραμένουν διαθέσιμα όσο υπάρχει η μεταβλητή που τα αφορά[35]. Άρα οι μεταβλητές είναι απλώς ένα όνομα που καταχωρίζεται για κάποιο αντικείμενο που έχει ήδη δημιουργηθεί δυναμικά στη μνήμη. Εφόσον υπάρχει το όνομα (και άρα υπάρχει πιθανότητα μελλοντικής πρόσβασης στο αντικείμενο), το αντικείμενο θεωρείται χρήσιμο και διατηρείται. Σε άλλη περίπτωση το αντικείμενο διαγράφεται.

Έτσι, στην R ο τύπος μιας μεταβλητής δεν χρειάζεται να έχει οριστεί εκ των προτέρων καθώς ουσιαστικά μια μεταβλητή δεν έχει τύπο, το αντικείμενο στο οποίο αναφέρεται έχει. Πρώτα γίνεται η δημιουργία του νέου αντικειμένου (με αποτίμηση του σχετικού κώδικα από όπου προφανώς προκύπτει και ο τύπος του) και ακολούθως το αντικείμενο αυτό ανατίθεται στη μεταβλητή. Η μεταβλητή δημιουργείται τη στιγμή που θα της ανατεθεί κάτι και το μόνο που πρέπει να έχει οριστεί είναι το όνομά της και το (υπάρχον) αντικείμενο στο οποίο αναφέρεται. Ο τύπος της μεταβλητής (ή έστω ό,τι κοντινότερο σε αυτό έχει η R) θα είναι ο τύπος του αντικειμένου που είναι συνδεδεμένο με αυτή, τη συγκεκριμένη στιγμή. Αν ακολούθως στη μεταβλητή γίνει καταχώριση άλλου αντικειμένου διαφορετικού τύπου από αυτόν που ήδη έχει, αυτό πρακτικά αλλάζει και τον τρέχοντα τύπο της μεταβλητής. Για αυτόν τον λόγο, η R είναι μια «dynamically typed» γλώσσα προγραμματισμού[36].

---

[30] O Chambers προσθέτει στο «οτιδήποτε συμβαίνει στην R είναι αποτέλεσμα συνάρτησης» (που αναφέρθηκε παραπάνω) πως «οτιδήποτε υπάρχει στην R είναι αντικείμενο» [62].

[31] Το παράδειγμα είναι απλουστευμένο, βλ. παρακάτω.

[32] βλ. Κεφάλαιο 4.

[33] βλ. Κεφάλαιο 6.

[34] Μεταξύ πολλών άλλων οι C++, C#, Java καθώς και η «Γλώσσα» που διδάσκεται στην Ελληνική Μέση Εκπαίδευση.

[35] Όπως έχει γράψει ο Wickam, συγγραφέας πολλών βιβλίων για την R, στη γλώσσα αυτή «τα ονόματα έχουν αντικείμενα, τα αντικείμενα δεν έχουν όνομα».

[36] Όπως και οι Python, PHP κ.α. Αντίθετα, static type είναι η προσέγγιση που απαιτεί να καθορίζεται στον κώδικα ο συγκεκριμένος τύπος αντικειμένων που μπορεί να αποθηκευτεί σε μια μεταβλητή. Τέτοιες γλώσσες είναι οι C,C++,C#, Java κ.α.

2.2.1 Δημιουργία, χρήση, μετατροπή μεταβλητών και αριθμητικά δεδομένα

Όπως αναφέρθηκε παραπάνω, στην R οι μεταβλητές δημιουργούνται απλώς αναθέτοντας κάποιο αντικείμενο σε αυτές[37]. Οι μεταβλητές της R είναι μόνο ένα όνομα για το αντικείμενο, το οποίο έχει ήδη δημιουργηθεί. Η ανάθεση (ή καταχώριση) γίνεται συνήθως με τον τελεστή **<-** τον οποίο μπορείτε να σκεφτείτε σαν ένα βέλος που στέλνει το αντικείμενο στη μεταβλητή. Για τον ίδιο σκοπό υπάρχουν και άλλοι τελεστές που χρησιμοποιούνται λιγότερο συχνά, όπως το **->** (με αντίθετη κατεύθυνση) αλλά και το **=** το οποίο είναι σχεδόν ισοδύναμο[38] με το <-. Άλλοι τελεστές καταχώρισης τιμής είναι οι **<<-** και **->>** η λειτουργία των οποίων περιγράφεται στην §4.2.2.1 Περιβάλλοντα και συναρτήσεις).

Δοκιμάστε:	**Σχόλιο**
`x <- 10`	Ανάθεση της αριθμητικής τιμής 10 σε μία μεταβλητή με όνομα x.
`10 -> x`	Ίδιο με το παραπάνω.
`x = 10`	Ίδιο με τα παραπάνω (για τις περισσότερες περιπτώσεις)[39].
`2*x+1`	Υπολογίζει και επιστρέφει 21 (εφόσον το x περιέχει το 10).
`y <- 2*x+1`	Υπολογίζει και καταχωρεί το αποτέλεσμα (21) σε μεταβλητή y.
`y <- y+2`	Προσθέτει 2 στο y και καταχωρεί το αποτέλεσμα (23) στο y.
`y`	Επιστρέφει την τιμή της y (23, αν έγιναν τα παραπάνω βήματα).
`X <- 100`	Το X (με κεφαλαίους χαρακτήρες) είναι άλλη μεταβλητή από το x.
`a <- 10; b <- 20`	Ορισμός μεταβλητών a, b (εντολές στην ίδια γραμμή χωρίζονται με ;).
`player <- 100`	Επιτρέπονται περιγραφικά ονόματα μεταβλητών.
`PlayER1 <- 100`	Επιτρέπονται κεφαλαία/μικρά/ψηφία κ.α. σε ονόματα μεταβλητών.
`παίχτης <- 100`	Επιτρέπονται ελληνικοί χαρακτήρες (αλλά δεν συνιστώνται)[40].
`player 1 <- 100`	Λάθος, τα ονόματα μεταβλητών δεν μπορούν να έχουν κενά.
`player@1 <- 100`	Λάθος, εκτός της κάτω παύλας (_), ειδικοί χαρακτήρες δεν επιτρέπονται.
`3 <- 100`	Λάθος, τα ονόματα μεταβλητών δεν ξεκινούν με αριθμητικό ψηφίο.
`1player <- 100`	Λάθος, τα ονόματα μεταβλητών δεν ξεκινούν με αριθμητικό ψηφίο.
`_player <- 100`	Λάθος, τα ονόματα μεταβλητών δεν ξεκινούν "_", παρόλα αυτά…
`'_player' <- 100`	… αν δοθούν τα ονόματα ως κείμενο (δεν ενδείκνυται), γίνονται δεκτά.
`.test <- 100`	Ορισμός κρυφής (hidden) μεταβλητής με όνομα .test και τιμή 100. [41]
`. <- 100`	Ορισμός κρυφής μεταβλητής με όνομα . και τιμή 100.

Παρατηρήστε πως σε όσα παραπάνω βήματα έγιναν υπολογισμοί με ανάθεση σε μεταβλητή (όπως π.χ. στο y <- 2*x+1), δεν εμφανίστηκε τίποτα στο Console (δεν έγινε auto-print). Το αποτέλεσμα ανατέθηκε σε κάποιο όνομα μεταβλητής και για να το εμφανίσουμε πρέπει να την ανακαλέσουμε (π.χ. γράφοντας y) [42]. Οι μεταβλητές πάντως διατηρούν τα περιεχόμενά τους μέχρι αυτά να αντικατασταθούν από άλλα (με κάποια νέα ανάθεση), ενώ διαγράφονται όταν τερματιστεί η συνεδρία με την R. Ένας τρόπος να διατηρηθούν, είναι να αποθηκευτεί η τρέχουσα κατάσταση των μεταβλητών ("workspace") σε ένα αρχείο ώστε να μπορεί στο μέλλον να ανακληθεί από το αρχείο αυτό (βλ.§2.8 Αποθήκευση του User Workspace)

Επίσης, αντίθετα από ότι συμβαίνει στα προγράμματα υπολογιστικών φύλλων, δεν γίνεται κάποιος αυτόματος επαναϋπολογισμός ή αυτόματη ενημέρωση των περιεχομένων των μεταβλητών που σχετίζονται μεταξύ τους:

---

[37] Στην R οι μεταβλητές είναι απλώς ένα ζεύγος (όνομα, αντικείμενο), δηλαδή ένα όνομα που συνδέεται με κάποιο αντικείμενο στη μνήμη. Εσωτερικά τα αντικείμενα είναι τύπου SEXP (δείκτης) ή SEXPREC που υποστηρίζει διάφορους τύπους αντικειμένων, για περισσότερα βλ. [130].

[38] Για διαφορές ανάμεσα στους τελεστές = και <- καθώς και τον ρόλο των τελεστών ανάθεσης τιμής <<- και ->> βλ. help(assignOps) καθώς και §4.2.2.1 Περιβάλλοντα και συναρτήσεις.

[39] βλ. παραπάνω υποσημείωση 38.

[40] βλ. και Παράρτημα Π.1.1 Προβλήματα που σχετίζονται με τα Ελληνικά. Η χρήση ελληνικών μπορεί να μην υποστηρίζεται σωστά σε κάποιους συνδυασμούς ρυθμίσεων λειτουργικού συστήματος ή/και locale της R.

[41] Μεταβλητές με όνομα που ξεκινά από τελεία (.) θεωρούνται κρυφές (hidden), άρα δυσκολότερα εντοπίσημες από τις εντολές. Αυτό προσφέρει κάποια μορφή προστασίας στις μεταβλητές, (από διαγραφές κλπ.). Η χρήση του χαρακτήρα '.' για τον σκοπό αυτό υιοθετήθηκε στην R από τον αντίστοιχο τρόπο επισήμανσης κρυφών αρχείων στο Unix.

[42] Συγκεκριμένα, γράφοντας y ζητάμε να υπολογιστεί το y. Εφόσον αυτό το αποτέλεσμα δεν ανατίθεται κάπου, εμφανίζεται αυτόματα μέσω auto-print.

Δοκιμάστε:	Σχόλιο
x <- 10	Ανάθεση της αριθμητικής τιμής 10 σε μία μεταβλητή με όνομα x.
y <- x	Ανάθεση του (αντικειμένου στο) x, δηλαδή της αριθμητικής τιμής 10, σε μία μεταβλητή y.
x <- 20	Αντικατάσταση του 10 από το 20 στο x. Τι περιέχει το y τώρα; 10 ή 20;
y	Επιστρέφει την τρέχουσα τιμή του y, τελικά παραμένει 10.
y <- x	Για ενημέρωση του y με τη νέα τιμή του x, επαναλαμβάνουμε τη σχετική εντολή.
y	Επιστρέφει την τρέχουσα τιμή του y, δηλαδή 20.

Μία σημείωση: κατά το δεύτερο βήμα στο παραπάνω παράδειγμα (y <- x) η R δημιούργησε εσωτερικά απλώς ένα δεύτερο όνομα (y) για το ίδιο αντικείμενο στο οποίο αναφέρεται και το x (δηλαδή το αριθμητικό αντικείμενο 10). Μετά το βήμα αυτό, τόσο το x όσο και το y είναι ονόματα που αναφέρονται στο ίδιο μοναδικό αντικείμενο στη μνήμη. Όταν όμως γίνει κάποια τροποποίηση στο αντικείμενο μέσω μιας εκ των δύο μεταβλητών, η R δημιουργεί νέο αντικείμενο αντιγράφοντας το αρχικό. Ακολούθως, εφαρμόζει την αλλαγή στο νέο αντικείμενο και το αναθέτει στη μεταβλητή μέσω της οποίας έγινε η αλλαγή. Έτσι, τα αντικείμενα στα οποία αναφέρονται (δίνουν όνομα) οι μεταβλητές είναι πλέον διαφορετικά και οι μεταβλητές λειτουργούν ανεξάρτητα. Αυτή η προσέγγιση ονομάζεται copy-on-modify και μπορεί να επηρεάσει την απόδοση (ταχύτητα εκτέλεσης, απαιτούμενη μνήμη κλπ.) σε απαιτητικά προβλήματα όπως π.χ. ανάλυσης μεγάλων δεδομένων[43]. Για περισσότερα σχετικά με το θέμα βλ.§3.1.4 Εργαλεία προφίλ (profiling) και [33]. Προφανώς σε περιπτώσεις όπου πολλαπλές μεταβλητές αναφέρονται στο ίδιο αντικείμενο στη μνήμη (όπως η αρχική κατάσταση στο 2° παραπάνω βήμα), το αντικείμενο διαγράφεται αυτόματα μόνο όταν δεν υπάρχει πλέον καμία μεταβλητή που να αναφέρεται σε αυτό.

Σε όλα τα παραπάνω παραδείγματα η R θεώρησε πως ο τύπος των δεδομένων είναι αριθμητικός. Αυτός ο τύπος αντικειμένων ονομάζεται numeric, και επιλέγεται αυτόματα για αντικείμενα που θα αποθηκεύουν αριθμούς. Τα αντικείμενα numeric υποστηρίζουν αποθήκευση αριθμών κινητής υποδιαστολής[44]. Πέραν του numeric (που ονομάζεται επίσης και double), η R υποστηρίζει και άλλα είδη αριθμών, με αντίστοιχους τύπους αντικειμένων όπως ακέραιος (integer) και μιγαδικός (complex). Η R κάνει αυτόματα μετατροπές (coercion) σε άλλους τύπους αντικειμένου όταν αυτό είναι απαραίτητο, εφικτό και ασφαλές (χωρίς π.χ. απώλεια ακρίβειας αν πρόκειται για αριθμούς):

Δοκιμάστε:	Σχόλιο
x <- 10L	Ανάθεση της ακέραιας τιμής 10 στο x, το L σημαίνει ακέραιος (long integer).
y <- 20+5i	Ανάθεση της μιγαδικής (complex) τιμής 20+5i στο y.
z <- x+y	Υπολογίζει και καταχωρεί το αποτέλεσμα (αντικείμενο complex με τιμή 30+5i) στο z.

Ένας άλλος τρόπος να δημιουργηθεί ένα αντικείμενο συγκεκριμένου τύπου, ο οποίος μάλιστα μπορεί να χρησιμοποιηθεί για οποιονδήποτε τύπο (κλάση) αντικειμένων, είναι μέσω της συνάρτησης **new**. Πέραν του ονόματος της κλάσης[45], οι πρόσθετες παράμετροι στη συνάρτηση αυτή προσδιορίζονται από τον ίδιο τον τύπο που θα δημιουργηθεί καθώς σχετίζονται με τις ιδιαιτερότητές του. Για παράδειγμα η εντολή:

```
x <- new("integer", 3)
```

δημιουργεί ένα αντικείμενο integer (με τιμή 3) και το αναθέτει στο x. Οι συναρτήσεις[46] της μορφής **as.[τύπος]** όπως και η γενικότερη συνάρτηση **as** επιβάλλουν (ή εξαναγκάζουν, coerce) τη μετατροπή ενός αντικειμένου σε άλλον τύπο (εφόσον η μετατροπή αυτή υποστηρίζεται από τον τύπο). Κατά τη μετατροπή αυτή από έναν τύπο σε άλλον μπορεί να υπάρξει αλλοίωση των δεδομένων. Π.χ. στο αντικείμενο που θα προκύψει από τη μετατροπή ενός numeric σε integer με τη συνάρτηση as.integer θα χαθούν τα δεκαδικά ψηφία που ίσως

[43] Κάποιοι τύποι αντικειμένων προσφέρουν πιθανές λύσεις στο πρόβλημα αυτό, όπως οι τύποι environment (βλ.§4.2.2.2 Περιβάλλοντα και ο μηχανισμός αναφοράς), τα data.table (βλ.§4.3.1 Οι τύποι tibble και data.table) και οι κλάσεις αναφοράς (βλ.§6.5 Κλάσεις αναφοράς).

[44] Floating point, δηλαδή αριθμούς μεταβλητής ακρίβειας με ακέραιο και δεκαδικό μέρος. Οι ψηφιακοί υπολογιστές μας έχουν πεπερασμένες δυνατότητες αποθήκευσης αριθμών και έτσι υπάρχουν περιορισμοί στην ακρίβεια, εύρος τιμών κλπ. των αριθμών που μπορούν να αποθηκευτούν.

[45] Η new προορίζεται για δημιουργία αντικειμένων κάποιας κλάσης, βλ.§6.4 Κλάσεις S4, Οι βασικοί τύποι αντικειμένων δεν έχουν δημιουργηθεί ως κλάσεις, αλλά ανήκουν αυτόματα σε αντίστοιχα formal classes που έχουν δημιουργηθεί για τον σκοπό αυτό, βλ. π.χ. help("integer-class"). Έτσι αντικείμενα βασικών τύπων (π.χ. "integer") ανήκουν σε αντίστοιχη κλάση (επίσης "integer") και αυτό επιστρέφει η συνάρτηση class (αν και δεν υπάρχει ιδιότητα class στο αντικείμενο).

[46] Οι συναρτήσεις αυτής της ενότητας παρέχονται από τα προ-εγκατεστημένα πακέτα 'base' και 'utils' (για τη str).

υπάρχουν στα δεδομένα. Υπάρχει πληθώρα τέτοιων συναρτήσεων, όχι μόνο για αριθμητικούς τύπους (π.χ. as.numeric ή as.double, as.integer, as.complex, as.logical, as.character, as.raw, as.vector, as.array, as.matrix, as.data.frame).

Αντίστοιχα, συναρτήσεις της μορφής **is.[τύπος]** και η γενικότερη συνάρτηση **is** ελέγχουν αν ένα αντικείμενο ανήκει σε κάποιον συγκεκριμένο τύπο[47]: Ο έλεγχος αυτός αφορά τον τύπο του αντικειμένου και όχι το περιεχόμενό του. Για παράδειγμα, η συνάρτηση is.integer δεν ελέγχει αν τα δεδομένα είναι ακέραιοι αριθμοί αλλά αν το αντικείμενο είναι τύπου integer. Επιπρόσθετα, υπάρχει ιεραρχία στους τύπους. Έτσι, π.χ. τα αντικείμενα τύπου integer είναι numeric[48] αλλά όχι το αντίθετο.

Δοκιμάστε:	**Σχόλιο**
`x<-as(3.9,"integer")`	Μετατροπή του 3.9 σε ακέραιο (κρατά το 3) και ανάθεση στο x.
`x<-as.integer(3.9)`	Ίδιο με το παραπάνω.
`x<-as.complex(x)`	Μετατροπή του αντικειμένου με όνομα x σε μιγαδικό (3+0i).
`x<-as.numeric(x)`	Μετατροπή του αντικειμένου με όνομα του x σε numeric (3).
`x<-as.double(x)`	Ίδιο με το παραπάνω, το double είναι συνώνυμο του numeric.
`is(x,"numeric")`	Είναι το x τύπου numeric; Επιστρέφει αληθές (TRUE).
`is.numeric(x)`	Ίδιο με το παραπάνω.
`is.numeric(10)`	Είναι το 10 numeric; Επιστρέφει αληθές (TRUE).
`is.integer(10)`	Είναι το 10 ακέραιος; Αναληθές (FALSE), είναι numeric[49].
`is.complex(10)`	Είναι το 10 τύπου μιγαδικός; Επιστρέφει αναληθές (FALSE).
`is.integer(as.integer(10))`	Αληθές (TRUE) η μετατροπή του 10 σε ακέραιο είναι ακέραιος.

Άλλες σχετικές συναρτήσεις για την εξέταση μεταβλητών (οποιουδήποτε τύπου) είναι οι **isa**, **class**, **mode**, **typeof**, **length** και **str** (συντόμευση της λέξης δομή/structure):

Δοκιμάστε:	**Σχόλιο**
`x<-3.9`	Ορισμός ονόματος x για την τιμή 3.9.
`isa(x,"integer")`	Είναι το x τύπου ακέραιος; Αναληθές (FALSE), είναι numeric.
`class(x)`	Η κλάση (τύπος) του αντικειμένου με όνομα x, εδώ numeric.
`mode(x)`	Το τρέχον mode (κατάσταση αποθήκευσης) του x, εδώ numeric.[50]
`typeof(x)`	Εσωτερικός τύπος[51], εδώ κινητής υποδιαστολής διπλής ακρίβειας (double).
`length(x)`	Μήκος (αριθμός περιεχομένων) του x, εδώ 1 αφού είναι μόνο ένας αριθμός.
`str(x)`	Σύνοψη της δομής του αντικειμένου με όνομα x (εδώ επιστρέφει num 3.9)[52].

Το γεγονός πως οι συναρτήσεις class και mode στο παραπάνω παράδειγμα επέστρεψαν το ίδιο αποτέλεσμα ("numeric") οφείλεται στο ότι το x αναφέρεται σε αντικείμενο κάποιου βασικού τύπου δεδομένων. Ένα αντικείμενο μπορεί να βρίσκεται σε ένα μόνο mode (τα διάφορα διαθέσιμα mode είναι αλληλοαποκλειστικά) και το mode σχετίζεται με τον τρόπο που είναι εκείνη τη στιγμή αποθηκευμένο. Για τα βασικά δεδομένα (όπως numeric, integer, complex και άλλους τύπους που θα δούμε αργότερα στο κεφάλαιο αυτό) το αρχικό mode του αντικειμένου που θα τα αποθηκεύσει προέρχεται από τα ίδια τα δεδομένα (π.χ. το 3.9 είναι αριθμός και αυτό οδηγεί σε αντικείμενο που βρίσκεται σε "numeric" mode). Τέτοιοι βασικοί τύποι δημιουργούν αυτόματα και ομώνυμης κλάσης αντικείμενα (κάτι που επιβεβαίωσε η κλήση της class[53]), αλλά ένα αντικείμενο μπορεί να ανήκει σε περισσότερες από μία κλάσεις[54].

---

[47] βλ. και υποσημείωση 45. Οι γενικότερες συναρτήσεις is, extends και as παρέχονται από το πακέτο 'methods' και αφορούν κλάσεις S4 (βλ.§6.4 Κλάσεις S4). Η as αφορά μετατροπή αντικειμένου σε άλλο τύπο, η is ελέγχει αν ένα αντικείμενο είναι κάποιου συγκεκριμένου τύπου, ενώ η extends ελέγχει αν ένας τύπος είναι επέκταση άλλου, βλ. σχετική τεκμηρίωση π.χ. help(is).

[48] Ο τύπος integer είναι επέκταση του τύπου numeric, κάτι που επιβεβαιώνει η εντολή extends("integer","numeric").

[49] Ο έλεγχος δεν αφορά αν η αριθμητική τιμή είναι ακέραια αλλά αν ο τρέχον τύπος του αντικειμένου είναι integer.

[50] Για περισσότερα σχετικά με τη σημασία του mode, βλ.§4.1 Συνήθεις ατομικοί τύποι αντικειμένων.

[51] Για τον μέσο χρήστη της R η συνάρτηση typeof έχει περισσότερο ενημερωτική παρά χρηστική αξία.

[52] βλ.§2.7 Οι συναρτήσεις-βοηθήματα: str, summary και dput.

[53] βλ. και υποσημείωση 45.

[54] βλ. Για τις κλάσεις και την κληρονομικότητα βλ. Κεφάλαιο 6.

Συναρτήσεις όπως οι mode, class και length δεν επιστρέφουν απλώς ένα αποτέλεσμα (εδώ την κλάση ή το μήκος του αντικειμένου, αντίστοιχα) αλλά επιτρέπουν και την αλλαγή στις σχετικές ιδιότητες του αντικειμένου[55]. Έτσι:

Δοκιμάστε:	**Σχόλιο**
`x<-3.9`	Ορισμός ονόματος x για την τιμή 3.9.
`class(x)<-"complex"`	Αλλαγή της κλάσης του x σε μιγαδικό (ίδιο με x<-as.complex(x)).
`x`	Επιστρέφει την τιμή του x (δηλαδή 3.9+0i).
`class(x)<-"integer"`	Αλλαγή της κλάσης του x σε ακέραιο (ίδιο με x<-as.integer(x)).
`x`	Επιστρέφει την τιμή του x (δηλαδή 3).
`length(x)<-5`	Αλλαγή του μήκους (αριθμού στοιχείων) του x σε 5.
`x`	Επιστρέφει την τιμή του x (3 NA NA NA NA)

Τα τελευταία δύο βήματα στο παραπάνω παράδειγμα (τα βήματα που αφορούν την αλλαγή του μήκους του x σε 5) δίνουν μια πρόγευση από τα διανύσματα, τον πλέον βασικό τύπο αντικειμένων με τον οποίο θα ασχοληθούμε αργότερα. Όταν αποθηκεύει βασικά είδη δεδομένων (αριθμούς, κείμενο, λογικές τιμές κλπ.) η R χειρίζεται με τον ίδιο ακριβώς τρόπο, τόσο την περίπτωση ενός μεμονωμένου στοιχείου, όσο και την περίπτωση πολλών. Και στις δύο περιπτώσεις, τα τοποθετεί σε ένα διάνυσμα. Έτσι, ακόμα και μια μεμονωμένη αριθμητική τιμή για την R είναι ένα διάνυσμα που απλά περιέχει ένα μόνο στοιχείο (περισσότερα για τα διανύσματα αναφέρονται στην §2.4 Εισαγωγή στα διανύσματα). Η παραπάνω εντολή length(x)<-5 ζητά να μετατραπεί το x σε αντικείμενο 5 στοιχείων, επεκτείνοντας το μονομελές numeric διάνυσμα σε ένα 5 στοιχείων, όπου το πρώτο στοιχείο είναι το τρέχον x, τα υπόλοιπα τέσσερα είναι προφανώς άγνωστα και για αυτόν τον λόγο έχουν πάρει την ειδική τιμή NA (Not Available). Άλλες ειδικές τιμές που μπορεί να προκύψουν ως αποτέλεσμα αριθμητικών πράξεων ή άλλων συναρτήσεων περιγράφονται στην §2.2.4 Ειδικές τιμές.

2.2.2 Το καθολικό περιβάλλον (Global Environment)

Οι μεταβλητές τοποθετούνται μέσα σε περιβάλλοντα. Αυτό γίνεται για λόγους οργάνωσης και περιορισμού της εμβέλειας (scope) τους. Το «καθολικό περιβάλλον» (Global Environment) είναι το πλαίσιο μέσα στο οποίο εξ ορισμού καταχωρούνται μεταβλητές από τον χρήστη κατά την τρέχουσα συνεδρία (session). Ακριβέστερα, είναι το περιβάλλον στο οποίο τοποθετούνται τα ονόματα για αντικείμενα που δημιουργεί ο κώδικας του χρήστη εφόσον εργάζονται στο κορυφαίο επίπεδο (top level) και όχι π.χ. μέσα σε κάποια συνάρτησή του (περισσότερα για αυτό, βλ.§4.2.2.1 Περιβάλλοντα και συναρτήσεις). Πάντως, αρκεί να θυμάστε πως το επίπεδο με το οποίο συνήθως αλληλοεπιδράτε εσείς (και οι κώδικες σας) είναι το Global Environment και εκεί εξ ορισμού τοποθετούνται οι μεταβλητές με τα αντικείμενα σας. Για αυτόν τον λόγο το Global Environment αναφέρεται και ως «χώρος εργασίας του χρήστη» (user workspace)[56]. Όλες οι μεταβλητές στα παραδείγματα της προηγουμένης ενότητας δημιουργήθηκαν μέσα στο Global Environment.

Οι μεταβλητές (άρα και τα αντικείμενα στα οποία αναφέρονται) που ορίζονται στο Global Environment είναι καθολικές. Έχουν καθολική εμβέλεια (scope) και είναι προσπελάσιμες από οποιονδήποτε κώδικα εκτελείται. Χρήσιμες συναρτήσεις[57] για τη δημιουργία, χειρισμό και διαγραφή μεταβλητών μέσα στο Global Environment είναι οι **assign**, **ls** (και **objects**), **ls.str**, **object.size** και **rm**. Για παράδειγμα:

Δοκιμάστε:	**Σχόλιο**
`assign("a",10)`	Ορισμός μεταβλητής a με τιμή 10 (ίδιο με a<-10).
`ls()`	Επιστρέφει τα ονόματα όλων των μεταβλητών (εκτός των κρυφών)[58].
`ls(all.names = T)`	Επιστρέφει τα ονόματα όλων των μεταβλητών (και των κρυφών).
`objects()`	Ίδιο με το παραπάνω.
`ls.str()`	Επιστρέφει τα ονόματα και τη δομή όλων των μεταβλητών (όπως η str).
`object.size(a)`	Επιστρέφει τον χώρο που καταλαμβάνει το a στη μνήμη (κατά προσέγγιση).
`rm(a)`	Διαγράφει τη μεταβλητή a.
`rm(list=ls())`	Διαγράφει όλες τις (μη-κρυφές) μεταβλητές από το Global Environment.

[55] βλ.§4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων.
[56] βλ. και help(environment).
[57] Παρέχονται από τα προ-εγκατεστημένα πακέτα ‘base’ και ‘utils’.
[58] Μεταβλητές με όνομα που ξεκινά από τελεία (.) είναι κρυφές, βλ. υποσημείωση 41.

Οι παραπάνω εντολές εφαρμόστηκαν στις μεταβλητές του Global Environment, αλλά παρέχουν παράμετρο με την οποία μπορεί να οριστεί άλλο περιβάλλον και να εφαρμοστούν σε αυτό. Τέλος, αν χρειάζεται, μπορείτε να αναφερθείτε στο Global Environment με το όνομά του .GlobalEnv ή μέσω της συνάρτησης **globalenv**.

2.2.3 Ο ρόλος των περιβαλλόντων

Ο ρόλος του καθολικού περιβάλλοντος (Global Environment) και όσα σχετικά γράφτηκαν στην προηγουμένη παράγραφο αρκούν για την τυπική χρήση της R. Οι περισσότεροι χρήστες δεν θα χρειαστούν να γνωρίζουν κάτι περισσότερο σχετικά με τα περιβάλλοντα για να ξεκινήσουν να αξιοποιούν τη γλώσσα. Η ενότητα αυτή δίνει λίγα πρόσθετα στοιχεία για τα περιβάλλοντα, τον ρόλο και τον τρόπο λειτουργίας τους, στοιχεία που είναι επίσης χρήσιμο να γνωρίζουν αν σκοπεύουν να εμβαθύνουν στη γλώσσα.

Τα περιβάλλοντα ορίζουν την εμβέλεια των αντικειμένων και έτσι θέτουν περιορισμούς σε αυτή. Στην R, τα περιβάλλοντα είναι και αυτά αντικείμενα (όπως είναι τα πάντα στη γλώσσα αυτή) και κάθε περιβάλλον είναι ουσιαστικά μια ειδικού τύπου λίστα[59] που ονομάζεται «πλαίσιο» (frame) και περιέχει τις μεταβλητές που έχουν δημιουργηθεί μέσα σε αυτό. Έτσι, οι μεταβλητές αυτές είναι περιορισμένες (κατά κάποιο τρόπο, κρυμμένες) μέσα στο περιβάλλον που ανήκουν. Για να βρεθούν πρέπει να γίνει πρόσβαση στη λίστα και να αναζητηθούν μέσα σε αυτή. Αυτό, κατ' αρχάς επιτρέπει την οργάνωση των μεταβλητών. Αν δεν υπήρχαν τα περιβάλλοντα, όλα τα ονόματα θα ήταν ορισμένα σε ένα μοναδικό ενιαίο χώρο, δημιουργώντας προβλήματα με συνωνυμίες μεταβλητών, ασάφειες στον προσδιορισμό της μεταβλητής, αδυναμία δημιουργίας μεταβλητών με ονόματα που ήδη υπάρχουν κ.α. Τα περιβάλλοντα επιτρέπουν να υπάρχει τάξη στις μεταβλητές (και κατ' επέκταση στα αντικείμενα στα οποία αναφέρονται). Για τον λόγο αυτό κάθε πακέτο που συνδέεται, δημιουργεί ένα δικό του περιβάλλον για τα αντικείμενα που περιέχει και έναν χώρο ονομάτων (namespace) αντικειμένων από το περιβάλλον αυτό που εξάγονται, άρα είναι προσβάσιμα από τους χρήστες του πακέτου και τον κώδικά τους[60].

Όμως αν ισχύουν τα παραπάνω, πως μπορέσαμε να χρησιμοποιήσουμε αντικείμενα (π.χ. συναρτήσεις) από τα πακέτα χωρίς να χρειαστεί κάποιος ιδιαίτερος κώδικας; Πως εκτελέσαμε π.χ. παραπάνω τη συνάρτηση rm ή άλλες συναρτήσεις του πακέτου 'base'; Δεν είναι κρυμμένες μέσα στο περιβάλλον του πακέτου αυτού; Είναι, αλλά υπάρχουν μηχανισμοί που επιτρέπουν τελικά την έξωθεν πρόσβαση στα αντικείμενα ενός περιβάλλοντος. Ένας τρόπος είναι να προσδιοριστεί επακριβώς το namespace (αν πρόκειται για κάποιο πακέτο) και μετά το αντικείμενο. Αυτό γίνεται με τον τελεστή ::, γράφοντας π.χ. base::rm αντί απλώς rm[61]. Άλλος τρόπος είναι να αφήσουμε την R να αναζητήσει το αντικείμενο. Απλουστεύοντας αρκετά την περιγραφή της διαδικασίας αυτής, υπάρχει μια ιεραρχία αναζήτησης αντικειμένων στα περιβάλλοντα. Όταν π.χ. συνδέεται ένα πακέτο (με τις συναρτήσεις library ή require, βλ.§1.5.1 Χρήση πακέτων) και δημιουργείται το περιβάλλον του, ορίζεται αυτόματα ένας πρόγονος του, που ονομάζεται γονικό περιβάλλον (parent environment) ή περιέχον περιβάλλον (enclosing environment) και χρησιμοποιείται για τη συνέχιση της αναζήτησης αντικειμένων όταν αυτά δεν υπάρχουν στο τρέχον. Το γονικό περιβάλλον ορίζεται δυναμικά (συχνά αυτόματα από την R) και μπορεί να αλλάζει, αλλά είναι μόνο ένα για κάθε περιβάλλον. Όταν εκτελείται μια εντολή με κάποιο όνομα αντικειμένου ή κάποιος κώδικας σε κάποιο πακέτο χρειάζεται ένα αντικείμενο, ξεκινά μια διαδικασία αναζήτησης του αντικειμένου αυτού. Η αναζήτηση ξεκινά από το περιέχον περιβάλλον της εντολής που εκτελείται (συνήθως το Global Environment) και εφόσον το όνομα του αντικειμένου δεν βρεθεί σε αυτό, συνεχίζεται στο αμέσως παρακάτω (γονικό) περιβάλλον του και με τον ίδιο τρόπο στα επόμενα μέχρι να εντοπιστεί. Αυτό δημιουργεί μια ιεράρχηση της σειράς με την οποία η R αναζητεί αντικείμενα μέσα στα διάφορα περιβάλλοντα. Το κορυφαίο επίπεδο (top level) της ιεραρχίας αυτής είναι συνήθως το Global Environment. Αμέσως παρακάτω (γονικό περιβάλλον του και επόμενο στην αναζήτηση αντικειμένων) είναι το τελευταίο περιβάλλον που προστέθηκε από την R στη σειρά αναζήτησης. Αυτό μπορεί να σχετίζεται με το τελευταίο πακέτο που συνδέθηκε στην τρέχουσα συνεδρία της R (με τη συνάρτηση library) ή με κάποιο αντικείμενο που προστέθηκε από τον χρήστη[62]. Ακολουθούν άλλα αντικείμενα (συνήθως περιβάλλοντα

---

[59] Τα environment είναι παρεμφερή με τον τύπο list που περιγράφεται στην §4.2.1 Ο τύπος list (λίστα).

[60] Για τον τρόπο που ορίζεται αυτό κατά τη δημιουργία των πακέτων, βλ.§8.5.1.2 Αρχείο NAMESPACE.

[61] Ο τελεστής :: μπορεί να χρησιμοποιηθεί ακόμα και αν δεν έχει συνδεθεί το πακέτο, αφού (αν χρειάζεται) το φορτώνει. Όμως δεν το συνδέει, άρα δεν ενεργοποιεί την αναζήτηση σε αυτό.

[62] Με τη συνάρτηση attach μπορούν να προστεθούν αντικείμενα τύπου list, data.frame και environment.

πακέτων) που είχαν προστεθεί στη σειρά αναζήτησης πριν από αυτό. Στο κατώτερο επίπεδο βρίσκεται πάντα το περιβάλλον του πακέτου ‘base’, όπου σταματά και η αναζήτηση[63]. Αν το αντικείμενο δεν βρεθεί ούτε εδώ, υπάρχει λάθος και εμφανίζεται το σχετικό μήνυμα. Κατά την αναζήτηση, αντικείμενα σε ανώτερα επίπεδα μπορεί να έχουν ίδιο όνομα με άλλα αντικείμενα σε κατώτερα επίπεδα και έτσι να τα κρύβουν (mask) από την αναζήτηση. Όταν προστίθενται νέα αντικείμενα στη σειρά αναζήτησης (π.χ. συνδέεται ένα νέο πακέτο) και συμβαίνει να προκαλούν κάποια τέτοια συνωνυμία, η R προειδοποιεί εμφανίζοντας σχετικό μήνυμα.

Άρα από την πλευρά του χρήστη της R, η αναζήτηση ξεκινά στο το Global Environment και προχωρά προς τα κατώτερα επίπεδα, κάτι που μπορείτε να το επιβεβαιώσετε εκτελώντας τη συνάρτηση search. Μερικές άλλες σχετικές συναρτήσεις[64] είναι οι **environment**, **parent.env**, και **globalenv**, ενώ η as.environment επιτρέπει τη μετατροπή άλλων αντικειμένων σε αντικείμενα τύπου «περιβάλλον». Μπορείτε να δοκιμάσετε τα παρακάτω:

Δοκιμάστε:	**Σχόλιο**
`search()`	Σειρά αναζήτησης αντικειμένων.
`environment()`	Το τρέχον περιβάλλον στο οποίο γίνεται η επεξεργασία.
`environment(ls)`	Το περιβάλλον συσχετισμένο με αντικείμενο (συνάρτηση) ls.
`parent.env(globalenv())`	Το γονικό περιβάλλον του Global.
`as.environment("package:base")`	Το περιβάλλον συνδεδεμένου πακέτου ("package:base").
`ls(as.environment("package:base"))`	Όλα τα ονόματα αντικειμένων στο συγκεκριμένο περιβάλλον.
`ls(pos="package:base")`	Ίδιο με το παραπάνω[65].

Στο επόμενο παράδειγμα, όλες οι εντολές καλούν τη συνάρτηση sqrt του ‘base’ για να υπολογίσουν την τετραγωνική ρίζα του 9, αλλά εντοπίζουν τη συνάρτηση αυτή με διαφορετικό τρόπο. Προφανώς, παρόμοιες προσεγγίσεις μπορούν να χρησιμοποιηθούν για αντικείμενα και από άλλα περιβάλλοντα πακέτων (όχι μόνο του ‘base’). Εκτός του πρώτου παραδείγματος, στα παρακάτω υπάρχει πλεονασμός στη γραφή και δίνονται μόνο για καλύτερη κατανόηση του τρόπου σύνταξης των σχετικών εντολών:

Δοκιμάστε:	**Σχόλιο**
`sqrt(9)`	Η συνάρτηση εντοπίζεται με αναζήτηση[66].
`base::sqrt(9)`	Καλεί την sqrt στον χώρο ονομάτων ‘base’.
`as.environment("package:base")$sqrt(9)`	Πρόσβαση στο περιβάλλον συνδεδεμένου ‘base’.
`as.environment("package:base")[["sqrt"]](9)`	Πρόσβαση στο περιβάλλον συνδεδεμένου ‘base’

Όπως έχει ήδη αναφερθεί, αν εργάζεστε στο RStudio, η καρτέλα Environment μπορεί να εμφανίσει τα περιεχόμενα ενός περιβάλλοντος, είτε πρόκειται για το Global Environment είτε για άλλα περιβάλλοντα που έχουν προστεθεί στη σειρά αναζήτησης από τα πακέτα ή από τον χρήστη (βλ.§1.4.2.7 Η καρτέλα Environment). Για περισσότερα, σχετικά με τη δημιουργία και τον χειρισμό αντικειμένων τύπου «περιβάλλον» καθώς και για την πρόσβαση σε αντικειμένα μέσα σε αυτά, βλ.§4.2.2 Ο τύπος environment (περιβάλλον).

2.2.4 Ειδικές τιμές

Ειδικές τιμές που έχουν οριστεί στην R (πακέτο ‘base’) και χρησιμοποιούνται συχνά είναι οι NULL, NA, Inf και NaN. Όλες μπορούν να γραφούν ως μέρος μιας εντολής, να ανατεθούν σε μεταβλητές, να επιστρέφονται από συναρτήσεις κλπ. Το **NULL** συμβολίζει το «τίποτα», είναι δεσμευμένη λέξη και δεν θεωρείται αριθμός αλλά ένα «κενό αντικείμενο»[67]. Το **NA** (Not Available) συμβολίζει κάτι που δεν είναι διαθέσιμο ή δεν είναι γνωστό και είναι ειδική τιμή τύπου logical (βλ.§2.5 Λογικές πράξεις, συγκρίσεις και ο βασικός τύπος logical). Ο ρόλος του είναι να συμβολίζει κάποιο δεδομένο που λείπει (βλ.§2.4.2 Δύο παραδείγματα χρήσης vector και

[63] Υπάρχει ένας εικονικός πρόγονος του base, το «κενό περιβάλλον» όπου σταματά η αναζήτηση, βλ. help(environment).
[64] Παρέχονται από το προ-εγκατεστημένο πακέτο ‘base’.
[65] Για την εμφάνιση και των κρυφών μεταβλητών (που το όνομά τους ξεκινά με τελεία) στην ls υπάρχει η παράμετρος all.names, άρα για το παράδειγμα αυτό θα γράφαμε ls(pos="package:base", all.names=T). Επιπρόσθετα, η τιμή που είναι αποθηκευμένη στη ρύθμιση max.print θέτει όριο σε πόσα ονόματα θα εμφανιστούν και μπορεί να αλλάξει π.χ. η εντολή options(max.print=2000) αλλάζει το όριο σε 2000 (βλ.§4.2.1.5 Η λίστα επιλογών συνεδρίας).
[66] Τελικά εντοπίζεται στο πακέτο ‘base’.
[67] Σε αντίθεση με άλλες ειδικές τιμές, το mode του NULL είναι NULL.

η σημασία της ειδικής τιμής NA)[68]. Οι ειδικές τιμές **Inf** (Infinity/άπειρο) και **NaN** (Not-A-Number/μη αριθμός) είναι ειδικοί αριθμοί (τύπου numeric), άρα μπορούν να προκύψουν και ως αποτέλεσμα αριθμητικών πράξεων ή συναρτήσεων:

Δοκιμάστε:	**Σχόλιο**
`3/0`	Διαίρεση με 0, επιστρέφει Inf, την ειδική τιμή που σημαίνει «άπειρο» (infinity).
`sqrt(-4)`	Επιστρέφει NaN την ειδική τιμή που σημαίνει «όχι αριθμός» (Not a Number).
`0/0`	Η διαίρεση του μηδέν με μηδέν δεν είναι αριθμός, επίσης επιστρέφει NaN.
`Inf-1`	Επιστρέφει Inf (πράξεις με άπειρο επιστρέφουν άπειρο).
`NaN-1`	Επιστρέφει NaN (πράξεις με μη αριθμούς επιστρέφουν μη αριθμούς).

Έτσι το Inf είναι μια ειδική τιμή που συμβολίζει το «άπειρο»[69], ενώ το NaN είναι μια ειδική τιμή ένδειξης πως το αποτέλεσμα δεν είναι αριθμός (Not a Number). Και τα δύο είναι τύπου numeric, άρα μπορούν να γίνουν πάνω τους αριθμητικές πράξεις χωρίς να σταματήσει η εκτέλεση του κώδικα. Όμως ο χειρισμός τους από την R γίνεται με ειδικό τρόπο, π.χ.:

- οι αριθμητικές πράξεις με NaN επιστρέφουν NaN.
- οι αριθμητικές συγκρίσεις (όπως >, >= κλπ.[70]) με NaN επιστρέφουν NA.
- οι συνήθεις αριθμητικές πράξεις με Inf επιστρέφουν Inf, εκτός από ειδικές περιπτώσεις διαίρεσης και αφαίρεσης.
- διαίρεση με Inf επιστρέφει 0 εκτός αν ο διαιρετέος είναι Inf οπότε επιστρέφει NaN.
- αφαίρεση Inf από Inf επιστρέφει NaN.
- σε συγκρίσεις, το Inf είναι μεγαλύτερο από κάθε άλλη αριθμητική τιμή.
- σε συγκρίσεις, το -Inf είναι μικρότερο από κάθε άλλη αριθμητική τιμή.

Ο έλεγχος για τέτοιες ειδικές τιμές γίνεται με τις συναρτήσεις **is.na**, **is.nan**, **is.infinite** και **is.finite**. Για παράδειγμα:

Δοκιμάστε:	**Σχόλιο**
`is.infinite(3/0)`	Είναι το αποτέλεσμα του 3 δια 0 άπειρο; (επιστρέφει αληθές, TRUE).
`is.finite(3/0)`	Είναι το αποτέλεσμα του 3 δια 0 πεπερασμένος αριθμός; (επιστρέφει αναληθές, FALSE).
`is.finite(NA)`	Είναι πεπερασμένος αριθμός το NA; (επιστρέφει αναληθές, FALSE).
`is.nan(3/0)`	Είναι NaN το 3 δια 0; (επιστρέφει αναληθές, FALSE).
`is.nan(0/0)`	Είναι NaN το 0 δια 0; (επιστρέφει αληθές, TRUE).
`is.na(0/0)`	Είναι NA ένα NaN. Τα NaN θεωρούνται και NA (επιστρέφει αληθές, TRUE).

2.3 Συναρτήσεις κειμένου και ο βασικός τύπος character

Η R θεωρεί ως κείμενο οτιδήποτε βρίσκεται μέσα σε εισαγωγικά. Κάθε κείμενο (δηλαδή λεκτικό, ακολουθία χαρακτήρων, συμβολοσειρά) πρέπει να γράφεται είτε μέσα σε μονά (') είτε μέσα σε διπλά εισαγωγικά ("). Συνιστάται η χρήση διπλών εισαγωγικών[71], αφενός γιατί μέσα σε διπλά εισαγωγικά εμφανίζει η ίδια η R τα λεκτικά αποτελέσματα της, αφετέρου επειδή διπλά εισαγωγικά χρησιμοποιούν για ορισμό κειμένου και πολλές άλλες γλώσσες προγραμματισμού ή και προγράμματα (όπως τα υπολογιστικά φύλλα). Προσοχή όμως: τα μονά εισαγωγικά καταχωρούνται πιέζοντας το πλήκτρο με τα σύμβολα ' και " (αριστερά του Enter), ενώ τα διπλά εισαγωγικά καταχωρούνται πατώντας Shift - ('), δηλαδή έχοντας πατημένο το Shift και το ίδιο πλήκτρο. Το να πληκτρολογήσετε δύο φορές μονά εισαγωγικά δεν καταχωρίζεται ως διπλά εισαγωγικά. Ο τύπος του κειμένου (ή καλύτερα, των αντικειμένων που αποθηκεύουν κείμενο) ονομάζεται character:

---

[68] Αν και το NA είναι logical, μετατρέπεται σε άλλους τύπους αν χρειάζεται. Π.χ. σε ένα διάνυσμα που περιέχει numeric τιμές NA είναι τύπου numeric αφού το δεδομένο που απουσιάζει είναι αριθμός. Συγκεκριμένα υπάρχουν πολλές εσωτερικές «παραλλαγές» ειδικών τιμών για το NA όπως NA_integer_, NA_real_, NA_complex_ και NA_character_ (βλ. help(reserved).

[69] Δεν υπάρχει «άπειρο» στις πεπερασμένες ψηφιακές μας συσκευές. Όμως κάποιες πράξεις εξ ορισμού επιστέφουν «άπειρο» (όπως η διαίρεση μιας μη μηδενικής τιμής με το 0) και εκεί η R χρησιμοποιεί την ειδική αριθμητική τιμή Inf.

[70] Για τους τελεστές σύγκρισης βλ.§2.5.1 Συναρτήσεις σύγκρισης.

[71] βλ. help(Quotes).

Δοκιμάστε:	Σχόλιο
`x<-"Hello"`	Μεταβλητή με όνομα x για το κείμενο “Hello”.
`is.character(x)`	Είναι το x τύπου κείμενο (character); Επιστρέφει Αληθές (TRUE).
`y<-as.character(10)`	Μεταβλητή με όνομα y για τη μετατροπή του numeric 10 σε κείμενο ("10").
`y`	Η τρέχουσα τιμή του y, δηλαδή το κείμενο "10".
`y+1`	Λάθος, δεν γίνεται αριθμητική πράξη (πρόσθεση) κειμένου με αριθμό.
`as.numeric(y)+1`	Επιστρέφει 11, προσθέτει 1 στη μετατροπή του y σε αριθμό (10)
`as.numeric(x)`	Δεν γίνεται να μετατραπεί το κείμενο “hello” σε αριθμό, επιστρέφει NA[72].
`z<-"Μαθαίνω R"`	Μεταβλητή z για το κείμενο “Μαθαίνω R”.
`z<-'Μαθαίνω R'`	Ίδιο με το παραπάνω.
`z<-"Μαθαίνω 'R'"`	Για μονά εισαγωγικά σε κείμενο το ορίζουμε μέσα σε διπλά. Το z είναι Μαθαίνω ‘R’.
`z<-'Μαθαίνω "R"'`	Για διπλά εισαγωγικά σε κείμενο το ορίζουμε μέσα σε μονά: Το z είναι Μαθαίνω "R".

Στα προ-εγκατεστημένα πακέτα υπάρχει πληθώρα συναρτήσεων για χειρισμό κειμένου, δημιουργία κειμένου, εμφάνιση κειμένου στο Console, ανάγνωση και αποθήκευση κειμένου από και προς αρχεία ή άλλους προορισμούς κλπ. Βασικές τέτοιες συναρτήσεις είναι οι sprintf, paste, cat, print, readline και scan[73].

2.3.1 Σύνθεση κειμένου

Η συνάρτηση **sprintf**, επιστρέφει ένα κείμενο συνθέτοντάς το από άλλα αντικείμενα βάσει κάποιων κανόνων. Οι κανόνες σύνθεσης του κειμένου ίσως ξενίσουν τον νέο χρήστη αλλά είναι παρόμοιοι με αυτές που χρησιμοποιούνται σε αντίστοιχες εντολές βιβλιοθηκών της γλώσσας C και των συγγενικών της γλωσσών (C++, C#, Java κ.α.). Η συνάρτηση επεξεργάζεται ένα αρχικό κείμενο που δίνεται από τον χρήστη προς μορφοποίηση (το κείμενο διαμόρφωσης ή format string) και επιστρέφει ένα τελικό. Στο αρχικό κείμενο διαμόρφωσης χρησιμοποιούνται ειδικές ακολουθίες χαρακτήρων για να δείξουν πώς πρέπει να γίνει η σύνθεση του τελικού κειμένου. Ενδεικτικά, τέτοιες ακολουθίες χαρακτήρων[74] είναι οι:

- %d που δείχνει το σημείο στο οποίο θα τοποθετηθεί κάποιος ακέραιος αριθμός.
- %g που δείχνει το σημείο στο οποίο θα τοποθετηθεί κάποιος πραγματικός αριθμός.
- %.xf π.χ. όπου x ο αριθμός των δεκαδικών ψηφίων που θα εμφανιστεί (π.χ. %.2f).
- %x.yf π.χ. όπου y ο αριθμός των δεκαδικών ψηφίων που θα εμφανιστεί (π.χ. %8.2f) σε συνολικό χώρο x χαρακτήρων (κάποιοι ίσως είναι κενά).
- %s που δείχνει το σημείο στο οποίο θα τοποθετηθεί κάποιο κείμενο.

Στην sprint, μετά το κείμενο διαμόρφωσης δίνονται τα αντικείμενα που θέλουμε να χρησιμοποιήσει στην κατασκευή του τελικού κειμένου. Π.χ. στον κώδικα:

```
w<-6; d<-2
s<-sprintf("Το σπίτι έχει %d πόρτες και %d παράθυρα", d, w)
```

Στο s θα αποθηκευτεί το κείμενο: "Το σπίτι έχει 2 πόρτες και 6 παράθυρα ". Αυτό το κείμενο θα επιστρέψει η sprintf αντικαθιστώντας το πρώτο %d με την πρώτη παράμετρο μετά το κείμενο διαμόρφωσης (δηλαδή το w που είναι 6) και το δεύτερο %d με την επόμενη παράμετρο (δηλαδή το d που είναι 2). Ακολουθούν μερικά ακόμα παραδείγματα της sprintf:

Δοκιμάστε:	Σχόλιο
`sprintf("Το %d διά %d κάνει %g",3,4,3/4)`	%d για ακέραιο, %g για γενική εμφάνιση.
`sprintf("Το %g διά %g κάνει %g",3,4,3/4)`	%f για αριθμό με υποδιαστολή.
`sprintf("Το %.3f διά %.3f κάνει %.3f",3,4,3/4)`	%.3f για εμφάνιση 3 δεκαδικών ψηφίων.
`sprintf("Το %s είναι κείμενο","abcd")`	%s για κείμενο

Η συνάρτηση **paste** προσφέρει έναν άλλο τρόπο σύνθεσης κειμένου. Δημιουργεί (μέσω της συνάρτησης as.character) ένα κείμενο για καθένα από τα αντικείμενα που της δίνονται και μετά συγχωνεύει τα κείμενα αυτά σε ένα. Η παράμετρος sep της paste ορίζει το κείμενο που θα χρησιμοποιηθεί για να διαχωριστούν τα επιμέρους

[72] Η ειδική τιμή NA συμβολίζει πως τα σχετικά δεδομένα δεν είναι διαθέσιμα, βλ. παραπάνω §2.2.4 Ειδικές τιμές.
[73] Οι συναρτήσεις αυτής της ενότητας παρέχονται από το προ-εγκατεστημένο πακέτο ‘base’.
[74] Για άλλες ακολουθίες χαρακτήρων μετατροπής αντικειμένων σε κείμενο (conversion specifications) βλ. help(sprintf).

στο τελικό (ως προεπιλογή χρησιμοποιεί ένα κενό/space). Π.χ. το paste(1,5,10,sep=" και ") θα επιστρέψει το αντικείμενο character "1 και 5 και 10". Μερικά ακόμα παραδείγματα της paste:

Δοκιμάστε:	**Σχόλιο**
`paste("Η τετ. ρίζα του είναι",sqrt(3))`	Η συνάρτηση sqrt επιστρέφει την τετραγωνική ρίζα.
`paste("Σήμερα είναι", date())`	Η συνάρτηση date επιστρέφει την τρέχουσα ημερομηνία.
`paste("Τεστ","συνάρτησης","paste")`	Θα μπορούσε να γραφεί ως ένα αντικείμενο character.

Η paste μπορεί να διαχειριστεί και αντικείμενα πολλών στοιχείων (με μήκος > 1)[75], εφαρμόζοντας ανακύκλωση τιμών (αν χρειάζεται) και επιστρέφοντας διάνυσμα πολλών κειμένων. Π.χ. η εντολή paste(1:10, "και", 1:5, "ισούται με", 1:10 + 1:5) επιστρέφει ένα διάνυσμα 10 αντικειμένων character, δηλαδή το c("1 και 1 ισούται με 2", "2 και 2 ισούται με 4", "3 και 3 ισούται με 6", … , "9 και 4 ισούται με 13", "10 και 5 ισούται με 15"). Άλλα σχετικά παραδείγματα:

Δοκιμάστε:	**Σχόλιο**
`paste("Τετρ.ρίζα(",1:10,")=",sqrt(1:10))`	Η paste υποστηρίζει αντικείμενα με length>1.
`paste("Και",1:4)`	Καλό για γυμναστική.
`paste(paste("Και", 1:4),collapse=", ")`	Με συγχώνευση σε ένα τελικό κείμενο.

2.3.2 Έξοδος και εμφάνιση κειμένου

Οι επόμενες δύο συναρτήσεις (cat και print) έχουν βασικό στόχο να εμφανίζουν κείμενο στο Console. Αν μέχρι τώρα δουλεύετε απευθείας στο Console της R και τα αποτελέσματα των εντολών σας είτε εμφανίζονται άμεσα (λόγω auto-print) είτε ανατίθενται σε μεταβλητές. Επειδή μπορείτε εύκολα να εμφανίσετε τη μεταβλητή γράφοντας το όνομά της, ίσως αναρωτιέστε τον λόγο για συναρτήσεις όπως αυτές. Αργότερα όμως, όταν ο κώδικάς σας θα τρέχει αυτόνομα (π.χ. εκτελείτε ένα σενάριο R, βλ.§3.1 Σενάρια (R-script)) θα καταφεύγετε στη χρήση συναρτήσεων εμφάνισης κειμένου, καθώς το auto-print απενεργοποιείται και τα πιθανά αποτελέσματα δεν εμφανίζονται αυτόματα.

Στη συνάρτηση **cat** (concatenate and print) έχουμε ήδη αναφερθεί (βλ.§1.4 Μία πρώτη επαφή με την R στο RStudio Desktop). Η cat δέχεται σειρά από παραμέτρους δεδομένων, τις συγχωνεύει σε κείμενο και το εμφανίζει στο Console (ή το γράφει σε κάποιο αρχείο). Όπως στην paste, η παράμετρος sep ορίζει το κείμενο που θα χρησιμοποιηθεί για να διαχωριστούν τα επιμέρους. Η cat δεν επιστρέφει[76] το συγχωνευμένο κείμενο, απλώς το εμφανίζει (ή το προσθέτει στο αρχείο). Η μετατροπή σε κείμενο είναι μινιμαλιστική ενώ μόνο μερικώς[77] υποστηρίζονται αντικείμενα με μήκος>1.

Στο κείμενο που δίνεται σε συναρτήσεις εξόδου κειμένου όπως η cat, μπορούν να χρησιμοποιηθούν «χαρακτήρες διαφυγής» (escape characters)[78] οι οποίοι έχουν κάποια ειδική λειτουργία. Αυτοί δίνονται ως συνδυασμοί χαρακτήρων που ξεκινούν με το \, όπως οι:

- \n (χαρακτήρας newline) που δείχνει το σημείο στο οποίο θα συνεχίσει στην επόμενη γραμμή.
- \\ που δείχνει το σημείο στο οποίο ζητείται να εμφανιστεί ο χαρακτήρας \.
- \014 χαρακτήρας που καθαρίζει όλο το κείμενο από το Console (στέλνει Ctrl+L).

Ακολουθούν μερικά παραδείγματα:

Δοκιμάστε:	**Σχόλιο**
`cat(1,2,3,sep = " και ")`	Εμφανίζει «1 και 2 και 3»
`cat("Η τετ. ρίζα του 3 =",sqrt(3))`	Εμφανίζει «Η τετ. ρίζα του 3 = 1.732051»
`cat("Η ρίζα των",3:5,"=",sqrt(3:5))`	Εμφανίζει «Η ρίζα των 3 4 5 = 1.732051 2 2.236068»
`cat("Γειά σου");cat("Κόσμε")`	Η cat δεν αφήνει αυτομάτως κενά ή αλλάζει γραμμή.
`cat("Γειά σου\nΚόσμε")`	Μετά το \n θα αλλάξει η γραμμή.

[75] Για μια εισαγωγή στα πολυμελή αντικείμενα βλ.§2.4 Εισαγωγή στα διανύσματα (vector).
[76] Επιστρέφει NULL.
[77] Μερικώς υποστηρίζονται πολυμελή αντικείμενα (αντικείμενα που μπορούν να έχουν length>1 σε ατομική κατάσταση (atomic mode) δηλαδή με δεδομένα ενός μόνο από κάποιον βασικό τύπο "logical", "integer", "numeric"/"double", "complex", "character" και "raw", όπως π.χ. διανύσματα).
[78] Για όλες τις ακολουθίες χαρακτήρων μετατροπής αντικειμένων σε κείμενο (conversion specifications) βλ. help(sprintf).

`cat("\014Γειά σου Κόσμε")`	Καθαρίζει την οθόνη, μετά εμφανίζει το μήνυμα.
`cat(sprintf("Ένα τρίτο ίσον %g\n ",1/3)`	Συνδυασμός cat και sprintf.

Η συνάρτηση **writeLines** χρησιμοποιείται για έξοδο ενός αντικειμένου character[79] που προσδιορίζεται στην 1η παράμετρό της (text). Η έξοδος οδηγείται σε κάποια «σύνδεση» (συνήθως αρχείο)[80] που προσδιορίζεται στη 2η παράμετρό της (con). Αν δεν προσδιοριστεί σύνδεση, οδηγεί την έξοδο στο stdout εμφανίζοντας το κείμενο στο Console. Μετά την έξοδο ενός στοιχείου του text, η writeLines περνά στην επόμενη γραμμή.

Δοκιμάστε:	**Σχόλιο**
`writeLines("Γειά σου Κόσμε")`	Εμφανίζει «Γειά σου Κόσμε» σε μια γραμμή.

Η συνάρτηση **print** χρησιμοποιείται συχνά για να εμφανιστεί στο Console ένα αντικείμενο. Δέχεται ως δεδομένα προς εμφάνιση μόνο μια παράμετρο, αλλά αυτή (την πρώτη παράμετρο x) τη μετατρέπει αν χρειάζεται σε κείμενο με τον καταλληλότερο (ή πλέον επεξηγηματικό) τρόπο. Μετά εμφανίζει και επιστρέφει το αποτέλεσμα:

Δοκιμάστε:	**Σχόλιο**
`print("test")`	Η print εμφανίζει και επιστρέφει την 1η παράμετρό της.
`print("test",quote=F)`	Το quote επιτρέπει την απάλειψη εισαγωγικών από το κείμενο εξόδου.
`print(1/3)`	Η print εμφανίζει και επιστρέφει την 1η παράμετρό της.
`print(1/3,digits=2)`	Το digits αναφέρεται στα δεκαδικά ψηφία που θα εμφανιστούν.
`x<-print(1/3,digits=2)`	Η παραπάνω στρογγυλοποίηση αφορά μόνο την εμφάνιση του αριθμού.
`print(1/3,digits=20)`	Παρατηρήστε τη στρογγυλοποίηση λόγω ακριβείας.

Η print υποστηρίζει πολύπλοκα αντικείμενα και (παρά το απλοϊκό της όνομα που παραπέμπει σε εντολές εμφάνισης κειμένου άλλων γλωσσών προγραμματισμού), ο μηχανισμός που χρησιμοποιεί για μετατροπή ενός αντικειμένου σε κείμενο είναι πολυπλοκότερος από αυτόν της cat. Όπως αναφέρει και η τεκμηρίωση (βλ. help(print)), η print είναι μια generic function[81], άρα κάθε τύπος αντικειμένου αναλαμβάνει ο ίδιος να ορίσει τον τρόπο με τον οποίο θα γίνεται η περιγραφή του αντικειμένου ως κείμενο (και εμφάνιση του κειμένου αυτού), παρέχοντας μια σχετική μέθοδο. Ο τρόπος μετατροπής σε κείμενο μπορεί εύκολα να οριστεί και για νέους τύπους (κλάσεις) αντικειμένων (βλ.§6.2 Αντικειμενοστραφής προγραμματισμός). Αυτός είναι ο λόγος για τον οποίο η print παίρνει μόνο μια παράμετρο προς εμφάνιση, ένα αντικείμενο που θα περιγράφει ως κείμενο (αν μάλιστα δοθεί σε αυτή ένα αντικείμενο που είναι ήδη κείμενο, το εμφανίζει μέσα σε εισαγωγικά δίνοντας έμφαση στο ότι αυτό είναι όντως ένα αντικείμενο τύπου character - δεν σας φάνηκε περίεργο;). Μπορεί όμως να εμφανίσει αντικείμενα όπως πίνακες (matrix), λίστες (list), συναρτήσεις (function), μοντέλα και πολλά άλλα (που θα δούμε αργότερα) και αυτό να γίνει με τρόπο που οι ίδιοι οι τύποι ορίζουν[82]. Αντίθετα, η cat μπορεί να χρησιμοποιηθεί σε λιγότερους τύπους αντικειμένων αλλά παρέχει έναν ευέλικτο τρόπο σύνθεσης, συγχώνευσης και εμφάνισης κειμένου που είναι πιθανό να προέρχεται από πολλά αντικείμενα.

Έχει ήδη αναφερθεί ότι στην R υπάρχουν επιλογές που σχετίζονται με την τρέχουσα συνεδρία και μπορούν να οριστούν με τη συνάρτηση options (για περισσότερα βλ.§4.2.1.5 Η λίστα επιλογών συνεδρίας). Κάποιες από τις επιλογές αυτές αφορούν την προεπιλεγμένη συμπεριφορά κατά την έξοδο και εμφάνιση κειμένου. Τέτοιες επιλογές είναι οι max.print και digits. Συναρτήσεις που εμφανίζουν κείμενο (όπως οι cat, print ή άλλες συναρτήσεις που αξιοποιούν αυτές τις επιλογές) θα περιορίσουν τον μέγιστο αριθμό γραμμών που καταλαμβάνει η εμφάνιση ενός αντικειμένου στον αριθμό που ορίστηκε στη max.print, ενώ θα εμφανίσουν αριθμούς με όσα δεκαδικά ψηφία έχουν οριστεί στην επιλογή digits.

[79] Με ένα ή περισσότερα στοιχεία.
[80] Για πιθανά είδη συνδέσεων και συναρτήσεις δημιουργίας τους, βλ. help(connections).
[81] Το ίδιο ισχύει για πάρα πολλές άλλες συνήθεις συναρτήσεις όπως οι sum, mean, plot, summary κ.α. βλ.§2.7 Οι συναρτήσεις-βοηθήματα: str, summary και dput. Για τον ρόλο των generic συναρτήσεων, βλ. Κεφάλαιο 6 και ειδικότερα §6.3 Κλάσεις S3.
[82] Οι τύποι μπορεί να ορίζουν και πρόσθετες παραμέτρους για τη «δική τους» print. Αν δεν υπάρχει τέτοια για κάποιον τύπο, θα γίνει εμφάνιση του αντικειμένου με έναν γενικευμένο τρόπο, δηλαδή καλώντας τη συνάρτηση print.default. Παράμετροι της print.default περιλαμβάνουν τη max στην οποία μπορεί να οριστεί ο μέγιστος αριθμός γραμμών που θα εμφανιστούν, παρακάμπτοντας την επιλογή max.print που αναφέρεται παρακάτω.

2.3.3 Ανάλυση και επεξεργασία κειμένου

Το κείμενο (στις διάφορες μορφές του) είναι βασικό στοιχείο της ανθρώπινης δραστηριότητας, οπότε η ανάλυση και επεξεργασία του έχει πολλές και διαφορετικές εφαρμογές. Τα ενσωματωμένα πακέτα της R, προσφέρουν συναρτήσεις για τη βασική επεξεργασία και ανάλυση κειμένου, το οποίο συνήθως είναι αποθηκευμένο σε αντικείμενα τύπου character. Κάποιες από αυτές τις συναρτήσεις παρουσιάζονται στις ενότητες που ακολουθούν. Όμως πέραν των βασικών αυτών εργαλείων, στο CRAN διατίθεται μεγάλος αριθμός από πακέτα (που συνεχώς εμπλουτίζεται) τα οποία προσθέτουν ποικίλες ιδιαίτερες δυνατότητες που σχετίζονται με τον χειρισμό κειμένου. Μεταξύ αυτών (ενδεικτικά), το πακέτο ‘stringr΄ [34] με συναρτήσεις χειρισμού και ανάλυσης κειμένου[83] που βελτιώνουν ή προσθέτουν σε όσα παρέχει το πακέτο ‘base’, το πακέτο ‘xml2’ [35] για εξαγωγή του κειμένου από διάφορες μορφές πηγών (html, xml, docx κ.α.)[84], το πακέτο ‘quanteda’ [36] για λεκτική ανάλυση, το πακέτο tm [37] για υποβοήθηση της εξόρυξης δεδομένων από κείμενο, τα πακέτα ‘syuzhet’ [38] και ‘SentimentAnalysis’ [39] για εξόρυξη γνώμης (ανάλυση συναισθήματος) στο κείμενο ή ακόμα και πακέτα όπως το ‘wordcloud’ για δημιουργία οπτικοποιήσεων βασισμένων στη συχνότητα εμφάνισης των λέξεων στο κείμενο.

2.3.3.1 Χρήσιμες συναρτήσεις επεξεργασίας κειμένου

Το βασικό πακέτο ‘base’ παρέχει χρήσιμες συναρτήσεις επεξεργασίας αντικειμένων character, μεταξύ αυτών τις **toupper**, **tolower**, **nchar** και **substr**, Οι πρώτες δύο επιστρέφουν ένα character όπου έχει απλώς γίνει αλλαγή των χαρακτήρων σε κεφαλαία ή πεζά γράμματα. Η nchar επιστρέφει τον αριθμό χαρακτήρων που περιέχει το δοθέν κείμενο, ενώ η substr επιστρέφει ή αντικαθιστά κάποιο τμήμα του κειμένου. Μερικά παραδείγματα[85]:

Δοκιμάστε:	**Σχόλιο**
`x<-"Snowboard"`	Μεταβλητή με όνομα x, για το κείμενο “Snowboard ”.
`nchar(x)`	Αριθμός χαρακτήρων στο x, επιστρέφει 9.
`tolower(x)`	Επιστρέφει character με όλα τα γράμματα πεζά ("snowboard")
`toupper(x)`	Επιστρέφει character με όλα τα γράμματα κεφαλαία ("SNOWBOARD")
`substr(x,1,4)`	Επιστρέφει character με τμήμα του x από το $1^{ο}$ ως το $4^{ο}$ γράμμα, δηλαδή "Snow".
`substr(x,1,5)<-"Surf"`	Αντικαθιστά τμήμα του x ($1^{ο}$ ως $4^{ο}$ γράμμα) με "Surf", το x γίνεται "Surfboard".

2.3.3.2 Κανονικές εκφράσεις (regular expressions)

Ένα ευέλικτο εργαλείο για ανάλυση και αξιοποίηση κειμένου είναι οι κανονικές εκφράσεις (regular expressions)[86]. Οι κανονικές εκφράσεις (ή κανονικές παραστάσεις) περιγράφουν μοτίβα (πρότυπα) τα οποία θα αναζητηθούν σε κάποιο κείμενο, κάτι που έχει πολλές εφαρμογές στους υπολογιστές γενικότερα. Για παράδειγμα, η επεξεργασία του κώδικα μιας οποιασδήποτε συμβατικής γλώσσας προγραμματισμού ξεκινά με ανάλυση του κειμένου μέσω κανονικών εκφράσεων. Λόγω της ευρείας χρήσης τους σε πολλές εφαρμογές, υπάρχει σημαντική διαθέσιμη βιβλιογραφία σχετικά με τον τρόπο σύνταξης και δυνατότητες των κανονικών εκφράσεων (βλ. ενδεικτικά [40] και [41],).

Διάφορες συναρτήσεις παρέχονται από πακέτο ‘base’ για επεξεργασία κειμένου μέσω κανονικών εκφράσεων και περιλαμβάνουν τις **grep**, **grepl**, **regexpr** και **gregexpr**. Οι συναρτήσεις κανονικών εκφράσεων υποστηρίζουν δύο μορφές σύνταξης: τη μορφή που ορίζεται από το πρότυπο POSIX 1003.2 και την παραλλαγή που προέρχεται από τη γλώσσα Perl. Τα παραδείγματα που ακολουθούν, βασίζονται στο πρώτο πρότυπο.

Μια κανονική έκφραση είναι κείμενο. Το κείμενο αυτό περιέχει στοιχεία που περιγράφουν το μοτίβο, το πρότυπο δηλαδή που πρέπει να εντοπιστεί σε κάποιο άλλο κείμενο. Έτσι, μια κανονική έκφραση μπορεί απλώς

[83] Στον προγραμματισμό, η λέξη string είναι συνώνυμη του κειμένου (σημαίνει σειρά χαρακτήρων).

[84] βλ. και §9.1 Εισαγωγή και εξαγωγή δεδομένων.

[85] Γενικότερα για τη σωστή εκτέλεση των παραδειγμάτων που περιέχουν ελληνικούς χαρακτήρες, αλλά και ειδικότερα σε αυτή την ενότητα που γίνεται επεξεργασία σε επίπεδο χαρακτήρων κειμένου (κάτι που επηρεάζεται ιδιαίτερα από τις τοπικές ρυθμίσεις) απαραίτητο είναι να έχει εκτελεστεί η εντολή Sys.setlocale για Ελληνικά (βλ.§1.7 Σχετικά με τα παραδείγματα του βιβλίου).

[86] Δεν πρέπει να συγχέεται με τον τύπο αντικειμένου expression της R.

να περιέχει το κείμενο που αναζητείται. Ας ορίσουμε πως το κείμενο προς επεξεργασία περιέχεται στο παρακάτω αντικείμενο character με όνομα x:

```
x <- "Δοκιμή,123,ΑΒΓΑΒΓ,ΑΒΔ,δοκιμής τέλος"
```

Οι συναρτήσεις grep, grepl, regexpr, gregexpr μπορούν να χρησιμοποιηθούν για να αναζητηθεί το κείμενο "ΑΒΓ" στο x:

Δοκιμάστε:	**Σχόλιο**
`grep("ΑΒΓ",x)`	Επιστρέφει 1, τον αριθμό του στοιχείου στο x που περιέχει "ΑΒΓ" (βλ. παρακάτω)
`grepl("ΑΒΓ",x)`	Επιστρέφει Αληθές (TRUE)[87], αφού το x περιέχει "ΑΒΓ".
`grepl("ΑΒΓΔ",x)`	Επιστρέφει Αναληθές (FALSE), αφού το x δεν περιέχει "ΑΒΓΔ".
`regexpr("ΑΒΓ",x)`	Επιστρέφει θέση και μήκος του πρώτου (από αριστερά) "ΑΒΓ" στο x (βλ. παρακάτω).
`gregexpr("ΑΒΓ",x)`	Επιστρέφει θέση και μήκος κάθε "ΑΒΓ" που υπάρχει στο x (βλ. παρακάτω).

Ο αριθμός 1 που επέστρεψε η εντολή grep("ΑΒΓ",x) καθώς και τα αποτελέσματα των εντολών regexpr και gregexpr στις εντολές του παραδείγματος είναι προσαρμοσμένα στο γεγονός πως ένα αντικείμενο τύπου character[88] μπορεί να περιέχει περισσότερα του ενός στοιχεία (κείμενα). Ειδικά σε εφαρμογές κανονικών εκφράσεων συμβαίνει συχνά το κείμενο προς επεξεργασία να αποτελείται από πολλά τμήματα κειμένου που ίσως αντιστοιχούν σε διαφορετικές γραμμές ή προτάσεις κλπ., ανάλογα με τον τύπο του κειμένου και την εφαρμογή. Καθένα από αυτά είναι συνήθως αποθηκευμένο σε ένα διαφορετικό στοιχείο του ίδιου αντικειμένου τύπου character.

Άρα εδώ η επιστροφή (ο αριθμός 1) της grep καταγράφει ότι το μοτίβο "ΑΒΓ" βρέθηκε στο 1ο (και εδώ μοναδικό) στοιχείο του x[89]. Αντίστοιχα, η επιστροφή της εντολής gregexpr("ΑΒΓ",x) είναι λίστα[90] (που εδώ όμως έχει μόνο ένα στοιχείο):

```
> gregexpr("ΑΒΓ",x)
[[1]]
[1] 12 15
attr(,"match.length")
[1] 3 3
```

Αν επικεντρωθούμε στη χρησιμότητα των συναρτήσεων κανονικών εκφράσεων και όχι στις τεχνικές λεπτομέρειές τους, παρατηρούμε ότι οι τιμές 12 και 15 που επέστρεψε η gregexpr αντιστοιχούν στη θέση όπου εντοπίζεται το πρώτο και το δεύτερο "ΑΒΓ". Εδώ ως θέση θεωρείται ο αριθμός των χαρακτήρων (συμπεριλαμβανομένων και πιθανών κενών) αν καταμετρηθούν από το αριστερό άκρο του x. Οι αριθμοί 3 και 3 που ακολουθούν[91] αντιστοιχούν στον αριθμό χαρακτήρων που βρέθηκαν να ταιριάζουν στην κανονική έκφραση "ΑΒΓ" και είναι και στις δύο περιπτώσεις 3, αφού το μοτίβο αυτό αναζητά το συγκεκριμένο σταθερό κείμενο ("ΑΒΓ") και αυτό έχει 3 γράμματα.

Άλλη χρήσιμη συνάρτηση είναι η **strsplit** με την οποία το αρχικό κείμενο μπορεί να χωριστεί σε τμήματα στα σημεία όπου εντοπίζεται το μοτίβο που δίνεται στη 2η παράμετρο split[92]. Για παράδειγμα:

```
> strsplit(x, "ΑΒΓ")
[[1]]
[1] "Δοκιμή,123,"     ""     ",ΑΒΔ,δοκιμής τέλος"
```

Ή για ένα πιο σύνηθες παράδειγμα εφαρμογής της strsplit:

[87] Για τις λογικές τιμές TRUE και FALSE βλ.§2.5 Λογικές πράξεις, συγκρίσεις και ο βασικός τύπος logical.

[88] Όλοι οι βασικοί τύποι αντικειμένων (numeric, character, logical κλπ.) είναι ατομικά διανύσματα, βλ.§4.1.1 Ο τύπος vector (διάνυσμα). Άρα όλα τα αντικείμενα τύπου character είναι διανύσματα είτε ενός είτε περισσότερων στοιχείων.

[89] Αν δεν είχε εντοπιστεί το "ΑΒΓ" στο x, η ίδια εντολή grep θα επέστρεφε αντικείμενο integer μηδενικού μήκους (χωρίς στοιχεία), δηλαδή numeric(0).

[90] βλ.§4.2.1 Ο τύπος list (λίστα). Αν η επεξεργασία αφορά ένα μόνο κείμενο (όπως εδώ) το αποτέλεσμα μπορεί να μετατραπεί σε διάνυσμα με τη συνάρτηση unlist, π.χ. η εντολή unlist(gregexpr("ΑΒΓ",x)) επιστρέφει διάνυσμα με στοιχεία 12 και 15 (δηλαδή c(12 15)).

[91] Οι αριθμοί αυτοί έχουν καταγραφεί ως ιδιότητες του πρώτου (και εδώ μοναδικού) στοιχείου της λίστας (βλ. §4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων).

[92] Όπως και με τη gregexpr, έτσι και στην strsplit το αποτέλεσμα επιστρέφεται ως λίστα, μπορεί όμως να μετατραπεί σε διάνυσμα με την unlist, εδώ π.χ. με την εντολή unlist(strsplit(x, ",")). Βλ. και παραπάνω υποσημείωση 90.

```
> strsplit(x, ",")
[[1]]
[1] "Δοκιμή"   "123"   "ΑΒΓΑΒΓ"  "ΑΒΔ"   "δοκιμής τέλος"
```

Όμως το συντακτικό των κανονικών εκφράσεων επιτρέπει τη χρήση και τον συνδυασμό ειδικών χαρακτήρων[93] για την περιγραφή του μοτίβου. Οι ειδικοί αυτοί χαρακτήρες και συνδυασμοί, προσθέτουν τη δυνατότητα αναζήτησης και «ταιριάσματος» πολλαπλών παραλλαγών του κειμένου με το μοτίβο. Ενδεικτικοί τέτοιοι χαρακτήρες και συνδυασμοί είναι:

- το ^ σημαίνει ταίριασμα μόνο στο αριστερό άκρο («αρχή») του κειμένου.
- το $ για ταίριασμα μόνο στο δεξί άκρο («τέλος») του κειμένου.
- ο συνδυασμός [και] αφορά ταίριασμα εναλλακτικών χαρακτήρων (κλάσεων χαρακτήρων). Επίσης υπάρχουν προκαθορισμένες κλάσεις χαρακτήρων (όπως η [:digit:] για αριθμητικά ψηφία ή η [:upper:] για κεφαλαία γράμματα), που αξιοποιούν τις σχετικές τοπικές ρυθμίσεις του συστήματος.
- το ^ εντός [και] σημαίνει μη ταίριασμα.
- το . για ταίριασμα με οποιοδήποτε χαρακτήρα.
- το * για πολλαπλό ταίριασμα (0 η περισσότερες φορές).
- ο συνδυασμός { και } για ορισμό αριθμού ζητούμενων επαναλήψεων του μοτίβου.
- οι παρενθέσεις (και) για ομαδοποίηση εντολών στο μοτίβο.
- το ? για προαιρετικό μοτίβο.
- το | για καταχώρηση εναλλακτικών μοτίβων που ταιριάζουν.
- το – για ορισμό περιοχής χαρακτήρων (π.χ. α-ε σημαίνει α,β,γ,δ ή ε).

Εκφράσεις που χρησιμοποιούν χαρακτήρες όπως οι παραπάνω συνδυάζονται ώστε να δημιουργηθεί ο ζητούμενος κανόνας για το μοτίβο που οι εντολές (grep, gregexpr κλπ) θα προσπαθήσουν να ταιριάξουν στο κείμενο. Αν πρέπει οι χαρακτήρες να χρησιμοποιηθούν κυριολεκτικά (και όχι με το ειδικό νόημα που έχουν ως μέρος μιας κανονικής έκφρασης), οι χαρακτήρες αυτοί καταχωρούνται με πρόθεμα το \ (δηλαδή π.χ. αν το μοτίβο αναζητά τον χαρακτήρα αστερίσκο, αυτό καταχωρίζεται ως *)[94].

Μερικά παραδείγματα ταιριάσματος (στο x που ορίστηκε παραπάνω):

- Η κανονική έκφραση "[δΔ]οκιμή" ταιριάζει στο "Δοκιμή" και "δοκιμή" του x.
- Η κανονική έκφραση "^[δΔ]οκιμή" ταιριάζει μόνο στο πρώτο "Δοκιμή" του x, καθώς το ^ απαιτεί ταίριασμα στην αρχή του κειμένου.
- Η κανονική έκφραση "ΑΒ." ταιριάζει στα τμήματα "ΑΒΓ", "ΑΒΓ" και "ΑΒΔ" του x, επειδή το . σημαίνει ταίριασμα με οποιονδήποτε χαρακτήρα.
- Η κανονική έκφραση "Α.*?," στο x θα ταιριάξει τα τμήματα "ΑΒΓΑΒΓ," και "ΑΒΔ," αφού αναζητά κείμενα που αρχίζουν από Α, ακολουθούνται από οτιδήποτε και ολοκληρώνονται με κόμμα.
- Η κανονική έκφραση "(ΑΒΓ){2}" ταιριάζει στο "ΑΒΓΑΒΓ" του x, καθώς αναζητά ΑΒΓ που επαναλαμβάνεται δύο φορές.
- Η κανονική έκφραση "..," θα ταιριάξει με τα κείμενα "μή,", "23,", "ΒΓ," και "ΒΔ," στο x, δηλαδή δύο οποιουσδήποτε χαρακτήρες ακολουθούνται από κόμμα.
- Η κανονική έκφραση ".*," θα ταιριάξει με το τμήμα "Δοκιμή,123,ΑΒΓΑΒΓ,ΑΒΔ," του x, αφού ζητά (το μεγαλύτερο) κείμενο που τελειώνει με κόμμα.
- Η κανονική έκφραση ".*?," στο x θα ταιριάξει με τα τμήματα "Δοκιμή,", "123,", "ΑΒΓΑΒΓ," και "ΑΒΔ,", αφού ζητά κείμενα που τελειώνουν με κόμμα.
- Η κανονική έκφραση "(.*?),|(.*?)$" αν εφαρμοστεί στο x ταιριάζει κείμενα που τελειώνουν με κόμμα ή είναι στο τέλος, άρα θα εντοπίσει τα "Δοκιμή,", "123,", "ΑΒΓΑΒΓ,", "ΑΒΔ," και "δοκιμής τέλος".
- Η κανονική έκφραση "...ι..." όπως και η ".{3}ι.{3} αφορά ταίριασμα τριών οποιονδήποτε χαρακτήρων πριν και μετά ένα ιώτα. Θα εντοπίσει τα τμήματα "Δοκιμή," και "δοκιμής" στο x.
- Τέλος η κανονική έκφραση "[δΔ]ο((.*?),|(.*?)$)" θα ταιριάξει κείμενα που ξεκινούν από δέλτα (κεφαλαίο ή μικρό), ακολουθούμενο από όμικρον και μετά οποιοδήποτε κείμενο είε

[93] Αποκαλούνται μετα-χαρακτήρες (meta-characters).

[94] Σε κάποιες περιπτώσεις απαιτούνται πολλαπλά \, π.χ. * για το κείμενο *. Χρήση του \ σε καταχώρηση χαρακτήρων διαφυγής (escape characters) αναφέρθηκε και στη §2.3.2 Έξοδος και εμφάνιση κειμένου.

ολοκληρώνεται με κόμμα είτε βρίσκεται στο τέλος του κειμένου. Θα ταιριάξει με τα τμήματα "Δοκιμή," και "δοκιμής τέλος" στο x.

Τέτοιες κανονικές εκφράσεις μπορούν να δοθούν ως πρώτη παράμετρος (pattern) στις συναρτήσεις grep, grepl, regexpr, gregexpr ή δεύτερη παράμετρος (split) στη συνάρτηση strsplit, με αντίστοιχα αποτελέσματα αυτών που προέκυψαν όταν η κανονική έκφραση περιείχε μόνο σταθερό κείμενο (το ΑΒΓ στο προηγούμενο παράδειγμα). Π.χ.

Δοκιμάστε:	**Σχόλιο**
`gregexec("[δΔ]οκιμή",x)`	Το αποτέλεσμα καταγράφει θέσεις του "δοκιμή" ή "Δοκιμή" στο x (1 και 23).
`gregexec("ΑΒ.",x)`	Αντίστοιχα, θέσεις των "ΑΒΓ", "ΑΒΓ", και "ΑΒΔ" στο x (12 , 15 και 19).

Άλλη σχετική συνάρτηση είναι η **regmatches** που εξάγει το τμήμα ή τα τμήματα κειμένου που εντοπίστηκαν μέσω της κανονικής έκφρασης, ξεκινώντας από το αποτέλεσμα της regexec ή gregexec. Για παράδειγμα τα τμήματα που ταίριαξαν (μέσω της gregexec) με την κανονική έκφραση "ΑΒ." είναι τα "ΑΒΓ", "ΑΒΓ" και "ΑΒΔ", και αυτά θα επιστρέψει η regmatches:

```
> regmatches(x, gregexec("ΑΒ.",x))
[[1]]
     [,1]  [,2]  [,3]
[1,] "ΑΒΓ" "ΑΒΓ" "ΑΒΔ"
```

Κανονικές εκφράσεις μπορούν επίσης να δοθούν στις συναρτήσεις **sub** και **gsub** που στόχο έχουν την αντικατάσταση του κειμένου που αναγνωρίστηκε από την κανονική έκφραση με κάποιο άλλο. Η sub θα αντικαταστήσει μόνο την αριστερότερη περίπτωση ταιριάσματος, ενώ η gsub όλες. Το επόμενο παράδειγμα επιστρέφει το κείμενο στο x αλλά με το τμήμα "Δοκιμή" (με μικρό ή κεφαλαίο αρχικό δέλτα) να έχει αντικατασταθεί από το κείμενο "Έναρξη":

Δοκιμάστε:	**Σχόλιο**
`sub("[δΔ]οκιμή","Έναρξη",x)`	Επιστρέφει το κείμενο "Έναρξη,123,ΑΒΓΑΒΓ,ΑΒΔ,δοκιμής τέλος".
`gsub("[δΔ]οκιμή","Έναρξη",x)`	Επιστρέφει το κείμενο "Έναρξη,123,ΑΒΓΑΒΓ,ΑΒΔ,Έναρξης τέλος".

Τέλος για όσους έχουν οικειότητα με τα μοτίβα (wildcards) που υποστηρίζουν πολλά ΛΣ υπολογιστών για την αναζήτηση αρχείων, καλό είναι να αναφερθεί η συνάρτηση **glob2rx** που δέχεται ένα κείμενο με αυτού του τύπου χαρακτήρες «μπαλαντέρ» και επιστρέφει την αντίστοιχη κανονική έκφραση[95]. Για παράδειγμα η εντολή glob2rx("no*.exe") θα επιστρέψει την κανονική έκφραση που χρειάζεται για να αναζητηθεί οποιοδήποτε κείμενο ξεκινά από "no" και τελειώνει σε ".exe" (όπως π.χ. το notepad.exe).

Περισσότερα για τις κανονικές εκφράσεις υπάρχουν στο help("regex"), ενώ για παραδείγματα εφαρμογής τους βλ. [18].

2.3.4 Είσοδος κειμένου

Αναφέρθηκαν ήδη οι συναρτήσεις cat, και print που χρησιμοποιούνται για να εμφανίσουν κείμενο στο Console. Η επόμενη συνάρτηση **readline** εμφανίζει μεν ένα κείμενο ως προτροπή (prompt) προς τον χρήστη αλλά ακολούθως περιμένει να καταγράψει (και να επιστρέψει) μια γραμμή κειμένου που ο χρήστης θα καταχωρήσει στο Console. Διαβάζει δηλαδή μία γραμμή κειμένου από τον χρήστη:

Δοκιμάστε:	**Σχόλιο**
`n<-readline(prompt="Πως λέγεσαι;")`	Περιμένει γραμμή κειμένου από τον χρήστη και το βάζει στο n.
`cat ("Γειά σου",n,"τι κάνεις;")`	Εμφανίζει μήνυμα χρησιμοποιώντας ό,τι έγραψε ο χρήστης.
`x<-readline(prompt="Δώσε αριθμό:")`	Περιμένει γραμμή κειμένου από τον χρήστη και το βάζει στο x.
`x<-as.numeric(x)`	Μετατροπή του κειμένου x σε αριθμό.
`cat(x,"επί 6 ίσον",6*x)`	Εμφανίζει μήνυμα χρησιμοποιώντας ό,τι έγραψε ο χρήστης.

[95] Η συγκεκριμένη συνάρτηση περιέχεται στο προ-εγκατεστημένο πακέτο 'utils'.

Άλλη συνάρτηση που επιτρέπει την εισαγωγή κειμένου είναι η **readLines**. Μπορεί να διαβάσει γραμμές κειμένου από αρχείο ή άλλη πηγή (που ορίζεται στην παράμετρο της con). Αν δεν οριστεί πηγή, διαβάζει κείμενο από το Console (το stdin):

Δοκιμάστε:	**Σχόλιο**
`x<-readLines(n=1)`	Διαβάζει μία γραμμή κειμένου (n=1) από τον χρήστη και την αποθηκεύει στο x.

Παρεμφερής αλλά πιο ευέλικτος τρόπος να εισαχθούν δεδομένα είναι η χρήση της συνάρτησης **scan**. Όπως και η readLines, η scan μπορεί να δεχτεί δεδομένα από το πληκτρολόγιο, από αρχείο ή άλλη πηγή καθώς και να διαβάσει πολλαπλές τιμές[96] από διάφορους τύπους δεδομένων[97]. Μεταξύ των τύπων αυτών είναι και ο character:

Δοκιμάστε:	**Σχόλιο**
`n<-scan(n=1,what="character")`	Διαβάζει μία λέξη (n=1) από τον χρήστη και την αποθηκεύει στο n.

Οι readLines και scan μπορούν να διαβάσουν περισσότερα από ένα αντικείμενα (με μέγιστο αριθμό να ορίζεται στην παράμετρό τους n). Έτσι το αντικείμενο που επιστρέφεται μπορεί να έχει πολλά στοιχεία, οπότε ο χειρισμός του γίνεται με τον τρόπο που περιγράφεται στη §2.4 Εισαγωγή στα διανύσματα (vector).

2.3.5 Υποστήριξη τοπικών γλωσσών

Οι εντολές σε πολλά από τα παραπάνω παραδείγματα έχουν ενσωματωμένα κομμάτια κειμένου. Τέτοια κείμενα δεν αλλάζουν, κάτι που ίσως είναι πρόβλημα αν ο κώδικας πρέπει να υποστηρίζει πολλαπλές ανθρώπινες γλώσσες. Με συναρτήσεις όπως η gettext, η R παρέχει τη δυνατότητα να οριστούν εναλλακτικές μεταφράσεις των λεκτικών των μηνυμάτων[98] και να αναζητηθούν ανάλογα με τις τοπικές ρυθμίσεις (locale) υποκαθιστώντας τα αρχικά, ενώ ngettext προσθέτει στο παραπάνω και έναν τρόπο να υπάρχουν διαφορετικά μηνύματα ενικού και πληθυντικού αριθμού (εξαρτάται από τις ιδιαιτερότητες της γλώσσας αλλά λειτουργεί παρόμοια σε αγγλικά και ελληνικά). Πχ. η εντολή sprintf(ngettext(x, "Εγινε %d εγγραφή", "Εγιναν %d πωλήσεις"), x) θα επιστρέψει το πρώτο κείμενο "Εγινε 1 εγγραφή" αν το x είναι 1, ενώ σε κάθε άλλη περίπτωση το δεύτερο, π.χ. για x ίσο με 3 θα επιστρέψει "Εγιναν 3 εγγραφές".

2.3.6 Κείμενο και εντολές

Κλείνοντας την εισαγωγική ενότητα για τον τύπο character, καλό είναι να γίνει μια παρατήρηση σχετικά με το κείμενο στις εντολές R: η R θα προσπαθήσει να μεταφράσει, να εκτελέσει, να αποτιμήσει οτιδήποτε στον κώδικα δεν βρίσκεται μέσα σε εισαγωγικά και - συνήθως - θα θεωρήσει αντικείμενο κειμένου οτιδήποτε βρίσκεται μέσα σε εισαγωγικά, άρα δεν θα το μεταφράσει ούτε θα το εκτελέσει. Επειδή όμως τα ονόματα των αντικειμένων (μεταβλητών, συναρτήσεων κλπ.) είναι κείμενο, αν οι κανόνες σύνταξης της εντολής που δόθηκε το επιτρέπει (ή σε κάποιες περιπτώσεις το επιβάλλει), τα ονόματα των αντικειμένων μπορούν να γραφούν και μέσα σε εισαγωγικά και η R θα τα αναγνωρίσει και θα εκτελέσει την εντολή.

Αντίστροφα, αν πρέπει όντως να εκτελεστεί κάποιος κώδικας R που είναι αποθηκευμένος ως κείμενο (μέσα σε εισαγωγικά), η εντολή **parse** μπορεί θα κάνει τη λεκτική ανάλυση του κειμένου (το απαραίτητο πρώτο στάδιο της μετάφρασης για να γίνει η εκτέλεση του κώδικα), ενώ το αποτέλεσμα της ανάλυσης[99] είναι ένα αντικείμενο τύπου «έκφραση» (expression) και μπορεί να εκτελεστεί με τη συνάρτηση **eval**. Η συνάρτηση eval εκτελεί την έκφραση που της δίνεται ως όρισμα. Για περισσότερα βλ. και §4.3.4 Αντικείμενα τύπου language

---

[96] Ο αριθμός δεδομένων που θα διαβαστεί ελέγχεται από την παράμετρο n της scan. Αν δεν οριστεί παράμετρος n, η scan θα ζητά δεδομένα έως να καταχωρηθεί μια κενή γραμμή. Αν διαβαστούν περισσότερα του ενός δεδομένα, αυτά επιστρέφονται σε ένα αντικείμενο τύπου vector (atomic ή list). Οι τύποι αυτοί περιγράφονται σε άλλες ενότητες του βιβλίου.

[97] Οι τύποι που υποστηρίζονται από τη scan είναι: logical, integer, numeric, complex, character, raw και list. Οι τύποι αυτοί περιγράφονται σε άλλες ενότητες του βιβλίου.

[98] Αφορά κυρίως κώδικα που ενσωματώνεται σε πακέτα (βλ. Κεφ. 8).

[99] Το αποτέλεσμα του parse είναι μια δομή τύπου expression (έκφραση).

(expression, call, name και formula). Για παράδειγμα, το αποτέλεσμα των παρακάτω εντολών θα είναι τρείς μεταβλητές, x (numeric με τιμή 4), y (character με τιμή "z<-sqrt(x)") και z (numeric με τιμή 2):

Δοκιμάστε:	**Σχόλιο**
`x<-"sqrt"(16)`	Η συνάρτηση sqrt ως κείμενο, επιστρέφει 4 και το ονομάζει x.
`y<-"z<-sqrt(x)"`	Μία εντολή R ως κείμενο στη μεταβλητή y (τύπου character).
`eval(parse(text=y))`	Εκτέλεση εντολής που υπάρχει ως κείμενο στη μεταβλητή y.

2.4 Εισαγωγή στα διανύσματα (vector)

Η R χειρίζεται τις δομές δεδομένων με την ίδια ευκολία που χειρίζεται τις μοναδικές τιμές. Οι δομές δεδομένων είναι τρόποι οργάνωσης των δεδομένων. Το πακέτο ‘base’ της R παρέχει διάφορους έτοιμους τύπους αντικείμενων που είναι σχεδιασμένοι να υλοποιούν κάποιο τύπο δεδομένων, να περιέχουν πολλαπλά δεδομένα ή άλλα αντικείμενα οργανωμένα σε κάποια δομή. Ο πλέον στοιχειώδης τέτοιος τύπος δομής δεδομένων είναι το διάνυσμα και υλοποιείται από τον τύπο αντικειμένων vector που αποτελεί σημαντικότατο στοιχείο της γλώσσας R. Πρόκειται ουσιαστικά για μια λίστα αντικειμένων συνήθως του ίδιου τύπου[100] τα οποία (όπως οι μαθηματικές ακολουθίες) έχουν συγκεκριμένη θέση μέσα στο διάνυσμα και ο αριθμός τους αναφέρεται ως μήκος.

2.4.1 Οι βασικές εντολές των διανυσμάτων

Τα διανύσματα (vector), είναι ο βασικότερος τύπος αντικειμένων στην R. Ακόμα και αν πρόκειται για μια απλή τιμή (π.χ. έναν μοναδικό αριθμό) για την R είναι ένα διάνυσμα. Έτσι, οι βασικοί τύποι αντικειμένων που χρησιμοποιήθηκαν ήδη σε προηγούμενα παραδείγματα (integer, numeric, complex, character, logical κλπ.) είναι διανύσματα[101] και μπορούν να περιέχουν πολλές τιμές, πολλά στοιχεία του ίδιου τύπου δεδομένων. Απλά τα αντικείμενα στα έως τώρα παραδείγματα είχαν μόνο ένα στοιχείο, ήταν (εκείνη τη στιγμή) μονομελή, με μήκος (length) ίσο με 1. Πάντως η έννοια του διανύσματος συνήθως παραπέμπει σε αντικείμενα που όντως (μπορούν να) περιέχουν περισσότερα από ένα στοιχεία.

Έτσι τα vector είναι η βάση και για πολλούς άλλους τύπους αντικειμένων που παρουσιάστηκαν ήδη αλλά και άλλους που θα συναντήσουμε σε επόμενες ενότητες (βλ. Κεφάλαιο 4). Καθώς τα vector μπορούν να περιέχουν πολλά στοιχεία, στην ενότητα αυτή θα είναι η πρώτη φορά που θα έχουμε αντικείμενα με «μήκος» (length, αριθμό στοιχείων που περιέχει το αντικείμενο) μεγαλύτερο του 1.

Πολλές συναρτήσεις που παρέχονται με την R, δημιουργούν αντικείμενα τύπου vector, δηλαδή διανύσματα. Συνήθης συνάρτηση δημιουργίας πολυμελών διανυσμάτων είναι η ‘c’ (combine values into vector or list) του πακέτου ‘base’, η οποία συνθέτει ένα διάνυσμα από άλλα αντικείμενα. Αν πρόκειται για αντικείμενα οποιουδήποτε βασικού τύπου (logical, integer, numeric, complex, character κλπ.), το νέο αντικείμενο που θα προκύψει θα είναι του ίδιου βασικού τύπου ή παρεμφερούς αν χρειαστεί να γίνει κάποια μετατροπή των δεδομένων κατά την ομογενοποίηση αυτή. Αν όμως δεν μπορεί να βρεθεί ένας τέτοιος ενιαίος τρόπος αποθήκευσης (mode) των τιμών των αντικειμένων που να αντιστοιχεί σε κάποιον βασικό τύπο δεδομένων, η συνάρτηση c θα επιστρέψει ένα διάνυσμα σε ειδικό mode που ονομάζεται list (βλ.§4.2.1 Ο τύπος list (λίστα)).

Πάντως πολύ συχνά τα στοιχεία του διανύσματος (δηλαδή τα δεδομένα που περιέχει σε κάθε θέση) είναι του ίδιου βασικού τύπου και το αντικείμενο που προκύπτει ανήκει στο τύπο αυτό. Όπως αναφέρθηκε ήδη, ο τύπος αυτός μπορεί να είναι οποιοσδήποτε βασικός τύπος, όχι μόνο αριθμοί. Στην ορολογία της R, ένα τέτοιο διάνυσμα ονομάζεται ατομικό (atomic).

Το επόμενο παράδειγμα στοχεύει να δώσει μια πρώτη γεύση της ευκολίας με την οποία η R δημιουργεί διανύσματα και τα χρησιμοποιεί σε πράξεις, συναρτήσεις κλπ.:

Δοκιμάστε:	**Σχόλιο**
`x<-c(4,0,100)`	Ανάθεση ονόματος x σε νέο διάνυσμα που περιέχει τρείς αριθμούς (numeric).
`x<-new("numeric",4,0,100)`	Ίδιο με το παραπάνω.
`length(x)`	Το «μήκος», δηλαδή ο αριθμός των στοιχείων στο x (επιστρέφει 3).

[100] Ασχέτως τύπου μπορούν να περιέχουν και την τιμή NA (η οποία είναι τύπου logical).

[101] Επιβεβαιώστε π.χ. ότι τα numeric βασίζονται στο vector δοκιμάζοντας εντολές όπως π.χ. extends("numeric","vector") ή is(3, "vector"), όπου και οι δύο επιστρέφουν αληθές (TRUE).

`is.numeric(x)`	Είναι το x numeric; Είναι[102], άρα επιστρέφει TRUE.
`x+1/2`	Στο διάνυσμα x (δηλαδή σε κάθε στοιχείο του) να προστεθεί[103] το 1/2.
`sqrt(x)`	Υπολογίζει την τετραγωνική ρίζα του x (δηλαδή κάθε στοιχείου του).
`1/x`	Διαιρεί το 1 με το διάνυσμα x (δηλαδή με κάθε στοιχείο του).

Σε καθεμία από τις τελευταίες τρείς εντολές το αποτέλεσμα είναι επίσης διάνυσμα 3 στοιχείων. Παρατηρήστε πως στο 1/x του παραπάνω παραδείγματος το αποτέλεσμα είναι:

```
> 1/x
[1] 0.25  Inf 0.01
```

Εδώ, ένα από τα αποτελέσματα των πράξεων στα στοιχεία προέκυψε να είναι ειδική τιμή (Inf)[104] και αυτή περιέχει το διάνυσμα αποτελεσμάτων στην αντίστοιχη θέση. Στη συνέχεια του βιβλίου, αποτελέσματα όπως το παραπάνω θα το γράφουμε με την ισοδύναμη εντολή, δηλαδή εδώ ως c(0.25, Inf, 0.01)[105]. Οι συνήθεις αριθμητικές πράξεις ανάμεσα σε διανύσματα (εφόσον ο τύπος τους υποστηρίζει αριθμητικές πράξεις) γίνονται ανάμεσα στα αντίστοιχα στοιχεία:

Δοκιμάστε:	**Σχόλιο**
`d1<-c(10,20)`	Ανάθεση ονόματος d1 σε διάνυσμα αριθμών (numeric) με δύο στοιχεία.
`d2<-c(2,4,6,8)`	Ανάθεση ονόματος d2 σε διάνυσμα αριθμών (numeric) με τέσσερα στοιχεία
`d3<-c(1,2)`	Ανάθεση ονόματος d3 σε διάνυσμα αριθμών (numeric) με δύο στοιχεία.
`d1+d3`	Πρόσθεση (ανά στοιχείο) 2 διανυσμάτων, επιστρέφει το διάνυσμα c(12,22).
`d1+d2`	Πρόσθεση (ανά στοιχείο) 2 διανυσμάτων, επιστρέφει το διάνυσμα c(12,24,16,28).
`d1*d3`	Πολλαπλασιασμός (ανά στοιχείο) 2 διανυσμάτων, επιστρέφει c(10,40).
`d1%*%d3`	Εσωτερικό γινόμενο, επιστρέφει d1[1]*d3[1]+d1[2]*d3[2] άρα 50[106].

Στο παράδειγμα, το d1+d3 επέστρεψε διάνυσμα με 2 στοιχεία τα οποία προκύπτουν από την άθροιση των αντίστοιχων στοιχείων (element-wise) στα d1 και d3. Το d1+d2 επέστρεψε διάνυσμα με 4 στοιχεία. Εδώ η R εφάρμοσε αυτό που ονομάζει «ανακύκλωση τιμών» (recycling). Εφόσον το μήκος του μεγαλύτερου (εδώ του d2 με μήκος 4) είναι ακέραιο πολλαπλάσιο του μήκους του μικρότερου (εδώ το d1 με μήκος 2), η R χρησιμοποιεί πολλές φορές το ίδιο (μικρότερο) διάνυσμα στην εκτέλεση της εντολής. Άρα στο d1+d2, στα δύο πρώτα στοιχεία του (μεγαλύτερου) d2 προστέθηκαν τα δύο του d1, μετά στα επόμενα δύο του d2 προστέθηκαν τα δύο του d1 και ούτω καθεξής. Έτσι το αποτέλεσμα του d1+d2 είναι το διάνυσμα c(d1[1]+d2[1], d1[2]+d2[2], d1[1]+d2[3], d1[2]+d2[4]). Αντίστοιχα έγινε και ο υπολογισμός της τιμής του d1*d3 που επιστρέφει c(d1[1]*d3[1], d1[2]*d3[2]), αλλά εδώ τα μήκη είναι ίδια άρα δεν γίνεται «ανακύκλωση». Τέλος, ο τελεστής %*% υπολογίζει το εσωτερικό γινόμενο (dot product) δύο συμβατού μήκους διανυσμάτων[107], ενώ ο τελεστής %/% το εσωτερικό πηλίκο (άθροισμα της διαίρεσης των επιμέρους στοιχείων). Προφανώς, στα διάφορα πακέτα (ενσωματωμένα ή που έχουν προστεθεί από τον χρήστη) υπάρχει τεράστιος αριθμός από συναρτήσεις που αφορούν την επεξεργασία διανυσμάτων. Ως ενδεικτικό παράδειγμα (που είναι συνέχεια του προηγούμενου), υποστηρίζονται πράξεις συνόλων:

Δοκιμάστε:	**Σχόλιο**
`union(d2,d3)`	Ένωση των d2 και d3, επιστρέφει c(2, 4, 6, 8, 1)
`intersect(d2,d3)`	Τομή, τα κοινά στοιχεία των d2 και d3, επιστρέφει c(2)
`setdiff(d2,d3)`	Διαφορά, τα στοιχεία του d2 που δεν υπάρχουν στο d3, επιστρέφει c(4, 6, 8)

Εκτός της συνάρτησης c, για δημιουργία διανυσμάτων που περιέχουν ακολουθίες αριθμών χρησιμοποιείται συχνά ο τελεστής ‘:’ ο οποίος επιστρέφει ως διάνυσμα την κανονική ακολουθία τιμών (τύπου integer αν ξεκινά από ακέραια τιμή) σε αύξουσα (κατά βήμα 1) ή μειούμενη (κατά βήμα -1) σειρά, για παράδειγμα:

---

[102] Αυτή τη στιγμή το διάνυσμα βρίσκεται σε numeric mode και έτσι είναι κλάσης "numeric".
[103] Η διαίρεση έχει προτεραιότητα από την πρόσθεση και γίνεται πρώτη, βλ. help(Syntax).
[104] βλ. §2.2.4 Ειδικές τιμές.
[105] Είναι το αποτέλεσμα της εντολής dput(1/x), δηλαδή ο κώδικας που αναπαράγει το αντικείμενο αυτό.
[106] Το αποτέλεσμα είναι πίνακας 1x1, μπορεί να μετατραπεί σε αριθμό με τη συνάρτηση drop, βλ.§4.1.3.2 Ο τύπος array.
[107] Ή εκτελεί πολλαπλασιασμό μήτρας αν γίνει ανάμεσα σε δυο πίνακες με συμβατές διαστάσεις, βλ.§4.1.3 Πίνακες (matrix και array).

Δοκιμάστε:	Σχόλιο
`y<-10:30`	Ανάθεση ονόματος y σε διάνυσμα των ακεραίων από το 10 έως το 30 με βήμα 1.

Το διάνυσμα y περιέχει 21 ακεραίους και είναι αντικείμενο τύπου integer με μήκος 21, κάτι που θα επιστρέψει και η εντολή length(y). Αν ανακληθεί το ίδιο το αντικείμενο (γράφοντας y) θα εμφανιστεί το παρακάτω:

```
> y
[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
[19] 28 29 30
```

Οι αριθμοί μέσα σε αγκύλες στην αρχή κάθε γραμμής είναι ο δείκτης του στοιχείου που ακολουθεί. Έτσι το [1] δείχνει πως το 10 (αμέσως μετά) είναι το $1^{ο}$ στοιχείο στο y, ενώ αντίστοιχα το [19] στην δεύτερη γραμμή δείχνει πως το 28 που ακολουθεί είναι το $19^{ο}$. Επειδή η έξοδος προσαρμόζεται στο μέγεθος του παραθύρου, η διάταξη στην οθόνη σας (και οι αριθμοί μέσα στις αγκύλες) ίσως διαφέρουν από το παραπάνω, όμως τα εμφανιζόμενα στοιχεία του y θα είναι τα ίδια. Οι δείκτες δίνουν πρόσβαση στο στοιχείο που βρίσκεται σε συγκεκριμένη θέση στο διάνυσμα[108]. Η αρίθμηση των δεικτών ξεκινά από τον αριθμό 1, ο οποίος αντιστοιχεί στο πρώτο στοιχείο του διανύσματος[109]. Έτσι, π.χ. το στοιχείο στην $1^{η}$ θέση του y (ή αλλιώς το πρώτο στοιχείο του y) γράφεται ως y[1] ενώ αυτό στην $5^{η}$ θέση ως y[5] κλπ.

Δοκιμάστε:	Σχόλιο
`y[1]+1`	Το στοιχείο στην 1η θέση του y συν τον αριθμό 1.
`y[1]<-20`	Το στοιχείο στην 1η θέση του y να γίνει 20.
`y[2]<-y[21]`	Το στοιχείο στην 2η θέση του x να γίνει ίσο με το αυτό στην 21η θέση.

Μπορεί επίσης να γίνει προσδιορισμός δεικτών με κριτήρια ή με διανύσματα που προσδιορίζουν πολλαπλές θέσεις, επιτρέποντας έτσι ταυτόχρονη πρόσβαση σε πολλά στοιχεία του διανύσματος. Συνεχίζοντας από το παραπάνω:

Δοκιμάστε:	Σχόλιο
`y[5:8]`	Τα στοιχεία στις θέσεις 5 έως 8 του y.
`y[5:8]<-100`	Τα στοιχεία στις θέσεις 5 έως 8 του y να πάρουν την τιμή 100.
`y[-5:-8]`	Όλα τα στοιχεία στο y εκτός αυτών στις θέσεις 5 έως 8.
`y[-5:-8]<-2`	Όλα τα στοιχεία στο y εκτός αυτών στις θέσεις 5 έως 8 να πάρουν την τιμή 2.
`y[c(1,4,10)]`	Τα στοιχεία στις θέσεις 1, 4, και 10 του y.
`y[-c(1,4,10)]`	Όλα τα στοιχεία στο y εκτός αυτών στις θέσεις 1, 4, και 10.
`y[c(1,4,10)]<-c(9,4,6)`	Τα στοιχεία στις θέσεις 1, 4, και 10 του y να πάρουν τιμές 9, 4 και 6 αντίστοιχα.
`y[c(1,4,10)]<-6`	Τα στοιχεία στις θέσεις 1, 4, και 10 του y να πάρουν την τιμή 6.
`y[y>4]`	Τα στοιχεία του y που περιέχουν τιμή μεγαλύτερη του 4.
`y[y>4]<-40`	Όσα στοιχεία του y περιέχουν τιμή μεγαλύτερη του 4 να πάρουν την τιμή 40.
`y[y<=4]<-y[y<=4]/2`	Όσα στοιχεία του y περιέχουν τιμή μικρότερη ή ίση του 4 να διαιρεθούν δια 2.

Όταν στα παραπάνω παραδείγματα χρησιμοποιούνται κριτήρια για την επιλογή στοιχείων, όπως π.χ. στο y[y>4], αυτό που περνά στον τελεστή [] είναι ένα διάνυσμα λογικών τιμών (TRUE ή FALSE, βλ. §2.5 Λογικές πράξεις, συγκρίσεις και ο βασικός τύπος logical) με την τιμή TRUE να αντιστοιχεί σε θέση στοιχείου που θα επιλεγεί. Έτσι, στο y[y>4] η σύγκριση (y>4) επιστρέφει ένα διάνυσμα λογικών τιμών με TRUE στις θέσεις που το στοιχείο έχει τιμή >4. Μετά την εκτέλεση των εντολών του παραδείγματος, το y είναι:

```
> y
 [1] 40 1 1 40 40 40 40 40  1 40 1 1 1 1 1 1 1 1 1 1 1
```

Χρήσιμες συναρτήσεις σε πολυμελή αντικείμενα είναι οι **which, match** και ο τελεστής **%in%**:. Συνεχίζοντας από τα προηγούμενα:

Δοκιμάστε:	Σχόλιο
`which(y>10)`	Ποια στοιχεία είναι > 10; Επιστρέφει τις θέσεις τους ως c(1, 4, 5, 6, 7, 8,10).

[108] βλ. για τους διάφορους τελεστές επιλογής στοιχείων προς εξαγωγή ή αντικατάσταση, βλ. help(Extract).

[109] Σε πολλές άλλες γλώσσες προγραμματισμού η αρίθμηση δεικτών ξεκινά από το 0.

`c(1,0,40) %in% y`	Υπάρχουν τα 1,0 και 40 στο y; Επιστρέφει τις απαντήσεις c(T, F, T).
`match(c(1,0,40),y)`	Που πρωτοεμφανίζονται τα 1,0 και 40; Επιστρέφει τις θέσεις c(2, NA, 1).

Τα παραπάνω δίνουν μια πρώτη επαφή με τα διανύσματα, ενώ ακολουθεί ένα παράδειγμα χρήσης τους. Παρόμοια σύνταξη με τα διανύσματα θα βρούμε να χρησιμοποιούνται και σε άλλους τύπους αντικειμένων που βασίζονται σε vector. Τέτοιοι τύποι παρουσιάζονται στο κεφάλαιο 4. Στο ίδιο κεφάλαιο θα βρείτε περισσότερα και για τα ίδια τα ατομικά (atomic) διανύσματα (§4.1.1 Ο τύπος vector (διάνυσμα)) καθώς και για τα μη-ατομικά διανύσματα list (§4.2.1 Ο τύπος list (λίστα)) που υποστηρίζουν την αποθήκευση στοιχείων διαφορετικού τύπου.

2.4.2 Δύο παραδείγματα χρήσης vector και η σημασία της ειδικής τιμής NA

Το vector είναι ο πρώτος τύπος αντικειμένου που παρουσιάζεται στο βιβλίο αυτό και μπορεί να περιέχει πολλά δεδομένα. Αυτό τα κάνει ιδιαίτερα χρήσιμα. Ακολουθούν δύο απλά παραδείγματα που χρησιμοποιούν αντικείμενα τύπου vector και κάποιες βασικές σχετικές συναρτήσεις.

Παράδειγμα 1ο: Μία επένδυση κατανέμει ένα ποσό (140000) σε τέσσερα διαφορετικά επενδυτικά προϊόντα. Το καθένα από τα τέσσερα επενδυτικά προϊόντα έχει διαφορετικό (σταθερό) επιτόκιο για μία περίοδο, το οποίο είναι 0.3%, 1.1%, 0.5% και 2.5%, αντίστοιχα. Σε καθένα από τα τέσσερα προϊόντα θα επενδυθούν τα ποσά 40000, 30000, 10000 και 60000, αντίστοιχα. Ποιο θα είναι το τελικό ποσό στο τέλος μίας περιόδου;

Δοκιμάστε:	**Σχόλιο**
`e <- c(0.3, 1.1, 0.5, 2.5) / 100`	Το επιτόκιο ανά επένδυση.
`p <- c(40000, 30000, 10000, 60000)`	Τα ποσά που θα τοποθετηθούν στην αντίστοιχη επένδυση.
`a <- p * e`	Οι αποδόσεις ανά επένδυση, εδώ c(120, 330, 50, 1500)
`teliko  <- sum( p + a )`	Το σύνολο των αρχικών ποσών συν τις αποδόσεις, εδώ 142000.

Τα δύο βασικά διανύσματα είναι στις μεταβλητές e (επιτόκιο ανά επένδυση) και p (με τα ποσά ανά επένδυση). Ο πολλαπλασιασμός τους δίνει ένα διάνυσμα αποδόσεων ανά επένδυση, συγκεκριμένα το c(120, 330, 50, 1500), που ονομάστηκε a. Το άθροισμά των p + a είναι επίσης διάνυσμα (συγκεκριμένα το c(40120, 30330, 10050, 61500)). Η συνάρτηση sum αθροίζει όλες τις τιμές του διανύσματος αυτού και επιστρέφει 142000 στη μεταβλητή teliko.

Παράδειγμα 2ο: Πέραν του πιθανού μαθηματικού νοήματος, στον προγραμματισμό τα διανύσματα χρησιμοποιούνται συχνά για να τοποθετούνται σε αυτά δεδομένα που έχουν κάποιον κοινό χαρακτήρα. Ας θεωρήσουμε λοιπόν πως μία ομάδα αγροτών καταγράφει ανά μήνα το ύψος βροχής (την ποσότητα νερού που φτάνει στο έδαφος). Έχουν εγκαταστήσει 5 σταθμούς καταγραφής της βροχής σε διάφορες θέσεις στην περιοχή. Το ύψος βροχής καταγράφεται με μονάδα μέτρησης τα χιλιοστά (mm) βροχής. Αυτόν τον μήνα κατέγραψαν τα παρακάτω δεδομένα βροχής, τα οποία τοποθετήθηκαν σε ένα διάνυσμα με όνομα r (το διάνυσμα έχει 5 στοιχεία που αντιστοιχούν στους 5 σταθμούς καταγραφής). Κοινό για όλους τους αριθμούς στο διάνυσμα είναι πως αφορούν καταγραφές του ύψους της βροχής σε συγκεκριμένα σημεία καταγραφής:

Δοκιμάστε:	**Σχόλιο**
`r<-c(0,5,8,0,0)`	Τα δεδομένα ανά σταθμό (χιλιοστά βροχής).
`r[5]<-3`	Αλλαγή των δεδομένων του 5ου σταθμού σε 3.
`r`	Τα αλλαγμένα δεδομένα ανά σταθμό, δηλαδή το διάνυσμα c(0, 5, 8, 0, 3).
`r[2]/25.4`	Τα δεδομένα στον 2ο σταθμό σε ίντσες βροχής (1 inch = 25.4 mm): 0.1968504
`r/100`	Τα δεδομένα ανά σταθμό σε εκατοστά βροχής: c(0.00, 0.05, 0.08, 0.00, 0.03).
`r[2:4]`	Τα δεδομένα των σταθμών 2 έως 4: c(5, 8, 0).

Εδώ έχουμε ένα κλασσικό πρόβλημα όπου θα ήταν χρήσιμο να ονομάσουμε τα στοιχεία, να βάλουμε δηλαδή μια ετικέτα σε κάθε μέτρηση, (π.χ. κάποιο όνομα ενδεικτικό της θέσης του σταθμού καταγραφής). Στα αντικείμενα vector αυτό το επιτρέπει η συνάρτηση **names** που δίνει πρόσβαση σε (προαιρετικό) διάνυσμα ονομάτων που είναι οι ετικέτες για τα δεδομένα, π.χ.:

```
names(r)<-c("Λόφος","Δρόμος","Αποθήκη","Κέντρο","Βρύση")
```

το οποίο δίνει την ονομασία «Λόφος» στα δεδομένα του 1ου σταθμού, «Δρόμος» στα δεδομένα του 2ου κλπ. Θα μπορούσαμε να έχουμε χρησιμοποιήσει ονόματα εξρχής κατά τη δημιουργία του διανύσματος γράφοντας:

```
r<-c(Λόφος=0,Δρόμος=5,Αποθήκη=8,Κέντρο=0,Βρύση=0)
```

Σε κάθε περίπτωση, τα ονόματα βοηθούν στην εμφάνιση των αποτελεσμάτων με πιο κατανοητό τρόπο σε λεκτικές ή άλλες μορφές εξόδου αποτελεσμάτων (όπως τα γραφήματα). Έτσι, π.χ. αν ζητήσουμε τα δεδομένα των σταθμών 2 έως 4 (r[2:4]) έχοντας κάνει τον παραπάνω ορισμό με ονόματα, το αποτέλεσμα (που παραμένει απλώς ένα διάνυσμα) θα εμφανιστεί ως:

```
> r[2:4]
 Δρόμος Αποθήκη  Κέντρο
      5       8       0
```

Επιπρόσθετα οι ετικέτες των στοιχείων μπορούν πλέον να χρησιμοποιηθούν ως δείκτες:

Δοκιμάστε:	**Σχόλιο**
`names(r)[3]<-"Δυτικός"`	Αλλαγή του ονόματος του 3ου σταθμού καταγραφής σε «Δυτικός».
`r["Λόφος"]`	Τα δεδομένα του (1ου) σταθμού με όνομα «Λόφος».
`r["Βρύση"]<-3`	Τα δεδομένα του (5ου) σταθμού με όνομα «Βρύση» να γίνουν 3.
`r[c("Δένδρο", "Λόφος")]`	Τα δεδομένα για αυτά τα ονόματα. «Δένδρο» δεν υπάρχει, άρα NA.

Ας δούμε τώρα τι συμπεράσματα μπορεί να εξάγουμε από τα δεδομένα αυτά μέσω μερικών απλών συναρτήσεων πάνω στο διάνυσμα r (μεταξύ αυτών και κάποιων βασικών συναρτήσεων περιγραφικής στατιστικής[110]). Τα συμπεράσματα που θα προκύψουν μπορεί να φαίνονται απλοϊκά καθώς τα δεδομένα είναι λίγα, αλλά θα είχαν μεγαλύτερο ενδιαφέρον και αν ο αριθμός των δεδομένων (εδώ σταθμών καταγραφής) ήταν μεγάλος.

Τα τρέχοντα δεδομένα του διανύσματος r μετά τα παραπάνω βήματα απεικονίζονται σε γράφημα στηλών στην Εικόνα 2.1. Το γράφημα αυτό δημιουργήθηκε με την εντολή **barplot**(r)[111]. Ακολουθούν κάποιες εντολές επεξεργασίας των δεδομένων αυτών[112]:

Δοκιμάστε:	**Σχόλιο**
`sum(r)`	Άθροισμα των στοιχείων του r, η συνολική βροχή που καταγράφηκε: 16 mm.
`mean(r)`	Αριθμητικός μέσος όρος. Οι σταθμοί κατέγραψαν 3.2 mm κατά μέσο όρο.
`median(r)`	Διάμεσος. Οι μισοί σταθμοί κατέγραψαν 3 mm ή λιγότερο.
`min(r)`	Ελάχιστο. Η λιγότερη βροχή που καταγράφηκε σε κάποιον σταθμό ήταν 0 mm.
`max(r)`	Μέγιστο. Η περισσότερη βροχή που καταγράφηκε σε κάποιον σταθμό ήταν 8 mm.

Άλλες συναφείς συναρτήσεις που θα μπορούσαν να χρησιμοποιηθούν είναι οι **median** (διάμεσος), **quantile** (τεταρτημόριο) και **summary** (σύνοψη)[113].

Δοκιμάστε:	**Σχόλιο**
`rev(r)`	Τα στοιχεία του r σε αντεστραμμένη θέση.
`order(r,decreasing=T)`	Η ταξινόμηση θέσεων ως προς την τιμή (3,2,5,1,4, το 1ο είναι στη θέση 3)
`unique(r)`	Οι μοναδικές τιμές που κατέγραψαν οι σταθμοί, δηλαδή 0, 5, 8 και 3 mm.
`table(r)`	Πόσες φορές καταγράφηκε κάθε τιμή: 2 φορές το 0, 1 φορά οι υπόλοιπες.
`sort(r,decreasing=T)`	Οι καταγραφές από τη μεγαλύτερη στη μικρότερη: 8, 5, 3, 0 και 0 mm.
`r[rev(order(r))]`	Ίδιο με το προηγούμενο, καταγραφές από τη μεγαλύτερη στη μικρότερη.
`sort(r,decreasing=T)[1:3]`	Οι 3 μεγαλύτερες καταγραφές: 8, 5, 3 mm.
`rank(r,ties.method ="min")`	Σειρά κατάταξης, π.χ. ο «Δυτικός» στη μεγαλύτερη (5η) θέση κατάταξης.
`which(r>4)`	Ποια στοιχεία είναι πάνω από 4 mm; Το 2ο και το 3ο στοιχείο του r.
`which.max(r)`	Ένα στοιχείο που έχει τη μέγιστη τιμή; Το 3ο («Δυτικός»).
`which(r==min(r))`	Ποια στοιχεία έχουν την ελάχιστη τιμή; Το 1ο και το 4ο («Κέντρο»).

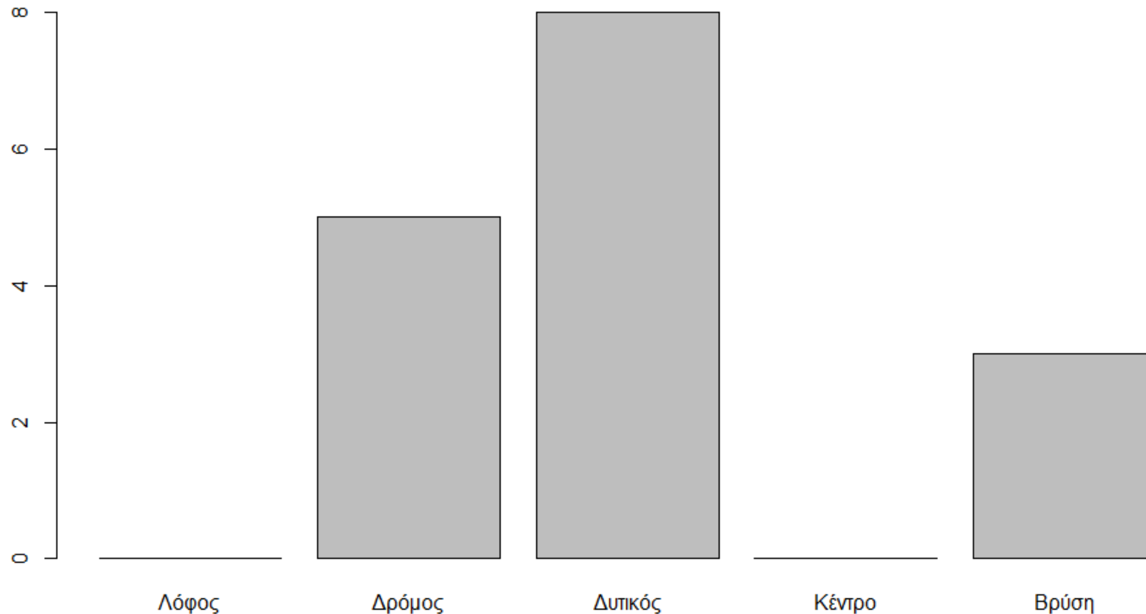
[110] Από τα προ-εγκατεστημένα πακέτα 'base' και 'stats'.

[111] Η εντολή barplot παρέχεται από το προ-εγκατεστημένο πακέτο 'graphics'.

[112] Οι συναρτήσεις sum, mean, median (και πολλές άλλες) είναι generic, οπότε εδώ εφαρμόζεται η υλοποίηση τους για vector. Ο τρόπος λειτουργίας των generic συναρτήσεων περιγράφεται στο Κεφ. 6 και ειδικότερα στην §6.3 Κλάσεις S3.

[113] Για περισσότερα σχετικά με τη summary βλ.§2.7 Οι συναρτήσεις-βοηθήματα: str, summary και dput.

`r[r==max(r)]`	Τα στοιχεία του r με μέγιστη τιμή: Εδώ μόνο ένα, το «Δυτικός».



Εικόνα 2.1 *Γράφημα στηλών των δεδομένων από τους σταθμούς καταγραφής βροχής.*

Συνεχίζοντας το παράδειγμα, ας θεωρήσουμε πως οι σταθμοί 2 ("Δρόμος") και 3 ("Δυτικός") είχαν βλάβη τον προηγούμενο μήνα. Πως θα μπορούσαν να καταγραφούν οι τιμές βροχής του προηγούμενο μήνα σε ένα διάνυσμα με όνομα r0 έτσι ώστε να είναι συγκρίσιμες με τα δεδομένα του τρέχοντος μηνός (στο r); Μια λανθασμένη προσέγγιση θα ήταν να μπει η τιμή 0 στις αντίστοιχες θέσεις, π.χ. αν εναπομείναντες σταθμοί που λειτουργούσαν, δηλαδή οι 1^{ος}, 4^{ος} και 5^{ος} είχαν καταγράψει 1, 2 και 6 mm βροχής αντίστοιχα να δημιουργήσουμε το διάνυσμα r0 ως:

```
r0<-c(1,0,0,2,6)
```

Ή, όπως συμβαίνει κάποιες φορές κατά την καταχώρηση δεδομένων, θα μπορούσε να εισαχθεί μια προσυμφωνημένη αρνητική τιμή για τους σταθμούς που δεν λειτουργούσαν (π.χ. -9999 ή -1):

```
r0<-c(1,-1,-1,2,6)
```

Αρνητικό ύψος βροχής δεν μπορεί να υπάρξει, άρα είναι προφανές ότι αυτές οι τιμές είναι άκυρες, σωστά; Λάθος! Το λάθος οφείλεται στο ότι οι τιμές αυτές δεν παύουν να είναι αριθμοί και εύκολα μπορεί κάποιος να παραβλέψει ότι καταχωρήθηκαν υπό ειδικές συνθήκες (συμβολίζουν μη διαθέσιμα δεδομένα από τους χαλασμένους σταθμούς). Ειδικά αν επρόκειτο για μεγάλο αριθμό δεδομένων και όχι μόνο 5 που έχει το συγκεκριμένο παράδειγμα, ένα τέτοιο λάθος θα ήταν αρκετά πιο πιθανό. Και έτσι τα δεδομένα αυτά θα μπορούσαν να παρεισφρήσουν μέσα στην επεξεργασία και να παράγουν λανθασμένα αποτελέσματα και συμπεράσματα. Για παράδειγμα, αν θέλουμε να μελετήσουμε τη διαφορά βροχόπτωσης αυτού του μήνα (στο r και ίσο με c(0, 5, 8, 0, 3)) με τη βροχόπτωση του προηγούμενου μήνα (στο r0), τα παρακάτω αποτελέσματα είναι λάθος, όπως λάθος θα είναι και τα συμπεράσματα που θα εξάγουμε από αυτά:

Δοκιμάστε:	**Σχόλιο**
`r0<-c(1,-1,-1,2,6)`	Δεδομένα προηγούμενου μήνα, με -1 στους σταθμούς εκτός λειτουργίας.
`r-r0`	Η διαφορά βροχής ανά σταθμό, επιστρέφει c(-1, 6, 9, -2, -3) και όχι την πραγματική.
`barplot(r-r0)`	Το παραπάνω αποτέλεσμα ως γράφημα στηλών, προφανώς επίσης λανθασμένο.
`sum(r-r0)`	Ποια η συνολική διαφορά βροχής από τον προηγούμενο; Επιστρέφει 9, λάθος.

Πιο ασφαλής και ορθός τρόπος να καταγραφεί η απουσία κάποιας καταγραφής, ότι δηλαδή κάποιο δεδομένο δεν είναι διαθέσιμο, είναι με την ειδική τιμή NA. Η ύπαρξη NA σε ατομικές δομές όπως τα διανύσματα δεν αλλάζει το mode τους και έτσι η επεξεργασία μπορεί να συνεχιστεί. Όμως πράξεις που ενδεχομένως γίνουν με NA θα επιστρέψουν NA, καθώς δεν είναι δυνατόν να υπάρχει σαφές αποτέλεσμα με ελλιπή δεδομένα:

Δοκιμάστε:	**Σχόλιο**
`r0<-c(1,NA,NA,2,6)`	Δεδομένα προηγούμενου μήνα, με NA στους σταθμούς εκτός λειτουργίας.
`r-r0`	Η διαφορά βροχής ανά σταθμό, επιστρέφει c(-1, NA, NA, -2, -3), σωστό.
`barplot(r-r0)`	Το παραπάνω αποτέλεσμα ως γράφημα στηλών (ορθότερο από το προηγούμενο).
`sum(r-r0)`	Η συνολική διαφορά βροχής δεν μπορεί να υπολογιστεί, άρα NA (σωστά).

2.4.2.1 Χειρισμός ειδικών τιμών (NA, NaN κλπ.)

Γενικά η παρουσία τιμών NA / NaN κλπ. σε δεδομένα είναι ένα πρόβλημα, καθώς συμβολίζουν ασάφεια. Όμως συχνά είναι αναπόφευκτη. Συνεχίζοντας στην ίδια ιδέα με το προηγούμενο παράδειγμα, ας θεωρήσουμε πως θέλουμε να επεξεργαστούμε δύο διανύσματα που καταγράφουν επίπεδα βροχής, τα rA και rB ως εξής:

```
> rA
  Λόφος  Δρόμος Αποθήκη  Κέντρο   Βρύση
     NA       5      NA       4       1
> rB
  Λόφος  Δρόμος Αποθήκη  Κέντρο   Βρύση
      6       3      NA      NA       7
```

Τα δεδομένα περιέχουν τιμές NA. Είναι σύνηθες φαινόμενο τα δεδομένα που επεξεργαζόμαστε να περιλαμβάνουν τέτοιες τιμές για κάτι που απλώς δεν μπορεί να προσδιοριστεί. Όπως αναφέρθηκε παραπάνω, πράξεις και συγκρίσεις με NA θα επιστρέψουν NA, δηλαδή ασαφές αποτέλεσμα. Έτσι, οι αναλυτές των δεδομένων καλούνται να εφαρμόσουν μια στρατηγική χειρισμού των στοιχείων με τέτοιες τιμές. Μια συνήθης στρατηγική είναι να αγνοηθούν οι τιμές αυτές όπου αυτό είναι εφικτό και έχει νόημα:

Δοκιμάστε:	**Σχόλιο**
`sum(rA,na.rm=T)`	Συνολική βροχή στο rA, για όπου υπάρχουν διαθέσιμα δεδομένα, επιστρέφει 10.
`sum(na.omit(rA))`	Ίδιο με το παραπάνω.
`sum(rB-rA,na.rm=T)`	Συνολική διαφορά rB από rA, όπου υπάρχουν διαθέσιμα δεδομένα, επιστρέφει 4.
`complete.cases(rA,rB)`	Για ποιες θέσεις υπάρχουν πλήρη δεδομένα; Επιστρέφει c(F,T,F,F,T)[114].
`rA[!is.na(rA)]`	Τα στοιχεία του rA που δεν είναι NA, επιστρέφει c(5,4,1)[115].
`na.omit(rA)`	Ίδιο (ή ορθότερα παρεμφερές) με το παραπάνω.

Πολλές από τις μαθηματικές και στατιστικές συναρτήσεις που έρχονται με την R υποστηρίζουν την παράμετρο na.rm (NA remove/διαγραφή τυχόν NA και NaN). Έτσι, ορίζοντας σε τέτοιες συναρτήσεις την παράμετρο na.rm=T (όπως στη sum στο παραπάνω παράδειγμα) τυχόν μη έγκυρα δεδομένα δεν χρησιμοποιούνται στους υπολογισμούς. Επιπρόσθετα, συναρτήσεις όπως οι **complete.cases**, **na.omit**, **na.exclude**, **na.fail** και **na.pass** μπορούν να εφαρμοστούν τόσο σε διανύσματα όσο και σε πιο πολύπλοκες δομές (όπως πίνακες matrix ή data.frame) και διευκολύνουν τον χειρισμό NA στα δεδομένα τους. Για παράδειγμα, η complete.cases ελέγχει αν όλα τα δεδομένα κάθε καταγραφής είναι πλήρη (χωρίς NA), η na.omit αφαιρεί τα NA από τα δεδομένα, ενώ η na.fail εγείρει λάθος και σταματά την εκτέλεση αν υπάρχουν NA στα δεδομένα[116]. Τέλος, υπάρχουν και πολλές άλλες στρατηγικές χειρισμού NA, πέραν της απλής απόρριψης των σχετικών δεδομένων. Τέτοιες προσεγγίσεις στοχεύουν στο να αντικαταστήσουν τις τιμές NA με τεχνητές τιμές, υπολογισμένες έτσι ώστε κατά το δυνατόν να μην επηρεάζονται τα αποτελέσματα της όποιας ανάλυσης θα εφαρμοστεί στη συνέχεια. Αυτή η διαδικασία ονομάζεται «απόδοση τιμών» (imputation) και διαφορετικοί

[114] Για τις τιμές F (δηλαδή αναληθές/FALSE) και T (δηλαδή αληθές/TRUE), βλ.§2.5 Λογικές πράξεις, συγκρίσεις και ο βασικός τύπος logical.

[115] Για τον τελεστή ! (NOT) βλ.§2.5.2 Λογικές πράξεις, για τη συνάρτηση is.na βλ.§2.2.4 Ειδικές τιμές.

[116] Υπάρχουν σχετικές επιλογές που ρυθμίζουν τον χειρισμό NA στη συνεδρία με την R (βλ. §4.2.1.5 Η λίστα επιλογών συνεδρίας). Δοκιμάστε π.χ. getOption(“na.action”).

τρόποι να επιτευχθεί παρέχονται από διάφορα πακέτα στο CRAN[117], όπως το πακέτο ‘mice’ [42] και το πακέτο ‘VIM’ [43].

2.5 Λογικές πράξεις, συγκρίσεις και ο βασικός τύπος logical

Στην R, οι πράξεις δυαδικής λογικής (λογικής Bool όπως τα όχι/not, και/and, ή/or κλπ.) επεξεργάζονται αντικείμενα τύπου logical. Υπάρχουν τρείς δεσμευμένες λέξεις που αντιστοιχίζονται στις τιμές που επιτρέπεται να έχουν τέτοια αντικείμενα. Αυτές είναι το TRUE (αληθές, με συντομογραφία το T), το FALSE (αναληθές, με συντομογραφία το F) και το NA (Not Available, δηλαδή μη διαθέσιμο αποτέλεσμα). Οι συντομογραφίες T, F όπως και το NA πρέπει να γράφονται με κεφαλαίους λατινικούς χαρακτήρες. Έχουν ήδη αναφερθεί παραδείγματα όπου χρησιμοποιούνται τέτοιες τιμές σε παραμέτρους συναρτήσεων (π.χ. στη detach) ή επιστρέφονται από αυτές (π.χ. στην is.numeric).

Αν μετατραπούν τιμές logical σε αριθμητικές, το FALSE μετατρέπεται στον αριθμό 0, το TRUE μετατρέπεται σε 1, ενώ το NA δεν μετατρέπεται. Αντίστροφα, ο αριθμός 0 θα γίνει FALSE αν μετατραπεί σε logical, ενώ οποιαδήποτε μη-μηδενική αριθμητική τιμή θα μετατραπεί σε TRUE. Οι συναρτήσεις isTRUE και isFALSE ελέγχουν αν ένα αντικείμενο είναι όντως TRUE ή FALSE αντίστοιχα. Εδώ το αντικείμενο πρέπει να είναι να είναι τύπου logical, μήκους 1 και το (μοναδικό) στοιχείο του να έχει την αντίστοιχη τιμή. Άρα isTRUE(1) επιστρέφει FALSE ενώ isTRUE(as.logical(1)) επιστρέφει TRUE. Αν το αντικείμενο είναι πολυμελές (με μήκος>1) τότε οι δύο αυτές συναρτήσεις επιστρέφουν FALSE ασχέτως των λογικών τιμών οι οποίες περιέχονται στα αντικείμενα που ελέγχουν.

2.5.1 Συναρτήσεις σύγκρισης

Οι τελεστές σύγκρισης[118] επιστρέφουν αντικείμενα τύπου logical. Μπορούν να εφαρμοστούν σε αντικείμενα βασικών (atomic) τύπων (με ένα ή περισσότερα στοιχεία)[119]. Γενικότερα, μπορούν να εφαρμοστούν σε τύπους αντικειμένων (συμπεριλαμβανομένων και νέων κλάσεων αντικειμένων) για τους οποίους έχουν οριστεί μέθοδοι σύγκρισης[120]. Ακολουθούν μερικά παραδείγματα συγκρίσεων:

Δοκιμάστε:	**Σχόλιο**
`1>2`	Είναι το 1 μεγαλύτερο του 2; Επιστέφει FALSE (F).
`x<-1>2`	Ανάθεση σε μεταβλητή x του αποτελέσματος σύγκρισης (αντικείμενο logical).
`1<=2`	Είναι το 1 μικρότερο ή ίσο του 2; Επιστέφει FALSE (F).
`1==2`	Είναι το 1 ίσο με 2; Επιστέφει FALSE (F).
`3-2==2`	Είναι το 3-2 ίσο με 2; Επιστέφει FALSE (F).
`2==3-2`	Είναι το 2 ίσο με 3-2; Επιστέφει FALSE (F).
`1!=2`	Είναι το 1 διάφορο του 2; Επιστέφει FALSE (F).
`c(1,2,3)>c(1,2,2)`	Σύγκριση ανά στοιχείο, επιστρέφει διάνυσμα logical c(FALSE,FALSE,TRUE).
`c(1,3)==c(1,2,1,3)`	Οι τιμές ανακυκλώνονται, επιστρέφει c(TRUE,FALSE,TRUE,TRUE).
`c(1,3)!=c(1,2,1,3)`	Οι τιμές ανακυκλώνονται, επιστρέφει c(FALSE,TRUE,FALSE,FALSE).
`1==c(1,2,1,3)`	Οι τιμές ανακυκλώνονται, επιστρέφει c(TRUE,FALSE,TRUE,FALSE).

Οι τέσσερεις τελευταίες εντολές στο παραπάνω παράδειγμα είναι ενδεικτικές του τρόπου λειτουργίας των τελεστών σύγκρισης σε αντικείμενα με length>1. Σε τέτοια αντικείμενα, τελεστές όπως το == ή το != δεν ελέγχουν αν ολόκληρα τα αντικείμενα είναι ίσα ή διαφέρουν, αλλά συγκρίνουν ανά δύο τα στοιχεία τους. Έτσι η σύγκριση δεν επιστρέφει ένα αποτέλεσμα, μια λογική τιμή, αλλά ένα logical αντικείμενο παρόμοιου τύπου με αυτά που συγκρίνουν (στην περίπτωση αυτή, διανύσματα) που περιέχει πολλές λογικές τιμές οι οποίες αντιστοιχούν στις συγκρίσεις των επιμέρους ζευγαριών τιμών. Στα παραπάνω τέσσερα παραδείγματα εμπλέκονται vectors, το μεγαλύτερο με μήκος 4, οπότε επιστρέφεται επίσης ένα (logical) vector που περιέχει 4 στοιχεία. Όπως και στις αριθμητικές πράξεις, κατά τη σύγκριση εφαρμόζεται ανακύκλωση τιμών εφόσον χρειάζεται.

[117] βλ.§1.5 Χρήση και διαχείριση πακέτων.
[118] βλ.help(Comparison).
[119] βλ.§4.1.1 Ο τύπος vector (διάνυσμα).
[120] βλ.§6.2 Αντικειμενοστραφής προγραμματισμός στην R και help(Compare).

Αντίθετα, υπάρχουν δύο σημαντικές συναρτήσεις οι οποίες όντως ελέγχουν ολόκληρα τα αντικείμενα και επιστρέφουν μια μοναδική λογική τιμή ως απάντηση. Οι συναρτήσεις αυτές είναι η identical και η all.equal[121]. Η identical ελέγχει αν δύο αντικείμενα είναι ταυτόσημα, ενώ η all.equal μπορεί να ελέγξει αν δύο αντικείμενα είναι «περίπου» ίσα, μέσα σε κάποιο ανεκτό όριο διαφοράς τιμών (tolerance):

Δοκιμάστε:	**Σχόλιο**
`identical(3-2,1)`	Είναι το 3-2 ακριβώς ίσο με 1; Επιστέφει TRUE (T).
`identical(c(1,2),c(1,2))`	Είναι τα δύο ακριβώς ίσα; Επιστέφει TRUE (T).
`identical(c(1,2),c(1,2,1,2))`	Είναι τα δύο ακριβώς ίσα; Επιστέφει FALSE (F).
`all.equal(c(1,2),c(1,2))`	Είναι όλες οι τιμές ίσες; Επιστέφει TRUE (T).
`all.equal(c(1,2),c(1,2,1,2))`	Όλες οι τιμές ίσες; Επιστέφει λεκτική περιγραφή της διαφοράς.
`all.equal(1,1.01)`	Όλες οι τιμές ίσες; Επιστέφει λεκτική περιγραφή της διαφοράς.
`all.equal(1,1.01,tolerance=0.1)`	Όλες οι τιμές περίπου ίσες με ανοχή 0.1; Επιστέφει TRUE (T).

Έτσι, η identical είναι ο πλέον κατάλληλος και προτεινόμενος τρόπος να γίνει έλεγχος αν δύο αντικείμενα είναι απολύτως ίδια και είναι χρήσιμη ακόμα και στη σύγκριση μη αριθμητικών αντικειμένων. Για παράδειγμα, η εντολή identical (sqrt, as.environment("package:base")$sqrt) ελέγχει αν το αντικείμενο (συνάρτηση) sqrt είναι ταυτόσημο με αυτό που παρέχει το πακέτο 'base'.

2.5.2 Λογικές πράξεις

Οι συναρτήσεις λογικών πράξεων[122] συνδυάζουν αντικείμενα τύπου logical, επιστρέφοντας επίσης αντικείμενα τύπου logical. Μερικά παραδείγματα τέτοιων συναρτήσεων (πολλές από αυτές έχουν τη μορφή τελεστών) δίνονται παρακάτω:

<table>
<tr><th>Δοκιμάστε:</th><th>Σχόλιο</th></tr>
<tr><td><code>!T</code></td><td>Το ! είναι τελεστής άρνησης («όχι/not»). Επιστρέφει όχι TRUE δηλ. F.</td></tr>
<tr><td><code>!c(T,T,F,F)</code></td><td>Άρνηση σε πολυμελές αντικείμενο. Επιστρέφει c(F,F,T,T).</td></tr>
<tr><td><code>T&&T&&F</code></td><td>Το && είναι το «και/and» για αντικείμενα με ένα στοιχείο. Επιστρέφει F.</td></tr>
<tr><td><code>1<2&&2>3</code></td><td>Είναι το 1 μικρότερο του 2 «και/and» το 2 μεγαλύτερο του 3; Επιστρέφει F.</td></tr>
<tr><td><code>c(T,T,F,F)&&c(T,F,T,F)</code></td><td>Το && σε πολυμελή αντικείμενα ελέγχει το 1ο στοιχείο. Επιστρέφει T.</td></tr>
<tr><td><code>c(T,T,F,F)&c(T,F,T,F)</code></td><td>Το & είναι το «και/and» για πολυμελή αντικείμενα. Επιστρέφει c(T,F,F,F).</td></tr>
<tr><td><code>T||T||F</code></td><td>Το || είναι το «ή/or» για αντικείμενα με ένα στοιχείο. Επιστρέφει T.</td></tr>
<tr><td><code>1<2||2>3</code></td><td>Είναι το 1 μικρότερο του 2 «ή/or» το 2 μεγαλύτερο του 3; Επιστρέφει T.</td></tr>
<tr><td><code>c(T,T,F,F)||c(T,F,T,F)</code></td><td>Το || σε πολυμελή αντικείμενα εφαρμόζεται στο 1ο στοιχείο. Επιστρέφει T.</td></tr>
<tr><td><code>c(T,T,F,F)|c(T,F,T,F)</code></td><td>Το | είναι το «ή/or» για πολυμελή αντικείμενα. Επιστρέφει c(T,F,F,F).</td></tr>
<tr><td><code>xor(c(T,T,F,F),c(T,F,T,F))</code></td><td>Το xor (exclusive or) ελέγχει αν ένα από τα δυο, αλλά όχι και τα 2, είναι T.</td></tr>
</table>

Οι τελεστές **&&** και || είναι σχεδιασμένοι για χρήση σε αντικείμενα που περιέχουν μια μοναδική logical τιμή. Αν χρησιμοποιηθούν σε πολυμελή αντικείμενα (δηλαδή αντικείμενα με length>1) επεξεργάζονται μόνο το πρώτο στοιχείο τους. Η λογική πράξη (AND ή OR, αντίστοιχα) γίνεται μόνο ανάμεσα στα πρώτα στοιχεία των δύο αντικειμένων και επιστρέφει μια μοναδική λογική τιμή ως αποτέλεσμα.

Αντίθετα, οι τελεστές **&** ή | είναι εφαρμόσιμοι τόσο σε μονομελή όσο και σε πολυμελή αντικείμενα. Η λογική πράξη (AND ή OR, αντίστοιχα) γίνεται ανά στοιχείο (όπως οι αριθμητικές πράξεις), ενώ αν χρειάζεται εφαρμόζεται και ανακύκλωση τιμών (βλ.§2.4.1 Οι βασικές εντολές των διανυσμάτων). Επιστρέφουν όμως πολλαπλές απαντήσεις δηλαδή τα logical αποτελέσματα των επιμέρους λογικών πράξεων, αποθηκευμένες σε ένα logical αντικείμενο παρεμφερούς τύπου με αυτά που εμπλέκονται. Έτσι, αν εμπλέκονται δύο διανύσματα, οι τελεστές αυτοί θα επιστρέψουν επίσης logical διάνυσμα που θα περιέχει τα αποτελέσματα των επιμέρους λογικών πράξεων. Για να συνδυαστούν αυτά σε ένα μοναδικό τελικό logical αποτέλεσμα, μπορούν να χρησιμοποιηθούν οι συναρτήσεις any και all:

Δοκιμάστε:	**Σχόλιο**
`all(c(T,F,T))`	Είναι όλα TRUE (T); Δεν είναι, άρα επιστρέφει FALSE (F).

[121] Οι συναρτήσεις της ενότητας αυτής παρέχονται από το προ-εγκατεστημένο πακέτο 'base'.
[122] Οι λογικοί τελεστές περιγράφονται στο help(Logic), η προτεραιότητα εφαρμογής τους στο help(Syntax).

`all(1<2,10<20,1==1)`	Είναι όλα TRUE (T);); Είναι, άρα επιστρέφει TRUE (T).
`any(c(T,F,T))`	Είναι ένα τουλάχιστον TRUE (T); Είναι, άρα επιστρέφει TRUE (T).

Όπως αναφέρθηκε παραπάνω, η κύρια διαφορά των δύο παραλλαγών του AND (του && με το «διανυσματικό» &) και αντίστοιχα των δύο παραλλαγών του OR (|| και |) είναι πως η μακρότερη μορφή (&& και ||) είναι καταλληλότερη για μονομελή αντικείμενα και επιστρέφει ένα μοναδικό αποτέλεσμα ενώ η κοντότερη μορφή (& και |) είναι καταλληλότερη για χρήση σε πολυμελή αντικείμενα και επιστρέφει πολυμελή αντικείμενα αποτελεσμάτων. Σπανίως μπορεί να είναι θεμιτό να χρησιμοποιηθεί η μακρότερη μορφή (&& και ||) σε πολυμελή αντικείμενα, αν *όντως* είναι ζητούμενο να γίνει η λογική πράξη μόνο ανάμεσα στα πρώτα στοιχεία τους. Όμως υπάρχει επίσης και μια περίπτωση που είναι θεμιτό το αντίστροφο, δηλαδή να χρησιμοποιηθούν οι «διανυσματικοί» τελεστές (& και |) σε μονομελή αντικείμενα και οφείλεται στο παρακάτω: τα && και || είναι «αυστηρά» και σταματούν την επεξεργασία μίας έκφρασης όταν μπορεί ήδη να προκύψει το αποτέλεσμα. Για παράδειγμα, στον υπολογισμό του F&&T ο τελεστής && σταματά την επεξεργασία της έκφρασης στο F, χωρίς να επεξεργαστεί (ή να ελέγξει καν) το δεύτερο μέρος της έκφρασης. Αφού υπάρχει ήδη ένα FALSE, το αποτέλεσμα μιας λογικής πράξης AND με οτιδήποτε ακολουθεί θα είναι FALSE. Αυτή η προσέγγιση υλοποίησης λογικών πράξεων ονομάζεται «υπολογισμός με βραχυκύκλωμα» (short-circuit evaluation) και την εφαρμόζουν οι «αυστηροί» τελεστές && και ||. Αντίθετα, οι «διανυσματικοί» τελεστές (& και |) κάνουν πλήρη έλεγχο και επεξεργασία όλων των αντικειμένων που εμπλέκονται στη σύγκριση. Άρα στον υπολογισμό F&T θα ελεγχθούν όλα τα μέρη και θα υπολογιστεί το αποτέλεσμα (που βέβαια θα είναι και πάλι FALSE). Αυτό έχει σημασία όταν οι τιμές που συνδυάζονται πρέπει πρώτα να υπολογιστούν, αφού οι δύο παραλλαγές θα εκτελεστούν με διαφορετικό τρόπο. Ένα παράδειγμα με τις παραλλαγές του OR (|| και |):

<table>
<tr><th>Δοκιμάστε:</th><th>Σχόλιο</th></tr>
<tr><td><code>(1>0)||(1>2)</code></td><td>Επιστρέφει T. Αφού το (1>0) είναι T, το 2° μέρος (1>2) δεν υπολογίζεται.</td></tr>
<tr><td><code>(1>0)||"λάθος"</code></td><td>Επιστρέφει T. Αφού το (1>0) είναι T, το 2° μέρος δεν χρησιμοποιείται.</td></tr>
<tr><td><code>(1>0)|(1>2)</code></td><td>Επιστρέφει T. Το 1>0 γίνεται T, το 1>2 γίνεται F, αποτέλεσμα T OR F = T.</td></tr>
<tr><td><code>(1>0)|"λάθος"</code></td><td>Γίνεται επεξεργασία όλων των αντικειμένων, αδύνατη η επιστροφή τιμής.</td></tr>
</table>

2.5.3 Τι χρησιμεύουν τα αντικείμενα τύπου logical

Η χρησιμότητα των λογικών τιμών είναι ότι συνήθως συμβολίζουν κάτι, έχουν κάποιο νόημα ή σημασία για τον χρήστη ή το πρόγραμμα. Έτσι, οι λογικές πράξεις επιτρέπουν την υλοποίηση κανόνων και κριτηρίων που σχετίζονται με το νόημα αυτό. Ένα απλοϊκό παράδειγμα: ένας αθλητής (π.χ. δρομέας) έχει θέσει ως κριτήριο ότι θα προπονείται τις ημέρες που (α) η θερμοκρασία είναι τουλάχιστον 16 βαθμοί Κελσίου ενώ ταυτόχρονα δεν εργάζεται ή (β) προπονείται η ομάδα του. Κωδικοποιεί λοιπόν τα δεδομένα της επόμενης εβδομάδας (αναμενόμενες θερμοκρασίες, ημέρες που εργάζεται και ημέρες που θα προπονηθεί η ομάδα του) σε τρεις μεταβλητές (καθεμία είναι ένα διάνυσμα με 7 στοιχεία, κάθε στοιχείο τους αντιστοιχεί σε μια ημέρα της εβδομάδας) και υλοποιεί τον κανόνα με λογικές πράξεις για να υπολογίσει ποιες μέρες θα προπονηθεί:

<table>
<tr><th>Δοκιμάστε:</th><th>Σχόλιο</th></tr>
<tr><td><code>εργασία<-c(F,T,T,T,F,T,T)</code></td><td>T επισημαίνει ημέρα που ο αθλητής εργάζεται.</td></tr>
<tr><td><code>θερμοκρασία<-c(15,19,20,19,21,18,16)</code></td><td>Η αναμενόμενη θερμοκρασία ανά ημέρα.</td></tr>
<tr><td><code>ομάδα<-c(T,F,F,F,F,F,T)</code></td><td>T επισημαίνει ημέρα προπόνησης της ομάδας.</td></tr>
<tr><td><code>προπόνηση<-(θερμοκρασία>=16&!εργασία)|ομάδα</code></td><td>Ο κανόνας, T σημαίνει ημέρα προπόνησης.</td></tr>
</table>

Το τελευταίο βήμα παραπάνω (όπου υπολογίζεται το κριτήριο) μπορείτε να το διαβάσετε και ως έναν συνδυασμό από λογικές πράξεις: «(θερμοκρασία μεγαλύτερη ή ίση του 16 και όχι εργασία), ή ομάδα». Αν αλλάξουν τα δεδομένα («εργασία», «θερμοκρασία» ή/και «ομάδα») και υπολογιστεί πάλι το κριτήριο, ίσως προκύψουν αλλαγές και στις τιμές της μεταβλητής «προπόνηση». Για τον αθλητή, η τιμή T στο logical διάνυσμα «προπόνηση», σημαίνει ότι είναι αληθές πως θα προπονηθεί την αντίστοιχη ημέρα, ενώ αν το !any(προπόνηση) επιστρέφει T, σημαίνει πως εκείνη την εβδομάδα δεν θα προπονηθεί. Με τα παραπάνω δεδομένα η μεταβλητή προπόνηση περιέχει:

```
> προπόνηση
[1]  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE
```

Άρα, αν έχουμε ορίσει πως το 1ο στοιχείο αντιστοιχεί στο Σάββατο, ο αθλητής θα προπονηθεί το Σάββατο, την Τετάρτη και την Παρασκευή[123]. Όπως έχει ήδη αναφερθεί, στα διανύσματα, ο τελεστής επιλογής ([]) επιτρέπει την επιλογή στοιχείων μέσω κριτηρίων (βλ.§2.4.1 Οι βασικές εντολές των διανυσμάτων), κάτι που ουσιαστικά είναι logical τιμές:

Δοκιμάστε:	**Σχόλιο**
`θερμοκρασία[προπόνηση]`	Οι θερμοκρασίες τις ημέρες προπόνησης, επιστρέφει c(15, 21, 16).
`mean(θερμοκρασία[προπόνηση])`	Η μέση θερμοκρασία για τις ημέρες προπόνησης επιστρέφει 17.33.

Επίσης, εκμεταλλευόμενοι ότι το ισοδύναμο του TRUE αν μετατραπεί σε numeric είναι η τιμή 1 ενώ το FALSE είναι 0, μπορούν να γίνουν υπολογισμοί όπως οι παρακάτω:

Δοκιμάστε:	**Σχόλιο**
`sum(προπόνηση)`	Πόσες μέρες προπόνησης αυτή την εβδομάδα; Επιστρέφει 3.
`ώρες_ανά_ημέρα<-c(2,2,1,1,2,1,1)`	Αν ανάλογα με την ημέρα οι ώρες διαφέρουν…
`sum(προπόνηση*ώρες_ανά_ημέρα)`	Πόσες ώρες προπόνησης αυτή την εβδομάδα; Επιστρέφει 5.

Εδώ, (στη μεταβλητή ώρες_ανά_ημέρα) καταγράφεται η πληροφορία πως Σάββατα, Κυριακές και Τετάρτες ο αθλητής προπονείται δύο ώρες ενώ τις υπόλοιπες μία. Έτσι με μία εντολή υπολογίζονται οι συνολικές ώρες προπόνησης αυτή την εβδομάδα, ένας υπολογισμός που ίσως απαιτούσε τη χρήση βρόχων[124] και δομών επιλογής[125] σε άλλες γλώσσες προγραμματισμού.

2.6 Τυχαίοι αριθμοί

Τα προγράμματα υπολογιστών είναι αιτιοκρατικά, ντετερμινιστικά (deterministic) συστήματα και τίποτα εντός ενός τέτοιου συστήματος δεν είναι εντελώς τυχαίο. Ότι παράγει ένα πρόγραμμα ως έξοδο (output) είναι αποτέλεσμα της τρέχουσας κατάστασης του και των εισόδων (input) σε αυτό. Όμως σε πολλές περιπτώσεις (στατιστικές εφαρμογές, εφαρμογές εξομοίωσης, παιχνίδια κλπ.) υπάρχει ανάγκη χρήσης τυχαίων αριθμών. Για την παραγωγή τυχαίων (ή καλύτερα ψευδό-τυχαίων) αριθμών χρησιμοποιούνται γεννήτριες τυχαίων αριθμών (random number generator). Μια τέτοια γεννήτρια ξεκινά με κάποια δοθείσα αρχική τιμή (τον επονομαζόμενο σπόρο ή seed) και παράγει αριθμούς που επηρεάζονται από την τιμή αυτή. Για μια συγκεκριμένη τιμή seed η γεννήτρια θα παράγει την ίδια ακολουθία ψευδό-τυχαίων αριθμών. Για το λόγο αυτό η αρχικοποίηση της γεννήτριας συνήθως γίνεται με τιμές που προέρχονται από το εσωτερικό ρολόι πραγματικού χρόνου του υπολογιστή, ελαχιστοποιώντας έτσι την πιθανότητα να αρχικοποιηθεί πάλι η γεννήτρια με την ίδια τιμή seed στο μέλλον.

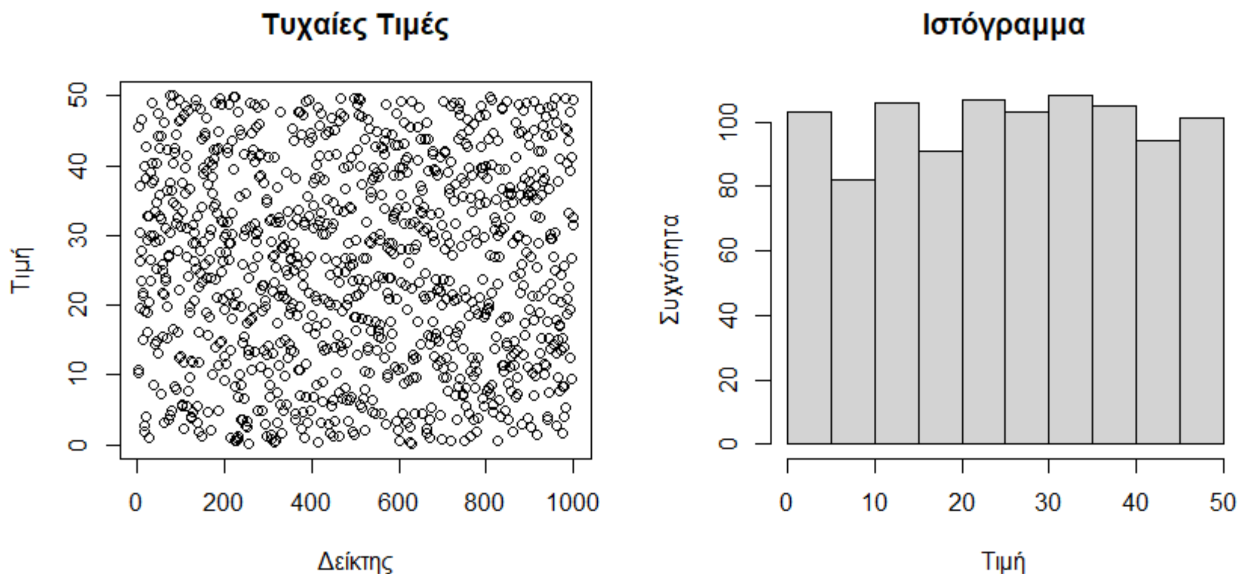
Συνήθως το αρχικό seed της γεννήτριας τυχαίων αριθμών ορίζεται αυτόματα από την R. Εναλλακτικά, η συνάρτηση **set.seed** επιτρέπει την αρχικοποίηση της γεννήτριας με συγκεκριμένη τιμή, έτσι ώστε τα αποτελέσματα να είναι ίδια αν εκτελεστεί πάλι ο κώδικας. Αυτό μπορεί να είναι χρήσιμο για την επιβεβαίωση των αποτελεσμάτων από τρίτους. Έτσι, για να προκύψουν τα ίδια αποτελέσματα στα παρακάτω παραδείγματα τυχαίων αριθμών θα πρέπει πριν εκτελεστούν να δοθεί η εντολή:

```
set.seed(2121)
```

Το πακέτο 'stats' παρέχει πληθώρα συναρτήσεων παραγωγής τυχαίων αριθμών. Οι συναρτήσεις αυτές παράγουν ψευδό-τυχαίες τιμές οι οποίες τείνουν να ακολουθούν μια συγκεκριμένη κατανομή. Έτσι η συνάρτηση **runif** (random uniform) παράγει αριθμούς που τείνουν να ακολουθούν ομοιόμορφη κατανομή (uniform distribution). Για όλες τις πιθανές τιμές στο διάστημα που ορίζεται, η πιθανότητα να παραχθεί κάποια τιμή από τη runif θεωρείται ίδια. Στο παρακάτω παράδειγμα ζητούνται 1000 τυχαίες τιμές στο διάστημα 0 έως 50. Οι τιμές επιστρέφονται ως numeric διάνυσμα 1000 τιμών που εδώ αποθηκεύεται σε μεταβλητή t:

```
t <- runif(1000, min=0, max=50)
```

[123] Θα μπορούσαν να έχουν οριστεί και τα σχετικά ονόματα (ετικέτες) για τα στοιχεία, με τη συνάρτηση names.
[124] βλ.§3.2.2.3 Επαναλήψεις (βρόχοι ή loop).
[125] βλ.§3.2.2.1 Η βασική δομή επιλογής (if και else).



Εικόνα 2.2 Τυχαίες τιμές που παρήχθησαν με τη συνάρτηση runif.

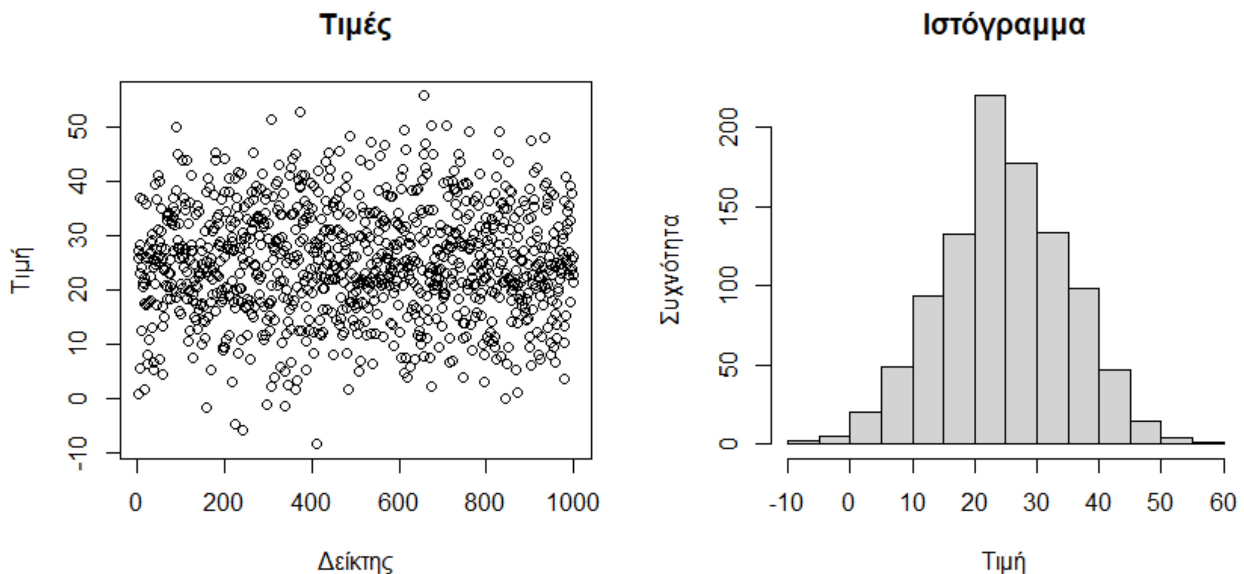
Οι τιμές των στοιχείων του t (βλ. Εικόνα 2.2 - αριστερά) τείνουν όντως να ακολουθούν ομοιόμορφη κατανομή όπως φαίνεται από το ιστόγραμμα (δεξιά) της ίδιας εικόνας[126]. Οι τιμές του t είναι numeric αλλά μπορούν να μετατραπούν σε ακέραιες με συναρτήσεις στρογγυλοποίησης όπως η **floor**.

Οι συναρτήσεις τυχαίων αριθμών που παρέχει το πακέτο 'stats' δίνονται σε τέσσερις, σχετικές μεταξύ τους, παραλλαγές με παρεμφερές όνομα: η συνάρτηση που ξεκινά με το γράμμα r (όπως η runif) επιστρέφει τυχαίες τιμές που ακολουθούν τη συγκεκριμένη κατανομή, οι συναρτήσεις που ξεκινούν με το γράμμα p (όπως η **punif**) επιστρέφουν την αθροιστική κατανομή τους (cumulative distribution) η συνάρτηση που ξεκινά με το γράμμα q (όπως στην **qunif**), αναφέρεται στην ποσοστιαία συνάρτηση (quantile), ενώ όσες συναρτήσεις ξεκινούν με d (όπως η **dunif**) επιστρέφουν την πυκνότητα (density/mass).

Για την παραγωγή τυχαίων αριθμών που τείνουν να βρίσκονται σε κανονική κατανομή (normal distribution) χρησιμοποιείται η συνάρτηση **rnorm** (random normal). Στο επόμενο παράδειγμα παράγονται 1000 τυχαίες τιμές σε κανονική κατανομή με αριθμητικό μέσο όρο 25 και τυπική απόκλιση 10 (τα αποτελέσματα φαίνονται στην Εικόνα 2.3):

```
t<-rnorm(1000, mean=25, sd=10)
```

[126] Οι Εικόνες 2.2. και 2.3 δημιουργήθηκαν με τις εντολές: layout(matrix(c(1,2), nrow=1)); plot(t, main = "Τιμές", xlab = "Δείκτης", ylab = "Τιμή"); hist(t, main = "Ιστόγραμμα", xlab = "Τιμή", ylab = "Συχνότητα").



Εικόνα 2.3 Τυχαίες τιμές που παρήχθησαν με τη συνάρτηση rnorm.

Για την παραγωγή τυχαίων αριθμών σε εκθετική κατανομή (exponential distribution) χρησιμοποιείται η συνάρτηση **rexp** (random exponential), όπως π.χ. στην παρακάτω εντολή που ζητούνται 100 τυχαίοι αριθμοί σε εκθετική κατανομή με μέσο όρο 2000:

```
t=rexp(100, 1/2000)
```

Ενδεικτικό αποτέλεσμα της παραπάνω εντολής φαίνεται στην Εικόνα 2.4[127]. Για την παραγωγή τυχαίων τιμών σε διωνυμική κατανομή (binomial distribution) παρέχεται η συνάρτηση **rbinom** (random binomial). Οι αριθμοί που παράγονται από τη rbinom είναι ακέραιοι και μετρούν τις επιτυχίες σε ένα πείραμα με δύο πιθανά αποτελέσματα. Παρακάτω, καταγράφονται 10 τέτοιοι αριθμοί επιτυχιών (άρα δοκιμάζεται 10 φορές ένα πείραμα), κάθε φορά γίνεται μία μοναδική δοκιμή (size=1) και σε κάθε δοκιμή υπάρχει πιθανότητα επιτυχίας 50% (prob=0.5). Άρα αυτό είναι αντίστοιχο με το να καταγραφούν 100 πειράματα όπου σε καθένα γίνεται μία μοναδική ρίψη ενός νομίσματος, ενώ σε κάθε ρίψη καταγράφεται επιτυχία αν «ήρθε γράμματα» κάτι για το οποίο υπάρχει 50% πιθανότητα (αν το νόμισμα είναι «δίκαιο»):

```
> rbinom(100,size=1,prob=0.5)
  [1] 1 0 0 0 1 0 1 0 1 0 1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 1 0
 [28] 0 0 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0
 [55] 1 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 1 0 1
 [82] 1 0 1 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 0
```

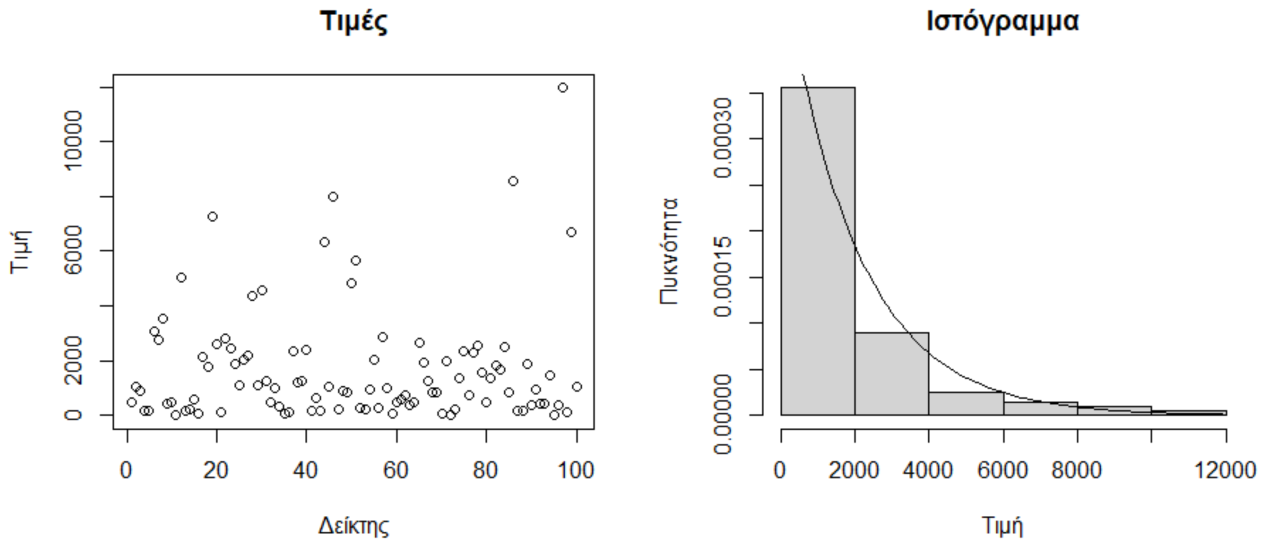
Για περισσότερες συναφείς συναρτήσεις του πακέτου ‘stats’ και παραγωγή τυχαίων αριθμών σε άλλες κατανομές (Poisson, Student's t, Weibull κ.α) βλ. help(Distributions).

Ιδιαίτερα χρήσιμη (και συνήθης) είναι η συνάρτηση **sample**. Εδώ γίνεται τυχαία επιλογή στοιχείων από ένα διάνυσμα (με ίδια πιθανότητα επιλογής κάθε στοιχείου). Η παράμετρος replace της συγκεκριμένης συνάρτησης ορίζει αν κατά την εκτέλεση της sample κάποιο στοιχείο που επιλέχθηκε μπορεί να επιλεχθεί ξανά (αν οριστεί replace=TRUE) ή όχι (αν οριστεί replace=FALSE κάτι που είναι και η προεπιλεγμένη τιμή).

Για παράδειγμα, στο παρακάτω επιλέγονται 15 στοιχεία από το διάνυσμα 1:20, ενώ επιλεγμένα στοιχεία μπορούν να επιλεγούν πάλι. Το αποτέλεσμα της sample είναι ένα διάνυσμα τιμών των επιλεγμένων στοιχείων:

```
> sample(1:20, size = 15, replace = T)
 [1] 20  9 15  4 19  4 15  7 10 11 13 11  6  8 18
```

[127] Η Εικόνα 2.4 δημιουργήθηκε με τις εντολές: layout(matrix(c(1,2), nrow=1)); plot(t, main = "Τιμές", xlab = "Δείκτης", ylab = "Τιμή"); hist(t, probability=TRUE, main = "Ιστόγραμμα", xlab = "Τιμή", ylab = "Πυκνότητα"); curve(dexp(x, 1/2000),add= T)



Εικόνα 2.4 *Τυχαίες τιμές που παρήχθησαν με τη συνάρτηση rexp.*

Παρατηρήστε πως οι τιμές 4, 11 και 15 έχουν επιλεγεί πάνω από μία φορά. Αντίθετα, η επόμενη εντολή επίσης επιλέγει 15 στοιχεία από το διάνυσμα 1:20, αλλά κάθε στοιχείο μπορεί να επιλεγεί μόνο μια φορά:

```
> sample(1:20, size = 15)
 [1] 11 14  4 10 20 17 15  1  6 12  9  2  7 18  3
```

Η sample δέχεται ως 1η παράμετρο x τα δεδομένα από τα οποία θα επιλέξει και αυτό μπορεί να είναι οποιοδήποτε διάνυσμα. Άρα στο τελευταίο παράδειγμα που ακολουθεί επιλέγονται τυχαία 5 στοιχεία από το δοθέν διάνυσμα στοιχείων character:

```
> sample(c("Αθήνα","Πάτρα","Βόλος"), size=5, replace = T)
[1] "Αθήνα" "Αθήνα" "Πάτρα" "Βόλος" "Αθήνα"
```

2.7 Οι συναρτήσεις-βοηθήματα: str, summary και dput

Καθώς εργάζεστε περισσότερο με την R ίσως βρείτε τρείς συναρτήσεις να είναι ιδιαίτερα χρήσιμες και αυτές είναι οι **str**, **dput** και **summary**, Και οι τρεις έχουν αναφερθεί σύντομα σε προηγούμενες ενότητες αλλά εδώ παρουσιάζονται εκτενέστερα.

Η str (συντομογραφία της λέξης structure, δομή), εμφανίζει συνοπτικά τα περιεχόμενα ενός αντικειμένου, δίνοντας πληροφορίες για τον τύπο των στοιχείων, ενδεικτικές τιμές που περιέχονται σε αυτά, ιδιότητες[128] κλπ. Είναι ιδιαίτερα χρήσιμη σε αντικείμενα που αποθηκεύουν δεδομένα, ειδικά αν πρόκειται για τύπους αντικειμένων που πιθανόν περιέχουν διαφορετικούς τύπους στοιχείων (όπως οι λίστες[129]). Στόχος της str είναι να περιγράφει κάθε βασικό στοιχείο του αντικειμένου με μια γραμμή κειμένου στην οθόνη. Αν και η τεκμηρίωση[130] την κατηγοριοποιεί ως συνάρτηση διαγνωστικής χρήσης, ο συνοπτικός αυτός τρόπος περιγραφής του αντικειμένου και της δομής του κάνει τη str ιδιαίτερα χρήσιμη για γενικότερη χρήση. Όπως συμβαίνει συχνά, κάποιες κλάσεις αντικειμένων (όπως τα data.frame[131]) παρέχουν τη δική τους μέθοδο str, προσαρμοσμένη στις ιδιαιτερότητες του συγκεκριμένου τύπου, άρα μπορούν να προσφέρουν χρήσιμη πληροφορία σε οποιονδήποτε τύπο αντικειμένου. Καθώς όμως δεν έχουμε παρουσιάσει ακόμα πολυπλοκότερους τύπους αντικειμένων, το παρακάτω παράδειγμα χρησιμοποιεί ένα vector x (με 100 τυχαίους αριθμούς[132]):

```
> x<-rnorm(100,mean=50,sd =20)
> str(x)
```

[128] βλ.§4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων.
[129] βλ.§4.2.1 Ο τύπος list (λίστα).
[130] βλ. help(str).
[131] βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων).
[132] βλ. παραπάνω §2.6 Τυχαίοι αριθμοί.

```
num [1:100] 55.2 74.2 68.8 48.5 36.2 ...
```

Ακόμα και σε ένα απλό vector (όπως το x σε αυτό το παράδειγμα), η str συνόψισε με χρήσιμο τρόπο τη δομή του αντικειμένου, δείχνοντας τον τύπο των δεδομένων (num), τον αριθμό των στοιχείων (100) και (ενδεικτικά) τα πρώτα του στοιχεία. Εδώ, λόγω της απλής δομής του αντικειμένου, χρειάστηκε μόνο μια γραμμή κειμένου.

Η **summary** αφορά τη σύνοψη (σε κείμενο) του αντικειμένου. Είναι generic συνάρτηση (όπως η print και πληθώρα άλλων[133]) οπότε οι διάφοροι τύποι (κλάσεις) αντικειμένων παρέχουν τη δική τους σχετική μέθοδο. Ως αποτέλεσμα, η μορφή της εξόδου της summary ποικίλει και εξαρτάται από τον τύπο του αντικειμένου στο οποίο εφαρμόζεται. Παρακάτω ζητείται μια σύνοψη του x:

```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  8.562  35.685  50.146  50.092  63.110  96.654
```

Σε αντικείμενα numeric (όπως το x) η summary επιστρέφει ένα αντικείμενο τύπου table όπου καταγράφονται[134] τα στοιχειώδη περιγραφικά στατιστικά στοιχεία των τιμών του (ελάχιστο, διάμεσος, αριθμητικός μέσος όρος κλπ.). Καθώς το αντικείμενο είναι table, μπορεί να γίνει και χρήση των στοιχείων του, π.χ. summary(x)[3] είναι η διάμεσος τιμή (median) στο x (το 3ο στοιχείο στο παραπάνω table). Αν όμως κληθεί σε ένα logical ή factor, η summary επιστρέφει τον αριθμό των περιπτώσεων ανά κατηγορία:

```
> summary(x>70)
   Mode   FALSE    TRUE
logical      86      14
```

Προκειμένου να δοθεί έμφαση στις διαφορετικές μορφές εξόδου που παρέχουν οι διάφοροι τύποι αντικειμένων μέσω της summary, το επόμενο παράδειγμα (που προτρέχει αρκετά) ζητά το summary ενός γραμμικού μοντέλου (τους συντελεστές δηλαδή μιας ευθείας γραμμής) που εφαρμόζει καλύτερα στα δεδομένα. Στο παράδειγμα, τα «δεδομένα» είναι το formula 2*x+3 ~ x άρα[135] εξετάζονται οι τιμές του x και οι αντίστοιχες τιμές του f(x)=2x+3:

```
> summary(lm(2*x+3 ~ x))

Call:
lm(formula = 2 * x + 3 ~ x)

Residuals:
       Min         1Q     Median         3Q        Max
-1.895e-13  4.770e-16  1.528e-15  2.843e-15  1.489e-14

Coefficients:
             Estimate Std. Error   t value Pr(>|t|)
(Intercept) 3.000e+00  5.217e-15 5.751e+14   <2e-16 ***
x           2.000e+00  9.354e-17 2.138e+16   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.954e-14 on 98 degrees of freedom
Multiple R-squared:      1,     Adjusted R-squared:      1
F-statistic: 4.571e+32 on 1 and 98 DF,  p-value: < 2.2e-16
```

Εδώ ο τύπος του επιστραφέντος αντικειμένου είναι summary.lm και περιέχει τόσο τους συντελεστές της γραμμής που εκτιμήθηκε (Coefficients) όσο και τα στοιχεία που αφορούν τη διαδικασία εφαρμογής της γραμμής στα δεδομένα (Residuals κλπ.). Στη συγκεκριμένη περίπτωση, η summary εγείρει και ένα μήνυμα προειδοποίησης (warning)[136] που δεν είναι μέρος του επιστραφέντος αντικειμένου. Στο μήνυμα αυτό ενημερώνει ότι τα δεδομένα είχαν τέλεια εφαρμογή (κάτι προφανές αφού το f(x)=2x+3 είναι μια απλή

[133] Για τις generic συναρτήσεις βλ.§6.2 Αντικειμενοστραφής προγραμματισμός στην R. Για την print βλ.§2.3 Συναρτήσεις κειμένου και ο βασικός τύπος character.
[134] Συγκεκριμένα ανήκει στις κλάσεις "summaryDefault" και "table".
[135] Για τον τύπο formula, βλ.§4.3.4 Αντικείμενα τύπου language (expression, call, name και formula).
[136] βλ. §5.1.4 Έγερση και χειρισμός σφαλμάτων.

συνάρτηση ευθείας γραμμής) και έτσι ίσως υπάρχουν ασαφείς τιμές στον υπολογισμό των στατιστικών στοιχείων του μοντέλου (αφού αυτά είναι πρακτικά ίσα με το 0).

Τέλος, η συνάρτηση **dput** (στην οποία μαζί με την dget θα αναφερθούμε πάλι στην ενότητα σχετικά με αρχεία, βλ.§9.1.2 Αποθήκευση και ανάκληση αντικειμένων) επιστρέφει μια έκφραση R η οποία αν εκτελεστεί δημιουργεί πάλι ένα υπάρχον αντικείμενο. Έτσι, π.χ. η εντολή dput(x) επιστρέφει την εντολή της συνάρτησης c που χρειάζεται για να δημιουργηθεί πάλι στο μέλλον το συγκεκριμένο numeric αντικείμενο με τις τρέχουσες τιμές του:

```
> dput(x)
c(55.1750542644995, 74.2080005582173, 68.780005582173 κλπ… )
```

Προφανώς η έξοδος της dput στο παράδειγμα αυτό περιέχει μια εντολή c με 100 αριθμητικές τιμές (τις 100 τρέχουσες τιμές των στοιχείων στο x), που όμως εδώ έχουν παραληφθεί για λόγους συντομίας.

2.8 Αποθήκευση του User Workspace

Πολλά από τα παραδείγματα που προηγήθηκαν δημιούργησαν κάποια αντικείμενα σε μεταβλητές. Όλα όμως τα αντικείμενα αυτά χάνονταν κάθε φορά που τελείωνε η συνεδρία της R (π.χ. τερματίζονταν το RStudio). Υπάρχουν διάφοροι τρόποι ώστε να μη χαθεί η εργασία που έχει γίνει με την R κατά την τρέχουσα συνεδρία. Ένας τρόπος διατήρησης των δεδομένων στην τρέχουσα μορφή τους είναι να αποθηκευτούν σε κάποιο αρχείο τα περιεχόμενα του Global Environment (που όπως έχει ήδη αναφερθεί ονομάζεται και «χώρος εργασίας του χρήστη» ή user workspace). Αν κλείνοντας το RStudio (ή το RGui), εμφανιστεί ένα παράθυρο διαλόγου με το ερώτημα “Save workspace image?” και επιλεχθεί το “Save”, τότε αποθηκεύεται το τρέχον περιβάλλον του χρήστη στο προεπιλεγμένο αρχείο “~\.RData”[137] και ανακαλείται όταν ξεκινήσει πάλι το RStudio επαναφέροντας έτσι τις όποιες μεταβλητές[138]. Το RStudio δίνει επίσης τη δυνατότητα να αποθηκευτεί/ανακληθεί το workspace σε/από συγκεκριμένο αρχείο με τα σχετικά κουμπιά της καρτέλας Environment (βλ.§1.4.2.7 Η καρτέλα Environment). Αν πρέπει να γίνει κάτι αντίστοιχο με κώδικα, οι συναρτήσεις save.image, save και load το επιτρέπουν. Για παράδειγμα η εντολή:

```
save.image("my_workspace_image")
```

αποθηκεύει το περιβάλλον σε αρχείο με όνομα "my_workspace_image" στον τρέχοντα φάκελο εργασίας, (βλ.§3.1.6 Φάκελοι αρχείων και ο τρέχων φάκελος εργασίας) και έτσι μπορεί να επαναφερθεί σε ύστερο χρόνο (ή άλλη συνεδρία) με την παρακάτω εντολή:

```
load("my_workspace_image")
```

Αν επιπρόσθετα χρειάζεται η επιλογή του αρχείου να γίνει διαδραστικά, μπορούν να αξιοποιηθούν συναρτήσεις εμφάνισης διαλόγων επιλογής αρχείου. Μεταξύ άλλων, τέτοιες συναρτήσεις είναι η file.choose ή (βλ.§3.3 Στοιχεία διεπαφής χρήστη (user interface)) ή οι σχετικές συναρτήσεις του πακέτου ΄tcltk’ για το οποίο υπάρχουν περισσότερα σε άλλη ενότητα (βλ.§7.2 Tcl/Tk):

```
library(tcltk)
fileName <- tclvalue(tkgetOpenFile())
load(fileName)
```

Η αποθήκευση του User Workspace είναι ένας καλός τρόπος να επαναφέρουμε την R στην κατάσταση που βρισκόταν κατά κάποια προηγουμένη συνεδρία, αλλά δεν λειτουργεί πάντα. Σε σπάνιες περιπτώσεις, κάποιοι ιδιαίτεροι τύποι αντικειμένων μπορεί να μη μπορούν να ανακληθούν από ένα τέτοιο αρχείο. Υπάρχουν και άλλοι τρόποι αποθήκευσης και ανάκλησης δεδομένων σε αρχεία (μεταξύ αυτών και με χρήση της συνάρτησης dput για τις μεταβλητές που μας ενδιαφέρουν[139]).

Ένας άλλος τρόπος διατήρησης της εργασίας που έχει γίνει, είναι να αποθηκευτεί ο κώδικάς σε ένα αρχείο σεναρίου ώστε να είναι δυνατή η επανεκτέλεσή του και η επαναδημιουργία των αποτελεσμάτων. Τα αρχεία σεναρίων προσφέρουν πολλές επιλογές, πρόσθετα πλεονεκτήματα και δυνατότητες και θα αναφερθούμε σε αυτά παρακάτω (βλ.§3.1 Σενάρια (R-script)).

[137] Στα Windows ο φάκελος ~ είναι τα “Έγγραφα” του χρήστη, άρα εκεί αποθηκεύεται το αρχείο .RData.

[138] Στο RStudio σχετικές ρυθμίσεις υπάρχουν στο μενού Tools/Global Options, επιλογή General, Καρτέλα Basic κάτω από τον τίτλο “Workspace”.

[139] βλ. παραπάνω §2.7 Οι συναρτήσεις-βοηθήματα: str, summary και dput και §9.1 Εισαγωγή και εξαγωγή δεδομένων για τη χρήση αρχείων στην αποθήκευση αντικειμένων.

Αναφορές Κεφαλαίου 2

[18] Peng, R. D. (2015). *R Programming for Data Science*. Lean Publishing. https://bookdown.org/rdpeng/rprogdatascience/

[33] Wickham, H. (2019). *Advanced R (2nd Editton)*. Chapman and Hall/CRC. https://adv-r.hadley.nz/

[34] Wickham, H. (2010, February). Stringr: modern, consistent string processing. *The R Journal, τόμ. 2, αρ. 2*. https://CRAN.R-project.org/package=stringr

[35] Wickham, H., Hester, J., & Ooms, J. (2021). Xml2: Parse XML. https://CRAN.R-project.org/package=xml2

[36] Benoit, K., Watanabe, K., & Wang, H. (2018). Quanteda: An R package for the quantitative analysis of textual data. *Journal of Open Source Software, τόμ. 3, αρ. 30*, p. 774. https://quanteda.io

[37] Feinerer, I., Hornik, K., & Meyer, D. (2008). Text Mining Infrastructure in R. *Journal of Statistical Software, τόμ. 5, αρ. 25*, pp. 1-54. https://www.jstatsoft.org/v25/i05/

[38] Jockers, M. L. (2015). Syuzhet: Extract Sentiment and Plot Arcs from Text. https://github.com/mjockers/syuzhet

[39] Proellochs, N., & Feuerriegel, S. (2021). SentimentAnalysis: Dictionary-Based Sentiment Analysis. https://CRAN.R-project.org/package=SentimentAnalysis

[40] Friedl, J. E. (2006). *Mastering Regular Expressions, Third Edition*. O'Reilly Media. ISBN 978-0596528126.

[41] Forta, B. (2018). *Learning Regular Expressions*. Addison-Wesley Professional. ISBN 9780134757063.

[42] Van Buuren, S. & Groothuis-Oudshoorn, K. (2011). Mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software, τόμ. 45, αρ. 3*, pp. 1-67. 10.18637/jss.v045.i03.

[43] Kowarik, A. & Templ, M. (2016). Imputation with the R Package VIM. *Journal of Statistical Software, τόμ. 74, αρ. 7*, pp. 1-16. 10.18637/jss.v074.i07.

[130] The R Core Team (2021). R Internals. https://cran.r-project.org/doc/manuals/r-release/R-ints.pdf

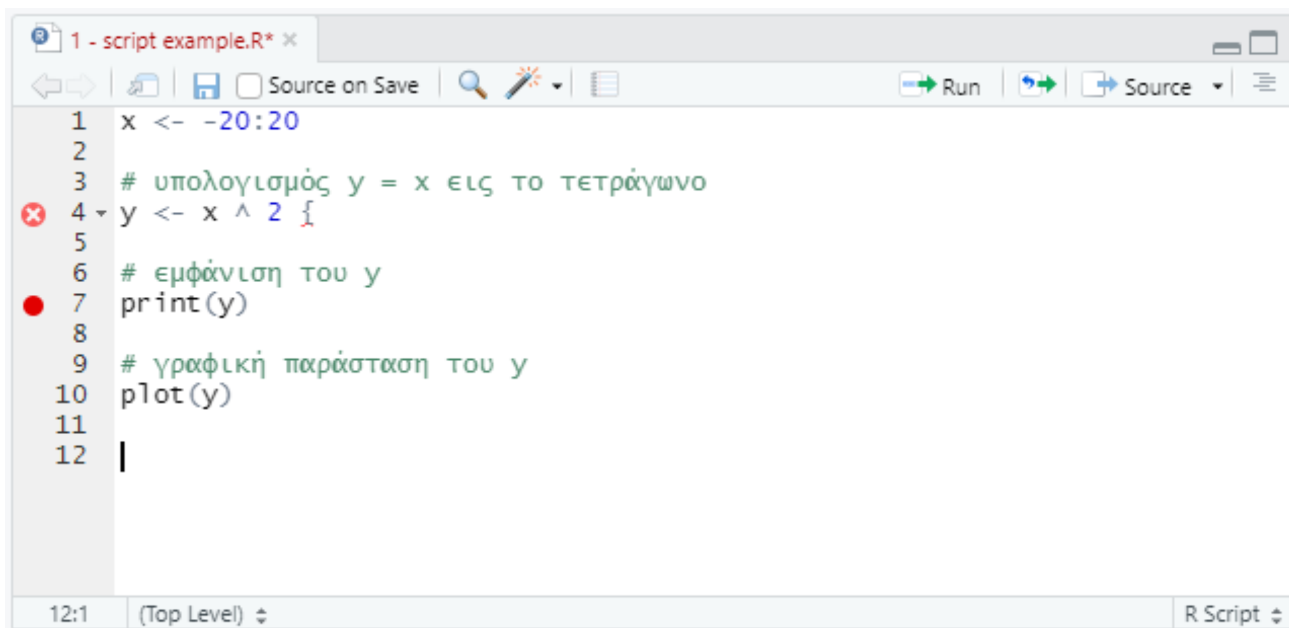
Κεφάλαιο 3: Τα βασικά εργαλεία του προγραμματιστή R

Σύνοψη
Το κεφάλαιο αυτό συγκεντρώνει διάφορα στοιχεία που σχετίζονται με τη δημιουργία σεναρίων (δηλαδή «προγραμμάτων» σε γλώσσα R). Τα θέματα που παρουσιάζονται περιλαμβάνουν τη δημιουργία και εκτέλεση σεναρίων (R-Script) και τα σχετικά βοηθητικά εργαλεία (όπως διόρθωσης λαθών / debugging), τεχνικές καταγραφής του χρόνου εκτέλεσης κλπ.

Προαπαιτούμενη γνώση
Εξοικείωση με το περιβάλλον της R και του RStudio (βλ. Κεφάλαιο 1), βασικές δεξιότητες χρήσης R, μεταβλητές και βασικοί τύποι (βλ. Κεφάλαιο 2).

3.1 Σενάρια (R-script)

Αν μέχρι τώρα δεν νιώσατε αρκετά πως κάνατε «προγραμματισμό», αυτό είναι δικαιολογημένο. Η χρήση της R όπως παρουσιάστηκε μέχρι στιγμής αφορούσε κυρίως εκτέλεση έτοιμων συναρτήσεων, απευθείας στο Console. Πολλά χρήσιμα αποτελέσματα μπορούν να επιτευχθούν με τον τρόπο αυτό, αλλά ταυτόχρονα είναι και ιδιαίτερα περιοριστικός. Γράφοντας εντολές απευθείας στο Console, ο κώδικάς σας ουσιαστικά χάνεται μετά την εκτέλεση του, μόνο κάποια αποτελέσματά του ίσως μένουν, εφόσον αποθηκεύτηκαν σε μεταβλητές. Και αυτό έως ότου να κλείσει η συνεδρία σας με την R. Ας προχωρήσουμε λοιπόν στο επόμενο επίπεδο.



Εικόνα 3.1: *Ο επεξεργαστής πηγαίου κώδικα (source editor) του RStudio.*

Στην R ο βασικός τρόπος να αποθηκευτεί και να επαναχρησιμοποιηθεί κώδικας είναι σε αρχεία σεναρίων, τα επονομαζόμενα R script. Τα αρχεία αυτά περιέχουν τις εντολές R και είναι ό,τι κοντινότερο σε «πρόγραμμα» έχει μια γλώσσα πού δρα ως “διερμηνέας” (interpreter) του κώδικα, όπως η R. Πέραν του προφανούς πλεονεκτήματος ότι διατηρούν τον κώδικά ώστε να μπορεί να χρησιμοποιηθεί ξανά, τα σενάρια μπορούν να δημιουργήσουν ξανά τα αποτελέσματα, γεγονός σημαντικό όταν πρέπει αυτά και η διαδικασία που τα επέφερε να είναι ελέγξιμη και αναπαράξιμη (reproducible, repeatable research). Επιπρόσθετα με τον κώδικα να βρίσκεται σε σενάρια, η όποια διαδικασία εφαρμόστηκε μπορεί να προσαρμοστεί σε κάποιο νέο παρεμφερές πρόβλημα, να διαμοιραστεί σε τρίτους, να ενσωματωθεί σε πακέτα (βλ.§8.2 Δομή φακέλου για δημιουργία πακέτου) ή να εκτελεστεί σε κάποιον διακομιστή (server).

Τα αρχεία σεναρίων αποθηκεύονται με όνομα που έχει την επέκταση .R στο όνομά τους (π.χ. test.R), και είναι αρχεία κειμένου. Έτσι, είναι επεξεργάσιμα από οποιαδήποτε εφαρμογή του υπολογιστή σας είναι

κατάλληλη για τον σκοπό αυτό, όπως π.χ. το Σημειωματάριο. Όμως, η συγγραφή σεναρίων R διευκολύνεται αν χρησιμοποιηθεί ένας επεξεργαστής που όχι μόνο επιτρέπει αλλαγές και διορθώσεις, αλλά παρέχει και άλλα σχετικά βοηθήματα στον προγραμματιστή. Ένα τέτοιο εργαλείο είναι ο επεξεργαστής πηγαίου κώδικα (source editor) που είναι ενσωματωμένος στο RStudio (Εικόνα 3.1) και παρέχει βοηθήματα στα οποία θα αναφερθούμε παρακάτω, όπως:

- Επισήμανση κώδικα (syntax highlighting) που εμφανίζει με διαφορετική μορφή (π.χ. με διαφορετικό χρώμα) τμήματα του κώδικα ανάλογα με τον σκοπό τους.
- Επισήμανση συντακτικών λαθών. Οι γραμμές με προφανή συντακτικά λάθη σημειώνονται με ένα εικονίδιο (Χ σε κόκκινο κύκλο, για παράδειγμα βλ. γραμμή 4 στην Εικόνα 3.1), ενώ με τον δείκτη του ποντικιού πάνω στο εικονίδιο αυτό εμφανίζεται περιγραφή του λάθους.
- Αυτόματη συμπλήρωση κώδικα (code completion) που ενεργοποιείται πατώντας το πλήκτρο Tab κατά την καταχώρηση εντολών.
- Δημιουργία μεταβλητών και συναρτήσεων με μετατροπή κάποιου επιλεγμένου τμήματος κώδικα (μενού Code/Extract Variable και Code/Extract Function).
- Άμεση πρόσβαση στην τεκμηρίωση (help) της τρέχουσας εντολής (πατώντας το πλήκτρο F1).
- Μεταφορά εντολών από το ιστορικό (βλ.§1.4.2.6 Η καρτέλα History).
- Αυτόματη δημιουργία εσοχών (indentation) και τοποθέτηση κενών ώστε να γίνει ο κώδικας πιο ευανάγνωστος (βλ. και §3.1.8 Μερικές παρατηρήσεις σχετικές με τη συγγραφή σεναρίων).
- Μετατροπή κειμένου σε σχόλια και αντίστροφα (βλ.§3.1.2 Σχολιασμός του κώδικα).
- Εργαλεία εντοπισμού σφαλμάτων (βλ.§3.1.3 Βοηθήματα εντοπισμού λαθών (debugging))
- Εργαλεία εκτίμησης της απόδοσης του κώδικα (βλ.§3.1.4 Εργαλεία προφίλ (profiling)).
- Δημιουργία απλών αναφορών των αποτελεσμάτων του κώδικα (βλ.§3.1.5 Εργαλείο αναφοράς (report)).

3.1.1 Δημιουργία και εκτέλεση R script

Η δημιουργία ενός νέου κενού σεναρίου (R script) στο RStudio γίνεται με την επιλογή μενού File/New File/R-Script[140]. Αρχικά το αρχείο θα είναι «ανώνυμο» - θα έχει το προσωρινό όνομα Untitled1.R (ή Untitled2.R κλπ. αν έχουν δημιουργηθεί ταυτόχρονα περισσότερα του ενός τέτοια αρχεία). Δοκιμάστε να δημιουργήσετε ένα νέο τέτοιο αρχείο R script και να καταγράψτε σε αυτό τις εντολές κάποιου από τα παραδείγματα που προηγήθηκαν σε προηγούμενες ενότητες. Μετά, προτείνουμε να το αποθηκεύσετε σε ένα συγκεκριμένο αρχείο σε θέση και με όνομα που θα επιλέξετε εσείς (μενού File/Save ή πατώντας το εικονίδιο δισκέτας στην καρτέλα του, ή απλώς πατώντας Ctrl+S στο πληκτρολόγιο). Προφανώς, ένα αποθηκευμένο σενάριο μπορεί μελλοντικά να φορτωθεί και πάλι στον source editor του RStudio για να γίνουν αλλαγές σε αυτό με την επιλογή μενού File/Open File (ή Ctrl+O). Αφού κλείσετε στο σενάριο από την επεξεργασία (μενού File/Close ή πατώντας το 'x' στην καρτέλα του), δοκιμάστε να εκτελέστε το σενάριο με τη συνάρτηση source που θα δημιουργηθεί επιλέγοντας το μενού Code/Source File (ή πατώντας Ctrl+Alt+G) και επιλέγοντας το αρχείο. Η συνάρτηση source φορτώνει κάποιο δοθέν αρχείο σεναρίου και εκτελεί τον κώδικα που περιέχεται σε αυτό. Η συνάρτηση μπορεί να κληθεί απευθείας στο Console αλλά και μέσα σε κάποιο σενάριο για να εκτελέσει από αυτό κάποιο άλλο σενάριο. Αν στον προσδιορισμό του αρχείου που περιέχει το σενάριο δεν οριστεί πλήρης διαδρομή, το σύστημα θα θεωρήσει πως βρίσκεται στον τρέχοντα φάκελο εργασίας (βλ.§3.1.6 Φάκελοι αρχείων και ο τρέχων φάκελος εργασίας).

Όταν ένα σενάριο εκτελείται (με source), οι εντολές που περιέχει περνούν μια - μια στον διερμηνευτή της R όπως θα γινόταν αν είχαν γραφτεί απευθείας στο Console. Με αυτό τον τρόπο μεταφράζονται και εκτελούνται. Τα αποτελέσματα των εντολών επηρεάζουν (και επηρεάζονται από) το τρέχον περιβάλλον, όπως ακριβώς αν οι εντολές είχαν δοθεί απευθείας. Η βασική διαφορά είναι πως το σενάριο εκτελείται ως μια ενιαία οντότητα και για τον λόγο αυτό η εκτέλεση όλου του σεναρίου θα σταματήσει αν διαπιστωθεί κάποιο σφάλμα στον κώδικα (βλ.§5.1.4 Έγερση και χειρισμός σφαλμάτων). Επίσης, επειδή ο κώδικας εκτελείται ως σύνολο, δεν εμφανίζονται αυτόματα (auto-print) οι επιστρεφόμενες τιμές των εκάστοτε εντολών. Εμφανίζονται μόνο τα αποξέσματα συναρτήσεων που παράγουν οπτική έξοδο, όπως π.χ. οι συναρτήσεις εμφάνισης κειμένου που αναφέρθηκαν ήδη σε προηγουμένη ενότητα (βλ.§2.3 Συναρτήσεις κειμένου και ο βασικός τύπος character).

[140] Στο RGui η αντίστοιχη εντολή για άνοιγμα ενός (απλούστερου) επεξεργαστή κώδικα είναι File/New Script.

Ένα σενάριο που βρίσκεται στον source editor του RStudio μπορεί επίσης να εκτελεστεί, χωρίς να ακολουθηθεί η παραπάνω διαδικασία (αποθήκευση, κλείσιμο κλπ.), απευθείας με το κουμπί Source (ή Ctrl+Shift+S). Μπορεί επίσης να εκτελεστεί με προβολή του επιστρεφόμενου αποτελέσματος κάθε εντολής (κουμπί Source with Echo ή Ctrl+Shift+Enter). Τέλος, αν έχει επιλεγεί κάποιο τμήμα του σεναρίου, η επιλογή Run (κουμπί Run ή Ctrl+Enter) εκτελεί μόνο το τμήμα αυτό με ταυτόχρονη εμφάνιση των επιστρεφόμενων αποτελεσμάτων, ενώ αν δεν υπάρχει επιλεγμένο τμήμα κώδικα, το Run εκτελεί την εντολή στην τρέχουσα γραμμή και προχωρά τον δρομέα (cursor) στην επόμενη, επιτρέποντας έτσι την εκτέλεση του κώδικα γραμμή-γραμμή.

3.1.2 Σχολιασμός του κώδικα

Τα σενάρια είναι ένας τρόπος να διατηρηθεί ο κώδικας για μελλοντική χρήση ή να χρησιμοποιηθεί από άλλους. Σε ένα καλογραμμένο σενάριο (ή όπου αλλού αποθηκεύεται κώδικας), είναι σημαντικό να υπάρχουν ενσωματωμένα σχόλια. Η ύπαρξη σχολίων στον κώδικα θα βοηθήσει άλλους χρήστες (ή ακόμα και τους δημιουργούς του, αν τον εξετάζουν πάλι στο μέλλον) να τον κατανοήσουν καλύτερα. Στην R, τα σχόλια ορίζονται με τον χαρακτήρα '#' και οποιοδήποτε κείμενο ακολουθεί τον χαρακτήρα αυτόν θεωρείται σχόλιο (βλ. Εικόνα 3.1, όπου τα σχόλια έχουν πράσινο χρώμα). Κατά την επεξεργασία σεναρίων στον source editor του RStudio, η επιλογή μενού Code/Comment-Uncomment Lines (ή Ctrl+Shift+C) μετατρέπει την τρέχουσα γραμμή (ή επιλεγμένο κείμενο) σε σχόλιο και αντίστροφα. Η μετατροπή σε σχόλιο είναι και ένας άτυπος τρόπος που χρησιμοποιούν οι προγραμματιστές για να απενεργοποιήσουν προσωρινά τμήματα κώδικα χωρίς να τα διαγράψουν οριστικά.

Αν και η ίδια η R δεν επεξεργάζεται κείμενο μέσα σε σχόλια, κάποια εργαλεία που σχετίζονται με την R μπορεί να το κάνουν. Έτσι, κάποια εργαλεία υποστηρίζουν ειδικές μορφές σχολίων μέσω των οποίων ο δημιουργός του σεναρίου R μπορεί να επηρεάσει, να περάσει «εντολές» ή να παραμετροποιήσει τον τρόπο λειτουργίας του εργαλείου. Ένα παράδειγμα είναι ο ίδιος ο source editor του RStudio. Ένα σχόλιο που ξεκινά με #---- ο source editor θεωρεί πως καθορίζει την αρχή ενός code chunk (κομμάτι κώδικα), μιας - κατά κάποιο τρόπο - νέας παραγράφου στον κώδικα. Προφανώς η R παραβλέπει και δεν επεξεργάζεται ούτε αυτό το σχόλιο, αλλά ο source editor το χρησιμοποιεί (μαζί με άλλα παρόμοια) για να δημιουργήσει ένα περίγραμμα του κώδικα (code outline) με το οποίο μπορεί ο προγραμματιστής να περιηγηθεί γρήγορα μέσα σε κάποιο - πιθανώς μεγάλο - αρχείο σεναρίου ή προσωρινά να αποκρύψει κομμάτια κώδικα για να διευκολύνει την εργασία του με άλλα. Το περίγραμμα εμφανίζεται με την επιλογή μενού Code/Show Document Outline (η πατώντας τα πλήκτρα Ctrl+Shift+O). Σχόλια που περιέχουν κείμενο και τελειώνουν με #### θεωρείται πως περιέχουν την ονομασία του κομματιού κώδικα που θέλουμε να εμφανίζεται στο περίγραμμα του κώδικα, π.χ. τα σχόλια:

```
# ----
# load data ####
# analyze data ####
```

τοποθετημένα σε σημεία ενός σεναρίου, οριοθετούν τρία κομμάτια κώδικα, το πρώτο χωρίς όνομα (untitled), ενώ το 2^{ο} και το 3^{ο} με όνομα 'load data' και ΄analyze data', αντίστοιχα. Τα ονόματα αυτά θα εμφανίζονται στο περίγραμμα του κώδικα (code outline) διευκολύνοντας τη μετακίνηση στα συγκεκριμένα σημεία κλπ. Για ένα ακόμα παράδειγμα χρήσης σχολίων R για την προσαρμογή κάποιου εργαλείου που επεξεργάζεται τον κώδικα βλ.§3.1.5 Εργαλείο αναφοράς (report).

3.1.3 Βοηθήματα εντοπισμού λαθών (debugging)

Τα εργαλεία εντοπισμού λαθών (debugging) υποβοηθούν τη διαδικασία διόρθωσης προβλημάτων στον κώδικα δίνοντας εικόνα του τρόπου με τον οποίο εκτελείται εσωτερικά το σενάριο. Η αξία τους αναδεικνύεται όταν εμφανίζεται πρόβλημα κατά την εκτέλεση κάποιου (συχνά πολύπλοκου) κώδικα και δεν μπορεί να εντοπιστεί διαφορετικά. Επειδή τα σενάρια εκτελούνται αυτόνομα, είναι σαν ένα «μαύρο κουτί» κάνοντας δύσκολο να γνωρίζουμε τι συμβαίνει εσωτερικά. Με τα εργαλεία debugging μπορεί να γίνει παύση της εκτέλεσης του σεναρίου σε επιλεγμένα σημεία του (τα οποία ονομάζονται breakpoints) και έλεγχος της τρέχουσας κατάστασης μέσα στο σενάριο και στο περιβάλλον όπου εκτελείται. Στην Εικόνα 3.1, η κόκκινη συμπαγής κουκίδα αριστερά της γραμμής 7 δείχνει πως στη γραμμή αυτή έχει τοποθετηθεί ένα breakpoint. Η τοποθέτηση ενός breakpoint μπορεί να γίνει με το ποντίκι (με κλικ στη θέση αριστερά, όπως στο παράδειγμα) ή με την επιλογή Toggle Breakpoint του μενού Debug. Με τον ίδιο τρόπο καταργείται ένα breakpoint. Τα breakpoints δεν

αποθηκεύονται με το σενάριο (το οποίο είναι ένα απλό αρχείο κειμένου), άρα ισχύουν μόνο τοπικά στον υπολογιστή μας, αλλά η διαδικασία debugging του RStudio απαιτεί το σενάριο να έχει πρώτα αποθηκευτεί σε αρχείο (να μην είναι προσωρινό).

Εφόσον έχουν οριστεί ένα ή περισσότερα breakpoints, όταν πατηθεί το κουμπί Source (ή Ctrl+Shift+S) η εκτέλεση του σεναρίου γίνεται σε κατάσταση debug και όταν έρθει η στιγμή να εκτελεστεί η γραμμή που είναι σημειωμένη ως breakpoint, θα παύσει προσωρινά την εκτέλεση. Οι βασικές επιλογές που δίνονται τότε στον χρήστη, είτε από το μενού Debug είτε από τα ειδικά κουμπιά που εμφανίζονται πάνω από το Console είναι:

(α) Να επιλέξει τη συνέχιση της εκτέλεσης του σεναρίου κανονικά ως το επόμενο breakpoint ή το τέλος του σεναρίου (Continue).

(β) Να εκτελέσει τον κώδικα της τρέχουσας γραμμής και να σταματήσει πάλι στην αμέσως επόμενη (Execute Next Line).

(γ) Να μπει εντός τυχούσας συνάρτησης, να πρέπει να εκτελεστεί από την τρέχουσα εντολή και να σταματήσει εκεί ώστε να εξεταστεί ο κώδικας της συνάρτησης (Step Into Function). Αν πρόκειται για συναρτήσεις κάποιου πακέτου γραμμένες σε R ή αντίστοιχα, για συναρτήσεις ορισμένες από τον χρήστη (βλ.§5.1.1 Δημιουργία συναρτήσεων) κατά την εκτέλεση, σε κατάσταση debug, τότε ο χρήστης μπορεί να επιλέξει αν θα προχωρήσει στον κώδικα εντός της συνάρτησης (Step Into Function) ή αν θα την εκτελέσει ως ένα ενιαίο βήμα (Execute Next Line).[141]

(δ) Να ολοκληρώσει τυχούσα συνάρτηση ή βρόχο (loop) μέσα στον οποίο σταμάτησε (Finish Function/Loop) και να παύσει πάλι προσωρινά την εκτέλεση αμέσως μετά.

(ε) Να τερματίσει εντελώς την εκτέλεση του σεναρίου (Stop Debugging).

Το εργαλείο debugging δίνει πολλές δυνατότητες εξέτασης του κώδικα καθώς αυτός εκτελείται. Κάθε φορά που η διαδικασία debug κάνει παύση της εκτέλεσης του σεναρίου, ο προγραμματιστής μπορεί να δίνει κανονικά εντολές στην Console, εξετάζοντας ή επηρεάζοντας το περιβάλλον εκτέλεσης όπως είναι εκείνη τη στιγμή. Ταυτόχρονα στην καρτέλα Environment είναι διαθέσιμες οι τρέχουσες σχετικές πληροφορίες, όπως π.χ. οι μεταβλητές με τα χαρακτηριστικά τους στα διάφορα περιβάλλοντα που είναι στην τρέχουσα σειρά αναζήτησης αντικειμένων (βλ.§1.4.2.7 Η καρτέλα Environment). Αυτό ισχύει ακόμα και αν πρόκειται για εσωτερικές (τοπικές) μεταβλητές[142] κάποιας συνάρτησης που η εκτέλεση της έχει παύσει προσωρινά. Επιπρόσθετα, στο παράθυρο Traceback απεικονίζεται η ιεραρχία κλήσεων συναρτήσεων που έχουν οδηγήσει στην τρέχουσα εντολή. Συχνά κατά την κλήση μίας συνάρτησης ο κώδικάς της καλεί κάποια άλλη, η οποία ίσως με τη σειρά της καλεί κάποια τρίτη και ούτω καθεξής. Αν η παύση γίνει σε ένα τέτοιο επίπεδο, σε κάποια εντολή του εσωτερικού κώδικα κάποιας συνάρτησης, το παράθυρο Traceback περιγράφει την όποια αλληλουχία κλήσεων οδήγησε στην τρέχουσα εντολή, ξεκινώντας από το τρέχον επίπεδο και κινούμενο αντίστροφα προς τις κλήσεις που προηγήθηκαν.

Τέλος, πρέπει να αναφερθεί πως στο επίπεδο της ίδιας της γλώσσας R, η ενεργοποίηση και απενεργοποίηση του μηχανισμού debugging γίνεται με συναρτήσεις όπως οι debug, debugonce, debugcall και undebuggcall του πακέτου ‘utils’. Π.χ. η εντολή debugcall(x()) έχει ως αποτέλεσμα να εκτελούνται σε κατάσταση debug οι κλήσεις κάποιας συνάρτησης με όνομα x (ώστε να εξεταστεί η λειτουργία της συνάρτησης x). Αντίστοιχα, η εντολή undebugcall(x()) θα απενεργοποιήσει την κατάσταση debug για τη συνάρτηση x, και ενδεχόμενες κλήσεις της θα εκτελούνται κανονικά. Επιπρόσθετα, υπάρχουν συναρτήσεις χειρισμού σφαλμάτων καθώς και εξέτασης της κατάστασης που οδήγησαν σε αυτό, για περισσότερα βλ.§5.1.4 Έγερση και χειρισμός σφαλμάτων.

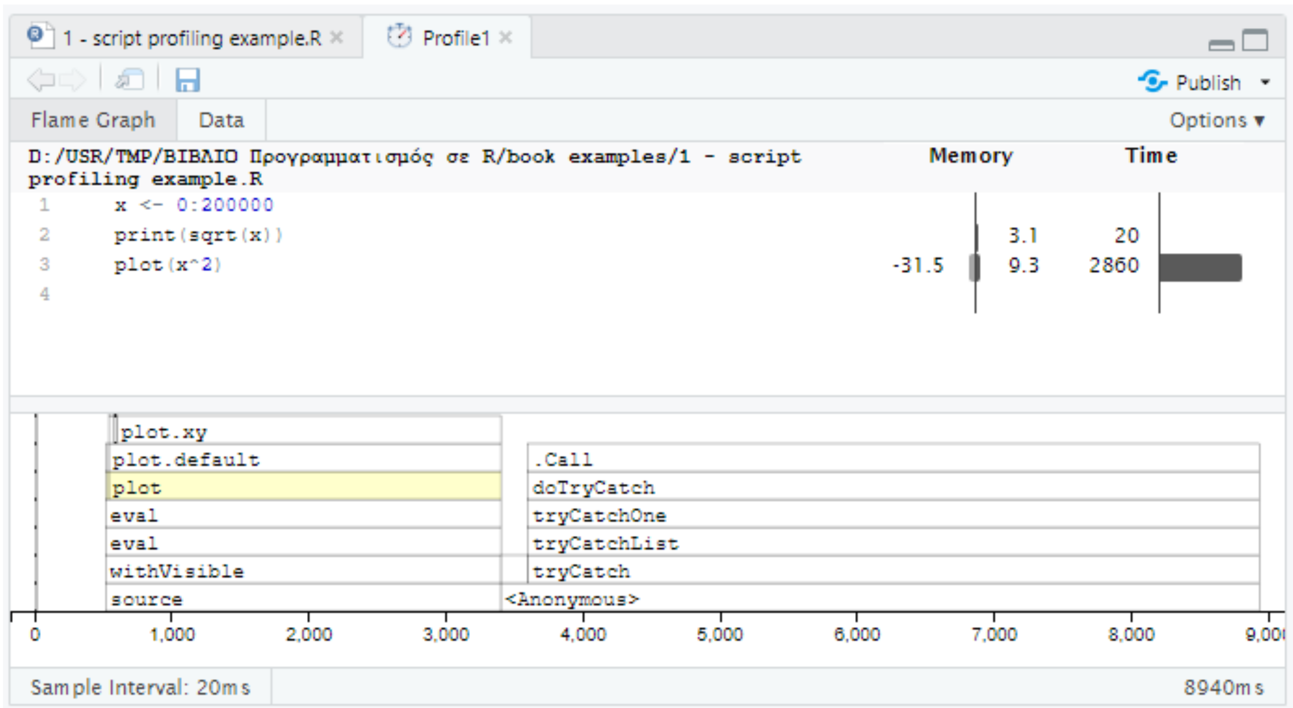
3.1.4 Εργαλεία προφίλ (profiling)

Το εργαλείο δημιουργίας προφίλ εκτέλεσης (profiling) συνοψίζει τον χρόνο και τη μνήμη που απαιτήθηκαν κατά την εκτέλεση ενός σεναρίου. Βοηθά να εντοπιστούν ποια τμήματα του κώδικα ήταν τα περισσότερο χρονοβόρα ή απαίτησαν την περισσότερη μνήμη σε πραγματικές συνθήκες. Όταν ενεργοποιηθεί το profiling οι

[141] Η διαδικασία Step Into Function δεν μπορεί να εισέλθει σε συναρτήσεις που παρέχονται ως ήδη μεταγλωττισμένες (compiled) σε γλώσσα μηχανής και εντολές του υπολογιστή. Εξάλλου, τέτοιες συναρτήσεις είναι γραμμένες σε άλλη γλώσσα προγραμματισμού.

[142] Για τις τοπικές μεταβλητές συναρτήσεων βλ.§5.1.5 Αλληλεπίδραση με το περιβάλλον.

εντολές που εκτελούνται παρακολουθούνται και καταγράφονται ο χρόνος και ο χώρος μνήμης που κατανάλωσαν.



Εικόνα 3.2 : *Παράδειγμα profile ενός απλού σεναρίου.*

Ένα απλό παράδειγμα: ο παρακάτω κώδικας δημιουργεί ένα διάνυσμα x που περιέχει τους ακέραιους αριθμούς από το 0 έως το 200000. Μετά υπολογίζει την τετραγωνική τους ρίζα (σε άλλο διάνυσμα) και το εμφανίζει (με τη συνάρτηση print). Τέλος υπολογίζει (σε άλλο διάνυσμα) το τετράγωνο των αριθμών στο x και δημιουργεί τη γραφική τους παράσταση (με τη συνάρτηση plot):

```
x <- 0:200000
print(sqrt(x))
plot(x^2)
```

Αν τοποθετηθεί ο παραπάνω κώδικας σε ένα σενάριο, ενεργοποιηθεί το profiling (επιλογή μενού Profile/Start Profiling), εκτελεστεί ο κώδικας (π.χ. με το κουμπί Source) και μετά την ολοκλήρωση της εκτέλεσης απενεργοποιηθεί το profiling (επιλογή μενού Profile/Stop Profiling), τότε θα εμφανιστεί το profile της εκτέλεσης του σεναρίου που φαίνεται στην Εικόνα 3.2. Τα αποτελέσματα που εμφανίζονται κάτω από το γράφημα με τίτλο Time δείχνουν πως οι δύο πρώτες γραμμές κώδικα του σεναρίου εκτελέστηκαν σε πολύ μικρότερο χρόνο (κατά προσέγγιση 20ms οι τιμές είναι 0 και 20ms αντίστοιχα), σε σχέση με αυτόν που καταναλώθηκε για τη δημιουργία της γραφικής παράστασης από τη συνάρτηση plot στη γραμμή 3 (περίπου 2860ms). Αντίστοιχα, στο γράφημα Memory εμφανίζεται η χωρητικότητα μνήμης που καταλήφθηκε και ελευθερώθηκε (αρνητικοί αριθμοί) κατά την εκτέλεση του κώδικα. Το εργαλείο εμφάνισης αποτελεσμάτων profiling στο RStudio είναι το πακέτο ΄profvis' [44] που λαμβάνει τις σχετικές πληροφορίες χρησιμοποιώντας τη συνάρτηση RProf του ενσωματωμένου πακέτου 'utils'. Το εργαλείο είναι διαδραστικό και αξίζει να εξερευνήσετε τις δυνατότητές του ειδικά αν σας ενδιαφέρει η βελτίωση κώδικα που απαιτεί μεγάλο χρόνο (ή χώρο μνήμης) κατά την εκτέλεσή του.

Στη περίπτωση αντικειμένων που περιέχουν μεγάλο αριθμό στοιχείων (και άρα καταλαμβάνουν μεγάλο χώρο στη μνήμη) κάτι που μπορεί να αυξήσει σημαντικά τον χρόνο εκτέλεσης του κώδικα, είναι ο κανόνας copy-on-modify που εφαρμόζει η R. Όπως έχει ήδη αναφερθεί, όταν γίνεται κάποια μεταβολή σε ένα αντικείμενο στη μνήμη, η R συνήθως αντιγράφει το αντικείμενο σε νέο και εφαρμόζει τις αλλαγές. Όμως υπάρχουν εξαιρέσεις στον κανόνα αυτό. Κάποιοι τύποι αντικειμένων επιτρέπουν αλλαγές στο ίδιο το υπάρχον αντικείμενο στη μνήμη, χωρίς δημιουργία νέου[143]. Επιπρόσθετα, ασχέτως τύπου, σε κάποιες περιπτώσεις η R

[143] Τέτοιες εξαιρέσεις περιλαμβάνουν τον τύπο environment (βλ.§4.2.2.2 Περιβάλλοντα και ο μηχανισμός αναφοράς) και τον τύπο data.table (βλ.§4.3.1 Οι τύποι tibble και data.table), καθώς και αντικείμενα που βασίζονται σε κλάσεις RS ή R6

εφαρμόζει «βαθιά αντιγραφή» (deep copy, άρα αντιγράφονται όλα τα δεδομένα), ενώ σε άλλες[144] εφαρμόζει «ρηχή» (shallow copy, άρα αντιγράφονται οι αναφορές προς τα στοιχεία του αρχικού αντικειμένου και συνεπώς κάποια στοιχεία παραμένουν κοινά στα δύο αντικείμενα και διαφοροποιούνται μόνο όταν γίνει κάποια τροποποίηση σε αυτά). Επιπροσθέτως, υπάρχουν και περιπτώσεις που η R αποφασίζει πως δεν χρειάζεται να δημιουργήσει νέο αντικείμενο και εφαρμόζει την αλλαγή στο υπάρχον. Καθένα από τα παραπάνω έχει διαφορετικό αντίκτυπο στον χρόνο εκτέλεσης της εντολής, ειδικά αν τα αντικείμενα περιέχουν μεγάλο αριθμό στοιχείων. Η εντολή **tracemem** ενεργοποιεί την παρακολούθηση κάθε αλλαγής του αντικειμένου στο οποίο αναφέρεται μια μεταβλητή[145], ενώ η **untracemem** την απενεργοποιεί. Η tracemem επιστρέφει τη θέση στη μνήμη του αντικειμένου με το οποίο είναι συσχετισμένο η μεταβλητή. Αν αυτή η θέση αλλάξει, σημαίνει πως κατά την εκτέλεση μιας εντολής χρειάστηκε να δημιουργηθεί ένα εξ ολοκλήρου νέο αντικείμενο και να ανατεθεί σε αυτή. Έτσι, αν έχουμε ήδη ορίσει το διάνυσμα x όπως παραπάνω, η επόμενη εντολή ενεργοποιεί την παρακολούθηση και επιστρέφει τη θέσης στη μνήμη στην οποία βρίσκεται το αντικείμενο με όνομα x:

```
tracemem(x)
```

Η εντολή επιστρέφει έναν αριθμό (σε δεκαεξαδικό σύστημα αρίθμησης) που είναι η τρέχουσα θέση του αντικειμένου με όνομα x, π.χ.

```
[1] "<00007FF4E93A0010>"
```

Προφανώς ο αριθμός αυτός αφορά το x στο συγκεκριμένο περιβάλλον εκτέλεσης εντολών, άρα θα διαφέρει αν δοκιμάσετε το παράδειγμα. Εφόσον το αντικείμενο x παρακολουθείται, οποιαδήποτε αλλαγή γίνει σε αυτό που θα οδηγήσει σε αλλαγή της θέσης του στη μνήμη, θα αναφερθεί με σχετικό μήνυμα. Για παράδειγμα, αν εκτελεστεί η εντολή:

```
x[3000]<-10
```

Θα εμφανιστεί ένα μήνυμα όπως το παρακάτω:

```
tracemem[0x00007ff4e93a0010 -> 0x00007ff4e9210010]:
```

Στο παράδειγμα γίνεται ανάθεση της τιμής 10 στη θέση 3000 του διανύσματος με όνομα x. Στην περίπτωση αυτή[146] προκάλεσε, τη δημιουργία ενός αντιγράφου του αντικειμένου x στη μνήμη, την εφαρμογή της αλλαγής σε αυτό και την ανάθεση του ονόματος x στο νέο αντικείμενο. Το παλαιό διαγράφηκε εφόσον δεν υπάρχει άλλη μεταβλητή (όνομα) που το αφορά. Το μήνυμα, είναι αποτέλεσμα της tracemem και δείχνει την προηγούμενη και τη νέα θέση στη μνήμη του αντικειμένου που αφορά το όνομα x. Η παρακολούθηση του x θα συνεχιστεί έως ότου να διακοπεί με την εντολή untracemem(x).

Εκτός του εργαλείου profiling, άλλοι τρόποι να γίνει μια (κατά προσέγγιση) εκτίμηση του χρόνου εκτέλεσης ενός τμήματος κώδικα είναι με πρόσθετες εντολές ενσωματωμένες σε αυτόν. Με χρήση των συναρτήσεων system.time και Sys.time του πακέτου ‘base’, ο επόμενος κώδικας χρονομετρά μία εντολή Sys.sleep(5). Η εντολή αυτή προκαλεί αδράνεια 5 δευτερολέπτων (δηλαδή ο κώδικάς μας «κοιμάται» για 5 δευτερόλεπτα)[147]:

system.time(Sys.sleep(5))

Παρεμφερές αποτέλεσμα θα φέρει ο παρακάτω κώδικας:

```
t1 <- Sys.time()  # έναρξη
Sys.sleep(5)       # η εντολή (ή εντολές) που χρονομετρούμε
t2 <- Sys.time()  # λήξη
t2 - t1            # η διαφορά (σε msec)
```

Πρέπει να αναφερθεί πως τέτοιες μετρήσεις (με το εργαλείο profiling ή με εντολές) επηρεάζονται προφανώς από το σύστημα στο οποίο εκτελείται ο κώδικας και εξαρτώνται από τον φόρτο εργασίας που εκτελεί ταυτόχρονα. Πιθανότατα, θα διαφοροποιηθούν αν γίνουν οι ίδιες δοκιμές είτε στο ίδιο είτε σε άλλο σύστημα. Τέλος, στο CRAN διατίθενται πακέτα για χρονομετρήσεις και εκτίμηση απόδοσης του κώδικα, όπως τα ΄tictoc’ και ‘rbenchmark’.

(βλ.§6.5 Κλάσεις αναφοράς).

[144] Ειδικά σε αντικείμενα τύπου list ή τύπων βασισμένων σε αυτό list (βλ.§4.2.1 Ο τύπος list (λίστα))., εφαρμόζεται συνήθως shallow copy.

[145] Παρέχονται από το πακέτο ΄base’. Βλ. και συνάρτηση .Internal π.χ. εντολή .Internal(inspect(x)).

[146] Αυτό δεν συμβαίνει πάντα, βλ. παραπάνω.

[147] Η συγκεκριμένη συνάρτηση επιλέχθηκε ως παράδειγμα γιατί είναι προκαθορισμένος ο χρόνος εκτέλεσής της. Αντί του Sys.sleep(5) μπορεί να «χρονομετρηθεί» οποιαδήποτε έκφραση, π.χ. η εντολή system.time(print("Hello")) επιστρέφει (κατά προσέγγιση) τον χρόνο που χρειάστηκε για να εκτελεστεί η δοθείσα εντολή print.

3.1.5 Εργαλείο αναφοράς (report)

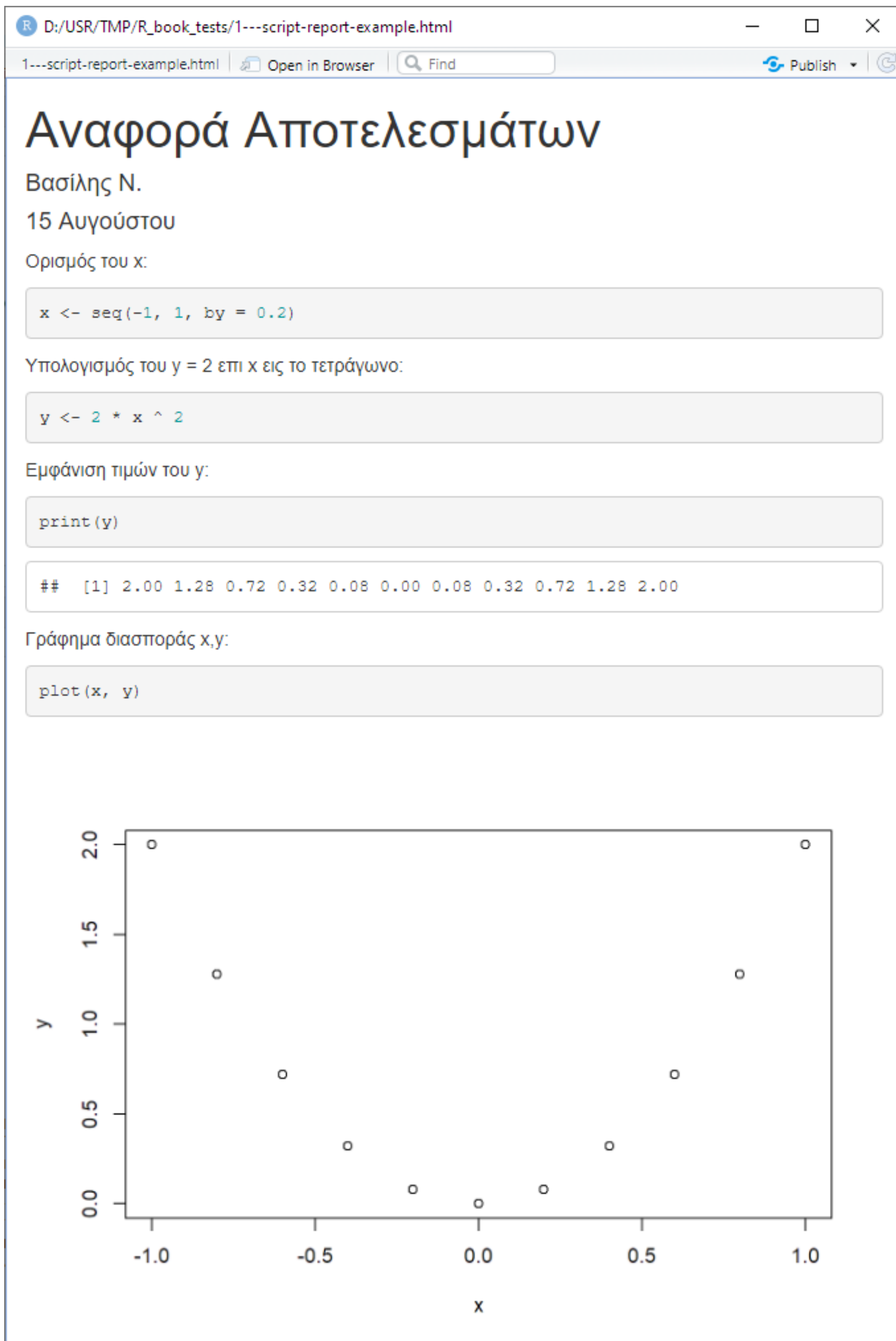
Το εργαλείο αυτό δημιουργεί ένα έγγραφο (αναφορά) που περιέχει τον κώδικα και τα αποτελέσματα της εκτέλεσής του. Η αναφορά αυτή μπορεί να είναι σε μορφή αρχείου για χρήση στο web (html), αρχείου Adobe Acrobat (pdf) ή αρχείου Microsoft Word (doc). Η αναφορά δημιουργείται απλώς επιλέγοντας το μενού File/Compile Report και είναι ένας εύκολος τρόπος να γίνει διαμοιρασμός των αποτελεσμάτων, κάτι ιδιαίτερα χρήσιμο σε πολλά εργασιακά περιβάλλοντα.

Σε πρώτο στάδιο το εργαλείο αναφοράς δημιουργεί αυτόματα ένα προσωρινό έγγραφο βασισμένο στο τρέχον σενάριο και γραμμένο σε γλώσσα RMarkdown. Το RMarkdown είναι μια παραλλαγή της γλώσσας δημιουργίας περιεχομένου markdown (md) με επεκτάσεις που σχετίζονται με την R (π.χ. επιτρέπει την ενσωμάτωση κώδικα R ή διαδραστικών R αντικειμένων Shiny). Η RMarkdown χρησιμοποιείται γενικότερα στην R για δημιουργία εγγράφων και παρουσιάζεται εκτενέστερα σε άλλη ενότητα του βιβλίου (βλ. §9.4 Δυναμικά έγγραφα). Εκεί, αναφέρονται περισσότερα για τη συγγραφή εγγράφων σε αυτή τη γλώσσα.

Σε αντίθεση με τα έγγραφα που γράφονται απευθείας σε RMarkdown, η αυτόματη αναφορά που δημιουργεί το εργαλείο αναφοράς είναι προκαθορισμένη και είναι δύσκολο να αλλάξει μορφή, καθώς βασίζεται αυστηρά στο τρέχον σενάριο και ο κώδικας RMarkdown διαγράφεται μετά τη δημιουργία της αναφοράς. Όμως, κάποια σχόλια ειδικής μορφής στον κώδικα R του σεναρίου, αξιοποιούνται από το εργαλείο αναφοράς για να οριστούν (αν χρειάζεται) κάποιες παράμετροι της αναφοράς (όπως το κείμενο στον τίτλο και το όνομα του συγγραφέα) ή για να προστεθεί κείμενο στο εσωτερικό της αναφοράς. Η σύμβαση ορίζει πως στα ειδικά αυτά σχόλια πρέπει να έχει προστεθεί μια απόστροφος μετά τον χαρακτήρα σχολίων #, δηλαδή να ξεκινούν με τους χαρακτήρες #'.

Για παράδειγμα, ο κώδικας σεναρίου που δίνεται παρακάτω παράγει την αναφορά της Εικόνας 3.3:

```
#' ---
#' title: "Αναφορά Αποτελεσμάτων"
#' author: "Βασίλης Ν."
#' date: "15 Αυγούστου"
#' ---
#' Ορισμός του x:
x <- seq(-1, 1, by = 0.2)
#' Υπολογισμός του y = 2 επι x εις το τετράγωνο:
y <- 2 * x ^ 2
#' Εμφάνιση τιμών του y:
print(y)
#' Γράφημα διασποράς x,y:
plot(x, y)
```



Εικόνα 3.3 : *Παράδειγμα αναφοράς σε μορφή html και βασισμένης σε ένα απλό σενάριο R.*

Στο παραπάνω σενάριο, οι πρώτες γραμμές (που περιέχουν σχόλια γραμμένα σε προκαθορισμένη μορφή) ορίζουν τον τίτλο, ημερομηνία και όνομα συγγραφέα που θα εμφανίζεται στην αναφορά, ενώ ο κυρίως κώδικας του σεναρίου ακολουθεί αμέσως μετά. Ο κώδικας ενσωματώνει μερικά ακόμα ειδικής μορφής σχόλια (που ξεκινούν επίσης με #') με τα οποία γίνεται προσθήκη κειμένου σε συγκεκριμένα σημεία της παραχθείσας αναφοράς.

3.1.6 Φάκελοι αρχείων και ο τρέχων φάκελος εργασίας

Έγινε πλέον η πρώτη ουσιαστική αναφορά στην αποθήκευση ή ανάκληση δεδομένων και κώδικα από και προς αρχεία. Η R, χρησιμοποιεί αρχεία σε διάφορα σημεία (φακέλους/υποκαταλόγους) στον υπολογιστή, όμως η τρέχουσα διαδρομή, στην οποία τόσο οι κινήσεις όσο και οι εντολές του χρήστη θα αναζητούν, θα διαβάζουν ή θα δημιουργούν αρχεία και φακέλους[148], αναφέρεται ως current working directory (τρέχων φάκελος/υποκατάλογος εργασίας). Περισσότερα για την πρόσβαση σε αρχεία και την εισαγωγή/εξαγωγή δεδομένων από αυτά μέσω κώδικα R παρουσιάζονται σε άλλη ενότητα (βλ.§9.1 Εισαγωγή και εξαγωγή δεδομένων). Όμως το current working directory είναι ήδη σημαντικό, καθώς ορίζει ένα σημείο εκκίνησης για τον κώδικα και τις εντολές του χρήστη που αναζητούν ή δημιουργούν αρχεία.

Στο RStudio μπορείτε να ορίσετε το current working directory από τις επιλογές του μενού Session/Set Working Directory ή να πλοηγηθείτε (σε κάποιο βαθμό) στους φακέλους της συσκευής σας και να ορίσετε το current working directory από τα εργαλεία της σχετικής καρτέλας (βλ.§1.4.2.4 Η καρτέλα Files). Επίσης, χρησιμότατη είναι η επιλογή που ορίζει ως current working directory τον φάκελο στον οποίον βρίσκεται ένα σενάριο[149] στο οποίο γίνεται επεξεργασία (επιλογή μενού: Session/Set Working Directory/To Source File Location). Συναρτήσεις που σχετίζονται με το current working directory είναι οι setwd και getwd:

Δοκιμάστε:	**Σχόλιο**
`getwd()`	Λεκτικό με τη διαδρομή για τον τρέχοντα φάκελο εργασίας.
`setwd("c:/test")`	Ορισμός του φακέλου test στο C: ως τον τρέχοντα φάκελο εργασίας.
`dir()`	Εμφανίζει τα αρχεία στον τρέχοντα φάκελο εργασίας[150].

Το setwd στο παραπάνω παράδειγμα θα λειτουργήσει μόνο εφόσον υπάρχει ο συγκεκριμένος φάκελος test στη συσκευή με C:, κάτι που κάνει τη διαδρομή του παραδείγματος «απόλυτη». Το πρόβλημα που δημιουργείται στην περίπτωση αυτή είναι ότι μπορεί η διαδρομή να μην υπάρχει σε άλλους υπολογιστές και τότε η παραπάνω εντολή θα αποτύχει. Επιπρόσθετα, σε τέτοιου είδους εντολές καλό είναι να λαμβάνεται υπόψη πως οι διαφορετικοί τύποι λειτουργικών συστημάτων (ΛΣ) τα οποία υποστηρίζει η R εφαρμόζουν διάφορες συμβάσεις για τις διαδρομές φακέλων. Μερικά παραδείγματα είναι τα ακόλουθα: Τα Windows διαχωρίζουν ονόματα φακέλων (καταλόγων) και υποφακέλων (υποκαταλόγων) με τον χαρακτήρα '\' ενώ στο Linux χρησιμοποιείται το '/' για τον ίδιο σκοπό. Σε διάφορους τύπους ΛΣ (εκτός των Windows) τα ονόματα των φακέλων και των αρχείων είναι case-sensitive και τα κεφαλαία δεν εξομοιώνονται με τους αντίστοιχους μικρούς χαρακτήρες. Τέλος, ενώ στα Windows η όποια διαδρομή ξεκινά συνήθως από τον προσδιορισμό της συσκευής αποθήκευσης (π.χ. C:\) στο Linux - για παράδειγμα - κάτι τέτοιο δεν υπάρχει. Η R χρησιμοποιεί μερικές συμβάσεις υιοθετημένες από το Linux (π.χ. τον χαρακτήρα '~' για το home folder του χρήστη και το '/' για τον διαχωρισμό φακέλων) τις οποίες κατά την εκτέλεση του κώδικα τις μεταφράζει στις αντίστοιχες του εκάστοτε ΛΣ. Καλό πάντως είναι να αποφεύγονται συμβάσεις και σύμβολα που ίσως κάνουν τον κώδικά R ασύμβατο με άλλους υπολογιστές ή ΛΣ, ειδικά αν πρόκειται να μοιραστεί ο κώδικας αυτός σε άλλους χρήστες καθώς και να αποφεύγεται ο ορισμός απόλυτων διαδρομών για τα αρχεία και να γίνεται αξιοποίηση του ορισμού ενός project (βλ.§3.1.7 Έργο (project)) για όλα τα αρχεία που ίσως αποτελούν μέρος της υλοποίησης μιας σύνθετης λύσης λογισμικού. Ένα ακόμα βοήθημα για την επίτευξη συμβατότητας του κώδικα με άλλα ΛΣ είναι η συνάρτηση file.path που δέχεται μια σειρά από λεκτικά φακέλων και υποφακέλων και τα ενώνει σε ένα ενιαίο λεκτικό της διαδρομής, κατάλληλα διαμορφωμένο για το ΛΣ στο οποίο εκτελείται ο κώδικας. Τη συνάρτηση αυτή καθώς και κάποιες άλλες συναρτήσεις χειρισμού φακέλων χρησιμοποιεί το επόμενο παράδειγμα:

Δοκιμάστε:	**Σχόλιο**
`op<-getwd()`	Η διαδρομή προς τον τρέχοντα φάκελο εργασίας σε μεταβλητή op.
`p<-file.path(op,"test")`	Στην p, η ίδια διαδρομή συν ένας φάκελος με όνομα "test".
`dir.create(p,recursive=T)`	Δημιουργία της διαδρομής που έχει η p (δηλαδή του φακέλου "test")
`dir.exists(p)`	Υπάρχει η διαδρομή p (με τον φάκελο "test"); Αν ναι τότε TRUE.
`setwd(p)`	Ορισμός της διαδρομής p ως τον τρέχοντα φάκελο εργασίας.

148 Αυτό ισχύει εφόσον στις σχετικές εντολές δεν έχουν οριστεί συγκεκριμένες «απόλυτες» διαδρομές για τη θέση του αρχείου στον υπολογιστή (κάτι το οποίο αντενδείκνυται).

149 βλ.§3.1 Σενάρια (R-script).

150 Η συνάρτηση δέχεται και παραμέτρους που επιτρέπουν την εμφάνιση των ονομάτων αρχείων σε άλλους φακέλους, το φιλτράρισμα τους με χρήση patterns κλπ.

`getwd()`	Ποιος είναι ο τρέχων φάκελος εργασίας; Αν όλα πήγαν καλά, ίδιο με το p.
`setwd(op)`	Επιστροφή στον αρχικό φάκελο εργασίας (στο op) ως τρέχοντα.
`unlink(p,recursive=T)`	Διαγραφή της διαδρομής στο p (δηλαδή του φακέλου “test”).

Για να διευκολυνθεί η συγγραφή κώδικα χειρισμού αρχείων και φακέλων που θα είναι περαιτέρω ευανάγνωστος και ανεξάρτητος πλατφόρμας (cross-platform), δηλαδή συμβατός με τα διαφορετικά ΛΣ στα οποία τρέχει η R, μπορούν εναλλακτικά να χρησιμοποιηθούν οι τυποποιημένες συναρτήσεις που παρέχει το πακέτο ‘fs’ [45], διαθέσιμο στο CRAN. Μετά την εγκατάσταση του πακέτου ‘fs’ (βλ.§1.5 Χρήση και διαχείριση πακέτων) το παραπάνω παράδειγμα μπορεί να γραφεί και ως:

Δοκιμάστε:	**Σχόλιο**
`library("fs")`	Σύνδεση του πακέτου ‘fs’ (προαιρετικό βήμα, αν δεν έχει γίνει ήδη).
`op<-getwd()`	Η διαδρομή προς τον τρέχοντα φάκελο εργασίας σε μεταβλητή op.
`p<-path(op,"test")`	(fs) Στην p, η ίδια διαδρομή συν ένας φάκελος με όνομα “test”.
`dir_create(p)`	(fs) Δημιουργία της διαδρομής που έχει η p (δηλαδή του φακέλου “test”).
`dir_exists(p)`	(fs) Υπάρχει η διαδρομή p (με τον φάκελο “test”); Αν ναι τότε TRUE.
`setwd(p)`	Ορισμός της διαδρομής p ως τον τρέχοντα φάκελο εργασίας.
`getwd()`	Ποιος είναι ο τρέχων φάκελος εργασίας; Αν όλα πήγαν καλά, ίδιο με το p.
`setwd(op)`	Επιστροφή στον αρχικό φάκελο εργασίας (στο op) ως τρέχοντα.
`dir_delete(p)`	(fs) Διαγραφή της διαδρομής στο p (δηλαδή του φακέλου “test”).

Τέλος, όταν στη λύση που υλοποιείται εμπλέκονται πολλά αρχεία, χρήσιμη είναι η δημιουργία ενός project και η τοποθέτησή τους σε αυτό, κάτι που περιγράφεται στην επόμενη ενότητα.

3.1.7 Έργο (project)

Ακόμα και με προσεκτική χρήση όσων αναφέρονται στην προηγούμενη ενότητα, η διαχείριση των αρχείων μιας λύσης λογισμικού, δηλαδή των αρχείων ενός έργου που υλοποιείται σε R ή και σε συνδυασμό με άλλες γλώσσες προγραμματισμού[151] μπορεί να παρουσιάσει δυσκολίες. Η υλοποιούμενη λύση μπορεί να αποτελείται από πολλά αρχεία κώδικα αλλά και αρχεία δεδομένων ή άλλου περιεχομένου. Τα αρχεία πρέπει να είναι τοποθετημένα προσεκτικά ώστε να είναι έγκυρη η όποια διαδρομή προς αυτά υπάρχει στον κώδικα. Το έργο μπορεί να αναπτύσσεται από ομάδα προγραμματιστών σε διαφορετικούς υπολογιστές και έτσι η απόλυτη διαδρομή προς τα αρχεία του έργου να διαφέρει από υπολογιστή σε υπολογιστή. Ή μπορεί απλώς τα αρχεία της λύσης να χρειαστεί να αλλάξουν θέση (άλλον φάκελο στον υπολογιστή ή να μεταφερθούν σε έναν server). Θα μπορούν να εντοπιστούν από κώδικα που τα αναζητεί; Ακόμα και στην περίπτωση που η ανάπτυξη γίνεται μόνο από έναν χρήστη και σε συγκεκριμένο υπολογιστή, πιθανότατα υπάρχουν και άλλα αρχεία που αφορούν άλλα έργα στα οποία ο ίδιος εργάζεται ταυτόχρονα ή έχει εργαστεί στο παρελθόν, αρχεία που δεν σχετίζονται με το τρέχον έργο. Εύκολα μπορούν όλα αυτά να αρχίσουν να συνωστίζονται στο home folder ή σε άλλους φάκελους εργασίας του χρήστη, δυσκολεύοντας την εργασία του.

Καλό είναι όλα τα αρχεία που δημιουργούνται και σχετίζονται με την ίδια λύση λογισμικού να τοποθετούνται μαζί σε έναν δικό τους φάκελο, διαχωρισμένα από τα υπόλοιπα αρχεία που είναι μη-σχετικά με το συγκεκριμένο έργο. Ένα project της R (και κατ’ επέκταση του RStudio) ξεκινά από αυτή την τακτική και προσθέτει διάφορες χρήσιμες λειτουργίες.

Για κάθε έργο μπορεί να γίνει η δημιουργία ενός νέου R project που θα φιλοξενεί τα σχετικά με αυτό αρχεία. Στο νέο project (το οποίο στο RStudio δημιουργείται με την επιλογή μενού File/New Project) δίνεται ένα συγκεκριμένο όνομα και ορίζεται ένας συγκεκριμένος φάκελος για τα αρχεία του. Σε κάποιους τύπους project η R επιβάλει συγκεκριμένη δομή που πρέπει να έχει ο σχετικός φάκελος, δηλαδή ποιους υποκαταλόγους πρέπει να περιέχει και σε ποιους από αυτούς πρέπει να τοποθετηθούν τα αρχεία ανάλογα με τον τύπο τους (περισσότερα για αυτό παρακάτω). Έτσι το RStudio δίνει διάφορες επιλογές προκαθορισμένων τύπων έργου και δημιουργεί αυτόματα τα βασικά μέρη της δομής τους (υποκαταλόγους κλπ.). Μέσα στον φάκελο του έργου τοποθετεί μεταξύ άλλων: (α) Ένα αρχείο aname.Rproj (αν aname το υποτιθέμενο όνομα του έργου) που περιέχει τις ρυθμίσεις του project. (β) Αποθηκευμένο User Workspace (βλ.§2.8 Αποθήκευση του User Workspace) που δημιουργήθηκε καθόσον γινόταν εργασία στο συγκεκριμένο έργο. Στην περίπτωση αυτή, το σχετικό κρυφό

[151] βλ.§7.1 Πολυγλωσσικές λύσεις.

αρχείο .RData αποθηκεύεται στον φάκελο του έργου ώστε κάθε έργο που φορτώνεται να επαναφέρει μόνο τις σχετικές με αυτό μεταβλητές. (γ) Αντίστοιχα, ένα κρυφό αρχείο .RHistory που επαναφέρει το ιστορικό εντολών κατά την εργασία με το συγκεκριμένο έργο[152].

Έτσι, με το άνοιγμα ενός υπάρχοντος project (δηλαδή του σχετικού αρχείου με επέκταση ονόματος .Rproj) επανέρχεται η κατάσταση που υπήρχε όταν αυτό αποθηκεύτηκε. Ξεκινά μία νέα συνεδρία της R, επαναφέρονται τα ανοικτά έγγραφα κώδικα κλπ., οι ρυθμίσεις του RStudio, οι μεταβλητές, το ιστορικό κλπ. Όμως βασικότερο είναι πως ο φάκελος του έργου ορίζεται ως ο τρέχων φάκελος εργασίας (current working directory).

Όλα τα παραπάνω κάνουν τον φάκελο του project μια αυτόνομη μονάδα που μπορεί να διαμοιραστεί ή να μετακινηθεί. Αν ταυτόχρονα υιοθετηθεί στον κώδικα η χρήση μόνο σχετικών διαδρομών (που πάντα ξεκινούν από τον τρέχοντα φάκελο εργασίας, άρα πλέον τον φάκελο του έργου) και οι διαδρομές αυτές αναφέρονται σε θέσεις εντός του έργου για όλα τα αρχεία και τους φακέλους στα οποία ίσως ανατρέχει ο κώδικας, εξαλείφονται σε μεγάλο βαθμό τα πιθανά προβλήματα που αναφέρθηκαν στην προηγουμένη παράγραφο (§3.1.6 Φάκελοι αρχείων και ο τρέχων φάκελος εργασίας).

Επιπρόσθετα, με τον φάκελο του project να περιέχει όλα όσα χρειάζονται για το συγκεκριμένο έργο διευκολύνεται και η αξιοποίηση συστημάτων ελέγχου έκδοσης (version control system) για την αποθήκευση του έργου και κατ' επέκταση η συνεργασία σε αυτό από μέλη μιας ομάδας. Για τον λόγο αυτόν το RStudio υποστηρίζει τη σύνδεση του project με τα σχετικά συστήματα και αναγνωρίζει αυτόματα την ύπαρξη των ειδικών φακέλων τους (.git για το σύστημα Git ή .svn για το Apache Subversion) [46].

Όπως αναφέρθηκε παραπάνω, κάποιοι τύποι project πρέπει να ακολουθούν ένα συγκεκριμένο πρότυπο για τη δομή τους. Έτσι, ένα έργο που έχει στόχο τη δημιουργία ενός πακέτου (package) πρέπει να ακολουθεί το σχετικό πρότυπο ώστε να μπορεί να δημιουργηθεί το τελικό package από τα σχετικά εργαλεία. Όμως, καθώς το πρότυπο ενσωματώνει κάποιες «βέλτιστες πρακτικές», είναι καλή τακτική να υιοθετείται (μερικώς ή πλήρως) η δομή που προτείνεται για τα πακέτα και σε άλλα απλούστερα έργα δημιουργίας κώδικα R, όπως π.χ. για μια εργασία ανάλυσης δεδομένων. Εξάλλου, τα βασικά στοιχεία του προτύπου δεν είναι ιδιαίτερα πολύπλοκα. Στο πρότυπο αυτό προτείνεται η τοποθέτηση όλου του πηγαίου κώδικα (σεναρίων R κλπ.) σε έναν υποκατάλογο με όνομα src (ή με όνομα R), τα αρχεία δεδομένων να βρίσκονται σε υποκατάλογο με όνομα data, να υπάρχει ένα κατάλληλα μορφοποιημένο αρχείο περιγραφής του έργου (με όνομα DESCRIPTION) στο οποίο να ορίζονται και οι εξαρτήσεις (dependencies) που ίσως έχει από διάφορα πακέτα, να υπάρχει τεκμηρίωση του κώδικα (με σχόλια στον κώδικα αλλά και με σχετικά αρχεία στον υποκατάλογο με όνομα man), ενώ αν υπάρχει και ευρύτερο συνοδευτικό υλικό που εξηγεί τη λειτουργία του κώδικα (π.χ. επεξήγηση της προσέγγισης, κάποια σχετική επιστημονική δημοσίευση κλπ.) να προστίθεται ως vignettes. Για περισσότερα για τη δομή που πρέπει να έχει ένας τέτοιος φάκελος αναφέρονται στη σχετική με ανάπτυξη πακέτων ενότητα[153]. Πάντως, όπως προαναφέρθηκε, αν στόχος δεν είναι η ανάπτυξη πακέτου, το παραπάνω πρότυπο δομής μπορεί να υιοθετηθεί μόνο μερικώς. Να εφαρμοστούν δηλαδή όσες από τις παραπάνω προτάσεις κριθούν κατάλληλες ή και να τροποποιηθούν για τις συγκεκριμένες ανάγκες και τον τύπο εργασίας (π.χ. να προστεθεί ένας φάκελος για όλα τα αρχεία εξόδου που παράγονται από τον κώδικα μιας εργασίας ανάλυσης δεδομένων κλπ.)

3.1.8 Μερικές παρατηρήσεις σχετικές με τη συγγραφή σεναρίων

Μόνο όταν ξεκινήσετε να δημιουργείτε και χρησιμοποιείτε τα δικά σας σενάρια (R script) θα κατανοήσετε την αξία τους. Ακόμα και αν αρχικά δοκιμάσετε να χρησιμοποιείτε τον επεξεργαστή πηγαίου κώδικα (source editor) απλώς για εισαγωγή εντολών R (και όχι απευθείας στο Console) θα έχετε ιδιαίτερη ευελιξία στον χειρισμό του κώδικά σας, κάτι που από μόνο του επιταχύνει την επίτευξη αποτελεσμάτων, ακόμα και όταν δεν είναι τελικός στόχος η δημιουργία ενός ολοκληρωμένου, αυτόνομου σεναρίου. Με τον κώδικα να γράφεται μέσω του επεξεργαστή πηγαίου κώδικα είναι πολύ πιο εύκολο να γίνουν διορθώσεις ή να επαναληφθούν τα όποια βήματα, εκτελώντας όλον τον κώδικα (με Source) ή τμήματα του (με Run). Όμως η πραγματική αξία των σεναρίων είναι ότι μπορούν να εξελιχθούν σε μια ολοκληρωμένη υλοποίηση κάποιας λύσης, να είναι ουσιαστικά αυτόνομα προγράμματα, άρα η υλοποίηση να αποθηκευτεί, να χρησιμοποιηθεί πάλι στο μέλλον, αλλά και να ελεγχθεί.

---

[152] Οι σχετικές ρυθμίσεις στο RStudio για τα (β) και (γ) βρίσκονται στο μενού Tools/Global Options, κατηγορία General, καρτέλα Basic.

[153] βλ.§8.2 Δομή φακέλου για δημιουργία πακέτου.

Όταν πλέον ο στόχος είναι τέτοιου τύπου σενάρια, καλό είναι να λαμβάνεται υπόψη πως τα αρχεία αυτά ίσως μοιραστούν σε άλλους χρήστες, εκτελεστούν (από εμάς ή άλλους) σε άλλα συστήματα κλπ. Όπως προαναφέρθηκε, όταν εκτελείται ο κώδικας από ένα σενάριο το αποτέλεσμα των εντολών που περιέχει είναι ίδιο με αυτό που θα έφερναν αν είχαν γραφτεί στο Console (πλην της μη εμφάνισης επιστρεφόμενων τιμών, αν εκτελείται με source). Έτσι, μεταβλητές που δημιουργούνται προστίθενται στο Global Environment και παραμένουν εκεί μετά την εκτέλεση του σεναρίου, μεταβλητές που προϋπήρχαν της εκτέλεσης του σεναρίου μπορούν να χρησιμοποιηθούν από τις εντολές του σεναρίου κλπ. Η σύσταση (όχι απαραίτητα κατάλληλη για όλες τις περιπτώσεις) προς τους δημιουργούς σεναρίων, είναι να γράφουν σε αυτά κατά το δυνατόν αυτόνομο και μεταφέρσιμο (portable) κώδικα. Για παράδειγμα, ο κώδικας να μην εξαρτάται από κάποιες προ-υπάρχουσες μεταβλητές στο Global Environment ή αρχεία δεδομένων σε συγκεκριμένες θέσεις στον υπολογιστή για να λειτουργήσει σωστά, καθώς οι μεταβλητές ή τα αρχεία αυτά μπορεί να μην υπάρχουν σε κάποια μελλοντική συνεδρία ή σε ένα άλλο σύστημα, στα οποία ίσως κληθεί να εκτελεστεί το σενάριο. Επιπρόσθετα, το σενάριο να συνδέει (αν όχι εγκαθιστά κιόλας) τα απαραίτητα για αυτό πακέτα (βλ.§1.5.3 Εγκατάσταση και διαχείριση πρόσθετων πακέτων) και να είναι κατά το δυνατόν συμβατό με τις εκδόσεις της R και των πακέτων αυτών σε άλλα συστήματα που ίσως εκτελεστεί. Να λαμβάνονται υπόψη διαφορές που ενδεχομένως να υπάρχουν ανάμεσα στα ΛΣ και να είναι κατά το δυνατόν ανεξάρτητο πλατφόρμας (π.χ. βλ.§3.1.6 Φάκελοι αρχείων και ο τρέχων φάκελος εργασίας). Τέλος, αν χρησιμοποιούνται Ελληνικοί ή άλλοι διεθνείς χαρακτήρες και συμβάσεις στον κώδικα του σεναρίου, καλό θα είναι στην αρχή του σεναρίου να υπάρχει η συνάρτηση Sys.setlocale με τις αντίστοιχες τοπικές ρυθμίσεις (για σχετικά προβλήματα εκτέλεσης σεναρίων και τις Ελληνικές ρυθμίσεις βλ.§Π.1.1 Προβλήματα που σχετίζονται με τα Ελληνικά). Συνοψίζοντας, μερικές συστάσεις (όχι απαραίτητα κατάλληλες σε όλες τις περιπτώσεις) προς τους δημιουργούς των R script είναι:

- Χρησιμοποιείτε επεξηγηματικά ονόματα στις μεταβλητές. Αυτό βοηθά στην κατανόηση του κώδικα από εσάς ή τρίτους και την αποφυγή λαθών. Ο επεξεργαστής πηγαίου κώδικα στο RStudio δίνει τη δυνατότητα μαζικής μετονομασίας μεταβλητών (μενού Code/Rename In Scope) που μπορείτε να χρησιμοποιήσετε για τον σκοπό αυτό.
- Χρησιμοποιείτε σχόλια στον κώδικα. Σωστά σχόλια ποτέ δεν βλάπτουν, ειδικά αν χρειαστεί να κατανοήσει τον κώδικα σας κάποιος τρίτος ή εσείς μετά από καιρό (βλ.§3.1.2 Σχολιασμός του κώδικα) .
- Υιοθετήστε ένα ενιαίο στυλ γραφής του κώδικα στο σενάριο ώστε να διαβάζεται εύκολα. Η ίδια η γλώσσα δεν επηρεάζεται από την οπτική διάταξη των εντολών (π.χ την ευθυγράμμιση κομματιών κώδικα (identation) με εσοχές και tab, τη χρήση κενών ή τις αλλαγές γραμμών στον κώδικα). Η R επιτρέπει να υπάρχουν πολλές εντολές στην ίδια γραμμή (αρκεί να τελειώνει καθεμία από αυτές με τον χαρακτήρα ';'). Όμως ο κώδικας γίνεται πιο ευανάγνωστος όταν τοποθετείτε την κάθε εντολή μόνη της σε δική της γραμμή (οπότε δεν είναι και απαραίτητο το ';'). Σε κάθε περίπτωση, αξιοποιήστε τα εργαλεία που δίνει ο επεξεργαστής πηγαίου κώδικα στο RStudio για αυτόματη μορφοποίηση (μενού Code/Reformat Code) και δημιουργία εσοχών (μενού Code/Reindent Lines), καθώς θα κάνουν τον κώδικα πιο ευανάγνωστο.
- Υιοθετήστε τη χρήση μπλοκ κώδικα. Θα αναφερθούμε στα μπλοκ κώδικα παρακάτω (βλ.§3.2.1 Μπλοκ κώδικα) αλλά καλό είναι να αναφερθεί και εδώ κάτι σχετικό. Η R δεν απαιτεί τη δημιουργία μπλοκ κώδικα όταν αφορά μόνο μια εντολή και είναι απαραίτητο μόνο όταν πρέπει να ομαδοποιηθούν πολλές εντολές. Όμως η χρήση μπλοκ κώδικα πάντα, (ακόμα και στην περίπτωση μίας και μοναδικής εντολής) σε ορισμό συναρτήσεων, βρόχων (loops) κλπ. είναι καλή προγραμματιστική συνήθεια προς αποφυγή αβλεψιών και λαθών. Το μπλοκ κάνει τον κώδικα πιο ευανάγνωστο ορίζοντας με ξεκάθαρο τρόπο το σημείο όπου αρχίζει και το σημείο όπου τελειώνει ένα λειτουργικό κομμάτι κώδικα (το σώμα της συνάρτησης, ο κώδικας που θα επαναλαμβάνει ο βρόχος κλπ.), άρα βοηθά στην αποφυγή λαθών (βλέπε και σχετική σημείωση στο help(Control) και §3.2.2.3 Επαναλήψεις (βρόχοι ή loop)).
- Σύντομα, τα σενάρια σας (και όχι μόνο) θα περιέχουν συναρτήσεις που εσείς αναπτύσσετε, (βλ.§5.1.1 Δημιουργία συναρτήσεων) καθώς είναι ο καλύτερος τρόπος να έχετε λειτουργίες και κώδικα που μπορούν να επαναχρησιμοποιηθούν (reusable code). Οι συναρτήσεις σας καλό είναι να χρησιμοποιούν ως είσοδο μόνο τις παραμέτρους τους. Όπως προαναφέραμε και για τα ίδια τα σενάρια, ένας κώδικας καλό είναι να είναι κατά το δυνατόν αυτοτελής, να μην εξαρτάται από άλλες εξωτερικές μεταβλητές και ένα συγκεκριμένο περιβάλλον για να λειτουργήσει. Αφενός, κανείς δεν μπορεί να εξασφαλίσει την ύπαρξη ενός συγκεκριμένου

περιβάλλοντος και εξωτερικών μεταβλητών όταν η συνάρτηση αυτή χρησιμοποιηθεί στο μέλλον, αφετέρου αυτό αναιρεί την έννοια των συναρτήσεων ως επαναχρησιμοποιήσιμων, αυτοτελών οντοτήτων που παρέχουν αυτόνομα κάποια λειτουργία.

3.2 Προγραμματιστικές τεχνικές

Τα σενάρια (βλ.§3.1 Σενάρια (R-script)) προσφέρουν την πρώτη ουσιαστική δυνατότητα προγραμματισμού που παρουσιάζεται στο βιβλίο αυτό. Η ενότητα που ακολουθεί συνοψίζει κάποιες βασικές προγραμματιστικές τεχνικές όπως τις παρέχει η γλώσσα R. Τα παρακάτω μπορούν να εκτελεστούν και ως απευθείας εντολές στο Console, αλλά συνήθως χρησιμοποιούνται ως μέρος κάποιας πιο πολύπλοκης υλοποίησης (π.χ. στον κώδικα μιας συνάρτησης ή ενός σεναρίου) όπου συνήθως είναι και πολύ πιο απαραίτητα.

3.2.1 Μπλοκ κώδικα

Ένα μπλοκ κώδικα (code block) ή μπλοκ εντολών ή γκρουπ κώδικα (code group) ομαδοποιεί εντολές έτσι ώστε αυτές να μην αντιμετωπίζονται ως ενιαίες αλλά ως μια αναπόσπαστη, ενοποιημένη ομάδα. Με αυτό τον τρόπο, πολλαπλές εντολές μπορούν να τοποθετηθούν σε οποιοδήποτε σημείο του κώδικα επιτρέπεται μια εντολή. Άρα, οπουδήποτε η R δέχεται την εισαγωγή μιας εντολής ενώ θα θέλαμε να εκτελεστούν πολλές, τότε θα πρέπει αυτές να δίνονται ομαδοποιημένες μέσα σε ένα μπλοκ κώδικα. Στην R ένα μπλοκ κώδικα ξεκινά με τον χαρακτήρα '{' και κλείνει με τον χαρακτήρα '}'[154]. Εντός του μπλοκ υπάρχουν μια ή περισσότερες εντολές (συνήθως περισσότερες από μία καθώς τότε έχει νόημα η ομαδοποίηση τους) αλλά επιτρέπεται να είναι και κενό (χωρίς κάποια εντολή). Επίσης, επιτρέπεται ο ορισμός «εμφωλευμένων» μπλοκ κώδικα, δηλαδή μπλοκ κώδικα μέσα σε άλλα μπλοκ.

Στο παράδειγμα που ακολουθεί έχει οριστεί ένα μπλοκ κώδικα που ξεκινά στη 2η γραμμή, τελειώνει στην 5η γραμμή και περιέχει δύο εντολές. Οι εσοχές που προστέθηκαν στις εντολές του μπλοκ (3η και 4η γραμμή) δεν έχουν κάποια συντακτική σημασία και θα μπορούσαν να παραληφθούν. Συχνά όμως χρησιμοποιούνται εσοχές στα περιεχόμενα ενός μπλοκ ώστε να δίνουν έμφαση στα όρια του, κάνοντας τον κώδικα πιο ευανάγνωστο:

```
x1 <- 1
{
  x1 <- x1 + 1
  x2 <- 2
}
x3 <- x1 + x2
```

Καθώς ο κώδικας του παραδείγματος περιλαμβάνει πολλαπλές γραμμές, συνιστάται να τον δημιουργήσετε μέσα στον επεξεργαστή πηγαίου κώδικα (βλ §3.1.1 Δημιουργία και εκτέλεση R script). Μπορείτε πάντως να εισάγετε και απευθείας τον κώδικα του παραδείγματος στο Console, αλλά παρατηρήστε πως η R δεν θα εκτελέσει τις εντολές μέσα στο μπλοκ πριν ολοκληρωθεί η καταχώρησή του (με το }) καθώς τις θεωρεί μια αναπόσπαστη ομάδα.

Το συγκεκριμένο μπλοκ εντολών του παραπάνω παραδείγματος δεν παρέχει κάποια ιδιαίτερη λειτουργία στον κώδικα. Συγκεκριμένα, αν αφαιρεθεί ο ορισμός του μπλοκ (τα { και }) από το παραπάνω παράδειγμα, δεν θα αλλάξει απολύτως τίποτα κατά την εκτέλεση του.

Αυτό συμβαίνει για δύο λόγους:

(α) Τα μπλοκ κώδικα χρησιμοποιούνται κυρίως για ορίζουν τον κώδικα που θα εκτελεστεί σε δομές ελέγχου ροής, συναρτήσεις κλπ. όταν αυτός αποτελείται από περισσότερες της μίας εντολές (κάτι που συμβαίνει συχνά και περιγράφεται σε άλλες ενότητες, βλ §3.2.2 Έλεγχος ροής εκτέλεσης (control flow) και §5.1.1 Δημιουργία συναρτήσεων). Όμως το παραπάνω παράδειγμα δεν αφορά κάτι τέτοιο.

(β) Στην R οι μεταβλητές που δημιουργούνται μέσα σε ένα μπλοκ κώδικα είναι ορατές και εκτός του μπλοκ, στον κώδικα που θα εκτελεστεί μετά. Οι εσωτερικές μεταβλητές ενός μπλοκ δεν έχουν ισχύ μόνο εντός του μπλοκ[155].

---

[154] Πολλές άλλες γλώσσες προγραμματισμού (όπως C++, Java, Python και άλλες) χρησιμοποιούν την ίδια σύνταξη (με { και }) για τον ορισμό μπλοκ κώδικα.

[155] Σε κάποιες άλλες γλώσσες προγραμματισμού αυτό δεν ισχύει: οι μεταβλητές που ορίζονται εντός ενός μπλοκ κώδικα δεν είναι ορατές εκτός αυτού, ενώ μεταβλητές που έχουν οριστεί σε εξωτερικά μπλοκ είναι ορατές μέσα στο (εσωτερικό)

Το τελευταίο, είναι μια βασική διαφορά της R από πολλές άλλες γλώσσες προγραμματισμού. Στην R τα μπλοκ δεν επηρεάζουν την εμβέλεια (scope) των μεταβλητών και έτσι ένα μπλοκ κώδικα δεν λειτουργεί αυτομάτως και ως μηχανισμός που περιορίζει την πρόσβαση στις μεταβλητές που δημιουργήθηκαν εσωτερικά. Όλες οι μεταβλητές που ορίζονται εντός ή εκτός ενός μπλοκ δημιουργούνται στο ίδιο κοινό περιβάλλον (συνήθως το Global Environment, βλ.§2.2.2 Το καθολικό περιβάλλον (Global Environment)). Ο κώδικας εντός του μπλοκ έχει πρόσβαση στις μεταβλητές που ορίστηκαν από εντολές εκτός του μπλοκ αλλά και το αντίστροφο, δηλαδή τα αντικείμενα που δημιουργήθηκαν εντός του μπλοκ δεν είναι περιορισμένα μέσα σε αυτό, αλλά είναι ορατά και μπορούν να χρησιμοποιηθούν από τον κώδικα που ακολουθεί εκτός του μπλοκ. Αυτό απλοποιεί τον χειρισμό των μεταβλητών καθώς δεν τίθεται ζήτημα πιθανής συνωνυμίας δυο διαφορετικών αντικειμένων με το ίδιο όνομα (ενός ορισμένου εντός και ενός άλλου ορισμένου εκτός του block): οπουδήποτε χρησιμοποιείται ένα συγκεκριμένο όνομα μεταβλητής, το όνομα αυτό αφορά το ίδιο αντικείμενο. Ως αποτέλεσμα, αν εκτελεστεί το παραπάνω παράδειγμα, δημιουργεί στο Global Environment τρείς μεταβλητές με τελικές τιμές x1 = 2, x2 = 2 και x3 =3.

3.2.1.1 Επιστρεφόμενο αποτέλεσμα ενός μπλοκ κώδικα

Στην R τα μπλοκ κώδικα επιστρέφουν αποτέλεσμα. Συγκεκριμένα, ένα μπλοκ θα επιστρέψει το αντικείμενο που προκύπτει όταν εκτελεστεί η τελευταία εντολή εντός του μπλοκ:

Δοκιμάστε:	**Σχόλιο**
`x <- { 1 + 1; 1 + 2 }`	Η τελευταία εντολή στο μπλοκ είναι το 1+2. Όταν εκτελεστεί, το x θα γίνει 3.
`x <- { }`	Το μπλοκ είναι κενό εντολών. Όταν εκτελεστεί το x θα πάρει την τιμή NULL.

3.2.1.2 Χρήση μπλοκ κώδικα για ορισμό εμβέλειας μεταβλητών

Η ομαδοποίηση εντολών που περιγράφτηκε παραπάνω είναι ο βασικός (και συνήθης) λόγος που χρησιμοποιούνται τα μπλοκ κώδικα στην R. Όμως, αν πρέπει να προστεθεί σε αυτό και ο περιορισμός της εμβέλειας των μεταβλητών που δημιουργούνται μέσα στο μπλοκ κώδικα (ώστε κατά την εκτέλεσή του να μην μπορεί να επηρεάσει το εξωτερικό του περιβάλλον) μπορεί να χρησιμοποιηθεί η συνάρτηση local[156]. Η local δέχεται ως παράμετρο μια εντολή ή (προφανώς) ένα μπλοκ εντολών[157] και εκτελεί τον κώδικα αυτό σε ένα προσωρινό περιβάλλον που δημιουργεί για τον σκοπό αυτό[158]. Μετά την ολοκλήρωσή εκτέλεσης του κώδικα που της έχει δοθεί, η local διαγράφει το προσωρινό περιβάλλον (διαγράφοντας έτσι και όποιες μεταβλητές περιέχει) και επιστρέφει το αποτέλεσμα της τελευταίας εντολής που εκτέλεσε. Οι μεταβλητές που ορίστηκαν στο τοπικό περιβάλλον δεν είναι πλέον προσβάσιμες από τον υπόλοιπο κώδικα που ακολουθεί.

Με τη χρήση συναρτήσεων όπως η local οι μεταβλητές που δημιουργούνται εντός του μπλοκ έχουν μόνο τοπική εμβέλεια, άρα:

(α) Όταν γίνεται ανάθεση τιμής σε κάποια μεταβλητή, κάτι που στην R σημαίνει δημιουργία μιας μεταβλητής, αυτή δημιουργείται στο προσωρινό τοπικό περιβάλλον (και διαγράφεται μετά την εκτέλεση του κώδικα του μπλοκ).

(β) Ο κώδικας εντός του μπλοκ μπορεί να ανακαλέσει εξωτερικές μεταβλητές (μέσω του μηχανισμού αναζήτησης που περιγράφεται αλλού). Αλλά, λόγω του (α), αν γίνει ανάθεση τιμής στο όνομα μιας εξωτερικής μεταβλητής, αυτό οδηγεί στη δημιουργία μιας νέας τοπικής μεταβλητής με το ίδιο όνομα και η τιμή ανατίθεται σε αυτήν. Η εξωτερική δεν αλλάζει. Έτσι δεν υπάρχουν παράπλευρες παρενέργειες (side effects), δηλαδή αλλαγές τιμών σε αντικείμενα εξωτερικά του μπλοκ[159].

μπλοκ εκτός εάν καλύπτονται από μεταβλητές με το ίδιο όνομα που έχουν δηλωθεί εντός του εσωτερικού μπλοκ.

[156] Παρέχεται από το πακέτο base. Υπάρχουν και άλλοι τρόποι δημιουργίας τοπικών περιβαλλόντων, αλλά η local είναι ένας από τους απλούστερους.

[157] Ορθότερο είναι πως δέχεται αντικείμενα τύπου ‘call΄, ΄expression’, ‘promise’ ή ‘name’ (αν το name είναι ένα όνομα αντικειμένου το οποίο μπορεί να εντοπιστεί) ή αντικείμενα βασικών τύπων όπως διανύσματα, συναρτήσεις και περιβάλλοντα στα οποία όμως δεν γίνονται αλλαγές, βλ. help(eval).

[158] Μπορεί να οριστεί κάποιο άλλο περιβάλλον μέσω της παραμέτρου envir. Ως προεπιλογή για περιβάλλον της local είναι η δημιουργία νέου (envir = new.env()).

[159] Δεν υπάρχουν side-effects χωρίς πρόσθετο κώδικα. Υπάρχουν πάντα οι συναρτήσεις για πρόσβαση σε άλλα περιβάλλοντα, π.χ. η εντολή parent.frame(n=2) επιστρέφει εδώ το εξωτερικό περιβάλλον (2 «γενεές» πίσω), άρα μπορεί

Συνοψίζοντας τα παραπάνω, ένα μπλοκ κώδικα που εκτελείται μέσω της local έχει πρόσβαση σε μεταβλητές που υπάρχουν σε εξωτερικά επίπεδα χωρίς να μπορεί να τις αλλάξει, ενώ μεταβλητές που δημιουργούνται μέσα στο μπλοκ ισχύουν μόνο τοπικά. Ένα παράδειγμα ακολουθεί:

```
x1 <- 10
local({
  x2 <- 20
  x1 <- x1 + x2
  print(x1)
  print(x2)
})
print(x1)
```

Μετά την εκτέλεση του παραπάνω, η μόνη μεταβλητή που δημιουργήθηκε (και παραμένει) είναι η x1 με τιμή 10. Η ανάθεση τιμής σε κάποια συνώνυμή της x1 που έγινε εσωτερικά στο μπλοκ κώδικα αφορούσε τη νέα τοπική μεταβλητή x1. Καθώς πριν γίνει η ανάθεση έπρεπε να βρεθεί η ίδια η τιμή (το δεξί μέρος της έκφρασης, δηλαδή το x1 + x2) και δεν υπήρχε ακόμα κάποιο τοπικό x1, έγινε ανάκληση του εξωτερικού x1. Η έξοδος των print του κώδικα είναι:

```
[1] 30
[1] 20
[1] 10
```

Καθώς επιτρέπεται ο ορισμός μπλοκ κώδικα μέσα σε άλλα μπλοκ, το παρακάτω παράδειγμα αφορά την πρόσβαση εντός του μπλοκ σε μεταβλητές σε εξωτερικά επίπεδα:

```
xo <- 5              # xo στο εξωτερικό επίπεδο (=5)
x1 <- 10             # x1 στο εξωτερικό επίπεδο (=10)
local({
  x1 <- x1 + 20      # x1 στο 1ο εσωτερικό επίπεδο (=10+20)
  local({
    x1 <- x1 + 20    # x1 στο 2ο εσωτερικό επίπεδο (=30+20)
    print(x1 + xo)   # x1 από 2ο επίπεδο + xo από εξωτερικό
  })
  print(x1 + xo)     # x1 από 1ο επίπεδο + xo από εξωτερικό
})
print(x1 + xo)       # x1 + xo από εξωτερικό
```

Το αποτέλεσμα εκτέλεσης του παραδείγματος είναι να δημιουργηθούν (και να παραμείνουν) οι εξωτερικές μεταβλητές xo (με τιμή 1) και x1 (με τιμή 10), καθώς και να εμφανιστεί (ως αποτέλεσμα των print):

```
[1] 55
[1] 35
[1] 15
```

Για να ακολουθεί τις αρχές του functional programming (βλ.§5.3 Συναρτησιακός προγραμματισμός), η R εφαρμόζει στις συναρτήσεις όσα περιγράφονται παραπάνω σχετικά με τον περιορισμό της εμβέλειας των τοπικών τους μεταβλητών και το αποτέλεσμα που επιστέφουν. Αυτό γίνεται ώστε οι συναρτήσεις να μην επηρεάζουν (εύκολα) το εξωτερικό τους περιβάλλον[160].

3.2.2 Έλεγχος ροής εκτέλεσης (control flow)

Οι δεσμευμένες λέξεις που αφορούν τον έλεγχο της ροής εκτέλεσης του προγράμματος είναι οι if, else, for, while, repeat, break, και next. Αυτές λειτουργούν παρόμοια με τις αντίστοιχες εντολές άλλων γλωσσών προγραμματισμού που έχουν επιρροές από τη γλώσσα Algol (βλ. και help(Control)). Η δομή επιλογής υλοποιείται μέσω των δεσμευμένων λέξεων if και else, ενώ οι for, while, repeat, break και next αφορούν τη δημιουργία και τον χειρισμό κώδικα που θα επαναλαμβάνεται, δηλαδή «βρόχων» (loop).

να γίνει και ανάθεση τιμής σε (εξωτερικές) μεταβλητές του, μέσω συναρτήσεων όπως η assign. Αυτή και άλλες παρόμοιες τεχνικές περιγράφονται αλλού (βλ.§4.2.2.1 Περιβάλλοντα και συναρτήσεις).

[160] βλ. και §5.1.5 Αλληλεπίδραση με το περιβάλλον.

3.2.2.1 Η βασική δομή επιλογής (if και else)

Η βασική δομή επιλογής υλοποιείται μέσω των δεσμευμένων λέξεων **if** και **else**. Η σύνταξη πρέπει να έχει μία από τις παρακάτω μορφές [47]:

if (*α*) *β* else *γ*

if (*α*) *β*

όπου *α* είναι ένα αντικείμενο τύπου logical (βλ.§2.5 Λογικές πράξεις, συγκρίσεις και ο βασικός τύπος logical), *β* η έκφραση που θα αποτιμηθεί αν το *α* είναι TRUE, ενώ *γ* η έκφραση που θα αποτιμηθεί αν το *α* είναι FALSE.

Δοκιμάστε:	**Σχόλιο**
`if(1==1) 10 else 20`	Αν 1=1 τότε 10 αλλιώς 20 (επιστρέφει 10)
`if(1>=2) 10 else 20`	Αν 1≥2 τότε 10 αλλιώς 20 (επιστρέφει 20)
`if(1!=2) 10 else 20`	Αν 1≠2 τότε 10 αλλιώς 20 (επιστρέφει 10)
`if(1!=2&&2!=3) 10 else 20`	Αν 1≠2 και 2≠3 τότε 10 αλλιώς 20 (επιστρέφει 10)
`if(1!=1||2!=3) 10 else 20`	Αν 1≠1 ή 2≠3 τότε 10 αλλιώς 20 (επιστρέφει 10)
`if(1==1) 10`	Αν 1=1 τότε 10 (επιστρέφει 10)
`if(1>=2) 10`	Αν 1≥2 τότε 10 (επιστρέφει NULL)

Χρησιμοποιήσαμε παραπάνω τη λέξη «αποτιμηθεί» και όχι εκτελεστεί, καθώς η if επιστρέφει το αποτέλεσμα της έκφρασης που επιλέχτηκε[161]. Αν δεν υπάρχει κάτι να αποτιμηθεί, επιστρέφει την ειδική τιμή που αντιστοιχεί στο «τίποτα», δηλαδή NULL:

Δοκιμάστε:	**Σχόλιο**
`x <- if(1>=2) 10 else 20`	Το x θα πάρει την τιμή 10.
`x <- if(1>=2) 10`	Το x θα πάρει την ειδική τιμή NULL.

Το κριτήριο (το logical αντικείμενο *α*) πρέπει να αντιστοιχεί σε μια τιμή TRUE ή FALSE ώστε να εκτελεστεί η μια από τις δύο επιλογές (αν το *α* είναι TRUE η *β,* αλλιώς η *γ*). Αν το αντικείμενο *α* που δίνεται είναι πολυμελές (π.χ. ένα διάνυσμα λογικών τιμών με length > 1), η if θα χρησιμοποιήσει μόνο το πρώτο στοιχείο του για να κάνει την επιλογή, αγνοώντας τα υπόλοιπα. Έτσι το παρακάτω θα επιστρέψει 20:

```
if(c(F,T,T,T)) 10 else 20
```

Επειδή αυτό ίσως κρύβει ένα πιθανό λάθος του προγραμματιστή, η R θα εγείρει σχετικό μήνυμα προειδοποίησης (warning)[162]. Περισσότερα σχετικά με τη δημιουργία κατάλληλων κριτηρίων (που οδηγούν σε μια μοναδική λογική τιμή) υπάρχουν στις παραγράφους §2.5.1 Συναρτήσεις σύγκρισης και §2.5.2 Λογικές πράξεις.

Για να αποτιμηθεί μια έκφραση πρέπει να εκτελεστεί ο κώδικας της και αυτή είναι και η βασική εφαρμογή της if, δηλαδή να επιλέγει βάσει του κριτηρίου *α* το αν θα εκτελεστεί το *β* ή (αν υπάρχει) το *γ*. Το παρακάτω παράδειγμα, επιλέγει να εκτελέσει την εντολή που εμφανίζει το κατάλληλο μήνυμα βάσει της τρέχουσας τιμής του x:

```
if(x>100)
  cat("η τιμή είναι μεγαλύτερη του 100.\n") else
  cat("η τιμή είναι 100 ή μικρότερη.\n")
```

Σε περίπτωση που ο κώδικας της if καταλαμβάνει πολλαπλές γραμμές καλό είναι αν υπάρχει else να γράφεται αμέσως μετά και στην ίδια γραμμή με το τέλος της έκφρασης *β* (του μέρους δηλαδή που θα εκτελεστεί αν το κριτήριο είναι TRUE) ώστε να αναγνωρίζεται κατά τη λεκτική ανάλυση η ύπαρξη else στη δομή.

Όπως έχει ήδη αναφερθεί, σε οποιοδήποτε σημείο του κώδικα γίνεται δεκτή μια εντολή (όπως η συνάρτηση cat στο παράδειγμα), μπορεί να χρησιμοποιηθεί ένα μπλοκ κώδικα που περιέχει περισσότερες εντολές. Στην παρακάτω παραλλαγή του προηγουμένου παραδείγματος ελέγχεται το x και εκτός από το κατάλληλο μήνυμα αλλάζει και η τιμή του (αν το x βρεθεί > 100 αλλάζει σε 100, αλλιώς σε 0). Επειδή και στις δύο περιπτώσεις πρέπει να εκτελεστούν περισσότερες από μια εντολές, είναι ομαδοποιημένες μέσα σε μπλοκ:

```
if(x>100)
```

[161] Όπως σχεδόν τα πάντα στην R, και η if επιστρέφει κάποια τιμή (ένα αντικείμενο). Το ότι η if επιστρέφει τιμή ίσως σας θυμίζει τον τρόπο που επιστρέφουν τιμές τα προγράμματα υπολογιστικών φύλλων με τη δική τους συνάρτηση IF.

[162] βλ.§5.1.4 Έγερση και χειρισμός σφαλμάτων.

```
{
  cat("Μεγαλύτερο του 100, αλλάζει σε 100.\n")
  x <- 100
} else
{
  cat("Μικρότερο ή ισο με 100, αλλάζει σε 0.\n")
  x <- 0
}
```

Επιπροσθέτως, χρειάζεται συχνά η δομή if να ενσωματωθεί μέσα σε άλλες δομές if (όπως και σε άλλες δομές ή συναρτήσεις γενικότερα). Όταν το πρόβλημα απαιτεί χρήση if μέσα σε άλλες if τότε αυτές αναφέρονται ως «εμφωλευμένες if» (nested if). Ένα παράδειγμα:

```
if(x>100)
  if (y > 100)
    z <- 4 else
    z <- 3 else
  if (y > 50)
    z <- 2 else
    z <- 1
print(z)
```

Στο παραπάνω, αν το x>100 τυχαίνει να είναι TRUE η εκτέλεση θα προχωρήσει το *α* μέρος της δομής (δηλαδή στο if (y > 100) z <- 4 else z <- 3) οπότε ελέγχεται αν το y>100. Αν τότε προκύψει πως το y>100 είναι TRUE, το z παίρνει την τιμή 4, αλλιώς την τιμή 3, η αρχική if ολοκληρώνεται και η εκτέλεση προχωρά στην print(z) που ακολουθεί. Αν όμως ο αρχικός έλεγχος επιστρέψει FALSE (άρα το x βρέθηκε μικρότερο ή ίσο του 100) η εκτέλεση παρακάμπτει το σχετικό *α* μέρος και προχωρά στο *β* μέρος (δηλαδή στο if (y > 50) z <- 2 else z <- 1) οπότε ελέγχεται αν το y>50 κλπ. Πάντως η ανάγκη για χρήση εμφωλευμένων if μπορεί συχνά να εξαλειφθεί με κατάλληλο συνδυασμό των κριτήριων μέσω λογικών πράξεων (βλ.§2.5.2 Λογικές πράξεις).

3.2.2.2 Άλλες μέθοδοι επιλογής (switch, ifelse)

Δύο συναρτήσεις του πακέτου base παρέχουν παρεμφερείς λειτουργίες με τη δεσμευμένη λέξη if. Αυτές είναι οι συναρτήσεις ifelse και switch.

Η συνάρτηση **switch** επιλέγει ποιος κώδικας θα αποτιμηθεί, με κριτήριο τη σύγκριση της τιμής κάποιας έκφρασης με συγκεκριμένες, προεπιλεγμένες, εναλλακτικές τιμές. Μπορεί να λειτουργήσει με βάση είτε αριθμητικές τιμές είτε κείμενο. Στην απλούστερη περίπτωση, η μορφή της είναι:

switch (*α*, *ε1, ε2, ε3,…)*

Το *α* πρέπει να είναι ένα αντικείμενο με μόνο ένα στοιχείο. Αν το *α* είναι ένας ακέραιος (ή άλλος τύπος που μετατρέπεται σε ακέραιο) η τιμή του θα χρησιμοποιηθεί για να αποτιμηθεί μία από τις επιλογές *ε1*, *ε2, ε3* κλπ. και να επιστρέψει το αποτέλεσμα. Έτσι, αν το *α* έχει την τιμή 1 θα αποτιμηθεί και θα επιστραφεί η πρώτη έκφραση *ε1*, ενώ αν το *α* είναι 3 θα επιστραφεί η 3η. Όταν δεν μπορεί να γίνει αντιστοίχιση του *α* με κάποια επιλογή, η switch θα επιστρέψει NULL:

Δοκιμάστε:	**Σχόλιο**
`x <- 2`	Μεταβλητή x με τιμή 2.
`switch(x, 10, 20, 30)`	Αφού το x είναι 2, επιστρέφει το 2ο από τις επιλογές, δηλαδή 20.
`switch(4, 10, 20, 30)`	4η επιλογή δεν υπάρχει, επιστρέφει την ειδική τιμή NULL.

Εφόσον συνδυαστεί με εντολές, η switch μπορεί να χρησιμοποιηθεί σαν μηχανισμός επιλογής της ροής της εκτέλεσης, για παράδειγμα:

```
switch(x,
       {
         y <- 10
         z <- 20
         print(y + z)
       },
       {
```

```
        y <- 100
        z <- 200
        print(y + z)
      })
```

Στην παραπάνω switch έχουν οριστεί δύο εναλλακτικές εκφράσεις που τυχαίνει να είναι μπλοκ κώδικα. Η switch θα επιλέξει κάποιο από τα δύο βάσει της τρέχουσας τιμής του x. Άρα, αν το x είναι 1, θα εκτελεστεί το 1ο μπλοκ (με αποτέλεσμα τη συγκεκριμένη ανάθεση τιμών 10 και 20 στις μεταβλητές y, z αντίστοιχα και τελικά την εμφάνιση και επιστροφή της τιμής 30). Αν το x είναι 2 θα εκτελεστεί το 2ο μπλοκ, με αντίστοιχο τελικό αποτέλεσμα 300. Αν τέλος το x δεν αντιστοιχεί σε κάποια από τις επιλογές (π.χ. είναι μηδέν ή 3) θα επιστραφεί NULL.

Εκτός από αριθμούς (τους οποίους μετατρέπει σε ακέραιους), η switch μπορεί να χρησιμοποιήσει κείμενο για την επιλογή, αξιοποιώντας την παρακάτω μορφή της εντολής:

switch (*α*, *κ1=ε1, κ2=ε2, κ3=ε3,...)*

όπου *κ1*, *κ2* κλπ. είναι κείμενα. Αν το *a* ταιριάζει με το *k1* θα εκτελεστεί η έκφραση *ε1*, ενώ αν ταιριάζει με το *k2* θα εκτελεστεί η έκφραση *ε2* και ούτω καθεξής. Για παράδειγμα, το παρακάτω θα επιστρέψει 30:

```
switch("c", a = 10, b = 20, c = 30)
```

Όταν η επιλογή γίνεται βάσει κειμένων, μπορεί να οριστεί η ίδια έκφραση ως επιλογή για περισσότερα του ενός. Μπορεί επίσης να οριστεί μια έκφραση που θα επιλεχθεί αν δεν υπάρξει ταίριασμα. Ο μηχανισμός αυτός φαίνεται στο επόμενο παράδειγμα:

```
d="Σάββατο"
switch(d,
  "Δευτέρα" = ,
  "Τρίτη" = ,
  "Τετάρτη" = ,
  "Πέμπτη" = ,
  "Παρασκευή" = 1,
  "Σάββατο" = ,
  "Κυριακή" = 2,
  100
)
```

Η παραπάνω switch θα επιστρέψει 1 αν το d είναι ίσο με το όνομα μίας ημέρας της εβδομάδας «Δευτέρα» έως «Παρασκευή», 2 αν το d είναι το κείμενο «Σάββατο» ή «Κυριακή», ενώ θα επιστρέψει 100 αν το κείμενο στο d δεν είναι κανένα από τα παραπάνω. Προφανώς και εδώ, αντί των 1, 2 και 100 θα μπορούσε να έχει οριστεί οποιαδήποτε άλλη έκφραση, εντολή ή μπλοκ εντολών.

Η συνάρτηση **ifelse** είναι ένας μηχανισμός επιλογής κατάλληλος για επεξεργασία πολυμελών αντικειμένων (αντικειμένων με length>1) τα οποία αποτελούνται από logical τιμές[163]. Βάσει της logical τιμής κάθε στοιχείου του αντικειμένου, η ifelse επιλέγει μία από τις δύο επιλογές που έχουν οριστεί και την τοποθετεί στην αντίστοιχη θέση σε ένα αντικείμενο ίδιας μορφής. Οπότε αν π.χ. της δοθεί ένα διάνυσμα logical τιμών θα επιστρέψει επίσης διάνυσμα ίδιου μήκους που θα περιέχει τα αποτελέσματα[164]. Η μορφή της είναι:

ifelse (*α*, *ε1, ε2)*

όπου *α είναι* ένα αντικείμενο με logical τιμές, *ε1* και *ε2* είναι οι εκφράσεις που θα χρησιμοποιηθούν αν το στοιχείο του *α* αποτιμηθεί σε τιμή TRUE και FALSE αντίστοιχα. Συνοψίζοντας, η ifelse ελέγχει κάθε στοιχείο του *a* και αν είναι TRUE αποτιμά το *ε1* ενώ αν είναι FALSE αποτιμά το *ε2*. Σε κάθε περίπτωση τοποθετεί το αποτέλεσμα ως στοιχείο στην αντίστοιχη θέση ενός αντικειμένου το οποίο τελικά θα επιστρέψει μετά την επεξεργασία όλων των στοιχείων του *α* και θα περιέχει όλα τα επιμέρους αποτελέσματα:

Δοκιμάστε:	**Σχόλιο**
`x<-c(1,-1,-2,2)`	Ένα διάνυσμα. Με ifelse, θα ελεγχθεί αν τα στοιχεία του είναι >0.
`ifelse(x>0,100,0)`	Επιστρέφει το διάνυσμα c(100, 0, 0, 100)
`ifelse(x>0,"Θετικό","Μη-Θετικό")`	Επιστρέφει c("Θετικό","Μη-Θετικό","Μη-Θετικό","Θετικό")
`ifelse(x>0,100,x)`	Επιστρέφει το διάνυσμα c(100, -1, -2, 100)

[163] Ένα αντικείμενο σε logical mode, ή κάποιο που μπορεί να μετατραπεί σε τέτοιο.

[164] Αντίστοιχα, αν δοθεί πίνακας θα επιστρέψει πίνακα ίδιων διαστάσεων κλπ.

Στο τελευταίο βήμα του παραπάνω παραδείγματος παρατηρήστε πως x πρακτικά σημαίνει «το τρέχον στοιχείο του x», κάτι που μπορεί να αξιοποιηθεί περεταίρω:

Δοκιμάστε:	**Σχόλιο**
`a<-c(1,2,5,2,1,5)`	Ένα διάνυσμα ακεραίων αριθμών.
`b<-c(5,2,5,1,1,6)`	Ένα άλλο διάνυσμα ακεραίων αριθμών.
`ifelse(a==b,10,b)`	Επιστρέφει το διάνυσμα c(5, 10, 10, 1, 10, 6).
`ifelse(a>4,10*a,b)`	Επιστρέφει το διάνυσμα c(5, 2 50 1 1 50).

Προφανώς οι εκφράσεις *ε1* και *ε2* μπορούν να είναι μπλοκ κώδικα, όμως αυτό απαιτεί προσοχή. Για παράδειγμα:

```
x<-c(1,-1,-2,2)

ifelse(x>0,
{
  y<-1
  z<-2
  x+y+z
},
{
  y<-10
  z<-20
  y+z
})
```

Ο κώδικας αυτός θα επιστρέψει το διάνυσμα c(4, 30, 30, 5). Αν και συντακτικώς επιτρέπεται, δεν συνιστάται η χρήση πολύπλοκών εκφράσεων ως *ε1* και *ε2* μέσα στην ifelse. Η σειρά με την οποία θα εκτελεστούν οι εκφράσεις αυτές δεν είναι προφανής ούτε εξασφαλισμένη[165]. Για παράδειγμα, είναι δύσκολο να προβλεφθεί με απλή ανάγνωση του κώδικα η επίδραση που θα έχει η εκτέλεση των *ε1* και *ε2* στις τελικές τιμές (μετά την ολοκλήρωση της ifelse) των μεταβλητών οι οποίες ενδεχομένως να αλλάζουν μέσα στις εκφράσεις αυτές (όπως γίνεται παραπάνω για τις μεταβλητές y και z). Άρα καλό θα είναι ο κώδικας των *ε1* και *ε2* να σχετίζεται μόνο με τη δημιουργία του αποτελέσματος που θα αποθηκευτεί στο αντικείμενο που θα επιστραφεί. Εξάλλου, ο ρόλος της ifelse δεν είναι να δρα ως κλασική δομή επιλογής, αλλά άλλος και σημαντικός: η δημιουργία πολλαπλών αποτελεσμάτων από συγκρίσεις πάνω σε πολυμελείς δομές αντικειμένων χωρίς την ανάγκη χρήσης βρόχων επανάληψης (loops) (βλ. επόμενη ενότητα). Άλλες παρεμφερείς συναρτήσεις (όπως η **case_when**), που επιτρέπουν τον προσδιορισμό της επεξεργασίας σε επίπεδο (πολυμελούς) αντικειμένου και όχι σε επίπεδο στοιχείου, αναφέρονται στη §5.4 Η συναρτησιακή προσέγγιση στον πραγματικό κόσμο.

Τέλος, μια ακόμα μορφή ελέγχου της ροής του προγράμματος αποτελούν οι συναρτήσεις για τον χειρισμό εξαιρέσεων και αντικανονικών συνθηκών κατά την εκτέλεση μίας εντολής, θέμα το οποίο περιγράφεται σε άλλη ενότητα (βλ.§5.1.4 Έγερση και χειρισμός σφαλμάτων).

3.2.2.3 Επαναλήψεις (βρόχοι ή loop)

Στον προγραμματισμό οι εντολές που αλλάζουν τη ροή εκτέλεσης προκαλώντας επανάληψη κάποιου μέρους του κώδικα ονομάζονται δομές βρόχου (δηλαδή δομές θηλιάς) ή loops. Αυτό γιατί, κατά κάποιον τρόπο, επιστρέφουν πίσω στο ίδιο αρχικό σημείο, δένονται και περιστρέφονται γύρω από τον εαυτό τους. Ο βρόχος είναι ένας τρόπος για να επαναλάβετε μια ακολουθία εντολών, ενώ η επανάληψη αυτή θα συνεχίζεται όσο πληρούνται κάποιες συνθήκες. Οι τυπικές δομές επανάληψής που υπάρχουν σε άλλες συνήθεις γλώσσες προγραμματισμού υπάρχουν και στην R και παρουσιάζονται σε αυτή την ενότητα.

Ο προγραμματισμός βρόχων είναι απαραίτητος σε διάφορες περιπτώσεις προβλημάτων. Ένας βασικός όμως λόγος για τον οποίο χρησιμοποιούνται βρόχοι είναι κατά την επεξεργασία πολυμελών αντικειμένων (π.χ. διανυσμάτων ή πινάκων), όπου ζητούμενο είναι ο ίδιος κώδικας να επαναληφθεί για κάθε στοιχείο του αντικειμένου. Αυτή η προσέγγισή μπορεί να ληφθεί και στην R, αλλά συνήθως αποφεύγεται. Στην R οι συναρτήσεις συνήθως μπορούν να επεξεργαστούν με μια εντολή όλα τα στοιχεία πολυμελών δομών

[165] Ούτε αναφέρεται κάτι σχετικό στην τεκμηρίωση, βλ. help(ifelse).

αντικειμένων. Αυτό κάνει τον κώδικα αρκετά πιο αποδοτικό, σύντομο, ευανάγνωστο, γρήγορο και αναιρεί την ανάγκη ορισμού βρόχων από τον προγραμματιστή για τον σκοπό αυτόν[166]. Οι λύσεις που κάνουν χρήση βρόχων είναι λιγότερο αποδοτικές και συνήθως πιο αργές κατά την εκτέλεση τους από τις διανυσματικές λύσεις που εκμεταλλεύονται το λεξιλόγιο της R[167] για επεξεργασία πολυμελών αντικειμένων. Μια εισαγωγή σε αυτή την προσέγγιση γίνεται στην §2.4 Εισαγωγή στα διανύσματα (vector). Χρήση εντολών που ορίζουν επεξεργασία σε επίπεδο (πολυμελούς) αντικειμένου γίνεται σε διάφορα σημεία του βιβλίου αυτού. Η προσέγγιση αυτή συνοψίζεται στην §5.4 Η συναρτησιακή προσέγγιση στον πραγματικό κόσμο.

Παρόλα αυτά η R παρέχει τη δυνατότητα δημιουργίας και χειρισμού δομών βρόχων (loop) με τις εντολές for, while, repeat, next και break. Οι εντολές αυτές δεν είναι συναρτήσεις και δεν παρέχονται από κάποιο πακέτο. Είναι δεσμευμένες λέξεις από την ίδια τη γλώσσα R[168], παρέχονται από αυτή, αφορούν τον έλεγχο ροής εκτέλεσης του κώδικα μέσω επαναλήψεων και δεν επιστρέφουν κάποια τιμή[169].

Η εντολή **for** δέχεται ένα διάνυσμα και επαναλαμβάνει κάποιον δοθέντα κώδικα δοκιμάζοντας σε κάθε επανάληψή του ένα διαφορετικό στοιχείο του διανύσματος (από το πρώτο έως και το τελευταίο). Η δομή της είναι:

for (*μ* in *δ*) *ε*

όπου *μ* μία μεταβλητή, *δ* ένα διάνυσμα, *ε* η έκφραση (κώδικας) που θα επαναληφθεί. Αυτό διαβάζεται και ως εξής: για κάθε στοιχείο στο *δ*, τοποθέτησε το στοιχείο αυτό σε μια μεταβλητή *μ* και εκτέλεσε μετά το *ε*. Προφανώς ο κώδικας που θα επαναλαμβάνεται (το *ε*) συνήθως εκμεταλλεύεται το γεγονός πως πριν από κάθε επόμενη επανάληψη, στο *μ* θα αντιγράφεται το επόμενο στοιχείο του διανύσματος *δ*. Η τιμή στο *μ* μπορεί ακόμα και να αλλαχτεί εσωτερικά από τον κώδικα του *ε,* αλλά με την έναρξη της επόμενης επανάληψης το *μ* θα έχει την τιμή του επόμενου στοιχείου του *δ*.

Οι επαναλήψεις θα σταματήσουν όταν δεν υπάρχουν άλλα στοιχεία στο διάνυσμα (ή κληθεί η εντολή break βλ. παρακάτω). Μετά την ολοκλήρωση των επαναλήψεων η for δεν επιστρέφει κάποια τιμή, όμως η μεταβλητή *μ* (με όποιο όνομα της έχει δοθεί) συνεχίζει να υπάρχει και έχει την τελευταία τιμή που της ανατέθηκε κατά την εκτέλεση της for.

Ακολουθούν μερικά παραδείγματα της εντολής for. Ο παρακάτω κώδικας ζητά για κάθε i στοιχείο στο διάνυσμα c(1,5,10) να εμφανιστεί (με τη συνάρτηση print) το τρέχον i:

```
for(i in c(1,5,10)) print(i)
```

Αποτέλεσμα εκτέλεσης του κώδικα είναι:

```
[1] 1
[1] 5
[1] 10
```

Ο παρακάτω κώδικας ζητά για κάθε i στοιχείο στην ακολουθία 1:5 (που είναι διάνυσμα) να εμφανιστεί μήνυμα με το αποτέλεσμα κάποιας αριθμητικής πράξης με το τρέχον i:

```
for (i in 1:5)
  cat("To", i, "επί 1.4 κάνει", i * 1.4, "\n")
```

Αποτέλεσμα εκτέλεσης του κώδικα είναι:

```
To 1 επί 1.4 κάνει 1.4
To 2 επί 1.4 κάνει 2.8
To 3 επί 1.4 κάνει 4.2
To 4 επί 1.4 κάνει 5.6
To 5 επί 1.4 κάνει 7
```

[166] Οι συναρτήσεις που παρέχουν τα ενσωματωμένα πακέτα της R όταν επεξεργάζονται ολόκληρα πολυμελή αντικείμενα, είτε εκμεταλλεύονται δυνατότητες επεξεργασίας μέσω διανυσματοποίησης (vectorization), είτε εκτελούν εσωτερικά τις απαραίτητες επαναλήψεις, είτε γίνεται συνδυασμός των δύο. Σε κάθε περίπτωση, οι συναρτήσεις αυτές είναι συνήθως ήδη μεταγλωττισμένος (compiled) κώδικας (βλ. π.χ. §7.4 C++) και επιπρόσθετα υλοποιούν διάφορες βελτιστοποιήσεις απόδοσης. Έτσι η επεξεργασία συνήθως γίνεται αρκετά ταχύτερα από όποια αντίστοιχη προσέγγιση βασίζεται σε βρόχους της R.

[167] Πολλά τέτοια παραδείγματα υπάρχουν στις προηγούμενες ενότητες όπου εκτελούνται μαθηματικές πράξεις και συναρτήσεις σε διανύσματα. Επίσης, χρήσιμες συναρτήσεις που υποκαθιστούν βρόχους είναι οι συναρτήσεις ifelse (βλ. §3.2.2.2 Άλλες μέθοδοι επιλογής (switch, ifelse)), η συνάρτηση outer (βλ.§4.1.3.3 Η συνάρτηση outer) και οι συναρτήσεις apply (βλ.§4.1.3.4 Η οικογένεια συναρτήσεων apply).

[168] βλ. help(Reserved).

[169] Επιστρέφουν NULL, για περισσότερα, βλ. help(Control).

Ο παρακάτω κώδικας ζητά για κάθε i στοιχείο στην ακολουθία 1:5 να εμφανιστεί η ακολουθία αριθμών από το 1 έως το τρέχον i:

```
for(i in 1:5) print(1:i)
```

Αποτέλεσμα εκτέλεσης του κώδικα είναι:

```
[1] 1
[1] 1 2
[1] 1 2 3
[1] 1 2 3 4
[1] 1 2 3 4 5
```

Ο παρακάτω κώδικας ζητά κάθε i στοιχείο σε διάνυσμα d (το οποίο εδώ είναι το c(15, 20, 10, 5)) να αθροιστεί σε μια μεταβλητή t (στην οποία αρχικά δίνεται η τιμή 0):

```
d <- c(15,20,10,5)
t <- 0
for (i in d) t <- t + i
```

Ως αποτέλεσμα του παραπάνω, η t θα έχει το άθροισμα όλων των αριθμών στο d, δηλαδή το 50. Κάτι που επαναφέρει όσα αναφέρθηκαν στην αρχή αυτής της ενότητας σχετικά με την επεξεργασία πολυμελών αντικειμένων στην R: το ίδιο αποτέλεσμα θα μπορούσε να επιτευχθεί με την εντολή y <- sum(d).

Συχνά οι βρόχοι for χρησιμοποιούνται για να δημιουργήσουν δείκτες με τους οποίους γίνεται πρόσβαση στα στοιχεία διανυσμάτων ή άλλων πολυμελών αντικειμένων. Στο επόμενο παράδειγμα χρησιμοποιούνται for και if για να πολλαπλασιάζονται επί 100 όσα στοιχεία του διανύσματος d είναι μεγαλύτερα του 10. Σε κάθε επανάληψη του βρόχου for το i θα πάρει μια νέα τιμή ξεκινώντας από το 1 έως το μήκος του διανύσματος (εδώ 4). Με το i αυτό θα ελεγχθεί αν το στοιχείο i του d (δηλαδή το d[i]) είναι μεγαλύτερο του 10 και αν είναι θα πολλαπλασιάζεται το στοιχείο αυτό επί 100.

```
d <- c(15, 20, 10, 5)
for (i in 1:length(d))
  if (d[i] > 10)
    d[i] <- d[i] * 100
```

Μετά την εκτέλεση, το d θα περιέχει την τιμή c(1500, 2000, 10, 5). Και εδώ, το ίδιο αποτέλεσμα θα μπορούσε να επιτευχθεί χωρίς βρόχο for και if, απλώς με την εντολή d[d>10] <- d[d>10] * 100.

Προφανώς (και συχνότατα) αυτό που ζητείται να επαναληφθεί είναι ένα μπλοκ κώδικα (βλ.§3.2.1 Μπλοκ κώδικα). Στην παρακάτω παραλλαγή του προηγούμενου παραδείγματος, επαναλαμβάνεται ένα μπλοκ κώδικα με δύο εντολές: η 1η συσσωρεύει όπως και πριν, ένα-ένα τα στοιχεία του d στο t, ενώ η δεύτερη εμφανίζει το τρέχον αποτέλεσμα:

```
d <- c(15,20,10,5)
t <- 0
for (i in d)
{
  t <- t + i
  print(t)
}
```

Αποτέλεσμα εκτέλεσης του κώδικα είναι πως η t θα έχει το άθροισμα όλων των αριθμών στο d (δηλαδή το 50) αλλά θα εμφανιστούν και οι ενδιάμεσες τιμές που πήρε:

```
[1] 15
[1] 35
[1] 45
[1] 50
```

Η εντολή **while** επαναλαμβάνει κάποιον δοθέντα κώδικα όσο ισχύει κάποιο κριτήριο. Η δομή της είναι:

while (*α*) *ε*

όπου *α* ένα κριτήριο, *ε* η έκφραση (κώδικας) που θα επαναληφθεί. Αυτό διαβάζεται και ως εξής: όσο ισχύει το *α* να εκτελείται το *ε*. Προφανώς ο κώδικας που θα επαναλαμβάνεται (το *ε*) πρέπει να επηρεάζει την τιμή του *α* αλλιώς οι επαναλήψεις (εφόσον ξεκινήσουν) θα συνεχίζονται για πάντα. Αυτό είναι μια κατάσταση που ονομάζεται «ατέρμων βρόχος» (endless loop) και ποτέ δεν γίνεται σκόπιμα. Το κριτήριο *α* είναι μια έκφραση που όταν αποτιμηθεί επιστρέφει ένα αντικείμενο τύπου logical, άρα για να ισχύει πρέπει να έχει την τιμή TRUE. Όσα αναφέρθηκαν για τα κριτήρια της εντολής if ισχύουν και εδώ (βλ.§3.2.2.1 Η βασική δομή επιλογής (if και else)). Όπως και στην if, το κριτήριο πρέπει να αποτιμάται σε μία μόνο logical τιμή (αν

αποτελείται από πολλές logical τιμές θα χρησιμοποιηθεί μόνο η πρώτη). Τέλος, αν το κριτήριο είναι εξαρχής FALSE δεν θα εκτελεστεί καμία επανάληψη.

Ακολουθεί ένα απλό παράδειγμα της εντολής while. Ο κώδικας ξεκινά δίνοντας σε κάποια μεταβλητή x την τιμή 10. Στη συνέχεια, όσο το τρέχον x είναι μικρότερο του 100 το εμφανίζει και το αυξάνει κατά 15:

```
x <- 10
while (x < 100) {
  print(x)
  x <-  x + 15
}
```

Αποτέλεσμα εκτέλεσης του κώδικα είναι:

```
[1] 10
[1] 25
[1] 40
[1] 55
[1] 70
[1] 85
```

Οι εντολές που επαναλαμβάνει η εντολή while (η έκφραση *ε* που αναφέρθηκε παραπάνω) είναι συνήθως μπλοκ κώδικα. Αυτό συμβαίνει γιατί, πέραν της όποιας χρήσιμης λειτουργίας επιτελούν οι εντολές αυτές, πρέπει κάποια στιγμή να αλλάζουν και το κριτήριο ώστε να τερματίζονται οι επαναλήψεις. Έτσι, συνήθως ο κώδικας αυτός αποτελείται από πολλαπλές εντολές που πρέπει να εκτελεστούν ως μια ομάδα, άρα τοποθετούνται μέσα σε μπλοκ κώδικα.

Η εντολή **repeat** επαναλαμβάνει έναν κώδικα για πάντα. Επειδή αυτό ποτέ δεν είναι θεμιτό, συνδυάζεται με την εντολή **break** η οποία σταματά («σπάει») τις επαναλήψεις ενός οποιουδήποτε τύπου βρόχου (από αυτούς που περιγράφονται σε αυτή την ενότητα δηλαδή for, while και repeat). Εκεί ακριβώς που καλείται η break θα σταματήσει η εκτέλεση του βρόχου και θα συνεχιστεί εκτός του βρόχου. Η δομή της repeat είναι:

repeat *ε*

όπου *ε* η έκφραση (κώδικας) που θα επαναληφθεί. Σε βρόχους μέσω της εντολής repeat είναι απαραίτητο η έκφραση *ε* που επαναλαμβάνεται να είναι μπλοκ κώδικα, αφού θα πρέπει να υπάρχει μεταξύ των εντολών αυτή που θα «σπάσει» τις επαναλήψεις (συνήθως καλώντας την break). Η απουσία ή η λανθασμένη χρήση του break, εύκολα οδηγεί (και εδώ) σε «ατέρμονες βρόχους». Το παράδειγμα που ακολουθεί είναι λειτουργικά ταυτόσημο με το παράδειγμα της εντολής while. Ξεκινά δίνοντας σε κάποια μεταβλητή x την τιμή 10 και ξεκινά να επαναλαμβάνει το μπλοκ κώδικα. Σε κάθε επανάληψη ελέγχει αν το τρέχον x είναι μεγαλύτερο ή ίσο του 100 και εφόσον αυτό ισχύει βγαίνει από το μπλοκ και σταματά τις επαναλήψεις, ενώ αν συνεχίσει εντός του μπλοκ εμφανίζει το x και το αυξάνει κατά 15:

```
x <- 10
repeat {
  if (x >= 100) break
  print(x)
  x <-  x + 15
}
```

Η εντολή **next** διακόπτει την τρέχουσα επανάληψη και συνεχίζει με την επόμενη. Λειτουργεί σε βρόχους οποιουδήποτε τύπου από αυτούς που περιγράφονται σε αυτή την ενότητα (δηλαδή for, while και repeat). Στο σημείο που καλείται η next θα σταματήσει η τρέχουσα εκτέλεση του βρόχου και θα ξεκινήσει η επόμενη επανάληψη (αν υπάρχει). Το επόμενο παράδειγμα βασίζεται στο προηγούμενο αλλά έχει προστεθεί κώδικας ώστε αν η τρέχουσα τιμή του x είναι 55 να σταματήσει η ροή εκτέλεσης και να συνεχίσει στην επόμενη επανάληψη:

```
x <- 10
repeat {
  if (x == 55)
  {
    x <-  x + 15
    next
  }
  if (x >= 100) break
  print(x)
```

```
    x <-  x + 15
  }
```

Το αποτέλεσμα της εκτέλεσης του κώδικα είναι παρόμοιο με το προηγούμενο, αλλά δεν εμφανίζεται το 55:

```
  [1] 10
  [1] 25
  [1] 40
  [1] 70
  [1] 85
```

Στο παράδειγμα που προηγήθηκε, πριν την εντολή next ο κώδικας φροντίζει να αλλάξει την τιμή του x στην επόμενη θεμιτή τιμή του (δηλαδή να το αυξήσει κατά 15). Αυτό γίνεται ώστε το x να έχει κατάλληλη νέα τιμή στην επόμενη επανάληψη που θα ξεκινήσει μετά τη διακοπή της τρέχουσας από τη next. Δοκιμάστε να διαγράψετε την εντολή αυτή (το x <- x + 15 στην 5η γραμμή του παραδείγματος). Αν το κάνετε και το εκτελέσετε, θα «κολλήσει»[170]. Γιατί συμβαίνει αυτό; Στο προγραμματισμό βρόχων, ειδικά όταν χρησιμοποιείται η repeat (όπως σε αυτή την περίπτωση) χρειάζεται ιδιαίτερη προσοχή καθώς παρεμφερή λάθη που οδηγούν σε «ατέρμονες βρόχους» είναι αρκετά εύκολο να γίνουν. Στο παράδειγμα χρησιμοποιείται repeat, που θα επαναλαμβάνει το μπλοκ κώδικα για πάντα, εκτός αν κληθεί break. Στον συγκεκριμένο κώδικα, η break θα κληθεί μόνο αν το x είναι μεγαλύτερο ή ίσο του 100, ενώ το x αυξάνει τιμή στο τέλος του μπλοκ κώδικα που επαναλαμβάνεται. Οπότε αν το x βρεθεί με τιμή 55 και κληθεί η next (που διακόπτει την τρέχουσα επανάληψη) χωρίς πριν να έχει γίνει η προσαύξηση του x κατά 15, το x θα παραμένει με την ίδια τιμή (δηλαδή 55) στην επόμενη επανάληψη. Μην έχοντας αλλάξει κάτι στο σχετικό περιβάλλον εκτέλεσης (που καθορίζει τη συμπεριφορά του κώδικα, εδώ αυτό είναι μόνο το x) τόσο η επόμενη όσο και κάθε επόμενη επανάληψη θα είναι πανομοιότυπες με την προηγούμενη. Το x θα παραμένει σταθερά 55 και ποτέ δεν θα γίνει μεγαλύτερο ή ίσο του 100 ώστε να κληθεί η break. Τότε ο κώδικας θα μπει σε ένα ατελείωτο, ατέρμων loop.

Συχνά τέτοια προβλήματα οφείλονται σε παράληψη ορισμού του μπλοκ κώδικα που πρέπει να επαναληφθεί. Αν π.χ. αφαιρεθεί ο ορισμός μπλοκ κώδικα (τα { και }) από το παράδειγμα της while που χρησιμοποιήθηκε παραπάνω, ο κώδικας δεν θα διαφέρει πολύ οπτικά:

```
  x <- 10
  while (x < 100)
    print(x)
    x <-  x + 15
```

Η έξοδός του όμως θα αλλάξει δραματικά:

```
  [1] 10
  [1] 10
  [1] 10
  [1] 10
  [1] 10
```

ενώ το παραπάνω θα συνεχίζεται για πάντα[171]. Εδώ έχουμε μια περίπτωση όπου λανθασμένα δεν δημιουργήθηκε μπλοκ κώδικα. Έτσι η while εκτελεί μόνο την print, άρα το x δεν αλλάζει από την αρχική τιμή του (10), μένει πάντα μικρότερο του 100 και οι επαναλήψεις δεν τελειώνουν ποτέ. Τέτοια λάθη, καθώς δεν είναι συντακτικά, μπορούν να περάσουν απαρατήρητα, οδηγούν σε ατέρμονες βρόχους και είναι κλασική αιτία για την οποία τα προγράμματα ξαφνικά «κολλάνε». Για τον λόγο αυτό (ως «μέτρο αμυντικού προγραμματισμού» όπως το ονομάζει η σχετική τεκμηρίωση, βλ. help(Control)) προτείνεται να υπάρχει πάντα μπλοκ κώδικα για τις εντολές που θα επαναλαμβάνει μια δομή βρόχων, ακόμα και αν πρόκειται για μία μόνο εντολή.

Σε πολλά προβλήματα είναι απαραίτητο να οριστούν βρόχοι μέσα σε άλλους βρόχους, δηλαδή εμφωλευμένοι βρόχοι (nested loops). Προφανώς μπορεί να είναι ίδιου ή διαφορετικού τύπου βρόχοι καθώς και παραπάνω από δύο. Τα τελευταία δύο παραδείγματα της ενότητας αυτής χρησιμοποιούν εμφωλευμένους βρόχους. Στο αμέσως επόμενο, υπάρχει ένα βρόχος ορισμένος με for που επαναλαμβάνεται εντός ενός άλλου (επίσης for). Εδώ το c είναι απλώς ένας μετρητής που αυξάνει κατά 1 κάθε φορά που εκτελείται ο εσωτερικός κώδικας ώστε να εμφανίζεται (με την cat) το σχετικό μήνυμα:

---

[170] Για να σταματήσετε την εκτέλεση πατήστε το πλήκτρο Esc (ή Ctrl+C) στο πληκτρολόγιο ή το κόκκινο εικονίδιο Stop στο RStudio.

[171] βλ. προηγ. υποσημείωση.

```
{
  c = 0
  for (i in 10:11)
    for (j in 1:3)
    {
      c <- c + 1
      cat("Επανάληψη #", c, "όπου i=", i, "και j=", j, "\n")
    }
}
```

Η έξοδος είναι:

```
Επανάληψη # 1 όπου i= 10 και j= 1
Επανάληψη # 2 όπου i= 10 και j= 2
Επανάληψη # 3 όπου i= 10 και j= 3
Επανάληψη # 4 όπου i= 11 και j= 1
Επανάληψη # 5 όπου i= 11 και j= 2
Επανάληψη # 6 όπου i= 11 και j= 3
```

Ο εξωτερικός βρόχος (που επαναλαμβάνεται 2 φορές, για i από 10 έως 11) σε κάθε επανάληψή του προκαλεί την επανάληψη του εσωτερικού βρόχου (για j από 1 έως 3, άρα 3 επαναλήψεις). Συνολικά ο εσωτερικός κώδικας εκτελείται 6 (=2x3) φορές.

Το τελευταίο παράδειγμα στην ενότητα των βρόχων συνδυάζει while και repeat. Το υποθετικό σενάριο εδώ είναι πως ζητείται από τον χρήστη να καταχωρήσει μία-μία τις ποσότητες που δόθηκαν σε κάθε πώληση ενός προϊόντος με διαθέσιμη ποσότητα p (εδώ αρχικά 80) και να μετρηθούν πόσες πωλήσεις έγιναν μέχρι να εξαντληθεί. Ζητούμενο εδώ είναι να επαναληφθεί μια διεργασία, που αντιστοιχεί σε μία πώληση, όσο (while) υπάρχει ακόμα διαθέσιμη ποσότητα (p>0). Σε κάθε επανάληψη ζητείται από τον χρήστη η ποσότητα της πώλησης. Για να γίνει έλεγχος της εγκυρότητας των δεδομένων που εισάγει ο χρήστης, αυτό γίνεται μέσα σε έναν βρόχο repeat ο οποίος «σπάει» (break) μόνο αν δοθεί έγκυρη ποσότητα (x) δηλαδή θετική και μεγαλύτερη από το μηδέν[172], αλλιώς παραμένει στον βρόχο του repeat και άρα ξαναζητά νέο x:

```
{
  p <- 80              # η διαθέσιμη ποσότητα (εδώ αρχικά 80)
  n <- 0               # ένας μετρητής συναλλαγών (αρχικά 0)

  while (p > 0)   # όσο υπάρχουν διαθέσιμα…
  {
    repeat        # βρόχος, ζητά ποσότητα x από τον χρήστη.
    {
    x <- readline(
        paste("Διαθέσιμο",p,"- δώστε ποσότητα που ζητήθηκε:"))
    x <- as.numeric(x)

# αν δοθεί αριθμός x με 0 < x <= p, το x είναι έγκυρη ποσότητα.
# αν όχι, επιστρέφει στην αρχή του repeat και ζητά νέα.

    if (!is.na(x) && 0 < x && x <= p)
      break
    else
      cat("Λάθος! Ζητήθηκαν", x, "με", p, "διαθέσιμα.\n")
    }

    # δόθηκε έγκυρη ποσότητα x, γίνεται η πώληση.
```

[172] Για απλούστευση της επεξήγησης του παραδείγματος παραλείπεται παραπάνω πως γίνεται και έλεγχος ότι το κείμενο που καταχωρεί ο χρήστης είναι όντως αριθμός. Εδώ, η καταχώριση στο x γίνεται μέσω της συνάρτηση readline (βλ. §2.3 Συναρτήσεις κειμένου και ο βασικός τύπος character) που επιστρέφει κείμενο (character). Ακολούθως γίνεται μετατροπή του όποιου κειμένου στο x σε αριθμό (numeric, μέσω της συνάρτηση as.numeric). Αν αυτή η μετατροπή δεν μπορεί να γίνει, η τιμή που θα πάρει το x είναι NA. Αυτός είναι ο λόγος που ο πρώτος έλεγχος στην if είναι πως το x δεν είναι NA, δηλαδή !is.na(x).

```
        p <- p - x          # αφαιρείται το x από τα διαθέσιμα.
        n <- n + 1          # καταμετράται άλλη μια πώληση.
      }

      cat("Εξαντλήθηκε όλη η ποσότητα σε", n, "πωλήσεις.\n")
    }
```

Αποτέλεσμα εκτέλεσης του κώδικα, με υποθετική καταχώρηση από τον χρήστη τις ποσότητες 30, 60 (μη διαθέσιμη), 20, -10 (αρνητική) και 30 είναι:

```
Διαθέσιμο 80 - δώστε ποσότητα που ζητήθηκε:30
Διαθέσιμο 50 - δώστε ποσότητα που ζητήθηκε:60
Λάθος! Ζητήθηκαν 60 με 50 διαθέσιμα.
Διαθέσιμο 50 - δώστε ποσότητα που ζητήθηκε:20
Διαθέσιμο 30 - δώστε ποσότητα που ζητήθηκε:-10
Λάθος! Ζητήθηκαν -10 με 30 διαθέσιμα.
Διαθέσιμο 30 - δώστε ποσότητα που ζητήθηκε:30
Εξαντλήθηκε όλη η ποσότητα σε 3 πωλήσεις.
```

3.2.2.4 Τερματισμός εκτέλεσης

Σε κάποιες περιπτώσεις πρέπει να σταματήσει η εκτέλεση του κώδικα πριν αυτή ολοκληρωθεί, συνήθως λόγω κάποιας κατάστασης την οποία ο κώδικας δεν μπορεί να χειριστεί. Οι συναρτήσεις **stop** και **stopifnot** του πακέτου base μπορούν να διακόψουν τον κώδικα, εμφανίζοντας ένα σχετικό μήνυμα λάθους. Τα παρακάτω παραδείγματα θα μπορούσαν να είναι μέρος ενός σεναρίου:

Δοκιμάστε:	**Σχόλιο**
`stop("Τέλος")`	Σταματά την εκτέλεση και εμφανίζει το μήνυμα λάθους «Τέλος».
`stopifnot(x>0,is.character(y))`	Σταματά την εκτέλεση αν το x<=0 ή το y δεν είναι character.

Οι συναρτήσεις αυτές χρησιμοποιούνται σε περίπτωση αντικανονικών συνθηκών κατά την εκτέλεση του κώδικα. Για περισσότερα σχετικά με αυτές και άλλες συναρτήσεις χειρισμού εξαιρέσεων, βλ.§5.1.4 Έγερση και χειρισμός σφαλμάτων.

3.3 Στοιχεία διεπαφής χρήστη (user interface)

Σε αντίθεση με την απευθείας καταχώρηση εντολών στο Console, ένα μεγαλύτερο τμήμα κώδικα R (π.χ. σε ένα σενάριο) πιθανότατα θα ξαναχρησιμοποιηθεί. Σε αυτή την περίπτωση (και ανάλογα με τη λύση που υλοποιείται) μπορεί δικαιολογείται η βελτίωση της διάδρασης του κώδικα με τους χρήστες του, με αξιοποίηση παραθύρων και άλλων συναφών στοιχείων διεπαφής χρήστη (user interface). Διάφορα πακέτα της R παρέχουν σχετικές συναρτήσεις και ένας μικρός ενδεικτικός αριθμός από αυτές συνοψίζονται στην ενότητα αυτή.

Τα προ-εγκατεστημένα πακέτα ‘base’ και ‘util’ παρέχουν κάποιες συναρτήσεις που διευκολύνουν τη χρήση του εκτελούμενου κώδικα, μεταξύ αυτών τις **askYesNo**, **file.choose**, **choose.dir** και **choose.files**. Η askYesNo εμφανίζει παράθυρο τύπου message box με επιλογές Ναι’, ‘Όχι’, Άκυρο’ και επιστρέφει τιμή ενδεικτική της επιλογής που έγινε. Οι υπόλοιπες εμφανίζουν παράθυρα επιλογής αρχείων ή φακέλων (file requester) και επιστρέφουν την επιλογή με τη διαδρομή της (path) στον δίσκο.

Δοκιμάστε:	**Σχόλιο**
`askYesNo("Να προχωρήσω;")`	Παράθυρο με μήνυμα «Να προχωρήσω;», επιστρέφει την επιλογή.
`file.choose()`	Παράθυρο επιλογής αρχείου.
`choose.dir()`	Παράθυρο επιλογής φακέλου.

Η choose.files επιτρέπει την επιλογή πολλαπλών αρχείων. Για παράδειγμα, η επόμενη εντολή εμφανίζει παράθυρο επιλογής αρχείων, ενώ δίνει τη δυνατότητα να εμφανίζονται μόνο τα αρχεία κειμένου (με επέκταση txt). Η παράμετρος index = 2 ορίζει ότι ως προεπιλογή θα χρησιμοποιηθεί το δεύτερο φίλτρο (All) εμφανίζοντας όλα τα αρχεία:

```
choose.files(filters=Filters[c('txt','All'),],index=2)
```

Στο πακέτο[173] ‘svDialogs' [48] υπάρχουν συναρτήσεις δημιουργίας τυπικών παραθύρων-διαλόγων[174] όπως οι **dlg_open, dlg_save**, **dlg_dir**, **dlg_input**, **dlg_message**, **dlg_list**, **dlg_form** κ.α. Παραδείγματα από τις συναρτήσεις αυτές:

Δοκιμάστε:	**Σχόλιο**
`dlgOpen()$res`	Παράθυρο επιλογής αρχείου για άνοιγμα.
`dlgSave()$res`	Παράθυρο επιλογής αρχείου για αποθήκευση.
`dlg_dir()$res`	Παράθυρο επιλογής φακέλου.
`dlg_input("Poso;",10)$res`	Με μήνυμα «Poso;», επιστρέφει το κείμενο από τον χρήστη.
`dlg_message("Go?","okcancel")$res`	Παράθυρο με μήνυμα «Go?», επιστρέφει “ok” ή “cancel”.
`dlg_list(choices=c(1,2,3))$res`	Παράθυρο με επιλογές 1,2 ή 3 επιστρέφει την επιλεγμένη.

Η δημιουργία ενός πιο πλήρους γραφικού περιβάλλοντος χρήστη (Graphical User Interface ή GUI) για τον κώδικα R μπορεί να επιτευχθεί με συναρτήσεις που παρέχουν πακέτα όπως το ‘RGtk2’ [49] το οποίο αξιοποιεί την εργαλειοθήκη GIMP ToolKit (GTK), το ‘qtbase’ για το σύστημα Qt, ή το ‘tcltk’ [50] για την εργαλειοθήκη Tk. Η δημιουργία σύνθετων GUI με τα πακέτα αυτά αναλύεται στο [51]. Η χρήση του πακέτου ‘tcltk’ παρουσιάζεται περαιτέρω στην §7.2 Tcl/Tk.

Το πακέτο ‘gWidgets2’ [52] (βασισμένο στο ‘gWidgets’), περιέχει συναρτήσεις που τυποποιούν τις εντολές δημιουργίας GUI και τις μεταφράζει σε οδηγίες προς τα προαναφερθέντα πακέτα (και εργαλειοθήκες). Στοχεύει στη διευκόλυνση του προγραμματιστή με απλοποίηση και τυποποίηση των εντολών δημιουργίας γραφικών στοιχείων, ανεξαρτήτως της εργαλειοθήκης που θα χρησιμοποιηθεί για την εμφάνισή τους. Η παραλλαγή του ‘gWidgets2’ που χρησιμοποιεί ως βάση το Tcl/Tk ονομάζεται ‘gWidgets2tcltk’, για περισσότερα βλ. §7.2 Tcl/Tk.

Αν και η ανάπτυξη σύνθετων GUI σε εφαρμογές R είναι δυνατή, η συνήθης χρήση της R ως εργαλείο ανάλυσης δεδομένων συνήθως δεν το απαιτεί. Πιο διαδεδομένη είναι η αξιοποίηση άλλων διαδραστικών μέσων που επιτρέπει το οικοσύστημα της R, όπως τα interactive plots[175], ή η αλληλεπίδραση με χρήστες μέσω εφαρμογής τεχνολογιών ιστού[176].

[173] Τα πρόσθετα πακέτα της παραγράφου αυτής μπορούν να εγκατασταθούν από το CRAN, βλ.§1.5.3 Εγκατάσταση και διαχείριση πρόσθετων πακέτων.

[174] Με ιδιότητες modal dialog, σταματούν την εκτέλεση του κώδικα μέχρι να κλείσουν από τον χρήστη.

[175] βλ.§9.2.5 Διαδραστικά γραφήματα.

[176] βλ.§9.3 Εφαρμογές web.

Αναφορές Κεφαλαίου 3

[44] Chang, W., Luraschi, W., & Mast, T. (2021). Profvis: Interactive Visualizations for Profiling R Code. https://CRAN.R-project.org/package=profvis

[45] Hester, J., & Wickham, H. (2020). Fs: Cross-Platform File System Operations Based on 'libuv'. https://CRAN.R-project.org/package=fs

[46] RStudio Support, «Using RStudio Projects». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2021, από https://support.rstudio.com/hc/en-us/articles/200526207-Using-RStudio-Projects

[47] Venables, W. N., Smith, D. M., & The R Core Team (2021). An Introduction to R. https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

[48] Grosjean, P., (2022). SciViews-R. https://www.sciviews.org/SciViews-R/

[49] Lawrence, M., & Temple Lang, D. (2010). RGtk2: A Graphical User Interface Toolkit for R. *Journal of Statistical Software, τόμ. 37, αρ. 8*, pp. 1--52. http://www.jstatsoft.org/v37/i08/

[50] R Core Team (2021). Tcltk: Tcl/Tk Interface, R Package version 4.1.2. https://www.R-project.org/

[51] Lawrence, M. F., & Verzani, J. (2012). *Programming Graphical User Interfaces in R*. New York: Chapman and Hall/CRC. http://doi.org/10.1201/9781315373898

[52] Verzani, J. (2022). gWidgets2: Rewrite of gWidgets API for Simplified GUI Construction. https://CRAN.R-project.org/package=gWidgets2

Κεφάλαιο 4: Συνήθεις τύποι αντικειμένων

Σύνοψη
Το κεφάλαιο αυτό περιγράφει συνήθεις τύπους (κλάσεις) αντικειμένων που χρησιμοποιούνται στην R και διάφορα θέματα που σχετίζονται με τη χρήση τους. Οι τύποι που περιγράφονται περιλαμβάνουν τα vector, matrix, array, list, environment, data.frame, tibble, data.table, ts, Matrix, language κ.α.. Το κεφάλαιο περιλαμβάνει επίσης θέματα που σχετίζονται με τις κλάσεις αυτές όπως ο μηχανισμός εμβέλειας μεταβλητών, οι ιδιότητες των αντικειμένων, η επεξεργασία στοιχείων σε τέτοιες δομές κλπ.).

Προαπαιτούμενη γνώση
Βασικές δεξιότητες χρήσης R, μεταβλητές και βασικοί τύποι (βλ. Κεφάλαιο 2).

4.1 Συνήθεις ατομικοί τύποι αντικειμένων

Στην R τα πάντα είναι αντικείμενα και ως τέτοια ανήκουν σε κάποιον τύπο (κλάση, class) αντικειμένων[177]. Έχουν ήδη παρουσιαστεί κάποιοι τύποι σχεδιασμένοι ώστε να περιέχουν βασικά δεδομένα. Τέτοιοι τύποι είναι οι numeric (για πραγματικούς αριθμούς), integer (για ακέραιους), complex (για μιγαδικούς), character (για κείμενο), logical (για τιμές άλγεβρας bool, δηλαδή TRUE ή FALSE) κλπ. Στο κεφάλαιο αυτό περιγράφονται μερικοί ακόμα συνήθεις τύποι αντικειμένων που είτε παρέχονται από τα προ-εγκατεστημένα πακέτα της R είτε από άλλα δημοφιλή πακέτα διαθέσιμα στο CRAN. Πολλοί από τους τύπους που αναφέρονται στο κεφάλαιο αυτό έχουν σκοπό να οργανώνουν τα δεδομένα υλοποιώντας κάποια δομή δεδομένων. Παρά τις ιδιαιτερότητες κάθε τύπου (κάποιες εκ των οποίων περιγράφονται στις αντίστοιχες ενότητες), διαφορετικοί τύποι υποστηρίζουν συχνά τις ίδιες συναρτήσεις όταν αυτό αρμόζει στον σκοπό του τύπου και τη φύση της συνάρτησης. Αυτό συχνά οφείλεται στο ότι πολλοί τύποι αντικειμένων είναι επέκταση κάποιων άλλων, άρα υποστηρίζουν τις ίδιες συναρτήσεις. Για παράδειγμα, όλοι οι τύποι υποστηρίζουν συναρτήσεις όπως η class, ενώ όσοι βασίζονται σε διανύσματα υποστηρίζουν τη συνάρτηση length. Τέλος, όταν αρμόζει και είναι εφικτό, παρεμφερούς λειτουργικότητας συναρτήσεις που υποστηρίζουν οι διάφοροι τύποι αντικειμένων ακολουθούν κοινά μοτίβα σύνταξης. Έτσι, εντολές που εφαρμόζονται σε έναν τύπο είναι συχνά εφαρμόσιμες σε άλλους τύπους, με λίγες ή καθόλου αλλαγές.

Στη βιβλιογραφία της R γίνεται συχνά αναφορά σε ατομικούς (atomic) και μη-ατομικούς τύπους αντικειμένων. Πρακτικά, η διαφοροποίηση σημαίνει το παρακάτω: ένας atomic τύπος αντικείμενων επιβάλλει πως όλα τα στοιχεία που θα αποθηκεύονται στα αντικείμενα αυτά πρέπει να είναι του ίδιου βασικού είδους δεδομένων (αριθμοί, λογικές τιμές, κείμενο κλπ.). Σε αυτή την περίπτωση η R μπορεί να εφαρμόσει μαζικά σε όλα τα δεδομένα τις ίδιες συναρτήσεις (π.χ. αριθμητικές πράξεις αν τα δεδομένα είναι αριθμοί).

Αυτή είναι η πρακτική προσέγγιση, όμως εσωτερικά συμβαίνει κάτι λίγο διαφορετικό. Τα διαθέσιμα mode των βασικών τύπων αντιστοιχούν στα είδη βασικών δεδομένων που υποστηρίζει η R και ορίζουν πώς αποθηκεύονται τα δεδομένα αυτά στη μνήμη του υπολογιστή. Τη στιγμή που γράφεται αυτό το βιβλίο, τα atomic modes είναι "logical", "integer", "numeric" (ή "double"), "complex", "character" και "raw"[178]. Είναι δηλαδή αυτά τα οποία έως τώρα αναφέραμε ως βασικούς τύπους δεδομένων. Ο έλεγχος αν ένα αντικείμενο λειτουργεί σε ατομική κατάσταση (atomic mode) γίνεται με τη συνάρτηση **is.atomic**, ενώ όλοι οι atomic (βασικοί) τύποι είναι βασισμένοι στον τύπο διάνυσμα. Επειδή λοιπόν το διάνυσμα (σε atomic mode) είναι ένας ιδιαίτερα βασικός τύπος αντικειμένων στην R, αναφερόμαστε σε αυτόν λίγο πιο αναλυτικά αμέσως παρακάτω.

4.1.1 Ο τύπος vector (διάνυσμα)

Σε προηγούμενο κεφάλαιο (§2.4 Εισαγωγή στα διανύσματα) έγινε μια σύντομη χρηστική εισαγωγή στα διανύσματα τα οποία είναι ο πλέον σημαντικός τύπος αντικειμένου στην R. Πριν προχωρήσετε παρακάτω, καλό είναι να έχετε ανατρέξει πρώτα στη σχετική ενότητα. Συνοψίζοντας όσα αναφέρονται εκεί, συνήθεις τρόποι δημιουργίας διανυσμάτων είναι η συνάρτηση '**c**' και ο τελεστής '**:**' ενώ όλα τα στοιχεία που περιέχει ένα

[177] βλ.§4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων. Σχετικά με τη δημιουργία νέων κλάσεων βλ. Κεφάλαιο 6.
[178] βλ. help(vector).

διάνυσμα πρέπει να είναι αντικείμενα ίδιου τύπου και μπορεί να ανήκουν σε κάποιον βασικό τύπο ο οποίος όμως δεν είναι απαραίτητα αριθμητικός (όπως logical, character κλπ.).

Αναφέρθηκε επίσης ότι στην R όλοι οι βασικοί τύποι αντικειμένων που παρέχονται για την αποθήκευση δεδομένων (logical, integer, numeric ή double, complex, character και raw) βασίζονται και επεκτείνουν τον τύπο vector. Στην περίπτωση αυτή το vector βρίσκεται σε "ατομική κατάσταση" ή atomic mode, ενώ αναφέρεται και ως atomic vector[179] ή ως vector στο αντίστοιχο mode (π.χ. σε numeric mode αν πρόκειται για αντικείμενο της κλάσης numeric). Έτσι, ακόμα και μια μοναδική τιμή, για παράδειγμα ένα μοναδικό numeric (π.χ. ο αριθμός 30) ή ένα δεδομένο οποιουδήποτε άλλου βασικού είδους είναι ένα atomic vector που περιέχει το μοναδικό αυτό στοιχείο και βρίσκεται σε αντίστοιχη κατάσταση λειτουργίας (mode)[180]. Για τους βασικούς αυτούς τύπους το mode αποθήκευσης των δεδομένων ορίζει την κλάση (επέκταση του vector) στην οποία θα ανήκει το αντικείμενο και κατ' επέκταση τις συναρτήσεις που υποστηρίζονται (π.χ. αριθμητικές πράξεις αν το vector είναι σε numeric mode, άρα είναι τύπου numeric). Αυτό ισχύει είτε το διάνυσμα περιέχει μόνο ένα στοιχείο, είτε περιέχει περισσότερα.

Επίσης έγινε αναφορά στον τρόπο που γίνεται επεξεργασία δύο διανυσμάτων και την ανακύκλωση (recycle) στοιχείων του διανύσματος με το μικρότερο μήκος (αριθμό στοιχείων) εφόσον τα μήκη των δύο διανυσμάτων είναι συμβατά. Τέλος, έγινε μια περιγραφή της χρήσης δεικτών (μέσα σε αγκύλες []) για πρόσβαση σε συγκεκριμένα στοιχεία του διανύσματος.

Ένας τρόπος να δημιουργηθούν διανύσματα είναι με τη συνάρτηση[181] **vector**:

Δοκιμάστε:	**Σχόλιο**
`vector(mode="numeric",7)`	Διάνυσμα 7 numeric στοιχείων (σε numeric mode). Αρχικά όλα 0.
`numeric(7)`	Ίδιο με το παραπάνω.
`vector(mode="numeric",0)`	Κενό διάνυσμα (σε numeric mode) με μήκος = 0, επιτρέπεται.
`numeric(0)`	Ίδιο με το παραπάνω.

Ίσως φαίνεται περίεργο ότι μπορούν να δημιουργηθούν κενά (μηδενικού μήκους) διανύσματα, αλλά το μήκος του διανύσματος μπορεί να αλλάξει δυναμικά. Έτσι το διάνυσμα μπορεί να επεκταθεί[182] είτε απλά αναθέτοντας νέα τιμή στο length του, ή με ανάθεση τιμής σε θέση στοιχείου εκτός του τρέχοντος length:

Δοκιμάστε:	**Σχόλιο**
`x<-vector(mode="numeric",7)`	Διάνυσμα 7 numeric στοιχείων (σε numeric mode). Αρχικά όλα 0.
`length(x)`	Το τρέχον μήκος (αριθμός στοιχείων) στο x, εδώ 7.
`length(x)<-3`	Αλλαγή του μήκους του διανύσματος από 7.
`x[10]<-2`	Ανάθεση σε θέση 10 (εκτός τρέχοντος length) αλλάζει το μήκος σε 10.

Αυτό επιτρέπει έναν δεύτερο εναλλακτικό (και δυσανάγνωστο) τρόπο δημιουργίας ενός διανύσματος επεκτείνοντας κάποιο αντικείμενο βασικού τύπου:

Δοκιμάστε:	**Σχόλιο**
`x<-"test"`	Ένα αντικείμενο σε character mode στη μεταβλητή x.
`length(x)<-3`	Επέκταση του x σε 3 στοιχεία (πάντα σε character mode).

Για να μετατραπεί ένα αντικείμενο σε vector ή να ελεγχθεί το mode ενός vector μπορούν να χρησιμοποιηθούν οι συναρτήσεις **as.vector** και **is.vector**:

Δοκιμάστε:	**Σχόλιο**
`as.vector(TRUE)`	Διάνυσμα 1 στοιχείου, σε logical mode (το TRUE είναι logical).
`x<-c(3,10,4)`	Το x είναι διάνυσμα σε numeric mode.
`x<-as.vector(x,mode="character")`	Αλλαγή του x σε character mode.
`x<-x+1`	Λάθος, ως character δεν υποστηρίζονται αριθμητικές πράξεις.

[179] Για vector τα οποία δεν είναι σε atomic mode βλ.§4.2.1 Ο τύπος list (λίστα).
[180] Δοκιμάστε για παράδειγμα την εντολή is.vector(30) η οποία επιστρέφει TRUE αφού το numeric αντικείμενο 30 είναι vector με ένα στοιχείο.
[181] Όπου δεν αναφέρεται διαφορετικά, όλες οι συναρτήσεις του κεφαλαίου 4 παρέχονται από το πακέτο 'base'.
[182] Η R πιθανότατα θα αντικαταστήσει το αντικείμενο στο οποίο αναφέρεται η μεταβλητή με άλλο, μεγαλύτερου μήκους.

`is.vector(x, mode="any")`	Είναι το x οποιουδήποτε τύπου vector; Επιστρέφει TRUE.
`is.vector(x, mode = "numeric")`	Είναι το x vector σε numeric mode; Επιστρέφει FALSE.

Όμως, όπως έχει ήδη αναφερθεί, ο πλέον συνήθης τρόπος δημιουργίας διανυσμάτων είναι με τη συνάρτηση '**c**'. Η συνάρτηση c δέχεται μια σειρά από αντικείμενα και αν ο τύπος τους το επιτρέπει (είναι ένα από τα βασικά είδη δεδομένων που αναφέρθηκαν ήδη), επιστρέφει ένα αντικείμενο τύπου vector στο αντίστοιχο atomic mode, αλλιώς επιστρέφει ένα αντικείμενο τύπου list (που είναι απλώς ένα vector σε ειδικό (μη ατομικό) mode, βλ.§4.2.1 Ο τύπος list (λίστα). Κατά τη δημιουργία του vector από τη c αν τα στοιχεία που έχουν δοθεί ως παράμετροι βρίσκονται σε διαφορετικά atomic mode θα αλλάξουν σε ένα ενιαίο[183] ώστε το διάνυσμα που θα προκύψει να είναι atomic (και όχι list). Η ιεραρχία μετατροπής (από το χαμηλότερο στο υψηλότερο) είναι: NULL, raw, logical, integer, numeric (double), complex, character, list, expression. Παρατηρώντας την ιεραρχία βλέπουμε πως σχετίζεται με την ευκολία με την οποία αλλάζουν τα δεδομένα από το ένα mode στο άλλο. Για παράδειγμα, είναι ανώδυνο να αλλάξει ένα οποιοδήποτε integer σε numeric αλλά το αντίθετο ίσως οδηγήσει σε απώλεια του δεκαδικού μέρους. Για αυτό, το numeric είναι ψηλότερα στην ιεραρχία από το integer. Έτσι, αν δοθεί στη c ένα μείγμα τέτοιων αντικειμένων, θα μετατραπούν όσα βρίσκονται σε mode χαμηλότερα στην ιεραρχία σε όποιο mode βρίσκεται υψηλότερα. Π.χ. αν δοθεί ένα μείγμα από logical και numeric θα προκύψει vector σε numeric mode, καθώς το logical mode είναι χαμηλότερα στην ιεραρχία από το numeric:

Δοκιμάστε:	**Σχόλιο**
`c(6,2,2,4)`	Διάνυσμα 4 στοιχείων. Επιστρέφει διάνυσμα σε numeric mode.
`c(6,T,2,4)`	Μετατροπή του T. Επιστρέφει διάνυσμα σε numeric mode, δηλαδή c(6,1,2,4).
`c(6,"a",2,4)`	Μετατροπές των numeric. Επιστρέφει διάνυσμα σε character mode, το c("6","a","2","4").
`mode(c(6,"a"))`	Το διάνυσμα μετατρέπεται σε c("6","a"). Το επιστραφέν mode είναι "character"

Έγινε ήδη στην προηγούμενη σχετική ενότητα (§2.4 Εισαγωγή στα διανύσματα) μια πρώτη αναφορά στον τρόπο πρόσβασης στα επιμέρους στοιχεία ενός διανύσματος. Αυτό γίνεται με προσδιορισμό δεικτών μέσα σε αγκύλες [] και ήδη χρησιμοποιήσαμε δείκτες σε κάποια από τα παραπάνω παραδείγματα. Οι αγκύλες [] είναι ένας τελεστής επιλογής στοιχείων ενός αντικειμένου ώστε να εξαχθούν από αυτό ή να αλλάξουν. Ως τελεστής, είναι μια συνάρτηση, η οποία μάλιστα δέχεται εκτός από δείκτες και άλλες παραμέτρους[184]. Όταν δοθεί ως παράμετρος ένας μοναδικός αριθμός θέσης (δείκτης) μέσα στις αγκύλες [ ], αυτό δίνει πρόσβαση στο αντίστοιχο στοιχείο του διανύσματος. Για παράδειγμα το v[5] δίνει πρόσβαση στο 5° στοιχείο ενός διανύσματος v. Μπορούν όμως να οριστούν περισσότεροι δείκτες. Σε κάθε περίπτωση, ο τελεστής χειρίζεται τα επιλεγμένα στοιχεία ως διάνυσμα, π.χ. στο v[7:9] τα επιλεγμένα στοιχεία είναι οι θέσεις 7 έως 9 του διανύσματος v. Αν οι τιμές τους τροποποιηθούν, το διάνυσμα που θα προκύψει θα έχει αλλαγές μόνο στα συγκεκριμένα στοιχεία, ενώ αν ανακληθούν οι τιμές τους αντιγράφονται και επιστρέφονται σε ένα νέο διάνυσμα:

Δοκιμάστε:	**Σχόλιο**
`v<-1:9`	Διάνυσμα 9 numeric, με τιμές από το 1 έως το 9.
`v[1]<-0`	Αντικατάσταση του 1ου στοιχείου με το 0.
`v[2:4]<-1`	Αντικατάσταση του 2ου , 3ου και 4ου στοιχείου με το 1.
`v[6:8][2]<-10`	Αντικατάσταση του 2ου στοιχείου της επιλογής στοιχείων 6 έως 8 (το 7ο στοιχείο) με 10.
`v[4:9]`	Ανακαλεί τιμές στοιχείων στις θέσεις 4 έως 9, επιστρέφει διάνυσμα c(1, 5, 6, 10, 8, 9).

Το επόμενο παράδειγμα συνοψίζει άλλες τυπικές χρήσεις του τελεστή [], με τις θέσεις να προσδιορίζονται ως διανύσματα αριθμών (πολλαπλοί δείκτες), διανύσματα αρνητικών αριθμών (για θέσεις που δεν πρέπει να συμπεριληφθούν) και διανύσματα λογικών τιμών (όπου το TRUE επιλέγει το στοιχείο στην αντίστοιχη θέση του διανύσματος):

Δοκιμάστε:	**Σχόλιο**
`x<-c(11,21,31,41)`	Το x είναι διάνυσμα 4 numeric στοιχείων.
`x[3]`	Το 3ο στοιχείο του x (περιέχει 31).
`x[0]`	Η ειδική «θέση» 0 δίνει πρόσβαση στο mode (εδώ numeric).
`x[-3]`	Τα στοιχεία του x εκτός του 3ου (αυτά που περιέχουν τα 11, 12 και 41).

[183] Αυτή η διαδικασία έχει ήδη αναφερθεί ως coercion (αναγκασμός) καθώς εξαναγκάζει την αλλαγή, βλ. και help(c).
[184] βλ. help("[").

x[-1:-3]	Τα στοιχεία του x εκτός του $1^{ου}$, $2^{ου}$ και $3^{ου}$ (μόνο $4^{ο}$ που περιέχει το 41).
x[c(F,F,T,T)]	Τα στοιχεία του x επιλεγμένα με T (το $3^{ο}$ και $4^{ο}$, που περιέχουν 31 και 41).
x[c(F,T)]	Κάθε $2^{ο}$ στοιχείο του x. Γίνεται ανακύκλωση τιμών (F,T,F,T), άρα $2^{ο}$ και $4^{ο}$.
x[x>30]	Τα στοιχεία του x που είναι > 30. Ίδιο αποτέλεσμα με το προηγούμενο.
x[x>30]<-100	Τα στοιχεία του x που είναι > 30 (το $3^{ο}$ και $4^{ο}$) να πάρουν την τιμή 100.

Αν μέσω του [] αντικατασταθεί κάποιο στοιχείο με δεδομένα που είναι άλλου mode, στο vector που θα προκύψει θα έχουν εφαρμοστεί οι κανόνες (και η ιεραρχία) μετατροπής που περιγράφτηκαν παραπάνω. Στην περίπτωση αυτή θα έχει αλλάξει είτε το mode του νέου στοιχείου (αν είναι χαμηλότερα στην ιεραρχία από το τρέχον mode του vector) είτε το mode του vector (αν το νέο στοιχείο βρίσκεται σε mode υψηλότερα στην ιεραρχία από το τρέχον):

Δοκιμάστε:	**Σχόλιο**
x<-c(6,2,2,4)	Διάνυσμα (σε numeric mode) που περιέχει 4 στοιχεία, με όνομα x.
x[3]<-TRUE	Αλλαγή του $3^{ου}$ στοιχείου σε logical τιμή, το logical TRUE μετατρέπεται σε numeric (άρα 1).
x[1]<-1+2i	Αλλαγή του $1^{ου}$ στοιχείου σε complex τιμή, το mode του διανύσματος αλλάζει σε complex.

Εξαίρεση στα παραπάνω αποτελεί η ειδική logical τιμή NA («μη διαθέσιμα δεδομένα») η οποία δεν αλλάζει το mode του διανύσματος, ενώ η ειδική τιμή NULL («τίποτα») δεν μπορεί να αποθηκευτεί σε vectors:

Δοκιμάστε:	**Σχόλιο**
as.vector(NULL)	Επιστρέφει NULL.
x<-c(NaN,5,NA,NULL,Inf)	Το x είναι διάνυσμα σε numeric mode (τα NaN, 5, Inf είναι numeric)
length(x)	Το μήκος του x είναι 4 (το NULL δεν αποθηκεύτηκε).
length(x[!is.na(x)])	Το μήκος των έγκυρων τιμών είναι 2 (το 5 και το Inf).

Έχουμε δει ήδη πως στα στοιχεία ενός vector μπορούν να προστεθούν ονόματα (ετικέτες)[185]. Η πρόσβαση στα ονόματα των στοιχείων που υπάρχουν σε ένα vector γίνεται με τη συνάρτηση **names** με την οποία μπορεί να γίνει ανάθεση ή επιστροφή των ονομάτων μέσω ενός άλλου vector από λεκτικά (character)[186]:

Δοκιμάστε:	**Σχόλιο**
x<-vector(mode="numeric",3)	Το x είναι διάνυσμα 3 numeric στοιχείων.
names(x)<-c('Γιωργος',"Μαρία","Ελένη")	Ονόματα για τα στοιχεία αυτά.
x["Γιώργος"]<-14	Πρόσβαση στο $1^{ο}$ στοιχείο μέσω ονόματος.
x["Ελένη"]<-19	Πρόσβαση στο $3^{ο}$ στοιχείο μέσω ονόματος.
x["Μαρία"]<-13	Πρόσβαση στο $2^{ο}$ στοιχείο μέσω ονόματος.

Με τη συνάρτηση c μπορούν να οριστούν απευθείας τα ονόματα των στοιχείων. Έτσι το παραπάνω παράδειγμα θα μπορούσε να συμπτυχθεί σε:

```
x<-c(Γιώργος=14,Μαρία=13,Ελένη=19)
```

Πέραν των ‘c’ και ‘:’, άλλες χρήσιμες συναρτήσεις δημιουργίας διανυσμάτων είναι οι **seq** (regular sequence/κανονική ακολουθία) και **rep** (repeat/επανάληψη):

Δοκιμάστε:	**Σχόλιο**
seq(10,30)	Διάνυσμα κανονικής ακολουθίας από 10 έως 30. Ίδιο με το 10:30.
seq(10,30,by=3)	Από το 10 έως το 30 με βήμα 3. Ίδιο με seq(10,30,3).
seq(10,30,length.out=5)	Από το 10 έως το 30 με όποιο βήμα παράγει 5 στοιχεία.
seq(30,10,length.out=9)	Από το 30 έως το 10 με όποιο (αρνητικό) βήμα παράγει 9 στοιχεία.
seq(10,by=.5,length.out=10)	Από το 10, με βήμα 0.5, να παραχθεί διάνυσμα με 10 στοιχεία.
rep(4,2)	Διάνυσμα όπου το 4 επαναλαμβάνεται 2 φορές, δηλαδή c(2,2).
rep(1:4,2)	1 έως 4, 2 φορές, δηλαδή c(1,2,3,4,1,2,3,4).

[185] βλ. και παράδειγμα χρήσης των ονομάτων στο $2^{ο}$ παράδειγμα της §2.4.2 Δύο παραδείγματα χρήσης vector και η σημασία της ειδικής τιμής NA.

[186] Τα ονόματα προστίθενται ως ιδιότητα στο σύνολο ιδιοτήτων του αντικειμένου, βλ. συνάρτηση attributes και §4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων.

`rep(c(5,2),2)`	Διάνυσμα όπου το c(5,2) επαναλαμβάνεται 2 φορές, δηλαδή c(5,2,5,2).
`rep("a",2)`	Διάνυσμα όπου το "a" επαναλαμβάνεται 2 φορές, δηλαδή c("a","a").
`rep(c(5,2),c(3,4))`	3 φορές το 5, 4 φορές το 2 (απαιτεί ίδιο αριθμό στοιχείων).
`rep(c(5,6,7),2:4)`	2 φορές το 5, 3 φορές το 6, 4 φορές το 7, δηλαδή c(5,5,6,6,6,7,7,7,7).
`rep(c(5,6,7),each=2)`	Επανάληψη κάθε μέλους 2 φορές, δηλαδή c(5,5,6,6,7,7).
`x<-2`	Ανάθεση μεταβλητής x.
`rep(x,x)`	x φορές το x, δηλαδή c(2,2) εφόσον x=2.
`c(rep(4,x),c(5,6),seq(7,10))`	Συνδυασμός, επιστρέφει c(4,4,5,6,7,8,9,10).

Συχνά χρειάζεται να επεκταθεί ένα vector με νέα δεδομένα, ή να συγχωνευτούν πολλά αντικείμενα vector σε ένα. Μερικοί τρόποι να γίνει αυτό δίνονται στα παρακάτω παραδείγματα:

Δοκιμάστε:	**Σχόλιο**
`x<-c(1,4)`	Ανάθεση σε μεταβλητή x του διανύσματος c(1,4).
`x<-c(x,5)`	Συγχώνευση του x με το αντικείμενο 5, το x γίνεται c(1,4,5).
`x<-append(x,6)`	Προσάρτηση στο x του 6, το x γίνεται c(1,4,5,6).
`x<-append(x,2:3,after=1)`	Προσάρτηση στο x του 2:3 μετά τη θέση 1, το x γίνεται c(1,2,3,4,5,6).
`x<-c(0,x,7:8)`	Συγχώνευση των 0, x και 7:8, το x γίνεται c(0,1,2,34,5,6,7,8).

Όμως, προσεγγίσεις όπως οι παραπάνω καλό είναι να γίνονται με σύνεση, ειδικά όταν εμπλέκονται διανύσματα μεγάλου μήκους. Όπως έχει προαναφερθεί (βλ.§2.2 Μεταβλητές, αριθμητικές συναρτήσεις και τύποι αντικειμένων) για να δημιουργηθεί μια μεταβλητή ή να αλλάξει η τιμή της πρέπει να γίνει σε αυτή μια ανάθεση νέας τιμής. Άρα πρώτα πρέπει να αποτιμηθεί ο κώδικας που υπολογίζει τη νέα τιμή (δηλαδή για ανάθεση με <- να δημιουργηθεί στη μνήμη το νέο αντικείμενο που αντιστοιχεί στον κώδικα της δεξιάς πλευράς της ανάθεσης) και στο αποτέλεσμα να δοθεί το όνομα της μεταβλητής. Έτσι π.χ. κατά την εκτέλεση του:

```
x<-c(1,2:100)
```

γίνονται εσωτερικά βήματα όπως: (α) Δημιουργείται το vector 2:100, με κατάληψη της σχετικής μνήμης. (β) Δημιουργείται νέο vector για το οποίο καταλαμβάνεται άλλο μέρος της μνήμης, στο οποίο αντιγράφονται τα περιεχόμενα των δυο αντικειμένων που συγχωνεύονται (τα 1 και 2:100). (γ) Διαγράφεται το vector 2:100 (αφού δεν οδηγεί σε αυτό καμία μεταβλητή) και ελευθερώνεται η μνήμη που καταλάμβανε. (δ) Ανατίθεται το αντικείμενο που δημιουργήθηκε στο (β) σε μεταβλητή x. Τα παραπάνω βήματα κατάληψης μνήμης, αντιγραφής δεδομένων και απελευθέρωσης μνήμης μπορεί να δημιουργήσουν σημαντικές αυξήσεις στον χρόνο εκτέλεσης και ανάγκες μνήμης του κώδικα, ειδικά αν εμπλέκονται ιδιαίτερα μεγάλα διανύσματα[187]. Για παράδειγμα:

```
x1<-rep(0,s)
x2<-c(rep(0,s))
x3<-c(0,rep(0,s-1))
```

Τα x0, x1 και x2 που δημιουργούνται με τον παραπάνω κώδικα θα είναι ίδια (όλα θα περιέχουν ένα vector μήκους s, με όλα τα στοιχεία του να είναι 0). Παρόλα αυτά, ο χρόνος εκτέλεσης και οι ανάγκες σε μνήμη για τη δημιουργία του x0 είναι υποπολλαπλάσια αυτών που απαιτούνται για τη δημιουργία των x1 και x2. Αυτό μπορεί να επιβεβαιωθεί με ένα ικανά μεγάλο s (π.χ. s<-1e+09) και χρησιμοποιώντας τεχνικές όπως αυτές που αναφέρονται στην §3.1.4 Εργαλεία προφίλ (profiling), συμπεριλαμβανομένης της συνάρτησης system.time. Συμπέρασμα των παραπάνω είναι πως αν στο πρόβλημα για το οποίο υλοποιείται κάποια λύση υπάρχει ανάγκη επεξεργασίας αντικείμενων με πολύ μεγάλο αριθμό στοιχείων, είναι καλό να εφαρμόζεται κάθε εφικτή πρόβλεψη που θα ελαχιστοποιεί τις άσκοπες τροποποιήσεις ή προσαρτήσεις των αντικειμένων αυτών.

4.1.2 Οι τύποι factor και ordered (παράγοντας)

Συγγενικός τύπος με τα διανύσματα είναι οι τύποι factor και ordered που περιέχουν πιθανές τιμές μιας ποιοτικής (qualitative) μεταβλητής[188]. Οι μεταβλητές αυτές συχνά ονομάζονται και κατηγορικές μεταβλητές ή παράγοντες[189]. Αν και αποθηκεύονται εσωτερικά σαν διανύσματα ακεραίων τιμών, οι τιμές στα στοιχεία τους συμβολίζουν μια κατηγορία και όχι μια αριθμητική τιμή, άρα οι συναρτήσεις που τα επεξεργάζονται (μπορούν

[187] Παρόμοια προβλήματα προφανώς εμφανίζονται και σε πολλούς άλλους τύπους αντικειμένων με πολλά στοιχεία.
[188] Ο τύπος factor είναι επέκταση του τύπου vector, κάτι που επιβεβαιώνεται με την εντολή: extends("factor","vector").
[189] Σε άλλες γλώσσες προγραμματισμού για τέτοια δεδομένα χρησιμοποιούνται τύποι απαριθμημένων τιμών (enumerated types).

να) τα αντιμετωπίζουν διαφορετικά από απλούς ακέραιους αριθμούς. Οι διαφορετικές τιμές που μπορεί να πάρει μία τέτοια μεταβλητή είναι συγκεκριμένες και ονομάζονται επίπεδα (levels). Ένα παράδειγμα χρήσης factor θα βρείτε στο προ-εγκατεστημένο σύνολο δεδομένων iris[190] όπου η 5η μεταβλητή (Species) είναι τύπου factor και περιέχει την κατηγορία (είδος λουλουδιού) της συγκεκριμένης περίπτωσης λουλουδιού iris. Έτσι, η μεταβλητή αυτή μπορεί να πάρει μόνο μια από τις τρεις διαφορετικές τιμές (levels) που έχουν οριστεί δηλαδή setosa, versicolor ή virginica (και αντιστοιχούν στους τρεις τύπους λουλουδιού iris που καταγράφει το συγκεκριμένο σύνολο δεδομένων).

Η συνάρτηση **factor** δημιουργεί εξ ορισμού ένα αντικείμενο παρεμφερές με διάνυσμα στο οποίο οι επιτρεπτές τιμές των στοιχείων του είναι ονομαστικές (nominal) και αντιστοιχούν σε συγκεκριμένες κατηγορίες. Ας δούμε ένα υποθετικό παράδειγμα: για τη συμμετοχή σε κάποια δραστηριότητα υπάρχουν 3 τύποι (κατηγορίες) εισιτηρίων: κανονικό, παιδικό και εκπτωτικό. Σήμερα έχουν καταγραφεί μέχρι στιγμής 5 συμμετέχοντες στη δραστηριότητα αυτή και ο τύπος του εισιτηρίου τους αποθηκεύτηκε σε μεταβλητή a:

```
a<-factor(c("κανονικό","παιδικό","κανονικό",
            "κανονικό","εκπτωτικό","κανονικό"))
```

Άρα το πρώτο εισιτήριο ήταν κανονικό, το επόμενο παιδικό και ούτω καθεξής. Η factor θεώρησε πως κάθε διαφορετικό κείμενο που δόθηκε στο αρχικό διάνυσμα αντιστοιχεί σε μια διαφορετική ονομαστική κατηγορία (level). Αν ανακαλέσετε το αντικείμενο a, τα επιτρεπτά levels αναφέρονται μετά τα ίδια τα δεδομένα:

```
> a
[1] κανονικό  παιδικό  κανονικό  κανονικό  εκπτωτικό κανονικό
Levels: εκπτωτικό κανονικό παιδικό
```

Συντακτικά, ο χειρισμός ενός αντικειμένου factor (και ordered που θα δούμε παρακάτω) είναι παρόμοιος με αυτόν των διανυσμάτων (vector). Η βασική διαφορά είναι πως επιτρέπονται ως στοιχεία μόνο οι προκαθορισμένες τιμές (τα «επίπεδα» ή level). Έτσι το γεγονός πως το αντικείμενο a είναι τύπου factor (με αυτά τα συγκεκριμένα επιτρεπτά level) περιορίζει τις αλλαγές που μπορούν να γίνουν στα δεδομένα που περιέχει:

Δοκιμάστε:	**Σχόλιο**
`a[3]<-"παιδικό"`	Επιτρέπεται και αλλάζει το $3^{ο}$ στοιχείο σε "παιδικό".
`a[4]<-"ειδικό"`	Λάθος. Δεν έχει οριστεί τέτοια κατηγορία. Το $4^{ο}$ στοιχείο αλλάζει σε NA.
`a[3:5]`	Η χρήση δεικτών είναι ίδια με τα vectors, εδώ τα $3^{ο}$, $4^{ο}$ και $5^{ο}$ στοιχείο του a.
`a[a>"παιδικό"]`	Λάθος. Κάποιες συγκρίσεις δεν έχουν νόημα σε ονομαστικές κατηγορίες.
`unclass(a)`	Το αντικείμενο χωρίς την κλάση (εδώ factor), επιστρέφει c(2, 3, 3, NA, 1, 2)[191].

Αν τα δεδομένα που πρέπει να χαρακτηριστούν με «ετικέτα» είναι αριθμητικά, ένας τρόπος δημιουργίας επιπέδων factor από τις ίδιες τις τιμές είναι μέσω της συνάρτησης **cut**. Η cut επιτρέπει να οριστούν όρια τιμών που αντιστοιχούν στο κάθε επίπεδο και μετά δημιουργεί ένα factor αντιστοιχώντας τα δεδομένα με τα επίπεδα αυτά. Επίσης επιτρέπει να «κοπούν» τα δεδομένα σε συγκεκριμένο αριθμό υποσυνόλων βάσει των τιμών τους:

Δοκιμάστε:	**Σχόλιο**
`a<-c(1,8,4,6,10,1)`	Μερικά αριθμητικά δεδομένα στο διάνυσμα a.
`f1<-cut(a,c(0,5,7,10))`	Δημιουργία στο f1 ενός factor που χωρίζει το a σε επίπεδα [0,5), (5,7), (7,10].
`f2<-cut(a,2)`	Δημιουργία στο f1 ενός factor που χωρίζει το a σε 2 μέρη βάσει των τιμών του.

Αν εκτελεστούν οι εντολές του παραπάνω παραδείγματος και ακολούθως ανακληθεί το factor f1, εμφανίζεται η κατηγοριοποίηση του a. Παρατηρήστε πως στο $1^{ο}$ στοιχείο του a (με τιμή 1) έχει αντιστοιχηθεί το επίπεδο (0,5], στο $2^{ο}$ στοιχείο του a (με τιμή 7) έχει αντιστοιχηθεί το επίπεδο (7,10) κλπ.:

```
> f1
[1] (0,5]  (7,10] (0,5]  (5,7]  (7,10] (0,5]
Levels: (0,5] (5,7] (7,10]
```

Αν ανακληθεί το f2 (όπου ζητήθηκε διαχωρισμός σε 2 υποσύνολα) το ενδιάμεσο όριο είναι η διάμεσος, με τιμή 5.5. Άλλος τρόπος δημιουργίας factor είναι η μετατροπή των δεδομένων ενός άλλου τύπου (συνήθως vector) μέσω της συνάρτησης **as.factor**:

---

[190] βλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων.

[191] Υπενθυμίζεται ότι εσωτερικά οι τιμές που αντιστοιχούν στα επίπεδα διατηρούνται ως ένα διάνυσμα ακέραιων αριθμών, εδώ 1 = εκπτωτικό, 2 = κανονικό και 3 = παιδικό.

Δοκιμάστε:	**Σχόλιο**
a<-c(1,1,4,5,1,4)	Ένα vector a με αριθμητικά δεδομένα.
f1<-as.factor(a)	Μετατροπή του a σε factor (επιτρεπτά level: "1", "4" και "5").
f2<-as.factor(a==4)	Μετατροπή του logical vector a=4 σε factor (level: "TRUE" και "FALSE").

Μπορείτε να ελέγξετε αν ένα αντικείμενο έχει τέτοιους περιορισμούς επιτρεπτών τιμών, δηλαδή συγκεκριμένων level, με τη συνάρτηση **is.factor**. Επιπροσθέτως, η συνάρτηση **levels** δίνει πρόσβαση στο διάνυσμα που περιέχει τις διαφορετικές επιτρεπόμενες τιμές ενός factor. Τα levels είναι πάντα λεκτικά (τύπου character) αλλιώς μετατρέπονται στον τύπο αυτό. Τα επίπεδα μπορούν να οριστούν εξαρχής ακόμα και σε ένα αντικείμενο χωρίς δεδομένα. Το επόμενο παράδειγμα εκμεταλλεύεται αυτές τις δυνατότητες:

Δοκιμάστε:	**Σχόλιο**
x<-factor(levels=c(11,21,31))	Factor με 3 επιτρεπτά level το 11, 21 και 31.
x[1]<-31	Ανάθεση κατηγορίας 31 στο $1^{ο}$ στοιχείο του x.
x[2]<-11	Ανάθεση κατηγορίας 11 στο $2^{ο}$ στοιχείο του x.
levels(x)<-c(levels(x),41)	Προσθήκη ενός ακόμα level (του 41).
levels(x)	Επιτρεπτά επίπεδα είναι τα λεκτικά c("11","21","31","41").
x+10	Ακόμα και αν μοιάζουν αριθμοί, αριθμητική με levels επιστρέφει NA.
x[1]<x[2]	Επίσης, τα ονομαστικά levels δεν έχουν «μέγεθος», επιστρέφει NA.
levels(x)<-c("a","b","c","d")	Αντικατάσταση των επιπέδων με ίδιο αριθμό άλλων (επιτρέπεται).
table(x)	Πόσα είναι σε κάθε κατηγορία; (1 a, 0 b, 1 c και 0 d).

Όπως φαίνεται στο παραπάνω παράδειγμα, δύο τιμές ονομαστικής κατηγορίας δεν μπορούν να συγκριθούν μεταξύ τους. Είναι μη-διατάξιμες κατηγορικές μεταβλητές και απλά συμβολίζουν τη συμμετοχή σε κάποια κατηγορία.

Υπάρχουν όμως περιπτώσεις που οι επιμέρους κατηγορίες είναι συγκρίσιμες, π.χ. εκπροσωπούν μια τάξη μεγέθους. Τέτοιες κατηγορικές μεταβλητές ονομάζονται διατάξιμες ή τακτικές ή τακτικοί παράγοντες και είναι χρήσιμες όταν υπάρχει μια ιεραρχία στις κατηγορίες. Κλασικό παράδειγμα είναι τα μεγέθη σε προϊόντα ρουχισμού: κατηγοριοποιούνται σε S (Small), M (Medium), L (Large) κλπ. Στο ίδιο μοντέλο ρούχου το S είναι «μικρότερο» από το L. Πόσο ακριβώς μικρότερο δεν είναι απαραίτητα γνωστό, αλλά κάθε κατηγορία θεωρείται «μικρότερη» από την επόμενη άρα έχουν νόημα οι σχετικές συγκρίσεις. Κατά τη δημιουργία του factor αν η σειρά που δίνεται εκπροσωπεί έναν τέτοιο τακτικό παράγοντα, πρέπει να οριστεί η παράμετρος ordered σε TRUE. Ο τύπος αντικειμένου που θα δημιουργηθεί είναι μια παραλλαγή του factor που ονομάζεται ordered. Η συνάρτηση is.ordered ελέγχει αν ένα factor όντως περιέχει τακτικές τιμές (είναι τακτικός παράγοντας):

Δοκιμάστε:	**Σχόλιο**
x<-factor(levels=c("S","M","L"),ordered=T)	Τακτικός παράγων με 3 επιτρεπτά level S, M, L.
x[1]<-"L"	Ανάθεση κατηγορίας L στο $1^{ο}$ στοιχείο του x.
x[2]<-"S"	Ανάθεση κατηγορίας S στο $2^{ο}$ στοιχείο του x.
x[1]<x[2]	Είναι το L μικρότερο του S; Επιστρέφει FALSE.
X	Παρατηρήστε ότι στα levels αναφέρει S < X < L.
is.factor(x)	TRUE, το x είναι όντως factor (παράγων).
is.ordered(x)	TRUE, το x είναι επίσης ordered (τακτικός).

Για τη δημιουργία του x παραπάνω θα μπορούσε να χρησιμοποιηθεί απευθείας η συνάρτηση ordered που δημιουργεί έναν τακτικό παράγοντα και είναι κατά τα άλλα παρεμφερής με τη factor. Τέλος, χρήσιμη σχετική συνάρτηση είναι η **gl** (Generate (factor) Levels) που δημιουργεί κατηγορικά δεδομένα (ordered ή όχι) βάσει των παραμέτρων της:

Δοκιμάστε:	**Σχόλιο**
gl(2,3)	Factor με 2 levels, 3 επαναλήψεις ανά level.
gl(2,3,12)	2 levels, 3 επαναλήψεις ανά level, 12 συνολικά.
gl(2,3,labels=c("S","L"),ordered=T)	Ordered, 2 levels ("S","L"), 3 επαναλήψεις ανά level.

Συνοψίζοντας, ο τύπος παράγοντας (factor) χρησιμοποιείται σε δεδομένα τα οποία προσδιορίζουν συμμετοχή σε κάποια κατηγορία. Αυτές οι κατηγορίες μπορεί να είναι ιεραρχημένες (ordered) ή όχι. Τέτοια δεδομένα έχουν σημαντικό ρόλο σε πολλές συναρτήσεις μοντελοποίησης όπως οι **lm**, **glm**, **aov**, σε μεθόδους μηχανικής μάθησης κλπ. Επίσης, οι συναρτήσεις δημιουργίας γραφημάτων μπορούν να εκμεταλλευτούν τέτοια δεδομένα ώστε να απεικονίσουν δεδομένα με τρόπο που να αποτυπώνει και την κατηγορία στην οποία αυτά ανήκουν. Μία όμως συνάρτηση με ενδιαφέρον για τον προγραμματιστή και η οποία χρησιμοποιεί εσωτερικά τον τύπο factor είναι η tapply που παρουσιάζεται παρακάτω.

4.1.3 Πίνακες (matrix και array)

Ένα vector στην R θεωρείται πως δεν έχει διαστάσεις (μόνο μήκος[192]). Για δομές τύπου πίνακα (2 ή περισσοτέρων διαστάσεων) χρησιμοποιούνται οι συναρτήσεις matrix και array οι οποίες επεκτείνουν τη δομή vector ώστε να μπορούν να οριστούν διαστάσεις για το αντικείμενο. Τι εννοούμε διαστάσεις όμως; Ένα παράδειγμα: αν κοιτάξετε κάποιον μήνα σε ένα μηνιαίο ημερολόγιο τοίχου, για κάθε μήνα υπάρχουν 5 γραμμές (μια για κάθε εβδομάδα) και 7 στήλες (μία για κάθε ημέρα της εβδομάδας). Αυτό που βλέπετε έχει 2 διαστάσεις: 6 γραμμές και 7 στήλες. Τις χρησιμοποιείτε όταν θέλετε να βρείτε την ημερομηνία π.χ. της 5ης ημέρας, της δεύτερης εβδομάδας, οπότε ανατρέχετε στη 2η γραμμή και 5η στήλη. Αν οι μήνες μας είχαν σταθερά 42 μέρες οπότε είχαμε και σχήμα ορθογώνιου παραλληλόγραμμου, θα ήταν ένα καλό παράδειγμα δισδιάστατου πίνακα. Συνήθως ένα τέτοιο ημερολόγιό έχει 12 σελίδες, μία σελίδα για κάθε μήνα του χρόνου, οπότε συνολικά είναι ένα αντικείμενο τριών διαστάσεων: προσθέτοντας τη διάσταση «σελίδα» στα προηγούμενα, η πληροφορία της ημερομηνίας για κάθε ημέρα του χρόνου καταγράφεται σε συγκεκριμένο μήνα (σελίδα), εβδομάδα (γραμμή στη σελίδα αυτή) και ημέρα της εβδομάδας (στήλη στη σελίδα αυτή).

4.1.3.1 Ο τύπος matrix (πίνακας 2 διαστάσεων)

Τα αντικείμενα που επιστρέφει η συνάρτηση **matrix** αντιπροσωπεύουν τη δομή ενός δισδιάστατου (2-d) πίνακα[193] που αποτελείται από δεδομένα του ίδιου τύπου τακτοποιημένα σε γραμμές και στήλες. Η πρόσβαση στα επιμέρους στοιχεία του πίνακα γίνεται με αγκύλες [][194], που στα matrix επιτρέπει δύο δείκτες (τον αριθμό γραμμής και στήλης) για τον προσδιορισμό της θέσης του στοιχείου που μας ενδιαφέρει:

Δοκιμάστε:	**Σχόλιο**
`x<-matrix(0,nrow=4,ncol=3)`	Ανάθεση στο x 2-d πίνακα με 4 γραμμές και 3 στήλες στοιχείων, όλα 0.
`x[1,2]<-50`	Το στοιχείο στην 1η γραμμή και 2η στήλη του x να πάρει την τιμή 50.
`x[1,2]+10`	Η τιμή του παραπάνω στοιχείου συν 10 (επιστρέφει 60).

Αν ακολούθως ανακληθεί το x στο Console, εμφανίζεται το παρακάτω:

```
> x
     [,1] [,2] [,3]
[1,]    0   50    0
[2,]    0    0    0
[3,]    0    0    0
[4,]    0    0    0
```

Το [1,] συμβολίζει όλες τις στήλες της γραμμής 1 (δηλαδή ολόκληρη τη γραμμή 1), ενώ το [,3] όλες τις γραμμές της στήλης 3 (δηλαδή ολόκληρη τη στήλη 3). Αυτή η σύνταξη χρησιμοποιείται και στον προσδιορισμό θέσεων σε εντολές. Συνεχίζοντας από το προηγούμενο:

---

[192] Προφανώς η ύπαρξη μήκους συνεπάγεται μία διάσταση, ένα μονοδιάστατο αντικείμενο. Αλλά για την R, το αντικείμενο θεωρείται χωρίς διάσταση. Η προσθήκη της ιδιότητας της διάστασης (dim) γίνεται μέσω συναρτήσεων όπως η dim ή κατά τη δημιουργία κάποιων τύπων αντικειμένων όπως π.χ. το matrix που αναφέρεται παρακάτω. Βλ. και §4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων.

[193] Η γενίκευση του vector ονομάζεται one-dimensional ή first-order tensor, του matrix ονομάζεται two-dimensional ή second-order tensor, κλπ.

[194] Όπως αναφέρθηκε στα vectors, ο τελεστής [] είναι μια συνάρτηση, δέχεται μάλιστα και επιπρόσθετες παραμέτρους, όπως την παράμετρο drop για μείωση των διαστάσεων όταν εφαρμόζεται σε πίνακες, βλ. help("[").

Δοκιμάστε:	Σχόλιο
`x[,1]=1:4`	Τα στοιχεία της 1ης στήλης να πάρουν τιμές 1, 2, 3 και 4 αντίστοιχα.
`x[3,]=c(-3,10,20)`	Τα στοιχεία της 3ης γραμμής να πάρουν τιμές -3, 10 και 20 αντίστοιχα.
`x[,1]+10`	Η 1η στήλη συν 10, επιστρέφει c(11, 12, 7, 14).

Αν ανακληθεί το x στο Console, εμφανίζεται το παρακάτω:

```
> x
     [,1] [,2] [,3]
[1,]    1   50    0
[2,]    2    0    0
[3,]   -3   10   20
[4,]    4    0    0
```

Όταν επιλέγονται ή επιστρέφονται τιμές στοιχείων ενός matrix μέσω του τελεστή [], η R επιστρέφει ένα κατάλληλου τύπου αντικείμενο, συχνά μικρότερης διάστασης και όχι απαραίτητα matrix). Η παράμετρος drop του τελεστή [] προσδιορίζει αν αυτό είναι θεμιτό. Με drop=F ως παράμετρο του [] επιστρέφεται πάντα αντικείμενο ίδιου αριθμού διαστάσεων με το αρχικό (δηλαδή εδώ matrix):

Δοκιμάστε:	Σχόλιο
`x[1,]`	Η 1η γραμμή του x, επιστρέφει διάνυσμα c(1, 50, 0).
`x[1,,drop=F]`	Η 1η γραμμή του x, επιστρέφει matrix 1x3 με στοιχεία 1, 50, 0 στην 1η γραμμή.
`x[3,2]`	Το στοιχείο στη θέση 3,2, επιστρέφει τον αριθμό 10.
`x[3,2,drop=F]`	Το στοιχείο στη θέση 3,2, επιστρέφει matrix 1x1 με μοναδικό στοιχείο τον αριθμό 10.

Ένα matrix μπορεί επίσης να περιέχει δείκτες που προσδιορίζουν θέσεις στοιχείων σε κάποιο άλλο matrix ή array. Ένα τέτοιο αντικείμενο ονομάζεται index matrix. Κάθε γραμμή του προσδιορίζει μια θέση, άρα πρέπει να υπάρχουν στήλες όσες και οι διαστάσεις του αντικειμένου πάνω στο οποίο θα χρησιμοποιηθεί:

Δοκιμάστε:	Σχόλιο
`p<-matrix(nrow=4,ncol=2)`	Ένα matrix που θα περιέχει 4 δείκτες για αντικείμενο 2 διαστάσεων.
`p[1,]=c(2,1)`	Ο 1ος δείκτης είναι η θέση 2,1.
`p[2,]=c(3,2)`	Ο 2ος δείκτης είναι η θέση 3,2.
`p[3,]=c(1,2)`	Ο 3ος δείκτης είναι η θέση 1,2.
`p[4,]=c(4,3)`	Ο 4ος δείκτης είναι η θέση 4,3.
`x[p]`	Τα στοιχεία του x στις παραπάνω θέσεις, επιστρέφει c(2, 10, 50, 0).

Όπως και στα vector, μπορεί να γίνει προσδιορισμός στοιχείων με χρήση logical vector, δηλαδή ουσιαστικά προσδιορισμός με κριτήρια. Η σύνταξη είναι παρόμοια με αυτή που αναφέρθηκε στα vector (βλ.§4.1.1 Ο τύπος vector (διάνυσμα)), προσαρμοσμένη στις περισσότερες διαστάσεις που έχει ο πίνακας.

Δοκιμάστε:	Σχόλιο
`x[x[,2]>0,]`	Οι γραμμές που στη 2η στήλη τους είναι > 0, επιστρέφει 1η ([1,]) και 3η ([3,]) γραμμή.
`x[x[,2]>0,1]`	Το 1ο στοιχείο γραμμών που στη 2η στήλη είναι > 0, δηλαδή τα στοιχεία [1,1] & [3,1].
`x[1,x[1,]<50]<-5`	Τα στοιχεία της γραμμής 1 που είναι <50 (τα [1,1] και [1,3]) να πάρουν την τιμή 5.

Ο τρόπος εκτέλεσης των παραπάνω παραδειγμάτων είναι εμφανής αν εκτελεστεί το κριτήριο μόνο του (αναπαράγοντας αυτό που θα κάνει και η R εσωτερικά κατά την εκτέλεση της εντολής). Στην τελευταία παραπάνω εντολή το κριτήριο είναι x[1,]<50. Το κριτήριο αυτό ελέγχει αν κάθε στοιχείο στην 1η γραμμή του x είναι μικρότερο του 50, μια σύγκριση που επιστρέφει c(TRUE, FALSE, TRUE). Οπότε, ο προσδιορισμός γίνεται με το x[1, c(TRUE, FALSE, TRUE)], άρα «επιλέγονται» τα στοιχεία στην 1η και 3η στήλη της 1ης γραμμής. Παρόμοια λειτουργούν και τα υπόλοιπα παραδείγματα. Αυτή η σύνταξη εντολών για προσδιορισμό στοιχείων του πίνακα βάσει κριτήριων, εξαλείφει σε πολλές περιπτώσεις την ανάγκη χρήσης βρόχων (βλ.§3.2.2.3 Επαναλήψεις (βρόχοι ή loop)) για τέτοιου τύπου επεξεργασία. Παρόμοια σύνταξη μπορεί να εφαρμοστεί και σε πολυδιάστατα array καθώς και σε άλλους τύπους με σχήμα πίνακα (όπως τα data.frame, βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων)). Αν τώρα ανακληθεί το x στο Console, έχουμε το παρακάτω:

```
> x
     [,1] [,2] [,3]
[1,]    5   50    5
[2,]    2    0    0
[3,]   -3   10   20
[4,]    4    0    0
```

Για να εμφανιστούν μόνο οι αρχικές ή οι τελικές γραμμές του πίνακα, μπορεί να χρησιμοποιηθούν οι συναρτήσεις **head** και **tail**. Οι συναρτήσεις αυτές είναι χρήσιμες σε αντικείμενα που απαιτούν πολλές γραμμές για να γίνει εμφάνιση τους ως κείμενο, περιορίζοντας την έξοδο στις πρώτες ή τελευταίες γραμμές αντίστοιχα. Π.χ. η εντολή head(x,2) θα εμφανίσει μόνο τις 2 πρώτες γραμμές του x ενώ αντίστοιχα η tail(x,2) τις 2 τελευταίες.

Όσα αναφέρθηκαν για το mode των vector (βλ.§4.1.1 Ο τύπος vector (διάνυσμα)) ισχύουν κατ' επέκταση στα αντικείμενα matrix και array. Άρα αν π.χ. αποθηκευτεί ένα αντικείμενο character σε κάποιο στοιχείο του παραπάνω x (που τώρα βρίσκεται σε numeric mode), το x θα αλλάξει σε character mode. Οι ίδιοι κανόνες (και ιεραρχία) μετατροπής ισχύουν για τα matrix και array (και για άλλους atomic τύπους αντικειμένων).

Ο λόγος είναι πως ο τύπος matrix (και γενικότερα τα array) βασίζεται (επεκτείνει) στον τύπο vector[195] προσθέτοντας την ιδιότητα dim που περιγράφει τη διάσταση[196]. Τα ίδια τα δεδομένα διατηρούνται σε ένα atomic vector. Έτσι τα matrix (και γενικότερα τα array) επιτρέπουν και τη χρήση όλων των συναρτήσεων που υποστηρίζει ο βασικός τους τύπος, το vector. Η διάταξη των δεδομένων στα matrix (δηλαδή τα δισδιάστατα array) είναι ως προς τη 2η διάσταση (δηλαδή τη στήλη, άρα τα πρώτα στοιχεία του διανύσματος είναι τα στοιχεία της 1ης στήλης, τα επόμενα της 2ης στήλης κλπ.). Στα στοιχεία αυτά μπορεί να γίνει πρόσβαση με σύνταξη εντολών όπως αναφέρθηκαν στα vector, π.χ. με έναν μόνο δείκτη ως προσδιοριστή θέσης στον τελεστή []. Συνεχίζοντας από το προηγούμενο παράδειγμα:

Δοκιμάστε:	**Σχόλιο**
`length(x)`	Ο αριθμός στοιχείων του x, επιστρέφει 12.
`mode(x)`	Το mode του x, επιστρέφει "numeric".
`x[1:12]`	Τα στοιχεία στις θέσεις 1 έως 12, δηλαδή c(5, 2, -3, 4, 50, 0, 10, 0, 5, 0, 20, 0).
`x[5]<-40`	Αλλάζει το 5ο στοιχείο του x σε 40 (από την προηγούμενή του τιμή 50).
`x[x>10]`	Τα στοιχεία του x που είναι μεγαλύτερα του 10, επιστρέφει c(40,20).

Η συνάρτηση **which** που έχουμε ήδη χρησιμοποιήσει σε vectors (βλ.§2.4.2 Δύο παραδείγματα χρήσης vector και η σημασία της ειδικής τιμής NA) είναι και εδώ χρήσιμη στον προσδιορισμό στοιχείων του αντικειμένου βάσει κάποιου κριτηρίου, με δύο διαφορετικούς τρόπους:

Δοκιμάστε:	**Σχόλιο**
`which(x>10)`	Ποιες θέσεις έχουν τιμή > 10; Επιστρέφει c(5,11), 5η & 11η στο vector δεδομένων.
`which(x>10,arr.ind=T)`	Ποιες θέσεις έχουν τιμή > 10; Επιστρέφει index matrix (βλ. παραπάνω).

Τα matrix και array απλά προσθέτουν διάσταση (και υποστηρίζουν σχετικές λειτουργίες) στο vector των δεδομένων τους. Οι συναρτήσεις **nrow** και **ncol** επιστρέφουν τον αριθμό γραμμών και στηλών αντίστοιχα, ενώ πρόσβαση στις διαστάσεις του αντικειμένου δίνει η συνάρτηση **dim**:

Δοκιμάστε:	**Σχόλιο**
`nrow(x)`	Επιστρέφει τον αριθμό γραμμών στο x, δηλαδή 4.
`ncol(x)`	Επιστρέφει τον αριθμό στηλών στο x, δηλαδή 3.
`dim(x)`	Οι τρέχουσες διαστάσεις του x, επιστρέφει c(4, 3) άρα 4 γραμμές, 3 στήλες.
`dim(x)<-c(2,6)`	Αλλαγή των διαστάσεων του x σε 2 γραμμές, 6 στήλες.

[195] Αυτό μπορεί να επιβεβαιωθεί εκτελώντας τη σχετική εντολή, δηλαδή extends("matrix", "vector"), η οποία επιστρέφει TRUE.

[196] Πρόσβαση στις ιδιότητες αντικειμένων (οποιουδήποτε τύπου) δίνει η συνάρτηση attributes. Αν εκτελεστεί με παράμετρο ένα αντικείμενο τύπου matrix θα εμφανίσει και την ιδιότητα διάστασης με όνομα dim, βλ.§4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων).

Στην αλλαγή των διαστάσεων μέσω dim πρέπει οι νέες διαστάσεις να είναι συμβατές[197] με τον αριθμό των στοιχείων που υπάρχουν ήδη στο matrix. Αν τώρα ανακληθεί το x στο Console, έχουμε το παρακάτω:

```
> x
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    5   -3   50   10    5   20
[2,]    2    4    0    0    0    0
```

Εξετάζοντας το αποτέλεσμα, επιβεβαιώνεται πως στη νέα διάσταση του πίνακα τα δεδομένα του διανύσματος χωρίστηκαν πάλι πρώτα ως προς τη 2η διάσταση (στήλες). Έτσι τα πρώτα 2 στοιχεία (καθώς έχουμε 2 γραμμές) του διανύσματος δεδομένων (βλ. εντολή x[1:12] παραπάνω) αφορούν την 1η στήλη, τα επόμενα 2 η 2η στήλη κ.ο.κ.

Το πρόβλημα που τίθεται τώρα είναι πως επεκτείνεται ένα υπάρχον αντικείμενο matrix. Όπως προαναφέρθηκε, οι διαστάσεις μπορούν να αλλάξουν αλλά ο αριθμός των στοιχείων παραμένει σταθερός. Επιπροσθέτως, κάθε προσπάθεια πρόσβασης εκτός των ορίων του πίνακα απαγορεύεται και εγείρει μήνυμα λάθους: δοκιμάστε π.χ. το x[2,7] και δεν θα επιτραπεί, καθώς δεν υπάρχει στήλη 7 στο x. Το ίδιο θα συμβεί αν δοκιμάσετε ανάθεση, π.χ. το x[7,1]<-5 καθώς στο x δεν υπάρχει γραμμή 7. Η προσθήκη γραμμών ή στηλών σε αντικείμενα matrix[198] μπορεί να γίνει μέσω των συναρτήσεων **rbind** και **cbind,** αντίστοιχα. Οι συναρτήσεις παίρνουν ως παράμετρο άλλα αντικείμενα και τα ενώνουν (με τη σειρά που δόθηκαν) σε ένα κατάλληλο αντικείμενο σε σχήμα πίνακα[199]. Για παράδειγμα, με δεδομένο το παραπάνω x, η εντολή rbind(200,100,x) θα επιστρέψει πίνακα 4 γραμμών x 6 στηλών στον οποίο όλα τα στοιχεία της 1ης γραμμής θα έχουν την τιμή 200, τα στοιχεία της 2ης γραμμής την τιμή 100 ενώ θα ακολουθούν ως υπόλοιπες γραμμές οι αντίστοιχες τιμές από το υπάρχον x, άρα θα επιστρέψει:

```
> rbind(200,100,x)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  200  200  200  200  200  200
[2,]  100  100  100  100  100  100
[3,]    5   -3   50   10    5   20
[4,]    2    4    0    0    0    0
```

Μερικά ακόμα παραδείγματα:

Δοκιμάστε:	**Σχόλιο**
`rbind(c(0,1,2,5,1,1),1:6)`	Πίνακας 2 γραμμών, μια με τις δοθείσες τιμές και μια με το 1 έως 6.
`rbind(x,NA)`	Πίνακας με το x και μια ακόμα γραμμή όπου όλα τα στοιχεία είναι NA.
`rbind(x,NA)`	Πίνακας με το x και μια ακόμα γραμμή όπου όλα τα στοιχεία είναι NA.
`rbind(x,c(0,1,2,5,1,1))`	Πίνακας με το x και μια ακόμα γραμμή με τις δοθείσες τιμές .
`rbind(x,x)`	Πίνακας με το x ακολουθούμενο (ως γραμμές) από ένα 2ο αντίγραφο του x.
`cbind(100,x)`	Πίνακας, τιμές 100 στην 1η στήλη, οι υπόλοιπες στήλες αντιγράφουν το x.
`cbind(x[,1:3],100,x[,4:6])`	Πίνακας ίδιος με το x, συν μια νέα στήλη τιμών 100 ανάμεσα στη 3η και 4η.
`x<-cbind(x,NA)`	Αντικατάσταση του x με το αποτέλεσμα cbind (προσθήκη στήλης από NA).

Η τελευταία εντολή στο παραπάνω παράδειγμα έχει πρακτικά το αποτέλεσμα να γίνει επέκταση του x με μια νέα στήλη. Θα μπορούσε να γίνει αντίστοιχα προσθήκη περισσότερων στηλών ή γραμμών με χρήση της rbind. Όμως, προσεγγίσεις όπως οι παραπάνω για επέκταση πινάκων καλό είναι να γίνονται με σύνεση, ειδικά όταν εμπλέκονται πίνακες μεγάλου μεγέθους. Ισχύουν και εδώ όσα αναφέρθηκαν σχετικά με την τροποποίηση ή επέκταση μεγάλων διανυσμάτων και τον τρόπο που μπορεί να προκαλέσουν καθυστερήσεις στην επεξεργασία (βλ.§4.1.1 Ο τύπος vector (διάνυσμα)) ενώ για κάποιες επιλογές επεξεργασίας μεγάλων πινάκων βλ. τους τύπους αντικειμένων Matrix (§4.3.2 Ο τύπος Matrix (αραιοί και πυκνοί πίνακες)) και data.table (§4.3.1 Οι τύποι tibble και data.table).

Οι συναρτήσεις **is.matrix** και **is.array** ελέγχουν αν ένα αντικείμενο είναι των αντίστοιχων τύπων. Παρατηρήστε πως το matrix x που ορίσαμε παραπάνω δεν θεωρείται πλέον απλό vector (ακόμα και αν είχε οριστεί στο dim μόνο μία γραμμή ή μόνο μία στήλη). Η μετατροπή του matrix (ή array) σε απλό vector μπορεί

[197] Στο παράδειγμα το x έχει 12 στοιχεία, με παλαιά διάταξη 4x3 που μπορεί όντως να αλλάξει σε 2x6.

[198] Οι συναρτήσεις rbind και cbind λειτουργούν και σε άλλους τύπους αντικειμένων όπως vector και data,frame καθώς και μπορούν να δεχτούν τέτοιους τύπους ως παραμέτρους τους.

[199] Αν κατά τη συνένωση δημιουργείται ένα διαφορετικού mode αντικείμενο, το mode του νέου αντικειμένου προκύπτει από τους γνωστούς κανόνες (και ιεραρχία) για coercion.

να γίνει με τη συνάρτηση as.vector ή καταργώντας την πληροφορία της διάστασης (π.χ. για το παραπάνω x με την εντολή dim(x)<-NULL). Όμως ένα αντικείμενο matrix είναι array, καθώς τα matrix είναι απλά η δισδιάστατη περίπτωση του τύπου array (που παρουσιάζεται στην αμέσως επόμενη ενότητα):

Δοκιμάστε:	**Σχόλιο**
`is.vector(x)`	Επιστρέφει FALSE, το matrix x δεν θεωρείται πλέον απλό vector.
`is.matrix(x)`	Επιστρέφει TRUE, το x είναι matrix.
`is.array(x)`	Επιστρέφει TRUE, επειδή τα matrix είναι και array.
`class(x)`	Επιστρέφει c("matrix", "array")[200].

Παρακάτω δημιουργούνται δυο αντικείμενα m1 και m2 τύπου matrix για τα παραδείγματα που ακολουθούν:

Δοκιμάστε:	**Σχόλιο**
`m1<-matrix(1:12,ncol=3,byrow=T)`	Οι ακέραιοι 1 έως 12, ως γραμμές σε πίνακα 3 στηλών (4x3).
`m2<-matrix(0,nrow=3,ncol=2)`	Στο m2, πίνακας 3x2, όλα τα στοιχεία 0.
`m3<-matrix(2,ncol=3,nrow=4)`	Στο m3, πίνακας 4x3, όλα τα στοιχεία 2.
`m2[,1]<-10`	Όλα τα στοιχεία της στήλης 1 του m2 να πάρουν την τιμή 10.
`m2[,2]<-20`	Όλα τα στοιχεία της στήλης 2 του m2 να πάρουν την τιμή 20.

Υπάρχει πληθώρα συναρτήσεων που εκμεταλλεύονται τη μορφή πίνακα που έχουν τα αντικείμενα αυτά. Για χειρισμό ονομάτων οι **rownames** (ονόματα γραμμών), **colnames** (ονόματα στήλών) και **dimnames** (λίστα ονομάτων ανά διάσταση).

Δοκιμάστε:	**Σχόλιο**
`rownames(m1)<-c("Α","Β","Γ","Δ")`	Ονόματα για τις γραμμές του m1.
`colnames(m1)<-c("Ε","Ζ","Η")`	Ονόματα για τις στήλες του m1.
`n<-list(c("α","β","γ"),c("δ","ε"))`	Στο n μια λίστα[201] δύο διανυσμάτων ονομάτων.
`dimnames(m2)<-n`	Ορίζει τα ονόματα γραμμών και στηλών του m2 από το n.

Αν ανακληθούν τα m1 και m2 στο Console, έχουμε:

```
> m1
   Ε  Ζ  Η
Α  1  2  3
Β  4  5  6
Γ  7  8  9
Δ 10 11 12

> m2
   δ  ε
α 10 20
β 10 20
γ 10 20
```

Αν το mode του αντικειμένου είναι αριθμητικό υποστηρίζονται αριθμητικές συναρτήσεις. Κάποιες άλλες χρήσιμες συναρτήσεις για βασικές πράξεις είναι οι **rowSums**, **colSums**, **rowMeans. colMeans**, αλλά και οι ήδη γνωστές από τα vector συναρτήσεις όπως οι sum, mean, summary κλπ.:

Δοκιμάστε:	**Σχόλιο**
`rowSums(m1)`	Αθροίσματα ανά γραμμή του m1, επιστέφει c(6, 15, 24, 33).
`colMeans(m1)`	Αριθμητικοί μέσοι όροι ανά στήλη, επιστέφει c(5.5, 6.5, 7.5).

[200] Ένα αντικείμενο μπορεί να ανήκει σε παραπάνω από μία κλάση, βλ.§6.3 Κλάσεις S3.
[201] Για τα αντικείμενα του τύπου list, βλ.§4.2.1 Ο τύπος list (λίστα).

Η συνάρτηση **t** κάνει αντιμετάθεση (transpose) ενός matrix[202] επιστρέφοντας ένα matrix όπου οι γραμμές μετατρέπονται σε στήλες, άρα το m2 ' γράφεται ως t(m2) και επιστρέφει:

```
> t(m2)
   α  β  γ
δ 10 10 10
ε 20 20 20
```

Η συνάρτηση **diag** αφορά δημιουργία πινάκων με τιμές μόνο στη διαγώνιο. Επίσης δίνει πρόσβαση στα στοιχεία που είναι στη διαγώνιο κάποιου υπάρχοντος matrix (εδώ τα m2[1,1] και m2[2,2]):

```
> diag(m2)
[1] 10 20
```

Προφανώς, αν το mode του αντικειμένου είναι αριθμητικό υποστηρίζονται και οι αριθμητικοί τελεστές. Πράξεις matrix (ή array) με αριθμούς ή με διανύσματα αριθμών επιτρέπονται και γίνονται ανάμεσα στα επιμέρους στοιχεία (element-wise). Στις πράξεις (και άλλες συναρτήσεις) με διανύσματα γίνεται ανακύκλωση τιμών αν χρειάζεται και είναι εφικτό (βλ.§2.4.1 Οι βασικές εντολές των διανυσμάτων). Όμως σε πράξεις μεταξύ δύο αντικειμένων matrix (ή array) οι διαστάσεις τους πρέπει να συμφωνούν και δεν γίνεται ανακύκλωση:

Δοκιμάστε:	**Σχόλιο**
`m1+100`	Επιστρέφει matrix (4x3) που περιέχει τα στοιχεία του m1 συν 100.
`m1[,1]+100`	Κάθε στοιχείο της 1ης στήλης του m1 συν 100, επιστρέφει c(101,104,107,110).
`m1[,1]+m2[2,]`	1ης στήλη του m1 συν 2η γραμμή του m2, επιστρέφει[203] c(11,14,17,30).
`m1[,"E"]+m2["β",]`	Ίδιο με το παραπάνω, αλλά με χρήση ονομάτων γραμμής και στήλης.
`m1+m2`	Λάθος, οι διαστάσεις δεν συμφωνούν.
`m1+m3`	Επιστρέφει matrix 4x3 με τις προσθέσεις των επιμέρους στοιχείων (element-wise).

Ακολουθούν μερικά ακόμα παραδείγματα χρήσης αριθμητικών τελεστών. Ο πίνακας που αποτελείται από τις γραμμές 1 και 2 του m1 συν την αντιμετάθεση του m2 (δηλαδή η εντολή: m1[1:2,]+t(m2)) επιστρέφει πίνακα 2x3:

```
> m1[1:2,]+t(m2)
   E  Z  H
A 11 12 13
B 24 25 26
```

Ο πολλαπλασιασμός πινάκων ή πολλαπλασιασμός μήτρας (matrix multiplication) γράφεται με τον τελεστή **%*%**. Έτσι, ο πολλαπλασιασμός πινάκων ανάμεσα σε m1 και m2 δηλαδή η εντολή: m1%*%m2) επιστρέφει πίνακα 4x2 (καθώς οι m1 και m2 είναι 4x3 και 3x2 αντίστοιχα):

```
> m1%*%m2
    δ   ε
A  60 120
B 150 300
Γ 240 480
Δ 330 660
```

Αντίστοιχα ο πολλαπλασιασμός πινάκων ανάμεσα στην 1η γραμμή του m1 και 2η στήλη του m2 (δηλαδή η εντολή: m1[1,]%*%m2[,2]) επιστρέφει πίνακα 1x1 με το εσωτερικό τους γινόμενο (dot product):

```
> m1[1,]%*%m2[,2]
     [,1]
[1,]  120
```

[202] Ή ενός data.frame. Η t εφαρμόζεται και σε αντικείμενα vector.

[203] Παρατηρήστε ότι έχει γίνει κατάλληλη ανακύκλωση των τιμών του m2[2,] και πως το επιστρεφόμενο αντικείμενο είναι vector.

Data Editor

File Edit Help

	Επιβατικά	Φορτηγά	Δίκυκλα	var4	var5	var6
1	16	2	3			
2	9	6	5			
3	0	0	0			
4	0	0	0			
5	0	0	0			
6	0	0	0			
7	0	0	0			
8	0	0	0			
9	0	0	0			
10	0	0	0			
11	0	0	0			
12	0	0	0			
13	0	0	0			
14	0	0	0			
15	0	0	0			
16	0	0	0			
17	0	0	0			
18	0	0	0			
19	0	0	0			

Εικόνα 4.1: *Ο ενσωματωμένος επεξεργαστής αντικειμένων τύπου matrix (και άλλων τύπων με σχήμα πίνακα).*

Ακολουθούν δύο παραδείγματα χρήσης matrix.

Παράδειγμα 1ο. Σε έναν σταθμό διοδίων καταγράφεται ο αριθμός των οχημάτων που περνά ανά ώρα της ημέρας (του 24ώρου). Υπάρχουν 3 κατηγορίες οχημάτων με διαφορετική χρέωση (επιβατικά, φορτηγά, δίκυκλα). Αυτό μπορεί να επιτευχθεί καταγράφοντας τον αριθμό οχημάτων σε ένα matrix 24 γραμμών (μία για κάθε ώρα της ημέρας) και 3 στηλών (μία για κάθε τύπο οχήματος). Στο παρακάτω, το matrix αυτό ονομάζεται mv:

```
mv <- matrix(0,nrow = 24, ncol = 3)
colnames(mv) <- c("Επιβατικά", "Φορτηγά", "Δίκυκλα")

# Δεδομένα για πρώτες 2 ώρες:

mv[1,1] <- 16   # 1η ώρα, 16 επιβατικά
mv[1,2] <- 2    # 1η ώρα, 2 φορτηγά
mv[1,3] <- 3    # 1η ώρα, 3 δίκυκλα
mv[2,1] <- 9    # 2η ώρα, 9 επιβατικά
mv[2,2] <- 6    # 2η ώρα, 6 φορτηγά
mv[2,3] <- 4    # 2η ώρα, 4 δίκυκλα

# Πρόσθεση ενός ακόμα δίκυκλου στον τρέχοντα αριθμό 2ης ώρας:
mv[2,3] <- mv[2,3] + 1
```

Με την εντολή **View**(mv) εμφανίζεται ο πίνακας mv σε δική του καρτέλα ενώ οι εντολές **fix**(mv) και mv<-**edit**(mv) ξεκινούν το εργαλείο επεξεργασίας τιμών (data editor), με το οποίο μπορεί να γίνει καταχώρηση ή αλλαγή των τιμών του mv επιλέγοντας θέση («κελί») με το ποντίκι και πληκτρολογώντας τη νέα τιμή για τη θέση αυτή (βλ. Εικόνα 4.1)[204]. Εφαρμόζοντας προαναφερθείσες συναρτήσεις μπορούν να εξαχθούν κάποιες βασικές πληροφορίες:

[204] Η συνάρτηση edit δίνει μεγάλη ευελιξία αλλαγών. Μια παραλλαγή της παραπάνω εντολής που θα προστάτευε από

Δοκιμάστε:	Σχόλιο
`sum(mv)`	Ο συνολικός αριθμός οχημάτων στον πίνακα.
`sum(mv[2,])`	Ο συνολικός αριθμός οχημάτων τη 2η ώρα (2η γραμμή του πίνακα).
`sum(mv[,2])`	Ο συνολικός αριθμός φορτηγών (2η στήλη του πίνακα).
`colSums(mv)[2]`	Ίδιο με το παραπάνω.
`summary(mv)`	Σε matrix αριθμών επιστρέφει βασικά περιγραφικά στατιστικά στοιχεία[205].

Ας υποθέσουμε πως το κόστος διέλευσης (διοδίων) ανά κατηγορία οχήματος είναι 1 για επιβατικά, 2 για φορτηγά, 0.5 για δίκυκλα και καταχωρήθηκε σε διάνυσμα με όνομα cv:

```
cv <- c(1, 2, 0.5)
```

Πώς μπορεί να υπολογιστεί το χρηματικό ποσό που εισπράχθηκε ανά τύπο οχήματος κάθε ώρα; Ζητούμενο εδώ είναι να πολλαπλασιαστεί κάθε γραμμή στο mv με το αντίστοιχο κόστος διέλευσης στο cv. Όμως τα matrix (όπως το mv) διατηρούν τα δεδομένα τους κατά σειρά στηλών (πρώτα τα δεδομένα της 1ης στήλης, μετά της 2ης κλπ.). Ένας πολλαπλασιασμός ανάμεσα στο matrix mv και το διάνυσμα cv θα πολλαπλασίαζε λανθασμένα τα 3 πρώτα μέλη της 1ης στήλης με το cv, μετά (ανακυκλώνοντας τις τιμές του cv) τα 3 επόμενα της ίδιας στήλης κλπ. και τα αποτελέσματα θα ήταν λανθασμένα. Μπορείτε να το δοκιμάσετε:

```
mv * cv   # Προσοχή, το εισπραχθέν ποσό θα είναι λάθος!
```

το οποίο επιστρέφει:

```
       Επιβατικά Φορτηγά Δίκυκλα
 [1,]        16       2       3
 [2,]        18      12      10
 [3,]         0       0       0
  …           …       …       …
 [24,]        0       0       0
```

Άρα πρέπει να γίνει αντιμετάθεση του matrix ώστε κάθε στήλη να έχει 3 στοιχεία (όσα και οι τύποι οχημάτων)[206] και θα μπορεί να πολλαπλασιαστεί με το ίδιου μήκους cv (που περιέχει το κόστος διέλευσης ανά αντίστοιχο τύπο οχήματος):

```
t(mv) * cv
```

το οποίο επιστρέφει τις σωστές εισπράξεις, δηλαδή:

```
            [,1]  [,2]  [,3]  [,4]  …  [,24]
Επιβατικά  16.0   9.0     0     0  …      0
Φορτηγά     4.0  12.0     0     0  …      0
Δίκυκλα     1.5   2.5     0     0  …      0
```

Προφανώς αν χρειάζεται να παραμένουν οι τύποι οχήματος σε στήλες μπορεί να ζητηθεί αντιμετάθεση του αποτελέσματος, δηλαδή t(t(mv) * cv). Υπάρχει ένα δευτερεύον νόημα στις αντιμεταθέσεις αυτές: η γενικότερη σύμβαση για τους πίνακες δεδομένων είναι πως κάθε γραμμή καταγράφει μια συγκεκριμένη περίπτωση (case), γεγονός (event) κλπ. Οι στήλες είναι ιδιότητες (χαρακτηριστικά, μεταβλητές) που σχετίζονται με την κάθε περίπτωση και την περιγράφουν. Αυτή τη σύμβαση ακολουθούν οι σχεσιακές βάσεις δεδομένων, οι πίνακες σε προγράμματα υπολογιστικών φύλλων, τα προγράμματα επεξεργασίας δεδομένων κλπ. Ίσως είναι επιρροή του τρόπου γραφής της Ελληνικής και άλλων δυτικών γλωσσών που χρησιμοποιούμε, αλλά είναι πιο φυσικό κοιτάζοντας έναν πίνακα να τον διαβάσουμε γραμμή-γραμμή. Δηλαδή, κοιτάζοντας τον πίνακα mv, να τον διαβάσουμε ως «στην περίπτωση της 1ης ώρας καταγράφηκαν 16 επιβατικά, 3 φορτηγά και 2 δίκυκλα, τη 2 ώρα καταγράφηκαν …», ενώ κοιτάζοντας την αντιμετάθεση t(mv) είναι ίσως πιο φυσικό να διαβάσουμε: «στην περίπτωση των επιβατικών, την 1η ώρα καταγράφηκαν 16, τη 2η καταγράφηκαν 9, την 3η κανένα…» κλπ. Μερικοί ακόμα υπολογισμοί σε συνέχεια των παραπάνω:

κάποια λάθη στην καταχώρηση είναι: nv[1:24,1:3] <- as.integer(edit(nv)[1:24,1:3]). Επίσης, στα Windows το εργαλείο επεξεργασίας είναι ενσωματωμένο (internal). Σε άλλα ΛΣ η συνάρτηση edit επιτρέπει τον ορισμό του λογισμικού επεξεργασίας μέσω παραμέτρου της με όνομα editor.

[205] Επιστρέφει table με βασικά στατιστικά ανά στήλη, βλ. help(summary.matrix). Η summary υποστηρίζεται και από πολλούς άλλους τύπους αντικειμένων, ενώ η μορφή εξόδου της ποικίλει ανάλογα με τον τύπο. Βλ. και §2.7 Οι συναρτήσεις-βοηθήματα: str, summary και dput.

[206] Θα μπορούσε ο πίνακας να έχει εξ αρχής οριστεί έτσι, οπότε θα αποφεύγονταν και οι αντιμεταθέσεις.

Δοκιμάστε:	**Σχόλιο**
`mv%*%cv`	Πολλαπλασιασμός πινάκων, επιστρέφει συνολικές εισπράξεις ανά ώρα.
`sum(mv%*%cv)`	Συνολικές εισπράξεις 24ωρου.
`sum(t(mv)*cv)`	Ίδιο με το παραπάνω.
`sum(apply(mv, 1, "*", cv))`	Ίδιο με το παραπάνω, βλ.§4.1.3.4 Η οικογένεια συναρτήσεων apply.

Παράδειγμα 2ο. Πέραν των μαθηματικών και της αποθήκευσης δεδομένων, στον προγραμματισμό οι πίνακες έχουν διάφορες χρήσεις. Για παράδειγμα, σε ένα matrix θα μπορούσε να καταγράφεται η τρέχουσα κατάσταση μιας πίστας παιχνιδιού (π.χ. μια σκακιέρα σε έναν πίνακα 8 x 8 με τις τιμές των στοιχείων του να συμβολίζουν τα πιόνια). Ένα τέτοιο παράδειγμα «ανορθόδοξης» χρήσης ακολουθεί, όπου ζητούμενο είναι να δημιουργηθεί ένας πίνακας ο οποίος θα λειτουργεί ως ένα «μηνιαίο ημερολόγιο» του τρέχοντος μήνα, παρόμοιο με αυτό που αναφέρθηκε στο ξεκίνημα της ενότητας αυτής (βλ.§4.1.3 Πίνακες (matrix και array)).

```
# μερος 1: Εύρεση των:
# - ημέρα που αντιστοιχεί στην 1η του μήνα (στο swday).
# - αριθμός ημερών στο μήνα (στο mdays).

library(lubridate)

cm <- month(today())       # τρέχων μήνας
cmy <- year(today())       # τρέχον έτος
# η 1η ημέρα του τρέχοντος μήνα:
cm1st <- make_date(day = 1, month = cm, year = cmy)

nm <- cm %% 12 + 1         # επόμενος μήνας
# έτος επόμενου μήνα (αν είναι Ιανουάριος, αλλάζει):
nmy <- if (nm == 1) cmy + 1 else cmy
# η 1η ημέρα του επόμενου μήνα:
nm1st <- make_date(day = 1, month = nm, year = nmy)

# Διαφορά ημερών από την 1η επόμενου και τρέχοντος:
mdays <- as.integer(nm1st - cm1st)
# Ημέρα της εβδομάδας που αντιστοιχεί στην 1η (1=>δευτέρα):
swday <- wday(cm1st, week_start = 1)
cat(paste("Ημερολόγιο ", cm, cmy, ":\n"))

# μερος 2:
# χρήση matrix για δημιουργία ημερολογίου:

calendar <- matrix(data = NA, nrow = 6, ncol = 7)
colnames(calendar)<-c("Δε","Τρ","Τε","Πε","Πα","Σα","Κυ")
rownames(calendar) <- paste(1:6, "η Εβδομάδα", sep = "")

calendar <- t(calendar)
calendar[swday:(swday + mdays - 1)] <- 1:mdays
calendar <- t(calendar)

# αν δεν υπάρχει 6η εβδομάδα, απορρίπτεται η 6 γραμμή:
if(all(is.na(calendar[6,]))) calendar <- calendar[-6,]

print(calendar, na.print = "")
```

Για λόγους πληρότητας, το μέρος 1 του παραπάνω παραδείγματος χρησιμοποιεί τις συναρτήσεις **today**, **month**, **year**, **make_date** και **wday** από το πακέτο 'lubridate' [53][207] που θα διευκολύνουν την εύρεση των παρακάτω, απαραίτητων για ένα ημερολόγιο, στοιχείων: (α) πόσες ημέρες έχει ο τρέχων μήνας (στη μεταβλητή mdays) και (β) ποια ήταν η ημέρα της εβδομάδας την 1η του μήνα (στη μεταβλητή swday, όπου επιλέχτηκε το

[207] Το πακέτο ΄lubridate' είναι διαθέσιμο στο CRAN, βλ. §1.5 Χρήση και διαχείριση πακέτων.

1 να σημαίνει Δευτέρα, 2 σημαίνει Τρίτη κλπ.). Μπορείτε να παρακάμψετε (ή να διαγράψετε εντελώς) το μέρος 1 και να ορίσετε τα στοιχεία αυτά για κάποιον τυχαίο μήνα του οποίου θα δημιουργηθεί το ημερολόγιο στο μέρος 2, εκτελώντας π.χ. mdays <- 31 αν ο μήνας έχει 31 ημέρες και swday <- 6 αν ο μήνας ξεκινά Σάββατο. Στο μέρος 2 γίνεται η δημιουργία του matrix που θα έχει ρόλο μηναίου ημερολογίου, στη μεταβλητή calendar. Είναι matrix 6 γραμμών (μία ανά εβδομάδα του μήνα) και 7 στηλών (μία ανά ημέρα εβδομάδας). Αρχικά, σε όλα τα στοιχεία δίνεται η τιμή NA. Ορίζονται κατάλληλα ονόματα σε στήλες (colnames) και γραμμές (rownames). Για τα ονόματα στηλών επιλέχτηκε ως πρώτη ημέρα η Δευτέρα για να συμφωνεί με τον τρόπο που καταγράφεται η ημέρα στη μεταβλητή swday.

Η αντιμετάθεση t(calendar) έχει μια στήλη ανά εβδομάδα του μήνα. Έχοντας τα στοιχεία σε σειρά ως προς την εβδομάδα (άρα και την ημέρα), το γέμισμα των ημερομηνιών 1,2,3… κλπ. στα στοιχεία του πίνακα μπορεί να γίνει εύκολα με πρόσβαση στον πίνακα ως διάνυσμα, ξεκινώντας από τη θέση swday του διανύσματος (π.χ. την 6η θέση αν ο μήνας ξεκινά Σάββατο) και συνεχίζοντας για τα επόμενα mdays-1 στοιχεία. Μετά ξαναγίνεται αντιμετάθεση ώστε το ημερολόγιο να πάρει πάλι τη γνωστή μορφή (μια στήλη ανά ημέρα της εβδομάδας). Τέλος, μερικοί μήνες καλύπτουν μόνο 5 εβδομάδες. Αν η 6η γραμμή περιέχει ακόμα μόνο NA σημαίνει πως δεν τοποθετήθηκε αριθμός ημερομηνίας σε αυτή, άρα η γραμμή (και η αντίστοιχη εβδομάδα) μπορεί να διαγραφεί. Στην περίπτωση αυτή το calendar αντικαθίσταται από το calendar πλην την 6η γραμμή του (δηλαδή με το calendar[-6,], θα μπορούσε να γραφεί και ως calendar[1:5,]). Το αποτέλεσμα που θα εμφανίσει η συνάρτηση print (στο τέλος του παραδείγματος) για mdays=31 και swday=6 είναι:

```
              Δε Τρ Τε Πε Πα Σα Κυ
1η Εβδομάδα                  1  2
2η Εβδομάδα   3  4  5  6  7  8  9
3η Εβδομάδα  10 11 12 13 14 15 16
4η Εβδομάδα  17 18 19 20 21 22 23
5η Εβδομάδα  24 25 26 27 28 29 30
6η Εβδομάδα  31
```

4.1.3.2 Ο τύπος array (πίνακας n διαστάσεων)

Η δομή δεδομένων που παρέχει το array είναι ένας πολυδιάστατος (multidimensional) πίνακας που αποτελείται από δεδομένα του ίδιου τύπου. Σε ένα array τα δεδομένα μπορούν να οργανωθούν ως πίνακας 1, 2, 3 ή και περισσότερων διαστάσεων. Το matrix που περιεγράφηκε παραπάνω είναι απλώς η υποπερίπτωση 2-διαστάσεων του array και πολλά από όσα αναφέρθηκαν για το matrix έχουν εφαρμογή σε array αρκεί να προσαρμοστούν στις παραπάνω διαστάσεις του αντικειμένου[208]. Για να δημιουργηθεί ένα array χρειάζεται ως παραμέτρους τα αρχικά δεδομένα (ένα διάνυσμα αρχικών τιμών με ένα ή περισσότερα στοιχεία) στην 1η παράμετρο data, καθώς και ένα διάνυσμα με το εύρος των διαστάσεων στη 2η παράμετρο dim. Αν δεν έχουν δοθεί αρκετά δεδομένα για την αρχικοποίηση όλων των στοιχείων του array, ανακυκλώνονται και επαναχρησιμοποιούνται τα δοθέντα.

Δοκιμάστε:	**Σχόλιο**
`a1<-array(100,c(5,7,2,10))`	Πίνακας 5x7x2x10 (4 διαστάσεων) με αρχικές τιμές 100.
`a2<-array(c(10,20,30,40))`	Δεν δόθηκαν διαστάσεις, άρα επιστρέφει μονοδιάστατο πίνακα.
`a3<-array(c(5,10,15,1,3,5),c(3,2))`	Πίνακας 3x2 (2 διαστάσεων, δηλαδή matrix).
`a4<-array(1:24,c(4,3,2))`	Πίνακας 4x3x2 (3 διαστάσεων), αρχικά δεδομένα το 1 έως 24.
`a5<-array(1:24,c(2,2,3,2))`	Πίνακας 4 διαστάσεων, αρχικά δεδομένα το 1 έως 24.

Λίγος πειραματισμός με το 3-διάστατο (3-d) array a4 του παραπάνω παραδείγματος βοηθά να κατανοήσουμε τη δομή των αντικειμένων τύπου array. Αν ανακληθεί το a4 στο Console, αυτό επιστρέφει:

```
> a4
, , 1

     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
```

[208] Ο τύπος matrix είναι επέκταση του τύπου array. Ο τύπος array οποίος είναι επέκταση του τύπου vector.

```
[3,]    3    7   11
[4,]    4    8   12

, , 2

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

Ανάγοντας σε 3 διαστάσεις την ίδια σύνταξη που χρησιμοποιήθηκε στα matrix, η ένδειξη « , , 1» που εμφανίζεται συμβολίζει ότι ακολουθούν όλα τα στοιχεία της 1ης και 2ης διάστασης με τιμή 1 στην 3η διάσταση. Η 3η διάσταση ορίζει διαφορετικά επίπεδα όπου καθένα περιέχει ένα 2-διάστατο[209]. Τα δεδομένα χωρίζονται με σειρά από την τελευταία διάσταση προς την πρώτη[210]. Πρώτα χωρίζονται ως προς την 3η διάσταση σε δύο «επίπεδα» και μετά καθένα από αυτά τα 2-d επίπεδα χωρίστηκε ως προς τη 2η διάσταση (τις στήλες του). Πρόσβαση στο a4 ως διάνυσμα δίνει το δοθέν 1:24. Τα στοιχεία 1 έως 12 ανήκουν στο 1ο επίπεδο της 3ης διάστασης (δηλαδή στο a4[,,1]), τα 13 ως 24 στο 2ο (δηλαδή στο a4[,,2]). Τα στοιχεία 1 έως 4 ανήκουν στην 1η στήλη του 1ου επιπέδου (δηλαδή στο a4[,1,1]), τα στοιχεία 5 έως 8 στη 2η στήλη του 1ου επιπέδου (δηλαδή στο a4[,2,1]) κλπ. Ακολουθούν μερικοί περαιτέρω χειρισμοί του a4:

Δοκιμάστε:	**Σχόλιο**
`dim(a4)`	Οι διαστάσεις του a4, επιστρέφουν c(4, 3, 2).
`length(a4)`	Ο αριθμός στοιχείων στο a4, επιστρέφει 24.
`a5<-a4/2`	Πράξη αριθμού με αριθμητικό array επιστρέφει array ιδίων διαστάσεων.
`a4[1,3,2]`	Το στοιχείο του a4 στη θέση [1,3,2], επιστρέφει 21.
`a4[1,3,2]<-100`	Το στοιχείο του a4 στη θέση [1,3,2] να πάρει την τιμή 100.
`a4[2,,1]`	Τα στοιχεία 2ης γραμμής στο επίπεδο 1 της 3ης διάστασης, επιστρέφει c(2,6,10).
`a4[2,,-1]`	Τα στοιχεία 2ης γραμμής που δεν είναι σε επίπεδο 1 της 3ης διάστασης, εδώ c(14,18,22).
`a4[2,,]<-200`	Όλα τα στοιχεία όπου η 1η διάσταση είναι 2 (σε 2η γραμμή) να πάρουν τιμή 200.
`aperm(a4,c(2,1,3))`	Αντιμετάθεση της 2ης με την 1η διάσταση στο a4, ενώ η 3η παραμένει.
`dim(a4)<-c(6,4)`	Μετατροπή του a4 σε 2-διαστάσεων (6x4).

Στα παραπάνω παραδείγματα παρατηρήστε πως η σύνταξη ακολουθεί (όπου είναι εφικτό) τους ίδιους κανόνες με τους αντίστοιχους που εφαρμόστηκαν σε vector και matrix. Όμως αυτό δεν είναι πάντα εφικτό. Για παράδειγμα η συνάρτηση αντιμετάθεσης **t** που αναφέρθηκε στα matrix δεν μπορεί να εφαρμοστεί σε αντικείμενα με διάσταση μεγαλύτερη του 2 καθώς δεν είναι προφανές ποιες από τις αρχικές διαστάσεις θα πρέπει να αντιμετατεθούν. Στην περίπτωση αυτή, η αντιμετάθεση γίνεται με χρήση της συνάρτησης **aperm** που επιτρέπει ακριβώς αυτό.

Μια ακόμα από τις πολλές βοηθητικές συναρτήσεις που αφορούν matrix και array και αξίζει να αναφερθεί είναι η συνάρτηση **drop**. Η drop απορρίπτει οποιαδήποτε διάσταση έχει μόνο ένα επίπεδο. Για παράδειγμα, αν οριστεί ένας 3-διάστατος πίνακας a5 με την εντολή a5 <- array(1:10, c(2,5,1)), τότε το a5 είναι:

```
> a5
, , 1

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Παρατηρήστε πως στην 3η διάσταση υπάρχει μόνο το επίπεδο [,,1]. Η drop μπορεί να καταργήσει αυτό το επίπεδο, επιστρέφοντας έναν πίνακα 2-διαστάσεων:

[209] Σε ένα αντικείμενο N διαστάσεων (N-d) κάθε επίπεδο μίας διάστασης «περιέχει» ένα (N-1)-διάστατο χώρο. Έτσι π.χ. σε ένα 4-d πίνακα, κάθε επίπεδο κάποιας διάστασης προσδιορίζει έναν 3-d πίνακα (όπως το a4 στο παράδειγμα) των υπολοίπων διαστάσεων.

[210] Δεν είναι πραγματικός διαχωρισμός, ούτε γίνεται κάποια εσωτερική μεταφορά δεδομένων. Π.χ. για τις συγκεκριμένες διαστάσεις του a4 που είναι c(4,3,2), το a4[,,1] σημαίνει «το πρώτο μισό (1/2) του διανύσματος δεδομένων στο a4», ενώ το a4[,2,1] σημαίνει «τα στοιχεία στο 2ο τρίτο (1/3) του πρώτου μισού (1/2)».

```
> drop(a5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Οι εφαρμογές των πολυδιάστατων πινάκων είναι πολλές. Αν αναρωτιέστε τι μπορεί να είναι μια τέτοια εφαρμογή που αξιοποιεί πίνακες με 3 ή περισσότερες διαστάσεις, επιστρέψτε στην προηγούμενη ενότητα στο 1ο παράδειγμα χρήσης matrix[211]. Στο παράδειγμα εκείνο, χρησιμοποιήθηκε ένας 2-d πίνακας (24 x 3) για να καταγράφεται ο αριθμός οχημάτων που περνά κάποια διόδια ανά ώρα της ημέρας, ενώ υπήρχαν 3 κατηγορίες οχημάτων. Αν όμως έπρεπε να γίνει καταγραφή πολλών 24ωρων, π.χ. καταγραφές μιας εβδομάδας, ίσως διευκόλυνε η χρήση ενός 3-d πίνακα, άρα ενός array. Στο επόμενο παράδειγμα επιλέχτηκε το array αυτό να έχει διαστάσεις 3 x 24 x 7 (3 τύποι οχήματος, 24 ώρες, 7 ημέρες) και ονομάζεται av. Επιπροσθέτως, χρησιμοποιείται η συνάρτηση **dimnames** που δίνει πρόσβαση στη λίστα[212] ονομάτων για τους δείκτες κάθε διάστασης (εδώ θεωρούμε ως 1η ημέρα της εβδομάδας τη Δευτέρα) και καταχωρούνται μερικά ενδεικτικά δεδομένα[213]:

```
av <- array(0,dim=c(3,24,7))

# Ορισμός ονομάτων στις τιμές δεικτών των διαστάσεων 1 και 3:

dimnames(av)[[1]]<-c("επιβατικά","φορτηγά","δίκυκλα")
dimnames(av)[[3]]<-c("Δε","Τρ","Τε","Πε","Πα","Σα","Κυ")

# Καταχώριση ενδεικτικών καταγραφών:

av[1,1,1] <- 16      # 16 επιβατικά την 1η ώρα της 1ης ημέρας
av[2,1,1] <- 2       #  2 φορτηγά την 1η ώρα της 1ης ημέρας
av[3,1,1] <- 3       #  3 δίκυκλα την 1η ώρα της 1ης ημέρας
av[1,2,1] <- 9       #  9 επιβατικά την 2η ώρα της 1ης ημέρας
av[2,2,1] <- 6       #  6 φορτηγά την 2η ώρα της 1ης ημέρας
av[3,2,1] <- 4       #  4 δίκυκλα την 2η ώρα της 1ης ημέρας

# Πρόσθεση ενός ακόμα δίκυκλου στη 2η ώρα της 1ης ημέρας
# με χρήση των ονομάτων των σχετικών δεικτών:

av["δίκυκλα",2,"Δε"] <- av["δίκυκλα",2,"Δε"] + 1
```

Παρακάτω εξάγονται κάποιες βασικές πληροφορίες από τα δεδομένα στο av:

Δοκιμάστε:	**Σχόλιο**
`sum(av)`	Ο συνολικός αριθμός οχημάτων στο array (εδώ επιστρέφει 41).
`sum(av[,2,1])`	Ο συνολικός αριθμός οχημάτων τη 2η ώρα της 1ης ημέρας (εδώ 20).
`sum(av[,2,])`	Ο συνολικός αριθμός οχημάτων τη 2η ώρα, ανεξαρτήτως ημέρας (επίσης 20).
`sum(av[3,,6:7])`	Σύνολο φορτηγών που καταγράφηκαν την 5η έως την 7η ημέρα (Πα,Σα,Κυ)
`rowSums(av)`	Σύνολο οχημάτων που έχουν καταγραφεί ανά είδος (εδώ c(25,8,8))
`rowSums(av,dims=2)`	Σύνολο οχημάτων ανά είδος και ώρα, ανεξαρτήτως ημέρας (matrix 3x24)[214].
`colSums(av[,,1])`	Σύνολο οχημάτων (κάθε είδους) ανά ώρα της 1ης ημέρας, εδώ c(21,20,0,0,…,0)
`sum(av[2,,])`	Ο συνολικός αριθμός φορτηγών ανεξαρτήτως ώρας και ημέρας (εδώ 8).
`rowSums(av)[2]`	Ίδιο με το παραπάνω.
`mean(colSums(av[,,1]))`	Μέσος όρος οχημάτων (κάθε είδους) ανά ώρα της 1ης ημέρας (εδώ 1.708333).
`colMeans(av,dims=2)`	Μέσος όρος οχημάτων (κάθε είδους) την ώρα, ανά ημέρα, c(0.5694,0,0,0,0,0,0).

[211] βλ.§4.1.3.1 Ο τύπος matrix.
[212] Για τα αντικείμενα του τύπου list, βλ.§4.2.1 Ο τύπος list (λίστα).
[213] Τα δεδομένα θα μπορούσαν να καταχωρηθούν και χειροκίνητα. Αν επιλεγεί η συνάρτηση edit για τον λόγο αυτό, θα πρέπει να γίνει επεξεργασία ανά 2-d επίπεδο, π.χ. για τα δεδομένα της 1ης ημέρας (Δευτέρας) με την εντολή av[, ,1] <- edit (av[, ,1]) ή με το σχετικό όνομα που έχουμε ορίσει για τη διάσταση αυτή, δηλαδή av[, ,"Δε"] <- edit (av[, ,"Δε"]).
[214] Σε συναρτήσεις όπως οι rowSums, rowMeans, colSums, colMeans, η παράμετρος dims ορίζει ποια διάσταση θα θεωρείται γραμμή (row) ή στήλη (column) του πίνακα κατά τους υπολογισμούς.

Αν, όπως έγινε και στο αντίστοιχο παράδειγμα με matrix, το κόστος διέλευσης ανά κατηγορία οχήματος έχει τοποθετηθεί σε διάνυσμα με όνομα, το χρηματικό ποσό που εισπράχθηκε ανά τύπο οχήματος μπορεί να υπολογιστεί ανά ημέρα, ώρα κλπ. με παρόμοιο τρόπο. Εδώ επειδή φροντίσαμε η κάθε στήλη να περιέχει 3 στοιχεία (που αντιστοιχούν στον τύπο οχήματος) ο πολλαπλασιασμός του πίνακα av με το διάνυσμα cv θα παράγει το ζητούμενο αποτέλεσμα:

Δοκιμάστε:	**Σχόλιο**
`cv <- c(1, 2, 0.5)`	Κόστος διέλευσης (διοδίων) ανά κατηγορία οχήματος.
`ev <- av * cv`	Εισπράξεις από διόδια, στο array ev (επίσης 3 x 24 x 7).
`sum(ev)`	Σύνολο εισπράξεων (εδώ επιστρέφει 45).
`sum(ev[2,,])`	Σύνολο εισπράξεων από φορτηγά (εδώ 16).
`sum(ev["φορτηγά",,])`	Ίδιο με το παραπάνω.
`cv %*% av[,,1]`	Πολλαπλασιασμός πινάκων, επιστρέφει σύνολο εισπράξεων ανά ώρα 1ης ημέρας.

4.1.3.3 Η συνάρτηση outer

Στη Γραμμική Άλγεβρα, το εξωτερικό γινόμενο (outer product) δύο διανυσμάτων μήκους m και n αντίστοιχα είναι ένας πίνακας (m x n) όπου κάθε στοιχείο του πίνακα αυτού είναι το γινόμενο των επιμέρους στοιχείων στις αντίστοιχες θέσεις των δύο διανυσμάτων. Εξωτερικό γινόμενο μπορούμε να παράγουμε τόσο για διανύσματα όσο και για αντικείμενα σχήματος πίνακα με περισσότερες διαστάσεις (matrix, array κλπ). Το αποτέλεσμα θα είναι ένα αντικείμενο του οποίου η διάσταση θα είναι ίση με το άθροισμα των διαστάσεων των δυο εμπλεκομένων αντικειμένων. Άρα αν πρόκειται για δύο vectors (που ουσιαστικά είναι μονοδιάστατα) το αποτέλεσμα θα είναι ένα δισδιάστατο array, δηλαδή ένα matrix. Ο τελεστής **%o%** παράγει το εξωτερικό γινόμενο, ενώ η συνάρτηση **outer** είναι η γενικευμένη μορφή του ίδιου μηχανισμού που επιτρέπει την εφαρμογή και άλλων συναρτήσεων πέραν του πολλαπλασιασμού.

Δοκιμάστε:	**Σχόλιο**
`v1 <- 1:2`	Ένα αριθμητικό διάνυσμα v1 (ίσο με c(1,2)).
`v2 <- 1:10`	Ένα άλλο αριθμητικό διάνυσμα v2 (ίσο με c(1,2,3,4,5,6,7,8,9,10)).
`v2 %o% v1`	Το εξωτερικό γινόμενο του v2 με το v1.
`outer(v2,v1)`	Το εξωτερικό γινόμενο του v2 με το v1. Ίδιο με το παραπάνω.
`v1 %o% v2`	Το εξωτερικό γινόμενο του v1 με το v2.
`outer(v1,v2)`	Το εξωτερικό γινόμενο του v1 με το v2. Ίδιο με το παραπάνω.

Έτσι π.χ. το αποτέλεσμα της εντολής outer(v2,v1) (ή της ισοδύναμης v2 %o% v1) είναι:

```
> outer(v2,v1)
     [,1] [,2]
 [1,]    1    2
 [2,]    2    4
 [3,]    3    6
 [4,]    4    8
 [5,]    5   10
 [6,]    6   12
 [7,]    7   14
 [8,]    8   16
 [9,]    9   18
[10,]   10   20
```

Το στοιχείο στη θέση [n,m] του πίνακα είναι ίσο με v2[n]*v1[m]. Έτσι το outer(v2,v1)[5,2] είναι ίσο με v2[5]*v1[2]. Η outer επεξεργάζεται (πολλαπλασιάζοντας) καθέναν από τους πιθανούς συνδυασμούς στοιχείων των δύο αντικειμένων. Όμως εκτός του πολλαπλασιασμού, η συνάρτηση outer επιτρέπει τον ορισμό άλλων συναρτήσεων που θα εφαρμοστούν σε κάθε συνδυασμό στοιχείων. Αυτό γίνεται μέσω της παραμέτρου FUN. Η συνάρτηση που θα οριστεί στη FUN για την επεξεργασία κάθε συνδυασμού στοιχείων θα πρέπει να δέχεται ως πρώτες παραμέτρους (ή X,Y) δύο στοιχεία που θα επεξεργαστεί. Μαζί με το ζεύγος στοιχείων θα περαστούν σε αυτή και επιπρόσθετες παράμετροι που ενδεχομένως έχουν οριστεί αλλά δεν αξιοποιούνται από την outer. Έτσι π.χ. η εντολή outer(v1,v2,FUN = "paste", sep="") καλεί τη συνάρτηση paste (με παράμετρο sep="") για

τα ζεύγη κάθε συνδυασμού στοιχείων, συγχωνεύοντας τα στοιχεία αυτά ως κείμενο σε ένα αντικείμενο character (βλ.§2.3 Συναρτήσεις κειμένου και ο βασικός τύπος character) και τελικά παράγοντας τον πίνακα:

```
> outer(v1,v2,FUN = "paste", sep="")
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] "11" "12" "13" "14" "15" "16" "17" "18" "19" "110"
[2,] "21" "22" "23" "24" "25" "26" "27" "28" "29" "210"
```

Αντίστοιχα, η εντολή outer(v1,v2,FUN = "+") θα προσθέσει τα δύο στοιχεία, παράγοντας τον πίνακα:

```
> outer(v1,v2,FUN = "+")
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    2    3    4    5    6    7    8    9   10    11
[2,]    3    4    5    6    7    8    9   10   11    12
```

Ο μηχανισμός της outer είναι ιδιαίτερα χρήσιμος. Εκτός από τελεστές και έτοιμες συναρτήσεις η παράμετρος FUN μπορεί να αναφέρεται σε κάποια συνάρτηση που έχει δημιουργήσει ο κώδικας του χρήστη (βλ.§5.1.1 Δημιουργία συναρτήσεων). Αυτές οι δυνατότητες ανάγουν την outer σε έναν εναλλακτικό τρόπο επεξεργασίας όλων των πιθανών συνδυασμών των στοιχείων δύο αντικειμένων, αποφεύγοντας τη χρήση βρόχων για τον σκοπό αυτόν.

Ακολουθεί ένα παράδειγμα αυτής της προσέγγισης. Ας θεωρήσουμε πως σε δύο διανύσματα v1 και v2 αναζητούνται οι θέσεις των στοιχείων του v1 τα οποία αν προστεθούν σε κάποιο (οποιοδήποτε) στοιχείο του v2 το αποτέλεσμα είναι 10. Αν π.χ. τα δύο διανύσματα είναι:

```
v1 <- c(1, 4, 2, 5, 6, 2, 9, 10)
v2 <- c(9, 2, 3, 1, 5, 7)
```

θα πρέπει να ελεγχθεί το άθροισμα κάθε συνδυασμού στοιχείων από τα v1 και v2, αντίστοιχα και αν το άθροισμα προκύπτει ίσο με 10, να καταγράφεται η θέση του εμπλεκομένου στοιχείου στο v1. Παρεμφερή προβλήματα είναι αρκετά συνήθη στον προγραμματισμό. Ο συνήθης τρόπος προσέγγισης ενός τέτοιου προβλήματος είναι με χρήση εμφωλευμένων βρόχων για τη δημιουργία δεικτών που θα διατρέξουν τα δύο διανύσματα. Μια τέτοια λύση εφαρμόζεται παρακάτω:

```
pa <-NULL
for (j in 1:length(v2))
  for (i in 1:length(v1))
    if ((v1[i] + v2[j]) == 10)
      pa <- c(pa, i)
```

Εδώ το pa θα είναι το διάνυσμα αποτελεσμάτων (αρχικά κενό). Οι δύο βρόχοι δημιουργούν δείκτες που διατρέχουν τα στοιχεία των δύο διανυσμάτων v1 και v2. Σε κάθε ζεύγος τιμών για τους δείκτες i και j, αν το άθροισμα των σχετικών στοιχείων (v1[i]+v2[j]) είναι 10 ο δείκτης του v1 (το τρέχον i) προσαρτάται στο pa. Για τα δοθέντα v1 και v2, το διάνυσμα pa που θα προκύψει θα περιέχει τις ζητούμενες θέσεις[215], δηλαδή οι 1, 7 και 4.

Αν αντί βρόχων χρησιμοποιηθεί η συνάρτηση outer για την εκτέλεση των συνδυασμών, ο παραπάνω κώδικας μπορεί να αντικατασταθεί από την παρακάτω μία εντολή:

```
pa <- which(outer(v1, v2, FUN = "+") == 10, arr.ind = T)[, 1]
```

Το αποτέλεσμα στο pa θα είναι το ίδιο. Η παραπάνω εντολή «σπάει» εσωτερικά σε 3 μέρη: (α) Πρώτα εκτελείται η συνάρτηση (εδώ ο τελεστής “+”) για όλους τους συνδυασμούς. Κάτι τέτοιο έχει ως αποτέλεσμα το:

```
> outer(v1,v2,FUN = "+")
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   10    3    4    2    6    8
[2,]   13    6    7    5    9   11
[3,]   11    4    5    3    7    9
[4,]   14    7    8    6   10   12
[5,]   15    8    9    7   11   13
[6,]   11    4    5    3    7    9
[7,]   18   11   12   10   14   16
```

[215] Στο συγκεκριμένο παράδειγμα κάποια θέση δύναται να καταγράφεται στο pa πάνω από μία φορά. Αν πρέπει κάθε θέση να εμφανίζεται μόνο μία φορά, μπορεί να γίνει εξάλειψη των επαναλαμβανόμενων τιμών με χρήση της συνάρτησης unique (εκτελώντας την εντολή pa <- unique(pa)).

```
[8,]   19   12   13   11   15   17
```

(β) Ακολούθως ζητούνται μέσω της συνάρτησης which οι θέσεις του παραπάνω πίνακα με στοιχεία που πληρούν το κριτήριο (ισούνται με 10), κάτι που παράγει:

```
> which(outer(v1,v2,FUN = "+")==10,arr.ind = T)
     row col
[1,]   1   1
[2,]   7   4
[3,]   4   5
```

(γ) Τέλος, κρατώντας ως ρα μόνο τη στήλη 1 (που αντιστοιχεί στις γραμμές του πίνακα που δημιούργησε η outer, δηλαδή σε δείκτες του v1) προκύπτει ίδιο αποτέλεσμα με το προηγούμενο, δηλαδή οι θέσεις 1, 7 και 4 του v1.

Στο συγκεκριμένο παράδειγμα, η outer απλώς καλεί τον τελεστή "+" και το ζητούμενο αποτέλεσμα προκύπτει μέσω της συνάρτησης which που εφαρμόζει ένα απλό κριτήριο ελέγχου ισότητας. Όμως στη θέση αυτών μπορούν να εφαρμοστούν άλλες συναρτήσεις με διαφορετική λειτουργία, επιτρέποντας σε πολλές περιπτώσεις την υποκατάσταση εμφωλευμένων βρόχων με επεξεργασία όλων των συνδυασμών στοιχείων δύο αντικειμένων μέσω της outer. Άλλοι τρόποι που επιτρέπουν την επεξεργασία πολυμελών δομών χωρίς τη χρήση βρόχων αναφέρονται στη §5.4 Η συναρτησιακή προσέγγιση στον πραγματικό κόσμο.

4.1.3.4 Η οικογένεια συναρτήσεων apply

Σε πολυμελή αντικείμενα είναι συχνά ανάγκη να γίνει εφαρμογή της ίδιας επεξεργασίας σε υποσύνολα των στοιχείων τους. Αν πρόκειται για αντικείμενα με μορφή πίνακα, μπορεί να χρειάζεται να γίνει η ίδια επεξεργασία ανά επίπεδο κάποιας διάστασης, π.χ. να εκτελεστεί κάποια συνάρτηση για καθεμία γραμμή ή καθεμία στήλη του πίνακα. Αν πρόκειται για αντικείμενα παρεμφερή με τα vector, μπορεί να χρειάζεται να οριστούν υποσύνολα των στοιχείων (π.χ. βάσει κάποιου κριτηρίου) και να επαναληφθεί κάποια επεξεργασία σε καθένα από αυτά. Οι συναρτήσεις που φέρουν το όνομα apply (apply, lapply, sapply, tapply κλπ) καθώς και οι sweep και aggregate είναι κατάλληλες για τον σκοπό αυτό.

Η συνάρτηση **apply** αφορά αντικείμενα με μορφή πίνακα (συμπεριλαμβανομένων και array με διαστάσεις περισσότερες των δύο). Η συνάρτηση δέχεται ως παραμέτρους το αντικείμενο, τη διάσταση (MARGIN) βάσει της οποίας θα γίνει ο διαχωρισμός του αντικειμένου σε τμήματα και τη συνάρτηση (FUN) που θα εκτελεστεί για την επεξεργασία των τμημάτων αυτών. Αν το αντικείμενο δεν είναι array θα μετατραπεί σε array. Επιπρόσθετες παράμετροι που ενδεχομένως δίνονται αλλά δεν αξιοποιούνται από την apply θα περάσουν ως παράμετροι στη συνάρτηση που ορίζεται με την παράμετρο FUN[216]. Η apply θα χωρίσει το N-διαστάσεων αντικείμενο για κάθε τιμή της διάστασης που προσδιορίζεται στο MARGIN και θα εφαρμόσει τη συνάρτηση πάνω σε αυτό το (N-1)-διαστάσεων τμήμα του αντικειμένου. Έτσι είτε πρόκειται για ένα matrix είτε για ένα array πολλών διαστάσεων, αν το MARGIN είναι 1 τότε η FUN θα εφαρμοστεί για κάθε επίπεδο της 1ης διάστασης, δηλαδή για κάθε γραμμή. Με τιμή στο MARGIN το 2 η FUN θα εφαρμοστεί για κάθε στήλη, ενώ αν το MARGIN οριστεί ως 3 η FUN θα εφαρμοστεί για κάθε επίπεδο της 3ης διάστασης, κλπ. Για τα παραδείγματα που ακολουθούν θεωρήστε πως έχει οριστεί ένα 3-διάστατο array με την εντολή a <- array(1:8, c(2,2,2)) και το a είναι:

```
> a
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

[216] Το ίδιο ίσχυε για την outer παραπάνω και εφαρμόζεται συχνά σε συναρτήσεις οι οποίες δέχονται ως παράμετρο άλλες συναρτήσεις. Για τον συμβολισμό "…" (τριών τελειών) στις παραμέτρους συναρτήσεων βλ.§5.1.2 Παράμετροι.

Η εντολή apply(a,MARGIN=1,FUN=sum) ή απλώς apply(a,1,sum) θα εφαρμόσει τη συνάρτηση sum σε κάθε επίπεδο της 1ης διάστασης (άρα σε κάθε γραμμή) του a. Η εντολή θα επιστρέψει:

```
> apply(a,1,sum)
[1] 16 20
```

Το πρώτο στοιχείο (16) προέκυψε από την επεξεργασία των στοιχείων του a που βρίσκονται στην 1η γραμμή, οπότε εδώ είναι η εφαρμογή της sum στα 1, 3, 5 και 7, το άθροισμα των τιμών αυτών (συγκεκριμένα είναι η εφαρμογή του sum σε matrix με 2 στήλες που περιέχουν τα 1, 2 και 3, 7 αντίστοιχα). Το δεύτερο (20) αντιστοιχεί στα στοιχεία του a στη 2η γραμμή, δηλαδή το άθροισμα των στοιχείων 2,4, 6 και 8. Ομοίως, αν εκτελεστεί η apply για τη 2η διάσταση (στήλες):

```
> apply(a,2,sum)
[1] 14 22
```

Ενώ για την 3η διάσταση έχουμε:

```
> apply(a,3,sum)
[1] 10 26
```

Η τιμή 1 στην παράμετρο MARGIN που δόθηκε παραπάνω χώρισε το a σε τμήματα με βάση την 1η διάσταση, τις γραμμές του πίνακα αυτού. Μετά εκτέλεσε τη FUN (εδώ sum) στα υποσύνολα αυτά. Ως MARGIN όμως επιτρέπεται να δοθεί και διάνυσμα με παραπάνω από έναν αριθμό προσδιορίζοντας περισσότερες διαστάσεις που θα χρησιμοποιηθούν για τον επιμερισμό του αντικειμένου. Για παράδειγμα, η εντολή apply(a, MARGIN=c(1,2), FUN=sum) είναι παραλλαγή του παραπάνω παραδείγματος όπου η sum θα εκτελεστεί για κάθε συνδυασμό επιπέδων 1ης (γραμμής) και 2ης διάστασης (στήλης). Άρα, στοιχεία του a σε ίδια γραμμή και στήλη (ασχέτως 3ης διάστασης), θα ορίσουν ένα από τα υποσύνολα πάνω στα οποία θα εφαρμοστεί η sum, κάτι που παράγει το αποτέλεσμα:

```
> apply(a, MARGIN=c(1,2), FUN=sum)
     [,1] [,2]
[1,]    6   10
[2,]    8   12
```

Η σειρά με την οποία δίνονται οι διαστάσεις στο MARGIN, ελέγχει τη σειρά με την οποία θα διαχωριστούν και θα παραχθεί το αποτέλεσμα. Έτσι το παραπάνω θα μπορούσε να διαβαστεί: «για κάθε επίπεδο στην 1η διάσταση (γραμμή) και ακολούθως για κάθε επίπεδο στη 2η διάσταση (στήλη) άθροισε τα στοιχεία στις υπόλοιπες διαστάσεις που ανήκουν σε αυτά τα επίπεδα». Η εντολή apply(a, MARGIN=c(2,1), FUN=sum) θα μπορούσε να διαβαστεί: «για κάθε επίπεδο στη 2η διάσταση (στήλη) και ακολούθως για κάθε επίπεδο στην 1η διάσταση (γραμμή) άθροισε τα στοιχεία στις υπόλοιπες διαστάσεις που ανήκουν σε αυτά τα επίπεδα», οπότε τα αποτελέσματα τοποθετούνται σε διαφορετική θέση στην έξοδο:

```
> apply(a,MARGIN=c(2,1),FUN=sum)
     [,1] [,2]
[1,]    6    8
[2,]   10   12
```

Ως τελευταίο σχετικό παράδειγμα, η εντολή apply(a, MARGIN=c(2,3), FUN=mean) υπολογίζει τον αριθμητικό μέσο όρο κάθε στήλης για κάθε επίπεδο της 3ης διάστασης. Το παράδειγμα θα ήταν χρήσιμο αν π.χ. το a ήταν ένα σύνολο δεδομένων (data set), με τις γραμμές να αντιστοιχούν σε περιπτώσεις (cases), οι στήλες σε μεταβλητές (variables) που περιγράφουν τις περιπτώσεις αυτές, ενώ η 3η διάσταση αντιστοιχούσε στην κατηγορία (class) των δεδομένων. Τέτοια οργάνωση δεν είναι σπάνια σε σύνολα δεδομένων[217] και η παραπάνω εντολή θα έβρισκε τη μέση τιμή κάθε μεταβλητής ανά κατηγορία:

```
> apply(a,MARGIN=c(2,3),FUN=mean)
     [,1] [,2]
[1,]  1.5  5.5
[2,]  3.5  7.5
```

Οι συναρτήσεις sum και mean που χρησιμοποιήθηκαν στα προηγούμενα παραδείγματα επιστρέφουν μια μοναδική τιμή (ένα αντικείμενο με length=1) για κάθε τμήμα του αντικειμένου που επεξεργάζονται. Όμως η apply μπορεί να προσπαθήσει να εφαρμόσει οποιαδήποτε συνάρτηση ορίσει ο χρήστης, συμπεριλαμβανομένων τελεστών, άλλων έτοιμων συναρτήσεων ή συναρτήσεων που έχει δημιουργήσει ο χρήστης. Πολλές από αυτές τις συναρτήσεις επιστρέφουν πολυμελή αντικείμενα. Στην περίπτωση αυτή, η παράμετρος simplify της apply

[217] Για παράδειγμα έτσι είναι οργανωμένο το σύνολο δεδομένων iris3 που υπάρχει στο προεγκατεστημένο πακέτο ‘datasets’ (για περισσότερα βλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων).

ορίζει αν οι επιστρεφόμενες τιμές θα «απλοποιηθούν». Ας δούμε τι σημαίνει αυτό με ένα ακόμα παράδειγμα. Στο παρακάτω, η apply εφαρμόζει τη συνάρτηση ταξινόμησης sort (βλ.§2.4.1 Οι βασικές εντολές των διανυσμάτων) με παράμετρο της τελευταίας το decreasing=TRUE. Ζητείται λοιπόν να ταξινομηθούν τα δεδομένα κατά φθίνουσα σειρά. Κάθε εφαρμογή της sort θα επιστρέφει ένα διάνυσμα με τα ταξινομημένα δεδομένα. Στην εντολή που ακολουθεί η παράμετρος decreasing θα περαστεί ως παράμετρος της sort (καθώς δεν είναι παράμετρος της apply για να την αξιοποιήσει). Η 2η παράμετρος (η τιμή 1) αντιστοιχεί στο MARGIN της apply, άρα η εφαρμογή της sort θα γίνει ανά γραμμή, ενώ η παράμετρος simplify=FALSE ζητά από την apply να επιστρέψει τα αποτελέσματα χωρίς «απλοποίηση», κάτι που εδώ σημαίνει ως μια λίστα[218] επιμέρους αποτελεσμάτων:

```
> apply(a,1,simplify=F,FUN=sort, decreasing=T)
[[1]]
[1] 7 5 3 1

[[2]]
[1] 8 6 4 2
```

Αν όμως επιτραπεί η απλοποίηση (όπως στην παρακάτω παραλλαγή της ίδιας εντολής), τα επιστρεφόμενα αντικείμενα θα συγχωνευτούν σε ένα array:

```
> apply(a,1,sort,decreasing=T)
     [,1] [,2]
[1,]    7    8
[2,]    5    6
[3,]    3    4
[4,]    1    2
```

Παρεμφερείς με την apply είναι οι συναρτήσεις βασισμένες στην **lapply** (lapply, sapply, vapply, mapply) οι οποίες μπορούν να εφαρμοστούν και σε αντικείμενα τα οποία δεν έχουν μορφή πίνακα. Οι συναρτήσεις αυτές μετατρέπουν (αν χρειάζεται) το αντικείμενο σε τύπο list και εφαρμόζουν σε κάθε μέλος του (το οποίο μπορεί να περιέχει περισσότερα του ενός στοιχεία) την εντολή που έχει οριστεί μέσω της παραμέτρου FUN. Για παραδείγματα χρήσης των συναρτήσεων αυτών, βλ. τη σχετική ενότητα §4.2.1.3 Εφαρμογή συναρτήσεων σε list.

Επίσης συναφής με τις παραπάνω είναι η συνάρτηση **tapply**. Όπως και στις προαναφερθείσες συναρτήσεις, η 1η παράμετρος της tapply είναι το αντικείμενο στο οποίο θα γίνει η επεξεργασία. Εδώ το αντικείμενο είναι ή μετατρέπεται σε τύπο χωρίς διάσταση (vector, list κλπ.). Όμως, όπως θα δούμε παρακάτω, η tapply βρίσκει εφαρμογές και σε αντικείμενα με σχήμα πίνακα (matrix, array, data.frame κλπ.). Σε κάθε περίπτωση, το αντικείμενο επιμερίζεται σε τμήματα βάσει των τιμών ενός factor ή κάποιας άλλης έκφρασης μετατρέψιμης σε factor[219]. Οι τιμές αυτές, με τις οποίες θα γίνει ο επιμερισμός των δεδομένων, δίνονται στη 2η παράμετρο INDEX. Η επεξεργασία των επιμέρους τμημάτων γίνεται μέσω της συνάρτησης που (και εδώ) ορίζεται με την παράμετρο FUN και μπορεί να είναι οποιαδήποτε συνάρτηση. Έτσι είναι εφικτός ο διαχωρισμός των στοιχείων βάσει κάποιας κατηγοριοποίησής τους (που συχνά προκύπτει ως εφαρμογή κριτηρίων) και χωριστή επεξεργασία κάθε υποσυνόλου των στοιχείων που παράγεται από τον διαχωρισμό αυτό. Ένα τέτοιο παράδειγμα θα ήταν το παρακάτω: ας υποθέσουμε πως έχουμε δεδομένα από ένα κατάστημα που καταγράφουν την ποσότητα κάποιων προϊόντων σε κάθε παραγγελία. Θέλουμε να μετρήσουμε την ποσότητα από κάθε προϊόν που παραγγέλθηκε συνολικά. Άρα ζητούμενο είναι να διαχωριστούν τα δεδομένα (οι ποσότητες) ως προς το προϊόν που αφορούν και να βρεθεί το σύνολο για κάθε υποσύνολο των δεδομένων (ποσοτήτων) που προέκυψε από τον διαχωρισμό αυτό. Τέτοια προβλήματα συνήθως αντιμετωπίζονται με χρήση βρόχων for και δομών επιλογής, όμως εναλλακτικά μπορεί να χρησιμοποιηθεί η συνάρτηση tapply. Ακολουθεί ένα παράδειγμα χρήσης της συνάρτησης αυτής:

Δοκιμάστε:	**Σχόλιο**
`p<-c(5,10,15,5,10)`	Διάνυσμα με δεδομένα στο p.
`t<-c("T1","T2","T3","T2","T3")`	Κατηγορίες για τα δεδομένα του p (θα χρησιμοποιηθεί ως factor).
`s<-tapply(p, t, sum)`	Αθροίζει τα στοιχεία του p ανά κατηγορία στο t.
`s["T2"]`	Επιστρέφει το sum για την κατηγορία "T2", δηλαδή 15.

[218] Για τα αντικείμενα τύπου list βλ.§4.2.1 Ο τύπος list (λίστα).
[219] βλ.§4.1.2 Οι τύποι factor και ordered (παράγοντας).

Αν ανακληθεί όλο το s εμφανίζει το αποτέλεσμα:

```
> s
T1 T2 T3
 5 15 25
```

Παρατηρήστε πως τα παραπάνω διανύσματα θα μπορούσαν να είναι τα δεδομένα του προαναφερθέντος παραδείγματος, δηλαδή το p να περιέχει τις ποσότητες ανά παραγγελία κάποιου προϊόντος, ενώ το t να προσδιορίζει ποιο προϊόν αφορά η αντίστοιχη ποσότητα στο p. Σε κάθε περίπτωση, στην παραπάνω tapply, τα στοιχεία του p χωρίστηκαν σε τρείς ομάδες βάσει των επιπέδων στο t, δηλαδή στα στοιχεία με «ετικέτα» T1 (εδώ ανήκει μόνο το 1ο στοιχείο του p με τιμή 5), στα στοιχεία με «ετικέτα» T2 (εδώ το 2ο με τιμή 10 και το 4ο με τιμή 5) και στα στοιχεία με «ετικέτα» T3 (εδώ το 3ο με τιμή 15 και το 5ο με τιμή 10). Καθώς ως συνάρτηση επεξεργασίας FUN ορίστηκε η sum, η tapply επέστρεψε το άθροισμα των στοιχείων σε κάθε ομάδα. Όμως στην παράμετρο FUN μπορεί να οριστεί οποιαδήποτε συνάρτηση είναι εφαρμόσιμη στα δεδομένα, συμπεριλαμβανομένων και συναρτήσεων που έχουν δημιουργηθεί από τον χρήστη. Σε συνέχεια του παραπάνω:

Δοκιμάστε:	**Σχόλιο**
`tapply(p, t, max)`	Η μέγιστη τιμή ανά κατηγορία, επιστρέφει 5, 10, 15 για T1,T2 και T3.
`tapply(p, t, sort)`	Ταξινόμηση μελών ανά κατηγορία, βλ. παρακάτω.

Το αποτέλεσμα της εφαρμογής της όποιας συνάρτησης ορίστηκε στην παράμετρο FUN δεν είναι απαραίτητα μονομελές (με length ίσο με 1). Για παράδειγμα, η sort που χρησιμοποιείται στην τελευταία εντολή παραπάνω, θα τοποθετήσει σε αύξουσα σειρά τα στοιχεία που ανήκουν σε κάθε κατηγορία (T1, T2 και T3), άρα θα επιστρέψει ανά κατηγορία ένα vector με τα ταξινομημένα δεδομένα[220]:

```
> tapply(p, t, sort)
$T1
[1] 5

$T2
[1]  5 10

$T3
[1] 10 15
```

Η tapply συνήθως επιστρέφει table («πίνακα») των αποτελεσμάτων και σε αυτό παραπέμπει το αρχικό γράμμα της ονομασίας της. Για τον σκοπό αυτόν η tapply μπορεί να χρησιμοποιήσει περισσότερες από μια κατηγοριοποιήσεις των στοιχείων για τον διαχωρισμό τους. Οι κατηγοριοποιήσεις δίνονται στην παράμετρο INDEX ως λίστα[221] που περιέχει τα διάφορα factors (ή σχετικές εκφράσεις) τα οποία ορίζουν την κάθε κατηγοριοποίηση:

Δοκιμάστε:	**Σχόλιο**
`p<-c(1,1,1,5,5,1,5,1)`	Διάνυσμα με δεδομένα στο p.
`f1<-c("A","B","A","A","A","B","B","B")`	Κατηγοριοποίηση για τα δεδομένα του p.
`f2<-c("C","C","C","D","D","D","E","E")`	Άλλη κατηγοριοποίηση για τα δεδομένα του p.
`s<-tapply(p, list(f1,f2), sum)`	Αθροίζει τα στοιχεία του p για τις κατηγορίες f1 και f2.

Αν ανακληθεί το s εμφανίζει το αποτέλεσμα ως έναν πίνακα. Έτσι π.χ. η τιμή 2 στη θέση [1,1] είναι το άθροισμα των στοιχείων του p που ανήκουν ταυτόχρονα στην κατηγορία A του f1 και C του f2 (δηλαδή του 1ου και του 3ου στοιχείου του p). Στη θέση [1,3] του πίνακα αποτελεσμάτων υπάρχει η τιμή NA[222] καθώς κανένα από τα στοιχεία του p δεν βρίσκεται στην κατηγορία A του f1 και E του f2 ταυτόχρονα:

```
> s
  C  D  E
A 2 10 NA
B 1  1  6
```

[220] Συγκεκριμένα επιστρέφει ένα list από vectors. Για τα αντικείμενα τύπου list βλ.§4.2.1 Ο τύπος list (λίστα).
[221] βλ. υποσημείωση 220 παραπάνω.
[222] βλ.§2.2.4 Ειδικές τιμές.

Η tapply είναι προσανατολισμένη στην επεξεργασία αντικειμένων χωρίς διάσταση όπως τα vectors, είναι όμως χρήσιμη και σε προβλήματα με δεδομένα αντικείμενα που έχουν μορφή πίνακα. Για παράδειγμα μπορεί να χρησιμοποιηθεί για να γίνει επεξεργασία μίας γραμμής ή στήλης ενός πίνακα βάσει των δεδομένων μιας άλλης (που θα χρησιμοποιηθεί για τη δημιουργία υποσυνόλων της πρώτης):

Δοκιμάστε:	**Σχόλιο**
`p<-matrix(c(1,2,3,4,1,1,2,3),4)`	Πίνακας p όπου 1η στήλη τα 1,2,3,4 και 2η στήλη τα 1,1,2,3.
`tapply(p[,1],p[,2],sum)`	Αθροίσματα στοιχείων 1ης στήλης βάσει τιμών της 2ης.
`tapply(p[,1],p[,2]==2,sum)`	Αθροίσματα στοιχείων 1ης στήλης αν στη 2η η τιμή είναι 2.

Η πρώτη εντολή παραπάνω (tapply(p[,1],p[,2],sum)) θα ομαδοποιήσει τα στοιχεία της 1ης στήλης σε αυτά που αντιστοιχούν σε τιμή 1 στη 2η (το 1 και το 2), σε αυτά που αντιστοιχούν σε τιμή 2 στη 2η (το 3) και σε αυτά που αντιστοιχούν σε τιμή 3 στη 2η (το 4). Αθροίζοντας τα στοιχεία σε κάθε ομάδα θα επιστρέψει τις τιμές 3,3 και 4. Η δεύτερη εντολή (tapply(p[,1],p[,2]==2,sum)) θα ομαδοποιήσει τα στοιχεία της 1ης στήλης σε αυτά που αντιστοιχούν σε τιμή 2 στη 2η στήλη (το 3) και στα υπόλοιπα (το 1,2 και 4). Αθροίζοντας τα στοιχεία σε κάθε ομάδα θα επιστρέψει τις τιμές 3 και 7, αντίστοιχα. Για άλλα παραδείγματα χρήσης της tapply βλ. [47], ενώ για περισσότερες εφαρμογές συναρτήσεων συναφών με την apply, βλ. [33].

Χρήσιμες βοηθητικές συναρτήσεις είναι οι sweep και aggregate. Η συνάρτηση **sweep** (που είναι βασισμένη στην apply) εφαρμόζει μια συνάρτηση σε τμήματα ενός αντικειμένου με σχήμα πίνακα. Η διάσταση προσδιορίζεται όπως στην apply στη 2η παράμετρο MARGIN. Στην 3η παράμετρο STATS δίνεται από τον χρήστη ένα ακόμα αντικείμενο που θα χρησιμοποιηθεί κατά την επεξεργασία. Η ίδια η συνάρτηση που θα χρησιμοποιηθεί για την επεξεργασία προσδιορίζεται από τον χρήστη στην 4η παράμετρο FUN. Η όποια συνάρτηση FUN θα πρέπει να δέχεται δύο τουλάχιστον παραμέτρους, καθώς η sweep θα καλέσει τη FUN με παραμέτρους, το τμήμα του αντικειμένου (π.χ. μια γραμμή του πίνακα αν το MARGIN είναι 1) μαζί με το αντικείμενο STATS, επαναλαμβάνοντας τη διαδικασία για όλα τα τμήματα. Αν αυτό ακούγεται πολύπλοκο, μερικά παραδείγματα θα βοηθήσουν:

Δοκιμάστε:	**Σχόλιο**
`p<-matrix(c(4,2,3,4,3,4,6,3),4)`	Πίνακας p όπου 1η στήλη τα 4,2,3,4 και 2η στήλη τα 3,4,6,3.
`sweep(p,1,c(10,20),"+")`	Πίνακας, 1η στήλη 14,22,13,24 και 2η στήλη 13,24,16,23.
`sweep(p,2,c(10,20),"+")`	Πίνακας, 1η στήλη 14,12,13,14 και 2η στήλη 23,24,26,23.
`sweep(p,2,apply(p,2,min),"-")`	Αφαιρεί από κάθε στοιχείο την ελάχιστη τιμή της στήλης του.

Η συνάρτηση **aggregate** είναι σχεδιασμένη για δημιουργία συγκεντρωτικών αποτελεσμάτων. Όπως η tapply και αυτή η συνάρτηση () χωρίζει τα στοιχεία του δοθέντος αντικειμένου σε τμήματα βάσει κάποιας κατηγοριοποίησης τους. Αυτή η κατηγοριοποίηση παρέχεται συνήθως από ένα σχετικό factor, ή συχνά από κάποιο συνδυασμό τέτοιων κατηγοριοποιήσεων. Ακολουθεί ένα απλοϊκό παράδειγμα εφαρμογής της aggregate σε array, όμως η συνάρτηση αυτή συνήθως εφαρμόζεται σε άλλους τύπους αντικειμένων όπως data.frame (βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων)) και formula (βλ.§4.3.4 Αντικείμενα τύπου language (expression, call, name και formula)):

Δοκιμάστε:	**Σχόλιο**
`p<-matrix(c(1,2,3,4,5,5,6,5),4)`	Πίνακας p όπου 1η στήλη τα 1,2,3,4 και 2η στήλη τα 5,5,6,5.
`aggregate(p[,1],list(p[,2]),sum)`	Συγκεντρωτικά αθροίσματα 1ης στήλης με κατηγορία στη 2η.

Εδώ ζητήθηκε από την aggregate να επεξεργαστεί την 1η στήλη του p (δηλαδή το p[,1], βάσει των ομάδων που ορίζει η τιμή στη δεύτερη παράμετρο (δηλαδή των p[,2]). Αυτό γίνεται ως εξής: η δεύτερη παράμετρος της συνάρτησης (με όνομα by) είναι μια λίστα με factors που θα χρησιμοποιηθούν για τον διαχωρισμό του αντικειμένου προς επεξεργασία. Στο παράδειγμα ορίζεται μόνο μια τέτοια κατηγοριοποίηση (μπορούν όμως να οριστούν περισσότερες). Συγκεκριμένα, ο διαχωρισμός θα καταγράφεται στη 2η στήλη του πίνακα p (δηλαδή το p[,2]). Έτσι η 2η στήλη θα χρησιμοποιηθεί ως factor για τον διαχωρισμό των στοιχείων της 1ης σε ομάδες (group). Από αυτό προκύπτουν δύο ομάδες, μία με το 1ο , 2ο και 4ο στοιχείο της 1ης στήλης του p, ενώ το 3ο στοιχείο αποτελεί μόνο του τη δεύτερη (μονομελή) ομάδα. Καθώς έχει οριστεί ως FUN (3η παράμετρος) η συνάρτηση sum, τα στοιχεία σε κάθε ομάδα αθροίζονται με αποτέλεσμα:

```
Group.1 x
```

```
1         5 7
2         6 3
```

Όπως σε όλες τις προαναφερθείσες εντολές, το αποτέλεσμα είναι ένα αντικείμενο και ως τέτοιο μπορεί να γίνει πρόσβαση στα στοιχεία του. Εδώ το αποτέλεσμα είναι ένας πίνακας, άρα η εντολή aggregate(p[,1],list(p[,2]),sum)[1,2] θα επιστρέψει το στοιχείο στη θέση [1,2] του πίνακα, το οποίο είναι το αποτέλεσμα για την 1η ομάδα και έχει την τιμή 7.

Η συνάρτηση **ave** μπορεί επίσης να χρησιμοποιηθεί για συγκεντρωτικά αποτελέσματα (εξ ορισμού επιστρέφει τον αριθμητικό μέσο όρο καλώντας τη mean), επιστρέφει όμως αριθμό αποτελεσμάτων (length) ίδιο με αυτό των δεδομένων, άρα μία απάντηση για κάθε δεδομένο, που περιέχει την τιμή που αντιστοιχεί στην ομάδα που ανήκει το δεδομένο αυτό βάσει του δοθέντος διαχωρισμού. Εφαρμόζοντας την ave στα ίδια αντικείμενα με παραπάνω:

Δοκιμάστε:	**Σχόλιο**
`ave(p[,1],p[,2])`	Ο αριθμητικός μέσος όρος των p[,1] κάθε ομάδας.
`ave(p[,1],p[,2],FUN = "sum")`	Το άθροισμα των δεδομένων κάθε ομάδας.

Έτσι στη δεύτερη εντολή τα στοιχεία της 1ης στήλης του p ομαδοποιήθηκαν ως προς τη 2η, εφαρμόστηκε η sum και επιστράφηκε το αποτέλεσμα που αντιστοιχεί σε κάθε στοιχείο, δηλαδή:

```
> ave(p[,1],p[,2],FUN = "sum")
[1] 7 7 3 7
```

Συναρτήσεις όπως οι παραπάνω (outer, apply, sweep κλπ.) καλούνται με την τιμή κάποιας παραμέτρου τους να είναι κάποια άλλη συνάρτηση την οποία χρησιμοποιούν εσωτερικά για την επεξεργασία. Αυτό είναι μέρος ενός γενικότερου προγραμματιστικού παραδείγματος που υποστηρίζει η R και αποκαλείται functional programming (για περισσότερα βλ.§5.4 Η συναρτησιακή προσέγγιση στον πραγματικό κόσμο).

4.2 Συνήθεις μη-ατομικοί τύποι αντικειμένων

Με εξαίρεση τους βασικούς τύπους (numeric, character, logical κλπ.) καθώς και τύπους που σχετίζονται άμεσα με αυτούς, όπως όσοι αναφέρθηκαν στη σχετική ενότητα (βλ.§4.1 Συνήθεις ατομικοί τύποι αντικειμένων), οι περισσότεροι άλλοι τύποι αντικειμένων είναι μη-ατομικοί. Οι τύποι αυτοί ονομάζονται και recursive (αναδρομικοί) και μπορούν να περιέχουν αντικείμενα διαφορετικών τύπων (συμπεριλαμβανομένου και ίδιου τύπου με αυτόν που τα περιέχει). Αυτό επιτρέπει υποστήριξη διάφορων λειτουργικοτήτων καθώς και την υλοποίηση διάφορων δομών δεδομένων. Ένα χαρακτηριστικό που μοιράζονται οι μη-ατομικοί τύποι αντικειμένων είναι πως υποστηρίζουν τον τελεστή '**$**' με τον οποίο μπορεί να προσδιοριστεί ένα συγκεκριμένο στοιχείο τους βάσει του ονόματός του (εφόσον έχουν οριστεί names). Ο έλεγχος αν ένα αντικείμενο ανήκει σε μη-ατομικό τύπο γίνεται με τη συνάρτηση **is.recursive**.

4.2.1 Ο τύπος list (λίστα)

Η δομή λίστας αξιοποιείται εκτενώς στα πακέτα της R και έχουν ήδη αναφερθεί, σε προηγούμενες ενότητες, συναρτήσεις, οι οποίες δέχονται ως παραμέτρους ή επιστρέφουν αντικείμενα τύπου list (π.χ. οι dimnames, tapply, aggregate κ.α.). Τα αντικείμενα τύπου list (λίστα) είναι διανύσματα (vector, βλ.§4.1.1 Ο τύπος vector (διάνυσμα)), αλλά δεν είναι σε atomic mode. Συγκεκριμένα, ο τύπος list είναι ένα ειδικό mode των vector που επεκτείνει τον τύπο αυτό. Η βασική διαφορά αυτού του mode από τα απλά (atomic) vector είναι πως ένα list μπορεί να περιέχει στοιχεία οποιουδήποτε τύπου και ο τύπος των στοιχείων αυτών δεν χρειάζεται να είναι ενιαίος. Επιπρόσθετα ο τύπος list, (ως recursive) μπορεί να περιέχει στοιχεία που είναι και αυτά list. Επειδή είναι είδος vector, οι λίστες έχουν μήκος (συνάρτηση length για τον αριθμό των στοιχείων τους) και δεν έχουν διάσταση (dim). Ένας τρόπος να δημιουργηθεί ένα list είναι με τη συνάρτηση vector, θέτοντας ως mode στη σχετική παράμετρο την τιμή "list". Έτσι, ένα list μπορεί να δημιουργηθεί κενό (και να προστεθούν στοιχεία αργότερα) ή και με προκαθορισμένο αριθμό στοιχείων, οι τιμές όμως των οποίων δεν έχουν οριστεί (είναι αρχικά NULL).

Δοκιμάστε:	**Σχόλιο**
`j<-vector(mode="list",5)`	Λίστα j με 5 στοιχεία, όλα αρχικά NULL.

`is.list(j)`	Είναι το j λίστα; Επιστρέφει TRUE.
`is.vector(j)`	Είναι το j διάνυσμα; Οι λίστες είναι διανύσματα, άρα επιστρέφει TRUE.

Άλλος τρόπος δημιουργίας μιας λίστας που επίσης έχει αναφερθεί στην ενότητα για τα διανύσματα, είναι μέσω της συνάρτησης **c**. Η **c** επιστρέφει ένα αντικείμενο τύπου vector αν όλα τα δεδομένα που της έχουν δοθεί ανήκουν σε βασικό τύπο (numeric, character, logical κλπ.) και θέτει το vector στο αντίστοιχο atomic mode. Αν όμως δεν είναι δυνατή η μετατροπή (που αναφέρεται και ως coercion, αναγκασμός) των δεδομένων σε ένα atomic διάνυσμα κάποιου ενιαίου βασικού τύπου, τότε η c τα συγχωνεύει σε vector με mode “list”, δηλαδή σε ένα αντικείμενο τύπου list. Προφανώς, η μετατροπή άλλων αντικειμένων σε λίστα γίνεται με τη συνάρτηση **as.list** ή με την **as.vector** (με παράμετρο mode="list"), ενώ η **is.list** ελέγχει αν ένα αντικείμενο είναι τύπου list. Ο βασικός όμως τρόπος να δημιουργηθεί μια λίστα είναι με τη σχετική συνάρτηση **list**:

Δοκιμάστε:	**Σχόλιο**
`j<-list(5,c(11,21,23),TRUE,10,"Athens")`	Λίστα j, με διάφορους τύπους στοιχείων.
`length(j)`	Ο αριθμός των στοιχείων στο j, επιστρέφει 5.
`dim(j)`	Οι λίστες δεν έχουν διάσταση, επιστρέφει NULL

Η παραπάνω λίστα περιέχει διαφόρων τύπων στοιχεία: ένα numeric, ένα διάνυσμα, ένα logical, ένα (ακόμα) numeric και ένα character. Παρατηρήστε πως το μήκος (length) δεν προσμετρά στον αριθμό των στοιχείων της λίστας τα στοιχεία των αντικειμένων που ενδεχομένως έχουν μήκος μεγαλύτερο του 1 (δηλαδή τα δικά τους στοιχεία). Τα αντικείμενα ενσωματώνονται στη λίστα αλλά διατηρούν τον τύπο τους και τη δομή τους. Έτσι, η παραπάνω λίστα j έχει 5 στοιχεία γιατί τα 3 στοιχεία που περιέχονται στο διάνυσμα (στη 2^{η} θέση του j) ανήκουν σε αυτό το αντικείμενο και όχι στη λίστα.

Κατά ακολουθία όσων ισχύουν γενικότερα για τα vector, στον τύπο list ο τελεστής αγκυλών ([]) επιλέγει στοιχεία. Και εδώ, τα επιλεγμένα στοιχεία συμπεριφέρονται σαν να βρίσκονται μόνα τους σε έναν ίδιο τύπο αντικειμένου (εδώ list) αν και παραμένουν στο αρχικό, ενώ αν ανακληθούν οι τιμές τους επιστρέφονται πάλι μέσα σε list. Έτσι, κατά την ανάκληση στοιχείων από μια λίστα, αν οριστεί εντός των αγκυλών μια μοναδική θέση, επιστρέφεται ένα list που περιέχει μόνο το στοιχείο στη συγκεκριμένη θέση. Αντίστοιχα, για την αντικατάσταση ενός στοιχείου που προσδιορίστηκε με αυτόν τον τρόπο (μέσω αγκυλών []), η νέα τιμή θα πρέπει να δίνεται ως list (ή να είναι αντικείμενο μετατρέψιμο με σαφή τρόπο σε list). Συνεχίζοντας από το προηγούμενο παράδειγμα:

Δοκιμάστε:	**Σχόλιο**
`j[4:5]`	Λίστα με τα στοιχεία στην 4^{η} και 5^{η} θέση (10 και "Athens").
`j[c(T,F,F,T,T)]`	Λίστα με τα στοιχεία στην 1^{η}, 4^{η} και 5^{η} θέση (5, 10 και "Athens").
`j[3]`	Λίστα με το στοιχείο στην 3^{η} θέση (logical τιμή TRUE).
`j[3]<-list(matrix(0,2,2))`	Αντικατάσταση στοιχείου στην 3^{η} θέση, με πίνακα 2x2.
`j[5]<-"Limon"`	Αντικατάσταση στοιχείου στην 5^{η} θέση με την τιμή "Limon".
`j[4:5]<-list(3,"Havana")`	Αντικατάσταση 4^{ης} και 5^{ης} θέσης με τιμές 3 και "Havana" αντίστοιχα.

Όταν ανακληθεί η παραπάνω λίστα j, εμφανίζεται:

```
> j
[[1]]
[1] 5

[[2]]
[1] 11 21 23

[[3]]
     [,1] [,2]
[1,]    0    0
[2,]    0    0

[[4]]
[1] 3
```

```
[[5]]
[1] "Havana"
```

Κατά την εμφάνιση ενός list, όπως παραπάνω, οι αριθμοί μέσα στις διπλές αγκύλες ([[1]], [[2]] κλπ.) συμβολίζουν τα αντίστοιχα αντικείμενα που περιέχονται στη λίστα (1ο, 2ο κλπ.). Ο τελεστής διπλών αγκυλών [[ ]] δίνει άμεση πρόσβαση στο αντικείμενο της αντίστοιχης θέσης. Μέσα σε διπλές αγκύλες συνήθως ορίζεται μόνο μία τιμή δείκτη, άρα η πρόσβαση αφορά μόνο ένα αντικείμενο. Αν δοθεί διάνυσμα αντί ενός δείκτη, αυτό προκαλεί πρόσβαση με ανάδρομή (recursion), π.χ. το [[c(2,4)]] σημαίνει «το 4ο αντικείμενο που περιέχεται στο 2ο αντικείμενο της λίστας». Για περισσότερα σχετικά με την πρόσβαση μέσω δεκτών σε αντικείμενα διαφόρων τύπων (μεταξύ αυτών και list) βλ. [54], ενότητα Indexing. Μερικά παραδείγματα, συνεχίζοντας από το προηγούμενο:

Δοκιμάστε:	**Σχόλιο**
`j[1]+100`	Λάθος, καθώς το j[1] είναι list (που περιέχει το 3).
`j[[1]]+100`	Ανάκληση του 1ου στοιχείου της λίστας[223] (με τιμή 5), επιστρέφει 105.
`j[[1]]<-100`	Αντικατάσταση του 1ου στοιχείου της λίστας με το 100 (αντί του 5).
`length(j[[2]])`	Μήκος του 2ου στοιχείου της λίστας (που είναι διάνυσμα), επιστρέφει 3.
`j[[2]][3]<-25`	Στο 2ο στοιχείο (που είναι διάνυσμα) αντικατάσταση του 3ου στοιχείου του με 25 (αντί 23).

Στα στοιχεία της λίστας μπορούν να δοθούν ονόματα μέσω της συνάρτησης names, με τον ίδιο τρόπο που ισχύει για τα απλά διανύσματα. Επίσης, υποστηρίζεται η ονομασία των στοιχείων κατά τη δημιουργία του list. Για παράδειγμα, ο παρακάτω ορισμός είναι ίδιος με αυτόν του προηγούμενου παραδείγματος αλλά δίνει ονόματα ("day","guest","on" κλπ.) στα στοιχεία του j που δημιουργείται:

```
j<-list(day=5, guest=c(11,21,23), on=T, tzn=10, loc="Athens")
```

Το παραπάνω αποτέλεσμα μπορεί να επιτευχθεί και με τη συνάρτηση names που δίνει πρόσβαση στα ονόματα των στοιχείων ενός αντικειμένου.

Τέλος, χρήσιμες είναι οι συναρτήσεις **unlist** και **split**. Η unlist κάνει μετατροπή των δεδομένων ενός list[224] σε έναν κοινό τύπο (μέσω coercion) και τα τοποθετεί σε ένα vector το οποίο επιστρέφει. Για παράδειγμα, αν εφαρμοστεί η unlist στο παραπάνω αντικείμενο j, επιστρέφει το παρακάτω atomic vector σε character mode:

```
> unlist(j)
    day   guest1   guest2   guest3      on      tzn      loc
    "5"     "11"     "21"     "23"  "TRUE"     "10" "Athens"
```

Η spit αντιγράφει τα στοιχεία που υπάρχουν σε άλλα αντικείμενα (τύπων που την υποστηρίζουν όπως vector και matrix, data.frame κλπ.) σε ένα list, χωρίζοντάς τα βάσει των τιμών ενός διαχωριστή. Ένα παράδειγμα θα διευκολύνει την κατανόηση της:

```
v1 <- c(2,3,5,6,7,3,2,0)
v2 <- c("a","a","b","c","c","a","b","c")
L1 <- split(v1,v2)
```

Η παραπάνω split χωρίζει τις τιμές του v1 με βάση τις αντίστοιχες τιμές στο v2 και επιστρέφει το αποτέλεσμα σε ένα list που εδώ ονομάστηκε L1. Όνομα των στοιχείων του L1 είναι οι τιμές διαχωρισμού:

```
> L1
$a
[1] 2 3 3

$b
[1] 5 2

$c
[1] 6 7 0
```

[223] Η ανάκληση ενός στοιχείου από αντικείμενα διαφόρων τύπων μπορεί να γίνει και με τη συνάρτηση getElement, δηλαδή εδώ χρησιμοποιώντας getElement(j,1) αντί του j[[1]].
[224] Η unlist είναι generic συνάρτηση και υποστηρίζεται και από άλλους τύπους αντικειμένων εκτός του τύπου list.

Στο επόμενο παράδειγμα της split γίνεται χρήση κριτηρίου για τον διαχωρισμό. Στο list που επιστρέφεται από τη split (και εδώ ανατίθεται στη μεταβλητή L2) υπάρχουν δύο στοιχεία, ένα με όνομα `FALSE` για τα στοιχεία του v1 που δεν ικανοποιούν το κριτήριο και ένα ακόμα με όνομα `TRUE` για τα υπόλοιπα.

```
> L2 <- split(v1,v1>4)
> L2
$`FALSE`
[1] 2 3 3 2 0

$`TRUE`
[1] 5 6 7
```

4.2.1.1 Ο τελεστής επιλογής $

Τα ονόματα που ανατίθενται σε στοιχεία ενός list αλλά και γενικότερα ενός recursive τύπου αντικειμένων, επιτρέπουν τον προσδιορισμό ενός στοιχείου με το όνομά του. Αυτό γίνεται με χρήση του τελεστή **$**. Έτσι, ένα στοιχείο με όνομα "day" σε ένα (recursive τύπου) αντικείμενο j αναφέρεται ως j$day. Στο παράδειγμα του αντικειμένου τύπου list με όνομα j που ορίστηκε αμέσως παραπάνω με ονόματα για τα στοιχεία του, μπορούν να γίνουν τα ακόλουθα:

Δοκιμάστε:	**Σχόλιο**
`j[[4]]+100`	Ανάκληση του 4ου στοιχείου της λίστας + 100, επιστρέφει 110.
`j[["tzn"]]+100`	Ίδιο με το προηγούμενο. Ανάκληση του στοιχείου με όνομα "tzn" + 100, επιστρέφει 110.
`j$tzn+100`	Ίδιο με το προηγούμενο, με χρήση όμως τελεστή $, επιστρέφει 110.
`x<-"tzn"`	Το όνομα ενός στοιχείου της λίστας (“tzn”) σε μεταβλητή x.
`j[[x]]+100`	Προσδιορισμός ονόματος μέσω της μεταβλητής x, επιστρέφει 110 (ίδιο με j$tzn+100).
`j$tz+100`	Προσδιορισμός του tzn με μέρος του ονόματος (partial matching), επιστρέφει 110.
`j[["tz",exact="F"]]`	Ίδιο με το παραπάνω.
`j[[2]][3]`	Το 3ο στοιχείο του 2ου στοιχείου της λίστας, επιστρέφει 23.
`j[[2]][[3]]`	Ίδιο με το παραπάνω, επιστρέφει 23.
`j[[c(2,3)]]`	Ίδιο με το παραπάνω, προσδιορισμός με αναδρομικό τρόπο (recursive), επιστρέφει 23.
`j[[4]]<-12`	Αντικατάσταση 4ου στοιχείου της λίστας j με το 12.
`j[["tzn"]]<-12`	Ίδιο με το προηγούμενο. Αντικατάσταση του στοιχείου με όνομα "tzn" με το 12.
`j$tzn<-12`	Ίδιο με το προηγούμενο, πρόσβαση στο στοιχείο tzn του j με χρήση του τελεστή $.
`j$guest[3]<-5`	Αντίστοιχα, πρόσβαση με $ στο διάνυσμα guest και αλλαγή του 3ου στοιχείου του σε 5.

4.2.1.2 Επεξεργασία αντικειμένων τύπου list

Ο χειρισμός αντικειμένων τύπου list είναι συχνά παρόμοιος με των απλών (σε atomic mode) vector, αλλά υπάρχουν και διαφορές. Επέκταση ενός list μπορεί να γίνει με ανάθεση τιμής σε θέση πέραν του τρέχοντος μήκους (length) του. Οι κενές νέες θέσεις θα πάρουν την τιμή NULL. Όμως, η ανάθεση NULL σε θέσεις του list διαγράφει τα στοιχεία στις θέσεις αυτές και τα αφαιρεί από τη λίστα, μειώνοντας έτσι το μήκος της λίστας:

Δοκιμάστε:	**Σχόλιο**
`j<-list(10,20)`	Ανάθεση στο j λίστας με 2 numeric στοιχεία: 10 και 20.
`j[4]<-40`	Επέκταση σε θέση πέραν του μήκους, το j περιέχει τα στοιχεία: 10, 20, NULL και 40.
`j[2:3]<-NULL`	Ανάθεση NULL σε θέσεις 2 και 3. Διαγράφονται τα στοιχεία, το j περιέχει 10 και 40.

Στο παράδειγμα η διαγραφή στοιχείων έγινε με ανάθεση NULL στα στοιχεία που ζητείται να αφαιρεθούν. Εναλλακτικά, μπορεί να εφαρμοστεί η γνωστή (από τα atomic vector) σύνταξη επιλογής με δείκτες. Για παράδειγμα, η εντολή j <- j[-c(2,3)] είναι λειτουργικά ισοδύναμη με την εντολή j[2:3]<-NULL, παραπάνω. Και οι δύο εντολές έχουν ως αποτέλεσμα τη διαγραφή των στοιχείων στις θέσεις 2 και 3 από τη λίστα j.

Η συνάρτηση c επιτρέπει τη συγχώνευση πολλών list (και άλλων αντικειμένων) σε μια λίστα, ενώ αν δημιουργηθεί μια λίστα με την εντολή list κατά την οποία κάποιο δεδομένο προς ένταξη στη λίστα είναι list, τότε θα προκύψει ένα list που περιέχει άλλο list:

Δοκιμάστε:	Σχόλιο
`j<-list(10,20)`	Ανάθεση στο j λίστας με 2 numeric στοιχεία: 10 και 20.
`c("q",j)`	Συγχωνεύει το "q" με τη λίστα j, επιστρέφει list με 3 στοιχεία: "q", 10 και 40.
`c(j,"q")`	Συγχωνεύει τη λίστα j με το "q", επιστρέφει list με 3 στοιχεία: 10, 40 και "q".
`c(j,j)`	Συγχωνεύει τη λίστα j με τη λίστα j, επιστρέφει list με 4 στοιχεία: 10, 40, 10 και 40.
`list(j,j)`	Επιστρέφει λίστα με 2 στοιχεία, καθένα από αυτά είναι λίστα με τιμές 10 και 40.

Κατά τη συγχώνευση λιστών, αν στις λίστες αυτές υπάρχουν ονόματα στοιχείων που τυχαίνει να είναι ίδια, τότε το list που θα προκύψει θα περιέχει πάνω από ένα στοιχεία με το ίδιο όνομα. Αυτό επιτρέπεται, αλλά δυσκολεύει τη χρήση του $ καθώς ο τελεστής αυτός επιλέγει πάντα το πρώτο στοιχείο στη λίστα με το δοθέν όνομα. Σε κάθε περίπτωση, σε χειρισμούς αλλαγής, συγχώνευσης, διαγραφής επέκτασης κλπ. που οδηγούν στη δημιουργία νέου αντικειμένου list, ο τύπος συχνά εφαρμόζει «ρηχή αντιγραφή» (shallow copy)[225] του αρχικού αντικειμένου, καθώς αυτή η μέθοδος δημιουργίας ενός αντιγράφου είναι υπολογιστικά αποδοτικότερη. Παρόλα αυτά, όταν εφαρμόζονται σε αντικείμενα list με πολύ μεγάλο αριθμό στοιχείων καλό είναι να λαμβάνονται υπόψη όσα έχουν γραφεί για τον υπολογιστικό φόρτο στη σχετική ενότητα με τα vectors.

4.2.1.3 Εφαρμογή συναρτήσεων σε list

Οι λίστες δεν είναι ατομικοί τύποι με ενιαίο τύπο στοιχείων. Για τον λόγο αυτό δεν επιτρέπουν αυτόματα τη μαζική εφαρμογή συναρτήσεων στα στοιχεία τους. Για παράδειγμα, ας ορίσουμε μια λίστα j1 που τυχαίνει να περιέχει αποκλειστικά numeric στοιχεία:

```
j1<-list(1,2,3)
```

Παρόλο που όλα τα στοιχεία του j1 είναι numeric, το αντικείμενο αυτό δεν βρίσκεται σε “numeric mode” αλλά σε “list mode”. Η κατάσταση αυτή δεν επιτρέπει να γίνουν αριθμητικές πράξεις πάνω στο αντικείμενο. Έτσι το j1+10 δεν εκτελείται (όπως θα γινόταν σε αντίστοιχο vector), αλλά εγείρει λάθος κατά την εκτέλεση. Πώς λοιπόν μπορεί να γίνει μια τέτοια αριθμητική πράξη (ή να εκτελεστεί οποιαδήποτε άλλη συνάρτηση) στα στοιχεία της λίστας j1; Οι εναλλακτικές επιλογές (όλες οδηγούν στην αύξηση κατά 10 της τιμής των στοιχείων του j1, άρα έχουν ως αποτέλεσμα το j1 να παραμένει λίστα με τρία στοιχεία, τα 11, 12, και 13) είναι οι ακόλουθες:

Επιλογή 1: Η χρήση κάποιου βρόχου για δημιουργία ενός δείκτη που θα διατρέξει τα στοιχεία εφαρμόζοντας τη συνάρτηση (εδώ πρόσθεση). Η πρόσβαση στα στοιχεία γίνεται με τον τελεστή διπλών αγκυλών [[ ]]:

```
for(i in 1:length(j1) j1[[i]]<- j1[[i]] + 10
```

Επιλογή 2: Η αλλαγή του mode από list σε κατάλληλο atomic (εδώ “numeric”) vector και εκτέλεση της συνάρτησης (πρόσθεσης). Προφανώς αφορά περιπτώσεις που για όλα τα στοιχεία υπάρχει κατάλληλο κοινό atomic mode που υποστηρίζει τη συνάρτηση η οποία πρέπει να εφαρμοστεί. Σε κάθε περίπτωση, παρακάτω στην 1η εντολή δημιουργεί ένα vector με τις τιμές στο j1 και εκτελεί την πρόσθεση επιστρέφοντας το αποτέλεσμα (vector) στο j1. Ακολούθως (στη 2η εντολή) το j1 μετατρέπεται πάλι από vector σε list (η ίδια μετατροπή θα μπορούσε να γίνει με την εντολή mode(j1)<- "list").

```
j1<-as.vector(j1,"numeric") + 10
j1<-as.list(j1)
```

Επιλογή 3: Η αξιοποίηση της προσέγγισης που επιτρέπουν οι συναρτήσεις της οικογένειας apply (βλ.§4.1.3.4 Η οικογένεια συναρτήσεων apply) και συγκεκριμένα η συνάρτηση **lapply**. Με τη συνάρτηση lapply μπορεί να γίνει εφαρμογή οποιαδήποτε συνάρτησης (συμπεριλαμβανομένων και συναρτήσεων ορισμένων από τον χρήστη[226]) σε κάθε στοιχείο ενός αντικειμένου τύπου vector ή list (ή τύπου μετατρέψιμου σε αυτά). Όπως και άλλες συναρτήσεις apply, η lapply δέχεται ως παραμέτρους το αντικείμενο και την συνάρτηση (FUN) που θα εκτελεστεί για την επεξεργασία των στοιχείων του, ενώ επιπρόσθετες παράμετροι θα περάσουν στη συνάρτηση που ορίζεται στην παράμετρο FUN. Έτσι, αν ζητούμενο είναι να υπολογιστούν όλες οι τετραγωνικές ρίζες των στοιχείων του j1, αρκεί η εντολή lapply(j1,sqrt), ενώ η ζητούμενη αύξηση των τιμών των στοιχείων κατά 10 γίνεται με:

---

[225] Αντιγράφονται οι αναφορές στα αντικείμενα και όχι τα ίδια τα αντικείμενα. Σχετικά με τη «ρηχή αντιγραφή» (shallow copy) βλ.§3.1.4 Εργαλεία προφίλ (profiling).

[226] βλ.§5.1.1 Δημιουργία συναρτήσεων.

```
j1<-lapply(j1,"+",10)
```

Στην περίπτωση που πρέπει να συνδυαστούν στοιχεία από περισσότερα του ενός αντικειμένου τύπου list, μπορεί να εφαρμοστεί η συνάρτηση **mapply** ή η συνάρτηση **Map**. Οι συναρτήσεις αυτές επίσης επιτρέπουν την εκτέλεση οποιασδήποτε συνάρτησης, με ορίσματα τα στοιχεία σε αντίστοιχες θέσεις δύο ή περισσότερων αντικειμένων (vector, list ή τύπων μετατρέψιμων σε αυτά). Παρεμφερής με την lapply παραπάνω και άλλες συναρτήσεις apply που έχουν ήδη αναφερθεί, η mapply δέχεται στην 1η παράμετρό της FUN τη συνάρτηση που θα εφαρμόσει, ακολουθούμενη από τα αντικείμενα με τα οποία θα γίνει η επεξεργασία, ενώ στην παράμετρο MoreArgs δέχεται λίστα με τυχόν πρόσθετες παράμετρους προς την συνάρτηση FUN. Τέλος, η παράμετρος SIMPLIFY ορίζει αν το επιστρεφόμενο αποτέλεσμα θα είναι ίδιου τύπου με τα δοθέντα ή θα απλοποιηθεί σε άλλον τύπο (π.χ μετατροπή list αποτελεσμάτων σε απλό vector). Κατά την κλήση της όποιας FUN από την mapply, τα δεδομένα που θα περαστούν την πρώτη φορά αποτελούνται από το 1ο στοιχείο κάθε αντικειμένου, μετά από το 2ο στοιχείο και ούτω καθεξής, ενώ, αν χρειάζεται, η mapply εφαρμόζει και ανακύκλωση.

Στο επόμενο παράδειγμα, έχουμε τις παρακάτω δύο λίστες με αποκλειστικά numeric στοιχεία:

```
j1<-list(1,2,3)
j2<-list(10,20,30)
j3<-list(100,200,300)
```

Ζητούμενο είναι να υπολογιστεί το άθροισμα των στοιχείων στις αντίστοιχες θέσεις και να αποθηκευτούν τα αποτελέσματα σε τρίτο list με όνομα js. Το αποτέλεσμα θα υπολογιζόνταν εύκολα με την εντολή js <- j1+j2+j3 αν τα αντικείμενα ήταν απλά vector σε numeric mode. Καθώς όμως είναι τύπου list, αυτό δεν επιτρέπεται. Μπορεί να επιτευχθεί με την παρακάτω εντολή:

```
js<-mapply(FUN=sum,j1,j2,j3,SIMPLIFY = F)
```

Η παραπάνω εντολή θα αποθηκεύσει στο js λίστα τριών στοιχείων (με τιμές 111, 222 και 333). Εναλλακτικά, θα μπορούσε να χρησιμοποιηθεί η συνάρτηση Map που είναι ισοδύναμη με την εκτέλεση της mapply για παράμετρο SIMPLIFY=FALSE (δηλαδή διατηρεί τον τύπο των δεδομένων στα αποτελέσματα). Έτσι, ίδια με την παραπάνω είναι η εντολή:

```
js<-Map(sum,j1,j2,j3)
```

Παρατηρήστε πως ως FUN χρησιμοποιήθηκε η συνάρτηση sum και όχι ο τελεστής "+". Αυτό έγινε απλώς επειδή ο τελεστής δέχεται μόνο δύο ορίσματα (και εδώ έχουμε τρείς λίστες με αριθμούς, άρα τρεις αριθμούς που πρέπει να προστεθούν κάθε φορά). Καθώς οι mapply και Map μπορούν να δεχτούν οποιαδήποτε συνάρτηση, είναι ιδιαίτερα χρήσιμες για την επεξεργασία συνδυασμών στοιχείων από τις αντίστοιχες θέσεις δύο ή περισσότερων αντικειμένων vector ή list. Για παράδειγμα, η παρακάτω εντολή καλεί τη συνάρτηση c, οπότε συνθέτει μια λίστα όπου κάθε στοιχείο είναι vector με τις τιμές στις αντίστοιχες θέσεις (δηλαδή λίστα με 3 στοιχεία, τα διανύσματα c(1,10,100), c(2,20,200) και c(3,30,300)):

```
Map(c,j1,j2,j3)
```

Συνδυάζοντας το παραπάνω με την προαναφερθείσα συνάρτηση lapply, μπορεί να γίνει μαζική επεξεργασία σε λίστες. Για παράδειγμα, για να βρεθεί ο αριθμητικός μέσος όρος των στοιχείων στις αντίστοιχες θέσεις των τριών λιστών αρκεί η εντολή:

```
lapply( Map(c,j1,j2,j3), "mean")
```

Εδώ, αφού η Map δημιουργήσει λίστα η οποία περιέχει τρία διανύσματα (τα οποία με τη σειρά τους περιέχουν τα στοιχεία στις αντίστοιχες θέσεις των λιστών j1, j2 και j3), εκτελέστηκε με την lapply η εντολή mean πάνω στη λίστα αυτή για εύρεση του μέσου όρου κάθε στοιχείου της. Το αποτέλεσμα επιστρέφεται επίσης σε αντικείμενο list τριών στοιχείων (των οποίων οι τιμές είναι 37, 74 και 111).

Περισσότερα σχετικά με την διανυσματοποίηση (δηλαδή την εφαρμογή συναρτήσεων σε επίπεδο αντικείμενου και όχι των στοιχείων του) περιγράφονται στην §5.4 Η συναρτησιακή προσέγγιση στον πραγματικό κόσμο.

4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων

Η παρούσα ενότητα περιγράφει σύντομα μια δομή που είναι παρεμφερής με λίστα και διατηρείται εσωτερικά σε κάθε αντικείμενο με σκοπό να καταγράφει τις τρέχουσες ιδιότητές του. Το αντικείμενο χρησιμοποιεί όσες από τις ιδιότητες έχουν οριστεί σε αυτό, εφόσον αναγνωρίζονται και είναι αξιοποιήσιμες από τον τρέχοντα τύπο του. Οι ιδιότητες είναι ουσιαστικά τιμές στις οποίες έχει δοθεί ένα όνομα Αν και παρεμφερής, η δομή ιδιοτήτων των αντικειμένων δεν είναι τύπου list. Είναι ένα σύνολο ονομασμένων στοιχείων χωρίς θέση[227].

[227] Μοιάζει στο τομέα αυτόν με τον τύπο environment που περιγράφεται παρακάτω, βλ. §4.2.2 Ο τύπος environment

Όμως πλήρης αντικατάσταση των τρεχουσών ιδιοτήτων ενός αντικειμένου γίνεται με ανάθεση τιμών από ένα αντικείμενο list, ενώ σε list αντιγράφεται η δομή των ιδιοτήτων ενός αντικειμένου αν ανακληθεί από αυτό. Σε κάθε περίπτωση, η εξέταση των ιδιοτήτων που διατηρούν εσωτερικά τα αντικείμενα είναι χρήσιμη στην κατανόηση της συμπεριφοράς τους.

Έχουμε ήδη κάνει χρήση ιδιοτήτων αντικειμένων αγνοώντας τη δομή που τις διατηρεί. Ο συνήθης τρόπος πρόσβασης ή/και αλλαγής ιδιοτήτων ενός αντικειμένου είναι με σχετικές συναρτήσεις που παρέχει και υποστηρίζει ο τύπος του. Για παράδειγμα, συναρτήσεις όπως η names (που αφορά τα ονόματα των στοιχείων ενός vector) ή η dim (που αφορά τις διαστάσεις ενός array) ή η class (που αφορά την κλάση, δηλαδή τον τύπο ή τους τύπους στους οποίους ανήκει το αντικείμενο) ουσιαστικά δίνουν πρόσβαση στις σχετικές ιδιότητες του αντικειμένου. Στις περιπτώσεις των συγκεκριμένων συναρτήσεων (names και dim) οι ιδιότητες του αντικειμένου που επηρεάζουν τυχαίνει να έχουν το ίδιο όνομα με τις συναρτήσεις, αλλά αυτό δεν ισχύει πάντα, ούτε υπάρχουν πάντα τέτοιες συναρτήσεις που αντιστοιχούν άμεσα με ιδιότητες (π.χ. δεν υπάρχει συνώνυμη συνάρτηση για την ιδιότητα srcref που αποθηκεύει τον κώδικα με τον οποίο δημιουργήθηκε ένα αντικείμενο τύπου function). Έτσι, για πρόσβαση σε κάποια ιδιότητα ενός αντικειμένου υπάρχουν συναρτήσεις όπως η **attr** που δέχεται ως παράμετρο το όνομα του αντικειμένου και το όνομα τις ιδιότητας. Για παράδειγμα:

Δοκιμάστε:	**Σχόλιο**
`a<-c(10,20)`	Ένα vector με δυο στοιχεία.
`attr(a,"names")<-c("Y","Z")`	Προσθήκη ιδιότητας names, ίδιο με την εντολή names(a)<-c("Y","Z").
`attr(a,"names")`	Επιστρέφει την ιδιότητα names, ίδιο με την εντολή names(a).

Επιπρόσθετα, πρόσβαση σε ολόκληρη την εσωτερική «λίστα» ιδιοτήτων ενός αντικειμένου μπορεί να γίνει με τη συνάρτηση **attributes**, επιτρέποντας επιπλέον και την επιλογή μίας συγκεκριμένης ιδιότητας με τον τελεστή $ και το όνομα της ιδιότητας αυτής (όπως έγινε παραπάνω με τα ονόματα στοιχείων των lists, βλ.§4.2.1.1 Ο τελεστής επιλογής $). Άρα, ίδιο αποτέλεσμα με την εντολή names(a)<-c("Y","Z") έχει και η εντολή:

```
attributes(a)$names<-c("Y","Z")
```

Η συνάρτηση attributes επιτρέπει την συνολική αντικατάσταση ή αντιγραφή όλων των ιδιοτήτων ενός αντικειμένου. Μια εικόνα της σημασίας των ιδιοτήτων σε ένα αντικείμενο δίνει το επόμενο παράδειγμα:

```
x<-c(10,20,30,40)
attributes(x)<-list(names=LETTERS[1:4],
                    dim=c(2,2),
                    dimnames=list(c("Γ1","Γ2"),c("Σ1","Σ2")))
```

Στον κώδικα αυτόν, αρχικά καταχωρίζεται στο x ένα απλό vector (σε numeric mode). Ακολούθως, προστίθενται σε αυτό μια σειρά από ιδιότητες μέσω της συνάρτησης attributes. Οι ιδιότητες αυτές αντιγράφονται από ένα list και περιλαμβάνουν ονόματα για τα στοιχεία (ιδιότητα names)[228], διαστάσεις (ιδιότητα dim), και ονόματα για την 1η και 2η διάσταση του αντικειμένου (ιδιότητα dimnames). Ανακαλώντας τις ιδιότητες του x, με την εντολή attributes(x) εμφανίζεται η τρέχουσα λίστα ιδιοτήτων:

```
> attributes(x)
$dim
[1] 2 2

$names
[1] "a" "b" "c" "d"

$dimnames
$dimnames[[1]]
[1] "Γ1" "Γ2"

$dimnames[[2]]
```

(περιβάλλον).

[228] Εδώ στα 4 στοιχεία δίνονται τα ονόματα "A","B","C" και "D". Το πακέτο 'base' της R παρέχει κάποιες σταθερές, μεταξύ αυτών πίνακες ονομάτων μηνών (στα Αγγλικά) και λατινικών χαρακτήρων. Έτσι το LETTERS[1:4] είναι οι 4 πρώτοι κεφαλαίοι λατινικοί χαρακτήρες, δηλαδή το c("A","B","C","D"), βλ. help(Constants).

```
[1] "Σ1" "Σ2"
```

Ανακαλώντας το αντικείμενο στο x, είναι ορατό το αποτέλεσμα της προσθήκης των παραπάνω ιδιοτήτων:

```
> x
   Σ1 Σ2
Γ1 10 30
Γ2 20 40
attr(,"names")
[1] "A" "B" "C" "D"
```

Συγκεκριμένα, η προσθήκη της ιδιότητας dim, άλλαξε τον τύπο αντικειμένου x από vector σε matrix, κάτι που μπορεί να επιβεβαιωθεί με τη σχετική εντολή is.matrix(x) η οποία επιστρέφει TRUE. Ως matrix πλέον, το x χρησιμοποιεί την άλλη ιδιότητα που προστέθηκε, την dimnames για ονόματα γραμμών και στηλών (όπως φαίνεται και παραπάνω). Καθώς στον τρέχοντα τύπο του x η ιδιότητα names δεν αξιοποιείται, η ιδιότητα αυτή απλά εμφανίζεται στο τέλος. Αν όμως γίνει πρόσβαση στο x ως vector (κάτι που επιτρέπεται, βλ.§4.1.3 Πίνακες (matrix και array)) τότε η ιδιότητα names (τα ονόματα των στοιχείων) αποκτά χρησιμότητα και αξιοποιείται:

```
> x[1:4]
 A  B  C  D
10 20 30 40
```

Η συνάρτηση attributes δίνει προφανώς και τη δυνατότητα αντιγραφής των ιδιοτήτων ενός αντικειμένου σε άλλο:

```
y<-c(15,25,35,45)
attributes(y)<- attributes(x)
```

Με τον παραπάνω κώδικα, όλες οι τρέχουσες ιδιότητες του αντικειμένου στο x αντικαθιστούν τις ιδιότητες ενός άλλου αντικειμένου y (που αρχικά είναι τύπου vector). Αυτό προκαλεί στο y τα ίδια αποτελέσματα που είχε η προσθήκη των ιδιοτήτων στο x, χωρίς βέβαια να επηρεάζει τις τιμές των στοιχείων (τα δεδομένα) του y.

Καθώς η δημιουργία ενός νέου αντικειμένου συχνά ανάγεται σε αποθήκευση δεδομένων και ιδιοτήτων σε μια ενιαία δομή, ενώ οι ιδιότητες ορίζουν σημαντικές παραμέτρους που επηρεάζουν την συμπεριφορά του αντικειμένου (όπως οι διαστάσεις ή οι κλάσεις στις οποίες ανήκει), η συνάρτηση **structure** επιτρέπει την δημιουργία αντικειμένων με αυτή ακριβώς την προσέγγιση. Δύο παραδείγματα:

```
p1 <- structure(.Data=1:12,dim=c(3,2,2))
p2 <- structure(.Data=list( a = c(21, 22, 26),
                             b = c("a","b","d")),
                row.names = c("Πρώτο", "Δεύτερο", "Τρίτο"),
                             class = "data.frame")
```

Η πρώτη εντολή δημιουργεί ένα αντικείμενο τύπου array σε μεταβλητή p1 ενώ αντίστοιχα η επόμενη εντολή δημιουργεί ένα data.frame[229] σε μεταβλητή p2. Και στις δύο περιπτώσεις, η 1η παράμετρος .Data ορίζει τα δεδομένα, ενώ οι υπόλοιπες παράμετροι γίνονται ιδιότητες του νέου αντικειμένου.

4.2.1.5 Η λίστα επιλογών συνεδρίας

Κατά τη συνεδρία με την R υπάρχει η δυνατότητα αξιοποίησης κάποιων επιλογών (ρυθμίσεων) που επηρεάζουν την εκτέλεση του κώδικα. Οι ρυθμίσεις αυτές είναι ένα list από αντικείμενα, τα στοιχεία της οποίας συνήθως δημιουργούνται και παίρνουν προκαθορισμένες τιμές κατά την εκκίνηση της συνεδρίας. Ακολούθως, τόσο τα πακέτα που συνδέονται στη συνεδρία, όσο και ο κώδικας του χρήστη, μπορούν να προσθέσουν ή να αλλάξουν τις επιλογές αυτές. Έχουν ήδη αναφερθεί δύο τέτοια στοιχεία της λίστας επίλογων, τα max.print και digits[230]. Ο κώδικας μπορεί να χρησιμοποιήσει τη λίστα ρυθμίσεων ανακαλώντας την τιμή μιας ρύθμισης με τη συνάρτηση **getOption.** Αν η ρύθμιση δεν έχει οριστεί, η συνάρτηση θα επιστρέψει την τιμή NULL. Με την συνάρτηση **options** μπορεί να γίνει χειρισμός των στοιχείων της λίστας:

[229] βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων).

[230] Οι προκαθορισμένες ρυθμίσεις μετά την εγκατάσταση της R καθώς και οι προεπιλεγμένες τιμές για αυτές αναφέρονται στο help(options).

Δοκιμάστε:	Σχόλιο
`getOption("digits")`	Η τρέχουσα τιμή της ρύθμισης digits (μόνο για ανάκληση).
`options()$digits`	Ίδιο με το παραπάνω.
`options()`	Λίστα ρυθμίσεων (μόνο για ανάκληση).
`options("digits")`	Λίστα ενός στοιχείου (της ρύθμισης digits).
`options(digits=5)`	Ανάθεση στη ρύθμιση digits της τιμής 5.
`options(myoption = 5:10)`	Ανάθεση σε (νέα) ρύθμιση myoption του διανύσματος 5:10.
`options(myoption = NULL)`	Διαγραφή της ρύθμισης myoption.

Η εντολή options() ανακαλεί την πλήρη λίστα, αλλά για την αποφυγή λανθασμένων επεμβάσεων σε κρίσιμες ρυθμίσεις η συνάρτηση δεν επιτρέπει την προσθήκη νέων στοιχείων (ή αλλαγή στα υπάρχοντα) παρά μόνο μέσα από τις παραμέτρους της, εφαρμόζοντας σχετικούς ελέγχους.

4.2.2 Ο τύπος environment (περιβάλλον)

Το καθολικό περιβάλλον (Global Environment) είναι ο γνωστός μας χώρος όπου ο χρήστης (εσείς) δημιουργεί μεταβλητές. Στην R οι μεταβλητές είναι ονόματα για αντικείμενα που δημιουργήθηκαν δυναμικά και πρέπει να διατηρηθούν. Όπως αναφέρθηκε σε προηγούμενη ενότητα (βλ.§2.2.3 Ο ρόλος των περιβαλλόντων), ένα περιβάλλον είναι και αυτό ένα αντικείμενο. Αντικείμενα αυτού του τύπου, δηλαδή περιβάλλοντα, δημιουργούνται από συναρτήσεις (για τα τοπικά τους αντικείμενα), από πακέτα (για τα αντικείμενα που παρέχουν) κλπ. Όμως σε κάποιες υλοποιήσεις λύσεων ίσως χρειαστεί να δημιουργήσετε δικά σας περιβάλλοντα (για λόγους οργάνωσης των αντικειμένων της λύσης σας, προσδιορισμό της εμβέλειας των ονομάτων τους ή απλώς για να εκμεταλλευτείτε κάποιες ιδιαιτερότητες της δομής αυτής). Τα περιβάλλοντα δημιουργούνται με τη συνάρτηση **new.env** και έχουν σε κάποια σημεία κοινό συντακτικό με τον τύπο list (που περιγράφεται στη.§4.2.1 Ο τύπος list (λίστα)). Όμως, αν και παρεμφερή με τον τύπο list, τα αντικείμενα τύπου environment δεν είναι list και έχουν διαφορές στον χειρισμό των περιεχομένων τους. Μια βασική τέτοια διαφορά είναι πως στα environment δεν ορίζεται θέση για τα αντικείμενα που περιέχουν. Άρα, δεν μπορεί να υπάρξει πρόσβαση στα αντικείμενα αυτά με δείκτες αριθμού θέσης και τον τελεστή [], ενώ ο τελεστής [[]] υποστηρίζεται μόνο με παράμετρο το όνομα του στοιχείου. Άλλη βασική διαφορά από τα list είναι ότι ένα αντικείμενο στο environment δεν διαγράφεται αν του ανατεθεί η τιμή NULL[231].

Έτσι, ένα αντικείμενο τύπου environment είναι ουσιαστικά ένα σύνολο από ονόματα που αντιστοιχούν σε αντικείμενα. Το environment αυτό ονομάζεται και «πλαίσιο» (frame) των αντικειμένων. Επιπρόσθετα για την αναζήτηση αντικειμένων εκτός του περιβάλλοντος ορίζεται και ένα γονικό περιβάλλον (parent environment) ή περιέχον περιβάλλον (enclosing environment), συνήθως αυτόματα και δυναμικά. Με δεδομένο πως ένα περιβάλλον μπορεί με τη σειρά του να περιέχει ένα ή περισσότερα άλλα περιβάλλοντα, ο ορισμός γονικού περιβάλλοντος (parent environment) δημιουργεί μια δενδροειδή δομή αναζήτησης μέσα στην οποία οι συναρτήσεις get και exists (που θα δούμε παρακάτω) αναζητούν ονόματα, ξεκινώντας από κάποιο υψηλότερο επίπεδο και κινούμενες προς τη «ρίζα» (δηλαδή το περιβάλλον του πακέτου 'base').

Το επόμενο παράδειγμα δημιούργει ένα environment και προσθέτει μερικά νέα αντικείμενα μέσα σε αυτό, με διαφορετικές προσεγγίσεις σύνταξης των απαραίτητων εντολών:

Δοκιμάστε:	Σχόλιο
`p <- new.env()`	Δημιουργεί p με νέο αντικείμενο τύπου environment.
`assign("x1",10,p)`	Δημιουργία x1 στο p με χρήση της assign.
`p$x2<-20`	Δημιουργία x2 στο p με χρήση του $.
`p$x3<-"3η μεταβλητή"`	Προφανώς, ένα περιβάλλον μπορεί να περιέχει αντικείμενα διαφόρων τύπων.
`p[["x4"]]<-40`	Δημιουργία x4 στο p με χρήση των [[και]] (όπως στις λίστες).
`names(p)`	Διάνυσμα με τα ονόματα των αντικειμένων στο p, εδώ c("x4","x1","x2","x3").

Έχοντας δημιουργήσει το περιβάλλον p μπορείτε να δείτε τα περιεχόμενα του από καρτέλα Environment του RStudio (με κλικ στο αντικείμενο, κάτι που εκτελεί την εντολή View(p)). Στο επόμενο παράδειγμα γίνεται

[231] Αδυναμία πρόσβασης με τον αριθμό θέσης και δυνατότητα αποθήκευσης τιμών NULL είδαμε ήδη στα attributes των αντικειμένων (βλ.§4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων).

χρήση των αντικειμένων στο p που ορίστηκε παραπάνω. Παρατηρήστε πως εδώ η σύνταξη είναι ίδια με αυτή των αντικειμένων τύπου list:

Δοκιμάστε:	**Σχόλιο**
`p$x5<-p$x2*10`	Πράξεις με το x2 του p και καταχώρηση αποτελέσματος ως x5 του p.
`p$x5`	Επιστρέφει το x5 του p (δηλαδή 200).

Άλλες συναρτήσεις[232] που είναι χρήσιμες κατά τον χειρισμό και τη χρήση περιβαλλόντων είναι οι parent.env, οι προαναφερθείσες ls, ls.str και rm, καθώς και οι eval, evalq και with που εκτελούν εντολές (ή αποτιμούν εκφράσεις) σε συγκεκριμένα περιβάλλοντα[233]:

Δοκιμάστε:	**Σχόλιο**
`parent.env(p)`	Το γονικό περιβάλλον του p (εδώ επιστρέφει το Global Environment).
`ls(p)`	Τα ονόματα αντικειμένων στο p.
`ls.str(p)`	Τα ονόματα και η δομή αντικειμένων στο p.
`rm("x3",envir = p)`	Διαγραφή του x3 από το p.
`evalq(x6<-6,p)`	Εκτελεί την έκφραση "x6<-6" στο p, άρα αποθηκεύει το 6 σε νέα x6 (του p).
`evalq(x7<-x6+a,p)`	Εκτελεί την έκφραση στο p, άρα αποθηκεύει το 36 σε (νέα) x7 (στο p).
`evalq(sqrt(x7),p)`	Εκτελεί την έκφραση στο p, επιστέφει την τετραγ. ρίζα του 36, δηλαδή 6.
`with(p,x8<-x1+x2)`	Στο p εκτελεί το x8<-x1+x2, άρα αποθηκεύει 30 σε x8 του p.
`a<-with(p,x1+x2)`	Στο p εκτελεί το x1+x2, άρα αποθηκεύει 30 σε a του Global Environment.

Επιπρόσθετα, οι συναρτήσεις get και exists αναζητούν αντικείμενα (μέσω του ονόματος τους) σε συγκεκριμένο περιβάλλον. Αν δεν τα εντοπίσουν εκεί, ως προεπιλογή[234] προχωρούν στην αναζήτηση στα γονικά περιβάλλοντα (με τον συνήθη τρόπο που περιγράφεται στην §2.2.3 Ο ρόλος των περιβαλλόντων):

Δοκιμάστε:	**Σχόλιο**
`exists("x5",p)`	Υπάρχει x4 προσβάσιμο από το p; Ναι, υπάρχει x4 στο p (TRUE).
`exists("a",p)`	Υπάρχει a προσβάσιμο από το p; Δεν υπάρχει (FALSE).
`a<-1`	Ορισμός μίας μεταβλητής a με την τιμή 1 στο Global Environment.
`exists("a",p)`	Υπάρχει a προσβάσιμο από το p; Ναι υπάρχει στο Global (TRUE).
`get("x5",p)`	Αναζητά το x5 στο περιβάλλον p και επιστρέφει την τιμή του.
`get("a",p)`	Αναζητά το a στο p. Δεν υπάρχει, άρα το αναζητά και εντοπίζει a στο Global.

Έτσι, αν σε ένα περιβάλλον p έχει οριστεί από τον χρήστη συγκεκριμένο γονικό περιβάλλον, οι συναρτήσεις get και exists θα αναζητήσουν πρώτα τη μεταβλητή στο p και εφόσον δεν βρεθεί εκεί, θα συνεχίσουν την αναζήτηση στο αμέσως κατώτερο επίπεδο, δηλαδή στο γονικό περιβάλλον του p, συνεχίζοντας μετά στα υπόλοιπα κατώτερα επίπεδα περιβαλλόντων αν χρειάζεται. Στο επόμενο παράδειγμα δημιουργούνται διάφορα περιβάλλοντα και συνδυασμοί αυτών. Σε κάποια έχει οριστεί γονικό περιβάλλον (στα p3, p4, p5 και p2$p7), ενώ στα υπόλοιπα το γονικό περιβάλλον ορίζεται αυτόματα και είναι το Global Environment. Αυτό ισχύει ακόμα και για τα περιβάλλοντα που τοποθετούνται μέσα σε άλλα, όπως το p6 που τοποθετείται παρακάτω μέσα στο p2 (δηλαδή το p2$p6) αλλά δεν έχει οριστεί γονικό περιβάλλον (άρα ορίζεται αυτόματα το Global Environment, το επίπεδο δηλαδή στο οποίο εργάζεται ο κώδικας τη στιγμή που δημιουργείται το αντικείμενο). Τέλος, στο επόμενο παράδειγμα στο p7 που τοποθετήθηκε στο περιβάλλον p2 (στο p2$p7) έχει οριστεί από τον χρήστη ως γονικό περιβάλλον το p1, κάτι που επιτρέπεται.

---

232 Παρέχονται από το προ-εγκατεστημένο πακέτο 'base', πλην της ls.str που παρέχεται από το 'utils'.

233 Οι συναρτήσεις αυτές καθώς και local (βλ.§3.2.1.2 Χρήση μπλοκ κώδικα για ορισμό εμβέλειας μεταβλητών) είναι χρήσιμες για εκτέλεση εντολών σε συγκεκριμένα (ή προσωρινά) περιβάλλοντα, βλ. help(eval). Η evalq κάνει αναθέσεις σε μεταβλητές στο ορισθέν περιβάλλον. Είναι παρεμφερής με την eval, η οποία όμως κάνει αναθέσεις σε μεταβλητές στο περιβάλλον όπου έγινε η κλήση της. Για τις «εκφράσεις» και τη συνάρτηση eval βλ.§4.3.4 Αντικείμενα τύπου language (expression, call, name και formula). Η συνάρτηση with μπορεί να δημιουργήσει προσωρινά περιβάλλοντα και από άλλους συμβατούς τύπους αντικειμένων, π.χ. αντικείμενα data.frame (βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων)).

234 Ρυθμίζεται από την παράμετρο inherits των συναρτήσεων αυτών.

Δοκιμάστε:	**Σχόλιο**
`rm(list=ls())`	(Προαιρετικό, διαγραφή των αντικειμένων στο Global Environment).
`p0 <- new.env()`	Δημιουργία περιβάλλοντος με όνομα p0 (στο Global Environment).
`p1 <- new.env()`	Δημιουργία περιβάλλοντος p1.
`p2 <- new.env()`	Δημιουργία περιβάλλοντος p2.
`p3 <- new.env(parent=p2)`	Δημιουργία περιβάλλοντος p3 με γονέα το p2.
`p4 <- new.env(parent=p2)`	Δημιουργία περιβάλλοντος p4 με γονέα το p2.
`p5 <- new.env(parent=p4)`	Δημιουργία περιβάλλοντος p5 με γονέα το p4.
`p2$p6 <- new.env()`	Δημιουργία περιβάλλοντος p6 στο p2 (χωρίς ορισμό γονέα).
`p2$p7 <- new.env(parent=p1)`	Δημιουργία περιβάλλοντος p7 στο p2 με γονέα το p1.
`x<-"μεταβλητή x στο Global"`	Δημιουργία ενός αντικειμένου με όνομα x (στο Global Environment).
`p1$x<-"μεταβλητή x στο p1"`	Δημιουργία άλλου αντικειμένου με όνομα x στο p1.
`p2$x<-"μεταβλητή x στο p2"`	Δημιουργία τρίτου αντικειμένου με όνομα x στο p2.
`get("x",p0)`	Το p0 δεν έχει x, επιστρέφει το x από το γονικό περιβάλλον (το Global).
`get("x",p1)`	Αντίστοιχα, επιστρέφει το x του p1 (αφού το p1 έχει x).
`get("x",p2)`	Το x του p2 (το p2 έχει x).
`get("x",p3)`	Το x του p2 (το p3 δεν έχει x, αλλά το γονικό περιβάλλον p2 έχει).
`get("x",p4)`	Το x του p2 (το p4 δεν έχει x, αλλά το γονικό περιβάλλον p2 έχει).
`get("x",p5)`	Το x του p2 (το p5 και ο γονέας p4 δεν έχουν x, ο γονέας του p4 έχει).
`get("x",p2$p6)`	Το x του Global Environment (το p2$p6 δεν έχει x).
`get("x",p2$p7)`	Το x του p1 (το p2$p7 δεν έχει x, αλλά ο γονέας του p1 έχει).

Οι συναρτήσεις get και exists κάνουν αναζήτηση στα γονικά περιβάλλοντα. Γενικά όμως αυτό δεν συμβαίνει αυτόματα. Π.χ. αν ζητήσετε απευθείας την τιμή του p5$x αυτό θα επιστρέψει NULL (την ειδική τιμή που σημαίνει κενό αντικείμενο δηλαδή ‘τίποτα’). Το ίδιο το p5 δεν περιέχει κάποιο x και δεν γίνεται αναζήτηση στα γονικά περιβάλλοντα. Ένα περιβάλλον πάντως μπορεί να προσαρτηθεί στη σειρά αναζήτησης μεταβλητών με τη συνάρτηση **attach** (θα τοποθετηθεί αμέσως κάτω από το Global Environment άρα θα οριστεί ως γονικό/περιέχον περιβάλλον αυτού) καθώς και να αποσυνδεθεί με την **detach**. Π.χ.:

Δοκιμάστε:	**Σχόλιο**
`p8 <- new.env()`	Δημιουργία περιβάλλοντος με όνομα p8 (στο Global Environment).
`p8$x1<-10`	Δημιουργία αντικειμένου με όνομα x1 στο p8.
`attach(p8)`	Προσθέτει το p8 στη σειρά αναζήτησης (αμέσως μετά το Global).
`x1`	Τα αντικείμενα του p8 μπορούν πλέον να εντοπιστούν (επιστρέφει 10).
`detach(p8)`	Αφαιρεί το p8 από τη σειρά αναζήτησης.
`x1`	Λάθος, τα αντικείμενα του p8 δεν μπορούν πλέον να εντοπιστούν.

4.2.2.1 Περιβάλλοντα και συναρτήσεις

Χρήσιμο είναι να διευκρινιστεί πως αν ένα περιβάλλον περιέχει μια συνάρτηση (ή ακριβέστερα, περιέχει ένα όνομα για κάποιο αντικείμενο τύπου function), η αναζήτηση από τη συνάρτηση μη-τοπικών, εξωτερικών ονομάτων αντικειμένων δεν ξεκινά από το περιβάλλον στο οποίο έχει τοποθετηθεί το όνομα τη συνάρτησης. Το περιβάλλον αυτό αναφέρεται και ως περιβάλλον δέσμευσης (binding environment) και είναι απλά η θέση ενός ονόματος για τη συνάρτηση. Η συνάρτηση θα αναζητά πάντα εξωτερικά (μη-τοπικά) αντικείμενα ξεκινώντας από το περιβάλλον μέσα στο οποίο λειτουργούσε ο κώδικας που τη δημιούργησε. Έτσι στο παρακάτω παράδειγμα η συνάρτηση s, αν και τοποθετήθηκε μέσα στο περιβάλλον p, δεν αναζητά το x στο το περιβάλλον p. Αντίθετα αναζητά και εντοπίζει το x ξεκινώντας από το Global Environment:

Δοκιμάστε:	**Σχόλιο**
`x<-"Μεταβλητή στο Global"`	Δημιουργία ενός αντικειμένου με όνομα x στο Global Environment.
`p<-new.env()`	Δημιουργία περιβάλλοντος p.
`p$x<-"Μεταβλητή στο p"`	Δημιουργία αντικειμένου με όνομα x στο p.
`p$s<-function() {x}`	Όνομα s στο p για συνάρτηση που απλώς επιστρέφει το (εξωτερικό) x.
`p$s()`	Κλήση της συνάρτησης, επιστρέφει το x από το Global Environment.

Συγκεκριμένα, οι τοπικές μεταβλητές των συναρτήσεων, δηλαδή τα αντικείμενα που ορίζονται μέσα στις συναρτήσεις και έχουν μόνο τοπική εμβέλεια (βλ.§5.1.5 Αλληλεπίδραση με το περιβάλλον) τοποθετούνται σε ένα προσωρινό εσωτερικό αντικείμενο τύπου environment και εκεί ξεκινά η αναζήτηση αντικειμένων από τον κώδικα της συνάρτησης. Ως γονικό (περιέχον) περιβάλλον αυτού του τοπικού περιβάλλοντος ορίζεται το περιβάλλον μέσα στο οποίο δημιουργήθηκε η συνάρτηση. Για παράδειγμα:

```
x<-"Μεταβλητή στο Global"
s1<-function() {x}
s2<-function()
  {
  x<-"Μεταβλητή τοπική στο s2"
  s1()
  }
s2()
```

Ο κώδικας που δημιούργησε τις s1 και s2 λειτουργεί μέσα στο Global Environment (αφού εκεί «τρέχει» εξ ορισμού ο κώδικας του χρήστη) και έτσι αυτό το περιβάλλον ορίζεται ως parent/enclosing environment του τοπικού τους περιβάλλοντος. Όταν κληθεί η συνάρτηση s1 αναζητά το x (εφόσον δεν υπάρχει στο τοπικό της environment) στο γονικό του περιβάλλον (δηλαδή το Global Environment). Ακόμα και όταν κληθεί η s2 και αυτή καλέσει την s1, η τελευταία θα αναζητήσει το x στο Global και όχι στο τοπικό περιβάλλον της s2 που την κάλεσε.

Αντίστοιχα, όταν μια συνάρτηση δημιουργείται μέσα σε άλλη συνάρτηση[235], ο κώδικας που δημιουργεί την εσωτερική συνάρτηση λειτουργεί μέσα στο τοπικό περιβάλλον της εξωτερικής συνάρτησης, άρα αυτό ορίζεται και ως parent/ enclosing environment του τοπικού περιβάλλοντος της εσωτερικής. Ως αποτέλεσμα, η εσωτερική συνάρτηση αναζητά μη-τοπικά αντικείμενα πρώτα στο περιβάλλον της εξωτερικής και μετά συνεχίζει στα γονικά περιβάλλοντα αυτού. Στο παρακάτω παράδειγμα, η συνάρτηση s1 είναι ορισμένη μέσα στην s2 (και άρα το τοπικό περιβάλλον της s2 έχει γονικό περιβάλλον το τοπικό περιβάλλον της s1). Όταν η s2 καλέσει την s1 αυτή αναζητά αντικείμενα πρώτα στο τοπικό περιβάλλον της, μετά στο τοπικό περιβάλλον της s2, μετά στο γονικό περιβάλλον της s2 (δηλαδή το Global Environment) και ούτω καθεξής. Άρα η s2 (και η s1) θα επιστρέψει το x που είναι τοπικό στο s2:

```
x<-"Μεταβλητή στο Global"
s2<-function()
  {
  s1<-function() {x}
  x<-"Μεταβλητή τοπική στο s2"
  s1()
  }
s2()
```

Όμως οι συναρτήσεις (και άλλα περιβάλλοντα εκτέλεσης κώδικα όπως αυτά που δημιουργεί η local, βλ.§3.2.1.2 Χρήση μπλοκ κώδικα για ορισμό εμβέλειας μεταβλητών) έχουν δυνατότητα πρόσβασης και στο περιβάλλον που τις καλεί. Το περιβάλλον αυτό ονομάζεται γονικό πλαίσιο (parent frame). Η χρήση εδώ του ίδιου επιθέτου «γονικό» (parent) όπως και στο γονικό περιβάλλον, οφείλεται σε ιστορικούς λόγους, είναι ατυχής και συχνά προκαλεί σύγχυση (όπως παραδέχεται και η σχετική τεκμηρίωση της R, βλ.help(environment)). Το parent frame (που επιστρέφει η συνάρτηση **parent.frame**) είναι το περιβάλλον από το οποίο καλείται κάποιο άλλο. Για αυτό τον λόγο το parent frame αναφέρεται και ως calling frame η calling environment. Η parent.frame μπορεί να εντοπίσει (αν υπάρχει) και το περιβάλλον που κάλεσε το parent, ορίζοντας (στην παράμετρό της n) τον αριθμό 2 (ή 3 αν αναζητείται το parent.frame αυτού, ή και μεγαλύτερο αριθμό για προηγούμενη «γενιά» κλήσεων που οδήγησαν στην τρέχουσα). Στο παρακάτω παράδειγμα η s1 επιστρέφει το x από όποιο περιβάλλον την κάλεσε, άρα αν κληθεί η s2 (που με τη σειρά της καλεί την s1) θα επιστραφεί το x που είναι τοπικό στο s2:

```
x<-"Μεταβλητή στο Global"
s1<-function() { parent.frame()$x }
s2<-function()
  {
  x<-"Μεταβλητή τοπική στην s2"
```

[235] Η εσωτερική ονομάζεται και εμφωλευμένη συνάρτηση (nested function) και είναι τοπική στη συνάρτηση που την περιέχει.

```
    s1()
    }
  s2()
```

Στο επόμενο παράδειγμα η s1 μέσω της συνάρτησης assign αλλάζει το x από όποιο περιβάλλον την κάλεσε (εδώ το s2). Άρα αν κληθεί η s2 (που με την σειρά της καλεί την s1) θα αλλάξει το x που είναι τοπικό στο s2:

```
x <- "Μεταβλητή στο Global"
s1 <-
  function() {
    assign("x", "Αλλαγή του x στο s2", envir = parent.frame())
  }
s2 <- function()
{
  x <- "Μεταβλητή τοπική στην s2"
  s1()
  x
}
s2()
```

Τέλος, μια ακόμα πιο τεχνική παραλλαγή του παραδείγματος είναι η παρακάτω, όπου το s1 διαγράφει το όποιο όνομα x υπάρχει στο περιβάλλον που την καλεί (εδώ στο s2). Όταν το s2 αμέσως μετά αναζητά κάποιο x το βρίσκει στο Global Environment, καθώς η τοπική στο s2 μεταβλητή x έχει διαγραφεί:

```
x<-"Μεταβλητή στο Global"
s1<-function() { rm("x", envir = parent.frame()) }
s2<-function()
{
  x<-"Μεταβλητή τοπική στην s2"
  s1()
  x
}
s2()
```

Με τα παραπάνω μπορεί να εξηγηθεί καλύτερα και η ύπαρξη διαφορετικών τελεστών ανάθεσης τιμών σε μεταβλητές (<-, =, <<- κλπ.). Οι συνήθεις τελεστές (<- και ->) δημιουργούν αντικείμενα στο περιβάλλον που εφαρμόζονται. Ο τελεστής = επίσης δημιουργεί αντικείμενα στο περιβάλλον που εφαρμόζεται, αλλά χρησιμοποιείται κυρίως σε κορυφαίο επίπεδο. Όμως οι τελεστές <<- και ->> αναζητούν τη μεταβλητή στην οποία θα δοθεί τιμή τόσο στο τρέχον όσο και στα γονικά περιβάλλοντα. Όταν βρεθεί, αναθέτουν την τιμή σε αυτή. Αν δεν βρεθεί, δημιουργούν μία στο Global Environment. Π.χ. στο παρακάτω παράδειγμα το <<- στην s1 αναζητά κάποιο x για να αλλάξει, βρίσκει το x που είναι τοπικό στο s2 (γονικό περιβάλλον του s1) και κάνει εκεί την αλλαγή. Αν δεν υπήρχε ήδη το x της s2, θα άλλαζε το x στο Global Environment, ενώ αν δεν υπήρχε ήδη x ούτε εκεί, θα το δημιουργούσε. Έτσι ο κώδικας αυτός θα επιστρέψει το x που είναι τοπικό στην s2 αλλαγμένο όμως από την s1. Το x στο Global Environment παραμένει ως έχει:

```
x<-"Μεταβλητή στο Global"
s2<-function()
{
  s1<-function() { x<<-"Αλλαγμένο x από s1"}
  x<-"Μεταβλητή τοπική στην s2"
  s1()
  x
}
s2()
```

4.2.2.2 Περιβάλλοντα και ο μηχανισμός αναφοράς

Μια ιδιαιτερότητα της δομής environment (περιβάλλον) είναι ότι προσφέρουν μηχανισμούς αναφοράς (reference semantics). Αυτό επιτρέπει μια σύνδεση πολλών μεταβλητών με το ίδιο αντικείμενο environment στη μνήμη, κάτι που δεν επιτρέπει μόνο την ανάκληση αλλά και την τροποποίηση των περιεχομένων τους. Κατ'

επέκταση αυτό οδηγεί σε δυνατότητα κλήσης συναρτήσεων (με πέρασμα παραμέτρων τύπου environment) ως κλήση με αναφορά (call-by-reference)[236]. Υπενθυμίζουμε πως στην R, όταν δυο μεταβλητές αναφέρονται στο ίδιο (οποιουδήποτε τύπου) αντικείμενο, στη μνήμη η γλώσσα διατηρεί μόνο ένα αντίγραφο του αντικειμένου. Αν όμως γίνει οποιαδήποτε τροποποίηση σε κάποια από τις δυο μεταβλητές, η R δημιουργεί ένα δεύτερο αντικείμενο, αντίγραφο του αρχικού αντικειμένου. Σε αυτό κάνει την αλλαγή και μετά αναθέτει στο αποτέλεσμα το όνομα της μεταβλητής που χρησιμοποιήθηκε για την αλλαγή, ενώ η άλλη μεταβλητή αναφέρεται στο αρχικό[237]. Η προσέγγιση αυτή (copy-on-modify) εφαρμόζεται κατά την ανάθεση τιμών από την R σε όλους τους συνήθεις τύπους αντικείμενων, αλλά τα αντικείμενα τύπου environment αποτελούν εξαίρεση[238]. Το επόμενο παράδειγμα δείχνει τη διαδικασία χρησιμοποιώντας δύο αντικείμενα τύπου list:

Δοκιμάστε:	**Σχόλιο**
`l1 <- list()`	Δημιουργία αντικειμένου με όνομα l1 και τύπου list.
`l1$x<-10`	Ανάθεση της τιμής 10 σε x στη λίστα l1.
`l2<-l1`	Δημιουργία l2 που αναφέρεται στο ίδιο αντικείμενο με το l1.
`l2$x<-20`	Ανάθεση της τιμής 20 σε x του l2. Τα l1 και l2 είναι πλέον διαφορετικά αντικείμενα.
`print(l1$x)`	Εμφάνιση της τιμής x στη λίστα l1, δηλαδή 10.
`print(l2$x)`	Εμφάνιση της τιμής x στη λίστα l2, δηλαδή 20.

Όμως τα αντικείμενα τύπου environment αποτελούν εξαίρεση και η R τα χειρίζεται με ειδικό τρόπο. Όταν δύο μεταβλητές αναφέρονται στο ίδιο αντικείμενο τύπου environment, αλλαγές που μπορεί να γίνουν μέσω οποιασδήποτε από τις δύο μεταβλητές εφαρμόζονται στο ίδιο, κοινό αντικείμενο (άρα ισχύουν και για τις δύο):

Δοκιμάστε:	**Σχόλιο**
`p1<-new.env()`	Δημιουργία αντικειμένου με όνομα p1 και τύπου environment.
`p1$x<-10`	Ανάθεση της τιμής 10 σε x στο περιβάλλον p2.
`p2<-p1`	Δημιουργία p2 που αναφέρεται στο ίδιο αντικείμενο με το p1.
`p2$x<-20`	Ανάθεση της τιμής 20 σε x του p2, αλλάζοντας το κοινό αντικείμενο στη μνήμη.
`print(p1$x)`	Εμφάνιση της τιμής x στο περιβάλλον p1, δηλαδή 20.
`print(p2$x)`	Εμφάνιση της τιμής x στο περιβάλλον p2, δηλαδή 20 (είναι το ίδιο με το παραπάνω).

Η ιδιαιτερότητα αυτή των αντικειμένων τύπου environment μπορεί να αξιοποιηθεί ως εναλλακτικός μηχανισμός επιστροφής αποτελεσμάτων από τις συναρτήσεις. Ας δούμε τη διαφορά στο επόμενο παράδειγμα:

Δοκιμάστε:	**Σχόλιο**
`f<-function(q){q$x<-20}`	Συνάρτηση f που θέτει στο x οποιασδήποτε παραμέτρου q την τιμή 20.
`l<-list()`	Δημιουργία l τύπου list.
`l$x<-10`	Ανάθεση σε x της λίστας l της τιμής 10.
`f(l)`	Κλήση της f με παράμετρο τη λίστα l.
`print(l1$x)`	Εμφάνιση της τιμής x στη λίστα l, παραμένει 10 (οι αλλαγές έμειναν τοπικές).
`p<-new.env()`	Δημιουργία p τύπου environment.
`p$x<-10`	Ανάθεση σε x στο περιβάλλον p της τιμής 10.
`f(p)`	Κλήση της f με παράμετρο το περιβάλλον p.
`print(p$x)`	Εμφάνιση της τιμής x στο περιβάλλον p, δηλαδή 20 (αλλάχτηκε από την f).

Η δυνατότητα πολλαπλής αναφοράς σε ένα environment (άρα και στα αντικείμενα που περιέχει) είναι μια χρήσιμη ιδιαιτερότητα του τύπου αυτού, υπάρχουν όμως και άλλες. Μια εξ αυτών είναι ότι τα αντικείμενα environment εφαρμόζουν (ως προεπιλογή) τεχνικές hashing για να υποβοηθήσουν την αναζήτηση αντικειμένων που περιέχονται σε αυτά, με αποτέλεσμα ο εντοπισμός του αντικειμένου να γίνεται σε σταθερό χρόνο (O(1)). Για περισσότερες αξιοποιήσιμες ιδιαιτερότητες του τύπου environment (βλ. [33]).

---

[236] βλ.§5.1.5 Αλληλεπίδραση με το περιβάλλον, καθώς και σχετική υποσημείωση 295.
[237] βλ.§2.2.1 Δημιουργία, χρήση, μετατροπή μεταβλητών καθώς και §3.1.4 Εργαλεία προφίλ (profiling) για την εντολή tracemem.
[238] Άλλες εξαιρέσεις περιλαμβάνουν τα data.table (βλ.§4.3.1 Οι τύποι tibble και data.table) καθώς και άλλα αντικείμενα που βασίζονται σε κλάσεις αναφοράς (βλ.§6.5 Κλάσεις αναφοράς).

4.2.2.3 Περιβάλλοντα για δημιουργία δομών δεδομένων

Η δυνατότητα χρήσης αναφοράς στα αντικείμενα τύπου environment (βλ. προηγ. ενότητα) επιτρέπει τη δημιουργία δυναμικών δομών δεδομένων όπως συνδεδεμένες λίστες (αλυσιδωτές λίστες/linked list), δένδρα (tree), δυναμικά διανύσματα (dynamic vector), σωροί (stack), ουρές (queue) κλπ. Οποιαδήποτε παρόμοιου τύπου δομή αποτελούμενη από κόμβους (στοιχεία /node) που περιέχουν αναφορές σε άλλους (ή και τον ίδιο) κόμβο, μπορεί να κατασκευαστεί με κόμβους που θα είναι αντικείμενα τύπου environment.

Στο παράδειγμα που ακολουθεί δημιουργείται μια μονά-συνδεδεμένη λίστα (singly-linked list), όπου κάθε νέος κόμβος που προστίθεται στη λίστα είναι ένα αντικείμενο τύπου environment, άρα μπορεί να δημιουργείται με κώδικα όπως ο παρακάτω:

```
new_node <- new.env()
new_node$data <- 0
new_node$next_node <- NULL
```

Κάθε τέτοιος κόμβος περιέχει δεδομένα (στη μεταβλητή data του κόμβου, εδώ αρχικά 0) και μια αναφορά στον επόμενο κόμβο (στη μεταβλητή next_node, εδώ χρησιμοποιείται η ειδική τιμή NULL ως ένδειξη ότι δεν υπάρχει επόμενος κόμβος). Για να χρησιμοποιηθεί η λίστα μετά τη δημιουργία της, το μόνο που απαιτείται να παραμένει γνωστό είναι ο πρώτος της κόμβος (head). Η πρόσβαση στα επόμενα στοιχεία μπορεί να γίνει ακολουθώντας τις αναφορές που καταγράφονται ως next_node κάθε κόμβου. Το πλήρες παράδειγμα[239] δίνεται παρακάτω:

```
# αρχικά το head είναι NULL καθώς η λίστα είναι κενή

head <- NULL

# προσθήκη 10 κόμβων στη λίστα:

for (i in 1:10)
{
  # δημιουργία ενός κόμβου (με ένα κείμενο ως data):

  new_node <- new.env()
  new_node$data <- paste("ΔΕΔΟΜΕΝΑ#",i)
  new_node$next_node <- NULL

  # καταγραφή του νέου κόμβου ως πρώτου (head) ή σύνδεση
  # με τον τρέχοντα τελευταίο κόμβο (last_node) στη λίστα:

  if (is.null(head))  head <- new_node
  else last_node$next_node <- new_node

  # ο νέος κόμβος καταγράφεται πλέον ως τελευταίος.

  last_node <- new_node
}

# Εκτός της μεταβλητής head, οι υπόλοιπες μεταβλητές
# που χρησιμοποιήθηκαν για τη δημιουργία της λίστας
# δεν είναι πλέον απαραίτητες και μπορούν να διαγραφούν:

rm(i, new_node, last_node)
```

Η λίστα δημιουργήθηκε (βλ. Εικόνα 4.2) και πλέον μπορεί να γίνει χρήση της.

[239] Το παράδειγμα είναι απλουστευμένο. Χρησιμότερο θα ήταν να δημιουργηθούν συναρτήσεις χειρισμού της λίστας (εισαγωγής κόμβων, διαγραφής κόμβων κλπ.) με τον τρόπο που περιγράφεται στην §5.1.1 Δημιουργία συναρτήσεων.

Εικόνα 4.2: *Η διασυνδεδεμένη λίστα του παραδείγματος.*

Η επόμενη εντολή αλλάζει την τιμή data του 2ου κόμβου στον αριθμό 2000:

```
head$next_node$data <- 2000
```

Η παρακάτω εντολή διαγράφει τον 3ο κόμβο της λίστας[240]:

```
head$next_node$next_node <- head$next_node$next_node$next_node
```

Τέλος, ο επόμενος κώδικας διατρέχει τη λίστα ξεκινώντας από το head και εκτυπώνει τα δεδομένα στο data του τρέχοντος κόμβου x. Ο κώδικας θα έχει διατρέξει όλη τη λίστα όταν ξεκινώντας από το head και ακολουθώντας τα next_node φτάσει σε ένα με τιμή NULL:

```
x <- head
while (!is.null(x))
{
  print(x$data)
  x <- x$next_node
}
```

Το αποτέλεσμα της εκτέλεσης του κώδικα είναι:

```
[1] "ΔΕΔΟΜΕΝΑ# 1"
[1] 2000
[1] "ΔΕΔΟΜΕΝΑ# 4"
[1] "ΔΕΔΟΜΕΝΑ# 5"
[1] "ΔΕΔΟΜΕΝΑ# 6"
[1] "ΔΕΔΟΜΕΝΑ# 7"
[1] "ΔΕΔΟΜΕΝΑ# 8"
[1] "ΔΕΔΟΜΕΝΑ# 9"
[1] "ΔΕΔΟΜΕΝΑ# 10"
```

Με παρόμοιο τρόπο μπορούν να κατασκευαστούν και άλλες δομές δεδομένων όπως δένδρα, ουρές κλπ. Ένα παράδειγμα δημιουργικής χρήσης του τύπου environment είναι το σύστημα R6 (βλ.§6.5.2 Κλάσεις R6). Στο σύστημα αυτό, τα environment χρησιμοποιούνται ως βάση για τον ορισμό δομών οι οποίες εξομοιώνουν μέλη κλάσεων, παρόμοια με όσα παρέχουν άλλες αντικειμενοστραφείς γλώσσες προγραμματισμού.

4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων)

Το data.frame υλοποιεί μια δομή σχήματος πίνακα δύο διαστάσεων, παρεμφερή δηλαδή με τα matrix. Σε αντίθεση με τα matrix όμως, κάθε στήλη μπορεί να περιέχει στοιχεία διαφορετικού τύπου από τις υπόλοιπες. Ως πίνακας, χωρίζεται σε γραμμές και στήλες και όλες οι στήλες έχουν τον ίδιο αριθμό στοιχείων. Καθεμία στήλη ενός data.frame πρέπει να περιέχει ενός τύπου στοιχεία, αλλά ο τύπος στοιχείων μιας στήλης δεν χρειάζεται να είναι ίδιος με αυτόν άλλων στηλών. Εναλλακτικός τρόπος να κατανοηθεί η δομή των data.frame, είναι ως επέκταση του τύπου list (βλ.§4.2.1 Ο τύπος list (λίστα)), με τα στοιχεία να είναι αντικείμενα vector ή factor. Όλα τα αντικείμενα που περιέχει το list έχουν με τη σειρά τους τον ίδιο αριθμό δεδομένων (στοιχείων) και καθένα αντιστοιχεί σε μια στήλη του πίνακα. Έτσι κάθε στήλη είναι ένα atomic αντικείμενο με δεδομένα ίδιου τύπου, ο οποίος συνήθως είναι κάποιος βασικός τύπος (αριθμητικός, logical, κείμενο κλπ.) ή factor.

Ο τύπος data.frame επιτρέπει την αξιοποίηση και τον συνδυασμό των δύο παραπάνω προσεγγίσεων. Σε ένα αντικείμενο data.frame μπορεί να εφαρμοστεί μείγμα εντολών με σύνταξη παρεμφερή αυτής των matrix (αν πρέπει να χρησιμοποιηθούν γραμμές και στήλες) ή εντολές παρεμφερείς με αυτές των list (αν η επιμέρους επεξεργασία μεταβλητών είναι περισσότερο κατάλληλη προσέγγιση). Στη δεύτερη περίπτωση, θα γίνεται επιλογή στήλης (μεταβλητής) με το όνομα της (βλ.§4.2.1.1 Ο τελεστής επιλογής $). Πέραν αυτών, ο τύπος

[240] Στην R ο μηχανισμός οριστικής διαγραφής από τη μνήμη (garbage collection) ενεργοποιείται αυτόματα διαγράφοντας το αντικείμενο όταν δεν υπάρχει κάποια αναφορά (π.χ. μεταβλητή) προς αυτό.

data.frame προσθέτει δικές του ιδιότητες και λειτουργίες, κατάλληλες για τον χειρισμό δεδομένων σε μορφή πίνακα.

Τα data.frame είναι ίσως ο βασικότερος τύπος αντικειμένων που ενσωματώνει η R. Η μορφή πίνακα (tabular) είναι ιδιαίτερα συνηθισμένη σε σύνολα δεδομένων (data set) και τα data.frame είναι ιδιαίτερα κατάλληλα για την αποθήκευσή τους. Κάθε γραμμή σε ένα data.frame αφορά μια περίπτωση (instance/case) ή παρατήρηση (observation) στα δεδομένα. Κάθε στήλη αντιπροσωπεύει ένα χαρακτηριστικό (feature), δηλαδή την τιμή μιας μεταβλητής, που καταγράφεται σε καθεμία από τις περιπτώσεις ή τις παρατηρήσεις αυτές. Έτσι αυτό που προκύπτει είναι μια συλλογή από μεταβλητές που είναι συνδεδεμένες στενά μεταξύ τους. Προφανώς, τα δεδομένα της κάθε μεταβλητής (άρα και στήλης) είναι όντως ίδιου τύπου για όλες τις περιπτώσεις ή παρατηρήσεις.[241]. Όταν σε ένα data.frame (ή γενικότερα σε ένα αντικείμενο σχήματος πίνακα) όλα τα δεδομένα που αφορούν κάθε συγκεκριμένη εξεταζόμενη περίπτωση (case) καταγράφονται σε μια μοναδική γραμμή, τότε ο πίνακας είναι σε ευρεία (wide) διάταξη. Αν όμως για κάθε εξεταζόμενη οντότητα επιτρέπεται να καταγράφονται στοιχεία σε πολλές γραμμές (π.χ. παρατηρήσεις ενός χαρακτηριστικού καταγεγραμμένες σε διαφορετικό χρόνο), τότε ο πίνακας είναι σε μακρά (long) διάταξη. Σε αυτή την περίπτωση, κάποια στήλη (ή στήλες) προσδιορίζει τα δεδομένα που βρίσκονται στις υπόλοιπες.

Ένα παράδειγμα αντικειμένου data.frame με δεδομένα σε wide διάταξη είναι το σύνολο δεδομένων iris που έχουμε ήδη χρησιμοποιήσει σε προηγούμενες ενότητες (για περιγραφή των δεδομένων αυτών βλ. και Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων). Αν ανακληθεί το αντικείμενο iris (τύπου data.frame) εμφανίζεται:

```
> iris
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          5.1         3.5          1.4         0.2     setosa
2          4.9         3.0          1.4         0.2     setosa
3          4.7         3.2          1.3         0.2     setosa
…
148        6.5         3.0          5.2         2.0  virginica
149        6.2         3.4          5.4         2.3  virginica
150        5.9         3.0          5.1         1.8  virginica
```

Ο πρώτος αριθμός που εμφανίζεται δεν είναι μέρος των δεδομένων, αλλά το όνομα της γραμμής που ακολουθεί[242]. Το iris είναι ένας πίνακας 150 γραμμών και 5 στηλών. Τα στοιχεία στις 4 πρώτες στήλες είναι numeric ενώ η 5η στήλη είναι factor. Κάθε γραμμή καταγράφει τα δεδομένα της εξέτασης ενός φυτού iris, για 150 διαφορετικά φυτά που ανήκουν σε 3 διαφορετικά είδη. Σε κάθε φυτό έχουν μετρηθεί 4 χαρακτηριστικά του άνθους του, ενώ στην 5η μεταβλητή έχει καταγραφεί το είδος του. Οι τιμές αυτών των 5 μεταβλητών για ένα φυτό σχηματίζουν μία γραμμή του πίνακα.

Δοκιμάστε:	**Σχόλιο**
`iris[2,1]`	Πρόσβαση με δείκτες πίνακα, στοιχείο 2ης γραμμής, 1ης στήλης , επιστρέφει 4.9.
`iris$Sepal.Length[2]`	Ίδιο με το παραπάνω[243], μεταβλητή Sepal.Length, 2ο στοιχείο, επιστρέφει 4.9.
`iris[,1]`	Η 1η στήλη του iris, επιστρέφει c(5.1, 4.9, 4.7 …).
`iris$Sepal.Length`	Ίδιο με το παραπάνω, μεταβλητή Sepal.Length, επιστρέφει c(5.1, 4.9, 4.7 …).
`iris[2,]`	Επιστρέφει data.frame με μόνο την 2η γραμμή.
`iris[1]`	Επιστρέφει data.frame με μόνο την 1η στήλη.
`iris["Sepal.Length"]`	Ίδιο με παραπάνω, επιστρέφει data.frame με μόνο την (1η) στήλη Sepal.Length.
`iris[c(1,3,5)]`	Επιστρέφει data.frame με τις στήλες 1, 3 και 5.
`iris[c(5,1:4)]`	Επιστρέφει data.frame με αναδιάταξη των στηλών (η 5η έγινε 1η).
`iris[-1]`	Επιστρέφει data.frame με όλες εκτός της 1ης στήλης.
`iris[1]+10`	Επιστρέφει data.frame, με στοιχεία της 1ης στήλης + 10.
`iris$Sepal.Length+10`	Όπως το παραπάνω, επιστρέφει c(15.1, 14.9, 14.7, …).

[241] Τα αντικείμενα data.frame μοιάζουν στον τομέα αυτό με τους πίνακες που καταγράφονται μέσα σε σχεσιακές βάσεις δεδομένων, καθώς κάθε στήλη περιέχει στοιχεία ίδιου τύπου.

[242] Μπορεί να αλλαχθεί με τη συνάρτηση row.names.

[243] Παρατηρήστε πως η σύνταξη της εντολής χρησιμοποιεί το data.frame ως ένα list με στοιχεία τύπου vector (όπου κάθε vector είναι μία στήλη, δηλαδή μια μεταβλητή των δεδομένων). Ένα data.frame (όπως το iris) είναι list, κάτι που επιβεβαιώνεται αν εκτελέσετε την εντολή is.list(iris) το οποίο επιστρέφει TRUE.

Οι εντολές αυτές επιβεβαιώνουν πως ο χειρισμός αντικειμένων data.frame είναι παρεμφερής αυτών που περιγράφονται για τα matrix και για τα list (βλ. τις αντίστοιχες ενότητες). Υποστηρίζονται συναρτήσεις λίστας, όπως ο τελεστής $ ή η διαγραφή μιας στήλης με ανάθεση σε αυτή του NULL. Ταυτόχρονα υποστηρίζονται συναρτήσεις πίνακα όπως οι τελεστές [], οι συναρτήσεις View, head και tail για εμφάνιση, οι edit και fix για αλλαγή τιμών διαδραστικά (στο Data editor, βλ. Εικόνα 4.1), οι rbind και cbind για προσθήκη γραμμών ή στηλών, οι nrow, ncol και dim για διαστάσεις, οι rownames (ή row.names), colnames (ή names) και dimnames για τις ονομασίες γραμμών ή στηλών (μεταβλητών), συναρτήσεις τύπου apply για επεξεργασία κλπ. Επιπροσθέτως, κάθε στήλη είναι atomic vector επιτρέποντας αντίστοιχους χειρισμούς (π.χ. αριθμητικές πράξεις αν είναι σε numeric mode), συναρτήσεις όπως οι which και mode, ανακύκλωση τιμών όταν γίνεται επεξεργασία δύο αντικειμένων κλπ. Ακολουθούν μερικά παραδείγματα. Η παρακάτω εντολή (iris[iris[1]>5.8 & iris[4]>2.4,]) επιλέγει τις γραμμές του iris με τιμές στην 1η στήλη μεγαλύτερες του 5.8 και ταυτόχρονα στην 4η στήλη μεγαλύτερες του 2.4. Το αποτέλεσμα επιστρέφεται ως data.frame:

```
> iris[iris[1]>5.8 & iris[4]>2.4,]
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
101          6.3         3.3          6.0         2.5 virginica
110          7.2         3.6          6.1         2.5 virginica
145          6.7         3.3          5.7         2.5 virginica
```

Στο επόμενο παράδειγμα η εντολή apply(iris[1:4],2,mean) υπολογίζει τη μέση τιμή (αριθμητικός μέσος όρος) κάθε στήλης (για τις στήλες 1 έως 4) του iris. Ίδιο αποτέλεσμα θα επιστρέψει και η εντολή colMeans(iris[1:4]):

```
> apply(iris[1:4],2,mean)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width 
    5.843333     3.057333     3.758000     1.199333 
```

Παρακάτω αξιοποιείται η συνάρτηση aggregate για να βρεθεί η μέση τιμή κάθε στήλης (για τις στήλες 1 έως 4) ανά είδος:

```
> aggregate(iris[1:4],iris[5],mean)
     Species Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa        5.006       3.428        1.462       0.246
2 versicolor        5.936       2.770        4.260       1.326
3  virginica        6.588       2.974        5.552       2.026
```

Αν τώρα χρησιμοποιηθεί το όνομα της μεταβλητής, θα γίνει επεξεργασία μόνο μιας στήλης. Για παράδειγμα, η παρακάτω εντολή υπολογίζει τη μέση τιμή της μεταβλητής Petal.Width (στήλη 4) ανά είδος[244]:

```
> aggregate(Petal.Width ~ Species, data=iris, mean)
     Species Petal.Width
1     setosa       0.246
2 versicolor       1.326
3  virginica       2.026
```

Τα δεδομένα που βρίσκονται σε αντικείμενα τύπου data.frame μπορούν να αντιγραφούν σε matrix με τη συνάρτηση as.matrix. Καθώς όμως το matrix είναι atomic τύπος, απαιτεί να υπάρχει ένας κοινός τύπος στα στοιχεία που περιέχει. Έτσι θα γίνει μετατροπή (coercion) των δεδομένων του data.frame σε έναν ενιαίο τύπο (η διαδικασία περιγράφεται στη §4.1.1 Ο τύπος vector (διάνυσμα)). Για παράδειγμα, η εντολή as.matrix(iris) θα επιστρέψει πίνακα σε character mode, καθώς αυτός είναι ο τύπος στον οποίο μπορούν να μετατραπούν τόσο οι αριθμητικές στήλες του iris (στήλες 1 έως 4) όσο και το factor (στήλη 5). Πάντως, αν ζητούμενο είναι να γίνει μετατροπή ενός data.frame σε matrix σε numeric mode (με αριθμητικές τιμές), χρήσιμη είναι η συνάρτηση **data.matrix**:

```
> data.matrix(iris)
     Sepal.Length Sepal.Width Petal.Length Petal.Width Species
  [1,]          5.1         3.5          1.4         0.2       1
  [2,]          4.9         3.0          1.4         0.2       1
  [3,]          4.7         3.2          1.3         0.2       1
...
```

[244] Χρησιμοποιείται ο τελεστής ~ για τη δημιουργία ενός formula. Για περισσότερα σχετικά με τα formula βλ.§4.3.4 Αντικείμενα τύπου language (expression, call, name και formula).

```
[148,]          6.5          3.0          5.2          2.0          3
[149,]          6.2          3.4          5.4          2.3          3
[150,]          5.9          3.0          5.1          1.8          3
```

Τα data.frame, υποστηρίζουν επίσης (μεταξύ άλλων) τη συνάρτηση split[245], που επιτρέπει να αντιγραφούν τα στοιχεία σε ένα list, διαχωρισμένα ως προς την τιμή κάποιου διαχωριστή (βλ.§4.2.1 Ο τύπος list (λίστα)). Π.χ. η παρακάτω εντολή επιστρέφει ένα list που χωρίζει τα δεδομένα του iris ως προς τις τιμές στη στήλη Species:

```
split(iris,iris$Species)
```

Τα τρία στοιχεία του επιστραφέντος list είναι ονομασμένα "setosa", "versicolor" και "virginica" (τα τρία είδη φυτών στο σετ δεδομένων iris) και καθένα είναι data.frame που περιέχει μόνο τα δεδομένα που αφορούν το σχετικό είδος.

Αντικείμενα τύπου data.frame δημιουργούνται συνήθως κατά την ανάκληση δεδομένων που προέρχονται από κάποια εξωτερική πηγή, όπως π.χ. κάποιο αρχείο (βλ.§9.1 Εισαγωγή και εξαγωγή δεδομένων), αλλά μπορούν να δημιουργηθούν και μέσω κώδικα. Για να δημιουργηθεί ένα data.frame από κώδικα, χρησιμοποιείται η συνάρτηση **data.frame**. Η συνάρτηση δέχεται ως παραμέτρους τα vectors ή factors που θα αντιγραφούν στις στήλες του data.frame που θα δημιουργηθεί. Η εισαγωγή atomic vectors η factors σε στήλες γίνεται χωρίς αλλαγές, ενώ το όνομα τους ορίζει το όνομα της στήλης. Η εισαγωγή όμως άλλων τύπων αντικειμένων σε στήλες ενός data.frame ακολουθεί πολύπλοκους κανόνες μετατροπής που δεν θα αναλυθούν εδώ, για περισσότερα βλ. help(data.frame). Ακολουθούν μερικά παραδείγματα δημιουργίας και χρήσης αντικειμένων data.frame με κώδικα:

Δοκιμάστε:	**Σχόλιο**
`d<-data.frame(p=c(6,2,4),q=c("a","b","a"))`	Στο d, ένα data.frame με δύο μεταβλητές (p, q).
`fix(d)`	Διαδραστική αλλαγή τιμών στοιχείων του d.
`d$p[2]<-3`	Η 2η τιμή της p να πάρει την τιμή 3.
`d[2,1]<-3`	Ίδιο με το παραπάνω.
`d$p[d$q=="a"]`	Τα p όπου το αντίστοιχο q είναι "a" (δηλ. c(6, 4)).
`names(d)[2]<-"r"`	Μετονομασία της μεταβλήτης (στήλης) q σε r.
`d$z<-1`	Προσθήκη στήλης z με όλα τα στοιχεία ίσα με 1.
`d$z<-NULL`	Διαγραφή της στήλης z.
`d["z"]<-3:1`	Προσθήκη στήλης z με στοιχεία 3,2 και 1.
`d["z"]<-NULL`	Διαγραφή της στήλης z.

Επίσης πολλοί άλλοι τύποι αντικειμένων μπορούν να μετατραπούν (με διαδικασία coerce) σε data.frame με τη συνάρτηση **as.data.frame**. Προφανές παράδειγμα είναι η μετατροπή ενός αντικειμένου matrix:

Δοκιμάστε:	**Σχόλιο**
`m<-matrix(0,nrow=4,ncol=2)`	Στο m πίνακας 4 γραμμών και 2 στηλών, με όλα τα στοιχεία 0.
`d<-as.data.frame(m)`	Μετατροπή σε data.frame, εδώ οι στήλες θα ονομαστούν V1,V2.

Μία άλλη δυνατότητα που δίνουν αντικείμενα με ονομασμένες μεταβλητές (όπως τα data.frame όπου κάθε στήλη έχει όνομα) είναι να χρησιμοποιηθούν ως ένα προσωρινό περιβάλλον εκτέλεσης των εντολών, αυτό δηλαδή που γίνεται και για αντικείμενα environment μέσω της συνάρτησης with (βλ.§4.2.2 Ο τύπος environment (περιβάλλον). Σε κάποιες περιπτώσεις διευκολύνει επίσης η σύνδεση των μεταβλητών ενός data.frame στη λίστα αναζήτησης αντικειμένων (μία διαδικασία που επίσης περιγράφεται στην ενότητα για το environment). Αυτό επιτρέπει άμεση πρόσβαση στις μεταβλητές μόνο με το όνομά τους και γίνεται με τις συναρτήσεις attach (για σύνδεση) και detach (για αποσύνδεση) των μεταβλητών του data.frame. Πάντως, βάσει της αρχής copy-on-modify που εφαρμόζεται στην R, αν γίνει κάποια αλλαγή στις τιμές των στοιχείων μιας τέτοιας μεταβλητής που προέκυψε μέσω attach, αυτές εφαρμόζονται σε ένα νέο αντικείμενο που δημιουργείται στο Global Environment με το ίδιο όνομα (άρα το πηγαίο data.matrix δεν επηρεάζεται):

Δοκιμάστε:	**Σχόλιο**
`d<-data.frame(p=c(6,2,4),q=c("a","b","a"))`	Στο d, ένα data.frame με δύο μεταβλητές (p, q).

[245] Έχει οριστεί η σχετική split.data.frame.

`with(d,sum(p))`	Αθροίζει τα στοιχεία του p στο «περιβάλλον» d.
`attach(d)`	Προσθέτει τις μεταβλητές του d στην αναζήτηση.
`p`	Πρόσβαση στη στήλη p του d, επιστρέφει c(6,2,4).
`p[2]<-30`	Αλλαγή στο p, εφαρμόζεται σε νέο p.
`p`	Το νέο αντικείμενο p είναι το c(6, 30, 4).
`d$p`	Η στήλη p του d παραμένει c(6, 2, 4).
`detach(d)`	Αποσυνδέει το d από την αναζήτηση.

Τα στοιχεία δύο αντικειμένων data.frame μπορούν να συνενωθούν σε ένα. Εφόσον τα αρχικά αντικείμενα έχουν ίδιο αριθμό γραμμών, απλή συγχώνευση στηλών δύο ή περισσοτέρων data.frame σε ένα νέο αντικείμενο μπορεί να γίνει με τη συνάρτηση data.frame ή τη συνάρτηση cbind. Αν όμως τα αρχικά data.frame μοιράζονται κοινά ονόματα στηλών (ή και γραμμών) μπορεί να γίνει δημιουργία ενός νέου data.frame με τρόπο παρεμφερή των πράξεων join στις σχεσιακές βάσεις δεδομένων. Η συνάρτηση **merge** επιτρέπει διάφορες τέτοιες συνενώσεις τύπου join. Ακολουθούν μερικά παραδείγματα πάνω σε τρία data.frame d1, d2 και d3 που ορίζονται ως:

```
d1<-data.frame(x=c(11,12,13,14),m=c("Ιαν","Φεβ","Μαρ","Απρ"))
d2<-data.frame(y=c(21,22,23,24),m=c("Ιαν","Απρ","Απρ","Σεπ"))
d3<-data.frame(z=c(31,32,33),   m=c("Ιαν","Μαρ","Σεπ"))
```

Απλή συγχώνευση στηλών από τα d1 και d2 (που έχουν ίδιο αριθμό γραμμών) σε νέο data.frame (η εντολή cbind(d1,d2) έχει το ίδιο αποτέλεσμα):

```
> data.frame(d1,d2)
   x   m  y m.1
1 11 Ιαν 21 Ιαν
2 12 Φεβ 22 Απρ
3 13 Μαρ 23 Απρ
4 14 Απρ 24 Σεπ
```

Συνένωση των στηλών από τα d1 και d2 με τη συνάρτηση merge. με βάση τα δεδομένα στην κοινή στήλη m. Εδώ, η εντολή θα μπορούσε να γραφεί με απλούστερο τρόπο, δηλαδή ως merge(d1,d2, sort=F):

```
> merge(d1,d2,by="m",sort=F)
    m  x  y
1 Ιαν 11 21
2 Απρ 14 22
3 Απρ 14 23
```

Αντίστοιχα, συνένωση των στοιχείων από τα d1 και d3 με τη συνάρτηση merge και πάλι με βάση τα δεδομένα στην κοινή μεταβλητή m:

```
> merge(d1,d3,by="m",sort=F)
    m  x  z
1 Ιαν 11 31
2 Μαρ 13 32
```

Η συγχώνευση γραμμών δύο ή περισσότερων data.frame μπορεί να γίνει με τη συνάρτηση rbind, αλλά αυτό απαιτεί στήλες με κοινό όνομα και συμβατό τύπο δεδομένων (αλλιώς θα εφαρμοστεί coercion):

```
names(d2)[1]<-"x"      # μετονομασία στήλης y του d2 σε x
names(d3)[1]<-"x"      # μετονομασία στήλης z του d3 σε x
d4<-rbind(d1,d2,d3)
```

Αν ανακληθεί το παραπάνω data.frame d4 εμφανίζεται:

```
> d4
    x   m
1  11 Ιαν
2  12 Φεβ
3  13 Μαρ
4  14 Απρ
5  21 Ιαν
6  22 Απρ
7  23 Απρ
8  24 Σεπ
```

```
9  31 Ιαν
10 32 Μαρ
11 33 Σεπ
```

Με τη συνάρτηση **unstack** μπορούν να συγκεντρωθούν όλες οι τιμές μιας μεταβλητής που έχουν ίδια τιμή σε άλλη. Για παράδειγμα, στο d4, αν θεωρήσουμε πως οι τιμές του x εξαρτώνται από τον μήνα m (x~m)[246], είναι χρήσιμο να συγκεντρωθούν όλα τα x για τον ίδιο μήνα:

```
> unstack(d4,x~m)
$Απρ
[1] 14 22 23

$Ιαν
[1] 11 21 31

$Μαρ
[1] 13 32

$Σεπ
[1] 24 33

$Φεβ
[1] 12
```

Η παραπάνω εφαρμογή της unstack επέστρεψε ένα αντικείμενο list όπου κάθε στοιχείο είναι μια ομάδα τιμών. Ένα τέτοιο αντικείμενο μπορεί να δοθεί ως παράμετρος στη συνάρτηση **stack** η οποία θα επιστρέψει ένα data.frame δύο στηλών. Εδώ η stack θα επιστρέψει ένα data.frame με τα δεδομένα του d4.

Η συνάρτηση reshape (του προεγκατεστημένου πακέτου ‘stats’) στοχεύει στη μετατροπή δεδομένων από διάταξη long σε διάταξη wide και αντίστροφα. Το επόμενο παράδειγμα δημιουργεί ένα data.frame με όνομα sales που περιέχει υποθετικά δεδομένα πωλήσεων από ένα κατάστημα: για μία περίοδο 5 ημερών σε κάθε πώληση καταγράφεται ο κωδικός του πελάτη που έκανε την αγορά (ΠΕΛΑ), το προϊόν και την ποσότητα που αγόρασε (ΠΡΟΙ και ΠΟΣΟ) και η ημέρα (ΗΜΕΡ) που έγιναν η αγορές αυτές (ας θεωρήσουμε πως κάθε πελάτης ψωνίζει μόνο μια φορά την ημέρα από το κατάστημα αυτό):

```
sales <- data.frame(ΠΕΛΑ=c(2,1,1,2,2,3),
                    ΠΡΟΙ=c("α","β","α","γ","β","β"),
                    ΠΟΣΟ=c(21,11,12,22,23,31),
                    ΗΜΕΡ=c(1,1,3,4,5,5))
```

Αν το προς μελέτη υποκείμενο είναι οι πελάτες (ή τα προϊόντα), το data.frame sales είναι σε διάταξη long καθώς για κάθε περίπτωση πελάτη (ή προϊόντος) μπορεί να σχετίζεται με παραπάνω από μια γραμμές στα δεδομένα. Αν ανακληθεί το s1, επιστρέφεται:

```
> sales
  ΠΕΛΑ ΠΡΟΙ ΠΟΣΟ ΗΜΕΡ
1    2    α   21    1
2    1    β   11    1
3    1    α   12    3
4    2    γ   22    4
5    2    β   23    5
6    3    β   31    5
```

Για να μετατραπούν τα δεδομένα πωλήσεων σε wide, π.χ. σε έναν πίνακα όπου κάθε γραμμή αντιστοιχεί σε έναν πελάτη, μπορεί να χρησιμοποιηθεί η παρακάτω εντολή (που επίσης αποθηκεύει το αποτέλεσμα στο data.frame με όνομα sales2):

```
sales2 <- reshape(sales,
              direction = "wide",
              idvar = "ΠΕΛΑ",
```

[246] Χρησιμοποιείται ο τελεστής ~ για τη δημιουργία ενός formula. Για περισσότερα σχετικά με τα formula βλ.§4.3.4 Αντικείμενα τύπου language (expression, call, name και formula).

```
                v.names = c("ΠΟΣΟ","ΠΡΟΙ"),
                timevar = "ΗΜΕΡ")
```

Το αποτέλεσμα (ταξινομημένο ως προς τον κωδικό πελάτη) είναι:

```
> sales2[order(sales2$ΠΕΛΑ),]
  ΠΕΛΑ ΠΟΣΟ.1 ΠΡΟΙ.1 ΠΟΣΟ.3 ΠΡΟΙ.3 ΠΟΣΟ.4 ΠΡΟΙ.4 ΠΟΣΟ.5 ΠΡΟΙ.5
2    1     11      β     12      α     NA   <NA>     NA   <NA>
1    2     21      α     NA   <NA>     22      γ     23      β
6    3     NA   <NA>     NA   <NA>     NA   <NA>     31      β
```

Σε όσες περιπτώσεις δεν υπήρχαν πωλήσεις έχει τοποθετηθεί η τιμή NA. Ως αποτέλεσμα της reshape, το sales2 είναι ένα data.frame με κάποιες πρόσθετες ιδιότητες[247] που βοηθούν στην επαναφορά του σε long διάταξη και έτσι η σχετική εντολή είναι ιδιαίτερα απλουστευμένη:

```
> reshape(sales2, direction = "long")
    ΠΕΛΑ ΗΜΕΡ ΠΟΣΟ ΠΡΟΙ
2.1    2    1   21    α
1.1    1    1   11    β
3.1    3    1   NA <NA>
2.3    2    3   NA <NA>
1.3    1    3   12    α
3.3    3    3   NA <NA>
2.4    2    4   22    γ
1.4    1    4   NA <NA>
3.4    3    4   NA <NA>
2.5    2    5   23    β
1.5    1    5   NA <NA>
3.5    3    5   31    β
```

Λόγω του σημαντικού ρόλου των data.frame στην R, μεγάλος αριθμός πρόσθετων πακέτων προσφέρει εκτεταμένες δυνατότητες επεξεργασίας αντικειμένων αυτού του τύπου (ή εξελίξεών του όπως ο τύπος tibble που αναφέρεται παρακάτω). Μεταξύ πλήθους άλλων, άξια αναφοράς είναι τα πακέτα 'dplyr' [55], 'reshape2' [56] και 'tidyr' που περιέχονται στο πακέτο 'tidyverse' [57]. Έτσι, π.χ. το πακέτο 'reshape2' περιέχει (μεταξύ άλλων) συναρτήσεις που απλοποιούν τη διαδικασία μετατροπής από long σε wide, υποκαθιστώντας την ανάγκη χρήσης της προαναφερθείσας συνάρτησης reshape του πακέτου 'stats'.

4.3 Άλλοι τύποι αντικειμένων

Πολλοί χρήσιμοι τύποι αντικειμένων ορίζονται σε πακέτα που είτε συνοδεύουν την R (ως μέρους του system library) είτε προστίθενται από τον χρήστη (στο user library). Ακολουθεί μια σύντομη περιγραφή κάποιων τέτοιων επιλεγμένων τύπων αντικειμένων.

4.3.1 Οι τύποι tibble και data.table

Οι τύποι tibble και data.table είναι βασισμένοι στα data.frame[248] και μπορούν να χρησιμοποιηθούν αντί αυτών για πινακοειδείς δομές δεδομένων. Οι δύο τύποι παρέχουν νέες συναρτήσεις αλλά υποστηρίζουν και (συχνά βελτιωμένες) συναρτήσεις των data.frame. Έτσι συνήθως μπορούν να αντικαταστήσουν τα data.frame χωρίς ιδιαίτερες αλλαγές στον κώδικα. Επιπρόσθετα, τα data.table είναι προσανατολισμένα στη βελτίωση των δυνατότητων χειρισμού μεγάλων συνόλων δεδομένων.

4.3.1.1 Ο τύπος tibble

Ο τύπος **tibble** περιέχεται στο ομώνυμο πακέτο 'tibble' [58] το οποίο είναι και μέρος της δημοφιλούς συλλογής πακέτων tidyverse [57][249]. Μπορεί να χρησιμοποιηθεί ως άμεσος αντικαταστάτης του τύπου data.frame, τις περισσότερες φορές χωρίς ιδιαίτερες αλλαγές στον κώδικα (πλην της προφανούς αντικατάστασης του λεκτικού

[247] βλ.§4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων.
[248] βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων).
[249] Τα πακέτα μπορούν να εγκατασταθούν από το CRAN, βλ. §1.5 Χρήση και διαχείριση πακέτων.

"data.frame" από το "tibble"). Τα αντικείμενα tibble είναι data.frame[250] με διαφοροποιημένες, πρόσθετες και βελτιωμένες συναρτήσεις, άρα συνήθως (αλλά όχι πάντα) μπορούν να εφαρμοστούν σε αυτά οι ίδιες συναρτήσεις και τελεστές. Τα παραδείγματα που αναφέρονται για το data.frame στη σχετική ενότητα, λειτουργούν με ελάχιστες αλλαγές και για αντικείμενα tibble[251].

Μερικές ενδεικτικές διαφορές των αντικειμένων τύπου tibble από τα data.frame είναι:

(α) Τα tibble επιτρέπουν στήλες που είναι list (όχι μόνο atomic vectors). Μια τέτοια στήλη μπορεί να περιέχει οποιονδήποτε συνδυασμό από στοιχεία διαφορετικών τύπων, ενώ επιτρέπεται κάθε τύπος αντικειμένου (όχι μόνο οι βασικοί). Το tibble δεν κάνει αυτόματα μετατροπές στον τύπο των δεδομένων που αποθηκεύονται σε αυτό. Έτσι π.χ. δεν αλλάζουν σε factor δεδομένα τύπου character. Ως παράδειγμα, το παρακάτω tibble d1 ορίζεται με τη δεύτερη στήλη του (με όνομα b) να είναι μια λίστα ανομοιογενούς τύπου αντικειμένων:

```
d1<-tibble( a = c(6,2,4),
            b = c(TRUE, c(11,21,23), "Athens"),
            c = c("a","b","a") )
```

Κάθε στοιχείο στη 2η στήλη παραμένει ως έχει και δεν γίνεται μετατροπή (coercion). Άρα η εντολή d1$b[[2]] αναφέρεται στο διάνυσμα c(11,21,23) (που είναι το 2ο στοιχείο της λίστας b), ενώ το d1$b[[2]][1] αναφέρεται στο στοιχείο αυτού του διανύσματος με τιμή 11.

(β) Κατά τη δημιουργία ενός tibble μια στήλη μπορεί να είναι αποτέλεσμα εφαρμογής συναρτήσεων πάνω στις προηγούμενες στήλες. Π.χ. μια νέα στήλη μπορεί να είναι αποτέλεσμα μαθηματικών υπολογισμών επί των άλλων:

```
d2<-tibble( a = c(30,10,20),
            b = c(100,200,300),
            c = a + sqrt(b) )
```

(γ) τα tibble εμφανίζονται με διαφορετικό τρόπο. Η μέθοδος print των tibble επιτρέπει να οριστούν ως παράμετροι ο αριθμός των γραμμών (παράμετρος n) και στηλών (παράμετρος width) που θα εμφανιστούν. Εξ ορισμού αν ένα αντικείμενο tibble έχει μεγάλο αριθμό γραμμών, εμφανίζονται μόνο οι αρχικές[252]:

```
> d2
# A tibble: 3 x 3
      a     b     c
  <dbl> <dbl> <dbl>
1    30   100  40
2    10   200  24.1
3    20   300  37.3
```

(δ) Τα tibble υποβοηθούν τη δυνατότητα ορισμού (μικρών συνήθως) data set μέσω κώδικα, επιτρέποντας την εισαγωγή των δεδομένων γραμμή-γραμμή. Αυτό επιτυγχάνεται με τη συνάρτηση **tribble** (transposed tibble), όπως στο επόμενο παράδειγμα:

```
d3<-tribble( ~a, ~b,
             10, 11,
             20, 21,
             31, 32  )
```

Στο παραπάνω παράδειγμα τα δεδομένα κάθε γραμμής του tibble γράφονται σε χωριστή γραμμή, αλλά αυτό δεν είναι απαραίτητο. Ισοδύναμη θα ήταν η εντολή d3 <- tribble(~a, ~b, 10, 11, 20, 21, 31, 32). Σε κάθε περίπτωση, η εντολή δημιουργεί tibble δύο στηλών (μεταβλητών a και b) γεμίζοντας κάθε γραμμή με τα δεδομένα που δόθηκαν:

---

[250] Η κλάση των αντικειμένων tibble είναι c("tbl_df", "tbl", "data.frame").

[251] Για να εκτελεστούν όλα τα παραδείγματα της σχετικής ενότητας (§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων)) με tibble αντί data.frame θα χρειαστούν αλλαγές σε δύο μόνο σημεία: (α) Στη μετατροπή matrix σε tibble μπορεί να χρησιμοποιηθεί η συνάρτηση as.tibble (αντί της as.data.frame) αλλά συνιστάται να χρησιμοποιηθεί η νεότερη συνάρτηση as_tibble. (β) Στη συνένωση δύο tibble d1, d2 σε ένα νέο, η συνάρτηση tibble (που θα χρησιμοποιηθεί αντί της data.frame) απαιτεί τιμή σε μια πρόσθετη παράμετρο (.name_repair) για τον χειρισμό της ύπαρξης στηλών με το ίδιο όνομα (της στήλης m) στα αρχικά d1,d2. Έτσι η εντολή θα ήταν tibble(d1,d2,.name_repair = "minimal") αντί της data.frame(d1,d2). Οι υπόλοιπες εντολές των παραδειγμάτων αυτών δεν χρειάζονται αλλαγές.

[252] Ο αριθμός αυτός (και άλλες ρυθμίσεις που σχετίζονται με τα tibble) μπορεί να οριστεί ως επιλογή συνεδρίας, (βλ.§4.2.1.5 Η λίστα επιλογών συνεδρίας). Οι σχετικές επιλογές έχουν όνομα tibble.print.min, tibble.print.max και tibble.print.width. Π.χ. με την εντολή options(tibble.print_max = 15, tibble.print_min = 10) ορίζεται η εμφάνιση έως 15 γραμμών για αντικείμενα tibble με πάνω από 10 γραμμές.

```
> d3
# A tibble: 3 x 2
      a     b
  <dbl> <dbl>
1    10    11
2    20    21
3    31    32
```

(ε) Κατά την εφαρμογή συναρτήσεων που αφορούν στήλες και άλλα αντικείμενα διαφορετικού μήκους, τα tibble δεν ανακυκλώνουν τιμές εκτός αν πρόκειται για μια μοναδική τιμή που θα επαναληφθεί.

(στ) Τα tibble έχουν λιγότερους περιορισμούς στα ονόματα στηλών που είναι αποδεκτά.

(ζ) Τα tibble έχουν συχνά βελτιωμένη απόδοση κατά την εκτέλεση κώδικα σε σχέση με τα data.frame.

Η μετατροπή άλλων συμβατών τύπων αντικειμένων σε tibble μπορεί να γίνει με τη συνάρτηση **as_tibble**, ενώ η βέλτιστη αξιοποίηση αντικειμένων tibble επιτυγχάνεται με τη χρήση των δυνατοτήτων που παρέχει η συλλογή πακέτων ‘tidyverse’ στην οποία ανήκει (και με την οποία είναι στενά συνδεδεμένος) ο συγκεκριμένος τύπος αντικειμένων.

4.3.1.2 Ο τύπος data.table

Ο τύπος **data.table** [59] περιέχεται στο ομώνυμο πακέτο ‘data,table’[253]. Όπως τα tibble, είναι βελτιωμένος τύπος data.frame. Έχει στόχο να υποβοηθήσει τον χρήστη στην επεξεργασία συνόλων δεδομένων που φορτώνονται στη μνήμη (in-memory processing) με μικρό ή μεγάλο αριθμό δεδομένων. Έτσι προσφέρει νέες συναρτήσεις αλλά και βελτιώνει το συντακτικό ή βελτιστοποιεί υπάρχουσες συναρτήσεις των data.frame. Η εκμετάλλευση των δυνατοτήτων του τύπου data.table επιτρέπει καλύτερες επιδόσεις και μειωμένους χρόνους επεξεργασίας σε σχέση με τη χρήση απλών data.frame ή tibble. Αυτό γίνεται πιο εμφανές όταν τα σύνολα δεδομένων που καλούνται να χειριστούν οι δομές αυτές είναι μεγάλα. Αξιοποιώντας τα χαρακτηριστικά αυτά, μεγάλος αριθμός πρόσθετων πακέτων επέκτασης που είναι διαθέσιμα από πηγές όπως το CRAN χρησιμοποιεί τον τύπο data.table και εξαρτάται από το σχετικό πακέτο.

Τα data.table είναι data.frame[254]. Άρα ο τύπος data.table κληρονομεί τις δυνατότητες των data.frame και μπορεί να τα υποκαταστήσει στον κώδικα χωρίς (ή με λίγες) αλλαγές. Έτσι, τα παραδείγματα της σχετικής ενότητας για τον τύπο data.frame ισχύουν αν αντικατασταθεί ο τύπος με data.table[255]. Όμως τα data.table παρέχουν βελτιστοποιήσεις απόδοσης στις συναρτήσεις εκμεταλλευόμενα τις υπολογιστικές δυνατότητες του συστήματος (π.χ. multithreading), βλ vignette("datatable-benchmarking"). Επίσης παρέχουν βελτιώσεις στο συντακτικό των εντολών που υποστηρίζουν. Αυτές περιλαμβάνουν μεταξύ άλλων:

(α) Νέες συναρτήσεις χειρισμού δεδομένων σε αρχεία, με βελτιωμένες δυνατότητες για τις περιπτώσεις μεγάλου αριθμού δεδομένων (όπως η συνάρτηση **fread** που είναι παρεμφερής με τη read.csv[256] για την εισαγωγή δεδομένων από αρχεία κειμένου).

(β) Νέες ή βελτιωμένες συναρτήσεις επεξεργασίας των δεδομένων. Κάποιες από τις βελτιώσεις είναι παρεμφερείς με αυτές των tibble, όπως π.χ. τα data.table εμφανίζονται (print) με διαφορετικό τρόπο (παραμετροποιήσιμο μέσω options) ή δεν μετατρέπουν αυτόματα στήλες character σε factor. Όμως υπάρχουν και πολλές αλλαγές που είναι ιδιαίτερες του τύπου data.table, όπως οι συναρτήσεις **setkey** και **setorder** (ταξινόμηση), **setcolorder**, **setnames**, ο τελεστής **like**. Σημαντικότατος είναι και ο βελτιωμένος τελεστής [] που παρουσιάζεται εκτενέστερα παρακάτω.

(γ) Αλλαγές στον τελεστή [] σε συνδυασμό με τον νέο τελεστή ανάθεσης τιμής **:=** που επιτρέπουν την επεξεργασία των στοιχείων σε ένα data.table με αποδοτικό τρόπο και χωρίς αντιγραφή όλου του αντικειμένου σε νέο όταν γίνονται αλλαγές. Έτσι, μπορεί να διατηρηθεί οποιαδήποτε αναφορά (reference) μπορεί να γίνει στο ίδιο αντικείμενο data.table στη μνήμη από πολλαπλές μεταβλητές και να γίνονται αλλαγές σε αυτό χωρίς να επηρεαστεί η πρόσβαση στο αντικείμενο από τις μεταβλητές αυτές. Ακολουθούν ενδεικτικά παραδείγματα για τα οποία το σύνολο δεδομένων iris[257] αντιγράφεται σε ένα αντικείμενο τύπου data.table με όνομα x. Σε ένα

[253] Το πακέτο είναι διαθέσιμο στο CRAN, βλ. §1.5 Χρήση και διαχείριση πακέτων.
[254] Η κλάση των αντικειμένων data.table είναι c(“data.table”,”data.frame”).
[255] Η μόνη αλλαγή που απαιτείται είναι στα σημεία όπου γίνεται χρήση του τελεστή [] με δείκτη κάποιο όνομα μεταβλητής. Ο τελεστής [] είναι διαφοροποιημένος στα data.table, βλ. παρακάτω.
[256] βλ.§9.1.1 Δεδομένα σε κείμενο.
[257] Βλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων.

αντικείμενο data.table ο τελεστής [] έχει διαφορετική συμπεριφορά από τον αντίστοιχο των data.frame και δέχεται πληθώρα παραμέτρων. Βασικές είναι οι πρώτες τρείς παράμετροι με όνομα i, j και by αντίστοιχα. Η 1η παράμετρος (i) αφορά την επιλογή γραμμών, η 2η παράμετρος (j) τους υπολογισμούς ή το αποτέλεσμα, ενώ η 3η παράμετρος (by) τις ομάδες πάνω στις οποίες θα εφαρμοστεί το j[258]:

Δοκιμάστε:	**Σχόλιο**
`x<-as.data.table(iris)`	Τα δεδομένα του iris αντιγράφονται σε data.table x.
`x[Sepal.Length>7.5 & Sepal.Width==3.8]`	Η 1η παράμετρος επιλέγει γραμμές μέσω κριτηρίων.

Η τελευταία εντολή επιστρέφει:

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1:          7.7         3.8          6.7         2.2 virginica
2:          7.9         3.8          6.4         2.0 virginica
```

Εντός του τελεστή [] των data.table οι στήλες είναι μεταβλητές. Αυτό από μόνο του απλοποιεί τις εκφράσεις, π.χ. η αντίστοιχη εντολή σε data.frame θα ήταν iris[iris[1]>7.5 & iris[2]==3.8,] ή iris[iris$Sepal.Length>7.5 & iris$Sepal.Width==3.8,]. Στο παράδειγμα επιλέχθηκαν μόνο γραμμές. Αν τώρα ζητούμενο είναι να επιστραφούν συγκεκριμένες στήλες των επιλεγμένων γραμμών, αυτές δίνονται στη 2η παράμετρο όπως για παράδειγμα:

```
x[ Sepal.Length>7.5 & Sepal.Width==3.8,
   list(Sepal.Length , Species) ]
```

που επιστρέφει τις επιλεγμένες στήλες :

```
   Sepal.Length   Species
1:          7.7 virginica
2:          7.9 virginica
```

Η εκτέλεση υπολογισμών είναι εφικτή με χρήση εκφράσεων. Η επόμενη εντολή ζητά το άθροισμα των γινομένων Sepal.Length * Petal.Width για τις επιλεγμένες γραμμές:

```
x[ Sepal.Length>7.5 & Sepal.Width==3.8,
   sum(Sepal.Length * Petal.Width) ]
```

το οποίο επιστρέφει 32.74 (δηλαδή 7.7 * 2.2 + 7.9 *2). Αν ήταν ζητούμενο να παραχθεί το αντίστοιχο αποτέλεσμα ανά είδος (Species) για όλες τις γραμμές, θα μπορούσε να αξιοποιηθεί η 3η παράμετρος του τελεστή [], όπως στην εντολή:

```
x[ ,
   sum(Sepal.Length * Petal.Width),
   Species ]
```

κάτι που επιστρέφει:

```
      Species     V1
1:     setosa  62.08
2: versicolor 396.29
3:  virginica 669.77
```

Ένα ακόμα παράδειγμα ομαδοποίησης αποτελεσμάτων με χρήση της 3ης παραμέτρου του τελεστή [] είναι η εντολή:

```
x[Sepal.Length>6, sum(Petal.Length<5), Species]
```

η οποία επιστρέφει ανά είδος και για τις περιπτώσεις με Sepal.Length > 5 πόσες είχαν Petal.Length < 5, δηλαδή:

```
      Species V1
1: versicolor 19
2:  virginica  3
```

Τα παραπάνω μοιάζουν με τα αποτελέσματα μιας εντολής SELECT στη γλώσσα SQL των σχεσιακών βάσεων δεδομένων. Όμως ο τελεστής [] μπορεί να χρησιμοποιηθεί και ως UPDATE, δηλαδή για την αλλαγή

[258] Όπως αναφέρει και η τεκμηρίωση του πακέτου ΄data.table' ο τελεστής [] έχει κάποια ομοιότητα με τις εντολές SELECT και UPDATE της γλώσσας SQL στις σχεσιακές βάσεις δεδομένων. Το i ορίζει τα υποσύνολα ή την ταξινόμηση των δεδομένων πάνω στα οποία υπολογίζεται το j λαμβάνοντας υπόψη την ομαδοποίηση που ορίζει το by. Έτσι μπορούν να επιτευχθούν αποτελέσματα παρεμφερή με αυτά π.χ. μιας εντολής SELECT (j) WHERE (i) GROUP BY (by).

των τιμών σε στοιχεία του data.table. Ο κώδικας αλλαγής τιμών θα οριστεί στη 2η παράμετρο j και θα χρησιμοποιεί εκεί τον τελεστή **:=** για επεξεργασία και ανάθεση τιμών που αφορά ολόκληρες στήλες. Το επόμενο είναι ένα παράδειγμα ανάθεσης τιμών σε στοιχεία του data.table μέσω του τελεστή **:=** (αντί του <-):

```
x[,Sepal.Length:=100]
x[2,Sepal.Length:=50]
x[Petal.Length<1.4,Sepal.Width:=Sepal.Width*100]
```

Η πρώτη εντολή παραπάνω θέτει την τιμή 100 σε όλα τα στοιχεία της στήλης Sepal.Length. Η επόμενη εντολή αναθέτει το 50 στο 2ο στοιχείο του Sepal.Length. Η τελευταία εντολή πολλαπλασιάζει επί 100 το Sepal.Width που αντιστοιχεί σε γραμμές με Petal.Length<1.4. Το αποτέλεσμα στο x είναι:

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1:          100         3.5          1.4         0.2    setosa
2:           50         3.0          1.4         0.2    setosa
3:          100       320.0          1.3         0.2    setosa
4:          100         3.1          1.5         0.2    setosa
(…)
```

Χρησιμοποιώντας τον ίδιο τρόπο (με τελεστή := στη δεύτερη παράμετρο j του []), μπορούν να δημιουργηθούν ή να διαγραφούν μεταβλητές (στήλες) στο data.table. Για παράδειγμα:

```
x[,PLplus10:=Petal.Length+10]
x[,c("Petal.Width","Petal.Length"):=NULL]
```

Η πρώτη εντολή δημιουργεί νέα στήλη με όνομα PLplus10 (με τις αντίστοιχες τιμές), ενώ η δεύτερη διαγράφει τις στήλες Petal.Width και Petal.Length από το x, το οποίο πλέον γίνεται:

```
> x
   Sepal.Length Sepal.Width  Species PLplus10
1:          100         3.5   setosa     11.4
2:           50         3.0   setosa     11.4
3:          100       320.0   setosa     11.3
4:          100         3.1   setosa     11.5
(…)
```

Ο τελεστής := δεν χρησιμοποιείται από την ίδια την R και το data.table αναλαμβάνει να εκτελέσει την έκφραση που έχει οριστεί στη 2η παράμετρο j του τελεστή []. Αυτό επιτρέπει στο data.table να εφαρμόσει τις αλλαγές εσωτερικά πάνω στο υπάρχον αντικείμενο, χωρίς τις υπολογιστικά ακριβές (άρα χρονοβόρες) αντιγραφές δεδομένων που επιβάλει συχνά η αρχή copy-on-modify της R[259]. Επιπροσθέτως, επειδή το αντικείμενο δεν καταγράφεται από την R ως αλλαγμένο, οι αλλαγές θα ισχύουν για όλες τις μεταβλητές που αναφέρονται σε αυτό. Οι αλλαγές τιμών δεν θα προκαλέσουν απώλεια της πρόσβασης από κάποια μεταβλητή στο ίδιο αντικείμενο, κάτι που επιτρέπει reference semantics[260]. Στο παρακάτω παράδειγμα γίνεται ανάθεση τιμών μέσω τελεστών της R:

Δοκιμάστε:	**Σχόλιο**
`x<-as.data.table(iris)`	Νέο αντικείμενο data.table, με όνομα x που περιέχει τα δεδομένα του iris.
`y<-x`	Το y αναφέρεται στο ίδιο αντικείμενο με το x.
`x$Sepal.Length<-100`	Αλλαγή τιμών (μέσω τελεστή <-) της στήλης Sepal.Length του x σε 100.
`x$Sepal.Length[2]<-50`	Αλλαγή τιμής σε 50 (μέσω τελεστή <-) του 2ου στοιχείου στο Sepal.Length.
`x$Sepal.Length`	Το Sepal.Length του x, επιστρέφει c(100,50,100, … ,100).
`y$Sepal.Length`	Το Sepal.Length του y, επιστρέφει c(5.1, 4.9, 4.7, 4.6, … ,5.9).

Αυτή είναι η αναμενόμενη συμπεριφορά αντικειμένων στην R. Όμως, με αξιοποίηση των δυνατοτήτων του data.table μπορεί να διατηρηθεί η αναφορά στο ίδιο αντικείμενο τόσο από τη μεταβλητή x όσο και από τη μεταβλητή y:

Δοκιμάστε:	**Σχόλιο**
`x<-as.data.table(iris)`	Νέο αντικείμενο data.table, με όνομα x που περιέχει τα δεδομένα του iris.
`y<-x`	Το y αναφέρεται στο ίδιο αντικείμενο με το x.

[259] Δημιουργείται ένα νέο αντικείμενο όταν γίνεται ανάθεση τιμής με τους τελεστές της R (όπως ο τελεστής <-).

[260] Για άλλους τύπους αντικειμένων στα οποία επιτρέπεται η επεξεργασία μέσω αναφοράς βλ.§4.2.2.2 Περιβάλλοντα και ο μηχανισμός αναφοράς και §6.5 Κλάσεις αναφοράς.

`x[,Sepal.Length:=100]`	Αλλαγή τιμών (μέσω τελεστή :=) της στήλης Sepal.Length του x σε 100.
`x[2,Sepal.Length:=50]`	Αλλαγή τιμής σε 50 (μέσω τελεστή :=) του 2ου στοιχείου στο Sepal.Length.
`x$Sepal.Length`	Το Sepal.Length του x, επιστρέφει c(100,50,100,…,100).
`y$Sepal.Length`	Το Sepal.Length του y είναι το ίδιο με του x, επίσης c(100,50,100,…,100).

Τα παραδείγματα αυτά είναι μόνο ενδεικτικά των δυνατοτήτων που παρέχουν τα data.table. Περισσότερα σχετικά με τον τύπο αντικειμένων data.table υπάρχουν στο [60] καθώς και στην εκτενή τεκμηρίωση του σχετικού πακέτου. Από αυτή, το εισαγωγικό vignette εμφανίζεται εκτελώντας την εντολή vignette("datatable-intro").

4.3.2 Ο τύπος Matrix (αραιοί και πυκνοί πίνακες)

Όπως και άλλοι τύποι αντικειμένων που έχουν προαναφερθεί (matrix, data.frame, tibble, data.table), ο τύπος Matrix επίσης αφορά πινακοειδείς δομές δεδομένων. Όταν μεγάλο μέρος από τα στοιχεία ενός πίνακα περιέχουν την τιμή 0 ονομάζονται «αραιοί» (sparce matrix ή sparce array). Αντίθετα, όταν τα περισσότερα στοιχεία περιέχουν μη-μηδενικές τιμές, οι πίνακες χαρακτηρίζονται ως «πυκνοί» (dense). Υπάρχουν πολλά προβλήματα στα οποία χρησιμοποιούνται αραιοί πίνακες και η χρήση τύπων όπως το matrix (βλ.§4.1.3.1 Ο τύπος matrix.) για τον χειρισμό τους δεν είναι αποδοτική. Η αποθήκευση στη μνήμη της πληθώρας των τιμών 0 είναι ουσιαστικά άσκοπη και ίσως αδύνατη αν ο πίνακας έχει πολύ μεγάλες διαστάσεις. Το πακέτο ‘Matrix’ που υπάρχει σε κάθε εγκατάσταση[261] της R, παρέχει τον τύπο Matrix[262] (με κεφαλαίο το αρχικό Μ) καθώς και άλλους σχετικούς τύπους αντικειμένων που βασίζονται στον τύπο αυτό και είναι υποκλάσεις του Matrix. Κάθε τέτοιος τύπος αντικειμένων είναι περισσότερο κατάλληλος για συγκεκριμένες εφαρμογές, μαθηματικές πράξεις κλπ. Κάποιοι είναι ιδιαίτερα κατάλληλοι για αραιούς πίνακες καθώς εκμεταλλεύονται την ύπαρξη επαναλαμβανόμενων τιμών και ελαχιστοποιούν τη μνήμη που απαιτείται για την αποθήκευση τους. Οι παραλλαγές (υποκλάσεις) του τύπου Matrix περιλαμβάνουν τα:

- dgCMatrix για αραιούς πίνακες με πραγματικούς (διπλής ακριβείας) αριθμούς. Το d σημαίνει double και αφορά το είδος των δεδομένων ενώ το C αναφέρεται στο compression (συμπίεση). Τα δεδομένα θα είναι αποθηκευμένα με συμπίεση ανά στήλη. Ένας τέτοιος πίνακας έχει μορφή CSC (Compressed Sparse Column) και αποθηκεύει μόνο τη γραμμή και τιμή των μη-μηδενικών στοιχείων ανά στήλη (κατά σειρά στηλών).
- dgeMatrix που είναι ο βασικός τύπος Matrix για πυκνούς πίνακες με πραγματικούς αριθμούς.
- dsCMatrix για πραγματικούς αριθμούς σε συμμετρικό πίνακα αποθηκευμένους με συμπίεση (το s σημαίνει symmetric).
- lgCMatrix για αραιούς πίνακες με λογικές (logical) τιμές.

Εκτός των παραπάνω, στο πακέτο ‘Matrix’ περιέχονται και άλλοι τύποι για συμμετρικούς, τριγωνικούς διαγώνιους πίνακες, για αραιά διανύσματα κλπ. Ως παράδειγμα του τελευταίου, η εντολή sparseVector(c(20,40),c(2,25),30) δημιουργεί ένα διάνυσμα τύπου dsparseVector 30 στοιχείων (πραγματικών αριθμών), όπου όλες οι τιμές των στοιχείων του έχουν τιμή 0 εκτός των στοιχείων στις θέσεις 2 και 25 που περιέχουν τις τιμές 20 και 40 αντίστοιχα. Οι τιμές 0 δεν καταλαμβάνουν χώρο μνήμης σε ένα τέτοιο διάνυσμα.

Οι κλάσεις αυτές παρέχουν συναρτήσεις (μεθόδους) για την επεξεργασία των δεδομένων καθώς και συναρτήσεις που υποβοηθούν τη χρήση τέτοιων αντικειμένων, τη μετατροπή σε άλλους τύπους, τη μεταφορά δεδομένων από και προς αρχεία (συναρτήσεις **readMM** και **writeMM**) κλπ. Το παρακάτω παράδειγμα αναδεικνύει την επίπτωση που έχει η χρήση Matrix στη δέσμευση μνήμης για την αποθήκευση (με συμπίεση) ενός πίνακα.

Δοκιμάστε:	**Σχόλιο**
`m1<-matrix(0,1000,10)`	Δημιουργία του m1, ενός matrix 1000x10 με τιμές στοιχείων 0.
`object.size(m1)`	Ο χώρος που καταλαμβάνει το m1, επιστρέφει περίπου 78.3Kb.
`s1<-Matrix(m1,sparse=T)`	Αντιγραφή του (αραιού) m1 σε αντικείμενο Matrix με όνομα s1.
`object.size(s1)`	Ο χώρος που καταλαμβάνει το s1, περίπου 1.5Kb.
`m2=matrix(1.5:10000.5,ncol=10)`	Στο m2 ένα matrix 1000x10 με τιμές 1.5, 2.5, 3.5,…, 10000.5.

[261] Ανήκει στα προ-εγκατεστημένα πακέτα. Συνδέεται στην τρέχουσα συνεδρία της R με την εντολή library(Matrix).

[262] Συγκεκριμένα είναι μια κλάση S4, βλ.§6.4 Κλάσεις S4. Η κλάση αντικειμένων dgcMatrix είναι υποκλάση (subclass) της κλάσης sparseMatrix η οποία με τη σειρά της επεκτείνει την κλάση Matrix, βλ. και help("Matrix-class").

`object.size(m2)`	Ο χώρος που καταλαμβάνει το m2 (ίδιος με του m1, περίπου 78.3Kb).
`s2<-Matrix(m2,sparse=T)`	Αντιγραφή του m2 σε αντικείμενο Matrix με όνομα s2.
`object.size(s2)`	Ο χώρος για το s2 είναι 118Kb καθώς ο πίνακας δεν είναι αραιός.
`s2[sample(1:10000,9500,F)]=0`	Ανάθεση της τιμής 0 σε 9500 τυχαία επιλεγμένα[263] στοιχεία του s2.
`object.size(s2)`	Ο χώρος του s2 άλλαξε σε περίπου 7Kb αφού ο πίνακας είναι αραιός.

Ένα αντικείμενο Matrix πρέπει να δημιουργείται έτσι ώστε να ανήκει σε υποκλάση της Matrix κατάλληλη για τη μορφή των δεδομένων που θα δεχτεί. Αν το αντικείμενο δημιουργείται μέσω της συνάρτησης **Matrix** τότε βασική είναι η παράμετρος sparce που καθορίζει αν ο πίνακας θα θεωρείται αραιός και άρα αν θα δημιουργήσει τύπο που εφαρμόζει συμπίεση στα δεδομένα (όπως ο dgCMatrix) ή όχι (όπως ο dgeMatrix). Ο απαιτούμενος χώρος που τελικά θα χρειαστεί για το Matrix εξαρτάται από τις επιλογές αυτές καθώς και τις τρέχουσες τιμές στα στοιχεία του πίνακα και θα αλλάζει δυναμικά. Τα αντικείμενα τύπου Matrix του παραπάνω παραδείγματος (μεταβλητές s1 και s2) είναι dgCMatrix. Ο τύπος αυτός αποθηκεύει αριθμούς διπλής ακρίβειας και δεν αποθηκεύει τα 0 κάθε στήλης. Αυτά επιβεβαιώνονται για π.χ. το s2 με την εντολή class(s2), αλλά εμφανίζονται επίσης και κατά την εκτύπωση του αντικειμένου s2:

```
> s2
1000 x 10 sparse Matrix of class "dgCMatrix"

 [1,]  .     .     .     .     .     .     .     .     .     .
 [2,]  .     .     .   3002.5  .     .     .     .     .     .
 [3,]  .     .     .   3003.5  .     .     .     .     .     .
 [4,]  .     .     .     .     .     .     .     .     .     .
 [5,] 5.5    .     .     .     .     .     .     .     .     .
 [6,]  .     .     .     .     .     .     .     .     .     .
…
```

Με τη χρήση dgCMatrix στο παραπάνω παράδειγμα, όταν τα δεδομένα έκαναν τον πίνακα να είναι όντως αραιός ο απαιτούμενος χώρος μνήμης για την αποθήκευσή τους μειώθηκε κατά τουλάχιστον 90% σε σχέση με τα αντίστοιχα matrix. Ένα αντικείμενο dgCMatrix συνήθως δημιουργείται με τη συνάρτηση **sparseMatrix** ακολουθούμενο από τρία διανύσματα που αντιστοιχούν στη γραμμή, στήλη και τιμή των μη-μηδενικών στοιχείων. Ακολουθούν μερικά παραδείγματα:

```
e1 <- sparseMatrix(i= c(1,1,3,2), j= c(2,4,4,5),
                   x = c(10,20,30,40), dims = c(6,5))
```

Το αντικείμενο που δημιουργήθηκε στο e1 είναι dgCMatrix, δηλαδή:

```
> e1
6 x 5 sparse Matrix of class "dgCMatrix"

[1,] . 10 . 20  .
[2,] .  .  .  . 40
[3,] .  .  . 30  .
[4,] .  .  .  .  .
[5,] .  .  .  .  .
[6,] .  .  .  .  .
```

Στην επόμενη εντολή οι τιμές που δόθηκαν είναι logical άρα το e2 που δημιουργείται αφορά πίνακα λογικών τιμών (lgCMatrix):

```
e2 <- sparseMatrix(i= c(1,5,6), j= c(2,4,5),
                   x = c(T,T,F), dims = c(6,5))
```

Για την περίπτωση αραιών συμμετρικών πινάκων δίνεται η τιμή TRUE στην παράμετρο symmetric της συνάρτησης sparseMatrix, ενώ η γραμμή, η στήλη και η τιμή των μη-μηδενικών στοιχείων αφορά μόνο στοιχεία που βρίσκονται στη διαγώνιο του πίνακα ή σε ψηλότερη από αυτήν θέση:

```
e3 <- sparseMatrix(i=c(1,2,2,5), j=c(2,4,2,5),
                   x=c(10,20,30,40), dims = c(5,5),
                   symmetric=TRUE)
```

Το αντικείμενο που δημιουργήθηκε στο e3 είναι dgCMatrix που περιέχει συμμετρικό πίνακα:

[263] Για τη συνάρτηση sample, βλ.§2.6 Τυχαίοι αριθμοί.

```
[1,]  . 10 .  .  .
[2,] 10 30 . 20  .
[3,]  .  . .  .  .
[4,]  . 20 .  .  .
[5,]  .  . .  . 40
```

Για διαγώνιους αραιούς πίνακες μπορεί να χρησιμοποιηθεί η συνάρτηση **bdiag**:

```
e4 <- bdiag(4,c(2,5),7,8,11)
```

Το αποτέλεσμα στο e4 είναι:

```
> e4
6 x 5 sparse Matrix of class "dgCMatrix"

[1,] 4 . . .  .
[2,] . 2 . .  .
[3,] . 5 . .  .
[4,] . . 7 .  .
[5,] . . . 8  .
[6,] . . . . 11
```

Προφανώς, αντικείμενα των παραπάνω τύπων μπορούν να δημιουργηθούν απευθείας και με τη συνάρτηση new. Για παράδειγμα, η επόμενη εντολή δημιουργεί ένα αντικείμενο κλάσης dgeMatrix (πυκνών πινάκων) με δεδομένα ίδια αυτών του e1. Οι απαιτούμενοι παράμετροι της new ορίζονται από την κλάση του αντικειμένου (βλ. help("dgeMatrix-class"):

```
e1a <-new('dgeMatrix', x = c( 0,  0,  0, 0, 0, 0,
                             10,  0,  0, 0, 0, 0,
                              0,  0,  0, 0, 0, 0,
                             20,  0, 30, 0, 0, 0,
                              0, 40,  0, 0, 0, 0),
                       Dim = c(6L,5L))
```

Η δυνατότητα αποθήκευσης στη μνήμη αραιών πινάκων μεγάλων διαστάσεων δεν έχει αξία αν δεν μπορεί να γίνει επεξεργασία με αυτούς. Οι κλάσεις του πακέτου Matrix παρέχουν πληθώρα μεθόδων (συναρτήσεων) που υποστηρίζουν τον χειρισμό τους με τις γνωστές συναρτήσεις που εφαρμόζονται στα matrix (του πακέτου 'base'). Αυτές περιλαμβάνουν αριθμητικές πράξεις και συναρτήσεις πάνω σε αντικείμενα Matrix ή μεταξύ αντικειμένων Matrix με αντικείμενα matrix, κλπ. Για παράδειγμα, παρακάτω είναι το αποτέλεσμα υπολογισμού της τετραγωνικής ρίζας των στοιχείων του e1, που επιστρέφεται επίσης ως dgCMatrix:

```
> sqrt(e1)
6 x 5 sparse Matrix of class "dgCMatrix"

[1,] . 3.162278 . 4.472136 .
[2,] . .        . .        6.324555
[3,] . .        . 5.477226 .
[4,] . .        . .        .
[5,] . .        . .        .
[6,] . .        . .        .
```

Παρακάτω, φαίνονται τα αποτελέσματα εφαρμογής σε Matrix κάποιων ενδεικτικών συναρτήσεων που έχουν ήδη αναφερθεί για τον τύπο matrix:

```
> colSums(e1)
[1]  0 10  0 50 40

> e1%*%t(e3)
6 x 5 sparse Matrix of class "dgCMatrix"

[1,] 100 700 . 200    .
[2,]   .   . .   . 1600
[3,]   . 600 .   .    .
[4,]   .   . .   .    .
[5,]   .   . .   .    .
```

```
[6,]   .   . .   .    .

> apply(e1,2,"max")
[1]  0 10  0 30 40
```

Τέλος, ακολουθεί ένα παράδειγμα πράξης του Matrix e1 με ένα matrix m3 (που δημιουργείται για τον σκοπό αυτό). Το αποτέλεσμα είναι αντικείμενο dgeMatrix (αλλά προφανώς μπορεί να μετατραπεί σε matrix με τη συνάρτηση as.matrix):

```
> m3<-matrix(1,nrow=6,ncol=5)
> m3+e1
6 x 5 Matrix of class "dgeMatrix"
     [,1] [,2] [,3] [,4] [,5]
[1,]    1   11    1   21    1
[2,]    1    1    1    1   41
[3,]    1    1    1   31    1
[4,]    1    1    1    1    1
[5,]    1    1    1    1    1
[6,]    1    1    1    1    1
```

4.3.3 Οι τύποι ts και xts (χρονοσειρές)

Οι χρονοσειρές (time series) είναι δεδομένα που αφορούν καταγραφή τιμών ως προς τον χρόνο. Για παράδειγμα, χρονοσειρές αποτελούν οι τιμές κλεισίματος μιας μετοχής ανά ημέρα, το ύψος ενός ηχητικού σήματος ανά δεκάκις χιλιοστό του δευτερολέπτου, η μέγιστη θερμοκρασία μιας περιοχής ανά ώρα. Ο τύπος δεδομένων ts περιέχεται στο προ-εγκατεστημένο πακέτο 'stats' και υποβοηθά τον χειρισμό τέτοιων δεδομένων, δηλαδή παρατηρήσεων μιας τιμής ανά τακτά, σταθερά, χρονικά διαστήματα. Πολλές συναρτήσεις επεξεργασίας χρονοσειρών απαιτούν τα δεδομένα να βρίσκονται σε αντικείμενα αυτού του τύπου. Ο τύπος ts είναι επέκταση του τύπου vector, με πρόσθετες πληροφορίες και μεθόδους προσαρμοσμένες στην επεξεργασία χρονοσειρών. Πρόσθετες ιδιότητες[264] των αντικειμένων τύπου ts περιλαμβάνουν τον φυσικό χρόνο που αφορά η αρχική και τελική καταγραφή (παρατήρηση) και τον αριθμό παρατηρήσεων ανά μονάδα φυσικού χρόνου. Παραδείγματα χρονοσειρών αποθηκευμένων σε αντικείμενα ts αποτελούν τα σύνολα δεδομένων[265] Nile και AirPassengers. Το Nile περιέχει καταγραφές της ετήσιας ροής νερού στον ποταμό Νείλο για τα έτη 1871-1970:

```
> Nile
Time Series:
Start = 1871
End = 1970
Frequency = 1
  [1] 1120 1160  963 1210 1160 1160  813 1230 1370 1140
 [11]  995  935 1110  994 1020  960 1180  799  958 1140
 [21] 1100 1210 1150 1250 1260 1220 1030 1100  774  840
 [31]  874  694  940  833  701  916  692 1020 1050  969
…
```

Βλέπουμε πως στο Nile καταγράφεται η τιμή μίας μεταβλητής (η ροή νερού), ο χρόνος πρώτης παρατήρησης (έτος 1981), ο χρόνος τελευταίας παρατήρησης (έτος 1970) και η συχνότητα παρατηρήσεων ανά μονάδα χρόνου (μια παρατήρηση ανά έτος). Ένα αντικείμενο ts δημιουργείται με την ομώνυμη συνάρτηση **ts**. Τα ίδια τα δεδομένα μπορούν να εισαχθούν από ένα διάνυσμα ή έναν πίνακα (matrix). Στην περίπτωση που τα δεδομένα είναι σε πίνακα, κάθε στήλη θα θεωρηθεί πως καταγράφει την τιμή μίας διαφορετικής μεταβλητής και κάθε γραμμή περιέχει τις τιμές των μεταβλητών αυτών που αφορούν την ίδια χρονική περίοδο παρατηρήσεων. Άρα κάθε στήλη θα θεωρηθεί μία διαφορετική χρονοσειρά για το ίδιο όμως χρονικό διάστημα και αριθμό παρατηρήσεων ανά μονάδα χρόνου. Το παράδειγμα που ακολουθεί αποθηκεύει σε ts δύο διακριτά ημιτονοειδή σήματα. Αν θεωρήσουμε πως τα δεδομένα αντιστοιχούν σε 1 δευτερόλεπτο καταγραφής, το ένα σήμα έχει συχνότητα 1Hz (1 κύκλος / δευτερόλεπτο) ενώ το δεύτερο έχει συχνότητα 4Hz (4 κύκλοι /

[264] βλ. §4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων.

[265] Τα σύνολα που αναφέρονται σε αυτή την ενότητα παρέχονται από το προ-εγκατεστημένο πακέτο 'datasets', bλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων.

δευτερόλεπτο). Αν υποθέσουμε πως τα δύο σήματα καταγράφηκαν ταυτόχρονα, μπορούν να τοποθετηθούν ως στήλες ενός matrix d (στο οποίο οι δύο στήλες ονομάστηκαν "Σήμα 1" και "Σήμα 2" αντίστοιχα):

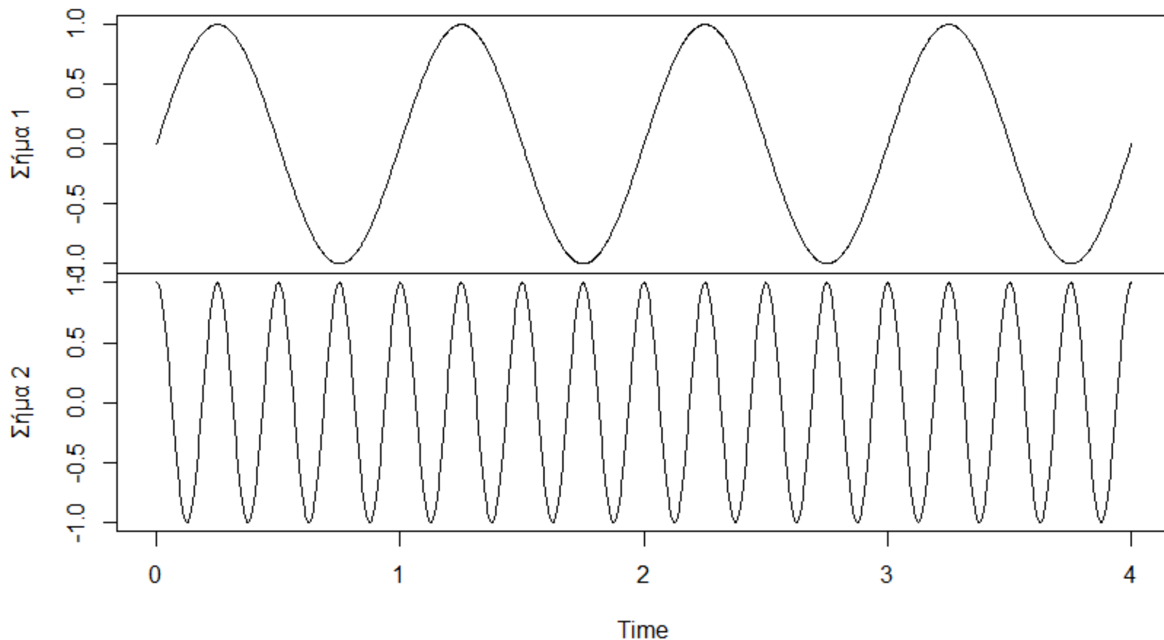
```
d<-matrix(c(sin((0:359)*1*pi/180),
            cos((0:359)*4*pi/180)),
            ncol=2)
colnames(d)= c("Σήμα 1","Σήμα 2")
```

Με βάση τα δεδομένα στο d, δημιουργείται παρακάτω αντικείμενο ts με όνομα t. Εδώ, κατασκευάζεται ώστε να περιέχει 4 περιόδους (αν μια περίοδος αντιστοιχεί σε ένα δευτερόλεπτο, θα είναι δεδομένα για 4 δευτερόλεπτα). Επίσης ορίζεται πως κάθε τιμή στα δεδομένα είναι η τιμή («δείγμα») που αντιστοιχεί στο 1/360 μίας περιόδου. Με βάση αυτή τη συχνότητα δειγματοληψίας και εφόσον υπάρχουν μόνο 360 τιμές ανά στήλη του d (άρα δεδομένα για μόνο μία περίοδο), στο αντικείμενο ts τα δεδομένα θα επεκταθούν θεωρώντας πως είναι περιοδικά, δηλαδή επαναλαμβάνονται σε κάθε περίοδο:

```
t <- ts(d, start=0, end=4, frequency=360)
```

Η Εικόνα 4.3 είναι το αποτέλεσμα της απεικόνισης σε ένα γράφημα του t (με την εντολή plot(t)).

Υπάρχουν πολλά πρόσθετα πακέτα για τον χειρισμό χρονοσειρών. Ιδιαίτερα διαδομένο είναι το πακέτο xts [61][266] μέσα στο οποίο ορίζεται ο ομώνυμος τύπος αντικειμένων. Τα αντικείμενα xts παρέχουν έναν εναλλακτικό τρόπο αποθήκευσης δεδομένων χρονοσειρών, με βελτιωμένες δυνατότητες σε σχέση με τα ts. Μεγάλος αριθμός άλλων πακέτων επεξεργασίας χρονοσειρών βασίζονται στον τύπο xts. Τέλος, πακέτα της δημοφιλούς συλλογής 'tidyverse' [57][267] χρησιμοποιούν αντικείμενα τύπου xts για αποθήκευση χρονοσειρών, αλλά χρησιμοποιούν επίσης για τον ίδιο σκοπό και τον τύπο tibble[268], καθώς το πακέτο ΄tibble΄ ανήκει στη συλλογή και αξιοποιείται από τα υπόλοιπα πακέτα της.



Εικόνα 4.3 *: Δύο χρονοσειρές σε ένα αντικείμενο τύπου ts.*

266 Το πακέτο είναι διαθέσιμο στο CRAN, βλ. §1.5 Χρήση και διαχείριση πακέτων.
267 Επίσης διαθέσιμης από το CRAN.
268 βλ.§4.3.1 Οι τύποι tibble και data.table.

4.3.4 Αντικείμενα τύπου language (expression, call, name και formula)

Μεταφράζοντας ελεύθερα δύο από τις τρείς αρχές που διέπουν την R, «οτιδήποτε υπάρχει στην R είναι αντικείμενο και οτιδήποτε συμβαίνει στην R είναι κλήση μιας συνάρτησης»[269] [62]. Έτσι, επιγραμματικά έστω, πρέπει να αναφερθούν κάποια από τα αντικείμενα που σχετίζονται με την ίδια τη γλώσσα και την εκτέλεση κώδικα. Αντικείμενα που θεωρούνται «γλώσσα» (language) είναι τα: «έκφραση» (expression), «κλήση» (call), «όνομα» (name) και «τύπος» (formula). Απλοποιώντας κάπως τα πράγματα, τα αντικείμενα αυτά περιγράφουν κώδικα και λειτουργικότητα σε γλώσσα R που όμως δεν έχει αποτιμηθεί (unevaluated). Π.χ. οι εκφράσεις είναι εντολές γραμμένες σε σύνταξη της R που «υπάρχουν» αλλά δεν έχουν απαραίτητα εκτελεστεί. Τέτοια αντικείμενα αξιοποιούνται με διάφορους τρόπους στην R. Μια επιφανειακή γεύση με τη συνάρτηση **expression** δίνουν τα παραδείγματα που ακολουθούν:

Δοκιμάστε:	**Σχόλιο**
`r1 <- expression(x <- 1:5)`	Μια έκφραση που περιγράφει ανάθεση τιμής (1:5) σε μεταβλητή x.
`r2 <- expression(sum (x))`	Μια ακόμα έκφραση R, ο κώδικας sum(x).
`r3 = expression(2 * x ^ 3)`	Μια ακόμα έκφραση R, ο κώδικας 2 * x ^ 3.
`r4 = parse(text = "y2<-10*x")`	Εξαγωγή έκφρασης από κείμενο, η έκφραση R είναι y2 <- 10 * x.

Οι μεταβλητές r1, r2, r3 και r4 περιέχουν εκφράσεις R. Αν γίνει ανάκληση ή εκτύπωσή τους, αποτυπώνει ακριβώς αυτό:

```
> r1
expression(x <- 1:5)
```

Για να εκτελεστεί η κάθε έκφραση, καλείται η συνάρτηση **eval**[270]:

Δοκιμάστε:	**Σχόλιο**
`eval(r1)`	Η εκτέλεση του r1 δημιουργεί μεταβλητή x με τιμή 1:5.
`eval(r2)`	Η εκτέλεση του r2 επιστρέφει 15.
`y1<-eval(r3)`	Η εκτέλεση του r3 επιστρέφει c(2, 16, 54, 128, 250) το οποίο εδώ ανατίθεται στο y1.
`eval(r4)`	Η εκτέλεση του r4 αναθέτει στο y2 το c(10, 20, 30, 40, 50).

Η συνάρτηση **quote** επιτρέπει τον χειρισμό μιας σειράς χαρακτήρων ως κώδικα της R. Το αντικείμενο που δημιουργεί η quote μπορεί να αποτιμηθεί αργότερα, επίσης με την eval:

Δοκιμάστε:	**Σχόλιο**
`q <- quote(2*x+sqrt(x))`	Αποθήκευση του κώδικα 2*x+sqrt(x) στο q.
`x<-100`	Ορισμός κάποιου x (εδώ με τιμή 100).
`eval(q)`	Αποτίμηση του κώδικα στο q, εδώ επιστρέφει 210.

Αντικείμενα γλώσσας σε mode «κλήση» (call) αφορούν την κλήση μιας συνάρτησης, ενώ αντικείμενα «όνομα» (name) αφορούν τον προσδιορισμό ενός αντικειμένου βάσει του ονόματός του:

Δοκιμάστε:	**Σχόλιο**
`r5 <- call("sum", x)`	Ένα αντικείμενο γλώσσας που περιγράφει κλήση συνάρτησης sum.
`r6 <- as.name("x")`	Ένα αντικείμενο γλώσσας που προσδιορίζει αντικείμενο με όνομα x.
`r7 <- call("sum", r6)`	Ένα αντικείμενο γλώσσας (κλήση) που συνδυάζει τα παραπάνω.

Τα νέα αντικείμενα γλώσσας που δημιουργήθηκαν είναι:

```
> r5
sum(1:5)
> r6
x
```

[269] Η 3η αρχή αναφέρει πως «διεπαφές (interfaces) προς άλλο λογισμικό είναι μέρος της R».

[270] Για τον μηχανισμό εκτέλεσης (αποτίμησης) εκφράσεων βλ. και §4.2.2 Ο τύπος environment (περιβάλλον). Βλ. επίσης την ενότητα «Substituting and Quoting Expressions» στην τεκμηρίωση του πακέτου ΄base΄, δηλαδή help(substitute).

```
> r7
sum(x)
```

Ενώ μπορούν επίσης να εκτελεστούν (αποτιμηθούν) μέσω της eval:

Δοκιμάστε:	**Σχόλιο**
`eval(r5)`	Η εκτέλεση του r5 επιστρέφει 15.
`eval(r6)`	Η εκτέλεση του r6 επιστρέφει το x, δηλαδή c(1, 2, 3, 4, 5).
`eval(r7)`	Η εκτέλεση του r7 επιστρέφει 15.

Σημαντικό ρόλο στην R έχουν και τα αντικείμενα τύπου formula, που επίσης είναι αντικείμενα language. Χρησιμοποιούνται κυρίως σε μεθόδους μοντελοποίησης (όπως οι συναρτήσεις lm και glm) αλλά και σε πολλές άλλες συναρτήσεις διαφόρων πακέτων της R. Έχουμε ήδη χρησιμοποιήσει αντικείμενα τύπου formula σε προηγούμενες ενότητες, χωρίς όμως να αναφερθούμε περισσότερο σε αυτά. Για παράδειγμα, χρησιμοποιήθηκαν formula στα παραδείγματα της συνάρτησης xyplot του πακέτου 'lattice' (στην §1.5.1 Χρήση πακέτων) αλλά και των συναρτήσεων aggregate και unstack (στην §4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων)). Τα αντικείμενα τύπου formula καταγράφουν την έκφραση μιας σχέσης που δεν έχει αποτιμηθεί, ένα πιθανό μοντέλο ανάμεσα σε μεταβλητές αλλά και (ως ιδιότητα του αντικειμένου) το περιβάλλον μέσα στο οποίο η σχέση αυτή δημιουργήθηκε και θα αποτιμηθεί.

Τα αντικείμενα τύπου formula δημιουργούνται με τον τελεστή '~' (tiddle). Αριστερά του τελεστή καταγράφεται η εξαρτημένη μεταβλητή, δεξιά οι ανεξάρτητες μεταβλητές χωρισμένες με τον τελεστή '+'. Έτσι στο επόμενο παράδειγμα το f είναι ένα formula που καταγράφει ότι η εξαρτημένη (dependent)[271] μεταβλητή y εξαρτάται από τις ανεξάρτητες (independent)[272] x1 και x2. Ο ορισμός ενός τέτοιου f μπορεί να γίνει με οποιονδήποτε από τους παρακάτω τρόπους:

```
f <- y ~ x1 + x2
f<-formula("y ~ x1 + x2")
f<-as.formula("y ~ x1 + x2")
```

Οποιαδήποτε από τις παραπάνω εντολές μπορεί να θεωρηθεί ότι καταγράφει (στο αντικείμενο f) πως το y εξαρτάται από τα x1 και x2, δηλαδή πως οι τιμές στο y είναι συνάρτηση των τιμών των x1 και x2. Παρατηρήστε ότι ο παραπάνω ορισμός του f γίνεται δεκτός από την R ακόμα και αν δεν έχουν οριστεί οι μεταβλητές y, x1 ή x2. Η έκφραση ορίζεται αλλά δεν αποτιμάται, άρα δεν είναι απαραίτητη και η ύπαρξη των εμπλεκόμενων μεταβλητών. Εκτός του τελεστή + (για ορισμό ανεξάρτητων μεταβλητών) στα formula διάφοροι τελεστές έχουν ειδικό νόημα, όπως οι '*', '%in%', ':' κ.α. που περιγράφονται στο help(formula).

Παραλλαγές των παραπάνω είναι function χωρίς εξαρτημένη μεταβλητή, π.χ.

```
f <- ~x1 + x2
```

καθώς και conditional εξαρτήσεις όπως η:

```
f <- y ~x1 | x2
```

όπου καταγράφεται πώς το y εξαρτάται από το x1 δεδομένης της τιμής του (πιθανότατα factor) x2. Κλείνουμε με δύο παραδείγματα που χρησιμοποιούν formula για να δημιουργήσουν δεδομένα. Αν οι τιμές για τα y, x1, x2 υπάρχουν σε κάποιο data.frame d:

```
d <- data.frame(y=c(0,1,2),x1=c(1,2,3),x2=c(1,5,10))
```

Η επόμενη εντολή καταγράφει στο αντικείμενο f ότι το sin(y) εξαρτάται από τις τιμές x1 και του log(x2).

```
f <- sin(y) ~ x1 + log(x2)
```

Άρα τα συγκεκριμένα δεδομένα για τη δημιουργία ενός μοντέλου βασισμένου στο f μπορούν να δημιουργηθούν με τη συνάρτηση **model.frame** και είναι:

```
> model.frame(f,d)
     sin(y) x1  log(x2)
1 0.0000000  1 0.000000
2 0.8414710  2 1.609438
3 0.9092974  3 2.302585
```

Ενώ αν ο τύπος είναι:

[271] Ανάλογα με την εφαρμογή, ονομάζονται επίσης απόκριση (response), αποτέλεσμα (outcome), ή ετικέτα (label).
[272] Αντίστοιχα, ανάλογα με την εφαρμογή, ονομάζονται και χαρακτηριστικά (feature), παράγοντες πρόβλεψης (predictor), ελεγχόμενες (controlled) μεταβλητές κλπ.

```
f <- y ~ I(x1 * x2)
```

η συνάρτηση **I()** («ως έχει» ή AsIs) ορίζει πως η έκφραση πρέπει να χρησιμοποιηθεί «ως έχει» και άρα το '*' δεν έχει τον ειδικό συμβολισμό που εφαρμόζεται εντός των formula. Έτσι το '*' θα σημαίνει απλά πολλαπλασιασμό και το f καταγράφει πώς το y εξαρτάται από το γινόμενο των x1 και x2, ενώ τα σχετικά δεδομένα, αν χρησιμοποιηθούν οι τιμές στο d, είναι:

```
> model.frame(f,d)
  y I(x1 * x2)
1 0          1
2 1         10
3 2         30
```

4.3.5 Ο τύπος function (συνάρτηση)

Στην R οι συναρτήσεις είναι αντικείμενα πρώτης τάξης (first-class objects), άρα συμπεριφέρονται όπως οποιοδήποτε άλλο αντικείμενο, μπορούν να αποθηκευτούν σε μεταβλητές, ως στοιχεία σε άλλες δομές, να περαστούν ως παράμετροι σε άλλες συναρτήσεις, να επιστραφούν ως αποτέλεσμα εκτέλεσης μιας συνάρτησης, κλπ., ό,τι δηλαδή ισχύει για αντικείμενα άλλων τύπων. Όλα αυτά είναι αποτέλεσμα των αρχών του συναρτησιακού προγραμματιστικού μοντέλου που ακολουθεί και προωθεί (αν και δεν επιβάλλει) η R. Οι συναρτήσεις παρέχουν πρακτικά όλες τις δυνατότητες της γλώσσας αυτής και για τον λόγο αυτό αφιερώνεται ολόκληρο το επόμενο κεφάλαιο σε θέματα που τις αφορούν (βλ.§5.1 Συναρτήσεις).

Αναφορές Κεφαλαίου 4

[33] Wickham, H. (2019). *Advanced R (2nd Editton)*. Chapman and Hall/CRC. https://adv-r.hadley.nz/

[47] Venables, W. N., Smith, D. M., & The R Core Team (2021). An Introduction to R. https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

[53] Grolemund, G., & Wickham, H. (2011). Dates and Times Made Easy with lubridate. *Journal of Statistical Software, τόμ. 40, αρ. 3*, pp. 1-25.

[54] The R Core Team (2021). R Language Definition. https://cran.r-project.org/doc/manuals/R-lang.html

[55] Wickham, H., François, R., Henry, L., Müller, K. (2021). dplyr: A Grammar of Data Manipulation. https://CRAN.R-project.org/package=dplyr και https://dplyr.tidyverse.org/

[56] Wickham, H. (2007). Reshaping Data with the reshape Package. *Journal of Statistical Software, τόμ. 21, αρ. 12*, pp. 1-20. http://www.jstatsoft.org/v21/i12/

[57] Wickham H. et al. (2019). Welcome to the tidyverse. *Journal of Open Source Software, τόμ. 4, αρ. 43*, p. 1686.

[58] Müller, K., & Wickham, H. (2021). Tibble: Simple Data Frames. https://CRAN.R-project.org/package=tibble

[59] Dowle, M., & Srinivasan, A. (2021). Data.table: Extension of `data.frame`. https://CRAN.R-project.org/package=data.table

[60] Peng, R. D., Kross, S., & Anderson, B. (2020). *Mastering Software Development in R*. Ηλεκτρονικό. https://bookdown.org/rdpeng/RProgDA/

[61] Ryan, J. A., & Ulrich, J. M. (2020). Xts: eXtensible Time Series. https://CRAN.R-project.org/package=xts

[62] Chambers, J. M. (2016). *Extending R, 1st Edition*. Chapman and Hall/CRC. ISBN 978-1-4987-7572-4.

Κεφάλαιο 5: Συναρτήσεις και συναρτησιακός προγραμματισμός

Σύνοψη
Η οργάνωση της λειτουργικότητας του κώδικά μας σε συναρτήσεις είναι ένας τρόπος με τον οποίο μπορούμε να διαχειριστούμε καλύτερα ένα έργο ανάπτυξης κώδικα, ειδικά καθώς αυτό αυξάνεται σε μέγεθος και πολυπλοκότητα. Το κεφάλαιο αυτό συγκεντρώνει διάφορα θέματα που σχετίζονται με τη δημιουργία συναρτήσεων, θέματα που σχετίζονται με τον χειρισμό των παραμέτρων τους (παράμετροι και προκαθορισμένες (default) τιμές, επιστροφές τιμών, η παράμετρος ellipsis, εμβέλεια και τοπικές μεταβλητές, χρήση μεταβλητών από άλλα περιβάλλοντα), έγερση και χειρισμός σφαλμάτων, εφαρμογή της αναδρομής (recursion), pipelining κ.α. Στην R, οι συναρτήσεις παίζουν ιδιαίτερο ρόλο καθώς η γλώσσα ακολουθεί τις αρχές του συναρτησιακού προγραμματισμού (functional programming), κάτι που επίσης παρουσιάζεται σε αυτό το κεφάλαιο.

Προαπαιτούμενη γνώση
Εξοικείωση με τα βασικά στοιχεία προγραμματισμού στην R, κεφάλαια 3 και 4.

5.1 Συναρτήσεις

Όπως αναφέρθηκε ήδη, η R χειρίζεται τις συναρτήσεις (function) ως πρώτης-τάξης (first-class) τύπο αντικειμένων (βλ.§4.3.5 Ο τύπος function (συνάρτηση)). Έτσι μια συνάρτηση μπορεί να αποθηκευτεί σε μεταβλητές και σε τύπους αντικειμένων που την υποστηρίζουν (π.χ. list), να αντιγραφεί από αυτά, να περαστεί ως παράμετρος σε άλλες συναρτήσεις, να επιστραφεί από άλλες συναρτήσεις κλπ. Στην R, οτιδήποτε εκτελείται είναι αποτέλεσμα κλήσης κάποιας συνάρτησης, άρα οι συναρτήσεις είναι βασικότατο στοιχείο της γλώσσας, όπως αναφέρεται και στο [62].

5.1.1 Δημιουργία συναρτήσεων

Μία νέα συνάρτηση ορίζεται με τη συνάρτηση **function**. Εκεί γίνεται ορισμός των παραμέτρων της συνάρτησης και του «σώματός» της. Η μορφή σύνταξης της εντολής είναι:

function (λίστα παραμέτρων) σώμα
ή
\(λίστα παραμέτρων) σώμα

Στο σώμα της συνάρτησης καταγράφονται οι εντολές που θα εκτελεί, οι εσωτερικές «τοπικές» μεταβλητές της καθώς και η τιμή που θα επιστρέφει μετά την εκτέλεση της. Ας ξεκινήσουμε με ένα ανορθόδοξο παράδειγμα:

```
function(x) 2*x+5
```

Στο παράδειγμα αυτό δημιουργείται ένα ανώνυμο αντικείμενο τύπου function. Σύμφωνα με τον παραπάνω ορισμό, έχει σχεδιαστεί να δέχεται μια παράμετρο x και έχει ως σώμα το 2*x+5, άρα επιστρέφει το όποιο x διπλασιασμένο συν 5. Όμως, όπως κάθε αντικείμενο στην R, εφόσον δεν αποθηκεύτηκε κάπου (π.χ. σε μεταβλητή) δεν θα διατηρηθεί. Το αντικείμενο θα διαγραφεί μετά την εκτέλεση της παραπάνω εντολής δημιουργίας του. Επίσης, η συνάρτηση (ή ακριβέστερα το σώμα της) δεν θα εκτελεστεί ποτέ. Πάντως, στη σύντομη ζωή της, ακόμα και μια ανώνυμη συνάρτηση μπορεί να χρησιμοποιηθεί:

```
> {function(x) 2*x+5}(1:10)
 [1]  7  9 11 13 15 17 19 21 23 25
```

Η παραπάνω εντολή όχι μόνο δημιουργεί το αντικείμενο συνάρτησης (μέσα στο μπλοκ κώδικα[273]) αλλά και ακολούθως το καλεί. Γράφοντας αμέσως μετά από ένα αντικείμενο συνάρτησης και μέσα σε παρένθεση τις τιμές που θέλουμε να πάρουν οι παράμετροί της, ο κώδικας της συνάρτησης εκτελείται για τις τιμές αυτές. Στο παραπάνω παράδειγμα δίνεται ως τιμή της παραμέτρου x το διάνυσμα αριθμών 1:10. Έτσι η συνάρτηση δημιουργείται και εκτελείται για x = 1:10, επιστρέφοντας το αποτέλεσμα (2* 1:10 + 5). Γενικά πάντως οι συναρτήσεις που δημιουργούνται, αποθηκεύονται κάπου, συνήθως με ανάθεσή τους σε μεταβλητές. Στο επόμενο παράδειγμα, το αντικείμενο function που δημιουργείται ανατίθεται στη μεταβλητή f:

[273] Υπενθυμίζουμε πως το μπλοκ κώδικα είναι ένα κομμάτι αναπόσπαστων εντολών κώδικα και οριοθετείται με { και }, βλ.§3.2.1 Μπλοκ κώδικα.

```
f <- function(x) 2*x+5
```

Εναλλακτικά το παραπάνω μπορεί να γραφεί ως:

```
f <- \(x) 2*x+5
```

Οποιαδήποτε σύνταξη και αν χρησιμοποιηθεί, αναθέτει τη συνάρτηση (αντικείμενο τύπου function) σε μια μεταβλητή με όνομα f. Αυτό επιτρέπει να καλέσουμε τη συνάρτηση μέσω του ονόματος f και όσο υπάρχει αυτό το όνομα το αντικείμενο δεν διαγράφεται (όπως ακριβώς συμβαίνει και με οποιοδήποτε άλλον τύπο αντικειμένου). Αντίθετα το όνομα f αναφέρεται στο αντικείμενο τύπου function που ορίστηκε παραπάνω και αυτό το αντικείμενο επιστρέφει αν ανακληθεί[274]:

```
> f
function(x)  2 * x + 5
```

Αν τώρα κληθεί να εκτελεστεί το f δίνοντας αμέσως μετά το όνομα της συνάρτησης (και μέσα σε παρένθεση) τα ορίσματά της π.χ. για x = 30 θα επιστρέψει το αποτέλεσμα:

```
> f(30)
[1] 65
```

ενώ αντίστοιχα για x = 1:10 θα επιστρέψει το αποτέλεσμα των αντίστοιχων πράξεων με το διάνυσμα:

```
f(1:10)
 [1]  7  9 11 13 15 17 19 21 23 25
```

Όπως κάθε τύπος αντικειμένων, μια μεταβλητή που περιέχει συνάρτηση μπορεί να αντιγραφεί σε άλλη, π.χ. η εντολή y <- f αντιγράφει το f σε μεταβλητή y, οπότε μια κλήση της y θα έχει ταυτόσημο αποτέλεσμα με την αντίστοιχη κλήση της f:

```
> y(1:10)
 [1]  7  9 11 13 15 17 19 21 23 25
```

Παρατηρήστε ότι στις συναρτήσεις της R δεν ορίζεται ο τύπος του αντικειμένου που θα επιστραφεί. Επίσης δεν ορίζεται ο τύπος των παραμέτρων (εδώ δηλαδή ο τύπος του x). Έτσι η τιμή που θα περαστεί στο x μπορεί να είναι αντικείμενο οποιουδήποτε τύπου.

Οι εκφράσεις που δίνονται ως ορίσματα της συνάρτησης αποτιμώνται και το αποτέλεσμα περνά ως τιμή της παραμέτρου και εκτελείται ο κώδικας στο σώμα της συνάρτησης. Έτσι η εντολή:

```
> f(1:(5+5))
 [1]  7  9 11 13 15 17 19 21 23 25
```

είναι ίδια με την εντολή f(1:10) αφού το 1:(5+5) εκτελείται (αποτιμάται) πριν την κλήση με αποτέλεσμα το διάνυσμα 1:10, το οποίο και περνά ως τιμή της παραμέτρου x της f.

Το μέρος του ορισμού της συνάρτησης με τις παραμέτρους επιστρέφει η συνάρτηση args, ενώ στο σώμα μιας συνάρτησης μπορεί να γίνει πρόσβαση με τη συνάρτηση **body**:

```
> body(f)
2 * x + 5
> body(f) <- quote(2*x+10)
```

Η τελευταία εντολή αντικατέστησε το σώμα της f με νέο κώδικα[275]. Έτσι:

```
> f(30)
[1] 70
```

Οι παράμετροι στις συναρτήσεις της R δεν είναι συγκεκριμένου τύπου, οπότε ενδεχόμενες ασυμβατότητες των αντικειμένων που περνούν ως ορίσματα με τον κώδικα της συνάρτησης θα διαπιστωθούν κατά την κλήση (εκτέλεση) των εντολών του σώματος της συνάρτησης. Π.χ. αν προσπαθήσουμε να καλέσουμε την f με τιμή της παραμέτρου x ένα αντικείμενο character (κείμενο), αυτό θα προκαλέσει σφάλμα σταματώντας την εκτέλεση. Αυτό θα συμβεί μόνο όταν η R προσπαθήσει να υπολογίσει το 2*x, και η συνάρτηση του πολλαπλασιασμού εγείρει το σφάλμα πως αυτό είναι αδύνατον να πραγματοποιηθεί για κείμενο:

[274] Εδώ εμφανίζεται ο ορισμός της συνάρτησης που συμπεριλαμβάνει και τον κώδικα στο σώμα της συνάρτησης. Το ίδιο θα συμβεί για πολλές από τις έτοιμες συναρτήσεις που περιέχονται στα διάφορα πακέτα της R (ενσωματωμένα ή μη). Όμως συχνά οι συναρτήσεις αυτές είναι απλώς «περιτύλιγμα» (wrapper functions, βλ. και υποσημείωση 391) και το σώμα τους απλώς καλεί άλλον κώδικα μέσω συναρτήσεων όπως οι .Primitive, .Internal, η UseMethod (αν πρέπει να επιλεγεί μια κατάλληλη για την κλάση των παραμέτρων μέθοδος (βλ.§6.2 Αντικειμενοστραφής προγραμματισμός στην R) κλπ. Οπότε ο κώδικας που καλείται και εκτελείται εσωτερικά μπορεί να είναι γραμμένος σε άλλες γλώσσες προγραμματισμού (C, C++ FORTRAN, βλ.§7.1 Πολυγλωσσικές λύσεις) και να έρχεται ήδη μεταφρασμένος (compiled) από αυτές σε μορφή εκτελέσιμου κώδικα. Σε αυτές τις περιπτώσεις η συνάρτηση απλώς καλεί τον κώδικα αυτό. Αυτό γίνεται με συναρτήσεις όπως οι .C, .Call, και σε πολλές περιπτώσεις οι προαναφερθείσες .Internal και .Primitive.

[275] Για τη συνάρτηση quote βλ.§4.3.4 Αντικείμενα τύπου language (expression, call, name και formula).

```
> f("Maria")
Error in 2 * x : non-numeric argument to binary operator
```

Σφάλμα θα προκληθεί επίσης αν μια συνάρτηση κληθεί με υπερβολικά λίγα ή υπερβολικά πολλά ορίσματα παραμέτρων. Εδώ η f απαιτεί ένα όρισμα (το x) άρα δεν θα εκτελεστεί αν κληθεί με κανένα (π.χ. f()) ή άνω του ενός (π.χ. f(30,40)) ορίσματα:

```
> f()
Error in f() : argument "x" is missing, with no default

> f(30,40)
Error in f(30, 40) : unused argument (40)
```

Πάντως υπό κανονικές συνθήκες η συνάρτηση που καλείται εκτελεί τον κώδικα του σώματός της και, εφόσον δεν υπάρξει λάθος, ολοκληρώνει επιστρέφοντας (πάντα) ένα μοναδικό αποτέλεσμα. Το αποτέλεσμα αυτό μπορεί επίσης να είναι αντικείμενο οποιουδήποτε τύπου και εξ ορισμού είναι το αποτέλεσμα της τελευταίας εντολής που εκτελέστηκε κατά την κλήση της συνάρτησης. Έτσι για το f(30) το τελευταίο βήμα που εκτελέστηκε στο σώμα της συνάρτησης ήταν το 60+5 και η συνάρτηση επιστρέφει το 65. Ακολουθούν μερικά ακόμα παραδείγματα:

```
epi2 <- function(x) x*2
```

Στη μεταβλητή epi2 αποθηκεύεται συνάρτηση η οποία δέχεται μια παράμετρο x. Το σώμα της συνάρτησης ορίζει πώς θα επιστρέφεται το x*2. Η παραπάνω συνάρτηση θα μπορούσε να γραφεί περιγραφικότερα, χωρίς να αλλάξει κάτι στη λειτουργία της:

```
epi2<-function(x) { x*2 }
```

Η διαφορά στον 2ο ορισμό της epi2 είναι πως το σώμα της συνάρτησης έχει οριοθετηθεί με ένα μπλοκ κώδικα (βλ.§3.2.1 Μπλοκ κώδικα). Αυτό είναι απαραίτητο μόνο όταν το σώμα περιέχει παραπάνω από μια εντολές. Δεν χρειάζεται να οριστεί μπλοκ κώδικα αν το σώμα περιέχει μόνο ένα βήμα ή εντολή όπως στην epi2 παραπάνω. Εξάλλου, αυτός είναι ο γενικότερος κανόνας για τα μπλοκ κώδικα, ότι δηλαδή, απαιτούνται μόνο για να οριστεί ένα κομμάτι κώδικα που περιέχει πολλές αναπόσπαστες εντολές. Μερικοί άλλοι ισοδύναμοι τρόποι να οριστεί η epi2 είναι:

```
epi2<-function(x) return(x*2)
```

και

```
epi2<-function(x) { return(x*2) }
```

Η λέξη **return** είναι μέρος του ορισμού των function (βλ. help(function)). Με το return προσδιορίζεται τι ακριβώς πρέπει να επιστραφεί από τη συνάρτηση. Όπως προαναφέρθηκε, αν δεν υπάρχει return, μια συνάρτηση επιστρέφει το αποτέλεσμα της τελευταίας εντολής που θα εκτελέσει κατά την κλήση της. Αυτό συχνά είναι θεμιτό και η return παραλείπεται (όπως έγινε στους αρχικούς ορισμούς της epi2). Πάντως, αν υπάρχουν return στις εντολές του σώματος μιας συνάρτησης, όταν η εκτέλεση του κώδικα του σώματος της συνάρτησης φτάσει σε return η κλήση τερματίζεται (αν εντολές υπάρχουν μετά το return στο σώμα της συνάρτησης δεν εκτελούνται) και ολοκληρώνεται επιστρέφοντας το αποτέλεσμα που έχει προσδιοριστεί εντός των παραμέτρων της return. Ακολουθούν μερικά παραδείγματα χρήσης της epi2:

Δοκιμάστε:	**Σχόλιο**
`epi2(30)`	Καλεί την epi2 για x=30, επιστρέφει 60.
`epi2(1:10)`	Καλεί την epi2 για x=1:10, επιστρέφει c(2, 4, 6, 8, 10, 12, 14, 16, 18, 20).
`epi2(seq(10,100,5))`	Καλεί την epi2 για x=c(10, 15, 20, …, 100), επιστρέφει c(20, 30, 40,…, 200).
`y<-epi2(15)`	Καλεί την epi2 για x=15, επιστρέφει 30 που αποθηκεύεται σε μεταβλητή y.
`epi2(y)`	Καλεί την epi2 για x=y (του προηγ. βήματος), επιστρέφει 60.
`epi2()`	Δεν εκτελείται καθώς δεν έχει οριστεί το x, εγείρει μήνυμα λάθους.
`epi2`	Ανάκληση του αντικειμένου epi2, επιστρέφει την ίδια τη συνάρτηση.
`args(epi2)`	Εμφανίζει τα ονόματα των παραμέτρων της epi2 και τις προεπιλεγμένες τιμές τους.[276]

Θα έχει γίνει ήδη απολύτως σαφές ότι μια συνάρτηση καλείται ακολουθούμενη από τα ορίσματα των παραμέτρων της μέσα σε παρένθεση. Όμως οι τελεστές είναι και αυτοί συναρτήσεις και συνήθως δεν καλούνται με τον τρόπο αυτό. Π.χ. δυαδικοί τελεστές όπως το '+' δεν καλούνται συνήθως ως "+"(2,4) (αν και επιτρέπεται)

[276] Η args επιστρέφει μια ανώνυμη συνάρτηση ίδια με τη δοθείσα (εδώ π.χ. μια συνάρτηση ίδια με την epi2) με το σώμα της νέας συνάρτησης να είναι το NULL.

άλλα ως 2+4, δηλαδή τοποθετημένοι ανάμεσα στα ορίσματα (infix). Για να οριστεί μια συνάρτηση που θα γράφεται με αυτόν τον τρόπο στην R, πρέπει να ανατεθεί σε όνομα μεταβλητής που ξεκινά και τελειώνει με %, για παράδειγμα:

```
"%epi%" <- function(x,y) x*y
```

Το παραπάνω όνομα επιτρέπει στη συνάρτηση να κληθεί τόσο με τον συνήθη τρόπο (π.χ. "%epi%" (2,3)) όσο και γραμμένη infix:

```
> 2 %epi% 3
[1] 6
```

5.1.2 Παράμετροι

Μια συνάρτηση μπορεί να οριστεί ώστε να μη χρειάζεται καμία παράμετρο, όπως π.χ.

```
ten<-function() 10
```

η οποία, αν κληθεί (με την εντολή ten()) επιστρέφει 10. Όμως συνήθως οι συναρτήσεις απαιτούν μία ή περισσότερες παραμέτρους. Όταν καλείται μια συνάρτηση, οι τιμές που ορίζονται για τις παραμέτρους ανατίθενται σε αυτές με τη σειρά που έχουν στον ορισμό της συνάρτησης. Επίσης, τιμές παραμέτρων μπορούν να οριστούν για συγκεκριμένες παραμέτρους με το όνομά τους, παρακάμπτοντας τη σειρά που έχουν οριστεί, ή να γίνει συνδυασμός των παραπάνω. Αυτά τα έχουμε ήδη δει καλώντας διάφορες έτοιμες συναρτήσεις πακέτων:

Δοκιμάστε:	**Σχόλιο**
`f<-function(x,y,z) c(x,y,z)`	Η f δέχεται τρεις παραμέτρους και τις συγχωνεύει σε διάνυσμα.
`f(3,2,1)`	Αναθέτει x=3, y=2, z=1 με τη σειρά ορισμού τους.
`f(x=3,y=2,z=1)`	Ίδιο με το παραπάνω.
`f(y=2,3,1)`	Καλεί για y=2, ενώ τις υπόλοιπες παράμετροι με τη σειρά ορισμού τους.
`f(z=1,3,y=2)`	Καλεί για y=2, z=1, ενώ το εναπομένων x παίρνει την τιμή 3.

Αν και γραμμένες με ελαφρώς διαφορετικό τρόπο, όλες οι παραπάνω κλήσεις της f θα περάσουν στις παραμέτρους τις τιμές x=3, y=2 και z=1 και άρα θα επιστρέψουν το διάνυσμα c(3, 2, 1). Για τον ορισμό τιμών με χρήση του ονόματός των παραμέτρων πρέπει να χρησιμοποιηθεί ο τελεστής = (αν λανθασμένα χρησιμοποιηθεί <- ή -> θα γίνει ανάθεση τιμής σε μεταβλητή που είναι συνώνυμη με την παράμετρο και βρίσκεται εκτός της συνάρτησης). Επιπρόσθετα, σε περίπτωση μεγάλων ονομάτων, μπορεί να χρησιμοποιηθούν τα αρχικά τους γράμματα (εφαρμόζεται partial matching για εντοπισμό μέσω του ονόματος[277]).

Κατά τη δημιουργία μιας συνάρτησης στη λίστα παραμέτρων της μαζί με το όνομα κάθε παραμέτρου μπορεί να οριστεί και η προεπιλεγμένη τιμή που θα παίρνει αν δεν της δοθεί τιμή κατά την κλήση. Αυτό πρέπει επίσης να γίνει με τον τελεστή = (δεν επιτρέπεται να γίνει με <- ή ->). Έτσι στην παρακάτω παραλλαγή του ορισμού της f προσδιορίζεται η τιμή 2 ως προεπιλεγμένη τιμή της παραμέτρου y, αν δεν οριστεί συγκεκριμένη τιμή για την παράμετρο y κατά την κλήση της f. Αντίστοιχα η προεπιλεγμένη τιμή για το z έχει οριστεί σε 1:

```
f<-function(x,y=2,z=1) c(x,y,z)
```

Δοκιμάστε:	**Σχόλιο**
`f(3,4,5)`	Όλες οι παράμετροι ορίζονται και χρησιμοποιούνται, επιστρέφει c(3,4,5).
`f(3,4)`	Η ανάθεση τιμών είναι x=3, y=4 και το προεπιλεγμένο z=1, επιστρέφει c(3,4,1).
`f(3,z=5)`	Καλεί για x=3, προεπιλεγμένο y=2 και z=5, επιστρέφει c(3,2,5).
`f(x=3)`	Καλεί για x=3, προεπιλεγμένο y=2 και προεπιλεγμένο z=1, επιστρέφει c(3,2,1).
`f(z=3)`	Λάθος, το x δεν ορίστηκε και δεν έχει προεπιλεγμένη τιμή.

Οι προεπιλεγμένες τιμές παραμέτρων μιας συνάρτησης μπορούν να ελεγχθούν, ακόμα και να αλλαχθούν δυναμικά μετά τη δημιουργία της με συναρτήσεις όπως η **formals**. Η συνάρτηση αυτή δίνει πρόσβαση στη λίστα[278] παραμέτρων και προεπιλεγμένων τιμών μιας συνάρτησης, π.χ. η εντολή:

```
formals(f)$y=20
```

[277] βλ.§4.2.1.1 Ο τελεστής επιλογής $.

[278] Είναι αντικείμενο τύπου pairlist βλ.§4.2.1 Ο τύπος list (λίστα).

αλλάζει για τη συνάρτηση f την προεπιλεγμένη τιμή της παραμέτρου y σε 20 (από 2 που είχε οριστεί παραπάνω). Έτσι αν κληθεί τώρα η f χωρίς να οριστούν y και z, το y=20 και το z=1:

```
> f(x=3)
[1]  3 20  1
```

Κατά τον ορισμό μίας συνάρτησης, μια τελεία (.) ή δυο τελείες (..) μπορούν να χρησιμοποιηθούν ως όνομα παραμέτρου. Δεν διαφέρουν από άλλα συμβατικά ονόματα παραμέτρων, αλλά καμιά φορά το όνομα παραμέτρου (.) χρησιμοποιείται για να συμβολίσει πως η θέση κρατιέται για οτιδήποτε, (είναι placeholder). Αυτό συμβαίνει απλώς γιατί το (.) έχει λειτουργία placeholder σε κάποιες άλλες γλώσσες προγραμματισμού. Όμως στην R αυτό δεν συμβαίνει. Για παράδειγμα ο ορισμός μιας συνάρτησης epi που πολλαπλασιάζει δυο παραμέτρους της ως:

```
epi<-function(x , .)  x * .
```

δεν διαφέρει από τον ισοδύναμο ορισμό:

```
epi<-function(x , y)  x * y
```

Όμως οι τρείς διαδοχικές τελείες (…)[279] στον ορισμό παραμέτρων έχουν ειδικό νόημα placeholder και σημαίνουν «και λοιπά» ή πιο συγκεκριμένα «λοιπές παράμετροι που ίσως δοθούν» (βλ. help(dots)). Αυτό επιτρέπει σε μια συνάρτηση να καλείται με παραμέτρους που δεν έχουν προσδιοριστεί κατά τον ορισμό της. Κατά την κλήση μιας συνάρτησης που δέχεται παραμέτρους "…", όποιες τιμές παραμέτρων δοθούν για συγκεκριμένες και ορισμένες με όνομα παραμέτρους χρησιμοποιούνται κανονικά. Οτιδήποτε άλλο όμως περαστεί ως παράμετρος δεν προκαλεί λάθος αλλά θεωρείται "λοιπές παράμετροι" που η συνάρτηση δεν αξιοποιεί άμεσα. Έτσι, η συνάρτηση μπορεί να περάσει αυτές τις λοιπές παραμέτρους σε άλλες συναρτήσεις που μπορούν να τις αξιοποιήσουν. Μερικά παραδείγματα:

```
f<-function(x, y, ...) x + y - sum(...)
```

Στον παραπάνω ορισμό μιας συνάρτησης (που ονομάστηκε f) υπάρχουν δύο συγκεκριμένες παράμετροι x και y. Οτιδήποτε παραπάνω δοθεί ως όρισμα κατά την κλήση της συνάρτησης θα περαστεί στη sum. Η συνάρτηση επιστρέφει το άθροισμα των x + y μείον το άθροισμα των πρόσθετων παραμέτρων που μπορεί να υπάρχουν. Άρα το παρακάτω είναι 100 + 200 – sum(0):

```
> f(100,200)
[1] 300
```

Ενώ το επόμενο επιστρέφει 100 + 200 – sum(10, 20, 30), δηλαδή 300 – 60:

```
> f(100,200,10,20,30)
[1] 240
```

Στο σώμα της συνάρτησης μπορεί να γίνει επιλογή συγκεκριμένων ορισμάτων από αυτά που δόθηκαν ως λοιπές παράμετροι. Το "…1" προσδιορίζει το 1ο από αυτά, το "…2" το 2ο και ούτω καθεξής, ενώ το "…elt(ν)" προσδιορίζει το ν-οστό τέτοιο όρισμα μέσω της συνάρτησης **…elt**. Έτσι οι δύο παρακάτω ορισμοί μιας συνάρτησης επιστρέφουν το άθροισμα των παραμέτρων x + y με την 1η και την 3η από τις πρόσθετες παραμέτρους που μπορεί να υπάρχουν:

```
f<-function(x, y, ...) x + y + ..1 + ..3
f<-function(x, y, ...) x + y + ...elt(1) + ...elt(3)
```

Όποιος από τους παραπάνω ορισμούς κληθεί θα πρέπει να έχει τουλάχιστον 5 ορίσματα. Άρα π.χ. αν κληθεί η f(1,2,3,4,5,7), θα επιστρέψει 1+2+3+5 καθώς x=1, y=2 ενώ η 1η και η 3η από τις υπόλοιπες παραμέτρους είναι το 3 και το 5 αντίστοιχα:

```
> f(1,2,3,4,5,7)
[1] 11
```

Ο αριθμός των πρόσθετων παραμέτρων που δόθηκαν κατά την κλήση επιστρέφεται από τη συνάρτηση **…length**, κάτι που επιτρέπει την επεξεργασία άγνωστου αριθμού παραμέτρων:

```
f <- function(...) ...elt(...length()-1) + ...elt(...length())
```

Η παραπάνω συνάρτηση δέχεται οποιοδήποτε αριθμό παραμέτρων[280] και επιστρέφει το άθροισμα των δύο τελευταίων:

```
> f(1,2,10,20)
[1] 30
```

Ένας συνήθης λόγος για τη χρήση μη προσδιορισμένων παραμέτρων "…" είναι σε περιπτώσεις που η συνάρτηση καλεί άλλες συναρτήσεις οι οποίες θα προσδιοριστούν δυναμικά κατά την εκτέλεση του κώδικα

[279] Αναφέρονται και ως παράμετρος ellipsis, three dots parameter, ή dynamic dots, βλ. help("dyn-dots").
[280] Προφανώς, χρειάζεται τουλάχιστον δύο ορίσματα αφού προσθέτει τα δύο τελευταία.

(π.χ. θα περαστούν στη συνάρτηση ως τιμή σε παραμέτρους της). Εδώ ζητούμενο είναι να επιτρέπεται ο ορισμός των τιμών και των δικών τους παραμέτρων. Στο επόμενο παράδειγμα η συνάρτηση f έχει δύο προσδιορισμένα ορίσματα με ονόματα x και FUN[281], αλλά δέχεται και πρόσθετα ορίσματα μέσω του "…". Σκοπός της συνάρτησης στο παράδειγμα είναι να καλέσει την όποια FUN για το διάνυσμα 5:x. Με το "…" επιτρέπεται στον χρήστη της f να περάσει πρόσθετα ορίσματα τα οποία η f με τη σειρά της θα χρησιμοποιήσει ως ορίσματα στην κλήση της FUN:

```
f<-function(x, FUN, ...) FUN(5:x, ...)
```

Έτσι αν κληθεί η f για FUN=sort μπορούν να προστεθούν παράμετροι που δέχεται η sort (όπως η decreasing). Η παρακάτω κλήση της f προκαλεί την κλήση της sort(5:10, decreasing=T):

```
> f(10, sort, decreasing=T)
[1] 10  9  8  7  6  5
```

Ενώ αντίστοιχα αν κληθεί η f για FUN=log μπορούν να προστεθούν παράμετροι που δέχεται η συνάρτηση αυτή (π.χ. η παράμετρος base), όπως παρακάτω, όπου η f προκαλεί την κλήση της log(5:10, base=10):

```
> f(10, log, base=10)
[1] 0.6989700 0.7781513 0.8450980 0.9030900 0.9542425 1.0000000
```

Δύο συναρτήσεις που σχετίζονται με τον ορισμό τιμών παραμέτρων και την ακόλουθη κλήση άλλων συναρτήσεων είναι οι **call** και **do.call**. Η call δημιουργεί ένα αντικείμενο (τύπου call) που καταγράφει τα απαραίτητα για να γίνει κλήση κάποιας συνάρτησης με συγκεκριμένες τιμές παραμέτρων. Η συνάρτηση που θα κληθεί, ορίζεται βάσει του ονόματός της. Για το παρακάτω παράδειγμα ορίστηκε μια απλή συνάρτηση f η οποία δέχεται μεταβλητό αριθμό παραμέτρων και επιστρέφει 1000 συν το άθροισμα των στοιχείων των παραμέτρων αυτών (υπολογισμένο μέσω sum). Ακολουθεί μια ενδεικτική κλήση της f μέσω της συνάρτησης call για παραμέτρους x, 2 και 3:

```
f<-function(...) sum(1000,...)
x<-1:9
a<-call("f",x,2,3)
```

Το a είναι πλέον μια μεταβλητή που περιέχει αντικείμενο τύπου call που αντιστοιχεί στην κλήση. Η ίδια η κλήση μπορεί να γίνει με τη συνάρτηση eval:

```
> a
f(1:9, 2, 3)
> eval(a)
[1] 1050
```

Παρατηρήστε ότι στο a καταγράφεται η κλήση με τις τιμές των παραμέτρων όπως αυτές αποτιμήθηκαν όταν δημιουργήθηκε το a. Έτσι ακόμα και αν αργότερα αλλάξει το περιβάλλον (π.χ. εδώ, να αλλάξει η τιμή του x), η εντολή eval(a) πάντα οδηγεί σε χρήση των ίδιων τιμών στα ορίσματα (εδώ, 45, 2 και 3). Μια ισοδύναμη κλήση της f μέσω της συνάρτησης do.call θα ήταν:

```
> do.call(f,list(x,2,3))
[1] 1050
```

Αν και στα παραπάνω παραδείγματά η κλήση της f με χρήση των call και do.call οδήγησαν στο ίδιο αποτέλεσμα[282] με μια απευθείας κλήση της συνάρτησης (f(x,2,3)), η μεθοδολογία που χρησιμοποιήθηκε έχει κάποιες διαφορές. Η call επιτρέπει να αποθηκευτεί η κλήση και να χρησιμοποιηθεί στο μέλλον. Η do.call επιτρέπει να περάσουν τα ορίσματα χωρίς να αποτιμηθούν (παράμετρος quote), καθώς και να οριστεί το περιβάλλον που θα γίνει η αποτίμηση της κλήσης (παράμετρος envir)[283]. Όμως βασικότερο είναι ότι στη do.call τα ορίσματα της συνάρτησης αποθηκεύονται σε αντικείμενο list[284] και γίνεται χειρισμός τους (προσθήκη ή διαγραφή ορισμάτων, ορισμός ονομάτων κλπ.) μέσω αυτού. Έτσι αν π.χ. η εφαρμογή απαιτεί κλήση κάποιας συνάρτησης που δέχεται μεταβλητό αριθμό ορισμάτων και αυτός θα καθορίζεται κατά τον χρόνο εκτέλεσης, τα ορίσματα μπορούν να αποθηκευτούν όπως:

```
l<-list()
```

---

[281] Το όνομα της 2ης παραμέτρου επιλέχτηκε να είναι FUN ως υπενθύμιση πως εδώ θα περαστεί κάποια συνάρτηση, αλλά και επειδή αυτό το όνομα παραμέτρου (FUN) χρησιμοποιούν για τον ίδιο σκοπό (προσδιορισμό μιας συνάρτησης) πολλές από τις συναρτήσεις ενσωματωμένων πακέτων της R, όπως π.χ. οι συναρτήσεις apply βλ.§4.1.3.4 Η οικογένεια συναρτήσεων apply.

[282] Αυτό συμβαίνει συνήθως. Για την do.call υπάρχουν περιπτώσεις που δεν ισχύει απόλυτα, βλ. help(do.call).

[283] βλ.§4.2.2.1 Περιβάλλοντα και συναρτήσεις.

[284] βλ. §4.2.1 Ο τύπος list (λίστα).

```
l[[1]]<-x
l[[2]]<-2
l[[3]]<-3
```

και αργότερα να κληθεί η συνάρτηση με τα ορίσματα στη λίστα:

```
> do.call(f,l)
[1] 1050
```

Αυτή η δυνατότητα χειρισμού (κατά τον χρόνο εκτέλεσης) του list που περιέχει τις παραμέτρους κλήσης κάποιας συνάρτησης (και όχι μόνο των τιμών τους) κάνει την do.list ιδιαίτερα χρήσιμη σε διάφορες εφαρμογές, ιδιαίτερα όταν η συνάρτηση που θα κληθεί υποστηρίζει μεταβλητό αριθμό παραμέτρων.

5.1.3 Έλεγχος παραμέτρων

Σε αντίθεση με πολλές άλλες συνήθεις γλώσσες προγραμματισμού, στην R κατά τη δημιουργία μιας συνάρτησης δεν ορίζονται οι τύποι που πρέπει να έχουν οι παράμετροί της. Μπορούν μόνο να οριστούν προεπιλεγμένες τιμές που θα χρησιμοποιούνται αν δεν δοθούν συγκεκριμένες τιμές για αυτές κατά την κλήση της συνάρτησης. Έτσι, η συνάρτηση μπορεί να κληθεί με τα ορίσματα να είναι αντικείμενα οποιουδήποτε είδους. Η R δεν είναι compiled γλώσσα ώστε να ελέγχει στη φάση της μεταγλώττισης τη συμβατότητα των ορισμάτων που περνούν στη συνάρτηση. Αν υπάρξουν ασυμβατότητες των ορισμάτων με τον κώδικα της συνάρτησης (ή έστω με τον σκοπό του δημιουργού της συνάρτησης για αυτά) δεν μπορούν να ελεγχθούν εκ των προτέρων, αλλά διαπιστώνονται την ώρα που η συνάρτηση καλείται και εκτελείται ο κώδικας στο σώμα της συνάρτησης. Αν τα ορίσματα είναι εκτός προδιαγραφών, ασύμβατα με τις εντολές που θα τα χρησιμοποιήσουν ή εκτός των στόχων του δημιουργού της συνάρτησης, είτε κάποια εντολή στο σώμα θα αποτύχει εγείροντας λάθος εκτέλεσης (runtime error) είτε η εκτέλεση της συνάρτησης θα ολοκληρωθεί χωρίς πρόβλημα, αλλά το αποτέλεσμα που θα επιστρέψει θα είναι μη αξιοποιήσιμο. Όμως, η R και οι συναρτήσεις της συχνά χρησιμοποιούνται διαδραστικά με τον χρήστη. Σε έναν τέτοιο τρόπο λειτουργίας, η αποτυχία μιας συνάρτησης είναι κάτι σχετικά αποδεκτό: η εκτέλεση της συνάρτησης σταματά, εμφανίζεται ένα μήνυμα λάθους και ο/η χρήστης μπορεί να ξαναδοκιμάσει να καλέσει τη συνάρτηση με άλλα ορίσματα στις παραμέτρους της. Το περιβάλλον συνήθως δεν αλλοιώνεται (για λόγους που θα δούμε παρακάτω, βλ.§5.1.5 Αλληλεπίδραση με το περιβάλλον), άρα δεν υπάρχει ιδιαίτερο πρόβλημα.

Σε κάθε περίπτωση, μια καλοσχεδιασμένη συνάρτηση καλό είναι να περιέχει και κώδικα που ελέγχει τις παραμέτρους και χειρίζεται με τον καλύτερο δυνατό τρόπο πιθανές ασυμβατότητες ανάμεσα στα ορίσματα που δίνονται κατά την κλήση της και τον κώδικά της. Αυτό είναι επιπρόσθετα σημαντικό αν η συνάρτηση καλείται ως μέρος ενός μεγαλύτερου έργου (π.χ. μέσα σε σενάρια R, ως τμήμα ενός μεγαλύτερου κομματιού κώδικα η κλάσης κλπ.), όπου ζητούμενο είναι να εκτελείται συνολικά και χωρίς διακοπή, ή αν η συνάρτηση παρέχεται για χρήση από τρίτους (π.χ. ως μέρος ενός πακέτου). Βέβαια, ποιος ακριβώς θα είναι ο «καλύτερος δυνατός τρόπος» χειρισμού προβληματικών ορισμάτων ή σφαλμάτων κατά την εκτέλεση μιας συνάρτησης εξαρτάται από τους στόχους που καλείται να καλύψει και τη λειτουργία της.

Για παράδειγμα, ας θεωρήσουμε την παρακάτω απλή συνάρτηση με όνομα goodmorning που σχεδιάζεται ώστε να δέχεται μια παράμετρο name. Λειτουργικά, ως δημιουργοί της συνάρτησης θέλουμε (ή μάλλον ελπίζουμε) το name να είναι ένα όνομα ανθρώπου και η συνάρτηση να επιστρέφει το κείμενο «Καλημέρα» ακολουθούμενο από το όνομα αυτό.

Κατά τη δημιουργία της συνάρτησης ίσως είναι δύσκολο να διασφαλίσουμε ότι η τιμή που θα περαστεί ως name αντιστοιχεί όντως σε όνομα ανθρώπου, αλλά μπορούμε έστω να διασφαλίσουμε ότι είναι ένα μοναδικό λεκτικό (character) δεδομένο. Στην παρακάτω υλοποίηση της goodmorning, η εντολή is.character(name) ελέγχει αν το αντικείμενο στο name είναι τύπου character, ενώ η length(name)==1 ελέγχει αν το name περιέχει μόνο ένα στοιχείο. Αν δεν ισχύουν τα παραπάνω, θεωρούμε πως τα δεδομένα στο name είναι εκτός προδιαγραφών και η συνάρτηση επιστρέφει «Καλημέρα άγνωστε φίλε»:

```
goodmorning <- function(name)
{
  if (!is.character(name) || !length(name) == 1)
    return("Καλημέρα άγνωστε φίλε")
  return (paste("Καλημέρα", name))
}
```

Έτσι, αν κάποιος χρήστης της συνάρτησης goodmorning την καλέσει με τιμή name ένα διάνυσμα αριθμών, το αποτέλεσμα θα είναι:

```
> goodmorning(1:10)
[1] "Καλημέρα άγνωστε φίλε"
```

Ενώ αν δοθεί ως name ένα μοναδικό λεκτικό (ασχέτως αν είναι πραγματικά ανθρώπινο όνομα) η συνάρτηση θα επιστρέψει:

```
> goodmorning("Μαρία")
[1] "Καλημέρα Μαρία"
```

Εναλλακτικά, αν τα ορίσματα είναι εκτός προδιαγραφών, η συνάρτηση θα μπορούσε να μην επιστρέφει τίποτα. Βέβαια στην R μια συνάρτηση πάντα επιστρέφει ένα αντικείμενο, αλλά αυτό μπορεί να είναι η ειδική τιμή NULL ή (ακόμα καλύτερα) invisible NULL. Η συνάρτηση **invisible** ορίζει πως το αντικείμενο που της δίνεται ως όρισμα είναι προσωρινά αόρατο (δηλαδή δεν εκτυπώνεται αυτόματα μέσω auto-print αν εκτελεστεί η εντολή στο Console). Η εντολή return(invisible(NULL)), ή return(NULL), ή ακόμα και return(), ουσιαστικά η συνάρτηση, επιστρέφει ότι κοντινότερο στο «τίποτα» μπορεί να οριστεί στην R. Η παρακάτω παραλλαγή της goodmorning εφαρμόζει ακριβώς αυτή την προσέγγιση αν οι προαναφερθέντες έλεγχοι της τιμής της παραμέτρου name αποτύχουν:

```
goodmorning <- function(name)
{
  if (!is.character(name) || !length(name) == 1)
    return (invisible(NULL))
  return (paste("Καλημέρα", name))
}
```

Αντί του NULL θα μπορεί να επιλεγεί κάποια άλλη ειδική τιμή (βλ.§2.2.4 Ειδικές τιμές) αν ταιριάζει καλύτερα στους στόχους της συνάρτησης, ενώ κάποιοι προγραμματιστές επιλέγουν να επιστρέψουν οι συναρτήσεις τους αντικείμενα μηδενικού μήκους (όπως το numeric(0)) για τον ίδιο σκοπό.

Ο δημιουργός της συνάρτησης έχει και άλλες επιλογές χειρισμού παραμέτρων εκτός προδιαγραφών. Μπορεί να επιλέξει να εγείρει μια προειδοποίηση (warning) ή ένα σφάλμα (error) με τεχνικές που περιγράφονται στην επόμενη ενότητα (βλ.§5.1.4 Έγερση και χειρισμός σφαλμάτων).

5.1.4 Έγερση και χειρισμός σφαλμάτων

Η ενότητα αυτή περιγράφει τα βασικά στοιχεία δημιουργίας (που αποκαλείται και «έγερση», raise) σφαλμάτων και προειδοποιήσεων για κάποια αντικανονική κατάσταση (condition) κατά τον χρόνο εκτέλεσης (run-time) του κώδικα. Τα σφάλματα (τα οποία αφορούν πιο κρίσιμα προβλήματα) ονομάζονται και «εξαιρέσεις» (exception).

Πέραν της τήρησης βασικών κανόνων σύνταξης, η R ως γλώσσα-διερμηνευτής (interpreted) δεν ελέγχει με αυστηρό τρόπο τον κώδικα για λάθη πριν την εκτέλεση του. Έτσι, ενδεχόμενα σφάλματα διαπιστώνονται κυρίως όταν (και εφόσον) εκτελεστεί κάποια προβληματική εντολή οπότε και εγείρονται τα σχετικά condition (σφάλματος, προειδοποίησης, διαγνωστικού μηνύματος κλπ.). Συναρτήσεις που ο κώδικάς μας καλεί, εγείρουν κάποιο condition κατά την εκτέλεση τους εφόσον κληθούν να κάνουν κάτι το οποίο είναι αντικανονικό ή δεν υποστηρίζουν.

Γενικώς, τα condition οποιουδήποτε τύπου έχουν βασικό στόχο να ενημερώνουν το περιβάλλον εκτέλεσης και τους χρήστες του κώδικα για κάποιο πρόβλημα. Έτσι, η ενότητα αυτή περιγράφει τεχνικές έγερσης αλλά και τεχνικές χειρισμού (από τον κώδικα) πιθανών σφαλμάτων και άλλων καταστάσεων που μπορεί να προκύψουν κατά την εκτέλεση. Ο χειρισμός αντικανονικών συνθηκών δεν αφορά μόνο συναρτήσεις[285] αλλά οποιονδήποτε κώδικα, εντός ή εκτός συναρτήσεων. Όμως, τόσο η ειδοποίηση για προβλήματα και αντικανονικές συνθήκες όσο και ο χειρισμός τους, είναι ιδιαιτέρως σημαντικός στις συναρτήσεις καθώς αυτές αποτελούν συνήθως ολοκληρωμένες μονάδες αυτόνομου και αυτοτελούς κώδικα. Επιπρόσθετα, ο χειρισμός των condition από τον κώδικά μας απαιτεί δημιουργία συναρτήσεων.

Καλό είναι προβληματικές καταστάσεις και σφάλματα που εντοπίζονται κατά την εκτέλεση του κώδικα να επικοινωνούνται προς τα έξω, προς το περιβάλλον και τους χρήστες του κώδικα. Στην R, τα προκαθορισμένα επίπεδα μιας προβληματικής κατάστασης (ενός condition) είναι τρία: message (διαγνωστικά μηνύματα), warning (προειδοποιήσεις) και error (σφάλματα). Τα τελευταία (error) θεωρούνται αρκετά σοβαρά ώστε ο προκαθορισμένος χειρισμός τους να οδηγεί σε τερματισμό της εκτέλεσης του κώδικα. Η μόνη άλλη κατάσταση

[285] βλ. και §3.2.2.4 Τερματισμός εκτέλεσης.

που μπορεί να διακόψει αντικανονικά την εκτέλεση του κώδικα είναι η παρέμβαση του χρήστη (π.χ. πατώντας κάποιο πλήκτρο διακοπής, Ctrl+C ή Esc).

Πώς εγείρονται όμως τα condition αυτά; Αν εντοπιστεί κάποια προβληματική κατάσταση, ο δημιουργός του κώδικα μπορεί απλώς να επιλέξει να εμφανίσει κάποιο σχετικό ενημερωτικό μήνυμα που περιγράφει το πρόβλημα χρησιμοποιώντας συναρτήσεις όπως η cat (βλ.§2.3 Συναρτήσεις κειμένου και ο βασικός τύπος character). Όμως καλύτερη προσέγγιση για την εμφάνιση διαγνωστικών μηνυμάτων είναι η χρήση της ειδικής συνάρτησης **message**, που εγείρει και το σχετικό condition. Τα μηνύματα που δημιουργούνται με τη message θεωρούνται διαγνωστικού περιεχομένου και μπορούν να αγνοηθούν με τη συνάρτηση **suppressMessages**[286].

Για σοβαρότερα προβλήματα χρησιμοποιείται ο μηχανισμός έγερσης προειδοποιήσεων και λαθών που υποστηρίζεται (μεταξύ άλλων) από τις συναρτήσεις **warning** (για προειδοποιήσεις) και **stop** (για σοβαρότερα σφάλματα).

Στην πιο απλή της μορφή η συνάρτηση stop δέχεται ένα λεκτικό μήνυμα λάθους και εγείρει κατάσταση σφάλματος, τη σοβαρότερη δηλαδή μορφή condition που συνήθως τερματίζει την εκτέλεση του κώδικα εμφανίζοντας το δοθέν μήνυμα. Για διαγνωστικούς λόγους, το μήνυμα περιλαμβάνει αυτόματα και το όνομα της συνάρτησης στο οποίο προέκυψε το πρόβλημα. Το τελευταίο μήνυμα λάθους μπορεί να ανακληθεί αργότερα με τη συνάρτηση **geterrmessage**, ενώ χρήσιμο εργαλείο στην επίλυση του προβλήματος που οδήγησε στο σφάλμα μπορεί να αποτελέσει η συνάρτηση **traceback**. Όταν εγερθεί ένα σφάλμα, η R καλεί τη συνάρτηση που έχει οριστεί ως τρέχουσα τιμή της επιλογής "error"[287] στη συνεδρία της R όπου εκτελείται ο κώδικας (βλ.§4.2.1.3 Η λίστα επιλογών συνεδρίας). Η συνάρτηση αυτή που χειρίζεται τα σφάλματα μπορεί να αλλάξει μέσω επέμβασης στην επιλογή "error", αλλά όπως αναφέρθηκε παραπάνω, η συνήθης προκαθορισμένη συνάρτηση "error" οδηγεί σε τερματισμό εκτέλεσης του κώδικα.

Έτσι αν στο προηγούμενο παράδειγμα συνάρτησης goodmorning (βλ.§5.1.3 Έλεγχος παραμέτρων) επιλεγεί να γίνει έγερση κατάστασης σφάλματος όταν η παράμετρος name είναι εκτός προδιαγραφών, τότε ο σχετικός κώδικας θα μπορούσε να είναι ο παρακάτω:

```
goodmorning <- function(name)
{
  if (!is.character(name) || !length(name) == 1)
    stop("Το όνομα ",name," που δόθηκε δεν μπορεί να
χρησιμοποιηθεί")
  return (paste("Καλημέρα", name))
}
```

Εφόσον η παραπάνω συνάρτηση κληθεί με μια αναγνωρίσιμα ακατάλληλη τιμή για την παράμετρο name (όπως π.χ. στην εντολή goodmorning(10) όπου η τιμή 10 δεν είναι character), η εκτέλεση σταματά και δημιουργείται (από τα ορίσματα της stop) το σχετικό μήνυμα σφάλματος. Αν η συνάρτηση καλείται μέσα από ένα σενάριο ή ως μέρος ενός μεγαλύτερου τμήματος κώδικα, ο τερματισμός θα αφορά και τον κώδικα που την καλεί:

```
> goodmorning(10)
Error in goodmorning(10) :
  Το όνομα 10 που δόθηκε δεν μπορεί να χρησιμοποιηθεί
```

Όπως προαναφέρθηκε, η geterrmessage επιστρέφει (μορφοποιημένο) το πλέον πρόσφατο μήνυμα σφάλματος:

```
> cat(geterrmessage())
Error in goodmorning(10) :
  Το όνομα 10 που δόθηκε δεν μπορεί να χρησιμοποιηθεί
```

Επιπροσθέτως, η συνάρτηση traceback κατονομάζει τη σειρά κλήσεων συναρτήσεων (με τις τιμές των παραμέτρων τους), δηλαδή περιγράφει τη στοίβα κλήσεων (call stack)[288] που οδήγησαν στο σφάλμα:

---

[286] Οι message και suppressMessages δεν αναλύονται περαιτέρω καθώς είναι παρεμφερείς με τις συναρτήσεις warning και suppressWarnings που παρέχουν μεγαλύτερη λειτουργικότητα και περιγράφονται παρακάτω.

[287] Για ανάκληση της τιμής της επιλογής συνεδρίας "error" μπορεί να χρησιμοποιηθεί η εντολή getOption("error").

[288] Η στοίβα κλήσεων (call stack) είναι δομή στην οποία αποθηκεύεται (push) η τρέχουσα κατάσταση όταν γίνεται κλήση μιας συνάρτησης. Η τρέχουσα αυτή κατάσταση ανακαλείται (pop) όταν η συνάρτηση επιστρέψει ώστε να συνεχιστεί με βάση αυτή η εκτέλεση του κώδικα. Με τον τρόπο αυτό υλοποιείται ο μηχανισμός κλήσεων συναρτήσεων και στην R. Μια μικρή παραλλαγή του παραπάνω εφαρμόζεται όταν η R κάνει lazy evaluation (ή κλήση βάσει ανάγκης, call-by-need), οπότε η γλώσσα αποφασίζει να αναβάλει την αποτίμηση κάποιου μέρους του κώδικα (π.χ. την κλήση κάποιας συνάρτησης)

```
> traceback()
2: stop("Το όνομα ", name, " που δόθηκε δεν μπορεί να
χρησιμοποιηθεί") at #3
1: goodmorning(10)
```

Στο παραπάνω παράδειγμα το τελευταίο σφάλμα προήλθε από τη συνάρτηση goodmorning που ακολούθως κάλεσε τη stop (η οποία έκανε και έγερση του σφάλματος). Για τον λόγο αυτό η συνάρτηση goodmorning καταγράφεται ως 1η που κλήθηκε ενώ η stop ως 2η. Όταν οι κλήσεις που οδήγησαν στο σφάλμα είναι περισσότερες και άρα η στοίβα κλήσεων έχει περισσότερα επίπεδα, η εξέτασή της μπορεί να είναι ιδιαίτερα χρήσιμη στην αναγνώριση και επίλυση του προβλήματος.

Καθώς η ανάγκη για ελέγχους που οδηγούν σε έγερση κατάστασης σφάλματος είναι πολύ συνηθισμένη, εναλλακτικά της stop παρέχεται η συνάρτηση **stopifnot** η οποία τερματίζει την εκτέλεση αν αποτύχει ο έλεγχος και εμφανίζει μήνυμα λάθους που περιγράφει το πρόβλημα. Παρακάτω ακολουθεί παραλλαγή της goodmorning με χρήση της stopifnot:

```
goodmorning <- function(name)
{
  stopifnot(is.character(name) && length(name) == 1);
  return (paste("Καλημέρα", name))
}
```

Ειδικά για τον έλεγχο τιμών σε παραμέτρους συναρτήσεων, το πακέτο ‘base’ της R παρέχει τη συνάρτηση **match.arg**. Η συγκεκριμένη συνάρτηση δέχεται ένα διάνυσμα επιτρεπτών τιμών τύπου character και ελέγχει αν η τρέχουσα τιμή κάποιας μεταβλητής ή παραμέτρου περιέχεται σε αυτές, εγείροντας κατάσταση σφάλματος αν αυτό δεν ισχύει. Η παρακάτω παραλλαγή της goodmorning θα εκτελεστεί κανονικά (χωρίς σφάλμα) μόνο αν το name είναι κάποιο από τα ορισμένα στη match.arg ονόματα:

```
goodmorning <- function(name)
{
  match.arg(name,c("Μαρία","Αντρέας","Όλγα","Γιώργος"))
  return (paste("Καλημέρα", name))
}
```

Αν το πρόβλημα δεν είναι σημαντικό και η εκτέλεση του κώδικα δεν είναι απαραίτητο να τερματιστεί, μπορεί, αντί λάθους, να δημιουργηθεί μια προειδοποίηση. Αυτό γίνεται με τη συνάρτηση warning. Ο τρόπος που θα γίνει χειρισμός τέτοιων μηνυμάτων προειδοποίησης (αν θα εμφανιστούν και πότε) εξαρτάται από την τρέχουσα τιμή της επιλογής "warn"[289] στη συνεδρία της R όπου εκτελείται ο κώδικας (βλ.§4.2.1.3 Η λίστα επιλογών συνεδρίας). Αν η τιμή αυτή είναι αρνητική οι προειδοποιήσεις αγνοούνται, αν το "warn" είναι 0 οι προειδοποιήσεις εμφανίζονται (μαζί με άλλα) μετά την ολοκλήρωση εκτέλεσης όλου του κώδικα, αν είναι 1 εμφανίζονται καθώς προκύπτουν, ενώ για τιμή "warn"≥2 μετατρέπονται σε σφάλματα (αντίστοιχα κλήσης stop). Τέλος, αν δοθεί τιμή TRUE στην παράμετρο immediate. της συνάρτησης warning, η προειδοποίηση εμφανίζεται άμεσα, ασχέτως της τιμής στην επιλογή "warn". Στην επόμενη παραλλαγή της goodmorning η χρήση της warning αντί της stop δημιουργεί άμεσα ένα προειδοποιητικό μήνυμα όταν εντοπιστεί ακατάλληλη τιμή στην παράμετρο name:

```
goodmorning <- function(name)
{
  if (!is.character(name) || !length(name) == 1)
    warning("Το ",name," δεν είναι όνομα", immediate.=T)
  return (paste("Καλημέρα", name))
}
```

Παρατηρήστε πως αν κληθεί η παραπάνω συνάρτηση με ακατάλληλο name, το warning εμφανίζει μήνυμα (που, όπως γίνεται και στα λάθη, περιλαμβάνει αυτόματα το όνομα της συνάρτησης στην οποία προέκυψε η προβληματική κατάσταση) αλλά δεν τερματίζει την εκτέλεση της συνάρτησης:

```
> goodmorning(10)
Warning in goodmorning(10) : Το 10 δεν είναι όνομα
```

και να τον αποτιμήσει μόνο όταν και εφόσον υπάρξει ανάγκη από το αποτέλεσμα που θα προκύψει από τον κώδικα αυτό (για περισσότερα, βλ. [33]).

[289] Για ανάκληση της τιμής της επιλογής συνεδρίας "warn" μπορεί να χρησιμοποιηθεί η εντολή getOption("warn").

```
[1] "Καλημέρα 10"
```

Τέλος, η συνάρτηση **suppressWarnings** δέχεται μια έκφραση R την οποία εκτελεί αγνοώντας προειδοποιήσεις που ενδεχομένως προκύψουν. Έτσι, η εντολή suppressWarnings(goodmorning(10)) θα εκτελέσει την εντολή goodmorning(10) χωρίς προειδοποιήσεις.

Τα παραπάνω περιγράφουν κάποιους από τους μηχανισμούς έγερσης κατάστασης προβλήματος ή σφάλματος στην R. Οι μηχανισμοί αυτοί επιτρέπουν στις συναρτήσεις, τα σενάρια και τον κώδικα γενικότερα να ενημερώσει για αντικανονικές συνθήκες που προέκυψαν κατά την εκτέλεσή του και πιθανώς να τερματιστεί. Τα αντικείμενα που εμπλέκονται για την ειδοποίηση σφαλμάτων ανήκουν στην κλάση **condition**[290] και εγείρονται καλώντας τη συνάρτηση **signalCondition**. Για περισσότερα σχετικά με τον ορισμό και τη χρήση αντικειμένων κλάσης condition βλ. [33] και help(conditions).

Όπως προαναφέρθηκε (και θα έχετε σίγουρα διαπιστώσει ήδη αν χρησιμοποιείτε την R), καταστάσεις προβλήματος ή σφάλματος μπορούν να προκύψουν και αυτόματα, κατά την εκτέλεση των εντολών, οδηγώντας σε τερματισμό της εκτέλεσης. Η συνάρτηση **try** επιτρέπει να γίνει προσπάθεια εκτέλεσης κάποιου κώδικα συμπιέζοντας και αποθηκεύοντας πιθανά σφάλματα, αλλά η πιο γενικευμένη συνάρτηση **tryCatch** επιτρέπει τον ορισμό της δράσης που θα ληφθεί αν προκύψουν προβληματικές καταστάσεις κατά την εκτέλεση. Αυτός ο χειρισμός προβληματικών καταστάσεων ονομάζεται «χειρισμός εξαιρέσεων» (exception handling) και είναι μια ειδική μορφή ελέγχου της ροής εκτέλεσης (όπως η εντολή if, βλ.§3.2.2 Έλεγχος ροής εκτέλεσης (control flow)), αλλά εδώ το κριτήριο ελέγχου είναι η έγερση ή μη ενός condition. Με τον μηχανισμό αυτό, ο κώδικάς μπορεί να αναλάβει τον χειρισμό σφαλμάτων που μπορεί να προκύψουν, αποφεύγοντας έτσι τον τερματισμό της εκτέλεσης του. Η συνήθης σύνταξη της tryCatch για τον χειρισμό σφαλμάτων και προειδοποιήσεων είναι:

tryCatch(expr = *α*, warning = *β*, error = *γ*, finally = δ)

Στο πρώτο όρισμα (με όνομα expr), η τιμή *α* είναι η έκφραση που θα αποτιμηθεί, δηλαδή ο κώδικας που θέλουμε να εκτελεστεί και για τον οποίο θα χειριστούμε εμείς όποια warning (προειδοποίηση) ή error (σφάλμα) προκύψουν. Αν δεν εγερθεί μια προβληματική κατάσταση, η tryCatch θα εκτελέσει όλο τον κώδικα στο *α* (το οποίο συνήθως είναι ένα μπλοκ κώδικα) και θα επιστρέψει το αποτέλεσμα του. Τα *β* και *γ* που φαίνονται ως ορίσματα στις προαιρετικές παραμέτρους warning και error πρέπει να είναι συναρτήσεις οι οποίες δέχονται ένα όρισμα (στο οποίο η R θα τοποθετήσει αυτόματα το αντικείμενο κλάσης condition που προέκυψε και περιέχουν τον κώδικα που χειρίζεται το πρόβλημα. Η *β* καλείται αν έχει εγερθεί προειδοποίηση, ενώ η *γ* αν έχει εγερθεί σφάλμα. Σε περίπτωση κλήσης των *β* ή *γ*, το αποτέλεσμα των συναρτήσεων αυτών θα είναι και η τιμή που επιστρέφει η tryCatch. Τέλος, αν οριστεί η παράμετρος finally, το *δ* είναι έκφραση που θα εκτελεστεί ως τελευταίο βήμα, σε συνέχεια της έκφρασης *α* ή ως κώδικας εκκαθάρισης (clean-up code).

Συνοψίζοντας τα παραπάνω, η tryCatch εκτελεί το *α* και επιστρέφει το αποτέλεσμα. Αν όμως κατά την εκτέλεση του *α* προκύψει warning, εκτελεί και επιστρέφει το β, ενώ αν προκύψει error, εκτελεί και επιστρέφει το *γ*. Μετά (αν έχει οριστεί finally) εκτελεί τον κώδικα *δ*. Ας δούμε ένα παράδειγμα:

```
tryCatch( z+1, error=function(e){return(0)} )
```

Η παραπάνω εντολή αποτιμά και επιστρέφει την έκφραση z + 1. Αν όμως δεν υπάρχει μεταβλητή z στο περιβάλλον της εντολής, επιστρέφει 0. Με χρήση των συναρτήσεων eval και parse, οποιοδήποτε κείμενο κώδικα R μπορεί να χρησιμοποιηθεί ως όρισμα της παραμέτρου expr στην tryCatch και να γίνει χειρισμός σφαλμάτων. Το παράδειγμα που ακολουθεί είναι λειτουργικά όμοιο με το προηγούμενο αλλά ο κώδικας z+1 είναι αποθηκευμένος ως character σε μεταβλητή x:

```
a<-"z+1"
tryCatch( eval(parse(text=a)), error=function(e){return(0)} )
```

Ακολουθεί ένα παράδειγμα συνάρτησης που θα διαμορφωθεί ώστε να αξιοποιεί την tryCatch:

```
f <- function(x) log (2 * x , 10)
```

Με την παραπάνω μορφή, η συνάρτηση f δέχεται ένα όρισμα x και υπολογίζει τον λογάριθμο (με βάση 10) του 2x. Αν και η συνάρτηση αυτή είναι ιδιαίτερα απλή, μπορούν εύκολα να προκύψουν προβληματικές καταστάσεις. Π.χ. η συνάρτηση log (και κατά συνέπεια η f που την καλεί) θα εγείρει προειδοποίηση αν το 2x είναι μικρότερο του μηδενός, καθώς η log έχει σχεδιαστεί ώστε να επιστρέφει NaN και εγείρει σχετικό warning αν το όρισμά της είναι αρνητικό. Ακόμα πιο σοβαρό σφάλμα θα προκύψει αν το x ανήκει σε τύπο αντικειμένου ο οποίος δεν υποστηρίζει μαθηματικούς υπολογισμούς όπως ο πολλαπλασιασμός ή ο υπολογισμός λογαρίθμου. Έτσι μια εντολή όπως y <- f("test") θα εγείρει σφάλμα, η εκτέλεση θα τερματιστεί και δεν θα γίνει καμία ανάθεση τιμής στο y:

[290] Είναι κλάση S3 (βλ. §6.3 Κλάσεις S3), έτσι μπορούν να οριστούν και νέοι τύποι condition πέραν των προκαθορισμένων error, warning κλπ.

```
> y <- f("test")
Error in 2 * x : non-numeric argument to binary operator
```

Παρακάτω η f έχει προσαρμοστεί ώστε να χειρίζεται εσωτερικά προβλήματα κατάστασης warning ή error που ενδεχομένως να προκύψουν κατά την εκτέλεση της έκφρασης log(2 * x, 10):

```
f <- function(x)
{
  tryCatch(
    expr = { log (2 * x , 10) },
    warning = function(q) { return(-100) },
    error = function(q) { return(-200) }
    )
}
```

Εδώ (χάριν παραδείγματος) επιλέχτηκε να επιστρέφεται η τιμή -100 αν προκύψει warning και η τιμή -200 αν προκύψει error κατά την εκτέλεση του κώδικα log(2 * x, 10). Έτσι, f(50) θα επιστρέψει 2, f(0) θα επιστρέψει (φυσιολογικά) την τιμή Inf, f(-50) που κανονικά θα εγείρει warning επιστρέφει -100, ενώ f("test") που εγείρει σφάλμα θα επιστρέψει -200 και η εκτέλεση του κώδικα θα συνεχίσει κανονικά:

```
> f("test")
[1] -200
```

Έτσι η ανάθεση τιμής y <- f("test") που πριν απέτυχε και τερματίστηκε, τώρα θα ολοκληρωθεί κανονικά αναθέτοντας τη τιμή -200 στο y, ενώ η εκτέλεση μπλοκ, συνάρτησης, σεναρίου κλπ. στα οποία η εντολή περιέχεται, θα συνεχιστεί. Χάριν πληρότητάς ακολουθεί μια παραλλαγή της f που μετατρέπει warning και error σε απλά διαγνωστικά μηνύματα, αλλά και κάνει ενδεικτική χρήση της παραμέτρου finally στη συνάρτηση tryCatch:

```
f <- function(x)
{
  r <- tryCatch(
    expr = { log (2 * x , 10) },

    warning = function(q) {
      message("Προειδοποίηση στο ", q$call,":\n", q$message )
      return(-100)
    },

    error = function(q) {
      message("Σφάλμα στο ", q$call, ":\n", q$message)
      return(-200)
    },

    finally = { cat("Ολοκληρώθηκε και...") }
  )

  cat("ο κώδικας συνεχίζει.\n")
  return(r)
}
```

Με τον παραπάνω κώδικα η f παρουσιάζει την ακόλουθη συμπεριφορά:

```
> f(50)
Ολοκληρώθηκε και...ο κώδικας συνεχίζει.
[1] 2

> f(-50)
Προειδοποίηση στο log2 * x10:
NaNs produced
Ολοκληρώθηκε και...ο κώδικας συνεχίζει.
[1] -100
```

```
> f("test")
Σφάλμα στο *2x:
non-numeric argument to binary operator
Ολοκληρώθηκε και...ο κώδικας συνεχίζει.
[1] -200
```

Περισσότερα σχετικά με τον χειρισμό σφαλμάτων και προειδοποιήσεων θα βρείτε στο help(conditions) και μεταξύ άλλων στα [33] και [60]. Εκεί αναφέρονται και άλλες συναρτήσεις χειρισμού σφαλμάτων όπως η **withCallingHandlers** που είναι παρεμφερής με την tryCatch αλλά δημιουργεί χειριστές λαθών οι οποίοι είναι τοπικοί στο πλαίσιο εκτέλεσης του κώδικα. Επίσης, στο CRAN υπάρχουν πακέτα που παρέχουν εναλλακτικούς τρόπους χειρισμού καταστάσεων σφαλμάτων, όπως π.χ. το πακέτο 'tryCatchLog' [63] το οποίο βοηθά στην καταγραφή διαγνωστικών μηνυμάτων και σφαλμάτων καθώς και την εξέταση του περιβάλλοντος και της στοίβας κλήσεων στο οποίο προκλήθηκαν.

5.1.5 Αλληλεπίδραση με το περιβάλλον

Οι συναρτήσεις στην R ακολουθούν σε μεγάλο βαθμό τις αρχές του συναρτησιακού προγραμματιστικού μοντέλου (functional programming, βλ. §5.3 Συναρτησιακός προγραμματισμός). Για τον λόγο αυτό έχουν κάποιες διαφορές από τις συναρτήσεις που βρίσκουμε σε άλλες συνήθεις γλώσσες προγραμματισμού. Οι διαφορές αυτές συνήθως απλουστεύουν τη χρήση συναρτήσεων στην R και αφορούν κυρίως τον τρόπο με τον οποίο οι συναρτήσεις αλληλοεπιδρούν και ανταλλάσσουν τιμές με το περιβάλλον τους.

Όπως σε πολλές άλλες γλώσσες προγραμματισμού, το σώμα μίας συνάρτησης αποτελεί ένα πλαίσιο ορισμού τοπικών (local) μεταβλητών, δηλαδή ένα τοπικό environment εκτέλεσης του κώδικα. Αυτό είναι παρεμφερές με όσα αναφέρθηκαν για τη συνάρτηση local (βλ.§3.2.1.2 Χρήση μπλοκ κώδικα για ορισμό εμβέλειας μεταβλητών). Οι μεταβλητές που δημιουργούνται εντός μιας συνάρτησης (με τους συνήθεις τελεστές '<-', '->' και '=') έχουν εμβέλεια μόνο εντός της συνάρτησης αυτής, υπάρχουν μόνο για τη συνάρτηση[291].

Δοκιμάστε:	**Σχόλιο**
`f<-function(){x<-100}`	Συνάρτηση με όνομα f, αναθέτει την τιμή 100 σε μεταβλητή x.
`f()`	Κλήση της f. Το τοπικό x έγινε 2 μετά διαγράφηκε. Επιστρέφει 100.
`x`	Προσπάθεια ανάκλησής του x οδηγεί σε σφάλμα καθώς το x δεν υπάρχει.

Όταν μια συνάρτηση έχει οριστεί ως τοπικό αντικείμενο εντός κάποιας άλλης συνάρτησης, η αναζήτηση, από την εσωτερική (τοπική) συνάρτηση, αντικειμένων εξωτερικών σε αυτή, ξεκινά από το περιβάλλον της συνάρτησης μέσα στην οποία η τελευταία ορίστηκε:

```
x <- 10                # εξωτερικό x (καθολικό, στο Global)
y <- 2                 # εξωτερικό y (καθολικό, στο Global)

f1 <- function()  {
  f2 <- function() {       # f2 τοπικό στην f1
    z <- 30                # z τοπικό στην f2
    print(z)
    print(y)
    print(x)
  }                        # εδώ τελειώνει ο ορισμός της f2
  y <- 20                  # y τοπικό στην f1
  f2()                     # κλήση της f2 από την f1
}
```

Στο παραπάνω παράδειγμα η συνάρτηση f2 είναι τοπικό αντικείμενο της f1. Όταν κληθεί η f1 δημιουργεί τα τοπικά αντικείμενα f2 (συνάρτηση) και y (με τιμή 20) και ακολούθως καλεί την f2. Η f2 τυπώνει τα όποια z, y και x εντοπίσει. Αντικείμενο με όνομα z εντοπίζεται τοπικά στην f2 (και έχει τιμή 30), το y εντοπίζεται στο άμεσα εξωτερικό της περιβάλλον της f2 (την f1) και έχει τιμή 20, ενώ το x αναζητείται στα παραπάνω επίπεδα,

[291] Και περιβάλλοντα έχουν το περιβάλλον της συνάρτησης ως γονικό (βλ. παρακάτω. Για τα γονικά περιβάλλοντα, βλ. §2.2.3 Ο ρόλος των περιβαλλόντων).

και εντοπίζεται εξωτερικά (στο καθολικό περιβάλλον/Global Environment, με τιμή 10)[292]. Το αποτέλεσμα είναι:

```
> f1()
[1] 30
[1] 20
[1] 10
```

Το παράδειγμα αυτό δίνει μια εικόνα του τρόπου αναζήτησης εξωτερικών αντικειμένων από τις συναρτήσεις. Το θέμα αυτό αφορά την εμβέλεια μεταβλητών και την αναζήτησή τους σε περιβάλλοντα, κάτι που περιγράφεται εκτενώς σε άλλη ενότητα του βιβλίου (βλ.§2.2.3 Ο ρόλος των περιβαλλόντων).

Σε κάθε περίπτωση, τιμές από το περιβάλλον περνούν προς μια συνάρτηση μόνο μέσω των παραμέτρων της συνάρτησης ή με πρόσβαση από τη συνάρτηση σε εξωτερικές μεταβλητές που ήδη υπάρχουν στο εξωτερικό περιβάλλον της. Όμως στην R μια συνάρτηση μπορεί μόνο να διαβάσει τις τιμές εξωτερικών σε αυτή μεταβλητών χωρίς να μπορεί να τις αλλάξει (τουλάχιστον όχι με τους συνήθεις τελεστές ανάθεσης τιμής όπως το '<-'). Ο λόγος είναι πως στην R, λόγω της αρχής copy-on-modify[293], κάθε αλλαγή τιμής με τους συνήθεις τελεστές ανάθεσης τιμής δημιουργεί ένα νέο αντικείμενο. Και εφόσον η ανάθεση εκτελείται στο τοπικό environment της συνάρτησης, εντός του σώματος μιας συνάρτησης, το νέο αυτό αντικείμενο θα ανατεθεί σε μια νέα τοπική μεταβλητή (ακόμα και αν υπάρχει συνώνυμή της στο εξωτερικό περιβάλλον). Στο επόμενο παράδειγμα γίνεται πρόσβαση από τη συνάρτηση σε εξωτερική μεταβλητή (με όνομα x), όμως η ανάθεση τιμής γίνεται σε νέα τοπική μεταβλητή (που εδώ τυχαίνει να έχει επίσης το ίδιο όνομα):

Δοκιμάστε:	**Σχόλιο**
`x<-1`	Ανάθεση της τιμής 1 σε μεταβλητή x.
`f<-function(){x<-x+1}`	Συνάρτηση με όνομα f, αναθέτει την τιμή x+1 σε νέα τοπική μεταβλητή x.
`f()`	Κλήση της f. Το τοπικό x έγινε 2 μετά διαγράφηκε. Επιστρέφει 2.
`x`	Ανάκληση του x, επιστρέφει 1. Το εξωτερικό x δεν άλλαξε από την κλήση της f.

Όσον αφορά τις παραμέτρους των συναρτήσεων, για τους ίδιους λόγους που αναφέρθηκαν προηγουμένως, κατά την κλήση της συνάρτησης περνά στην παράμετρο μια αναφορά του αρχικού αντικειμένου που προσδιορίζεται από το όρισμα (όπως συμβαίνει όταν δυο μεταβλητές αναφέρονται στο ίδιο αντικείμενο). Έτσι το αντικείμενο δεν αντιγράφεται άμεσα. Αν όμως χρειαστεί να γίνει κάποια αλλαγή, το αντικείμενο αντιγράφεται. Το νέο αντικείμενο έχει τοπική εμβέλεια καθώς δημιουργείται στο τρέχον περιβάλλον που είναι το τοπικό περιβάλλον της συνάρτησης. Το παρακάτω παράδειγμα δημιουργεί μια συνάρτηση f που προσπαθεί να αναδείξει τον μηχανισμό[294]:

```
f<-function(a,b)
{
  cat(paste("Θέση αντικ. εξωτερικής x:",tracemem(x),"\n"))
  cat(paste("Θέση αντικ. παραμέτρου a:",tracemem(a),"\n"))
  cat(paste("Θέση αντικ. παραμέτρου b:",tracemem(b),"\n"))
  a<-a+b
  cat(paste("Θέση στο a μετά την αλλαγή:",tracemem(a),"\n"))
  x<-a+b
  cat(paste("Θέση στο x μετά την αλλαγή:",tracemem(x),"\n"))
  return(b)
}
```

Η συνάρτηση f δείχνει τη θέση (στη μνήμη) των αντικειμένων στα οποία αναφέρονται οι παράμετροι και οι μεταβλητές της. Αρχικά δημιουργούμε μια (καθολική, άρα εξωτερική) μεταβλητή με την εντολή x<-1. Αν ζητήσουμε τη θέση του αντικειμένου στο οποίο η x αναφέρεται:

```
> tracemem(x)
[1] "<000001EEF9FBF500>"
```

Αν ακολούθως καλέσουμε την f με παραμέτρους a=x και b=x:

```
> tracemem(f(x,x))
```

[292] Περισσότερα για την αναζήτηση αντικειμένων κατά την εκτέλεση εντολών αναφέρονται αλλού (βλ. π.χ. §2.2.3 Ο ρόλος των περιβαλλόντων.

[293] βλ.§2.2.1 Δημιουργία, χρήση, μετατροπή μεταβλητών και αριθμητικά .

[294] Για τη συνάρτηση tracemem βλ.§3.1.4 Εργαλεία προφίλ (profiling).

```
Θέση αντικ. εξωτερικής x: <000001EEF9FBF500>
Θέση αντικ. παραμέτρου a: <000001EEF9FBF500>
Θέση αντικ. παραμέτρου b: <000001EEF9FBF500>
Θέση στο a μετά την αλλαγή: <000001EEFECFF250>
Θέση στο x μετά την αλλαγή: <000001EEFECFF170>
[1] "<000001EEF9FBF500>"
```

Η τελευταία τιμή είναι η θέση του αντικειμένου που επέστρεψε η f, δηλαδή του αντικειμένου στο b. Παρατηρούμε πως με την κλήση f(x,x) αρχικά, τόσο το εξωτερικό x όσο και οι παράμετροι a και b αναφέρονται στο ίδιο αντικείμενο στη μνήμη. Ακολούθως, μετά τις αναθέσεις τιμής στα a και (της τοπικής πλέον) x δημιουργούνται για αυτά νέα αντικείμενα (σε άλλες θέσεις). Τέλος, το b στο οποίο δεν έγινε κάποια ανάθεση, εξακολουθεί να αναφέρεται στο ίδιο αντικείμενο με το εξωτερικό x, το οποίο επίσης παραμένει χωρίς αλλαγές και εξακολουθεί να περιέχει την τιμή 1. Ως αποτέλεσμα των παραπάνω, οι παράμετροι μπορούν να περάσουν τιμές μόνο προς τη συνάρτηση, οδηγώντας σε συμπεριφορά αντίστοιχη του call-by-value[295].

Συνοψίζοντας, η συνάρτηση δεν μπορεί να αλλάξει (με τους συμβατικούς τελεστές ανάθεσης <-, -> και =) τις τιμές μεταβλητών που δεν είναι τοπικές σε αυτή. Πρέπει πάντως να αναφερθεί ότι στον κανόνα αυτό υπάρχουν πολλές εξαιρέσεις, οι οποίες επιτρέπουν την αλλαγή τιμών σε εξωτερικές μεταβλητές. Μεταξύ αυτών είναι η ανάθεση με χρήση των τελεστών '<<-' ή '->>' ή η πρόσβαση μέσω σχετικών συναρτήσεων στο εξωτερικό αντικείμενο τύπου environment, τεχνικές που περιγράφονται στην §4.2.2.1 Περιβάλλοντα και συναρτήσεις. Επιπρόσθετα κάποιοι τύποι αντικειμένων επιτρέπουν αλλαγές στο ίδιο το αρχικό (εξωτερικό) αντικείμενο[296] χωρίς να γίνει αντιγραφή του σε νέο (άρα με χρήση της αναφοράς σε αυτό). Αν πάντως δεν αξιοποιηθούν οι εξαιρέσεις και χρησιμοποιηθούν μόνο οι συμβατικοί τελεστές ανάθεσης, τότε επιτυγχάνεται δημιουργία συναρτήσεων οι οποίες δεν αλλοιώνουν το περιβάλλον τους με παράπλευρο τρόπο (side-effect), κάτι που αποτελεί κανόνα του συναρτησιακού προγραμματιστικού παραδείγματος.

Τέλος, οι συναρτήσεις επιστρέφουν πάντα ένα (και μόνο ένα) αντικείμενο. Ο τύπος του αντικειμένου που επιστρέφεται μπορεί να είναι οποιοσδήποτε και δεν προσδιορίζεται κατά τον ορισμό της συνάρτησης. Η επιστροφή αυτού του αντικειμένου είναι ο συνήθης τρόπος με τον οποίο η συνάρτηση αποκρίνεται, απαντά και επιστρέφει αποτελέσματα στο περιβάλλον που την καλεί. Σε περιπτώσεις αδυναμίας επιστροφής κάποιας έγκυρης τιμής οι δημιουργοί της συνάρτησης μπορεί να επιλέξουν να επιστρέφει κάποια ειδική τιμή[297]. Όπως έχει αναφερθεί ήδη, οι συναρτήσεις επιστρέφουν το αντικείμενο που καθορίζεται με τη λέξη return. Αν η εκτέλεση του κώδικα της συνάρτησης φτάσει στο τέλος της χωρίς να συναντήσει return, η συνάρτηση επιστρέφει το αποτέλεσμα που παράχθηκε από την τελευταία εντολή που εκτέλεσε. Αν δεν εκτελέστηκε απολύτως τίποτα, η συνάρτηση επιστρέφει την ειδική τιμή NULL.

5.1.6 Συναρτήσεις ως ορίσματα και επιστρεφόμενες τιμές

Μια συνάρτηση μπορεί να περαστεί ως τιμή παραμέτρου άλλης συνάρτησης. Μπορεί επίσης να επιστραφεί από άλλες συναρτήσεις. Έχουν ήδη αναφερθεί παραδείγματα χρήσης μιας συνάρτησης ως όρισμα κάποιας άλλης[298] αλλά η ευελιξία αυτή απαιτεί μερικά ακόμα απλά παραδείγματα για την καλύτερη κατανόησή της. Το

[295] Υπάρχουν διάφορες μέθοδοι οι οποίες μπορεί να υποστηρίζονται από μια γλώσσα προγραμματισμού για το πέρασμα δεδομένων, αντικειμένων κλπ. προς μια συνάρτηση μέσω των παραμέτρων της. Όταν χρησιμοποιείται η μέθοδος call-by-value (ή pass-by-value) περνά στη συνάρτηση ένα αντίγραφο της τιμής της παραμέτρου ή αν πρόκειται για έκφραση που πρέπει να αποτιμηθεί, ένα αντίγραφο του αποτελέσματος της αποτίμησης της. Στην call-by-reference (ή pass-by-reference) και στη συναφή της call-by-address, περνά στη συνάρτηση πληροφορία εντοπισμού του ίδιου του αρχικού αντικειμένου που ορίστηκε στην παράμετρο, π.χ. η θέση του αντικειμένου αυτού στη μνήμη. Έτσι η συνάρτηση μπορεί να εντοπίσει, να επεξεργαστεί και ίσως να αλλάξει το αρχικό αντικείμενο. Αν πρόκειται για έκφραση που πρέπει να αποτιμηθεί (και το συντακτικό της γλώσσας επιτρέπει call-by-reference σε τέτοιες περιπτώσεις, κάτι που συνήθως δεν συμβαίνει) τότε η έκφραση αποτιμάται και περνά η θέση που βρίσκεται το αποτέλεσμα. Τέλος, στο call-by-name (ή pass-by-name) περνά το όνομα του αρχικού αντικειμένου ή αν πρόκειται για έκφραση, ή ίδια η έκφραση και η επεξεργασία στη συνάρτηση προχωρά βάσει αυτών. Παραλλαγή του call-by-name είναι και το call-by-need που επιτρέπει lazy evaluation, βλ. υποσημείωση 288.

[296] Όπως ο τύπος environment (βλ.§4.2.2.2 Περιβάλλοντα και ο μηχανισμός αναφοράς), ο τύπος data.table (βλ.§4.3.1 Οι τύποι tibble και data.table) και αντικείμενα που βασίζονται σε κλάσεις RS ή R6 (βλ. §6.5 Κλάσεις αναφοράς).

[297] βλ. και παραπάνω §5.1.3 Έλεγχος παραμέτρων.

[298] βλ. π.χ. §4.1.3.4 Η οικογένεια συναρτήσεων apply αλλά και το τελευταίο παράδειγμα στην §5.1.2 Παράμετροι.

παρακάτω είναι ένα παράδειγμα ορισμού και κλήσης μιας συνάρτησης (με όνομα f) όπου τιμές μιας παραμέτρου της (με όνομα a) είναι άλλες συναρτήσεις:

Δοκιμάστε:	**Σχόλιο**
`f<-function(x,a) return(a(x))`	Η f δέχεται παραμέτρους x και a, επιστρέφει a(x).
`q<-function(z) return(2*z)`	Η q είναι μια απλή συνάρτηση, δέχεται x και επιστρέφει 2*x.
`f(1:5,sum)`	Καλεί την f για x=1:5 και a=sum, επιστρέφει 15 (= sum(1:5)).
`f(1:5,q)`	Καλεί την f για x=1:5 και a=q, επιστρέφει c(2, 4, 6, 8, 10) (= q(1:5)).

Στο παράδειγμα που ακολουθεί γίνεται επιστροφή μίας συνάρτησης από άλλη συνάρτηση:

Δοκιμάστε:	**Σχόλιο**
`f1 <- function(x, y) { x * y }`	Η f1 δέχεται παραμέτρους x και y, επιστρέφει το γινόμενο τους.
`f2 <- function() { f1 }`	Η f2 απλώς επιστρέφει το f1 (αντικείμενο τύπου function).
`f2()(2, 20)`	Καλεί την f2 και επιστρέφει f1 που για x=2,y=4, επιστρέφει 8.

Η επιστρεφόμενη συνάρτηση μπορεί να είναι ορισμένη ως τοπική μεταβλητή[299]. Σε αυτή την περίπτωση, η αναζήτηση εξωτερικών αντικειμένων από την επιστρεφόμενη συνάρτηση θα ξεκινά πάντα από το πλαίσιο μέσα στο οποίο δημιουργήθηκε, δηλαδή από τις μεταβλητές της συνάρτησης που την περιέχει. Για περισσότερα βλ. §4.2.2.1 Περιβάλλοντα και συναρτήσεις. Ακολουθεί ένα παράδειγμα:

```
πρόσθεσε<-function(x1)
  {
  f<-function(x2) x1+x2
  return(f)
}
```

Η παραπάνω συνάρτηση δέχεται μια παράμετρο x1 και επιστέφει άλλη συνάρτηση που θα προσθέσει το x1 στη δική της παράμετρο x2. Οι παρακάτω δύο εντολές δημιουργούν μεταβλητές με όνομα π100 και π200 καλώντας τη συνάρτηση αυτή, άρα είναι συναρτήσεις που δέχονται μια παράμετρο (x2).

```
π100<-πρόσθεσε(100)
π200<-πρόσθεσε(200)
```

Το π100, συνάρτηση που δημιουργήθηκε μέσα στην κλήση πρόσθεσε(100) και θα προσθέτει 100 στο όποιο x1. Αντίστοιχα, το π200, που δημιουργήθηκε μέσα στην κλήση πρόσθεσε(200) θα προσθέτει 200 στο όποιο x1 του δοθεί:

```
> π100(10)
[1] 110
> π200(10)
[1] 210
```

Η δυνατότητα επιστροφής μιας εσωτερικής, τοπικά ορισμένης, συνάρτησης από άλλη, σε συνδυασμό με το ότι κάθε συνάρτηση αναζητά εξωτερικά αντικείμενα έχοντας εσωκλείσει και κρατήσει το πλαίσιο μέσα στο οποίο δημιουργήθηκε επιτρέπει τη δημιουργία συναρτήσεων κατασκευής άλλων, μια τεχνική που ονομάζεται function factories. Μετά από όλα αυτά, μπορείτε να δοκιμάσετε το παρακάτω ερώτημα-σπαζοκεφαλιά: η f()τελικά επιστρέφει 10, 20 ή 30;

```
f1 <- function() {
  x <- 20             # x τοπικό στο f1, πριν οριστεί η f2.
  f2 <- function(){x} # τοπική συνάρτηση f2, επιστρέφει x
  x <- 30             # νέα τιμή στο x τοπικό στο f1
  return(f2)          # η f1 επιστρέφει την τοπική f2
}

x <- 10               # εξωτερικό x (καθολικό, στο Global)
f <- f1()             # κλήση f1, ανάθεση f2 σε μεταβλητή f
f()                   # κλήση της f
```

[299] Να είναι δηλαδή μια εμφωλευμένη συνάρτηση (nested function).

Στη μεταβλητή f ανατέθηκε το αντικείμενο που επέστρεψε η κλήση της f1, δηλαδή η f2 (και το περιβάλλον της). Όταν επιστράφηκε η τιμή f2, η τιμή του x στην f1 (που περικλείει την f2 και μέσα στην οποία η f2 δημιουργήθηκε, άρα έχει ενσωματωθεί στην f2) είναι 30. Όταν τελικά καλείται η f2 (μέσω της f) θα αναζητήσει το x για να το επιστρέψει. Η αναζήτηση αυτή θα ξεκινήσει στο πλαίσιο που είχε η συνάρτηση όταν δημιουργήθηκε. Εκεί θα εντοπίσει το x με τιμή 30. Άρα το f() θα επιστρέψει 30. Αυτό ισχύει ακόμα και αν το f1 (γεννήτορας της f2) διαγραφεί ή αλλάξει:

```
> rm(f1)
> f()
[1] 30
```

Έτσι, η απάντηση είναι πως η f() θα επιστρέφει πάντα 30.

5.1.7 Αναδρομή (recursion)

Αναδρομή (recursion) υπάρχει όταν ο κώδικας στο σώμα μιας συνάρτησης καλεί (άμεσα ή έμμεσα) την ίδια τη συνάρτηση. Η τεχνική αυτή είναι ιδιαίτερα χρήσιμη στην υλοποίηση λύσεων κάποιων προβλημάτων. Συναρτήσεις που εφαρμόζουν αναδρομή συνήθως αφορούν την επίλυση ενός προβλήματος το οποίο απαιτεί να λυθούν πρώτα (και με τον ίδιο τρόπο) ένα ή περισσότερα «μικρότερα» προβλήματα.

Κάθε φορά που καλείται μια σωστά γραμμένη αναδρομική συνάρτηση, ο κώδικάς της εφαρμόζει κριτήρια με τα οποία αποφασίζει αν μπορεί να επιστρέψει κάποιο αποτέλεσμα χωρίς άλλες αναδρομικές κλήσεις ή αν θα συνεχίζει τις αναδρομικές κλήσεις καλώντας (έμμεσα ή άμεσα) πάλι την ίδια συνάρτηση. Αν δεν γινόταν αυτό, οι κλήσεις θα συνεχίζονται επ' άπειρον ή καλύτερα έως ότου εξαντληθεί ο διαθέσιμος χώρος αποθήκευσης της στοίβας κλήσεων[300]. Ακολουθεί ένα παράδειγμα τον επονομαζόμενο «τριγωνικό αριθμό» n δηλαδή το 1+2+3+…+n. Χάριν απλούστευσης το παράδειγμα θεωρεί πως η τιμή που θα δοθεί στην παράμετρο n θα είναι ένας μοναδικός ακέραιος και ελέγχει μόνο αν αυτός είναι μεγαλύτερος ή ίσος του 1 για να επιστρέψει αποτέλεσμα:

```
τριγωνικός <- function(n)
{
  if (n < 1)  stop("Λάθος") # στην περίπτωση n<1, σταματά.
  if (n == 1) return (1)          # για 1 ο τριγ. αριθ. είναι 1.
  return (τριγωνικός(n - 1)+n)    # για άλλο n, αναδρομή.
}
```

Η παραπάνω προσέγγιση βασίζεται στο ότι το τριγωνικός(n) είναι ίσο με το τριγωνικός(n-1)+n, π.χ. τριγωνικός(5) (ή 1+2+3+4+5) είναι ίσο με τριγωνικός(4)+5. Η προσέγγιση επίλυσης με αναδρομή για το συγκεκριμένο πρόβλημα είναι ίσως ανούσια. Αυτό συμβαίνει συχνά. Δεν είναι σπάνιο να μπορούν να υλοποιηθούν λύσεις για το ίδιο πρόβλημα με ή χωρίς εφαρμογή αναδρομής. Αλλά δεν ισχύει πάντα. Εδώ πάντως, παρόμοιο αποτέλεσμα μπορεί να επιτευχθεί με την εντολή sum(1:n) ή έστω με τη χρήση βρόχου και μιας μεταβλητής συσσώρευσης.

Ακολουθεί ένα ακόμη παράδειγμα, όπου γίνεται αναζήτηση της θέσης ενός αριθμού (x) μέσα σε κάποια δεδομένα (data). Ας θεωρήσουμε πως τα x και data που θα δίνονται είναι αριθμητικά (υποστηρίζουν σύγκριση τιμών) και τα data θα είναι ταξινομημένα σε αύξουσα σειρά καθώς εδώ θα εφαρμοστεί δυαδική αναζήτηση:

```
δυαδική_αναζήτηση <- function(x,data)
{
  if(length(x)!=1)    stop("Λάθος x")
  if(length(data)<1) stop("Λάθος δεδομένα")

  # η τοπική συνάρτηση που κάνει την αναζήτηση με αναδρομή:
  do_search <- function(x,data,lo,hi)
  {
    # Αν δεν βρέθηκε το x στα data:
    if(lo>hi) return (NULL)
```

[300] Όπως αναφέρθηκε παραπάνω, στη στοίβα κλήσεων (call stack) καταγράφεται η τρέχουσα κατάσταση όταν γίνεται κλήση μιας συνάρτησης ώστε να ανακληθεί όταν η συνάρτηση επιστρέψει.

```
        cat("Ψάχνω το",x,"στο [",data[lo:hi],"]\n")

        # το mid ειναι η μεσαία θέση στα data:
        mid <- (lo+hi)%/%2

        # ελέγχει αν βρέθηκε στη μέση:
        if(x == data[mid]) return ( mid )

        # αν όχι αναζητά στο ανάλογο μισό των data:
        if(x < data[mid]) return ( do_search (x,data,lo,mid-1) )
        if(x > data[mid]) return ( do_search (x,data,mid+1,hi) )
        return (mid)
      }

      # ξεκινά τη διαδικασία καλώντας την τοπική συνάρτηση:
      return ( do_search(x, data, 1, length(data)) )
    }
```

Ένα παράδειγμα κλήσης της συνάρτησης είναι το παρακάτω, στο οποίο η συνάρτηση επιστρέφει 7, δηλαδή τη θέση του 53 στα δοθέντα δεδομένα:

```
> δυαδική_αναζήτηση( 53, c(1,6,7,9,19,22,53,78) )
Ψάχνω το 53 στο [ 1 6 7 9 19 22 53 78 ]
Ψάχνω το 53 στο [ 19 22 53 78 ]
Ψάχνω το 53 στο [ 53 78 ]
[1] 7
```

Στον κώδικα του παραδείγματος αυτού ορίζεται μια συνάρτηση με όνομα δυαδική_αναζήτηση η οποία κάνει απλώς κάποιους ελέγχους και καλεί την τοπικά ορισμένη συνάρτηση με όνομα do_search. Αυτή η τοπικά ορισμένη συνάρτηση είναι που μας ενδιαφέρει καθώς εφαρμόζει αναδρομή. Η συνάρτηση do_search δέχεται το x (την αναζητούμενη τιμή), το data (τα δεδομένα στα οποία θα αναζητηθεί το x) και τα lo και hi που οριοθετούν την ελάχιστη και μέγιστη θέση στα data στις οποίες θα γίνει αναζήτηση του x. Αρχικά καλείται με ελάχιστο όριο τη θέση 1 και μέγιστο όριο το μήκος των data (length(data)), άρα για όλα τα δεδομένα. Η σχετική κλήση είναι η do_search(x, data, 1, length(data)). Αφού η do_search υπολογίσει τη θέση της μέσης (mid) των ορίων ελέγχει αν το x βρίσκεται εκεί. Αν όχι με νέο όριο τη μέση αναζητά είτε στο μισό με τις μικρότερες τιμές (θέσεις lo έως mid-1) είτε σε αυτό με τις μεγαλύτερες τιμές (θέσεις mid+1 έως hi) καλώντας (με αναδρομή) την ίδια συνάρτηση do_search αλλά με τα νέα, πιο περιορισμένα, όρια αναζήτησης.

5.1.8 Παραδείγματα

Κλείνουμε την ενότητα που αφορά τη δημιουργία συναρτήσεων παραθέτοντας μερικά ενδεικτικά παραδείγματα δημιουργίας και χρήσης απλών συναρτήσεων για διάφορους σκοπούς. Στα παραδείγματα αυτά εφαρμόζονται μερικά από όσα περιγράφηκαν παραπάνω. Προφανώς, τα διάφορα πακέτα της R ίσως παρέχουν συναρτήσεις παρεμφερείς κάποιων από τα παραδείγματα και οι οποίες είναι λειτουργικότερες και πιο πλήρεις από τα παραδείγματα αυτά. Παραθέτουμε τα παραδείγματα ως βοήθημα για τους αρχάριους δημιουργούς συναρτήσεων και όχι ως πρότυπο ή τέλειες λύσεις του όποιου προβλήματος αφορούν. Ο προγραμματισμός είναι μια δημιουργική διαδικασία και διαφορετικοί δημιουργοί κώδικα μπορεί να εφαρμόσουν άλλη προσέγγιση και να δημιουργήσουν διαφορετικό κώδικα για το ίδιο ζητούμενο αποτέλεσμα.

Παράδειγμα 1ο: μια συνάρτηση που μετατρέπει μοίρες σε ακτίνια. Οι βασικές τριγωνομετρικές συναρτήσεις που παρέχονται με την R (sin, cos, tan κλπ)[301] δέχονται ως παράμετρο γωνίες μετρημένες σε ακτίνια. Ζητούμενο είναι να γραφεί μια συνάρτηση που μετατρέπει τις μοίρες σε ακτίνια, βασισμένη στον τύπο: ακτίνια = μοίρες x π/180. Αν δεν δοθεί καμία τιμή (εδώ για την παράμετρο degrees), θεωρεί πως η γωνία είναι 0 μοιρών:

```
# Συνάρτηση που μετατρέπει μοίρες σε ακτίνια
radians <- function( degrees=0 ) degrees*pi/180
```

Με τη radians μπορούν να συνδυαστούν οι τριγωνομετρικές συναρτήσεις με γωνίες σε μοίρες, π.χ.:

---

[301] βλ. help(Trig).

```
> sin( radians( c(0,90,180,270) ) )
[1]  0.000000e+00  1.000000e+00  1.224606e-16 -1.000000e+00
```

Ακολουθεί μια παραλλαγή της συνάρτησης, η οποία δέχεται γωνίες σε μοίρες και λεπτά της μοίρας. Οι δημιουργοί του κώδικα αυτού επέλεξαν να ελέγχει αν οι παράμετροι που δίνονται είναι όντως αριθμοί (numeric). Σε αντίθετη περίπτωση, εγείρει σφάλμα. Σφάλμα θα είχε προκύψει ούτως ή άλλως από τις αριθμητικές πράξεις που ακολουθούν αν ο τύπος κάποιου από τα ορίσματα δεν τις υποστήριζε, αλλά η stopifnot παράγει μήνυμα που περιγράφει το πρόβλημα λίγο πιο αναλυτικά.

```
# Συνάρτηση που μετατρέπει μοίρες και λεπτά σε ακτίνια
radians <- function(degrees = 0, minutes = 0)
{
  stopifnot(is.numeric(degrees), is.numeric(minutes))
  return((degrees + (minutes / 60)) * pi / 180)
}
```

Η τελευταία γραμμή στην παραπάνω συνάρτηση θα μπορούσε να γραφεί και χωρίς τη λέξη return επιστρέφοντας πάλι το ίδιο αποτέλεσμα (αφού ο υπολογισμός θα ήταν η τελευταία εντολή που εκτελείται). Παρόλα αυτά, η χρήση της λέξης return ακόμα και στο τέλος της συνάρτησης (όπου είναι προαιρετική) δίνει έμφαση στο αποτέλεσμα που θα επιστραφεί. Η δεύτερη από τις παρακάτω δοκιμαστικές κλήσεις της radians δείχνει και τη χρήση ανακύκλωσης τιμών για την παραγωγή του επιστρεφόμενου διανύσματος:

```
> sin( radians(90) )
[1] 1
> sin( radians(90, c(15,30,45,60)) )
[1] 0.9999905 0.9999619 0.9999143 0.9998477
```

Τέλος, μια τρίτη παραλλαγή της συνάρτησης, δέχεται ένα διάνυσμα που περιέχει γωνίες σε μοίρες και λεπτά της μοίρας. Ο κώδικάς αυτός ελέγχει αν οι παράμετροι που δίνονται είναι όντως αριθμοί (numeric) και αν η παράμετρος deg_min περιέχει αριθμό στοιχείων πολλαπλάσιο του 2 (μια τιμή για τις μοίρες, μια για τα λεπτά). Αν δεν ισχύουν αυτά, εγείρει σφάλμα. Οι δύο μεταβλητές degrees και minutes είναι τοπικές:

```
# Συνάρτηση που μετατρέπει μοίρες και λεπτά σε ακτίνια
radians <- function(deg_min)
{
  if(!is.numeric(deg_min)) stop("Data must be numeric")
  if(length(deg_min)%%2!=0)stop("Data not in pairs")

  degrees <- deg_min[c(T,F)]
  minutes <- deg_min[c(F,T)]
  return((degrees + (minutes / 60)) * pi / 180)
}
```

Μία δοκιμαστική κλήση της συνάρτησης:

```
> sin( radians(c(90,0,90,15,90,30,90,45,90,60)) )
[1] 1.0000000 0.9999905 0.9999619 0.9999143 0.9998477
```

Παράδειγμα 2^{ο}: επίλυση πρωτοβάθμιας εξίσωσης ($\beta x + \alpha = 0$). Παραθέτουμε το παράδειγμα αυτό καθώς είναι ένα κλασικό παράδειγμα που δίνεται σε νεοεισερχόμενους στον προγραμματισμό[302]. Ζητούμενο είναι να γραφτεί μια συνάρτηση που θα επιστρέφει τις τιμές του x που ικανοποιούν τη $\beta x + \alpha = 0$ αν είναι γνωστά τα α και β. Χάριν απλούστευσης, τα α και β που υποστηρίζονται πρέπει να είναι απλοί αριθμοί (όχι πολυμελή αντικείμενα) οπότε γίνεται σχετικός έλεγχος (με την πρώτη if) και έγερση σφάλματος (με stop) αν ο έλεγχος αποτύχει:

```
# Συνάρτηση που λύνει την 1βαθμια εξίσωση bx+a=0 (ή bx=-a)
# Επιστρέφει NULL αν η εξίσωση αποδειχτεί αδύνατη.
# Επιστρέφει Inf αν η εξίσωση είναι αόριστη.
# Αλλιώς επιστρέφει τη λύση (τιμή του x που την ικανοποιεί)

firstdegree <- function(b = 0, a = 0)
```

[302] Η συνάρτηση solve(α,b) του πακέτου 'base' λύνει συστήματα πρωτοβάθμιων εξισώσεων. Για μια μόνο εξίσωση, π.χ. 2x-20=0, η solve(2,-20) επιστρέφει -10.

```
{
  if (length(b) != 1 || length(a) != 1)
    stop("equation parameters must be single numbers")

  if (b != 0)
    return(-a / b)  # Η συνάρτηση επιστρέφει τη λύση.

  # (Αν η εκτέλεση φτάσει εδώ, το b είναι 0)
  if (a == 0)
    return (Inf)    # Αόριστη, άπειρες λύσεις (περίπτωση 0x+0=0)

  # (Αν η εκτέλεση φτάσει εδώ, b και α είναι 0)
  return (NULL)     # Αδύνατη (b=0 και a όχι 0, π.χ. 0x+3=0
}
```

Δοκιμαστική κλήση για το 2x-20=0, επιστρέφει x=10:

```
> firstdegree(2,-20)
[1] 10
```

Παράδειγμα 3ο: παιχνίδι «μάντεψε τον αριθμό!». Ζητούμενο είναι να γραφτεί μια συνάρτηση που θα επιτρέπει στον χρήστη της να παίξει ένα παιχνίδι. Στο παιχνίδι αυτό ζητείται από τον χρήστη (παίκτη) να μαντέψει σε συγκεκριμένο αριθμό προσπαθειών έναν ακέραιο αριθμό που έχει επιλεγεί τυχαία. Η παρακάτω υλοποίηση βασίζεται σε δύο συναρτήσεις, τη readinteger που διαβάζει αριθμούς από το πληκτρολόγιο και την κύρια συνάρτηση του παιχνιδιού, με όνομα guessgame:

```
# Συνάρτηση που διαβάζει ακέραιο αριθμό από το πληκτρολόγιο.

readinteger <- function(prompt="Δώσε ακέραιο αριθμό:")
{
  n <- readline(prompt)
  if(!grepl("^[0-9]+$",n))return(readinteger(prompt))
  return(as.integer(n))
}

# Συνάρτηση παιχνίδι: μάντεψε τον αριθμό!

guessgame <- function (max_number = 100,
                       max_attempts = 10)
  {
  max_number   <- as.integer(max_number[1])
  max_attempts <- as.integer(max_attempts[1])

  a = readline("SHALL WE PLAY A GAME? (Ναι/Όχι):")
  a = tolower(a)
  if (a != "ναι" && a != "ναί" && a != "ν")
  {
    cat("Όχι; Ίσως κάποια άλλη φορά τότε.\n")
    return(invisible(NULL))
  }

  cat("\014Μάντεψε αριθμό από το 1 έως το", max_number,  ".\n")

  hidden_number <- sample(1:max_number, 1)
  for (i in 1:max_attempts)
  {
    cat("Προσπάθεια #", i, "έχεις άλλες", max_attempts - i)
    guess <- readinteger("Μάντεψε:")
```

```
    if (guess == hidden_number)
    {
      cat("Συγχαρητήρια το βρήκες σε", i, "προσπάθειες!\n")
      return(invisible(TRUE))
    }

    if (guess < hidden_number)
      cat("To", guess, "είναι πολύ μικρό!\n")

    if (guess > hidden_number)
      cat("To", guess, "είναι πολύ μεγάλό!\n")
  }

  cat("Έχασες, ο αριθμός ήταν", hidden_number, "\n\n")
  return(invisible(FALSE))
}
```

Ας εξηγήσουμε τις επιλογές που έγιναν στον παραπάνω κώδικα. Το παιχνίδι πρέπει να διαβάζει από το πληκτρολόγιο τους ακέραιους αριθμούς που καταχωρεί ο παίχτης. Για να απλοποιηθεί ο κυρίως κώδικας της συνάρτησης του παιχνιδιού, χρησιμοποιήθηκε μια δεύτερη συνάρτηση, η readinteger που κάνει ακριβώς αυτό. Αν και υπάρχουν απλούστεροι τρόποι να διαβαστεί ένας ακέραιος αριθμός από μια μονάδα εισόδου[303], η συγκεκριμένη συνάρτηση χρησιμοποιεί την grepl[304] για να ελέγξει ότι η καταχώρηση του χρήστη αποτελείται από αριθμητικά ψηφία και αν αυτό ισχύει μετατρέπει το συγκεκριμένο κείμενο σε ακέραιο και το επιστρέφει. Επίσης λειτουργεί με αναδρομή[305]: αν το κείμενο που καταχωρήθηκε δεν αποτελείται από αριθμητικά ψηφία, καλεί πάλι τη readinteger.

Για την guessgame, η συνάρτηση δέχεται ως παραμέτρους τα όρια τιμής για τον κρυφό αριθμό και τις προσπάθειες. Η διαδραστική φύση της συνάρτησης δεν αξιοποιεί πολυμελή αντικείμενα και έτσι βασίζεται σε βρόχους. Για τον λόγο αυτό οι δημιουργοί της συνάρτησης επέλεξαν να συνεχίζει η εκτέλεσή της ασχέτως τύπου αντικειμένου που ίσως δοθεί στις δυο παραμέτρους της, αρκεί αυτό να μπορεί να μετατραπεί σε integer και κρατώντας μόνο το 1ο στοιχείο του. Αφού ο παίχτης ερωτηθεί και δεχτεί να παίξει[306], επιλέγεται τυχαία ένας αριθμός με τη συνάρτηση sample[307] και ξεκινά ένας βρόχος for[308] όπου ο παίχτης καταχωρεί τον αριθμό που μάντεψε (guess) και αυτός συγκρίνεται με τον κρυφό (hidden_number). Οι επαναλήψεις διακόπτονται (και η συνάρτηση επιστρέφει TRUE) αν η τιμή που μάντεψε ο παίχτης βρεθεί ίση με τον κρυφό αριθμό, αλλιώς συνεχίζονται έως η μεταβλητή επαναλήψεων (i) ξεπεράσει το όριο (max_attempts), οπότε ο παίχτης χάνει και η συνάρτηση επιστρέφει FALSE. Μια δοκιμαστική κλήση της guessgame καταγράφεται παρακάτω:

```
> guessgame(10,5)
SHALL WE PLAY A GAME? (Ναι/Όχι): ναι

Μάντεψε αριθμό από το 1 έως το 10 .
Προσπάθεια # 1 έχεις άλλες 4
Μάντεψε:3
To 3 είναι πολύ μικρό!
Προσπάθεια # 2 έχεις άλλες 3
Μάντεψε:6
To 6 είναι πολύ μεγάλό!
Προσπάθεια # 3 έχεις άλλες 2
Μάντεψε:4
Συγχαρητήρια το βρήκες σε 3 προσπάθειες!
```

Παράδειγμα 4ο: Ισορροπία της αγοράς, προσφορά και ζήτηση προϊόντος σε σχέση με ποσότητα (βασισμένο σε πηγή [10]). Αφού οριστούν οι συναρτήσεις προσφοράς (demand) και ζήτησης (supply) που

303 βλ.§2.3.4 Είσοδος κειμένου.
304 βλ.§2.3.3.2 Κανονικές εκφράσεις (regular expressions).
305 βλ.§5.1.7 Αναδρομή (recursion).
306 Η ερώτηση είναι δανεισμένη από παλαιά κινηματογραφική ταινία.
307 βλ.§2.6 Τυχαίοι αριθμοί.
308 βλ.§3.2.2.3 Επαναλήψεις (βρόχοι ή loop).

υπολογίζουν την αναμενόμενη ποσότητα ενός προϊόντος ανάλογα με την τιμή του (p), καθώς και η surplus (η διαφορά της προσφοράς από τη ζήτηση), θα χρησιμοποιήσουμε μερικές συναρτήσεις του πακέτου 'graphics'[309] για να εμφανίσουμε τα αποτελέσματα και την **uniroot** του πακέτου 'stats' για να βρούμε το σημείο ισορροπίας της αγοράς, την τιμή δηλαδή για το προϊόν στην οποία η προσφορά είναι ίση με τη ζήτηση:

```
demand  <- function(p) 3000000-3*p^2
supply  <- function(p) 1000000+5*p^1.8
surplus <- function(p) supply(p)-demand(p)

price <- 1:1000
plot(price,demand(price),col="green",type = "l")
lines(price,supply(price),col="blue")

e <- uniroot(surplus,c(0,1000))$root
cat("Τιμή ισορροπίας = ",e)
```

Παράδειγμα 5ο: μια συνάρτηση που θα γράφεται ανάμεσα στις δύο παραμέτρους της (infix) της και θα καλεί τη δεύτερη παράμετρό της ως συνάρτηση με όρισμα την πρώτη (ο λόγος εξηγείται παρακάτω). Για να οριστεί μια συνάρτηση που θα καλείται με σύνταξη infix[310] πρέπει το όνομα της να αρχίζει και να τελειώνει με τον χαρακτήρα %, έτσι η συνάρτηση ονομάστηκε %_και_στείλε_το_στο_%:

```
# η παρακάτω συνάρτηση λειτουργεί ως ένα απλοϊκό pipe
"%_και_στείλε_το_στο_%" <- function(d,f) f(d)
```

Έχουν ήδη αναφερθεί αρκετά παραδείγματα κλήσης μιας συνάρτησης με όρισμα μια άλλη[311]. Το καινούργιο εδώ είναι ο συνδυασμός αυτής της τεχνικής με την infix γραφή, δηλαδή με τη δυνατότητα να γίνεται κλήση της συνάρτησης με το όνομά της γραμμένο ανάμεσα στις παραμέτρους τις. Το αριστερό μέρος θα είναι η 1η παράμετρος και θα χρησιμοποιείται ως παράμετρος για το δεξί. Και προφανώς, αυτό μπορεί να συνεχίζεται οδηγώντας σε μια αλυσίδα από κλήσεις συναρτήσεων που τροφοδοτούν η μία την άλλη.

```
x<-c(9,4,1,9,16,9)
> x %_και_στείλε_το_στο_% sqrt
[1] 3 2 1 3 4 3
> (1/x) %_και_στείλε_το_στο_% sqrt %_και_στείλε_το_στο_% sort
[1] 0.2500000 0.3333333 0.3333333 0.3333333 0.5000000 1.0000000
```

Το τελευταίο παράδειγμα είναι ισοδύναμο της εντολής sort(sqrt(1/x)) αλλά προσέξτε την αλλαγή στη γραφή: «[αποτίμησε] το 1 δια x και [το αποτέλεσμα] στείλε το στο sqrt και [το αποτέλεσμα] στείλε το στο sort». Η συνάρτηση %_και_στείλε_το_στο_% υλοποιεί κάτι που ονομάζεται pipe (σωλήνωση). Βέβαια η υλοποίηση αυτή είναι ιδιαίτερα απλοϊκή και απαιτεί το δεξί μέρος να είναι μια συνάρτηση που θα λειτουργεί με μια μόνο παράμετρο (το αριστερό μέρος). Αλλά στην R υπάρχουν διαθέσιμες συναρτήσεις pipe που δεν έχουν αυτούς τους περιορισμούς, χρησιμοποιούνται ευρέως και περιγράφονται στην §5.2 Σωληνώσεις (pipe), παρακάτω.

5.2 Σωληνώσεις (pipe)

Στην R οι συναρτήσεις επιστρέφουν πάντα κάποιο αντικείμενο. Συχνά το αντικείμενο αυτό τροφοδοτείται σε κάποια άλλη συνάρτηση ως μέρος κάποιας πιο σύνθετης εντολής. Για έναν τέτοιο συνδυασμό συναρτήσεων μπορεί να χρησιμοποιηθεί κάποια προσωρινή μεταβλητή (για να αποθηκεύει τα ενδιάμεσα αποτελέσματα) ή να συνδυαστούν οι κλήσεις των συναρτήσεων. Συνήθως επιλέγεται το δεύτερο[312], οπότε η συνήθης σύνταξη μιας τέτοιας εντολής θα ήταν:

$\alpha\ (\ \beta\ (x)\ )$

όπου α και β κάποιες συναρτήσεις και x κάποιο όρισμα. Για παράδειγμα, αν πρέπει να τυπωθεί η τετραγωνική ρίζα κάποιου x η εντολή δίνεται ως:

```
print ( sqrt ( x ) )
```

Το αποτέλεσμα της μιας συνάρτησης (sqrt) περνά ως τιμή της 1ης παραμέτρου της άλλης (print). Τέτοια σύνταξη εντολών έχει γίνει σε πολλά, από τα έως τώρα, παραδείγματα του βιβλίου, καθώς απαιτείται συχνά.

---

[309] βλ.§9.2.2 Πακέτο 'graphics'.
[310] βλ.§5.1.1 Δημιουργία συναρτήσεων.
[311] βλ.§5.1.6 Συναρτήσεις ως ορίσματα και επιστρεφόμενες τιμές.
[312] Αυτή η προσέγγιση επιβάλλεται και από το συναρτησιακό προγραμματιστικό μοντέλο.

Σε πολλές μάλιστα περιπτώσεις οι εμπλεκόμενες συναρτήσεις είναι πάνω από δύο, καθεμία με τις δικές της πρόσθετες παραμέτρους, μια κατάσταση που οδηγεί σε δυσανάγνωστο κώδικα. Τα pipes, είναι ένας εναλλακτικός τρόπος σύνταξης τέτοιων συνδυασμένων κλήσεων συναρτήσεων που περιγράφει τα βήματα με τη σειρά που θα εκτελεστούν και τροφοδοτούν την επόμενη συνάρτηση, δηλαδή κάτι αντίστοιχο με:

$x \rightarrow \beta \rightarrow \alpha$

Το παραπάνω θα μπορούσε να περιγράφει έναν μηχανισμό που εκτελεί μια αλυσίδα από εντολές στέλνοντας τα αποτελέσματα από το ένα κομμάτι κώδικα (π.χ. κλήση μίας συνάρτησης), στο επόμενο. Με ένα τέτοιο τρόπο γραφής είναι πιο προφανές ότι ζητούμενο εδώ είναι να ανακληθεί η τιμή του x, μετά να τροφοδοτηθεί στη β και το αποτέλεσμα να τροφοδοτηθεί στην α. Αν (όπως εδώ) η τροφοδότηση γίνεται από αριστερά προς τα δεξιά, αυτό ονομάζεται forward pipe. Στην R έχουν επικρατήσει δύο τρόποι σύνταξης ενός forward pipe. Ο ένας τρόπος σύνταξης pipe είναι με τον τελεστή |> που είναι ενσωματωμένος[313] στη γλώσσα R. Ακολουθεί το ισοδύναμο της εντολής του παραπάνω παραδείγματος, γραμμένο με χρήση του τελεστή αυτού:

```
x |> sqrt() |> print()
```

Έτσι, η σύνταξη με pipe μπορεί να οδηγήσει σε εντολές που είναι πιο ευανάγνωστες και λιγότερο επιρρεπείς σε προγραμματιστικά λάθη. Αυτό ισχύει ιδιαίτερα όταν η εντολή αποτελείται από συνδυασμό πολλών συναρτήσεων που πιθανώς έχουν και πρόσθετες παραμέτρους αλλά εμπλέκεται κυρίως ένα αντικείμενο (όπως εδώ το x) πάνω στο οποίο γίνονται διαδοχικά βήματα επεξεργασίας. Επεκτείνοντας το παραπάνω παράδειγμα, η εντολή που ακολουθεί (και είναι γραμμένη χωρίς χρήση pipe) ζητά να υπολογιστεί ο λογάριθμος (με βάση 10) κάποιου x, να ταξινομηθούν τα αποτελέσματα σε φθίνουσα σειρά και να τυπωθούν:

```
print( sort( log(x,10), decreasing = T) )
```

Η εντολή αυτή είναι ισοδύναμη με την παρακάτω, που χρησιμοποιεί τον τελεστή |> για pipe:

```
x |> log(10) |> sort(decreasing = T) |> print()
```

Αν επιπρόσθετο ζητούμενο είναι να γίνει καταχώρηση του αποτελέσματος[314] σε κάποια μεταβλητή y, η εντολή θα μπορούσε να γραφεί με οποιονδήποτε από τους παρακάτω δύο τρόπους:

```
x |> log(10) |> sort(decreasing = T) |> print() -> y
y <- x |> log(10) |> sort(decreasing = T) |> print()
```

Άλλος συνήθης τρόπος να γραφεί ένα pipe ορίζεται στο πακέτο ‘magrittr’ [64]. Το συγκεκριμένο πακέτο είναι διαθέσιμο στο CRAN[315] και αποτελεί μέρος της συλλογής πακέτων ‘tidyverse’ [57]. Ο βασικός τελεστής για forward pipe του ‘magritt’ είναι το %>%. Ο τελεστής αυτός προηγήθηκε χρονικά του αντίστοιχου τελεστή |> που ενσωματώνει η R και συνοδεύεται από παραλλαγές από άλλους σχετικούς τελεστές με διάφορες δυνατότητες. Αυτό, σε συνδυασμό με την υιοθέτησή του από άλλα πακέτα του ‘tidyverse’, έχει κάνει τη χρήση του %>% του πακέτου ’magrittr’ ιδιαίτερα διαδεδομένη. Ακολουθεί παρακάτω το ίδιο παράδειγμα γραμμένο με χρήση του τελεστή %>%. Παρατηρήστε πως εδώ δεν χρειάζονται παρενθέσεις σε συναρτήσεις χωρίς πρόσθετα ορίσματα (όπως η print στο παράδειγμα αυτό):

```
y <- x %>% log(10) %>%  sort(decreasing = T) %>%  print
```

Για την κάλυψη διαφόρων αναγκών κατά τη δημιουργία μιας αλυσίδας εντολών με pipe, το πακέτο ‘magrittr’ ορίζει και άλλους χρήσιμους τελεστές[316], όπως οι %T>% και %<>%. Μερικά παραδείγματα χρήσης των τελεστών του πακέτου ‘magrittr’:

Δοκιμάστε:	**Σχόλιο**
`3 %>% sin`	Ισοδύναμο με sin(3).
`3 %>% log(10)`	Ισοδύναμο με log(3,10).
`x %<>% sin`	Ισοδύναμο με x <- sin (x). Το %<>% κάνει ανάθεση του αποτελέσματος στο αριστερό μέρος.
`x %T>% sin`	Ισοδύναμο με {sin(x);x}. Το %>% εκτελεί το pipe αλλά επιστρέφει το αριστερό μέρος.

Επιπροσθέτως το πακέτο ‘magrittr’ παρέχει τον τελεστή %$% που εκθέτει στη δεξιά πλευρά τα ονόματα του αντικειμένου που βρίσκεται στην αριστερή. Εφαρμόζεται σε αντικείμενα που υποστηρίζουν το $ (list, data.frame κλπ.)[317].

---

[313] Περιέχεται στο πακέτο ΄base’. Το όνομα του τελεστή παραπέμπει στον αντίστοιχο τελεστή της γλώσσας F# για διαδοχική κλήση συναρτήσεων.

[314] Πρόκειται για το αποτέλεσμα που επιστρέφει η τελευταία συνάρτηση που καλείται (εδώ η print).

[315] Προφανώς για να χρησιμοποιηθούν οι τελεστές του πακέτου πρέπει αυτό να έχει ήδη εγκατασταθεί στο σύστημά και να συνδεθεί με την R (βλ.§1.5 Χρήση και διαχείριση πακέτων).

[316] βλ. help(magrittr)

[317] βλ.§4.2.1.1 Ο τελεστής επιλογής $.

Κατά τη διάρκεια της κλήσης της εντολής με %>% μια προσωρινή κρυφή μεταβλητή με όνομα '.' αποθηκεύει το αποτέλεσμα της αποτίμησης του κώδικα στηναριστερή πλευρά. Για παράδειγμα, ας θεωρήσουμε την παρακάτω εντολή που υπολογίζει με pipe το αντίστοιχο του sin(log(sqrt(10000), 10)):

```
10000 %>% sqrt %>% log(10) %>% sin
```

Για κάθε %>% το . έχει το αποτέλεσμα που τροφοδοτείται δεξιά. Ας το εμφανίσουμε:

```
> 10000 %>% sqrt %>% print(.) %>% log(10) %>% print(.) %>% sin
[1] 100
[1] 2
[1] 0.9092974
```

Το 100 είναι το αποτέλεσμα του sqrt(10000), το 2 αποτέλεσμα του log(100,10) ενώ η τελευταία τιμή (που εμφανίζεται αυτόματα αν εκτελεστεί το παραπάνω στο Console) είναι το αποτέλεσμα του sin(2). Η κρυφή αυτή μεταβλητή είναι ιδιαίτερα χρήσιμη όταν απαιτείται κάποια ιδιαίτερη λειτουργία από το τρέχον αποτέλεσμα στο pipe, όπως, π.χ. χρήση του ως δείκτη στοιχείων σε κάποιο άλλο αντικείμενο. Στο παρακάτω, έχουμε το ισοδύναμο της εντολής sqrt(x[1:5]) σε κάποιο x. Παρατηρήστε τη χρήση του . για να δείξει που πάει η τρέχουσα τιμή (το αντικείμενο αριστερά του πρώτου %>%, δηλαδή το 1:5):

```
1:5 %>% x[.] %>% sqrt
```

Ή μπορεί να γίνει το παρακάτω, ισοδύναμο του c("α","β","γ")[2:3], επιστρέφοντας c("β","γ"):

```
c("α","β","γ") %>% .[2:3]
```

Ενδιαφέροντα παραδείγματα χρήσης των τελεστών pipe του πακέτου 'magrittr' υπάρχουν (μεταξύ άλλων) στο [65].Τέλος, στο RStudio ο τελεστής pipe μπορεί να εισαχθεί μέσω συντόμευσης συνδυασμού πλήκτρων από το πληκτρολόγιο (η προεπιλεγμένη επιλογή είναι πατώντας τα πλήκτρα Ctrl + Shift + M) ενώ στο CRAN υπάρχουν διάφορα πακέτα που σχετίζονται με τη δημιουργία pipe.

5.3 Συναρτησιακός προγραμματισμός

Η ενότητα αυτή συνοψίζει το προγραμματιστικό μοντέλο (ή παράδειγμα, paradigm) που ονομάζεται συναρτησιακός προγραμματισμός (functional programming) και επανειλημμένα αναφέρθηκε σε διάφορα σημεία του βιβλίου. Προγραμματιστικά παραδείγματα όπως αυτό είναι προσεγγίσεις που αφορούν τον διαμερισμό ενός μεγαλύτερου προγραμματιστικού προβλήματος σε μικρότερα. Ένα άλλο προγραμματιστικό παράδειγμα (το οποίο θα μας απασχολήσει σε άλλη ενότητα) είναι ο αντικειμενοστραφής προγραμματισμός[318]. Στον συναρτησιακό προγραμματισμό, η οδηγία για τον κώδικα είναι να παράγει το αποτέλεσμα (έξοδο) μόνο μέσα από μια αλυσίδα κλήσεων αυτόνομων συναρτήσεων που καθεμία είναι ένα κλειστό σύστημα[319], να μετασχηματίζει την είσοδό της εκτελώντας κάποια - κατά προτίμηση βασική - λειτουργία και να τροφοδοτεί με την έξοδό της την επόμενη. Αν το συναρτησιακό μοντέλο εφαρμοστεί τέλεια, δεν χρειάζονται μεταβλητές.

Η R κατά βάση είναι συναρτησιακή γλώσσα προγραμματισμού, βασίζεται στο μοντέλο αυτό και το υιοθετεί. Στο [62] ο Chambers (ένας από τους βασικούς δημιουργούς της γλώσσας) αναλύει την εφαρμογή του συναρτησιακού προγραμματιστικού μοντέλου στην R. Εκεί, καταγράφεται και ένας ορισμός βασικών αρχών του συναρτησιακού προγραμματισμού ως εξής: (α) όλη η εκτέλεση κώδικα είναι αποτέλεσμα κλήσεων συναρτήσεων άρα ο προγραμματισμός ανάγεται σε ορισμό συναρτήσεων, (β) κάθε σωστά ορισμένη συνάρτηση επιστρέφει την ίδια τιμή για τις ίδιες τιμές στις παραμέτρους της άρα η επιστρεφόμενη τιμή εξαρτάται μόνο από τις τιμές των παραμέτρων και (γ) το αποτέλεσμα από την κλήση της συνάρτησης είναι μόνο η τιμή που επιστρέφει, πέραν αυτού η συνάρτηση δεν αλλάζει τίποτα το οποίο θα έχει επιπτώσεις στους υπολογισμούς που θα ακολουθήσουν.

Όλα τα παραπάνω βοηθούν τη δημιουργία κώδικα (υπό μορφή συναρτήσεων) που είναι προβλέψιμος και αξιόπιστος, μειώνει την πιθανότητα λαθών, επιτρέπει την εφαρμογή διαφόρων βελτιστοποιήσεων καθώς και την αξιοποίηση της παράλληλης επεξεργασίας για την εκτέλεση. Τα στοιχεία αυτά έχουν ενισχύσει ιδιαίτερα

[318] βλ.§6.2 Αντικειμενοστραφής προγραμματισμός στην R.

[319] Υπάρχουν δύο ορισμοί για το κλειστό σύστημα. Η μία απαιτεί ένα τέτοιο σύστημα να μην αλληλοεπιδρά καθόλου με οτιδήποτε εκτός του συστήματος. Η δεύτερη, που χρησιμοποιείται και εδώ, ονομάζει κλειστό ένα σύστημα που αλληλεπιδρά μεν, αλλά αυτό γίνεται μόνο με συγκεκριμένο, αυστηρά καθορισμένο τρόπο. Κάθε είσοδος στο σύστημα αυτό γίνεται μέσω κάποιας προκαθορισμένης τυπικής διαδικασίας εισόδου, η έξοδος από το σύστημα γίνεται μέσω προκαθορισμένης τυπικής διαδικασίας εξόδου. Εκτός αυτών, το σύστημα δεν επηρεάζεται, επηρεάζει ή αλληλεπιδρά με οτιδήποτε εκτός του συστήματος. Η λειτουργία ενός τέτοιου συστήματος παράγει μόνο την έξοδο, χωρίς να δημιουργεί άλλες επιρροές ή παρενέργειες (side effect) στο εξωτερικό του περιβάλλον.

την τάση προς υιοθέτηση του συναρτησιακού προγραμματιστικού παραδείγματος και διευρύνει τη χρήση γλωσσών προγραμματισμού που βασίζονται σε αυτό.

Το συντακτικό της R προτρέπει τους δημιουργούς κώδικα στην εφαρμογή του συναρτησιακού προγραμματιστικού παραδείγματος, χωρίς όμως να την επιβάλει. Τα παραπάνω (α), (β) και (γ) χαρακτηριστικά του συναρτησιακού προγραμματισμού παραπέμπουν σε κώδικα χωρίς τη χρήση εξωτερικών ή καθολικών μεταβλητών, όπου το αποτέλεσμα προκύπτει από αλληλουχίες κλήσεων συναρτήσεων που ανταλλάσσουν μεταξύ τους τιμές μέσω παραμέτρων και επιστροφών. Στην R όλη η εκτέλεση του προγράμματος είναι όντως αποτέλεσμα κλήσεων συναρτήσεων, άρα το προαναφερθέν χαρακτηριστικό (α) ικανοποιείται. Σχετικά με τα (β) και (γ), τέτοιες συναρτήσεις που δεν επηρεάζουν με παράπλευρο τρόπο το περιβάλλον εκτέλεσης (δεν έχουν παρενέργειες, side effect) και έχουν έξοδο που εξαρτάται μόνο από τις παραμέτρους τους ονομάζονται αμιγείς (pure). Για να διευρυνθούν οι δυνατότητες προγραμματισμού με χρήση μόνο pure συναρτήσεων πρέπει να επιτρέπεται σε μια συνάρτηση να δέχεται μέσω των παραμέτρων της άλλες συναρτήσεις αλλά και να μπορεί να έχει επιστρεφόμενη τιμή που θα είναι συνάρτηση. Στην R, αυτά ισχύουν, καθώς οι συναρτήσεις είναι αντικείμενα πρώτης τάξης (first-class objects) όπως οι υπόλοιποι τύποι αντικειμένων[320]. Όμως στην R οι συναρτήσεις δεν είναι απαραίτητα pure, εκτός αν οι δημιουργοί τους το επιλέξουν. Η πρόσβαση σε τιμές μεταβλητών που υπάρχουν εκτός μίας συνάρτησης επιτρέπεται, άρα η έξοδος της συνάρτησης δεν είναι υποχρεωτικό να εξαρτάται μόνο από τις τιμές των παραμέτρων, μπορεί να επηρεαστεί από τις τιμές εξωτερικών μεταβλητών κατά την ώρα της κλήσης της. Για να τηρηθεί το (β) πρέπει οι δημιουργοί της συνάρτησης να επιλέξουν να μην κάνουν χρήση τιμών από μεταβλητές που έχουν οριστεί εκτός της συνάρτησης και να χρησιμοποιούν μόνο τις τιμές των παραμέτρων της. Τέλος, για το (γ), οι συνήθεις τελεστές ανάθεσης τιμών σε μεταβλητές δεν επιτρέπουν αλλαγές σε μεταβλητές εκτός της συνάρτησης, άρα το (γ) ικανοποιείται, αλλά αυτό ισχύει μόνο εφόσον χρησιμοποιούνται αποκλειστικά οι τελεστές αυτοί[321]. Ακόμα και με αυτές τις επιλογές από τους δημιουργούς μιας συνάρτησης δεν είναι απαραίτητο πως η συνάρτηση είναι pure (με την αυστηρή έννοια του ορισμού αυτού) αφού πολύ συχνά για λόγους λειτουργικότητας και επίτευξης αποτελεσμάτων μια συνάρτηση θα αλληλοεπιδρά με τον χρήστη, θα ανταλλάσσει τιμές με το υπολογιστικό σύστημα στο οποίο εκτελείται ο κώδικας (π.χ. για την παραγωγή τυχαίων αριθμών), θα διαβάζει ή θα γράφει αρχεία κλπ., ενέργειες που μπορεί να έχουν επιπτώσεις στην εκτέλεση του κώδικα που θα ακολουθήσει ακόμα και αν δεν αλλάζουν άμεσα τις τιμές εξωτερικών μεταβλητών. Περισσότερα για τα ζητήματα αυτά παρουσιάζονται στο [33].

Παρόλα αυτά, η στόχευση για υλοποίηση κάθε νέας συνάρτησης με τρόπους που θα ακολουθούν κατά το δυνατόν το pure πρότυπο (δηλαδή χωρίς αλληλεπίδραση με το περιβάλλον πέραν των παραμέτρων και της επιστρεφόμενης τιμής της) είναι σωστή στρατηγική, καθώς αποσαφηνίζει τη λειτουργία της (διευκολύνοντας έτσι και τον εντοπισμό πιθανών λαθών), κάνει τη συνάρτηση περισσότερο προβλέψιμη, εξαλείφει την πιθανότητα λαθών που προκύπτουν όταν υπάρχει εξάρτηση από εξωτερικές μεταβλητές, επιτρέπει τον καλύτερο έλεγχο των εισόδων προς τη συνάρτηση και της εξόδου από αυτή, επιτρέπει την εφαρμογή διαφόρων βελτιστοποιήσεων απόδοσης κατά την εκτέλεσή του κώδικα ή την εκτέλεσή του με παράλληλη επεξεργασία, επιτρέπει την καλύτερη διαχείριση του συνόλου του κώδικα σε περίπτωση που η συνάρτηση αποτελεί μέρος μιας μεγαλύτερης λύσης λογισμικού κ.α.

5.4 Η συναρτησιακή προσέγγιση στον πραγματικό κόσμο

Αναφέρθηκαν ήδη μερικά από τα πλεονεκτήματα του συναρτησιακού προγραμματιστικού μοντέλου. Το μοντέλο οδηγεί σε υλοποίηση λύσεων με ελάχιστη (ή χωρίς) ανάγκη χρήσης μεταβλητών και αναδεικνύει ακόμα περισσότερο τη συνάρτηση ως βασική, ολοκληρωμένη λειτουργική μονάδα, μέσα στην οποία πρέπει να υπάρχει πλήρης, αυτόνομη υλοποίηση κάποιας λειτουργίας αλλά και να αναζητηθούν πιθανά λάθη. Η R διαθέτει τα χαρακτηριστικά που αρμόζουν με το συγκεκριμένο προγραμματιστικό παράδειγμα και επιτρέπουν την εφαρμογή του σε πραγματικά προβλήματα: η ευκολία δημιουργίας αντικειμένων με πολλά στοιχεία αλλά και επεξεργασίας όλων των στοιχείων του αντικειμένου με μια εντολή (διανυσματοποίηση), οι συναρτήσεις που είναι αντικείμενα πρώτης τάξης (και εν ’δυνάμει χωρίς side-effect), οι σωληνώσεις που δίνουν έμφαση στη συναρτησιακή γραφή, κ.α.

Στον απόλυτο, «καθαρό», συναρτησιακό προγραμματισμό απαξιώνονται οι μεταβλητές, ιδιαίτερα μάλιστα απαξιώνονται οι μεταβλητές που είναι εξωτερικές (μη τοπικές) στις συναρτήσεις, καθώς και οι

[320] βλ. παραπάνω §5.1.6 Συναρτήσεις ως ορίσματα και επιστρεφόμενες τιμές.
[321] βλ. παραπάνω §5.1.5 Αλληλεπίδραση με το περιβάλλον.

μεταβλητές (τοπικές ή εξωτερικές) που αφορούν αυστηρά ένα στοιχείο (οι μεταβλητές scalar[322]). Έτσι, χάνουν τη σημαντικότητά τους και οι κλασσικές δομές, όπως η επιλογή if και οι βρόχοι. Π.χ. πώς ενσωματώνεται ένας βρόχος for σε μια σωλήνωση που περιγράφει μια ακολουθία κλήσεων συναρτήσεων; Και αν τελικά χρησιμοποιηθεί for, πώς μπορεί αυτό να επιτευχθεί χωρίς μεταβλητή ελέγχου; Σε προηγούμενα κεφάλαια έχουν ήδη προταθεί προσεγγίσεις που μπορούν να εφαρμοστούν προς την κατεύθυνση αυτή, όπως η χρήση της συνάρτησης ifelse αντί της if[323], η χρήση της συνάρτησης outer[324], των συναρτήσεων της οικογένειας apply[325], ή της Map[326] αντί βρόχων.

Χρήσιμο εργαλείο μπορεί επίσης να είναι η συνάρτηση **Vectorize** του πακέτου 'base'. Η συνάρτηση παίρνει ως παράμετρο μια συνάρτηση η οποία δεν έχει σχεδιαστεί να επεξεργάζεται αντικείμενα με πολλά στοιχεία και επιστρέφει μια συνάρτηση «περιτύλιγμα» (wrapper function)[327] που το επιτρέπει. Για παράδειγμα, έχουμε την παρακάτω συνάρτηση που ελέγχει αν το a είναι μεγαλύτερο ή ίσο του 100:

```
Test_100_or_more <- function(a)
{
  if (a >= 100)   return(TRUE)
  return(FALSE)
}
```

Αν η συνάρτηση χρησιμοποιηθεί σε ένα αντικείμενο με πολλά στοιχεία, θα επιστρέψει απάντηση μόνο για το πρώτο στοιχείο του αντικειμένου. Αυτό οφείλεται στην if που χρησιμοποιείται για τον έλεγχο. Το παρακάτω στέλνει το vector 98:103 στη συνάρτηση[328], κάτι που εγείρει και σχετικό warning:

```
> 98:102 %>% Test_100_or_more
[1] FALSE
```

Δημιουργώντας μια διανυσματοποιημένη έκδοση της συνάρτησης μέσω της Vectorize, το πρόβλημα μπορεί να επιλυθεί:

```
vTest_100_or_more <- Vectorize(Test_100_or_more)
```

Η νέα συνάρτηση επιτρέπει επεξεργασία πολυμελών αντικειμένων:

```
> vTest_100_or_more(98:102)
[1] FALSE FALSE  TRUE  TRUE  TRUE
> 98:102 %>% vTest_100_or_more
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

Πολλά πακέτα της R έρχονται να προσθέσουν συναρτήσεις που διευκολύνουν την υιοθέτηση της συναρτησιακής προγραμματιστικής προσέγγισης ή ενισχύουν το οπλοστάσιό της. Βέβαια οι συναρτήσεις αυτές είναι εξίσου εφαρμόσιμες και σε κώδικα που δεν ακολουθεί αυστηρά το συναρτησιακό προγραμματιστικό μοντέλο. Άξια αναφοράς (την περίοδο συγγραφής του βιβλίου) είναι τα πακέτα 'dplyr' [55] και 'purrr' [66] τα οποία είναι μέρος της συλλογής πακέτων 'tidyverse' [57]. Ενδεικτική χρήση κάποιων συναρτήσεών τους παρατίθεται παρακάτω.

Συναρτήσεις του πακέτου 'purrr' [66] περιλαμβάνουν (μεταξύ πολλών άλλων) τις **map**, **mutate**, **accumulate**, **reduce** και διάφορες παραλλαγές τους. Το 'purrr' επεκτείνει τις δυνατότητες που παρέχουν συναρτήσεις ενσωματωμένες στην R, όπως η προαναφερθείσα Map αλλά και οι Reduce, Filter, Find, Negate, Position του πακέτου 'base'[329]. Οι συναρτήσεις του πακέτου 'purr' είναι προσανατολισμένες στην επεξεργασία αντικειμένων vector (atomic ή list) αλλά μπορούν να εφαρμοστούν και σε άλλους τύπους όπως data.frame. Μερικά παραδείγματα εντολών, με την έξοδό τους, ακολουθούν παρακάτω:

```
> 98:102 %>% map_lgl(Test_100_or_more)
[1] FALSE FALSE  TRUE  TRUE  TRUE

> map(iris[1:2],sum)
$Sepal.Length
```

[322] Στον προγραμματισμό, μεταβλητές που μπορούν να αποθηκεύουν μόνο ένα βασικό δεδομένο (έναν αριθμό, μία λογική τιμή κλπ.) αναφέρονται ως scalar.
[323] βλ.§3.2.2.2 Άλλες μέθοδοι επιλογής (switch, ifelse).
[324] βλ.§4.1.3.3 Η συνάρτηση outer.
[325] βλ.§4.1.3.4 Η οικογένεια συναρτήσεων apply.
[326] βλ.§4.2.1.3 Εφαρμογή συναρτήσεων σε list.
[327] βλ. υποσημείωση 391.
[328] Μπορεί να γραφτεί και ως Test_100_or_more(98:102).
[329] βλ. help(funprog).

```
[1] 876.5

$Sepal.Width
[1] 458.6

> iris[1:2] %>% map_dbl(sum)
Sepal.Length  Sepal.Width
       876.5        458.6
```

Οι συναρτήσεις **map_lgl** και **map_dbl** είναι παραλλαγές της map. Οι συναρτήσεις αυτές εφαρμόζουν τη συνάρτηση που ορίζεται στη δεύτερη παράμετρο σε όλα τα μέλη του αντικειμένου στην πρώτη[330]. Η map επιστρέφει αποτέλεσμα ως list, ενώ οι παραλλαγές της επιστρέφουν atomic vector. Στις παραλλαγές της map (αλλά και άλλων συναρτήσεων του 'purrr') το δεύτερο συνθετικό του ονόματος προσδιορίζει το mode του vector που θα επιστραφεί (_lgl για logical, _dlb για double, _chr για character κλπ.). Έτσι η map_lgl επιστρέφει logical ενώ η map_dlb επιστρέφει numeric (double).

Παρατηρήστε την πρώτη εντολή του παραδείγματος. Εδώ ζητείται να εφαρμοστεί σε κάθε στοιχείο του vector 98:102 η συνάρτηση Test_100_or_more. Η συνάρτηση αυτή δεν είναι σχεδιασμένη για vectors, αλλά η map (εδώ η παραλλαγή της, map_lgl) επιτρέπει τη χρήση της αναλαμβάνοντας την εφαρμογή της σε κάθε στοιχείο του 98:102.

Στη δεύτερη εντολή ζητείται να εφαρμοστεί η sum στις 4 πρώτες στήλες του data.frame με όνομα iris[331]. Χρησιμοποιώντας τη map το αποτέλεσμα επιστρέφεται ως list (ένα στοιχείο για κάθε άθροισμα στήλης). Η επόμενη παραλλαγή του ίδιου παραδείγματος (με χρήση της map_dbl) επιστρέφει το ίδιο αποτέλεσμα ως numeric (atomic vector).

Η συνάρτηση **map2** (και οι παραλλαγές της) συνδυάζουν τα στοιχεία από δύο αντικείμενα με ίδιο αριθμό στοιχείων. Παρακάτω, δίνεται ένα παράδειγμα με την παραλλαγή της (που επιστρέφει character) **map2_chr**:

```
> x<-c("Α","Β","Γ","Δ")
> y<-c("α","β","γ","δ")
> map2_chr(x, y, paste, sep=",")
[1] "Α,α" "Β,β" "Γ,γ" "Δ,δ"
```

Η map2 κάλεσε τη συνάρτηση (εδώ paste(sep=",")) για κάθε ζεύγος στοιχείων των δύο αντικειμένων x και y επιστρέφοντας το αποτέλεσμα. Η συνάρτηση **accumulate** επίσης επεξεργάζεται ένα αντικείμενο με πολλά στοιχεία εφαρμόζοντας κάποια συνάρτηση που της έχει δοθεί ως παράμετρος. Εδώ όμως ανατροφοδοτεί το προηγούμενο αποτέλεσμα στη συνάρτηση, μαζί με το στοιχείο του αντικειμένου (από το πρώτο στο τελευταίο) το οποίο επεξεργάζεται. Έτσι, κατά την επεξεργασία κάθε στοιχείου του αντικειμένου, χρησιμοποιείται η τιμή του και το αποτέλεσμα που είχε προκύψει από την επεξεργασία του προηγούμενου. Το παράδειγμα που ακολουθεί χρησιμοποιεί το x που ορίστηκε παραπάνω και καλεί την paste(sep=","):

```
> accumulate(x, paste, sep=",")
[1] "Α"       "Α,Β"     "Α,Β,Γ"   "Α,Β,Γ,Δ"
```

Σε κάθε βήμα επεξεργασίας, ένα στοιχείο του αντικειμένου (εδώ του x) περνά ως 2η παράμετρος στη συνάρτηση (εδώ της paste), ενώ το προηγούμενο αποτέλεσμα (αρχικά NULL ή 0 ή κάποια τιμή που έχει οριστεί στην παράμετρο .init) περνά ως πρώτη παράμετρός της. Η reduce[332] ανάγει το παραπάνω σε ένα μοναδικό αποτέλεσμα επιστρέφοντάς μόνο το τελικό αποτέλεσμα της accumulate:

```
> reduce(x, paste, sep=",")
[1] "Α,Β,Γ,Δ"
```

Ακολουθεί ένα παράδειγμα με αριθμητικές τιμές. Εδώ η accumulate ξεκινά από την τιμή 1000 και για κάθε στοιχείο του διανύσματος εφαρμόζει τη συνάρτηση y συνδυάζοντας το προηγούμενο αποτέλεσμα και το νέο αυτό στοιχείο. Εφόσον η y αφαιρεί από το πρώτο το δεύτερο, η επιστροφή είναι c(1000, 1000-100, 1000-100-20, 1000-100-20-10), ενώ το αποτέλεσμα της reduce είναι το τελικό αποτέλεσμα 1000-100-20-10:

```
> y<-function(x0,x1) x0-x1
> accumulate(c(100,20,10), y, .init=1000)
[1] 1000  900  880  870
```

[330] Παρεμφερές αποτέλεσμα με τα παραπάνω δίνει η εντολή Map(sum,iris[1:2]) χρησιμοποιώντας τη συνάρτηση Map του πακέτου 'base', βλ.§4.2.1.3 Εφαρμογή συναρτήσεων σε list.

[331] βλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων.

[332] Παρεμφερής είναι η συνάρτηση Reduce του πακέτου 'base'. Εδώ η εντολή θα ήταν Reduce(paste,x).

```
> reduce(c(100,20,10), y, .init=1000)
[1] 870
```

Το πακέτο ‘dplyr’ [55] επικεντρώνεται στον χειρισμό δεδομένων παρέχοντας συναρτήσεις όπως οι **select**, **filter**, **arrange, mutate group_by**, **ungroup**, **count**, **summarize** κ.α. Οι συναρτήσεις είναι generic[333] άρα μπορούν να επαναπροσδιοριστούν ανάλογα με τον τύπο του αντικειμένου. Η υλοποίησή τους για αντικείμενα τύπου data.frame παρέχεται από το ‘dplyr’. Το παράδειγμα, χρησιμοποιεί τις συναρτήσεις στο data.frame iris:

```
iris %>%
  select(Sepal.Width,Sepal.Length,Species) %>%
  filter(Sepal.Width>3) %>%
  group_by(Species) %>%
  summarise(Μέσο_Μήκος=mean(Sepal.Length))
```

Η παραπάνω εντολή ξεκινά από το iris, επιλέγει τις μεταβλητές του Sepal.Width, Sepal.Length, και Species (απορρίπτοντας τις υπόλοιπες), απορρίπτει μετά τις εγγραφές με Sepal.Width <= 3, ομαδοποιεί τα δεδομένα ως προς το είδος (Species) και υπολογίζει το μέσο Sepal.Length ανά ομάδα σε νέα μεταβλητή (στήλη) με όνομα Μέσο_Μήκος. Το αποτέλεσμα είναι:

```
# A tibble: 3 x 2
  Species    Μέσο_Μήκος
  <fct>           <dbl>
1 setosa           5.08
2 versicolor       6.49
3 virginica        6.81
```

Ένα παράδειγμα εντολής με χρήση των select, arrange και mutate είναι το παρακάτω:

```
iris %>%
  select(Sepal.Length,Sepal.Width) %>%
  arrange(Sepal.Length) %>%
  mutate(Sepal.Area = Sepal.Length * Sepal.Width) %>% head
```

Η παραπάνω εντολή ξεκινά πάλι από το iris, επιλέγει τις μεταβλητές Sepal.Length και Sepal.Width, ταξινομεί τα δεδομένα ως προς την πρώτη (συνάρτηση arrange) και δημιουργεί μέσω της συνάρτησης **mutate**, μια νέα μεταβλητή με όνομα Sepal.Area η οποία καταγράφει το γινόμενο των άλλων δύο. Το αποτέλεσμα τροφοδοτείται στη head για εμφάνιση μόνο των πρώτων εγγραφών, οι οποίες είναι:

```
  Sepal.Length Sepal.Width Sepal.Area
1          4.3         3.0      12.90
2          4.4         2.9      12.76
3          4.4         3.0      13.20
4          4.4         3.2      14.08
5          4.5         2.3      10.35
6          4.6         3.1      14.26
```

Για την αποτίμηση εκφράσεων στη mutate, το ‘dplyr’ παρέχει διάφορες χρήσιμες συναρτήσεις[334]. Μια από αυτές είναι η **case_when**:

```
> x <- 1:10
> case_when(x<2 ~ 0,  x<5 ~ 1, x<8 ~ 2, TRUE ~ 3)
 [1] 0 1 1 1 2 2 2 3 3 3
```

Η case_when είναι μια διανυσματοποιημένη δομή απόφασης που υλοποιεί πολλαπλά if...else. Επιστρέφει διάνυσμα αποτελεσμάτων με ίδιο αριθμό στοιχείων με το ελεγχόμενο. Το TRUE ορίζει την default τιμή που θα τοποθετηθεί στο νέο διάνυσμα, αν όλοι οι άλλοι έλεγχοι στο αντίστοιχο στοιχείο αποτύχουν. Η εντολή εδώ, θα αναθέσει την τιμή 0 στο στοιχείο του νέου διανύσματος αν το αντίστοιχο στοιχείο του x είναι <2, αλλιώς αν είναι <5 θα δώσει την τιμή 1, αλλιώς αν είναι <8 τη τιμή 2, ενώ αν όλα τα προηγούμενα αποτύχουν θα δώσει την τιμή 3. Η case_when μπορεί να αξιοποιηθεί σε κάποια εντολή mutate όπως:

```
iris %>%
  select(Sepal.Length,Sepal.Width) %>%
  mutate(Sepal.Type = case_when(Sepal.Length>5  ~ "Long",
```

[333] βλ. Κεφάλαιο 6. Οι generic συναρτήσεις παρουσιάζονται στην §6.3 Κλάσεις S3.
[334] βλ. help(mutate) του πακέτου ‘dplyr’, ενότητα: Useful mutate functions.

```
                    Sepal.Length>4.7~ "Mid",
                    TRUE ~ "Short")) %>% head
```

Όπως και σε παράδειγμα που προηγήθηκε, η εντολή ξεκινά από το iris και επιλέγει τις μεταβλητές Sepal.Length και Sepal.Width. Στο αποτέλεσμα, μέσω της συνάρτησης **mutate** προσθέτει μια νέα μεταβλητή (στήλη) με όνομα Sepal.Type η οποία θα έχει την τιμή που ορίζουν οι κανόνες ("Long" όπου το Sepal.Length είναι > 5 κλπ.). Το αποτέλεσμα τροφοδοτείται στη head για εμφάνιση μόνο των πρώτων εγγραφών, οι οποίες είναι:

```
  Sepal.Length Sepal.Width Sepal.Type
1          5.1         3.5       Long
2          4.9         3.0        Mid
3          4.7         3.2      Short
4          4.6         3.1      Short
5          5.0         3.6        Mid
6          5.4         3.9       Long
```

Οι συναρτήσεις που παρουσιάστηκαν στην ενότητα αυτή είναι ενδεικτικές όσων προσφέρονται από τα πακέτα αυτά. Αλλά και οι συναρτήσεις που αναφέρθηκαν παραπάνω υποστηρίζουν παραμέτρους για σύνθετες λειτουργίες που δεν παρουσιάστηκαν εδώ. Περισσότερα υπάρχουν στην τεκμηρίωσή των πακέτων και σε πηγές όπως τα [33] και [67]. Πακέτα όπως τα ‘purr’ και ‘dplyr’ είναι βάση για μεγάλο αριθμό άλλων. Το οικοσύστημα της R εμπλουτίζεται συνεχώς με νέες δυνατότητες και πακέτα που αναδεικνύουν τα πλεονεκτήματα του συναρτησιακού προγραμματιστικού μοντέλου και διευκολύνουν την υιοθέτησή του σε πραγματικά προβλήματα.

Αναφορές Κεφαλαίου 5

[10] Σταυρακούδης, Α. (2012). *Εισαγωγή στις υπολογιστικές μεθόδους για τις οικονομικές και επιχειρησιακές σπουδές.* Αθήνα: Κλειδάριθμος. ISBN 978-960-461-511-7.

[33] Wickham, H. (2019). *Advanced R (2nd Editton)*. Chapman and Hall/CRC. https://adv-r.hadley.nz/

[55] Wickham, H., François, R., Henry, L., Müller, K. (2021). dplyr: A Grammar of Data Manipulation. https://CRAN.R-project.org/package=dplyr και https://dplyr.tidyverse.org/

[57] Wickham H. et al. (2019). Welcome to the tidyverse. *Journal of Open Source Software, τόμ. 4, αρ. 43*, p. 1686.

[60] Peng, R. D., Kross, S., & Anderson, B. (2020). *Mastering Software Development in R*. Ηλεκτρονικό. https://bookdown.org/rdpeng/RProgDA/

[62] Chambers, J. M. (2016). *Extending R, 1st Edition*. Chapman and Hall/CRC. ISBN 978-1-4987-7572-4.

[63] Altfeld, J. (2021). TryCatchLog: Advanced 'tryCatch()' and 'try()' Functions. https://CRAN.R-project.org/package=xts

[64] Milton Bache, S., & Wickham, H. (2022). Magrittr: A Forward-Pipe Operator for R. https://CRAN.R-project.org/package=magrittr

[65] Wickham, H., & Grolemund, G. (2016). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data, 1st Edition.* O'Reilly Media, Inc. ISBN 978-1491910399.

[66] Henry, L., & Wickham, H. (2020). Purrr: Functional Programming Tools. https://CRAN.R-project.org/package=purrr

[67] Rodrigues, B. (2020). Modern R with the tidyverse, Leanpub. https://b-rodrigues.github.io/modern_R/

Κεφάλαιο 6: Κλάσεις και αντικειμενοστραφής προγραμματισμός

Σύνοψη

Η οργάνωση της λειτουργικότητας σε κλάσεις είναι ένας τρόπος με τον οποίο μπορεί να γίνει καλύτερη διαχείριση της λύσης που υλοποιείται προγραμματιστικά και να καθοριστεί καλύτερα ο ρόλος κάθε τμήματός του κώδικα. Το κεφάλαιο αυτό παρουσιάζει διάφορα θέματα που σχετίζονται με τον αντικειμενοστραφή προγραμματισμό καθώς και τις ιδιαιτερότητες με τις οποίες εφαρμόζεται το αντικειμενοστραφές προγραμματιστικό παράδειγμα στην R και πώς συνδυάζεται με το συναρτησιακό προγραμματιστικό μοντέλο που η γλώσσα ακολουθεί. Παρουσιάζονται επίσης οι κλάσεις S3, S4 και τα συστήματα αναφοράς RS και R6, δηλαδή οι πλέον διαδεδομένοι τρόποι ορισμού κλάσεων στη γλώσσα αυτή.

Προαπαιτούμενη γνώση

Βασικά στοιχεία προγραμματισμού στην R (Κεφ. 3), συνήθεις τύποι αντικειμένων (Κεφ. 4), δημιουργία συναρτήσεων (Κεφ. 5).

6.1 Γιατί κλάσεις;

Όπως έχει ήδη αναφερθεί, κάθε αντικείμενο στην R ανήκει σε κάποια κλάση, που επιστρέφεται από τη συνάρτηση class[335]. Στον αντικειμενοστραφή προγραμματισμό (Object Oriented ή OO programing), οι κλάσεις είναι πρότυπα για αντικείμενα. Περιγράφουν τις ιδιότητες και τη συμπεριφορά που θα έχει κάθε αντικείμενο που ανήκει στην κλάση αυτή. Η προσέγγιση αυτή έρχεται από τον πραγματικό κόσμο. Για παράδειγμα αν περιγράφουμε σε κάποιον ότι «έχουμε τρία ποτήρια», θα καταλάβει ότι αναφερόμαστε σε συγκεκριμένα αντικείμενα που ανήκουν στη γενικότερη κατηγορία, τύπο, σύνολο ή κλάση ομογενών αντικειμένων που ονομάζουμε «ποτήρια». Επειδή ακριβώς καθένα από αυτά είναι μέλος αυτής της κλάσης αντικειμένων, γνωρίζουμε τι χαρακτηριστικά πρέπει να έχει και τι ενέργειες μπορούμε να κάνουμε με αυτό. Γνωρίζουμε ότι μπορούμε να το γεμίσουμε, να το αδειάσουμε, να το μετακινήσουμε, να το πλύνουμε κλπ., δηλαδή ξέρουμε πώς να αλληλεπιδρούμε μαζί του, γνωρίζουμε τη συμπεριφορά του και αυτή είναι παρεμφερής για κάθε αντικείμενο της κλάσης αυτής. Επιπρόσθετα, η κλάση υποβάλει και τον τρόπο (τη μέθοδο), με την οποία γίνεται ο χειρισμός του αντικειμένου. Ένα αντικείμενο της κλάσης «ποτήρι» γεμίζει. Το ίδιο και κάθε αντικείμενο της κλάσης «επαναφορτιζόμενη μπαταρία». Αλλά ο τρόπος με τον οποίο γίνεται αυτό, καθώς και τα απαιτούμενα για να γίνει, είναι εντελώς διαφορετικά και επιβάλλονται από την κλάση στην οποία ανήκουν τα δύο αντικείμενα.

Έτσι, σε μια κλάση ανήκουν ομογενή, παρεμφερή αντικείμενα. Όμως κάθε συγκεκριμένη περίπτωση (instance) αντικειμένου της κλάσης, δηλαδή κάθε μέλος της κλάσης διαφέρει από τα άλλα. Αυτό συμβαίνει γιατί δεν μοιράζεται με τα άλλα την τρέχουσα κατάστασή του. Αν αναφερόμαστε στην κλάση αντικειμένων «ποτήρια», κάθε ποτήρι έχει κάποια χαρακτηριστικά και ιδιότητες όπως το χρώμα του, το σχήμα του, τη θέση του, τον βαθμό στον οποίο είναι γεμάτο, το υγρό που περιέχει (κάτι που ίσως εμπλέκει κάποια άλλη κλάση αντικειμένων, τα «ρευστά υλικά»), αν είναι πλυμένο κλπ. Τέτοια είναι ίσως τα χαρακτηριστικά που περιγράφουν οποιοδήποτε αντικείμενο της κλάσης «ποτήρι». Όμως η κατάσταση που παρατηρείται στα χαρακτηριστικά αυτά για καθένα συγκεκριμένο ποτήρι πιθανότατα διαφέρει από τα υπόλοιπα. Οπότε, όλα τα μέλη τις κλάσης «ποτήρια» περιγράφονται από τα ίδια χαρακτηριστικά, όλα έχουν κάποιο σχήμα, χρώμα, κλπ., αλλά κάθε συγκεκριμένο ποτήρι έχει τα δικά του δεδομένα, τις δικές του τιμές στις ιδιότητες αυτές.

Επίσης, η κλάση «ποτήρια» ίσως διαχωρίζεται (στη σκέψη μας) σε υποκατηγορίες (υπο-κλάσεις), π.χ. στην κλάση «ανακυκλώσιμα ποτήρια» και «μη-ανακυκλώσιμα ποτήρια». Και τα αντικείμενα που ανήκουν στην κλάση «ανακυκλώσιμα ποτήρια», ίσως να κληρονομούν ιδιότητες και συμπεριφορά τόσο από την κλάση αντικειμένων «ποτήρια» αλλά και από κάποια άλλη κλάση αντικειμένων που αφορά «ανακυκλώσιμα αντικείμενα». Ένα αντικείμενο μπορεί να ανήκει ταυτόχρονα σε πολλές κλάσεις.

Είναι δύσκολο να διαφωνήσει κανείς στο ότι τα «ποτήρια» είναι όντως αντικείμενα. Αλλά η λέξη «αντικείμενο» δεν αφορά μόνο υλικές οντότητες. Ένα ταξίδι, μια αίτηση, μία εξίσωση, μια τηλεπικοινωνιακή σύνδεση ή μια τηλεφωνική κλήση, μια φιλία, ένα παράθυρο στην οθόνη κάποιας ψηφιακής συσκευής, ένας πίνακας αριθμών, όλα είναι περιπτώσεις «οντοτήτων». Όλα μοιράζονται χαρακτηριστικά και συμπεριφορά με

[335] βλ. Κεφάλαιο 4.

άλλες παρεμφερείς περιπτώσεις, οπότε μπορεί να θεωρηθούν ότι ανήκουν σε κάποια κοινή κλάση αντικειμένων.

Παραπάνω έγινε μια ιεράρχηση κλάσεων αντικειμένων που υπάρχουν ήδη στο περιβάλλον μας. Ο δικός σας τρόπος να αντιλαμβάνεστε ομάδες ομογενών αντικειμένων και η ιεράρχηση που κάνετε σε αυτές μπορεί να είναι διαφορετική και ίσως καλύτερη. Ένα μέρος της γνωστικής διαδικασίας, της αντίληψης του κόσμου που μας περιβάλλει, βασίζεται σε (συχνά ασυναίσθητη) κατάταξη αντικειμένων, υλικών ή αφηρημένων, σε κατηγορίες. Για να μπορούμε να δράσουμε ευκολότερα μέσα στο περιβάλλον μας ομαδοποιούμε τις διάφορες οντότητες γύρω μας (βάσει κοινών χαρακτηριστικών) ή τις κατηγοριοποιούμε (μέσω κανόνων και οδηγιών που μας δίνονται ή έχουμε μόνοι μας δημιουργήσει από πρότερη εμπειρία)[336]. Αυτή η κατηγοριοποίηση συχνά γίνεται αυθαίρετα ή και λανθασμένα ή αλλάζει. Αλλά βολεύει καθώς οργανώνει τον τρόπο με τον οποίο αντιλαμβανόμαστε και αλληλοεπιδρούμε με το περιβάλλον μας.

Αυτή την προσέγγιση προσπαθεί να μιμηθεί ο αντικειμενοστραφής προγραμματισμός και να την εφαρμόσει στην οργάνωση των λειτουργικών μονάδων του κώδικα. Βασική διαφορά είναι ότι εδώ, οι προγραμματιστές δεν είναι αποδέκτες αλλά δημιουργοί των κλάσεων, κατασκευάζοντας το πρότυπο των αντικειμένων, ορίζοντας τι θα χαρακτηρίζει την κατάστασή τους (τι δεδομένα θα διατηρούν εσωτερικά) και με τι μεθόδους (έτσι ονομάζονται οι συναρτήσεις που παρέχει μια κλάση στον αντικειμενοστραφή προγραμματισμό) θα χρησιμοποιούνται. Στην προσέγγιση αυτή, τα δεδομένα διατηρούνται σε αντικείμενα κάποιας κλάσης. Η κλάση του αντικειμένου ορίζει ποια μορφή θα έχουν τα δεδομένα που διατηρεί κάθε αντικείμενο-μέλος της και ποιες μέθοδοι (συναρτήσεις) θα μπορούν να χρησιμοποιούνται σε αυτό.

Πολλές από τις ευρύτατα χρησιμοποιούμενες γλώσσες προγραμματισμού είναι αντικειμενοστραφείς. Τέτοιες γλώσσες με ισχυρό, πολυσύνθετο αντικειμενοστραφές συντακτικό (όπως οι C++, C#, Java κ.α.), επιτρέπουν τον ορισμό κλάσεων αντικειμένων και τη δημιουργία αντικειμένων από αυτές. Επιπροσθέτως παρέχουν στους δημιουργούς κλάσεων τη δυνατότητα να εφαρμόσουν στον σχεδιασμό τους τέσσερις αρχές του αντικειμενοστραφούς μοντέλου: (α) την αφαιρετικότητα (abstraction[337]) που επιτρέπει να δημοσιοποιείται μόνο η διεπαφή (interface) των αντικειμένων της κλάσης προς το περιβάλλον τους κρύβοντας τον τρόπο που έχει γίνει η υλοποίηση τους, (β) την ενθυλάκωση (encapsulation) που επιτρέπει δεδομένα και μέθοδοι κάθε αντικειμένου να συσκευάζονται εντός του και να ορίζεται επακριβώς η πρόσβαση που θα έχει το περιβάλλον σε αυτά, απαγορεύοντας επιλεκτικά την πρόσβαση σε κάποια δεδομένα ή μεθόδους ενός αντικειμένου από αντικείμενα κάποιων ή όλων των άλλων κλάσεων, (γ) την κληρονομικότητα (inheritance), δηλαδή τη δυνατότητα μια κλάση να είναι υπο-κλάση άλλων, κληρονομώντας κάποιες ή όλες τις ιδιότητες και συμπεριφορές τους, (δ) τον πολυμορφισμό (polymorphism) που επιτρέπει πολλαπλές μεθόδους με το ίδιο όνομα (και προφανώς τον ίδιο σκοπό) να διαφέρουν ως προς τον αριθμό και τον τύπο των παραμέτρων τους και αυτά να χρησιμοποιούνται στην επιλογή της πλέον κατάλληλης μεθόδου που θα κληθεί.

6.2 Αντικειμενοστραφής προγραμματισμός στην R

Η γλώσσα S, πάνω στην οποία βασίζεται η R, εξελίχθηκε αρχικά χωρίς κάποιο αντικειμενοστραφές προγραμματιστικό μοντέλο. Οι βασικοί τύποι αντικειμένων, που είναι βασικά δομικά στοιχεία της γλώσσας, δεν υλοποιούνται με αντικειμενοστραφή τρόπο[338]. Επίσης δεν υπάρχει ένα μοναδικό ενιαίο μοντέλο δημιουργίας κλάσεων στην S, όπως δεν υπάρχει και στην R. Σύντομα όμως προστέθηκαν δυνατότητες αντικειμενοστραφούς προγραμματισμού. Αυτό έπρεπε να γίνει διατηρώντας τη διαδραστική φύση των γλωσσών αυτών. Όλες οι εντολές, ακόμα και αυτές που αφορούσαν ορισμό κλάσεων, έπρεπε να μπορούν να εκτελούνται με απευθείας καταχώρησή τους σε γραμμή εντολών (το Console), ενώ η ανάλυση του κώδικα να γίνεται μόνο σε επίπεδο εντολής. Επιπροσθέτως, έπρεπε να τηρούνται οι αρχές του συναρτησιακού προγραμματιστικού μοντέλου που είναι ιδιαίτερα σημαντικό για τις γλώσσες αυτές και πάνω στο οποίο είναι βασισμένες.

---

[336] Οι αντίστοιχες λειτουργίες στη μηχανική μάθηση ονομάζονται συσταδοποίηση (clustering) και αναγνώριση προτύπων (pattern recognition).

[337] Η συγγραφή συναρτήσεων (βλ.§5.1.1 Δημιουργία συναρτήσεων) προσφέρει ήδη ένα επίπεδο abstraction: οι χρήστες της συνάρτησης χρειάζεται να γνωρίζουν μόνο το όνομα και τις παραμέτρους της συνάρτησης για να τη χρησιμοποιήσουν. Δεν χρειάζεται (συνήθως) να γνωρίζουν περισσότερα για το πώς γίνεται η υλοποίηση της και πώς λειτουργεί εσωτερικά, αλλά τη χρησιμοποιούν ως «μαύρο κουτί».

[338] Αυτό μπορεί να ελεγχθεί με τη συνάρτηση **is.object** του πακέτου 'base'. Για ένα numeric (όπως is.object(c(10,20)) επιστρέφει FALSE, ενώ για ένα data.frame (όπως is.object(iris)) επιστρέφει TRUE.

Έτσι η γλώσσα R έχει εμπλουτιστεί με πολλαπλές αντικειμενοστραφείς μεθοδολογίες, διαφορετικά συστήματα ορισμού κλάσεων, τα οποία μπορούν να συνυπάρχουν και να χρησιμοποιούνται ταυτόχρονα στον κώδικα. Καθένα προσφέρει διαφορετικά πλεονεκτήματα και μειονεκτήματα, ενώ κάποια διαφέρουν αρκετά από την προσέγγιση άλλων αντικειμενοστραφών γλωσσών.

Οτιδήποτε υπάρχει στη γλώσσα R, κάθε αντικείμενο που δημιουργείται, είναι μέλος τουλάχιστον μίας κλάσης αντικειμένων. Συνήθως η πληροφορία για την κλάση (ή τις κλάσεις) στην οποία ανήκει ένα αντικείμενο καταγράφεται σε ιδιότητά του με όνομα class (βλ. .§4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων). Εξαίρεση αποτελούν οι τύποι αντικειμένων για τους οποίους η κλάση θεωρείται εγγενής (implicit class) και δεν καταγράφεται ως ιδιότητα, αλλά προκύπτει έμμεσα από τη φύση του αντικειμένου. Τέτοιοι τύποι περιλαμβάνουν τα βασικά δομικά στοιχεία της R, τύποι όπως numeric, logical, character και άλλα vector, list, matrix, array, function κλπ. Όλες οι υπόλοιπες κλάσεις αντικειμένων έχουν οριστεί με άμεσο τρόπο (explicit class) χρησιμοποιώντας κάποιο από τα συστήματα κλάσεων (S3, S4, RS κλπ.) που υπάρχουν στη γλώσσα. Σε κάθε περίπτωση, για όλους τους τύπους αντικειμένων (ορισμένους είτε με implicit είτε με explicit τρόπο και είτε καταγεγραμμένους ως ιδιότητα είτε όχι), η πληροφορία της κλάσης του αντικειμένου επιστρέφεται από τη συνάρτηση class.

Βασικά συστήματα ορισμού κλάσεων στην R είναι οι κλάσεις S3, S4 (ή Formal Classes) και RS (ή Reference Classes). Τα συστήματα αυτά ορίζονται σε πακέτα που συμπεριλαμβάνονται με τη γλώσσα. Οι ιδιαιτερότητες των βασικών αντικειμενοστραφών συστημάτων S3 και S4 της R περιγράφονται μεταξύ άλλων στα [33], [54] και [68]. Επιπροσθέτως, το οικοσύστημα της R έχει δημιουργήσει και άλλα συστήματα αντικειμενοστραφούς προγραμματισμού (όπως το σύστημα R6), που παρέχονται σε πακέτα τα οποία μπορούν να εγκατασταθούν στο σύστημα. Τέλος, η R επιτρέπει την ενσωμάτωση και συνεργασία με κώδικα (και κλάσεις) γραμμένες σε άλλες γλώσσες αντικειμενοστραφούς προγραμματισμού, όπως οι C++, Java και Python (βλ. Κεφάλαιο 7).

6.3 Κλάσεις S3

Οι κλάσεις S3 [17] [69] είναι ένα από τα επικρατέστερα συστήματα αντικειμενοστραφούς προγραμματισμού στην R. Το σύστημα κλάσεων S3 είναι ενσωματωμένο στο πακέτο ‘base’ της R και χρησιμοποιείται ευρέως από βασικά πακέτα της γλώσσας (όπως το ‘stats’), αλλά και από πολλά πακέτα τρίτων. Το σύστημα είναι ιδιαίτερα απλό και απολύτως συνεπές με το συναρτησιακό προγραμματιστικό μοντέλο το οποίο είναι πρωτεύον στην R. Από το σύστημα S3 απουσιάζουν πολλά από τα χαρακτηριστικά που έχουν άλλες αντικειμενοστραφείς προσεγγίσεις (π.χ. δεν υπάρχει κληρονομικότητα μεταξύ κλάσεων[339], ούτε ενθυλάκωση).

Η ομορφιά του συστήματος S3 βρίσκεται στην απλότητά του. Μια κλάση S3 είναι απλώς ένα όνομα που καταχωρίζεται σε συγκεκριμένη ιδιότητα[340] των αντικειμένων της. Η ιδιότητα αυτή ονομάζεται class. Δεν υπάρχει κάποιος άλλος ορισμός της κλάσης ούτε (άμεσος) προσδιορισμός των δεδομένων που θα διατηρούν τα αντικείμενά της. Επίσης, ένα αντικείμενο μπορεί να ανήκει σε περισσότερες της μίας κλάσης[341], οπότε η τιμή στην ιδιότητα class του αντικειμένου αυτού θα είναι ένα vector με περισσότερα του ενός ονόματα κλάσεων και θα το αναφέρουμε ως class vector. Ας δούμε τη χρήση του συστήματος S3 στην πράξη:

Δοκιμάστε:	**Σχόλιο**
`a<-0`	Κάποιο αντικείμενο (εδώ numeric με τιμή 0) σε κάποια μεταβλητή (εδώ a).
`class(a)<-c("κλάση1")`	Ορίζει πως η κλάση που ανήκει το αντικείμενο στο a ονομάζεται «κλάση1».

Αν ένα αντικείμενο ανήκει σε πολλές κλάσεις, τότε το class vector του απλώς περιέχει περισσότερα του ενός ονόματα κλάσεων. Η σειρά τους έχει σημασία (για λόγους που θα αναφερθούν παρακάτω) και επιτρέπεται η ένταξη ενός αντικειμένου σε μια κλάση δυναμικά (όπως και η απένταξή του από αυτή).

Δοκιμάστε:	**Σχόλιο**
`b<-20`	Αντικείμενο (εδώ numeric με τιμή 20) σε κάποια μεταβλητή (εδώ b).
`class(b)<-c("κλάση1","κλάση2")`	Ορίζονται ως κλάσεις του αντικειμένου β οι: «κλάση1» και «κλάση2».

[339] Αν και ένα αντικείμενο μπορεί να κληρονομεί μεθόδους από περισσότερες της μιας κλάσεις, βλ. παρακάτω.
[340] βλ.§4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων.
[341] Έχουν ήδη αναφερθεί περιπτώσεις αντικειμένων που ανήκουν σε πολλές κλάσεις όπως π.χ. τα αντικείμενα τύπου matrix, που ανήκουν ταυτόχρονα στην κλάση matrix και στην κλάση array.

Οι συναρτήσεις **isa** και **inherits** ελέγχουν τη σχέση ενός αντικειμένου με κάποια κλάση. Η isa ελέγχει αν η κλάση του αντικειμένου είναι η ορισθείσα και μόνο αυτή. Η συνάρτηση inherits ελέγχει αν η κλάση είναι μία από αυτές στις οποίες ανήκει. Επίσης μπορεί να χρησιμοποιηθεί για τον ίδιο λόγο η συνάρτηση **is** που παρέχεται από το πακέτο 'methods' (το οποίο παρουσιάζεται στην επόμενη ενότητα).

Δοκιμάστε:	**Σχόλιο**
`isa(a,"κλάση1")`	Επιστρέφει TRUE, το a ανήκει στην «κλάση1» και μόνο σε αυτή.
`inherits(a,"κλάση1")`	Επιστρέφει TRUE, το a κληρονομεί μεθόδους[342] από την «κλάση1».
`inherits(a,"κλάση2")`	FALSE, το a δεν κληρονομεί μεθόδους από την «κλάση2».
`isa(b,"κλάση1")`	Επιστρέφει FALSE το b δεν ανήκει μόνο στην «κλάση1».
`inherits(b,"κλάση1")`	Επιστρέφει TRUE, το b κληρονομεί μεθόδους από την «κλάση1».
`inherits(b,"κλάση2")`	Επιστρέφει TRUE, το b κληρονομεί μεθόδους και από την «κλάση2».

Οι μέθοδοι που μπορεί να υποστηρίξουν τα αντικείμενα μιας οποιασδήποτε κλάσης S3 είναι συγκεκριμένες και κοινές για όλες τις κλάσεις S3. Αυτό που ίσως διαφέρει είναι ο τρόπος με τον οποίο υλοποιείται η μέθοδος για κάποια συγκεκριμένη κλάση. Οι μέθοδοι που υποστηρίζουν οι κλάσεις S3 σχετίζονται με τις «γενικές συναρτήσεις» (generic function). Στα πακέτα της R έχει οριστεί πληθώρα από «γενικές» συναρτήσεις και ο όρος generic function έχει αναφερθεί και αλλού στο βιβλίο αυτό καθώς πολλές από τις ευρέως χρησιμοποιούμενες συναρτήσεις (όπως οι print[343], summary[344], sum, mean, plot και άλλες), είναι «γενικές» (generic) συναρτήσεις. Έτσι, generic functions συμβαίνει να είναι πολλές από τις συναρτήσεις που χρησιμοποιήθηκαν ήδη σε παραδείγματα προηγούμενων κεφαλαίων.

Μια generic συνάρτηση συνήθως δεν έχει ιδιαίτερο κώδικα. Έτσι π.χ. ο κυρίως κώδικας της print αποτελείται μόνο από την εντολή UseMethod("print")[345]. Ο κώδικας αυτός απλώς καλεί τη συνάρτηση **UseMethod[346]** που με τη σειρά της εντοπίζει και εκτελεί την κατάλληλη μέθοδο βάση της κλάσης (ή των κλάσεων) στην οποία ανήκει το αντικείμενο της πρώτης παραμέτρου. Θεωρώντας πως αυτό είναι το βασικό αντικείμενο στο οποίο γίνεται επεξεργασία, μια generic συνάρτηση καλεί τη σχετική μέθοδο που έχει δημιουργηθεί για κάποια από τις κλάσεις στις οποίες το αντικείμενο ανήκει. Έτσι, στο σύστημα S3 η επιλογή της μεθόδου που θα κληθεί βασίζεται στην εξέταση μόνο μιας παραμέτρου, κάτι που ονομάζεται μέθοδος single dispatch.

Οι κλάσεις S3 δίνουν λοιπόν τη δυνατότητα στους δημιουργούς μιας κλάσης να ορίσουν πώς ακριβώς θα υλοποιούνται οι κλήσεις σε generic συναρτήσεις για τα αντικείμενα της κλάσης που έχουν δημιουργήσει. Όταν κληθεί μια generic συνάρτηση π.χ. η print, με πρώτο όρισμα αντικείμενο της κλάσης αυτής, τότε αναζητείται και εκτελείται η αντίστοιχη μέθοδος που έχει οριστεί για την κλάση. Τα ονόματα των μεθόδων για τις κλάσεις S3 έχουν τη μορφή:

[όνομα generic συνάρτησης].[όνομα κλάσης S3]

Έτσι π.χ. **print.factor** είναι η μέθοδος (συνάρτηση) που θα εκτελεστεί αν χρησιμοποιηθεί η print σε αντικείμενο της κλάσης factor. Αν η πρώτη κλάση στην οποία ανήκει το αντικείμενο δεν έχει μέθοδο για τη συγκεκριμένη generic συνάρτηση, αναζητείται στις επόμενες κλάσεις που έχουν οριστεί για το αντικείμενο. Για τον λόγο αυτό, η σειρά με την οποία έχουν οριστεί τα ονόματα των κλάσεων στο class vector είναι σημαντική. Τέλος, αν δεν βρεθεί καμία σχετική μέθοδος σε καμία κλάση του αντικειμένου, καλείται η συνάρτηση με όνομα:

[όνομα generic συνάρτησης].default

Άρα, αν δεν έχει οριστεί η σχετική συνάρτηση για τη generic συνάρτηση print σε καμία από τις κλάσεις του αντικειμένου, τότε θα κληθεί η προεπιλεγμένη συνάρτηση **print.default** και η εντολή θα υλοποιηθεί με τον προεπιλεγμένο τρόπο (που απλώς εμφανίζει τα δεδομένα του αντικειμένου)[347]. Όλες οι μέθοδοι που είναι

---

[342] Όπως αναφέρθηκε παραπάνω, μέθοδοι ονομάζονται οι συναρτήσεις που παρέχει μια κλάση και αφορούν τον χειρισμό των αντικειμένων της.

[343] βλ.§2.3.2 Έξοδος και εμφάνιση κειμένου.

[344] βλ.§2.7 Οι συναρτήσεις-βοηθήματα: str, summary και dput.

[345] Μπορείτε να το επιβεβαιώσετε καταχωρώντας print στο console.

[346] Αυτό γίνεται συνήθως αλλά όχι πάντα. Κάποιες generic συναρτήσεις χρησιμοποιούν την DispatchGroup ή η DispatchOrEval επειδή εμπλέκεται ήδη μεταγλωττισμένος κώδικας από άλλη γλώσσα προγραμματισμού. Η UseMethod παρέχεται από το πακέτο ΄base΄.

[347] Απευθείας κλήση των default συναρτήσεων δεν συνιστάται.

διαθέσιμες στην τρέχουσα συνεδρία για χειρισμό μιας generic συνάρτησης[348] επιστρέφονται από τη συνάρτηση **methods**[349]. Για παράδειγμα οι μέθοδοι που αναλαμβάνουν τον χειρισμό της generic συνάρτησης print ανά κλάση θα μπορούσαν να είναι:

```
> methods(print)
  [1] print.acf*
  [2] print.anova*
…
[187] print.xngettext*
[188] print.xtabs*
```

Μπορούν λοιπόν να προστεθούν μέθοδοι χειρισμού κάποιων generic συναρτήσεων και για τις νέες κλάσεις (με όνομα «κλάση1» και «κλάση2») του παραδείγματος. Αυτό γίνεται δημιουργώντας τις σχετικές συναρτήσεις με το κατάλληλο όνομα. Για παράδειγμα, οι παρακάτω είναι (απλοϊκές) μέθοδοι χειρισμού της print για την «κλάση1» και την «κλάση2», αντίστοιχα. Οι μέθοδοι οφείλουν να έχουν παρόμοια «υπογραφή» με τη συνάρτηση που χειρίζονται, δηλαδή να δέχονται τις ίδιες παραμέτρους (συμπεριλαμβανομένης της '…'[350]) και με την ίδια σειρά:

```
print.κλάση1<-function(x,...)
  { cat("Εμφανίζω αντικείμενο κλάσης 1\n") }

print.κλάση2<-function(x,...)
  { cat("Εμφανίζω αντικείμενο κλάσης 2\n") }
```

Έτσι αν ερωτηθεί τώρα το σύστημα ποια μέθοδος χειρίζεται την print για την κλάση «κλάση2» (με τη συνάρτηση **getS3method**) θα επιστρέψει την παραπάνω συνάρτηση:

```
> getS3method("print","κλάση2")
function(x,...)
  { cat("Εμφανίζω αντικείμενο κλάσης 2\n") }
```

Αποτέλεσμα των παραπάνω στα αντικείμενα που έχουν ήδη οριστεί είναι:

```
> print(a)
Εμφανίζω αντικείμενο κλάσης 1

> print(b)
Εμφανίζω αντικείμενο κλάσης 1
```

Στο a, που ανήκει μόνο στην «κλάση1», η εκτέλεση της print ανατέθηκε στη μέθοδο print.κλάση1. Το ίδιο έγινε και στο b, καθώς η πρώτη κλάση στην οποία ανήκει είναι επίσης η «κλάση1». Το επόμενο είναι ορισμός μιας μεθόδου χειρισμού της summary για αντικείμενα της «κλάση2»:

```
summary.κλάση2<-function(x,...) {
    cat("Συνοψίζω αντικείμενο κλάσης 2,\n")
    cat("το άθροισμά του είναι",sum(b),".\n") }
```

Αν κληθεί η summary:

```
> summary(b)
Συνοψίζω αντικείμενο κλάσης 2,
το άθροισμά του είναι 20 .

> summary(a)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0       0       0       0       0       0
```

Καθώς στην πρώτη κλάση του b («κλάση1») δεν έχει οριστεί μέθοδος χειρισμού της summary, εντοπίστηκε και εκτελέστηκε η σχετική μέθοδος που παρέχεται από την επόμενη κλάση του αντικειμένου, την «κλάση2». Για τον λόγο αυτό το b κληρονομεί μεθόδους και από την «κλάση1» και από την «κλάση2». Όσο για το αντικείμενο a, το οποίο ανήκει μόνο στην «κλάση1», στη περίπτωση αυτή δεν βρέθηκε μέθοδος χειρισμού της summary οπότε κλήθηκε η summary.default. Το ίδιο αποτέλεσμα θα είχε κάποια κλήση σε

[348] Αυτό ονομάζεται overloading της συνάρτησης αυτής.
[349] Παρέχεται από το προ-εγκατεστημένο πακέτο 'utils'.
[350] Για τη λειτουργία της ειδικής παραμέτρου '…' βλ.§5.1.2 Παράμετροι.

οποιασδήποτε κλάσης αντικείμενο, αν αφαιρεθεί η ιδιότητα της κλάσης από αυτό, κάτι που κάνει η συνάρτηση **unclass**:

```
> summary(unclass(b))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
     20      20      20      20      20      20 
```

Εύκολα μπορεί δημιουργηθεί μια νέα generic συνάρτηση για την οποία θα μπορούν να ορίσουν μεθόδους οι κλάσεις S3, αρκεί η συνάρτηση αυτή να κάνει κλήση της UseMethod. Π.χ. εδώ ορίζεται μια generic συνάρτηση με όνομα κάνε_κάτι_με:

```
κάνε_κάτι_με <- function(x,...) UseMethod("κάνε_κάτι_με")
```

Αυτό επιτρέπει σε S3 κλάσεις να ορίσουν μεθόδους χειρισμού της, όπως:

```
κάνε_κάτι_με.κλάση2<-function(x,...)
    {cat("Κάνω κάτι με αντικείμενο κλάσης 2\n")}
```

Η παραπάνω θα καλείται για αντικείμενα της «κλάση2»:

```
> κάνε_κάτι_με(b)
Κάνω κάτι με αντικείμενο κλάσης 2
```

Προφανώς, για να ορίσουν και άλλες κλάσεις S3 μεθόδους χειρισμού της generic συνάρτησης «κάνε_κάτι_με» πρέπει να είναι γνωστή η ύπαρξή της. Για τον απλό αυτό λόγο, οι συνήθεις generic συναρτήσεις στις οποίες γίνεται ορισμός μεθόδων χειρισμού είναι αυτές που έρχονται σε προ-εγκατεστημένα ή άλλα ευρέως χρησιμοποιούμενα πακέτα ή σε κώδικα που αναπτύσσεται από τους ίδιους ή συνεργαζόμενους δημιουργούς. Αναλυτικότερη περιγραφή του τρόπου εντοπισμού της κατάλληλης μεθόδου (method dispatch) για μια generic συνάρτηση στις κλάσεις S3 (συμπεριλαμβανομένων και αυτών που ίσως περιέχονται σε πακέτα), αλλά και της συνάρτησης **NextMethod** για τη δημιουργία κλήσεων ανάμεσα σε μεθόδους διαφορετικών κλάσεων υπάρχουν στο [33].

Όπως αναφέρθηκε ήδη (και φαίνεται και από τα παραδείγματα που προηγήθηκαν) στο σύστημα S3 δεν γίνεται ορισμός των δεδομένων που έχουν τα αντικείμενα μιας κλάσης και το αντικείμενο μπορεί να είναι οτιδήποτε. Όμως, οι δημιουργοί της κλάσης μπορούν να βοηθήσουν τους χρήστες της κλάσης να δημιουργήσουν αντικείμενα κατάλληλα για τον σκοπό της κλάσης, παρέχοντας μερικές βοηθητικές συναρτήσεις δημιουργίας και ελέγχου των αντικειμένων της κλάσης αυτής. Το επόμενο παράδειγμα αφορά μια κλάση με όνομα «βιβλίο». Ας θεωρήσουμε πως στα αντικείμενα της κλάσης θέλουμε τα δεδομένα να είναι ένα list όπου καταγράφεται ο τίτλος (title), οι συγγραφείς (authors) και οι σελίδες (pages) του κάθε αντικειμένου κλάσης «βιβλίο». Ίσως λοιπόν βοηθούσε τους χρήστες της κλάσης να υπάρχει κάποια συνάρτηση δημιουργίας τέτοιων αντικειμένων (constructor) όπως π.χ. η παρακάτω:

```
βιβλίο <- function(title, authors, pages)
{
  stopifnot(is.character(title) &&
            is.character(authors) &&
            is.numeric(pages))

  νέο_βιβλίο  <- list(title = title[1],
                      authors = authors,
                      pages = as.integer(pages[1]))

  class(νέο_βιβλίο) <- "βιβλίο"
  return(νέο_βιβλίο)
}
```

Η συνάρτηση ζητά τρεις παραμέτρους, ελέγχει ότι πληρούν τα βασικά κριτήρια (π.χ. ο τίτλος και ο συγγραφέας να είναι κείμενο, ενώ οι σελίδες να είναι αριθμός), δημιουργεί το αντικείμενο (εδώ list) με τα δεδομένα αυτά, του προσθέτει την ιδιότητα της κλάσης «βιβλίο» και το επιστρέφει. Έτσι αντικείμενα της κλάσης μπορούν να δημιουργηθούν όπως παρακάτω:

```
b1 <- βιβλίο("Οδύσσεια","Όμηρος",126)
b2 <- βιβλίο("Compilers",c("Aho","Sethi","Ullman"),793)
```

Επιπροσθέτως, οι δημιουργοί της κλάσης μπορεί να επιλέξουν να παρέχουν και συνάρτησες μετατροπής (στα πρότυπα της οικογένειας συναρτήσεων as.[…]) ή ελέγχου (στα πρότυπα των is.[…]) αν και όχι απαραίτητα με αυτά τα ονόματα. Π.χ. μια συνάρτηση που κάνει κάποιους ελέγχους στη δομή ενός αντικειμένου για να διαπιστώσει αν αυτή ταιριάζει στην κλάση «βιβλίο», θα μπορούσε να είναι η παρακάτω:

```
is.βιβλίο <- function(x) {return(
                              inherits(x,"βιβλίο") &&
            all(c("title","authors","pages") %in% names(x)) &&
                              is.character(x$title) &&
                              is.character(x$authors) &&
                              is.numeric(x$pages) )}
```

Η συνάρτηση αυτή ελέγχει αν στις κλάσεις του αντικειμένου περιέχεται η κλάση «βιβλίο», αν στα ονόματα των στοιχείων του υπάρχουν τα αναμενόμενα title, authors και pages και αν αυτά είναι των αναμενόμενων τύπων (character για τα title και authors, numeric για το pages). Παρακάτω ορίζεται και μια μέθοδος χειρισμού της print για την κλάση:

```
print.βιβλίο <- function(x,...)
{
  if(is.βιβλίο(x))
  {
    txt <- sprintf("Βιβλίο %s, %d σελίδες, γραμμένο από %s",
                   x$title,
                   x$pages,
                   paste(x$authors,  collapse=", "))
    cat(txt)
    return(invisible(txt))
  }
  print(unclass(x))
}
```

Οπότε, αν χρησιμοποιηθούν οι συναρτήσεις αυτές σε αντικείμενα της κλάσης «βιβλίο», (όπως τα b1 και b2 που δημιουργήθηκαν παραπάνω) έχουμε το αποτέλεσμα:

```
> is.βιβλίο(b1)
[1] TRUE
> b1
Βιβλίο Οδύσσεια, 126 σελίδες, γραμμένο από Όμηρος
> print(b2)
Βιβλίο Compilers, 793 σελίδες, γραμμένο από Aho, Sethi, Ullman
```

Όμως πρέπει να τονιστεί ότι μια προσέγγιση δημιουργίας βοηθητικών συναρτήσεων (όπως έγινε εδώ με τις συναρτήσεις βιβλίο και is.βιβλίο για την κλάση «βιβλίο») ούτε επιβάλλεται ούτε αξιοποιείται αυτόματα από το σύστημα κλάσεων S3. Το σύστημα βασίζεται μόνο στην ιδιότητα class του αντικειμένου. Έτσι εξακολουθεί να μπορεί να οριστεί η «βιβλίο» ως κλάση οποιουδήποτε αντικειμένου:

```
b3 <- 1
class(b3)<- "βιβλίο"
```

Οπότε προκύπτει το παράδοξο:

```
> is.βιβλίο(b3)
[1] FALSE
```

ενώ

```
> isa(b3,"βιβλίο")
[1] TRUE
```

Η πρώτη εντολή αφορά κλήση κάποιας βοηθητικής συνάρτησης με όνομα is.βιβλίο που δημιουργήθηκε (παραπάνω) ώστε να μπορούν οι χρήστες της κλάσης να ελέγξουν αν το αντικείμενο έχει και την αναμενόμενη δομή. Η δεύτερη ελέγχει απλώς αν το αντικείμενο έχει το όνομα “βιβλίο” ως (μοναδική) ιδιότητα class, αυτό δηλαδή που ενδιαφέρει το σύστημα S3 για να καθορίσει την κλάση του αντικειμένου. Άρα ενώ το b3 είναι αντικείμενο κλάσης «βιβλίο», δεν έχει τη δομή που θα ήθελαν οι δημιουργοί της κλάσης. Τέτοια προβλήματα προσπαθεί να θεραπεύσει το σύστημα κλάσεων S4 που περιγράφεται στην επόμενη ενότητα.

6.4 Κλάσεις S4

Οι κλάσεις S4 [17] [62] αποκαλούνται και Formal Classes[351]. Το σύστημα κλάσεων S4 είναι επίσης ενσωματωμένο με την R (όπως το S3) καθώς περιέχεται στο προ-εγκατεστημένο πακέτο 'methods'[352]. Είναι παρεμφερές με το σύστημα S3 (που περιγράφεται στην προηγούμενη ενότητα). Όπως το S3, το σύστημα S4 είναι προσαρμοσμένο στις ιδιαιτερότητες της R: οι κλάσεις ορίζονται δυναμικά κατά τον χρόνο εκτέλεσης του κώδικα, ενώ επίσης στη φάση εκτέλεσης οριστικοποιείται και ο τύπος των αντικειμένων στα οποία γίνεται επεξεργασία. Όπως και το S3, στόχος του S4 είναι να μην παραβιάζει (κατά το δυνατόν) τις αρχές του συναρτησιακού προγραμματιστικού μοντέλου. Σε σχέση πάντως με το S3, το S4 γίνεται πιο αυστηρό, παρέχει περισσότερους μηχανισμούς χειρισμού κλάσεων και αντικειμένων καθώς και τη χρήση multiple dispatch για την επιλογή της μεθόδου που θα εκτελεστεί (περισσότερα για αυτό παρακάτω).

Στο σύστημα S4 κάθε νέα κλάση μπορεί να κληρονομεί μία ή περισσότερες άλλες κλάσεις. Αυτές θεωρούνται βάση (base class) της νέας κλάσης, ενώ η τελευταία ονομάζεται υποκλάση (subclass) αυτών. Οι υποκλάσεις, προσθέτουν τις λειτουργίες που αφορούν μόνο αυτές ή χρησιμοποιούν όσες λειτουργίες παρέχουν οι κλάσεις βάσης, επεκτείνοντας, προσθέτοντας ή τροποποιώντας όπου χρειάζεται.

Μια νέα κλάση του συστήματος S4 πρέπει να οριστεί με τη συνάρτηση **setClass** όπου δίνεται το όνομα της κλάσης αλλά και στοιχεία για τα slots, δηλαδή τις θέσεις που θα περιέχουν δεδομένα σε κάθε αντικείμενο της κλάσης αυτής[353]. Για παράδειγμα μπορεί να καταχωρηστεί μια νέα κλάση με όνομα «κλάση1» ως εξής[354]:

```
setClass("κλάση1")
```

Εδώ δόθηκε μόνο το όνομα της κλάσης και δεν ορίστηκαν slots. Έτσι δεν υπάρχει ορισμός των δεδομένων που διατηρεί κάθε αντικείμενο της κλάσης. Για μια τέτοια κλάση (χωρίς ορισμό slots) το S4 δεν μπορεί να δημιουργήσει αντικείμενα. Θεωρείται εικονική (virtual class), μια κλάση της οποίας ο βασικός ρόλος είναι να κληρονομηθεί από άλλες κλάσεις. Συνήθως η δημιουργία μιας virtual κλάσης δεν είναι το ζητούμενο, άρα στη setClass συνήθως ορίζονται τα slots - ονόματα και τύπος αντικειμένων - που θα περιέχεται σε κάθε αντικείμενο της κλάσης. Για παράδειγμα, το παρακάτω επαναπροσδιορίζει την «κλάση1» προσθέτοντας ότι σε κάθε αντικείμενο της κλάσης θα υπάρχει ένα numeric με όνομα d1:

```
setClass("κλάση1",slots = c(d1="numeric"))
```

Πλέον η «κλάση1» είναι αρκετά καθορισμένη ώστε να μπορούν να δημιουργηθούν αντικείμενα[355] της κλάσης. Αυτό γίνεται με τη συνάρτηση **new**:

```
a <- new("κλάση1")
a@d1 <- 10
```

Ο τελεστής **@** δίνει πρόσβαση σε ένα συγκεκριμένο slot του αντικειμένου. Έτσι οι παραπάνω εντολές δημιουργούν ένα αντικείμενο, μέλος της «κλάση1» με όνομα a και μετά αναθέτουν στο slot d1 του a την τιμή 10. Αυτό επιτρέπεται καθώς το d1 έχει οριστεί ως numeric για αντικείμενα της κλάσης αυτής[356]. Αν η τιμή ήταν αδύνατο να μετατραπεί σε numeric ώστε να ανατεθεί στο d1, αυτό θα προκαλούσε έγερση σφάλματος. Ταυτόσημο με τις δύο παραπάνω εντολές αποτέλεσμα έχει η εντολή:

```
a <- new("κλάση1", d1=10)
```

Άλλος τρόπος πρόσβασης στα slots είναι με τη συνάρτηση **slot**, ενώ τα ονόματα και τους τύπους των slot μιας κλάσης επιστρέφει (ως character διάνυσμα) η συνάρτηση **getSlots**:

Δοκιμάστε:	**Σχόλιο**
`slot(a,"d1")`	Επιστρέφει το slot d1 του a (εδώ 10).
`slot(a,"d1") <- 10`	Ανάθεση τιμής 10 στο slot d1 του a (ίδιο με a@d1 <- 10).
`getSlots("κλάση1")`	Επιστρέφει τα slots (εδώ ένα στοιχείο: "numeric" με όνομα d1).

[351] βλ. help(class) του πακέτου 'base'.

[352] Το πακέτο 'methods' παρέχει και τις συναρτήσεις που παρουσιάζονται σε αυτή την ενότητα.

[353] Σε άλλες αντικειμενοστραφείς γλώσσες ορίζονται οι μεταβλητές μελών (member variables ή instance variables) των αντικειμένων μιας κλάσης. Τα slots είναι παρεμφερή με αυτές τις μεταβλητές.

[354] Αν και οι στόχοι και τα ονόματα των κλάσεων στα παραδείγματα που ακολουθούν είναι παραπλήσιοι με αυτούς των παραδειγμάτων της §6.3 Κλάσεις S3, τα παραδείγματα δεν είναι απαραίτητο να ταυτίζονται.

[355] Στον αντικειμενοστραφή προγραμματισμό ένα συγκεκριμένο αντικείμενο που δημιουργείται βάσει μιας κλάσης ονομάζεται και instance, δηλαδή περίπτωση (αντικειμένου) της κλάσης. Για τη συνάρτηση new βλ. και §2.2.1 Δημιουργία, χρήση, μετατροπή μεταβλητών και αριθμητικά δεδομένα.

[356] Δεν υπάρχουν άλλοι περιορισμοί πρόσβασης όπως ιδιωτικά (private) ή προστατευμένα (protected) slots.

Καθώς κατά τον ορισμό μιας νέας κλάσης η setClass επιβάλει και τον ορισμό των slots των αντικειμένων της, δεν μπορεί να δημιουργηθεί νέο αντικείμενο της κλάσης αν δεν ακολουθεί αυτές τις προδιαγραφές. Η παρακάτω προσπάθεια δημιουργίας ενός αντικειμένου u της «κλάση1» αποτυχαίνει και εγείρει σφάλμα αφού η τιμή που δόθηκε για το slot d1 δεν είναι numeric (ούτε μπορεί να γίνει μετατροπή) όπως απαιτεί ο ορισμός της κλάσης αυτής:

```
> u <- new("κλάση1", d1="abc")
Error in validObject(.Object) :
  invalid class "κλάση1" object: invalid object for slot "d1"
in class "κλάση1": got class "character", should be or extend
class "numeric"
```

Σε κάποιες περιπτώσεις είναι χρήσιμο να επιβληθούν και πρόσθετοι έλεγχοι εγκυρότητας πριν δημιουργηθεί ένα νέο αντικείμενο. Η συνάρτηση **setValidity** μπορεί να χρησιμοποιηθεί για τους πρόσθετους αυτούς ελέγχους. Ας υποθέσουμε π.χ. ότι είναι ζητούμενο για τα αντικείμενα της «κλάσης1» να έχουν στο slot d1 όχι μόνο numeric τιμές, αλλά αυτές να έχουν μόνο ένα στοιχείο και η τιμή αυτού να είναι μεγαλύτερη ή ίση του 100:

```
setValidity("κλάση1",
            function(object) {
              if(length(object@d1)!=1)
                return("To d1 να έχει ένα στοιχείο")
              if(object@d1<100)
                return("To d1 πρέπει να είναι >= 100")
              return(TRUE)
            })
```

Η δεύτερη παράμετρος (method) της setValidity είναι μια συνάρτηση με μία μόνο παράμετρο (object). Η συνάρτηση αυτή θα κάνει τους ελέγχους στο object, επιστρέφοντας περιγραφή του προβλήματος ή TRUE αν το αντικείμενο πέρασε τους ελέγχους. Για παράδειγμα, το a που ορίστηκε παραπάνω δεν περνά τους ελέγχους καθώς η τιμή του d1 είναι 10 (άρα <100). Ο έλεγχος σε υπάρχοντα αντικείμενα γίνεται με τη συνάρτηση **validObject**:

```
> validObject(a)
Error in validObject(a) :
  invalid class "κλάση1" object: To d1 πρέπει να είναι >= 100
```

Άπαξ και οριστεί η setValidity μιας κλάσης δεν μπορούν να δημιουργηθούν πλέον νέα αντικείμενα (είτε της κλάσης, είτε κλάσεων που την κληρονομούν) αν δεν περνούν τους ελέγχους που εκτελεί η συνάρτηση αυτή. Έτσι η παρακάτω νέα προσπάθεια δημιουργίας ενός νέου αντικειμένου u θα αποτύχει και η αιτία περιγράφεται στο μήνυμα σφάλματος:

```
> u <- new("κλάση1", d1=c(10,20))
Error in validObject(.Object) :
  invalid class "κλάση1" object: To d1 να έχει ένα στοιχείο
```

Στις κλάσεις S4 τόσο τα slots όσο και η ίδια η πληροφορία της κλάσης καταχωρίζονται ως ιδιότητες του αντικειμένου[357], ενώ η ιδιότητα class περιέχει την πληροφορία της κλάσης. Είναι δηλαδή μια επέκταση της προσέγγισης που εφαρμόζεται και στο σύστημα S3. Έτσι, για το αντικείμενο a:

```
> mode(a)
[1] "S4"

> class(a)
[1] "κλάση1"
attr(,"package")
[1] ".GlobalEnv"
```

Από τα παραπάνω φαίνεται ότι το a είναι αντικείμενο κλάσης S4, συγκεκριμένα της "κλάση1" που ορίζεται στο Global Environment. Οι ιδιότητες του αντικειμένου περιλαμβάνουν κλάση και slots, άρα για το a είναι:

```
> attributes(a)
$d1
```

[357] βλ.§4.2.1.4 Η λίστα ιδιοτήτων των αντικειμένων.

```
[1] 10

$class
[1] "κλάση1"
attr(,"package")
[1] ".GlobalEnv"
```

Για μια κλάση ορίζονται μέθοδοι. Αυτό ισχύει ακόμα και αν η κλάση είναι εικονική, οπότε τις μεθόδους της θα τις κληρονομήσουν ή και θα τις αντικαταστήσουν (override) άλλες κλάσεις που βασίζονται στην κλάση αυτή ώστε να τις χρησιμοποιούν τα αντικείμενα τους. Όπως και στο σύστημα S3, οι μέθοδοι των κλάσεων S4 αφορούν generic συναρτήσεις. Οι μέθοδοι χειρίζονται την κλήση μιας generic συνάρτησης όταν αυτή αφορά αντικείμενα της κλάσης. Έτσι οι πιθανές μέθοδοι που ίσως υποστηρίζει μια κλάση S4 είναι κοινές για όλες τις κλάσεις S4. Η δημιουργία μιας μεθόδου χειρισμού κάποιας generic συνάρτησης γίνεται με τη συνάρτηση **setMethod**. Στο επόμενο παράδειγμα γίνεται ορισμός του χειρισμού της generic συνάρτησης print[358] για αντικείμενα της «κλάση1»:

```
setMethod("print","κλάση1",
          definition = function(x, ...)
          {cat("Εμφανίζω αντικείμενο κλάσης 1 με d1=",x@d1)},
          sealed = TRUE)
```

Στην παράμετρο definition καταχωρίζεται η ίδια η συνάρτηση (μέθοδος) που θα κάνει τον χειρισμό της print για τα αντικείμενα της «κλάση1». Οι παράμετροι της συνάρτησης αυτής πρέπει να ταιριάζουν με τις παραμέτρους της generic συνάρτησης που θα χειριστεί (εδώ δηλαδή της print). Η τιμή TRUE στην παράμετρο sealed της setMethod ορίζει ότι δεν μπορεί να αντικατασταθεί η μέθοδος αυτή (για τη συγκεκριμένη κλάση) από νέα εντολή setMethod που θα εκτελεστεί στη συνέχεια. Άρα εδώ η print είναι «σφραγισμένη» για την «κλαση1» κατά την τρέχουσα συνεδρία με την R [359]. Σε κάθε περίπτωση, ο παραπάνω ορισμός έχει ως αποτέλεσμα:

```
> print(a)
Εμφανίζω αντικείμενο κλάσης 1 με d1= 10
```

Τι γίνεται όμως αν κληθεί μια generic συνάρτηση ενώ δεν υπάρχει μέθοδος χειρισμού της στην κλάση; Για παράδειγμα, στην «κλάση1» δεν έχει οριστεί κάποια μέθοδος χειρισμού της generic συνάρτησης summary. Περιπτώσεις όπως αυτή έχουν ως αποτέλεσμα τον χειρισμό της generic συνάρτησης να τον αναλάβει μια προεπιλεγμένη default συνάρτηση που ισχύει για όλες τις κλάσεις S4:

```
> summary(a)
Length  Class   Mode
     1 κλάση1     S4
```

Όπως αναφέρθηκε ήδη, μια κλάση μπορεί να κληρονομεί άλλες. Αυτές ονομάζονται «βάση» της πρώτης, ή γράφεται ότι η κλάση είναι υποκλάση τους, τις επεκτείνει ή τις περιέχει. Μια τέτοια ιεράρχηση των κλάσεων μπορεί να έχει πολλά επίπεδα καθώς μια κλάση μπορεί να έχει βάση μια ή περισσότερες άλλες που με τη σειρά τους έχουν βάση μια ή περισσότερες άλλες κλάσεις, και ούτω καθεξής. Το παρακάτω ορίζει μια κλάση με όνομα «κλάση2» η οποία είναι υποκλάση της «κλάση1»[360]:

```
setClass("κλάση2",
         slots = c(d2="character"),
         contains = "κλάση1")
```

Η «κλάση2» έχει βάση την «κλάση1». Αυτό ορίστηκε στην παράμετρο contains της setClass. Καθώς μια κλάση μπορεί να είναι υποκλάση πολλών άλλων, στη ίδια παράμετρο contains θα μπορούσε να δοθεί character με περισσότερα του ενός στοιχεία, περιέχοντας όλα τα ονόματα των κλάσεων που η νέα κλάση κληρονομεί. Η νέα κλάση είναι επέκταση της βάσης της, άρα:

```
> extends("κλάση2","κλάση1")
[1] TRUE
```

Η δομή των αντικειμένων μιας κλάσης μπορεί να συνοψιστεί με τη συνάρτηση **showClass**:

---

[358] Για τις generic συναρτήσεις γενικότερα αλλά και τη generic συνάρτηση print βλ.§6.3 Κλάσεις S3.

[359] Όπως αναφέρει και η τεκμηρίωση (βλ. help(setMethod)) το «σφράγισμα» μιας μεθόδου συνήθως δεν ωφελεί τον σχεδιασμό των κλάσεων και καλό είναι να αποφεύγεται (παραλείποντας την παράμετρο sealed ή θέτοντας την τιμή της σε FALSE).

[360] Παρατηρήστε εδώ μια ακόμα διαφορά από το σύστημα S3. Στο S4 η ιεραρχία των κλάσεων (δηλαδή οι κλάσεις που κληρονομεί) ορίζεται στην κλάση και όχι στο αντικείμενο.

```
> showClass("κλάση2")
Class "κλάση2" [in ".GlobalEnv"]

Slots:

Name:          d2        d1
Class: character   numeric

Extends: "κλάση1"
```

Τα παραπάνω επιβεβαιώνουν ότι κάθε αντικείμενο της νέας κλάσης κληρονομεί slots και μεθόδους από τη βασική κλάση. Η επόμενη εντολή δημιουργεί ένα νέο αντικείμενο «κλάση2». Παρατηρήστε ότι δίνονται δεδομένα και για το slot d1 που αποτελεί μέρος του ορισμού των αντικειμένων «κλάση1»:

```
b <- new("κλάση2", d1=250, d2 = c("abc","def"))
```

Το αντικείμενο αυτό ανήκει στην «κλάση2» άρα και στην «κλάση1». Η συνάρτηση **is** ελέγχει αν ένα αντικείμενο κληρονομεί κάποια κλάση S4 (η συνάρτηση είναι παρεμφερής της inherits στις κλάσεις S3):

```
> is(b,"κλάση1")
[1] TRUE
> is(b,"κλάση2")
[1] TRUE
```

Προφανώς, νέα αντικείμενα της κλάσης μπορούν να δημιουργηθούν και με αντιγραφή υπαρχόντων μελών της κλάσης, ή με μετατροπή αντικειμένων άλλων κλάσεων. Ιδιαίτερα στη περίπτωση που η κλάση προορισμού βασίζεται στην κλάση προέλευσης (άρα την περιέχει), η μετατροπή είναι ιδιαιτέρως ασφαλής. Η επόμενη εντολή θα δημιουργήσει αντικείμενο c που ανήκει στην «κλάση1» με μετατροπή του b που ανήκει στην «κλάση2». Το c που θα δημιουργηθεί ανήκει στην «κλάση1» και έχει μόνο το slot d1 με τιμή 250.

```
c <- as(b, "κλάση1")
```

Επιστρέφοντας στο b, επειδή ως μέλος της «κλάση2» είναι και μέλος της «κλάση1», το αντικείμενο αυτό έχει πρόσβαση στην print που ορίστηκε παραπάνω για την κλάση αυτή:

```
> print(b)
Εμφανίζω αντικείμενο κλάσης 1 με d1= 250
> summary(b)
Length  Class   Mode
     1 κλάση2     S4
```

Καλώντας την print για το αντικείμενο b της «κλάσης2» χρησιμοποιήθηκε η σχετική μέθοδος της «κλάσης1». Για την περίπτωση της summary, καμία από τις κλάσεις που ανήκει το b δεν παρέχει μέθοδο χειρισμού της, οπότε έγινε χρήση του default τρόπου χειρισμού της που ισχύει για όλες τις κλάσεις S4. Βέβαια κάθε υποκλάση μπορεί να ορίζει τις δικές της μεθόδους, αντικαθιστώντας ή προσθέτοντας σε αυτές που παρέχουν οι κλάσεις στις οποίες βασίζεται. Παρακάτω αντικαθίσταται η print και προστίθεται χειρισμός της summary για την «κλάση2»:

```
setMethod("print","κλάση2",
          definition = function(x, ...)
          { cat("Εμφανίζω αντικείμενο κλάσης 2 με",
                x@d1,"και",x@d2,"\n") })

setMethod("summary","κλάση2",
          definition = function(object, ...) {
            cat("Συνοψίζω αντικείμενο κλάσης 2:\n")
            tl <- length(object@d1) + length(object@d2)
            cat("Δεδομένα:",tl,"\n")})
```

Με αυτούς τους ορισμούς το αποτέλεσμα για τα a (ή όποιο άλλο αντικείμενο ανήκει μόνο στην «κλαση1») δεν αλλάζει, καθώς δεν αφορά την κλάση αυτή. Αλλά για το b, που είναι «κλάση2»:

```
> print(b)
Εμφανίζω αντικείμενο κλάσης 2 με 250 και abc def
> summary(b)
Συνοψίζω αντικείμενο κλάσης 2:
Δεδομένα: 3
```

Η δημιουργία μιας νέας generic συνάρτησης (την οποία θα μπορούν να χειριστούν με μεθόδους τους όλες οι κλάσεις S4) γίνεται τη στιγμή που καταχωρίζεται κάποια μέθοδος χειρισμού της μέσω της setMethod. Εναλλακτικά, μια νέα συνάρτηση μπορεί να οριστεί ως generic ακόμα και αν δεν υπάρχει κλάση που παρέχει μεθόδους χειρισμού της. Αυτό γίνεται με τη συνάρτηση **setGeneric**:

```
setGeneric("κάνε_κάτι_με",
           function(object)
             { standardGeneric("κάνε_κάτι_με") })
```

Στο παράδειγμα αυτό δημιουργείται μια νέα generic συνάρτηση με όνομα κάνε_κάτι_με. Περιέχει (στη δεύτερη παράμετρό της, def) μια συνάρτηση με συγκεκριμένα ορίσματα η οποία απλώς καλεί τη **standardGeneric**. Αυτό είναι το μόνο που χρειάζεται να κάνει, καθώς ο πραγματικός χειρισμός μιας κλήσης της κάνε_κάτι_με θα γίνει από τις μεθόδους που θα παρέχουν οι κλάσεις (και θα οριστούν μέσω setMethod). Η νέα generic συνάρτηση ανήκει στην κλάση nonstandardGeneric και αν οι κλάσεις του αντικειμένου το οποίο καλείται να επεξεργαστεί δεν παρέχουν μέθοδο χειρισμού της, θα εγερθεί σφάλμα.

Άλλος τρόπος δημιουργίας μιας generic συνάρτησης είναι να μετατραπεί σε generic (ή συγκεκριμένα standardGeneric) μια υπάρχουσα. Αυτό γίνεται ως εξής:

```
κάνε_κάτι_με <- function(object)
  {cat("Δεν ξέρω τι να κάνω σε αντικείμενα",class(object))}

setGeneric("κάνε_κάτι_με")
```

Η πρώτη εντολή απλώς δημιουργεί μια συνάρτηση με όνομα κάνε_κάτι_με, ενώ ακολούθως η συνάρτηση αυτή καταχωρίζεται ως generic μέσω της συνάρτησης setGeneric. Η αρχική συνάρτηση γίνεται η default μέθοδος χειρισμού της κάνε_κάτι_με για τις κλάσεις S4. Το ίδιο αποτέλεσμα έχει η εντολή:

```
setGeneric("κάνε_κάτι_με", function(object)
 {cat("Δεν ξέρω τι να κάνω σε αντικείμενα",class(object))})
```

αλλά οι επιμέρους κλάσεις μπορούν να ορίσουν τις δικές τους σχετικές μεθόδους, πάντα με χρήση της συνάρτηση setMethod. Ένα παράδειγμα για την «κλάση2»:

```
setMethod("κάνε_κάτι_με", "κλάση2",
          definition = function(object)
          {cat("Κάνω κάτι με αντικείμενο κλάσης 2\n")})
```

Με την εντολή αυτή τα αντικείμενα κλάσης «κλάση2» αποκτούν μέθοδο χειρισμού της κάνε_κάτι_με. Οι υπόλοιποι τύποι αντικειμένων δεν έχουν. Αν κληθεί η κάνε_κάτι_με σε διάφορους τύπους αντικειμένων:

```
> κάνε_κάτι_με(100)
Δεν ξέρω τι να κάνω σε αντικείμενα numeric
> κάνε_κάτι_με(a)
Δεν ξέρω τι να κάνω σε αντικείμενα κλάση1
> κάνε_κάτι_με(b)
Κάνω κάτι με αντικείμενο κλάσης 2
```

Πολλές αντικειμενοστραφείς γλώσσες (όπως και το σύστημα S4 στην R) επιτρέπουν τη δημιουργία πολλαπλών παραλλαγών της ίδιας μεθόδου, κάτι που αποκαλείται πολυμορφισμός. Έτσι, στις περισσότερες από τις συνήθεις αντικειμενοστραφείς γλώσσες μπορούν να οριστούν μέθοδοι με το ίδιο όνομα (και πιθανότατα τον ίδιο σκοπό) που διαφέρουν όμως ως προς το είδος ή και τον αριθμό των παραμέτρων τους. Συνήθως στις γλώσσες αυτές ο τύπος κάθε παραμέτρου είναι γνωστός εκ των προτέρων, έτσι η επιλογή της κατάλληλης παραλλαγής (method dispatching) μπορεί να γίνει σε χρόνο πρότερο της εκτέλεσης του κώδικα, κατά τη φάση της μεταγλώττισης (compile time). Στην R, από την άλλη πλευρά, υπάρχει η ιδιαιτερότητα πως ο τύπος των αντικειμένων που δίνονται ως τιμές παραμέτρων μιας συνάρτησης οριστικοποιείται μόνο κατά τη φάση εκτέλεσης της εντολής. Παρόλα αυτά, το σύστημα S4 επιτρέπει πολλές πιθανές παραλλαγές κάποιας μεθόδου με το ίδιο όνομα και επιλέγει την πλέον κατάλληλη με βάση τον τύπο των παραμέτρων. Αυτό το είδαμε ήδη, καθώς η μέθοδος που καλείται για να χειριστεί π.χ. την print ή την κάνε_κάτι_με παραπάνω επιλέχτηκε με βάση την κλάση του αντικειμένου που δόθηκε ως όρισμα. Κατά τον ορισμό μιας μεθόδου με τη setMethod, η δεύτερη παράμετρος (signature) είναι ουσιαστικά μια οδηγία σχετικά με το πότε να κληθεί αυτή η συγκεκριμένη μέθοδος. Η οδηγία αυτή ονομάζεται και υπογραφή της μεθόδου. Π.χ. στη setMethod του αμέσως προηγούμενου παραδείγματος, στην παράμετρο signature δόθηκε η τιμή "κλάση2". Άρα η συγκεκριμένη μέθοδος θα χειριστεί την κάνε_κατι_με αν το πρώτο όρισμα κατά την κλήση της είναι αντικείμενο της «κλάση2».

Κάθε υπογραφή μπορεί να περιέχει πολλά στοιχεία, έναν συνδυασμό τύπων αντικειμένων. Το σύστημα S4 εφαρμόζει μια τεχνική ταιριάσματος των αντικειμένων που δόθηκαν ως παράμετροι κατά την κλήση με τις

διάφορες υπογραφές των μεθόδων που μπορεί να τη χειριστούν, ώστε να επιλέξει την πλέον κατάλληλη. Έτσι, η τελική επιλογή είναι βασισμένη όχι μόνο σε ένα αλλά σε όλα τα αντικείμενα που δίνονται ως τιμές παραμέτρων (αυτό αναφέρεται ως multiple dispatch). Για κάθε διαθέσιμη μέθοδο υπολογίζεται ένα «κόστος» (ή «απόσταση») με βάση τη σχέση των τύπων στην υπογραφή της με αυτά που έχουν δοθεί ως παράμετροι κατά την κλήση της generic συνάρτησης. Κάθε πιθανή ασυνέπεια ανάμεσα στους τύπους αυτούς προσθέτει στο «κόστος», όπως «κοστίζει» και κάθε πρόσβαση σε μεθόδους που ανήκουν σε κλάσεις ψηλότερα στην ιεραρχία, βάσεις δηλαδή των κλάσεων των αντικειμένων. Τελικά επιλέγεται η μέθοδος με το ελάχιστο «κόστος», π.χ. αν υπάρχει μέθοδος της οποίας η υπογραφή ταιριάζει απόλυτα με τους τύπους των παραμέτρων, το «κόστος» είναι 0 και η μέθοδος αυτή επιλέγεται να χειριστεί την κλήση.

Ακολουθεί ένα παράδειγμα για μια νέα generic συνάρτηση με όνομα παίξε_με. Το παράδειγμα είναι συνέχεια των προηγούμενων στην ενότητα αυτή, οπότε υπενθυμίζεται ότι η «κλάση2» έχει βάση την «κλάση1» ενώ τα αντικείμενα a και b ανήκουν στις «κλάση1» και «κλάση2», αντίστοιχα. Με την παρακάτω εντολή ορίζεται η νέα generic συνάρτηση η οποία δέχεται δύο παραμέτρους x και y και για την οποία δίνεται και ένας default τρόπος χειρισμού:

```
setGeneric("παίξε_με",function(x,y){"Default χειρισμός"})
```

Ακολουθούν μερικοί ορισμοί μεθόδων χειρισμού της generic συνάρτησης παίξε_με:

```
setMethod("παίξε_με",
          signature=c("κλάση1","ANY"),
          definition = function(x,y)
                       {"παίζω με κλάση1 σε 1η παράμετρο"})

setMethod("παίξε_με",
          signature=c("ANY","κλάση1"),
          definition = function(x,y)
                       {"παίζω με οτιδήποτε και κλάση1 στη 2η"})

setMethod("παίξε_με",
          signature=c("missing","κλάση1"),
          definition = function(x,y)
                      {"παίζω με μόνο κλάση1 στη 2η παράμετρο"})
```

Ο πρώτος ορισμός έχει υπογραφή ("κλάση1","ANY"). Το "ANY" είναι μια ειδική τιμή για τις υπογραφές και θα ταιριάξει με οποιονδήποτε τύπο δοθεί στη δεύτερη παράμετρο y εφόσον δεν υπάρχει κάτι ποιο συγκεκριμένο σε υπογραφές άλλων μεθόδων. Εδώ, αυτό είναι αντίστοιχο με μια signature="κλάση1". Η δεύτερη μέθοδος έχει "ANY" στην πρώτη θέση της υπογραφής της, άρα θα ταιριάξει με οποιονδήποτε τύπο δοθεί ως πρώτη παράμετρος x. Ο τρίτος ορισμός μεθόδου χρησιμοποιεί την ειδική τιμή "missing" που θα ταιριάξει όταν δεν έχει δοθεί τιμή στη συγκεκριμένη παράμετρο, δηλαδή εδώ στην πρώτη παράμετρο x). Ακολουθούν μερικοί ακόμα ορισμοί παραλλαγών της μεθόδου:

```
setMethod("παίξε_με",
          signature=c("κλάση1","κλάση2"),
          definition = function(x,y)
                       {"παίζω με κλάση1 και κλάση2"})

setMethod("παίξε_με",
          signature=c("κλάση1","numeric"),
          definition = function(x,y)
                       {"παίζω με κλάση1 και numeric"})

setMethod("παίξε_με",
          signature=c("logical","κλάση1"),
          definition = function(x,y)
                       {"παίζω με logical και κλάση1"})

setMethod("παίξε_με",
          signature=c("κλάση2","ANY"),
          definition = function(x,y){"παίζω με μόνο κλάση2"})
```

```
setMethod("παίξε_με",
         signature=c("κλάση2","numeric"),
         definition = function(x,y)
                     {"παίζω με κλάση2 και numeric"})
```
Ας δούμε το αποτέλεσμα:
```
> παίξε_με(1,2)
[1] "Default χειρισμός"
> παίξε_με(a)
[1] "παίζω με κλάση1 σε 1η παράμετρο"
> παίξε_με("aabbcc", a)
[1] "παίζω με οτιδήποτε και κλάση1"
> παίξε_με(y=a)
[1] "παίζω με μόνο κλάση1 στη 2η παράμετρο"
> παίξε_με(a,b)
[1] "παίζω με κλάση1 και κλάση2"
```
Όμως το παρακάτω συνοδεύεται από διαγνωστικό μήνυμα πιθανού προβλήματος:
```
> παίξε_με(b,a)
Note: method with signature 'κλάση2#ANY' chosen for function
'παίξε_με',
 target signature 'κλάση2#κλάση1'.
 "ANY#κλάση1" would also be valid
[1] "παίζω με μόνο κλάση2"
```
Το πρόβλημα προέκυψε καθώς κατά την αναζήτηση της πλέον ταιριαστής από τις διαθέσιμες μεθόδους χειρισμού της παίξε_με, υπήρξαν πέραν τις μιας πιθανές επιλογές με ισότιμο «κόστος». Έτσι έπρεπε να γίνει επιλογή από τη μέθοδο με υπογραφή c("κλάση2","ANY") ή αυτή με c("ANY","κλάση1") και τελικά επιλέχτηκε η πρώτη. Συνεχίζοντας το παράδειγμα:
```
> παίξε_με(a,2)
[1] "παίζω με κλάση1 και numeric"
> παίξε_με(1<2,a)
[1] "παίζω με logical και κλάση1"
> παίξε_με(b)
[1] "παίζω με μόνο κλάση2"
> παίξε_με(b,1:10)
[1] "παίζω με κλάση2 και numeric"
> παίξε_με("abcd",b)
[1] "παίζω με οτιδήποτε και κλάση1"
```
Στο προτελευταίο παράδειγμα (παίξε_με(b,1:10)) πρέπει να παρατηρηθεί ότι για το ταίριασμα έγινε μετατροπή από integer σε numeric, ενώ στο τελευταίο (παίξε_με("abcd",b)) η αναζήτηση δεν βρήκε υπογραφή μεθόδου που να ταιριάζει (character στην πρώτη παράμετρο και αντικείμενο της «κλάση2» στη δεύτερη) οπότε προχώρησε για το αντικείμενο της δεύτερης παραμέτρου σε αναζήτηση μεθόδων της γονικής κλάσης «κλάση1».

Η τεκμηρίωση του πακέτου 'methods' είναι εκτενής και καλύπτει πολλές τεχνικές λεπτομέρειες που αξίζει να συμβουλευτεί οποιοσδήποτε θέλει να εμβαθύνει στο σύστημα κλάσεων S4. Πληροφορία μπορεί να αντλήσει κανείς και από τον πηγαίο κώδικα του μεγάλου αριθμού πακέτων που χρησιμοποιούν το σύστημα S4 καθώς και από πολλές άλλες πηγές, μεταξύ αυτών τα [33], [60] και [68], όπου περιγράφονται διάφορα θέματα σχετικά με το multiple dispatch, ζητήματα ενσωμάτωσης των κλάσεων S4 σε πακέτα κ.α.

6.5 Κλάσεις αναφοράς

Τα δύο συστήματα κλάσεων RS (ή RC/R5) και R6 που περιγράφονται στην ενότητα αυτή έχουν πολλά κοινά σημεία τόσο μεταξύ τους όσο και με αντικειμενοστραφή συστήματα που βρίσκουμε σε άλλες συνήθεις γλώσσες (όπως οι C++, C#, Java κλπ.). Βασικό κοινό σημείο των δύο συστημάτων RS και R6 είναι ο τρόπος με τον οποίο γίνεται χειρισμός των αντικειμένων που ανήκουν στις κλάσεις τους (ανάθεση τους σε μεταβλητές, τροποποίηση των στοιχείων κλπ.). Συγκεκριμένα, στα αντικείμενα που δημιουργούνται από κλάσεις RS και R6

ο χειρισμός των μελών τους γίνεται με αναφορά (reference) σε αυτά. Απλουστεύοντας την περιγραφή του μηχανισμού, όταν ανατίθεται ένα αντικείμενο κλάσης RS ή R6 σε μια μεταβλητή, η μεταβλητή δεν θεωρείται ότι κρατά το αντικείμενο αλλά πληροφορίες για τη θέση του στη μνήμη, μια αναφορά στο αντικείμενο. Έτσι, όταν αντιγράφεται η μεταβλητή αυτή σε άλλη, δεν δημιουργείται νέο αντίγραφο του αντικειμένου (όπως συνήθως συμβαίνει στην R[361]) αλλά αντιγράφεται η αναφορά, η πληροφορία θέσης του αντικειμένου. Ως αποτέλεσμα, η δεύτερη μεταβλητή έχει την ίδια αναφορά προς το αντικείμενο με την πρώτη άρα έχει την ίδια πρόσβαση και στο ίδιο αντικείμενο με την πρώτη. Το ίδιο συμβαίνει όταν η αναφορά περαστεί ως τιμή παραμέτρου κάποιας συνάρτησης. Ο κώδικας στο σώμα της συνάρτησης έχει πρόσβαση στο αρχικό αντικείμενο άρα μπορεί να κάνει αλλαγές σε αυτό (και όχι σε κάποιο αντίγραφό του).

Αυτό είναι μια παραβίαση των οδηγιών του συναρτησιακού προγραμματισμού[362] καθώς επιτρέπει αλλαγές σε αντικείμενα εξωτερικά των συναρτήσεων, οπότε προάγει και διευκολύνει τη δημιουργία παρενεργειών (side-effect) από τις συναρτήσεις. Η παραβίαση όμως αυτή δίνει λύσεις σε πολλές περιπτώσεις όπου η κλασική προσέγγιση της R (αντιγραφή του αντικειμένου σε κάθε αλλαγή του, επίδραση προς το περιβάλλον μιας συνάρτησης μόνο μέσω της επιστρεφόμενης τιμής της κλπ.) είναι μη αποδοτική. Μια ακόμα βασική διαφορά των RS και R6 με τα συστήματα S3 και S4 είναι ότι οι μέθοδοι των κλάσεων δεν σχετίζονται με κάποια generic συνάρτηση. Στα συστήματα που περιγράφονται σε αυτή την ενότητα, οι μέθοδοι είναι συναρτήσεις που ανήκουν στην κλάση και μπορούν να εφαρμοστούν απευθείας πάνω στα αντικείμενα-μέλη της.

Συνοψίζοντας, τα αντικείμενα που βασίζονται σε κλάσεις RS και R6 επιτρέπουν απευθείας αλλαγές στην κατάσταση τους (στα στοιχεία τους) από πολλές μεταβλητές χωρίς το αποτέλεσμα να αποθηκεύεται σε κάποιο νέο αντίγραφό τους. Εκτός από τις κλάσεις που αναφέρονται στην ενότητα αυτή, τέτοιο τρόπο λειτουργίας είχαν και κάποιες άλλες ειδικές περιπτώσεις τύπων αντικειμένων που έχουν ήδη αναφερθεί, όπως τα environment[363] και data.table[364].

6.5.1 Κλάσεις RS

Το σύστημα κλάσεων RS (Reference System) παρέχεται από τα προ-εγκατεστημένα πακέτα της R και άρα είναι μέρος της γλώσσας αυτής. Το σύστημα RS αναφέρεται και ως Reference Classes (RC), refclasses, ή από διάφορες πηγές και ως R5. Το RS υποστηρίζεται από συναρτήσεις του πακέτου ‘methods’, όπως το σύστημα κλάσεων S4. Όμως το RS πλησιάζει περισσότερο από τα S3 και S4 τα αντικειμενοστραφή συστήματα άλλων γλωσσών προγραμματισμού.

Μια νέα κλάση RS ορίζεται με τη συνάρτηση **setRefClass** όπου καταχωρούνται οι μεταβλητές των μελών της κλάσης (τα στοιχεία που θα διατηρούν, αντίστοιχα με τα slots των κλάσεων S4) που εδώ ονομάζονται και πεδία (fields). Επίσης, ορίζονται (μεταξύ άλλων) οι μέθοδοι της κλάσης και οι κλάσεις που αποτελούν βάση της, θέματα που περιγράφονται παρακάτω. Η επόμενη εντολή ορίζει μια RS κλάση με όνομα «κλάση1» και δύο μεταβλητές που θα διατηρούν τα μέλη της κλάσης. Για το παράδειγμα[365], επιλέχθηκαν να έχουν όνομα d1a και d1b και να είναι και τα δύο τύπου numeric:

```
κλάση1 <- setRefClass("κλάση1",
                      fields = list(d1a = "numeric",
                                    d1b = "numeric"))
```

Το αποτέλεσμα της setRefClass (που στο παράδειγμα αποθηκεύτηκε σε μεταβλητή με όνομα κλαση1) είναι μια συνάρτηση γεννήτορας (αντικείμενο τύπου refObjectGenerator) η οποία, όταν κληθεί, δημιουργεί ένα αντικείμενο της συγκεκριμένης κλάσης και επιστρέφει τη θέση του (την αναφορά σε αυτό). Έτσι, για να καταχωριστεί στη μεταβλητή a (η αναφορά σε) ένα νέο αντικείμενο της «κλάση1» με δεδομένα 1 και 2 στα πεδία d1a και d1b αντίστοιχα, αρκεί μια εντολή όπως:

```
a <- κλάση1(d1a=1, d1b=2)
```

---

[361] Ακριβέστερα, το αντίγραφο θα πραγματοποιηθεί όταν γίνει προσπάθεια αλλαγής του αντικειμένου, λόγω της στρατηγικής copy-on-modify (βλ.§3.1.4 Εργαλεία προφίλ (profiling)).

[362] βλ.§5.3 Συναρτησιακός προγραμματισμός.

[363] βλ.§4.2.2.2 Περιβάλλοντα και ο μηχανισμός αναφοράς.

[364] βλ.§4.3.1 Οι τύποι tibble και data.table.

[365] Αν και οι στόχοι και τα ονόματα των κλάσεων στα παραδείγματα που ακολουθούν είναι παραπλήσιοι με αυτούς των παραδειγμάτων για τις κλάσεις S3 και S4, τα παραδείγματα δεν είναι απαραίτητο να ταυτίζονται.

Πρόσβαση στα πεδία του αντικειμένου γίνεται με τον τελεστή $ και το όνομά τους[366]. Για παράδειγμα:

Δοκιμάστε:	Σχόλιο
`a$d1a`	Επιστρέφει πεδίο d1a του a (εδώ επιστρέφει 1).
`a$d1a <- 10`	Ανάθεση τιμής 10 στο πεδίο d1a του a.

Αν μετά τα παραπάνω, εκτυπωθεί το a εμφανίζεται:

```
> a
Reference class object of class "κλάση1"
Field "d1a":
[1] 10
Field "d1b":
[1] 2
```

Τι γίνεται όμως αν ανατεθεί το a σε άλλη μεταβλητή; Επειδή αυτό που αντιγράφεται είναι η αναφορά στο αντικείμενο, οι δύο μεταβλητές θα αναφέρονται το ίδιο αντικείμενο στη μνήμη:

Δοκιμάστε:	Σχόλιο
`u<-a`	Αντιγράφει στο u την αναφορά του a σε αντικείμενο κλάσης «κλάση1».
`u`	Επιστρέφει το u, ταυτόσημο αποτέλεσμα με αυτό του a, καθώς αφορά το ίδιο αντικείμενο.
`u$d1a <- 100`	Ανάθεση τιμής 100 στο πεδίο d1a του u (άρα και του a).

Μετά τα παραπάνω είτε εκτυπωθεί το u είτε εκτυπωθεί το a, θα εμφανιστεί το κοινό τους αντικείμενο με την αλλαγμένη τιμή στο d1a:

```
> a
Reference class object of class "κλάση1"
Field "d1a":
[1] 100
Field "d1b":
[1] 2
```

Παρόμοια συμπεριφορά (και δυνατότητες) προκύπτουν όταν αναφορές σε αντικείμενα κάποιας κλάσης RS περνούν ως τιμές παραμέτρων μιας συνάρτησης:

Δοκιμάστε:	Σχόλιο
`f<-function(x) { x$d1b<-200 }`	Μια συνάρτηση f που δέχεται ένα x και αλλάζει το στοιχείο του d1b.
`f(a)`	Κλήση της f με παράμετρο το a.
`a`	Ανάκληση του a, παρατηρήστε ότι το d1b έχει πλέον τη τιμή 200.
`u`	Ανάκληση του u, ίδιο αντικείμενο με το a, άρα το d1b είναι 200.

Έτσι, τα αντικείμενα κλάσεων RS συμπεριφέρονται με τρόπο παρόμοιο όσων αναφέρθηκαν αναλυτικά για τα αντικείμενα τύπου environment στην §4.2.2.2 Περιβάλλοντα και ο μηχανισμός αναφοράς.

Κατά τον ορισμό μιας κλάσης RS με τη setRefClass καταγράφονται οι μέθοδοι που υποστηρίζει καθώς και οι κλάσεις που αποτελούν βάση της. Η νέα κλάση κληρονομεί από τις βάσεις της πεδία αλλά και μεθόδους τις οποίες μπορεί να χρησιμοποιήσει ως έχουν ή και να υποκαταστήσει (override) με δικές της. Οι κλάσεις-βάσεις ορίζονται σε διάνυσμα ονομάτων στην παράμετρο contains (με τρόπο παρόμοιο αυτού των κλάσεων S4), ενώ οι μέθοδοι δίνονται ως αντικείμενο list στην παράμετρο methods. Εδώ υπάρχει μια βασική διαφορά με τα συστήματα S3 και S4: οι μέθοδοι δεν σχετίζονται με κάποια generic συνάρτηση, είναι δηλαδή συναρτήσεις που μπορούν να εφαρμοστούν απευθείας πάνω στα αντικείμενα-μέλη της κλάσης. Επιπροσθέτως, όλες οι κλάσεις RS κληρονομούν αυτόματα την κλάση envRefClass η οποία παρέχει διάφορες κοινές μεθόδους για τον βασικό χειρισμό των αντικειμένων RS. Αν ανακαλέσουμε το «κλάση1» αυτό καταγράφεται ως εξής:

```
> κλάση1
Generator for class "κλάση1":

Class fields:
```

[366] Όπως στα στοιχεία ενός environment ή ενός list, βλ. και §4.2.1.1 Ο τελεστής επιλογής $.

```
Name:       d1a     d1b
Class: numeric numeric

Class Methods:
     "field","trace","getRefClass","initFields","copy",
     "callSuper",".objectPackage","export","untrace",
     "getClass","show","usingMethods",".objectParent",
     "import"

Reference Superclasses:
     "envRefClass"
```

Έτσι, αν και κατά τη δημιουργία της «κλάση1» που έγινε παραπάνω δεν έχουν οριστεί μέθοδοι, η κλάση κληρονόμησε (ως υποκλάση της envRefClass) διάφορες σημαντικές μεθόδους (αναφέρονται κάτω από το Class Methods). Μεταξύ αυτών περιλαμβάνεται η μέθοδος **show** που καλείται από τη generic συνάρτηση print για να εμφανίσει αντικείμενα κλάσεων RS:

Δοκιμάστε:	**Σχόλιο**
`a$show()`	Εμφανίζει τα περιεχόμενα του αντικειμένου a.
`print(a)`	Ίδιο με το παραπάνω, η print καλεί τη μέθοδο show στο αντικείμενο.

Η παραπάνω εμφάνιση έγινε χρησιμοποιώντας τον γενικό (default) τρόπο εμφάνισης αντικειμένων RS. Προφανώς μια κλάση μπορεί να ορίσει τη δική της μέθοδο show ώστε η εκτύπωση να προσαρμοστεί στις ιδιαιτερότητες της κλάσης αυτής. Άλλη συνάρτηση που κληρονομεί η «κλάση1» είναι η **copy**. Αυτή επιτρέπει τη δημιουργία ενός αντιγράφου του αντικειμένου, ώστε πλέον να υπάρχουν δύο ανεξάρτητα αντικείμενα, π.χ.:

Δοκιμάστε:	**Σχόλιο**
`c<-a$copy()`	Αναθέτει στο c (αναφορά σε) νέο αντικείμενο, αντίγραφο του a.
`c$d1a <- 999`	Ανάθεση τιμής 999 στο πεδίο d1a του c.
`c`	Ανάκληση του c, το d1a έχει τιμή 999, το d1b έχει τιμή 200.
`a`	Ανάκληση του c, το d1a έχει τιμή 100, το d1b έχει τιμή 200.

Παρατηρήστε ότι η πρόσβαση στις μεθόδους του αντικειμένου γίνεται με τον τελεστή $, όπως συμβαίνει και για πρόσβαση στα πεδία. Στο επόμενο παράδειγμα δημιουργείται μια νέα κλάση αντικειμένων με όνομα «κλάση2» η οποία είναι υποκλάση της «κλαση1», προσθέτει όμως ένα ακόμα numeric πεδίο d2 καθώς και δύο μεθόδους με όνομα πρόσθεσε_όλα_τα_πεδία και άλλαξε_όλα_τα_πεδία_σε:

```
κλάση2 <- setRefClass(
  "κλάση2",
  fields = list(d2 = "numeric"),
  contains = "κλάση1",
  methods = list(

    πρόσθεσε_όλα_τα_πεδία  = function()
    {
      return(sum(d1a, d1b, d2))
    },

    άλλαξε_όλα_τα_πεδία_σε  = function(x)
    {
      if (!is.numeric(x))
        return(FALSE)
      d1a <<- x
      d1b <<- x
      d2  <<- x
      return(TRUE)
    }
```

```
    ))
```

Η μέθοδος πρόσθεσε_όλα_τα_πεδία αθροίζει τα πεδία d1a, d1b και d2. Τα δύο πρώτα προέρχονται από την «κλάση1» αφού, εκτός των δικών του πεδίων και μεθόδων, ένα αντικείμενο της «κλάση2» κληρονομεί πεδία και μεθόδους από τις κλάσεις που έχει ως βάση, δηλαδή τις κλάσεις «κλάση1» και envRefClass. Η μέθοδος άλλαξε_όλα_τα_πεδία_σε αλλάζει τα d1a, d1b και d2 στην τιμή που δίνεται στην παράμετρό της x (αφού ελέγξει ότι είναι όντως numeric). Για την ανάθεση τιμής στα πεδία πρέπει να χρησιμοποιηθεί ο τελεστής <<- ή ο τελεστής ->> καθώς τα πεδία του αντικειμένου είναι εκτός του τοπικού περιβάλλοντος των μεθόδων[367].

Ας τα δούμε όλα αυτά στην πράξη. Η επόμενη εντολή δημιουργεί ένα αντικείμενο «κλάση2» με συγκεκριμένες τιμές στα πεδία του:

```
b <- κλάση2(d1a=1,d1b=2,d2=100)
```

Ενώ οι παρακάτω εντολές χειρίζονται το νέο αντικείμενο μέσω κάποιων μεθόδων του:

```
> b$πρόσθεσε_όλα_τα_πεδία()
[1] 103
> b$άλλαξε_όλα_τα_πεδία_σε(30)
[1] TRUE
> b$show()
Reference class object of class "κλάση2"
Field "d1a":
[1] 30
Field "d1b":
[1] 30
Field "d2":
[1] 30
> b$πρόσθεσε_όλα_τα_πεδία()
[1] 90
```

Το πακέτο 'methods' παρέχει εκτενή τεκμηρίωση για το σύστημα κλάσεων RS (π.χ. βλ. help(ReferenceClasses)), όπως σχετικές πληροφορίες παρέχουν και πολλές άλλες πηγές, μεταξύ αυτών τα [33] και [60].

6.5.2 Κλάσεις R6

Οι κλάσεις R6 παρέχονται από το ομώνυμο πακέτο 'R6' [70] το οποίο διατίθεται από το CRAN. Το πακέτο πρέπει να προστεθεί στο σύστημα της R[368] και να συνδεθεί με αυτό[369] πριν χρησιμοποιηθεί. Αν και δεν είναι ενσωματωμένες στην R, το σύστημα κλάσεων R6 αξιοποιείται από αρκετά άλλα πακέτα ως εναλλακτικό του συστήματος κλάσεων RS (το οποίο περιγράφεται στην προηγούμενη ενότητα). Πρόκειται για ένα σύστημα που βασίζεται σε μεγάλο βαθμό σε αντικείμενα τύπου environment και προσθέτει λειτουργίες οι οποίες λείπουν από το RS. Το R6 παρουσιάζει ακόμα περισσότερες ομοιότητες με συστήματα άλλων αντικειμενοστραφών γλωσσών από το RS και στοχεύει στη διευκόλυνση της χρήσης των κλάσεων αλλά και της συνεργασίας κλάσεων με άλλες που παρέχονται από άλλα πακέτα, δηλαδή την κληρονομικότητα ανάμεσα σε κλάσεις διαφορετικών πακέτων.

Το σύστημα R6 ορίζει δύο επίπεδα πρόσβασης στις μεθόδους (συναρτήσεις) και στα πεδία (άλλες μεταβλητές) των αντικειμένων. Έτσι αυτά μπορούν να είναι public (δημόσια) στα οποία υπάρχει πρόσβαση από όλους, ή private (ιδιωτικά) στα οποία έχουν πρόσβαση μόνο τα ίδια τα αντικείμενα[370]. Ακολουθεί παράδειγμα ορισμού μιας κλάσης R6 με όνομα «κλάση1», κάτι που γίνεται με τη συνάρτηση **R6Class**:

```
κλάση1 <- R6Class("κλάση1",
                   public = list(
                            d1a = 0,
                            d1b = 0,
```

[367] Οι τελεστές ανάθεσης <<- και ->> περιγράφονται στις §4.2.2.1 Περιβάλλοντα και συναρτήσεις και §5.1.5 Αλληλεπίδραση με το περιβάλλον.
[368] βλ.§1.5.3 Εγκατάσταση και διαχείριση πρόσθετων πακέτων.
[369] βλ.§1.5.1 Χρήση πακέτων.
[370] Σε άλλες αντικειμενοστραφείς γλώσσες υπάρχουν και άλλοι διαχωρισμοί πρόσβασης όπως το protected για μεταβλητές ή μεθόδους στις οποίες έχουν πρόσβαση τα μέλη της κλάσης αλλά και των κλάσεων που βασίζονται σε αυτή.

```
                                      initialize = function(a = 0, b = 0)
                                             {
                                             stopifnot(is.numeric(a),
                                                      is.numeric(b))
                                             self$d1a <- a
                                             self$d1b <- b
                                             }
                                           )
                     )
```

Στις παραμέτρους της **R6Class** ορίζεται ότι η νέα «κλάση1» έχει δύο δημόσια (public) πεδία d1a και d1b, ενώ ως δημόσια έχει οριστεί και μια μέθοδος με όνομα initialize. Η συγκεκριμένη μέθοδος έχει ειδικό σκοπό στις κλάσεις R6 καθώς καλείται αυτόματα κατά τη δημιουργία κάθε νέου αντικειμένου[371] (κάτι που θα δούμε παρακάτω). Έτσι εδώ μπορούν να οριστούν οι αρχικές τιμές των πεδίων του νέου αντικειμένου αλλά και να γίνουν έλεγχοι εγκυρότητας αυτών, όπως έγινε στο παραπάνω παράδειγμα (ελέγχεται ότι οι αρχικές τιμές είναι numeric). Για τα πεδία (εδώ d1a και d1b) δεν ορίζεται τύπος όπως χρειάζονταν στις κλάσεις RS, ενώ όλα τα παραπάνω δίνονται σε αντικείμενο list που ανατίθεται στην παράμετρο public (εφόσον είναι δημόσια).

Στη μέθοδο initialize φαίνεται επίσης ο τρόπος με τον οποίο γίνεται πρόσβαση στα δημόσια πεδία ενός αντικειμένου. Και εδώ (όπως στις κλάσεις RS) η πρόσβαση γίνεται με τον τελεστή $, ενώ όταν εκτελείται μια μέθοδος πάνω σε ένα αντικείμενο, το public μέρος του[372] ονομάζεται self[373]. Έτσι το self$d1a αναφέρεται στο δημόσιο πεδίο d1a του αντικειμένου για το οποίο καλείται η μέθοδος. Επίσης πρέπει να σημειωθεί ότι με τον τρόπο αυτό η ανάθεση σε πεδία γίνεται με τους συνήθεις τελεστές <- και ->.

Σε κάθε περίπτωση, η επιτυχής κλήση της συνάρτησης R6Class επιστρέφει ένα αντικείμενο-γεννήτορα (αντικείμενο τύπου R6ClassGenerator[374]) το οποίο περιέχει συναρτήσεις δημιουργίας και χειρισμού αντικειμένων της συγκεκριμένης κλάσης. Στο παραπάνω παράδειγμα το αντικείμενο-γεννήτορας καταχωρήθηκε σε μεταβλητή με όνομα κλάση1. Οι επόμενες εντολές είναι προσπάθειες δημιουργίας αντικειμένων της «κλάση1»:

```
> a <- κλάση1$new(1,2)
> u <- κλάση1$new(1,"r")
 Error in initialize(...) : is.numeric(b) is not TRUE
```

Η πρώτη εντολή ήταν επιτυχής και δημιούργησε αντικείμενο a, μέλος της «κλάση1»[375]. Η δημιουργία ενός νέου αντικειμένου γίνεται καλώντας τη συνάρτηση **new** του γεννήτορα. Κατά τη δημιουργία του a τα πεδία του d1a και d1b πήραν αρχικές τιμές 1 και 2, αντίστοιχα. Για τον σκοπό αυτό η συνάρτηση new του γεννήτορα καλεί τη μέθοδο initialize της νέας κλάσης. Αυτός είναι και ο λόγος που η δεύτερη εντολή απέτυχε και δεν δημιούργησε το αντικείμενο u: κατά την κλήση της initialize που έχει οριστεί για την «κλάση1» γίνεται έλεγχος αν οι τιμές είναι numeric και στη συγκεκριμένη περίπτωση ο έλεγχος απέτυχε. Ακολουθούν μερικοί χειρισμοί του αντικειμένου μέσω της a καθώς και μέσω μιας δεύτερης μεταβλητής u στο οποίο αντιγράφεται η a:

```
> a$d1a<-100
> u<-a
> u$d1b <- 200
> a
<κλάση1>
  Public:
    clone: function (deep = FALSE)
    d1a: 100
    d1b: 200
    initialize: function (a = 0, b = 0)
```

Από τα παραπάνω βλέπουμε ότι στις κλάσεις R6 (όπως και στις RS) πολλαπλές μεταβλητές μπορούν να προβαίνουν σε αλλαγές στο ίδιο αντικείμενο διατηρώντας την αναφορά σε αυτό. Ό λόγος είναι πως τα

[371] Μέθοδοι με αντίστοιχο ρόλο ονομάζονται constructor σε άλλες αντικειμενοστραφείς γλώσσες προγραμματισμού.
[372] Είναι ένα environment που περιέχει που περιέχει τις δημόσιες μεταβλητές (πεδία), μεταξύ αυτών και συναρτήσεις (μεθόδους).
[373] Παρεμφερές του this σε άλλες αντικειμενοστραφείς γλώσσες.
[374] Βασισμένο στον τύπο environment.
[375] Επίσης βασισμένο στον τύπο environment.

αντικείμενα κλάσεων R6 (όπως τα a και u) είναι υλοποιημένα ως environment άρα ισχύουν όσα αναφέρονται στην §4.2.2.2 Περιβάλλοντα και ο μηχανισμός αναφοράς.

Αν όντως πρέπει να δημιουργηθεί αντίγραφο του αντικειμένου, αυτό γίνεται με τη μέθοδο **clone** (αντίστοιχη της copy στο σύστημα RS). Κατά τη δημιουργία μιας κλάσης R6 με τη συνάρτηση R6Class μπορεί να απαγορευτεί η αντιγραφή για τα αντικείμενα της κλάσης (παράμετρος clonable=FALSE) ή εφόσον η αντιγραφή επιτρέπεται, με ποιο τρόπο θα γίνεται. Άρα για να ανατεθεί στη μεταβλητή u (αναφορά σε) ένα νέο αντικείμενο, αντίγραφο αυτού στο οποίο αναφέρεται η μεταβλητή a, η εντολή είναι:

```
u <- a$clone()
```

Ας δούμε τώρα ένα παράδειγμα όπου υπάρχει κάποια κληρονομικότητα καθώς και ιδιωτικό (private) μέρος[376] του αντικειμένου (με ιδιωτικά πεδία και μεθόδους).

```
κλάση2 <- R6Class( "κλάση2",
  inherit =   κλάση1,

  # Δημόσιο περιβάλλον:

  public = list(
    d2a = 0,

    άλλαξε_όλα_τα_πεδία_σε    = function(x)
    {
      stopifnot (is.numeric(x))
      self$d1a <- x
      self$d1b <- x
      self$d2a <- x
      private$d2b <- x
      s <- private$πρόσθεσε_όλα_τα_πεδία()
      cat("Άλλαξα όλα σε", x, "σύνολο", s, "\n")
      return(TRUE)
    }
  ),

  # Ιδιωτικό περιβάλλον:

  private = list(
    d2b = 0,

    πρόσθεσε_όλα_τα_πεδία    = function()
    {
      return(sum(self$d1a, self$d1b,
                 self$d2a, private$d2b))
    }
  )
)
```

Η παραπάνω νέα «κλάση2» έχει βάση την «κλάση1», όπως καταγράφεται στην παράμετρο inherits. Αρα κληρονομεί πεδία και μεθόδους από αυτή, συγκεκριμένα τα d1a, d1b και initialize. Σε αυτά προσθέτει ως δημόσια (public) το πεδίο d2a καθώς και τη μέθοδο άλλαξε_όλα_τα_πεδία_σε. Πρόσβαση από τις μεθόδους στα δημόσια αντικείμενα γίνεται με το πρόθεμα self. Επίσης η κλάση προσθέτει και κάποια ιδιωτικά (private) αντικείμενα στα οποία έχει πρόσβαση μόνο το ίδιο το αντικείμενο. Αυτά είναι η μεταβλητή d2b και η μέθοδος πρόσθεσε_όλα_τα_πεδία. Πρόσβαση στο ιδιωτικό μέρος του αντικειμένου γίνεται με το όνομα private και μόνο από μεθόδους στο ίδιο το αντικείμενο.

Έτσι, παρατηρώντας τη δημόσια μέθοδο άλλαξε_όλα_τα_πεδία_σε, αρχικά ελέγχει ότι η παράμετρος είναι numeric και μετά την αντιγράφει στα διάφορα πεδία. Για τα d1a, d1b και d2a που είναι δημόσια τα

[376] Είναι ένα environment που περιέχει τις ιδιωτικές μεταβλητές (πεδία), μεταξύ αυτών και συναρτήσεις (μεθόδους) και στο οποίο δεν υπάρχει εξωτερική πρόσβαση.

εντοπίζει ως self$d1a, self$d1b και self$d2a, ενώ για το d2b που είναι ιδιωτικό ως private$d2b. Μετά, χάριν παραδείγματος, καλεί την ιδιωτική μέθοδο πρόσθεσε_όλα_τα_πεδία που επιστρέφει το άθροισμα των παραπάνω. Το παρακάτω, δημιουργεί νέο αντικείμενο «κλαση2». Δεδομένου ότι δεν έχει οριστεί κάποιο initialize, καλείται το αντίστοιχο της κλάσης1 που αποτελεί βάση:

```
> b <- κλάση2$new(10,20)
```

Πρόσβαση στα πεδία του b:

```
> b$d1b<-30
> b$d2a<-50
> b$d2b<-70
Error in b$d2b <- 70 : cannot add bindings to a locked
environment
```

Η τελευταία εντολή απέτυχε καθώς το d2b είναι ιδιωτικό. Για τον ίδιο λόγο η ιδιωτική πρόσθεσε_όλα_τα_πεδία είναι αδύνατο να κληθεί εκτός από τον κώδικα σε άλλες μεθόδους της κλάσης:

```
> b$πρόσθεσε_όλα_τα_πεδία()
Error: attempt to apply non-function
```

Αν εμφανιστεί το b:

```
> b
<κλάση2>
  Inherits from: <κλάση1>
  Public:
    clone: function (deep = FALSE)
    d1a: 10
    d1b: 30
    d2a: 50
    initialize: function (a = 0, b = 0)
    άλλαξε_όλα_τα_πεδία_σε: function (x)
  Private:
    d2b: 0
    πρόσθεσε_όλα_τα_πεδία: function ()
```

Και τέλος, αν κληθεί η άλλαξε_όλα_τα_πεδία_σε, κάτι που προφανώς επιτρέπεται αφού είναι δημόσια:

```
> b$άλλαξε_όλα_τα_πεδία_σε(30)
Άλλαξα όλα σε 30 σύνολο 120
[1] TRUE
```

Το σύστημα R6 είναι ένα εναλλακτικό εργαλείο για δημιουργία κλάσεων αναφοράς. Στην ενότητα αυτή έγινε μια μικρή εισαγωγή. Το πακέτο διαθέτει εκτενή τεκμηρίωση (βλ. help("R6")) ενώ ενδιαφέροντα παραδείγματα χρήσης του καθώς και άλλες δυνατότητες που προσφέρει περιγράφονται στα [33] και [70].

Αναφορές Κεφαλαίου 6

[17] Chambers, J. M. (1998). *Programming with Data*. New York: Springer-Verlag,The Green Book. ISBN 978-0-387-98503-9.

[33] Wickham, H. (2019). *Advanced R (2nd Editton)*. Chapman and Hall/CRC. https://adv-r.hadley.nz/

[54] The R Core Team (2021). R Language Definition. https://cran.r-project.org/doc/manuals/R-lang.html

[60] Peng, R. D., Kross, S., & Anderson, B. (2020). *Mastering Software Development in R*. Ηλεκτρονικό. https://bookdown.org/rdpeng/RProgDA/

[62] Chambers, J. M. (2016). *Extending R, 1st Edition*. Chapman and Hall/CRC. ISBN 978-1-4987-7572-4.

[68] Burns, P. (2012). *The R Inferno, Second Edition*. lulu.com. ISBN 978-1471046520. https://www.burns-stat.com/documents/books/the-r-inferno/

[69] Chambers, J., & Hastie, T. (1992). *Statistical Models in S*. New York: Chapman & Hall/ CRC.The White Book. ISBN 9780203738535.

[70] Chang, W. (2021). R6: Encapsulated Classes with Reference Semantics. https://CRAN.R-project.org/package=R6 και https://r6.r-lib.org/index.html

Κεφάλαιο 7: Συνεργασία με άλλες γλώσσες προγραμματισμού

Σύνοψη
Η R μπορεί να συνεργαστεί με κώδικα γραμμένο σε άλλες γλώσσες προγραμματισμού. Το κεφάλαιο αυτό περιγράφει σύντομα τον τρόπο με τον οποίο γίνεται αυτό, επικεντρώνοντας στις γλώσσες TCL, C++ και Python.

Προαπαιτούμενη γνώση
Εξοικείωση με τα βασικά στοιχεία προγραμματισμού στην R, κεφάλαια 3 και 4. Βασική γνώση των γλωσσών προγραμματισμού που αναφέρονται.

7.1 Πολυγλωσσικές λύσεις

Η συνεργασία μεταξύ γλωσσών προγραμματισμού στοχεύει στην υλοποίηση λύσεων που θα συνδυάζουν και θα αξιοποιούν τα πλεονεκτήματα καθεμίας. Αυτό συχνά παρουσιάζει τεχνικές δυσκολίες. Σπάνια μια γλώσσα προγραμματισμού δημιουργείται με έμφαση στη συνεργασία με άλλες, ενώ συχνά οι σχεδιαστικές διαφορές ανάμεσα σε διαφορετικές γλώσσες είναι μεγάλες. Παρόλα αυτά, υπάρχει ποικιλία από εργαλεία που επιτρέπουν τον συνδυασμό κώδικα γραμμένου σε διαφορετικές γλώσσες και τη συνεργασία των διαφόρων αυτών τμημάτων στην υλοποίηση μιας λύσης λογισμικού. Στο κεφάλαιο αυτό θα ασχοληθούμε κυρίως με την ενσωμάτωση κώδικα άλλων γλωσσών σε μια λύση βασισμένη στην R. Όμως υπάρχουν και πολλά εργαλεία που ξεκινούν με βάση κάποια άλλη γλώσσα προγραμματισμού και επιτρέπουν την ενσωμάτωση της R στην υλοποιούμενη λύση. Για παράδειγμα, το JRI [71] που ενσωματώνει την R σε έργα βασισμένα σε Java. Αντίστοιχα το πακέτο ΄RInside' [72] για ενσωμάτωση R σε εφαρμογές C++. Δημοφιλής, είναι και η προσέγγιση της χρήσης R σε μορφή εξυπηρετητή, επιτρέποντας την πρόσβαση μέσω κλήσεων σε αυτόν από κώδικα γραμμένο σε πρακτικά οποιαδήποτε γλώσσα, καθώς οι περισσότερες γλώσσες προγραμματισμού επιτρέπουν τις σχετικές συνδέσεις (μεταξύ αυτών οι PHP, Java, C/C++, JavaScript κλπ).

Το κεφάλαιο αυτό επικεντρώνεται στην προσέγγιση όπου κύρια γλώσσα είναι η R. Η κλήση από τη γλώσσα R συναρτήσεων γραμμένων σε γλώσσες C και Fortran και μεταγλωττισμένων (compiled) από αυτές, είναι μια σχετικά απλή διαδικασία. Το μεταγλωττισμένο αρχείο (dll ή so) θα φορτωθεί στην R με τη συνάρτηση **dyn.load** και η συνάρτηση θα κληθεί με τις συναρτήσεις **.C** και **.Fortran** του πακέτου 'base'. Για τις συμβάσεις μετατροπής αντικειμένων από τη μια γλώσσα στην άλλη βλ. help(Foreign).

Στο κεφάλαιο αυτό παρουσιάζονται άλλες προσεγγίσεις που επιτρέπουν τη συνύπαρξη κώδικα γραμμένου σε διαφορετικές γλώσσες σε ένα έργο βασισμένο σε R. Συνοψίζονται τα πρώτα βήματα για την εκμετάλλευση (από κώδικα R) λειτουργιών υλοποιημένων στις γλώσσες Tcl, C++ και Python. Στο CRAN διατίθενται επίσης πακέτα που επιτρέπουν την ενσωμάτωση στην R και άλλων γλωσσών προγραμματισμού, όπως π.χ. το πακέτο 'rJava' (που περιλαμβάνει πλέον και το προαναφερθέν JRI) για ενσωμάτωση κώδικα Java [73], το πακέτο 'rGroovy' [74] που επιτρέπει την ενσωμάτωση σεναρίων γραμμένων στη γλώσσα Groovy, κ.α.

7.2 Tcl/Tk

Η Tcl (Tool Command Language) [75] είναι μια γλώσσα σεναρίων (scripting language) ανοικτού κώδικα που έχει σχεδιαστεί ώστε να είναι απλή, επεκτάσιμη αλλά και εφαρμόσιμη σε διαφορετικά λειτουργικά συστήματα (cross-platform). Σε συνδυασμό με την εργαλειοθήκη γραφικών στοιχείων Tk αλλά και διαφόρων επεκτάσεων που έχουν δημιουργηθεί για αυτή, η γλώσσα Tcl χρησιμοποιείται για ταχεία ανάπτυξη εφαρμογών (RAD - rapid application development), δημιουργία πρωτοτύπων (prototype development), αλλά και για υλοποίηση γραφικών διεπαφών χρήστη (GUI-graphical user interface) με κώδικα που είναι ανεξάρτητος του υπολογιστικού συστήματος στο οποίο θα εκτελείται. Έτσι ο συνδυασμός Tcl/Tk μπορεί να χρησιμοποιηθεί ως βάση για το παραθυρικό περιβάλλον μιας εφαρμογής και αυτή να εκτελεστεί χωρίς αλλαγές σε διαφορετικά λειτουργικά συστήματα. Στην R υποστήριξη των Tcl/Tk παρέχεται από το προ-εγκατεστημένο πακέτο 'tcltk' [1] [50]. Το πακέτο ενσωματώνει τα Tcl/Tk άρα δεν χρειάζεται κάποια περαιτέρω εγκατάσταση από τον χρήστη. Ο τρόπος συνεργασίας του πακέτου με τα Tcl/Tk δεν επιτρέπει την απευθείας χρήση σεναρίων Tcl/Tk. Αντί αυτού, το πακέτο παρέχει εντολές που έχουν το πρόθεμα tcl η tk οι οποίες επικοινωνούν με αντίστοιχες του ενσωματωμένου συστήματος Tcl/Tk. Έτσι απαιτείται κάποια προσαρμογή του κώδικα, αλλά αυτή είναι σχετικά εύκολο να γίνει. Ως το απλούστερο δυνατό παράδειγμα, οι εντολές της Tcl:

```
puts $tcl_version
puts "Hello"
```

που εμφανίζουν την έκδοση της tcl και το κείμενο "hello" έχουν ως αντίστοιχες εντολές τις συναρτήσεις **tclputs** και **tclVersion** του πακέτου 'tcltk'[377] στην R:

Δοκιμάστε:	Σχόλιο
`tclputs(tclVersion())`	Η Tcl εμφανίζει την έκδοσή της.
`tclputs("hello")`	Η Tcl εμφανίζει το κείμενο "Hello".

Τέτοιες συναρτήσεις ονομάζονται συναρτήσεις «κόλλας» (glue functions) καθώς δρουν ως διαμεσολαβητής ανάμεσα στην R και τον διερμηνευτή της γλώσσας Tcl που εκτελείται παράλληλα. Ο εντολές μετατρέπονται σε εντολές Tcl που εκτελούνται από τη γλώσσα αυτή στον δικό της χώρο. Για να δημιουργηθεί μια απλή μεταβλητή στην Tcl χρησιμοποιείται η συνάρτηση **tclVar** όπου δίνεται και η αρχική τιμή της μεταβλητής. Στην Tcl δεν ορίζεται τύπος μεταβλητής. Είναι dynamically typed γλώσσα όπως η R. Το παρακάτω δημιουργεί μια τέτοια μεταβλητή στην Tcl με τιμή "10". Παρατηρήστε την αυτόματη μετατροπή σε κείμενο, που είναι βασικός τρόπος αποθήκευσης δεδομένων στην Tcl[378]. Επίσης η εντολή ορίζει ότι από την πλευρά της R η πρόσβαση στη μεταβλητή αυτή θα γίνεται μέσω αντικειμένου (τύπου tclVar) της R με όνομα tv:

```
tv <- tclVar(10)
```

Η πρόσβαση στην ίδια τη μεταβλητή (που θυμίζουμε ότι βρίσκεται στον χώρο της Tcl, όχι της R, γίνεται μέσω της συνάρτησης **tclvalue**: και του σχετικού αντικειμένου της R (εδώ του tv):

Δοκιμάστε:	Σχόλιο
`tclvalue(tv)`	Επιστρέφει την τιμή της Tcl μεταβλητής στην οποία αναφέρεται το tv, δηλαδή "10".
`tclvalue(tv)<-20`	Αναθέτει στην παραπάνω μεταβλητή την τιμή "20".
`rv <- tclvalue(tv)`	Επιστρέφει την τιμή της παραπάνω μεταβλητής και την αναθέτει στην rv (της R).
`rv`	Ανακαλεί την rv, εμφανίζοντας "20".

Όλα αυτά ίσως φαίνονται υπερβολικά πολύπλοκα για μια απλή ανάθεση και ανάκληση τιμών μεταβλητής. Είναι λογικό να αναρωτηθεί κανείς ποιος είναι ο λόγος να μπει σε τόσο κόπο. Βασικό κίνητρο για χρήση της Tcl είναι η εργαλειοθήκη Tk που τη συνοδεύει. Η Tk είναι μια εργαλειοθήκη για δημιουργία και χειρισμό στοιχείων γραφικού περιβάλλοντος χρήστη, παράθυρα και «διαλόγους» (dialogs), κουμπιά, μενού και άλλα εργαλεία (αποκαλούμενα widgets[379]), μέσω των οποίων οι χρήστες αλληλεπιδρούν με μια εφαρμογή. Ο συνδυασμός χρήσης Tcl και Tk για τον σκοπό αυτό είναι αρκετά διαδεδομένος[380], η εργαλειοθήκη είναι ιδιαίτερα πλήρης, ενώ υπάρχουν και πολλές επεκτάσεις της. Επιπροσθέτως, (και αυτό ίσως είναι το σημαντικότερο) η χρήση της Tk οδηγεί σε κώδικα ο οποίος μπορεί να εκτελεστεί χωρίς αλλαγές σε διαφορετικά λειτουργικά συστήματα. Έτσι είναι ένας cross-platform τρόπος να δημιουργηθεί το γραφικό περιβάλλον χρήσης μιας εφαρμογής. Για το σύστημα Tcl/Tk υπάρχουν άφθονες πηγές και βιβλιογραφία (ενδεικτικά [76] [77]) και η παρουσίαση των δυνατοτήτων του συστήματος ξεπερνά τον σκοπό αυτού του βιβλίου.

Παρακάτω παρουσιάζονται μερικά εισαγωγικά παραδείγματα χρήσης Tcl/Tk για τη δημιουργία στοιχείων GUI που θα αλληλεπιδρούν με κώδικα R. Τα παραδείγματα προσπαθούν να παραμείνουν σχετικά απλά και σύντομα, ενώ ταυτόχρονα να αναδείξουν κάποια βασικά στοιχεία της δημιουργίας GUI με Tcl/Tk. Όμως είναι σύνηθες στην ανάπτυξη εφαρμογών που διαθέτουν GUI ένα σημαντικό μέρος του τελικού κώδικα να αφορά αποκλειστικά τη διαχείριση των στοιχείων αυτών, δηλαδή της γραφικής διεπαφής του λογισμικού με τον χρήστη.

Πριν το πρώτο παράδειγμα, καλό είναι να αναφερθούν μερικά σημεία: (α) Τα παράθυρα και άλλα widgets που θα δημιουργηθούν υπάρχουν και τρέχουν στον χώρο της Tcl. Η R απλώς αλληλεπιδρά με αυτά. Άρα ο κώδικας R μπορεί να συνεχίσει να εκτελείται ενώ παράλληλα η Tcl διαχειρίζεται τα στοιχεία του GUI. (β) Τδιάφορα στοιχεία του GUI τοποθετούνται ιεραρχικά, π.χ. τα κουμπιά έχουν «γονέα» το παράθυρο μέσα στο οποίο βρίσκονται ενώ και αυτό μπορεί να έχει «γονέα» κάποιο άλλο, κλπ. (γ) Την τακτοποίηση, τοποθέτηση

[377] Προφανώς όλα τα παραδείγματα απαιτούν να έχει πρώτα συνδεθεί το πακέτο 'tcltk' στην τρέχουσα συνεδρία, με εντολές όπως library(tcltk) ή require(tcltk).

[378] Η Tcl μετατρέπει αυτόματα τα δεδομένα σε άλλους τύπους, π.χ. σε αριθμητικούς αν χρειάζεται να γίνουν αριθμητικές πράξεις

[379] Για μια λίστα των widgets που παρέχει το Tk, βλ. help(TkWidgets).

[380] Ειδικά στο Unix και συγγενικά ΛΣ.

και τελική εμφάνιση των widgets αναλαμβάνουν ειδικά στοιχεία που αποκαλούνται «διαχειριστές διάταξης» (layout manager), όπως τα pack και grid. (δ) Τα διάφορα στοιχεία αντιδρούν σε «συμβάντα» (events). Το GUI είναι ένα event-driven (ή message-driven) σύστημα που διαχειρίζεται συμβάντα. Ένα τέτοιο συμβάν μπορεί να είναι το πάτημα ενός κουμπιού, η αλλαγή τιμής σε ένα slider, η εντολή κλεισίματος ενός παράθυρου κλπ. Μέρος του προγραμματισμού του GUI περιλαμβάνει τον ορισμό κώδικα που θα χειριστεί τα συμβάντα που μας ενδιαφέρουν. Ακολουθεί το πρώτο παράδειγμα:

```
επιλογέας_με_ui <- function( t1, t2, t3 )
{
  v<-NULL
  κύριο  <- tktoplevel()
  tkwm.title(κύριο, "Επίλεξε:")

  # μερικά κουμπιά που θα τοποθετηθούν στο κύριο:

  bt1  <- tkbutton(κύριο, text = t1,
                  command = function(){
                            tkdestroy(κύριο)
                            v<<-t1})

  bt2  <- tkbutton(κύριο, text = t2,
                   command = function(){
                            tkdestroy(κύριο)
                            v<<-t2})

  bt3  <- tkbutton(κύριο, text = t3,
                  command = function(){
                           tkdestroy(κύριο)
                           v<<-t3})

  tkpack (bt1,bt2,bt3,pady = 10)
  tkwait.window(κύριο)
  return(v)
}
```

Εδώ ορίζεται μια συνάρτηση (επιλογέας_με_ui) που θα δεχτεί τρεις παραμέτρους οι οποίες αντιστοιχούν σε τρείς πιθανές επιλογές του χρήστη. Η συνάρτηση ορίζει μια τοπική μεταβλητή v στην οποία θα καταγραφεί η τελική επιλογή του χρήστη, μια από τις τρείς προαναφερθείσες. Ακολούθως, δημιουργεί τα στοιχεία Tk που θα εμφανιστούν: (α) Δημιουργεί με τη συνάρτηση **tktoplevel** ένα top level παράθυρο, που δεν έχει άλλο παράθυρο για γονέα. Από την πλευρά της R αυτό θα αναφέρεται μέσω της μεταβλητής 'κύριο' στην οποία αναθέτει το σχετικό αντικείμενο (τύπου tkwin). (β) Αλλάζει τον τίτλο του παραθύρου αυτού, με τη συνάρτηση **tkwm.title**. (γ) Δημιουργεί τρία κουμπιά (τα οποία θα σχετιστούν με τις μεταβλητές bt1, bt2 και bt3). Καθένα από αυτά έχει γονέα το παράθυρο ΄κύριο΄ και θα τοποθετηθεί μέσα σε αυτό. Κάθε κουμπί εμφανίζει ως κείμενο (παράμετρος text) την αντίστοιχη πιθανή επιλογή για τον χρήστη. Επιπροσθέτως, στην παράμετρο command τοποθετείται η συνάρτηση που θα κληθεί να χειριστεί το συμβάν του πατήματος του κουμπιού, ο κώδικας δηλαδή που θα τρέξει αν ο χρήστης πατήσει το κουμπί. Στο παράδειγμα αυτό οι συναρτήσεις είναι ανώνυμες και ορίζονται απευθείας στον κώδικα ορισμού του κουμπιού, αλλά αντί αυτού θα μπορούσε να έχει δοθεί οποιαδήποτε συνάρτηση R[381] ως χειριστής του συμβάντος. Στις συγκεκριμένες συναρτήσεις (και χάριν απλούστευσης) επιλέχθηκε να κλείνει το κύριο παράθυρο (εντολή **tkdestroy**(κύριο)) και να ανατίθεται (με υπέρβαση του τοπικού περιβάλλοντος της συνάρτησης χειρισμού) η σχετική επιλογή στη μεταβλητή v (η οποία θα επιστραφεί στο τέλος). Την τελική εμφάνιση των κουμπιών αναλαμβάνει ένα layout manager τύπου pack (συνάρτηση **tkpack**). Το pack απλώς τοποθετεί τα widgets που του δίνονται σε οριζόντια (το ένα δίπλα στο άλλο) ή, όπως εδώ, σε κάθετη διάταξη (το ένα κάτω από το άλλο). Η τελευταία εντολή της συνάρτησης πριν επιστρέψει το v είναι η **tkwait.window**(κύριο). Με την εντολή αυτή η R περιμένει να κλείσει το κύριο

[381] Όταν χρησιμοποιείται αυτή η τεχνική και επιτρέπεται μια κλήση από το ξένο σύστημα (εδώ την Tcl) πίσω σε συνάρτηση της R, αναφερόμαστε στη συνάρτηση ως callback function.

παράθυρο πριν συνεχίσει την εκτέλεση κώδικα. Υπενθυμίζουμε ότι τα δύο συστήματα R και Tcl/Tκ λειτουργούν παράλληλα. Ανάλογα με τη λογική του προγράμματος μπορεί να επιλεγεί να συνεχίζεται η εκτέλεση κώδικα R (και όταν υπάρχει κάποιο συμβάν, το γραφικό περιβάλλον να κάνει κλήση των συναρτήσεων που το χειρίζονται) ή όπως εδώ, είναι προτιμότερο να περιμένει η R την ολοκλήρωση της εργασίας του χρήστη με το παράθυρο πριν συνεχίσει την εκτέλεση. Αν η παραπάνω συνάρτηση κληθεί θα εμφανιστεί το παράθυρο με τίτλο "Επίλεξε" της Εικόνας 7.1 και εφόσον επιλεγεί το 2º κουμπί θα έχουμε:

```
> επιλογέας_με_ui("Βουνό","Θάλασσα","Πεδιάδα")
[1] "Θάλασσα"
```

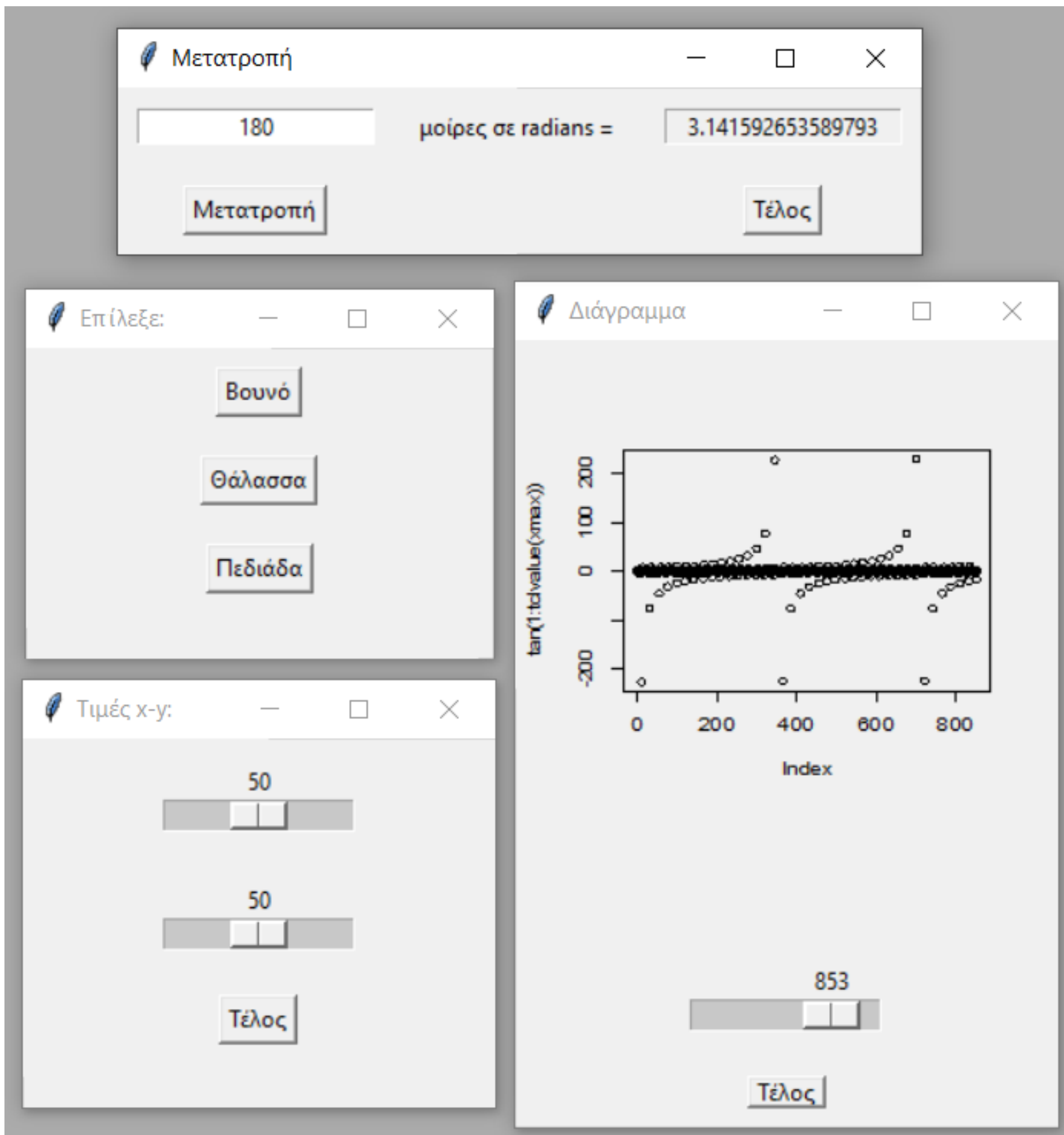
Η συνάρτηση επιστρέφει την επιλογή που αντιστοιχεί στο κουμπί που πατήθηκε. Αν το παράθυρο κλείσει χωρίς να επιλεγεί κάποιο κουμπί θα επιστρέψει την αρχική τιμή της μεταβλητής v, δηλαδή NULL. Το επόμενο παράδειγμα χρησιμοποιεί slide (άλλος τύπος widget) και ακολουθεί λίγο διαφορετική προσέγγιση στην ανταλλαγή δεδομένων μεταξύ R kai Tcl:

```
vx <- tclVar(50)
vy <- tclVar(50)

άθροισε<-function(...)
{
  x <- as.numeric(tclvalue(vx))
  y <- as.numeric(tclvalue(vy))
  cat("Άθροισμα",x,"και",y,"ίσον",x+y,"\n")
}

επιλογέας_με_slide <- function(data)
{
  κύριο  <- tktoplevel()
  tkwm.title(κύριο, "Τιμές x-y:")

  skx <- tkscale(κύριο, orient="horizontal", variable=vx,
                 command = άθροισε)
  sky <- tkscale(κύριο, orient="horizontal", variable=vy,
                 command = άθροισε)
  bto <- tkbutton(κύριο, text = "Τέλος",
                 command = function() {tkdestroy(κύριο)})

  tkpack (skx,sky,bto,pady = 10)
}
```

Εδώ αρχικά ορίζονται δύο μεταβλητές Tcl και σχετίζονται με τα vx και vy. Η συνάρτηση που ακολουθεί (άθροισε) ανακαλεί τις μεταβλητές αυτές ως αριθμούς, τις προσθέτει και εμφανίζει το άθροισμα με σχετικό μήνυμα. Η συνάρτηση επιλογέας_με_slide είναι αυτή που δημιουργεί το παράθυρο και τα widgets. Πολλά από τα βήματα ταυτίζονται με όσα έγιναν στο προηγούμενο παράδειγμα. Αρχικά δημιουργεί top level παράθυρο ('κύριο') και ορίζει τον τίτλο του, μετά δημιουργεί δύο slide widget (skx και sky) και ένα κουμπί (bto). Τα skx και sky συνδέονται με τις μεταβλητές των vx και vy αντίστοιχα. Έτσι, αν αλλάξει η θέση του slide, θα αλλάξει και η τιμή της σχετικής μεταβλητής, ενώ ως χειριστής του συμβάντος ορίζεται η συνάρτηση άθροισε που αναφέρθηκε παραπάνω. Μία κλήση της συνάρτησης μπορεί να γίνει ως:

```
επιλογέας_με_slide()
```

και θα έχει ως αποτέλεσμα το παράθυρο με τίτλο "Τιμές x-y" της Εικόνας 7.1. Στο συγκεκριμένο παράδειγμα η R θα προχωρήσει στην εκτέλεση των εντολών που ακολουθούν (δεν έγινε κλήση της tkwait.window) αλλά όταν (και αν) συμβεί συμβάν αλλαγής της θέσης των slide θα καλείται η συνάρτηση άθροισε.



Εικόνα 7.1 *Παραδείγματα συνεργασίας R με Tcl/Tk.*

Το τελευταίο παράδειγμα χρησιμοποιεί widgets κειμένου, label και entry (textbox). Κάνει μετατροπή κάποιας γωνίας από μοίρες σε ακτίνια.

```
μετατροπέας_με_ui <- function(x=0)
{

  v1 <- tclVar(x)
  v2 <- tclVar(radians(x))

  κύριο  <- tktoplevel()
  tkwm.title(κύριο, "Μετατροπή")

  en_είσοδος    <- tkentry(κύριο, textvariable = v1,
                          justify="center")

  en_αποτέλεσμα <- tkentry(κύριο, textvariable = v2,
```

```
                                  justify="center", state="readonly")

    lb_λεζαντα   <- tklabel(κύριο, text ="μοίρες σε radians = ")

    lb_κενό <- tklabel(κύριο, text = "")

    bt_μετατροπή  <- tkbutton(κύριο, text = "Μετατροπή",
                               command = function()
                               {
                                    d <- as.numeric(tclvalue(v1))
                                 r <- radians(d)
                                 tclvalue(v2) <- r
                                 })

    bt_τέλος  <- tkbutton(κύριο, text = "Τέλος",
                            command=function() {tkdestroy(κύριο)})


    tkgrid( en_είσοδος, lb_λεζαντα, en_αποτέλεσμα,
            padx = 10, pady = 10 )

    tkgrid(bt_μετατροπή, lb_κενό, bt_τέλος, pady = 10 )
  }
```

Επειδή οι ομοιότητες είναι πολλές με τα προηγούμενα παραδείγματα, θα επικεντρωθούμε σε κάποιες διαφορές. Εδώ τα πρώτα widgets που ορίζονται είναι δύο entry (textbox). Στο ένα (en_είσοδος) θα εισάγεται η γωνία σε μοίρες, ενώ το δεύτερο (en_αποτέλεσμα) είναι μόνο για ανάγνωση (state="readonly") και θα εμφανίζει το αποτέλεσμα της μετατροπής σε ακτίνια. Τα widgets συνδέθηκαν με τις Tcl μεταβλητές v1 και v2 που ορίστηκαν στην αρχή της συνάρτησης και θα συγχρονίζονται με αυτές. Ακολούθως ορίζονται δύο label (lb_λεζαντα και lb_κενό). Τα label είναι widgets για στατικό, σταθερό κείμενο. Ο σκοπός για τα δύο κουμπιά (bt_μετατροπή και bt_τέλος) που ορίζονται αμέσως μετά είναι να ενεργοποιούν τη μετατροπή (bt_μετατροπή) ή να ολοκληρώνουν τη χρήση κλείνοντας το παράθυρο (bt_τέλος). Όταν πατηθεί το bt_μετατροπή, η συνάρτηση χειρισμού (command) διαβάζει την τιμή της v1 (συνδεδεμένης με το en_είσοδος), κάνει τη μετατροπή μέσω της συνάρτησης radians (που δημιουργήθηκε στην §5.1.8 Παραδείγματα και δέχεται ως μοναδικό όρισμα τη γωνία σε μοίρες). Το αποτέλεσμα της radians ανατίθεται στη v2 που ως συνδεδεμένη με το en_αποτέλεσμα εμφανίζει εκεί το αποτέλεσμα. Για τη διάταξη των widgets επιλέχτηκαν δυο grid (πάλι χάριν απλούστευσης του παραδείγματος), με το πρώτο να περιέχει τα στοιχεία entry ενώ το δεύτερο τα button. Μία κλήση της συνάρτησης μπορεί να γίνει ως

```
μετατροπέας_με_ui(180)
```

με αποτέλεσμα ένα παράθυρο με τίτλο "Μετατροπή" όπως αυτό της Εικόνας 7.1. Προφανώς αν επιλεγεί στο παράθυρο άλλη τιμή μοιρών και το σχετικό κουμπί "Μετατροπή", εμφανίζεται η αντίστοιχη γωνία σε radians.

Την απεικόνιση γραφικών παραστάσεων σε παράθυρα Tcl/Tk βοηθούν τα πακέτα 'tkrplot' [78] και 'tkRplotR'. Οι γραφικές παραστάσεις μπορούν να είναι δυναμικές και να επηρεάζονται από επιλογές που γίνονται σε άλλα στοιχεία του παραθύρου. Το παράδειγμα που ακολουθεί χρησιμοποιεί το πακέτο 'tkrplot' και το ομώνυμο widget για την απεικόνιση μιας γραφικής παράστασης με τη συνάρτηση **tkrreplot** [79] να αναλαμβάνει την ενημέρωση των περιεχομένων του γραφήματος αν αλλάξει η επιλεγμένη τιμή στο slide (widget με όνομα skx). Το αποτέλεσμα κλήσης της συνάρτησης διάγραμμα() είναι το παράθυρο με τίτλο "Διάγραμμα" της Εικόνας 7.1, ενώ ο κώδικας της συνάρτησης είναι:

```
διάγραμμα<-function()
  {
  xmax <- tclVar(100)

  κάνε_το_plot<-function(...){plot(tan(1:tclvalue(xmax)))}
  ξανακάνε_το_plot = function(...)tkrreplot(wim)
```

```
  κύριο <- tktoplevel()
  tkwm.title(κύριο, "Διάγραμμα")

  wim  <- tkrplot(κύριο,κάνε_το_plot)
  skx  <- tkscale(κύριο, orient="horizontal", variable = xmax,
                    from = 1, to = 1000,
                    command = ξανακάνε_το_plot)
  bto  <- tkbutton(κύριο, text = "Τέλος",
                command = function() {tkdestroy(κύριο)})

  tkpack(wim,skx,bto,side='top',pady=10)
}
```

Η συνεργασία Tcl/Tk επιτρέπει τη δημιουργία πολυσύνθετων GUI για τον κώδικα R, με πλήρη λειτουργικότητα και πολλές δυνατότητες, που τα παραδείγματα σαφώς και δεν αναδεικνύουν. Αυτό όμως απαιτεί εξοικείωση με τα ίδια τα Tcl/Tk. Πάντως, ακόμα και όταν δεν είναι απαραίτητος ο σχεδιασμός κάποιου ειδικά προσαρμοσμένου για την εφαρμογή παραθυρικού περιβάλλοντος, το πακέτο ΄tcltk' είναι χρήσιμο. Το πακέτο παρέχει μερικά προκατασκευασμένα βασικά στοιχεία αλληλεπίδρασης με τον χρήστη, τους τυποποιημένους διαλόγους (standard dialogs) που εμφανίζουν παράθυρα μηνυμάτων (message box) ή επιλογής αρχείων και φακέλων. Οι σχετικές συναρτήσεις δεν απαιτούν ιδιαίτερη παραμετροποίηση για να χρησιμοποιηθούν και περιλαμβάνουν συναρτήσεις για επιλογή αρχείων (**tkgetOpenFile** και **tkgetSaveFile**) ή φακέλων (**tkchooseDirectory**) και για δημιουργία παράθυρων μηνυμάτων (messagebox), π.χ. η εντολή:

```
fn <- tclvalue(tkgetOpenFile())
```

δημιουργεί ένα παράθυρο επιλογής αρχείου και όταν αυτή γίνει, επιστρέφει ως κείμενο τη διαδρομή προς αυτό (περνώντας το κείμενο σε μεταβλητή fn της R μεσω της tclvalue), ενώ η εντολή:

```
tk_messageBox("Να γίνει κάτι;", type = "okcancel")
```

εμφανίζει παράθυρο τύπου messagebox με το κείμενο "Να γίνει κάτι;" και δύο κουμπιά ("Ok" και "Ακύρωση") επιστρέφοντας αντίστοιχα την επιλογή του χρήστη.

Σε προηγούμενη ενότητα (§3.3 Στοιχεία διεπαφής χρήστη (user interface)) έγινε μια σύντομη παρουσίαση συναρτήσεων που μπορούν να χρησιμοποιηθούν για δημιουργία διαλόγων ή και ενός πλήρους GUI ως διεπαφή χρήστη με κώδικα R. Εκεί αναφέρθηκε και το πακέτο 'gWidgets2' που ορίζει διάφορες ισχυρές συναρτήσεις χειρισμού GUI και απλοποιεί τον χειρισμό των στοιχείων με τυποποιημένο κώδικα ο οποίος μπορεί να εκτελεστεί σε διαφορετικές εργαλειοθήκες. Το πακέτο 'gWidgets2tcltk' [52] υλοποιεί τις συναρτήσεις του ΄gWidgets2' σε Tcl/Tk. Παρακάτω δημιουργείται ένα παράθυρο παρεμφερές σε εικόνα και λειτουργία με αυτό του παραδείγματος μετατροπέας_με_ui που προηγήθηκε παραπάνω, αλλά εδώ ο κώδικας χρησιμοποιεί συναρτήσεις του ΄gWidgets2':

```
μετατροπέας_με_gwidgets_ui <- function(x=0)
{
  y <- radians(x)
  w <- gwindow("Μετατροπή",visible=FALSE,
             width = 300, height = 100 )
  g1  <- gvbox(container=w)
  lyt <- glayout(container=g1)
  lyt[1,1] <- gedit(x,  container=lyt)
  lyt[1,2] <- "μοίρες σε radians="
  lyt[1,3] <- gedit(y, container=lyt)
  enabled(lyt[1,3]) <- FALSE

  g2 <- ggroup(container=g1)
  addSpring(g2)
  gbutton("Μετατροπή", container=g2,
          handler=function(h, ...)
          {
            a <- svalue(lyt[1,1])
            b <- radians(as.numeric(a))
            svalue(lyt[1,3])<- b })
```

```
    gbutton("Τέλος", container=g2,
          handler=function(h, ...)
           { dispose(w) })
  visible(w) <- TRUE
  }
```

Ο κώδικας, δημιουργεί ένα παράθυρο w στο οποίο προστίθεται ένα container g1 για στοιχεία UI (widgets). Το layout lyt αναλαμβάνει την τοποθέτηση κάποιων από τα widgets στο g1 (ενός edit-box εισόδου, μιας λεζάντας και ενός edit-box εξόδου). Πρόσβαση στα περιεχόμενα των widgets αυτών γίνεται με τη συνάρτηση svalue και τη θέση του widget στο layout. Το αποτέλεσμα κλήσης της παραπάνω συνάρτησης είναι ένα παράθυρο με τίτλο "Μετατροπή", παρεμφερές με αυτό της Εικόνας 7.1. Υπάρχουν πολλές ομοιότητες με τον αντίστοιχο κώδικα Tcl/Tk όμως εδώ ο κώδικας μπορεί να εκτελεστεί και από άλλες εργαλειοθήκες στοιχείων UI.

7.3 Python

Η Python [80] [81] είναι διαδεδομένη γλώσσα γενικής χρήσης, που χρησιμοποιείται όμως και σε εφαρμογές που σχετίζονται με δεδομένα και μηχανική μάθηση. Ο κώδικας Python εκτελείται μέσω διερμηνευτή, όπως εκτελείται και ο κώδικας των R και Tcl. Υπάρχουν διάφοροι τρόποι συνεργασίας R με Python. Από την πλευρά της Python, η βιβλιοθήκη rpy2 [82] συνδέει τη γλώσσα αυτή με την R επιτρέποντας την ανταλλαγή δεδομένων ανάμεσα σε κώδικες των δύο γλωσσών. Την αντίστροφη κατεύθυνση εξυπηρετεί το πακέτο 'reticulate'[382] [83] της R, που ξεκινά από την R και συνδέει με αυτή την Python. Με το 'reticulate' η Python φορτώνεται ως βιβλιοθήκη στη διεργασία της R. Αυτό απλοποιεί τη συνεργασία ανάμεσα στα δύο περιβάλλοντα (R και Python). Το 'reticulate' επιτρέπει την πρόσβαση και αντιγραφή (με αυτόματη μετατροπή) αντικειμένων της μιας γλώσσας στην άλλη. Επιπρόσθετα έχει αυτοματοποιηθεί η εγκατάσταση της Python, υποστηρίζονται διαχειριστές περιβαλλόντων (μέσω anaconda, miniconda κλπ.), διαχείριση πακέτων Python[383] κ.α. Το RStudio υιοθετεί το 'reticulate' και υποστηρίζει ταυτόχρονα τις δύο γλώσσες ενώ παρέχει και απαραίτητα βοηθήματα όπως επεξεργαστή πηγαίου κώδικα[384] Python με επισήμανση της σύνταξης (syntax highlighting) , υποστήριξη εκτέλεσης του κώδικα, εμφάνιση των μεταβλητών Python στην καρτέλα Environment[385] κλπ. Όλα τα παραπάνω επιτρέπουν την ανεμπόδιστη συνεργασία κώδικα γραμμένου σε R, με κώδικα στη γλώσσα Python (και αντίστροφα) σε ένα έργο[386], πακέτο της R[387], έγγραφο RMarkdown[388], κλπ.

Αν δεν υπάρχουν ήδη μία ή περισσότερες εκδόσεις της Python στο σύστημα, το 'reticulate' θα προτείνει αυτόματη εγκατάσταση της γλώσσας μέσα από το περιβάλλον miniconda. Αυτό γίνεται με τη βοηθητική συνάρτηση **install_miniconda**. Άλλες βοηθητικές συναρτήσεις που παρέχει το πακέτο 'reticulate' περιλαμβάνουν τη **use_python** για επιλογή της εγκατάστασης Python που θα χρησιμοποιηθεί κατά την εκτέλεση κώδικα, την **py_install** για εγκατάσταση βιβλιοθηκών Python, τη **use_condaenv** και τη **use_miniconda** για το σχετικό περιβάλλον anaconda ή miniconda, τη **use_virtualenv** και την οικογένεια συναρτήσεων **virtualenv_*** για επιλογή, δημιουργία ή διαχείριση του εικονικού περιβάλλοντος conda που θα χρησιμοποιηθεί, την **conda_list** που εμφανίζει τα εγκατεστημένα περιβάλλοντα και τη συνάρτηση **repl_python** που εμφανίζει τη γραμμή εντολών της Python επιτρέποντας απευθείας καταχώρηση εντολών[389]. Τα παραδείγματα που ακολουθούν βασίζονται σε χρήση Python μέσω miniconda, έτσι έχει προηγηθεί εγκατάστασή του με την εντολή reticulate::install_miniconda()[390]. Επίσης εκτελέστηκε η εντολή reticulate::py_install("pandas") για να εγκατασταθεί η βιβλιοθήκη 'pandas' της Python.

---

382 Διαθέσιμο από το CRAN.

383 Η εγκατάσταση Python μέσα από το περιβάλλον miniconda γίνεται με τη συνάρτηση install_miniconda. Άλλες βοηθητικές συναρτήσεις που παρέχει το 'reticulate' περιλαμβάνουν: τη συνάρτηση use_python για επιλογή της εγκατάστασης Python που θα χρησιμοποιηθεί, τη virtualenv_root για το εικονικό περιβάλλον, τη use_condaenv και use_miniconda για το σχετικό περιβάλλον anaconda ή miniconda, την conda_list που εμφανίζει τα εγκατεστημένα περιβάλλοντα κ.α.

384 Αντίστοιχο με αυτόν που παρέχεται για την R, βλ.§3.1.1 Δημιουργία και εκτέλεση R script.

385 βλ.§1.4.2.7 Η καρτέλα Environment.

386 βλ.§3.1.7 Έργο (project).

387 βλ.Κεφ. 8.

388 βλ.§9.4 Δυναμικά έγγραφα.

389 Η γραμμή εντολών (ή κέλυφος) της Python ονομάζεται REPL (Read-Evaluate-Print Loop).

390 Προφανώς, για να εκτελεστούν τα παραδείγματα, απαιτείται να εγκατασταθεί το πακέτο 'reticulate'και να συνδεθεί

Ο βασικός ρόλος του 'reticulate' είναι να συνδέει τον διερμηνευτή της R με τον διερμηνευτή της Python καθώς τα δύο αυτά περιβάλλοντα συνυπάρχουν και εκτελούνται παράλληλα (το δεύτερο ως διεργασία του πρώτου). Έτσι το πακέτο παρέχει:

- Αυτόματη μετατροπή και εξαγωγή αντικειμένων του ενός περιβάλλοντος σε αντίστοιχα αντικείμενα του άλλου. Οι μετατροπές γίνονται ανάλογα με τον τύπο αντικειμένου και περιγράφονται στην τεκμηρίωση του πακέτου, βλ. vignette("calling_python"). Περιλαμβάνουν wrapper functions για τις συναρτήσεις[391] καθώς και μετατροπές πέραν των βασικών τύπων δεδομένων, π.χ. ένα Python αντικείμενο τύπου dictionary αντιγράφεται σε R list, ένα pandas DataFrame σε R data.frame, ένα NumPy ndarray σε matrix ή array και αντίστροφα.
- Μεγάλο αριθμό από συναρτήσεις μετατροπής (όπως η **py_to_r**) και χειρισμού από την R των αντικειμένων στο περιβάλλον της Python για την κάλυψη απαιτήσεων που δεν ικανοποιούνται από την αυτόματη μετατροπή.
- Συναρτήσεις χειρισμού της Python από την R (διαγραφή αντικειμένων, εμφάνιση βοήθειας, καταγραφή του κειμένου εξόδου που παράγεται και πολλά άλλα).
- Συναρτήσεις εκτέλεσης κώδικα Python από την R, όπως οι **py_run_string**, **py_run_file** και **source_python**. Οι δύο πρώτες εκτελούν κώδικα στο περιβάλλον της Python, επιστρέφοντας dictionary με τα αντικείμενα που υπάρχουν εκεί μετά την εκτέλεση. Η source_python εκτελεί μεν ένα σενάριο Python, αλλά επιτρέπει και την αυτόματη εξαγωγή των αντικειμένων Python[392] σε αντίστοιχα αντικείμενα R (μέσω των προαναφερθέντων μετατροπών) καθώς και τον ορισμό του environment στο οποίο αυτά θα δημιουργηθούν.
- Αντικείμενα ("**py**" και "**r**") διεπαφής ανάμεσα στις δύο γλώσσες. Με τα αντικείμενα αυτά μπορεί να γίνει πρόσβαση αλλά και δημιουργία αντικειμένων από το ένα περιβάλλον στο άλλο.

Ας δούμε ένα παράδειγμα. Οι εντολές (της R) που ακολουθούν, στέλνουν προς εκτέλεση τις δοθείσες (ως κείμενο) εντολές στην Python:

Δοκιμάστε:	**Σχόλιο**
`py_run_string("t1=10")`	Εκτελεί εντολή Python που δημιουργεί μεταβλητή με τιμή 10.
`py_run_string("print('t1=',t1)")`	Εκτελεί την εντολή print της Python εμφανίζοντας "t1= 10".

Μετά την πρώτη παραπάνω εντολή δημιουργείται στο περιβάλλον της Python μια μεταβλητή t1 με τιμή 10. Το ειδικό αντικείμενο py επιτρέπει την πρόσβαση από την R σε αντικείμενα που υπάρχουν στην πλευρά της Python. Το py επιτρέπει επίσης τη δημιουργία νέων αντικειμένων στο περιβάλλον της Python με αυτόματη μετατροπή αντικειμένων R. Το όνομα του αντικειμένου προσδιορίζεται με το $. Για παράδειγμα:

Δοκιμάστε:	**Σχόλιο**
`2 * py$t1`	Πρόσβαση στο t1 της Python, με τον πολλαπλασιασμό επιστρέφει 20.
`py$t2 <- c(10,20,30)`	Δημιουργία στην Python μεταβλητής t2 τύπου list (μετατροπή του R vector).
`py_run_string("print(t2)")`	Εκτελεί την εντολή print της Python εμφανίζοντας [10.0, 20.0, 30.0].
`x <- py$t2`	Ανάθεση του αντικείμενου t2 της Python σε μεταβλητή x της R.

Με την τελευταία εντολή του παραδείγματος, το x που δημιουργείται στην R είναι αντικείμενο numeric (vector με μήκος 3) καθώς γίνεται αυτόματα μετατροπή του t2 της Python (το οποίο είναι τύπου list με αριθμητικές τιμές στοιχείων). Το py υποστηρίζει την πρόσβαση σε διάφορους τύπους αντικειμένων της Python, μέσω των μετατροπών που αναφέρθηκαν παραπάνω. Μεταξύ άλλων, το py δίνει πρόσβαση και στις συναρτήσεις που έχουν οριστεί στο κύριο περιβάλλον της Python. Έτσι, π.χ. η εντολή py$f(10) καλεί από την R κάποια συνάρτηση της Python με όνομα f, περνώντας το 10 ως τιμή παραμέτρου της.

στην τρέχουσα συνεδρία της R με εντολές όπως οι library(reticulate) ή require(reticulate).

[391] Συνάρτηση «περιτύλιγμα» ή wrapper function είναι μια συνάρτηση που απλώς κάνει μια προσαρμοσμένη κλήση άλλης συνάρτησης, εδώ η συνάρτηση R που δημιουργείται αυτόματα διαχειρίζεται (με μετατροπή παραμέτρων, επιστρεφόμενων τιμών κλπ.) την κλήση της συνάρτησης Python. Βλ. και υποσημείωση 274.

[392] Μετατρέπονται τα δημόσια αντικείμενα στο κύριο περιβάλλον της Python. Δεν μετατρέπονται αυτόματα όσα ανήκουν σε modules.

Στην πλευρά της Python το αντίστοιχο του py είναι το αντικείμενο r. Έτσι, είτε πρόκειται για εντολές Python που δίνονται μέσω της γραμμής εντολών (ενεργοποιείται με την εντολή repl_python()), είτε πρόκειται για κώδικα Python που εκτελείται με οποιονδήποτε άλλο τρόπο μέσω του 'reticulate', η Python μπορεί να έχει πρόσβαση στα αντικείμενα της R. Ο προσδιορισμός του ονόματος του αντικειμένου γίνεται με την τελεία ('.'), άρα για παράδειγμα η εντολή r.x εξάγει στη Python το αντικείμενο x της R ενώ η εντολή r.y=10 αναθέτει από την Python την τιμή 10 σε μεταβλητή y της R. Τα αντικείμενα της R στα οποία μπορεί να γίνει πρόσβαση από την Python συμπεριλαμβάνουν τύπους πέραν των βασικών, όπως data.frame και συναρτήσεις. Για παράδειγμα, ας ορίσουμε τα παρακάτω στην R:

```
rnv  <- c(1,5,10)
rdf  <- data.frame( a=c(1,2,4), b=c(2,2,4), c=c("a","a","b") )
rfn  <- function(x) {cat("Συνάρτηση της R, κλήθηκε με",x)}
```

Ενδεικτικά αποτελέσματα φαίνονται παρακάτω όπου δίνονται απευθείας εντολές στην Python (σημείωση: το '>>>' είναι σύμβολο προτροπής/prompt της γραμμής εντολών, δεν αποτελεί μέρος της εντολής):

```
>>> r.rnv
[1.0, 5.0, 10.0]
>>> type(r.rnv)
<class 'list'>
>>> pdf = r.rdf
>>> pdf
     a    b  c
0  1.0  2.0  a
1  2.0  2.0  a
2  4.0  4.0  b
>>> type(pdf)
<class 'pandas.core.frame.DataFrame'>
>>> r.rfn(30)
Συνάρτηση της R, κλήθηκε με 30
```

Σε πολλές περιπτώσεις η αξιοποίηση ενός αρχείου (σεναρίου) Python από την R υποβοηθείται ακόμα περισσότερο αν γίνει αυτόματη εξαγωγή των αντικειμένων Python σε αντίστοιχα της R, αναιρώντας έτσι την ανάγκη χρήσης της py για εισαγωγή των αντικειμένων αυτών. Η συνάρτηση source_python επιτρέπει την εκτέλεση ενός αρχείου Python και (προαιρετικά) τη δημιουργία σχετικών αντικείμενων στην R. Επιπροσθέτως, μπορεί και να προσδιοριστεί το environment που θα φιλοξενήσει τα νέα αντικείμενα, εφόσον αυτό δεν είναι το Global. Ας δούμε ένα παράδειγμα, ξεκινώντας από ένα σενάριο Python όπως το παρακάτω:

```
import numpy as np

p_nu = 10
p_li = ["EN", "DY", "TR", "TE"]
p_ar = np.array([[1, 1, 1], [2, 2, 2]], np.int32)
p_dc = {"Pop": 225, "City": "Athens"}

def p_fn(x):
  # print("Αυτή είναι μια συνάρτηση Python που")
  # print("κλήθηκε με",x,"και επιστρέφει",2*x)
  return(2*x)

class p_cl:
  def __init__(self,p1,p2):
    self.x = p1
    self.y = p2
  def total(self):
    return self.x+self.y

p_clm = p_cl(10,20)

print("Αυτό είναι το p_nu (αριθμός) από την Python:",p_nu)
```

```
print("Αυτό είναι το p_li (list) από την Python:\n",p_li)
print("Αυτό είναι το p_ar (array) από την Python:\n",p_ar)
print("Αυτό είναι το p_dc (dictionary) από την Python:\n",p_dc)
print("Κλήση p_fn (function) στην Python:",p_fn(10))
print("Κλήση μεθόδου κλάσης p_cl σε Python:", p_clm.total())
```

Το σενάριο αυτό ορίζει διάφορους τύπους αντικειμένων Python. Ας θεωρήσουμε πως το όνομα αρχείου του σεναρίου είναι “test.py” και βρίσκεται στον τρέχοντα φάκελο εργασίας[393]. Στην επόμενη εντολή χρησιμοποιείται η συνάρτηση source_python για να εκτελεστεί το σενάριο, εμφανίζοντας την έξοδο που παράγει η Python:

```
> source_python("test.py")
Αυτό είναι το p_nu (αριθμός) από την Python: 10
Αυτό είναι το p_li (list) από την Python:
 ['EN', 'DY', 'TR', 'TE']
Αυτό είναι το p_ar (array) από την Python:
 [[1 1 1]
 [2 2 2]]
Αυτό είναι το p_dc (dictionary) από την Python:
 {'Pop': 225, 'City': 'Athens'}
Κλήση p_fn (function) στην Python: 20
Κλήση μεθόδου κλάσης p_cl σε Python: 30
```

Μετά την εκτέλεση μιας εντολής source_python εξάγονται αυτόματα τα αντικείμενα από το περιβάλλον της Python και δημιουργούνται στην R[394] συνώνυμά τους αντικείμενα. Έτσι, αν δοθούν τα παρακάτω στο Console της R ανακαλούνται οι τιμές των σχετικών αντικειμένων της Python:

```
> p_nu
[1] 10
> p_li
[1] "EN" "DY" "TR" "TE"
> p_ar
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
> p_dc
$Pop
[1] 225

$City
[1] "Athens"

> p_fn(10)
[1] 20
> p_fn(c(1,2))
[1] 1 2 1 2
> p_clm$total()
[1] 30
```

Τα νέα αντικείμενα έχουν το ίδιο όνομα με αυτά της Python και συνήθως είναι αντίγραφα που δημιουργούνται μέσω μετατροπής τους σε αντίστοιχους τύπους της R. Εξαίρεση αποτελούν κάποια αντικείμενα Python όπως μέλη κλάσεων και συναρτήσεις. Για τις κλάσεις, το p_clm είναι εξαγωγή του ομώνυμου μέλους της custom κλάσης p_cl που ορίστηκε παραπάνω. Το αντίστοιχο αντικείμενο που δημιουργήθηκε στην R δεν είναι αντίγραφο αλλά αναφορά στο αρχικό αντικείμενο της Python. Μπορεί να γίνει άμεση πρόσβαση στα πεδία και τις μεθόδους του με τρόπο αντίστοιχο αυτών των κλάσεων RS (βλ.§6.5.1 Κλάσεις RS). Π.χ. η εντολή:

```
p_clm$y <- 2000
```

[393] βλ.§3.1.6 Φάκελοι αρχείων και ο τρέχων φάκελος εργασίας.

[394] Η συνάρτηση source_python κλήθηκε με προεπιλεγμένες τιμές: TRUE στην παράμετρό της convert, άρα γίνεται εξαγωγή αντικειμένων στην R και parent.frame() στην παράμετρο envir, άρα εδώ η εξαγωγή θα γίνει στο Global environment.

αλλάζει σε 2000 την τιμή του πεδίου (ιδιότητας) y του p_clm στο περιβάλλον της Python. Δεδομένου ότι η πρόσβαση μέσω του εξαχθέντος αντικειμένου σε μέλη κλάσεων γίνεται με αναφορά, η αλλαγή θα ισχύει τόσο στον χώρο της R όσο και της Python.

Για τις συναρτήσεις τώρα, ένα παράδειγμα εξαγωγής συνάρτησης είναι το p_fn στην R που προέκυψε από την ομώνυμη συνάρτηση Python. Αν ανακληθεί η p_fn στην R φαίνεται πως το εξαχθέν αντικείμενο είναι μια wrapper function[395]. Η συνάρτηση αυτή, αφού μετατρέψει τις τιμές παραμέτρων, καλεί τη συνάρτηση Python να επιτελέσει την εργασία.

Για να γίνει απρόσκοπτα η συνεργασία ανάμεσα στις δύο γλώσσες πρέπει να λαμβάνονται υπόψιν οι ιδιαιτερότητες καθεμίας από αυτές. Ως παράδειγμα, στις παραμέτρους συναρτήσεων (και μεθόδων) της Python κάποιοι τύποι αντικειμένων περνούν με τρόπο αντίστοιχο του call-by-value[396], οπότε η συνάρτηση δεν μπορεί να επιφέρει αλλαγές (οι οποίες θα ισχύουν μετά την ολοκλήρωσή της) στα αντικείμενα αυτά. Ταυτόχρονα, σε άλλους τύπους αντικειμένων η Python εφαρμόζει κλήση με αναφορά (call-by-reference)[397] άρα αλλαγές που ενδεχομένως κάνει η συνάρτηση εφαρμόζονται στο εξωτερικό αντικείμενο που δόθηκε ως τιμή παραμέτρου της και ισχύουν μετά την ολοκλήρωσή της. Παράδειγμα τέτοιων αντικειμένων είναι οι λίστες. Ας θεωρήσουμε λοιπόν την παρακάτω συνάρτηση Python που προσθέτει ένα αντικείμενο v σε μια λίστα l.

```
def p_fn2(l,v):
  l.append(v)
```

Αν κληθεί η συνάρτηση στην Python, οι αλλαγές στο l θα γίνουν στο εξωτερικό αντικείμενο, π.χ.:

```
>>> p_li = ["EN", "DY", "TR", "TE"]
>>> p_fn2(p_li, "PE")
>>> p_li
['EN', 'DY', 'TR', 'TE', 'PE']
```

Η R όμως λειτουργεί διαφορετικά, αποτρέποντας τις αλλαγές σε αντικείμενα του εξωτερικού περιβάλλοντος από μια συνάρτηση (όπως έχει ήδη αναφερθεί στη §5.1.5 Αλληλεπίδραση με το περιβάλλον και αλλού). Αυτό έχει το αποτύπωμά του και στη συμπεριφορά της παραπάνω συνάρτησης p_fn2 της Python όταν κληθεί από την πλευρά της R. Η συνάρτηση δεν θα επιφέρει αλλαγές στο εξωτερικό αντικείμενο. Αν π.χ. έχει χρησιμοποιηθεί η source_python και κληθεί η εξαχθείσα στην R συνάρτηση p_fn2:

```
> x <- c("EN", "DY", "TR", "TE")
> p_fn2(x, "PE")
> x
[1] "EN" "DY" "TR" "TE"
```

Το ίδιο ισχύει και αν χρησιμοποιηθεί απευθείας πρόσβαση στα αντικείμενα της Python μέσω του py:

```
> py$p_li
[1] "EN" "DY" "TR" "TE" "PE"
> py$p_fn2(py$p_li, "EX")
> py$p_li
[1] "EN" "DY" "TR" "TE" "PE"
```

Η εισαγωγή στο πακέτο ‘reticulate’ που έγινε σε αυτή την ενότητα δεν κάλυψε όλες τις δυνατότητές του. Το πακέτο είναι ένα πολύπλευρο εργαλείο που επιτρέπει να συνδυαστούν εύκολα τα δυνατά στοιχεία του οικοσυστήματος της Python με αυτά της R. Για τον λόγο αυτό υπάρχει ευρεία υιοθέτηση του από τους χρήστες των δύο γλωσσών αλλά και από άλλες ομάδες που δημιουργούν πακέτα της R στα οποία είναι απαραίτητη η ενσωμάτωση Python.

7.4 C++

Η σύνδεση της R με κώδικα C/C++ επιτρέπει την αξιοποίηση της ταχύτητας, χαρακτηριστικών και άμεσης πρόσβασης στο υπολογιστικό σύστημα που προσφέρουν οι γλώσσες αυτές. Σε αντίθεση με τις Tcl και Python με τις οποίες ασχολήθηκαν προηγούμενες ενότητες και εκτελούνται εντολή-εντολή μέσω διερμηνευτή (interpreter), ο κώδικας των C και C++ μεταφράζεται ολόκληρος από μεταγλωττιστή (compiler) οδηγώντας σε λειτουργικές μονάδες άμεσα εκτελέσιμες από το σύστημα. Η C++ είναι μια ιδιαίτερα ώριμη γλώσσα με

---

[395] βλ. παραπάνω υποσημείωση 391.

[396] Οι τύποι αυτοί ονομάζονται immutable ή unchangeable και περιλαμβάνουν αντικείμενα όπως μια αριθμητική τιμή, ένα κείμενο, ένα tupple, κ.α. Για το call-by-value βλ. υποσημείωση 295.

[397] Για το call-by-reference βλ. υποσημείωση 295.

τεράστιο αριθμό εφαρμογών και χρηστών αλλά και εργαλείων που την υποστηρίζουν. Κώδικας C/C++ αποτελεί τη βάση μεγάλου μέρους του λογισμικού που χρησιμοποιούμε καθημερινά. Η ίδια η R είναι γραμμένη σε C/C++, ενώ πολλά από τα πακέτα της περιέχουν κώδικα μεταφρασμένο από αυτές τις γλώσσες[398].

Υπάρχουν διεθνή (ISO) πρότυπα που αφορούν τη γλώσσα C++ με στόχο τη συμβατότητα των διαφορετικών compilers. Η γλώσσα προσφέρει ένα πανίσχυρο μοντέλο αντικειμενοστραφούς προγραμματισμού με metaprogramming, συνοδεύεται από τη standard template library (STL) για την εύκολη υλοποίηση χρήσιμων δομών δεδομένων με βέλτιστο τρόπο και υποστηρίζεται από μεγάλο αριθμό άλλων βιβλιοθηκών για μια ευρεία γκάμα εφαρμογών. Ταυτόχρονα η ίδια η C++ διατηρεί, σε μεγάλο βαθμό, τη συμβατότητα-προς-τα-πίσω (backward compatibility) με τον πρόγονό της, τη γλώσσα C. Έτσι επιτρέπει την αμεσότερη δυνατή πρόσβαση (για γλώσσα υψηλού επιπέδου) στο λειτουργικό σύστημα και το υλικό του υπολογιστικού συστήματος.

Οι βασικοί λόγοι που οδηγούν χρήστες της R στην ενσωμάτωση κώδικα C/C++ είναι τέσσερις: (α) η βελτίωση της ταχύτητας εκτέλεσης, (β) η αξιοποίηση των χαρακτηριστικών της C++ που διευκολύνουν την υλοποίηση πολύπλοκων δομών, κλάσεων και αλγορίθμων, (γ) η ανάγκη χρήσης λειτουργιών που έχουν ήδη υλοποιηθεί σε C/C++ και (δ) η ανάγκη άμεσης, χαμηλού επιπέδου, πρόσβασης στο σύστημα.

Με χρήση του πακέτου ‘Rcpp’[399] [84] [85] [86], η R συνεργάζεται εύκολα με τη γλώσσα C++, επιτρέποντας την ενσωμάτωσή κώδικα C/C++ σε σενάρια[400], έργα[401], πακέτα[402] κλπ. Επίσης, το RStudio παρέχει ικανό αριθμό βοηθημάτων για τη συγγραφή και εκτέλεση του σχετικού κώδικα.

Για να χρησιμοποιηθεί η C++ σε συνδυασμό με την R απαιτείται η εγκατάσταση του προαναφερθέντος πακέτου ‘Rcpp’ στην R αλλά και των εργαλείων C++ που διαθέτει το κεντρικό αποθετήριο της R. Το σχετικό αρχείο για τα Microsoft Windows ονομάζεται RTools [87] και περιέχει τον compiler και άλλα σχετικά προγράμματα και βιβλιοθήκες. Πρόκειται για ένα υποσύνολο του GNU compiler collection[403] και είναι το ίδιο σύστημα που χρησιμοποιείται για τη δημιουργία της ίδιας της R από πηγαίο κώδικα. Τα εργαλεία αυτά θα αναλάβουν τη μετάφραση του κώδικα C/C++. Οδηγίες εγκατάστασής υπάρχουν στο r-project.org, εκεί που διατίθενται και τα ίδια τα εργαλεία. Με εγκατεστημένα τα RTools, το πακέτο ‘Rcpp’ της R αναλαμβάνει αυτόματα την εκτέλεση τους όταν χρειάζεται να γίνει μετάφραση κώδικα C++ ενώ ταυτόχρονα παρέχει τα απαραίτητα εργαλεία και δομές που επιτρέπουν στα δύο μέρη, (R και μεταφρασμένο κώδικα C++), να συνεργαστούν.

Ο πλέον απλός τρόπος να εξαχθεί μια συνάρτηση C++ ώστε να μπορεί να κληθεί από κώδικα R είναι με τη συνάρτηση **cppFunction** του ‘Rcpp’[404]. Στο επόμενο παράδειγμα ορίζεται μια νέα συνάρτηση C++ με όνομα count_positives η οποία μετρά τα θετικά στοιχεία ενός αριθμητικού διανύσματος x προερχόμενο από την R:

```
cppFunction('
            int count_positives(NumericVector x)
            {
              int n = x.size();
              int p = 0;
              for(int i = 0; i < n; i++)
                if(x[i]>0) p++;
              return p;
            }
            ')
```

Η συνάρτηση cppFunction θα φροντίσει να γίνει μετάφραση του κώδικα C++ που δίνεται ως παράμετρος της πρώτης παραμέτρου της (code) και θα τοποθετήσει το αποτέλεσμα σε μια ομώνυμη wrapper function στην R. Εδώ ο κώδικας δόθηκε ως κείμενο (μέσα σε μονά εισαγωγικά, κάτι βολικό καθώς η C++ χρησιμοποιεί τα

---

[398] βλ. για παράδειγμα όσα αναφέρονται για τον ρόλο των wrapper functions ως διεπαφή με κώδικα C/C++ στην υποσημείωση 274.

[399] Διαθέσιμο από το CRAN.

[400] βλ.§3.1 Σενάρια (R-script).

[401] βλ.§3.1.7 Έργο (project).

[402] βλ. Κεφ. 8.

[403] Αντίστοιχα εργαλεία μπορούν να εγκατασταθούν σε MacOs. Συστήματα όπως το linux διαθέτουν προ-εγκατεστημένα τα σχετικά εργαλεία ή εγκαθιστούν τον GNU compiler μέσα από τα ενσωματωμένα εργαλεία διαχείρισης.

[404] Τα παραδείγματα R που ακολουθούν απαιτούν να έχει ήδη συνδεθεί το πακέτο ‘Rcpp’ στην τρέχουσα συνεδρία, με εντολές όπως library(Rcpp), require(Rcpp) κλπ.

διπλά εισαγωγικά για ορισμό κειμένου αλλά δεν χρησιμοποιεί τα μονά για τον σκοπό αυτό). Μετά την εκτέλεση της παραπάνω εντολής η count_positives είναι διαθέσιμη στην R, οπότε:

```
> x <- c(-1,2,5,-10,6,-2)
> count_positives(x)
[1] 3
```

Το ‘Rcpp’ παρέχει C++ κλάσεις για την ανταλλαγή αντικειμένων από και προς την R. Μια τέτοια κλάση είναι η NumericVector που ορίστηκε ως τύπος της παραμέτρου x στη συνάρτηση count_positives. Το NumericVector αντιστοιχεί σε ένα αντικείμενο numeric (ένα διάνυσμα σε numeric mode) της R. Στον κώδικα C++ του παραδείγματος χρησιμοποιούνται επίσης δυο από τις μεθόδους τις κλάσης NumericVector, η μέθοδος size που επιστρέφει τον αριθμό των στοιχείων (το length του vector στην R) και ο τελεστής [] για πρόσβαση σε συγκεκριμένα στοιχεία (με πρώτο στοιχείο αυτό στη θέση 0). Εκτός του NumericVector, από το πακέτο ‘Rcpp’ παρέχονται και άλλες ειδικές κλάσεις C++ όπως:

- IntegerVector, ComplexVector, LogicalVector, CharacterVector/StringVector, DateVector, DatetimeVector για διανύσματα που αντιστοιχούν στα integer, logical, complex, character, Date[405] και POSIXct[406] της R.
- NumericMatrix, IntegerMatrix, ComplexMatrix, LogicalMatrix, CharacterMatrix/StringMatrix για matrix[407] σε αντίστοιχο atomic mode της R.
- List και DataFrame για χειρισμό list[408] και data.frame[409] της R.
- S4 για τον χειρισμό αντικειμένων κλάσεων S4[410] της R. Το προαναφερθέν List μπορεί να χειριστεί κλάσεις S3[411].

Οι κλάσεις αυτές παρέχουν μεθόδους που μιμούνται τη συμπεριφορά κάποιων συναρτήσεων της R (όπως π.χ. για δημιουργία ακολουθιών, τυχαίων αριθμών κλπ. στα NumericVector) και μπορούν να χρησιμοποιηθούν ως τύποι παραμέτρων και επιστρεφόμενων τιμών μιας συνάρτησης ή μεθόδου C++ που επικοινωνεί με την R. Επιπρόσθετα, το ‘Rcpp’ εφαρμόζει μετατροπές από και προς scalar τύπους της C++ (int, bool, double, string, time_t, κ.α.), ενώ χειρίζεται και τις ασυμβατότητες που μπορεί να προκύψουν κατά την κλήση της συνάρτησης C++ με αντικείμενα R ως τιμές παραμέτρων. Στο παράδειγμα η επιστρεφόμενη τιμή είναι τύπου int και μετατρέπεται από το wrapper function που παρήγαγε η cppFunction του ‘Rcpp’ σε αντικείμενο integer της R.

Το ‘Rcpp’ παρέχει και άλλες υποστηρικτικές λειτουργίες χειρισμού της R, όπως εμφάνισης κειμένου στο Console, πρόσβασης σε αντικείμενα environment που έχουν οριστεί στην R, κλήσης συναρτήσεων R από τη C++ κ.α. Έχοντας ξεπεράσει το πρόβλημα επικοινωνίας με την R, μια συνάρτηση C++ μπορεί να αναλάβει μέρος της επεξεργασίας, εκτελώντας ταχύτατα μεταφρασμένο κώδικα ή να επιτελέσει ρόλο διεπαφής για το πέρασμα της επεξεργασίας σε «καθαρό» κώδικα C++ ο οποίος δεν χρειάζεται πλέον τις κλάσεις και υποστήριξη του ‘Rcpp’.

Όταν πρέπει να ενσωματωθούν ολόκληρα αρχεία C++ σε μια λύση βασισμένη σε R, το Rcpp παρέχει ένα σχετικό header (Rcpp.h) στο οποίο ορίζονται οι προαναφερθείσες κλάσεις (NumericVector, IntegerVector κλπ.) και οι υπόλοιπες υποστηρικτικές λειτουργίες, όλα τοποθετημένα στο namespace Rcpp. Παρακάτω είναι ένα παράδειγμα περιεχομένων ενός αρχείου C++ στο οποίο αναδεικνύονται μερικές ιδιαιτερότητες κατά τη χρήση του ‘Rcpp΄:

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector times_z_from_R(NumericVector x) {
  double u = Environment::global_env()["z"];
  return x * u;
}
```

405 βλ. help(as.Date).
406 βλ. help(as.POSIXct).
407 βλ.§4.1.3.1 Ο τύπος matrix (πίνακας 2 διαστάσεων).
408 βλ.§4.2.1 Ο τύπος list (λίστα).
409 βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων).
410 βλ.§6.4 Κλάσεις S4.
411 βλ.§6.3 Κλάσεις S3.

```
// [[Rcpp::export]]
NumericMatrix ya_diag(int d, double v) {
  NumericMatrix m = NumericMatrix(d,d);
  for(int i=0;i<d;i++)
    m(i,i)=v;           // παρατηρήστε τη χρήση () αντί []
  return m;
}


/*** R
z<-3
times_z_from_R(c(10,3,42))
ya_diag(5,2)
*/
```

Για να χρησιμοποιηθούν στον κώδικα τα παρεχόμενα από το 'Rcpp' εργαλεία, οι δύο πρώτες γραμμές ζητούν την ενσωμάτωση του σχετικού header (Rcpp.h) και δηλώνουν πως θα γίνει χρήση ορισμών που βρίσκονται στο namespace Rcpp. Ακολούθως, και για το παράδειγμα αυτό, ορίζονται δύο νέες συναρτήσεις C++ με όνομα times_z_from_R και ya_diag. Το ειδικό σχόλιο // [[Rcpp::export]] πριν τον ορισμό καθεμίας από τις συναρτήσεις αυτές ζητά να εξαχθούν στην R ώστε να μπορούν να κληθούν και από εκεί. Συναρτήσεις C++ χωρίς το σχόλιο αυτό δεν θα εξαχθούν. Ας δούμε τώρα τι κάνουν αυτά τα δύο παραδείγματα συναρτήσεων: η πρώτη συνάρτηση, times_z_from_R, δέχεται ένα numeric x και το πολλαπλασιάζει με κάποιο z το οποίο (κατά την κλήση της συνάρτησης) θα αναζητηθεί στο Global Environment της R. Καθώς το z (της R) πρέπει να ανατεθεί σε μεταβλητή double (που είναι scalar τύπος[412] της C++), το z που θα εντοπιστεί θα πρέπει να περιέχει μια μόνο τιμή και αυτή να είναι αριθμητική. Σε άλλη περίπτωση θα εγερθεί σφάλμα. Η δεύτερη συνάρτηση, ya_diag, δέχεται δυο παραμέτρους, την ακέραια d και την double v. Θα δημιουργήσει ένα τετραγωνικό matrix σε numeric mode (με διαστάσεις d γραμμές x d στήλες). Ο βρόχος for που εκτελείται στο σώμα της συνάρτησης θέτει τη τιμή v στα στοιχεία της διαγωνίου του matrix. Η πρόσβαση στα στοιχεία γίνεται με τη μέθοδο () του NumericMatrix όπου προσδιορίζεται γραμμή και στήλη του στοιχείου στο οποίο θα γίνει πρόσβαση[413]. Τέλος η συνάρτηση επιστρέφει το NumericMatrix που προέκυψε (άρα ένα matrix σε numeric mode στην R).

Το τελευταίο μέρος του παραπάνω παραδείγματος είναι ένα σχόλιο πολλαπλών γραμμών (τύπου /* … */ της C++). Εδώ είναι άλλη μια περίπτωση όπου σχόλια της C++ έχουν ειδική σημασία για την επεξεργασία μέσω 'Rcpp'. Συγκεκριμένα ένα σχόλιο που ξεκινά με /*** R σημαίνει πως τα υπόλοιπα περιεχόμενά του είναι κώδικας R και θα πρέπει να εκτελεστεί ως τέτοιος. Έτσι αν γίνει εκτέλεση του παραπάνω αρχείου C++ από την R (κάτι που μπορεί να γίνει με τη συνάρτηση **sourceCpp** του 'Rcpp' η με το σχετικό κουμπί του RStudio) το αποτέλεσμα θα είναι δύο νέες συναρτήσεις times_z_from_R και ya_diag διαθέσιμες στην R, ενώ θα εκτελεστεί και ο κώδικας R που βρίσκεται εντός του σχολίου εμφανίζοντας τα παρακάτω στο Console:

```
> z<-3

> times_z_from_R(c(10,3,42))
[1]  30   9 126

> ya_diag(5,2)
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    0    0    0    0
[2,]    0    2    0    0    0
[3,]    0    0    2    0    0
[4,]    0    0    0    2    0
```

[412] Η λέξη scalar χρησιμοποιείται συχνά στον προγραμματισμό για οτιδήποτε αποθηκεύει μια τιμή κάποιου βασικού τύπου δεδομένων.

[413] Για προσδιορισμό θέσης μέσω δύο δεικτών γραμμής και στήλης, χρησιμοποιείται το (). Με το [] και έναν δείκτη γίνεται πρόσβαση στο vector πάνω στο οποίο βασίζεται το αντικείμενο, π.χ. m[0] είναι το στοιχείο στη θέση 0,0 ενός αντικειμένου m που είναι NumericMatrix. LogicalMatrix, CharacterMatrix, κλπ. Τα αντικείμενα αυτά αντιστοιχούν στον τύπο matrix της R άρα η διάταξη των δεδομένων είναι ως προς τη στήλη, με τα πρώτα στοιχεία να είναι τα στοιχεία της 1ης στήλης, τα επόμενα της 2ης στήλης κλπ. Για περισσότερα βλ.§4.1.3.1 Ο τύπος matrix (πίνακας 2 διαστάσεων).

```
[5,]    0    0    0    0    2
```

Για να χρησιμοποιηθούν κλάσεις C++ σε κώδικα R, το ‘Rcpp’ τις εξάγει ως κλάσεις αναφοράς (βλ.§6.5.1 Κλάσεις RS). Για να γίνει αυτό πρέπει, αφού οριστεί η κλάση C++, να δημιουργηθεί ένα module που περιγράφει τις ιδιότητες (μεταβλητές μελών) και τις μεθόδους που εξάγονται στην R. Ακολουθεί ένα παράδειγμα:

```
class a_class
{
private:

  int m_data;

public:

  a_class()      { m_data=0; }
  a_class(int v) { set_data(v); }

  void set_data(int v) { m_data = v; }
  int get_data()       { return(m_data); }

  int add_to_data (int v)
    {
    m_data = m_data + v;
    return (m_data);
    }

  void show_data()
    { Rcout << "My data is "<< m_data <<"\n"; }
};
```

Η παραπάνω κλάση C++ δεν έχει κάποια ιδιαιτερότητα. Κάθε μέλος της κλάσης θα έχει μια μεταβλητή (m_data), ενώ ορίζονται δύο μέθοδοι δημιουργίας αντικειμένων της κλάσης (constructor), με κανένα ή ένα ακέραιο όρισμα. Ακολούθως ορίζονται δύο δημόσιες μέθοδοι πρόσβασης στο m_data (set_data και get_data), μια μέθοδος που προσθέτει έναν ακέραιο αριθμό στο m_data και το επιστρέφει. Τέλος υπάρχει μια μέθοδος (show_data) που εμφανίζει σε κείμενο την τιμή στο m_data του αντικειμένου. Εδώ υπάρχει μια μικρή αλλαγή από «καθαρό» κώδικα C++ καθώς χρησιμοποιήθηκε το Rcout, ένα stream που ορίζει το ‘Rcpp’. Το Rcout είναι παρεμφερές του std::cout αλλά οδηγεί την έξοδο κειμένου στο Console της R[414].

Για να χρησιμοποιηθεί μια κλάση C++ από την R θα πρέπει να οριστεί ένα “Rcpp Module” που καταγράφει τι (και πως) εξάγεται. Για την a_class που ορίστηκε παραπάνω, αυτό θα μπορούσε να γίνει προσθέτοντας μετά τον παραπάνω ορισμό της κλάσης κάτι όπως το ακόλουθο:

```
RCPP_MODULE(a_class) {
  class_<a_class>( "MyClass" )
  .constructor()
  .constructor<int>()
  .property("Data",&a_class::get_data,
                   &a_class::set_data,    "Access the data" )
  .method( "show", &a_class::show_data,   "Print the data"  )
  .method( "Add",  &a_class::add_to_data, "Add to the data" )
  ;
}
```

Εδώ περιγράφεται ότι ως διεπαφή με το a_class θα δημιουργηθεί μια κλάση της R με όνομα MyClass. Η συγκεκριμένη κλάση επιλέχτηκε να έχει δύο παραλλαγές constructor (αντιστοιχούν σε αυτούς της a_class), μια μεταβλητή μελών (property) με όνομα Data η τιμή της οποίας θα προκύπτει με κλήση της get_data, ενώ θα αλλάζει με κλήση της set_data. Θα υπάρχουν επίσης δύο μέθοδοι, show και Add που καλούν τις show_data και add_to_data του αντικειμένου a_class. Μετά την επεξεργασία του παραπάνω αρχείου από την R (με τη

[414] Για έξοδο κειμένου στο Console της R παρέχονται επίσης η συνάρτηση Rprintf (αντίστοιχη της printf) και το Rerr (παρεμφερές του std::cerr για μηνύματα λαθών).

συνάρτηση sourceCpp ή οποιονδήποτε άλλο τρόπο οδηγεί σε αυτό) το αποτέλεσμα είναι μια κλάση αναφοράς με όνομα MyClass που μπορεί να χρησιμοποιείται στην R με σύνταξη παρεμφερή αυτής των κλάσεων RS (βλ.§6.5.1 Κλάσεις RS). Για παράδειγμα:

```
> a<-new(MyClass,10)
> a$Add(40)
[1] 50
> a$show()
My data is 50

> b<-new(MyClass)
> b$Data <- 100
> b$Data <- b$Data + 30
> b$Add(20)
[1] 150
> print(b)
My data is 150
```

Για τη δημιουργία ενός αντικειμένου (όπως τα a και b, μέλη της κλάσης MyClass), χρησιμοποιείται η συνάρτηση new (του πακέτου ‘methods’ της R), ενώ έχοντας ορίσει στην κλάση μέθοδο με όνομα show αυτή αναλαμβάνει και τον χειρισμό της generic συνάρτησης print (όπως γίνεται και για τις κλάσεις RS).

Το πακέτο ‘Rcpp’ επιτρέπει την ολοκλήρωση κώδικα R με κώδικα C++. Μεγάλος αριθμός πρόσθετων πακέτων της R αξιοποιούν τις δυνατότητες που προσφέρονται από αυτό (για ενδεικτική λίστα βλ. [88])[415]. Οι δημιουργοί του ‘Rcpp’ παρέχουν εκτενές υλικό υποστήριξης (βλ. vignette(package="Rcpp") και [89]), μεταξύ αυτών έναν οδηγό αναφοράς στις λειτουργίες του πακέτου [90]. Χρήσιμη πηγή είναι ο ιστότοπος του πακέτου [88], το [91] είναι ένας ευανάγνωστος οδηγός χρήσης του ‘Rcpp’, ενώ στο [33] παρουσιάζονται διάφορα παραδείγματα χρήσης του πακέτου, με έμφαση στη βελτίωση του χρόνου εκτέλεσης σε υπολογιστικά προβλήματα.

[415] Ο συγγραφέας του βιβλίου έχει χρησιμοποιήσει το πακέτο ‘Rcpp’ για τη δημιουργία του πακέτου ‘nnlib2Rcpp’ [95].

Αναφορές Κεφαλαίου 7

[1] R Core Team (2021). R: A Language and Environment for Statistical Computing. Vienna, Austria. https://www.R-project.org/

[33] Wickham, H. (2019). *Advanced R (2nd Editton)*. Chapman and Hall/CRC. https://adv-r.hadley.nz/

[50] R Core Team (2021). Tcltk: Tcl/Tk Interface, R Package version 4.1.2. https://www.R-project.org/

[52] Verzani, J. (2022). gWidgets2: Rewrite of gWidgets API for Simplified GUI Construction. https://CRAN.R-project.org/package=gWidgets2

[71] rForce Team (2022). JRI - Java/R Interface. http://www.rforge.net/JRI/index.html

[72] Eddelbuettel, D., Francois, R., & Bachmeier, L. (2020). RInside: C++ Classes to Embed R in C++ (and C) Applications. https://CRAN.R-project.org/package=RInside

[73] Simon, U. (2021). rJava: Low-Level R to Java Interface. https://CRAN.R-project.org/package=rJava

[74] Fuller, T. P. (2018). rGroovy: Groovy Language Integration. https://CRAN.R-project.org/package=rGroovy

[75] Tcl Core Team, (2022). Tcl Developer Xchange. https://www.tcl.tk/

[76] Welch, B., & Jones, K. (2003). *Practical Programming in Tcl and Tk, 4th edition*. Pearson. ISBN 978-0130385604.

[77] Ousterhout, J. K., & Jones, K. (2009). *Tcl and the Tk Toolkit*. 2nd edition. Addison-Wesley Professional. ISBN: 978-0321336330.

[78] Tierney, L. (2021). tkrplot: TK Rplot. https://CRAN.R-project.org/package=tkrplot

[79] Campelo, F. (2021). tkRplotR: Display Resizable Plots https://CRAN.R-project.org/package=tkRplotR

[80] Python Software Foundation, «Python Language Reference,» 2022. url: http://www.python.org

[81] Van Rossum, G. (1995). Python tutorial, Technical Report CS-R9526. Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

[82] Belopolsky Alexander, Chapman Brad, Cock Peter, Eddelbuettel Dirk, Kluyver Thomas, Moreira Walter, Oget Laurent, Owens John, Rapin Nicolas, Slodkowicz Grzegorz, Smith Nathaniel, Warnes Gregory, the JRI author(s), the R authors et al. Documentation for rpy2. Ηλεκτρονικό. Προσπελάστηκε Απρίλιος, 2022, από https://rpy2.github.io/doc/v2.9.x/html/index.html

[83] Ushey, K., Allaire, J., & Tang, Y. (2022). Reticulate: Interface to 'Python'. https://CRAN.R-project.org/package=reticulate

[84] Eddelbuettel, D., & Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software, τόμ. 40, αρ. 8*, pp. 1-18. http://doi.org/10.18637/jss.v040.i08 .

[85] Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7. http://doi.org/10.1007/978-1-4614-6868-4 .

[86] Eddelbuettel, D., & Balamuta, J. J. (2018). Extending R with C++: A Brief Introduction to Rcpp. *The American Statistician, τόμ. 72, αρ. 1*, pp. 28-36. http://doi.org/10.1080/00031305.2017.1375990.

[87] RTools, Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://cran.r-project.org/bin/windows/Rtools/

[88] Eddelbuettel, D. et al., (2022, January). Rcpp: Seamless R and C++ Integration. Ηλεκτρονικό. https://www.rcpp.org/

[89] Eddelbuettel, D. (2022, February). Rcpp: Seamless R and C++ Integration. Ηλεκτρονικό. https://dirk.eddelbuettel.com/code/rcpp.html

[90] François, R., & Eddelbuettel, D. (2017). Rcpp Quick Reference Guide. https://dirk.eddelbuettel.com/code/rcpp/Rcpp-quickref.pdf

[91] Tsuda, M. E. (2020). Rcpp for everyone. Ηλεκτρονικό. https://teuder.github.io/rcpp4everyone_en/ [Πρόσβαση 2 2022].

Κεφάλαιο 8: Δημιουργία πακέτων

Σύνοψη
Το σχετικά μικρό αυτό κεφάλαιο καταπιάνεται με το σημαντικό θέμα, της δημιουργίας πακέτων επέκτασης για την R.

Προαπαιτούμενη γνώση
Εξοικείωση με τα βασικά στοιχεία προγραμματισμού στην R (Κεφ. 3 και 4), συναρτήσεις (Κεφ.5), θέματα συνεργασίας της R με άλλες γλώσσες (§7.1 Πολυγλωσσικές λύσεις).

8.1 Τι εξυπηρετούν τα πακέτα

Τα πακέτα, είναι σημαντικός παράγοντας στην επιτυχία της R, καθώς επεκτείνουν τις δυνατότητές της προσθέτοντας νέες λειτουργίες. Τα πακέτα είναι ένας τυποποιημένος τρόπος να συγκεντρωθούν συναρτήσεις, δεδομένα και άλλο περιεχόμενο που σχετίζονται μεταξύ τους, ενώ το γεγονός ότι συνδέονται και αποσυνδέονται δυναμικά από το σύστημα της R διευκολύνει τη διαχείρισή τους χωρίς να επιβαρύνει τη γλώσσα. Πακέτα δημιουργούνται με στόχο να γίνει διαμοιρασμός τους (συνήθως μέσα από κάποιο αποθετήριο[416]), ή για ιδιωτική χρήση, ή ακόμα και ως ένα έργο[417] με πρόσθετη εσωτερική δομή και δυνατότητες. Επιπροσθέτως, το ίδιο πακέτο μπορεί να δημιουργηθεί για διαφορετικά λειτουργικά συστήματα, επιτρέποντας τον διαμερισμό και τη χρήση των περιεχομένων του σε μεγαλύτερο εύρος υπολογιστών.

Το κεφάλαιο αυτό περιγράφει κάποιες διαδικασίες που μπορούν να εφαρμοστούν για να δημιουργηθεί ένα πακέτο. Στόχος είναι να αναδειχτεί ότι αυτό είναι εύκολα εφικτό. Οι πλήρεις οδηγίες του r-project για τη δημιουργία πακέτων περιέχονται στο [92]. Εκεί αναφέρονται πολλές πτυχές της δημιουργίας πακέτων που δεν θα παρουσιαστούν εδώ. Η δημιουργία πακέτων απασχολεί επίσης τα [93] και [94], που αποτελούν αξιόλογες πηγές για το θέμα και βοήθησαν στην οργάνωση του κεφαλαίου αυτού.

8.2 Δομή φακέλου για δημιουργία πακέτου

Ένα πακέτο δημιουργείται με βάση τα περιεχόμενα ενός φακέλου (υποκαταλόγου) στο σύστημα αρχείων του υπολογιστή. Τα περιεχόμενα του φακέλου αυτού πρέπει να είναι οργανωμένα σε υποφακέλους του, ανάλογα με τη φύση και τον σκοπό τους. Τέτοιοι φάκελοι είναι οι:

- R, για τον κώδικα R του πακέτου.
- man, για τη τεκμηρίωση (βοήθεια) που θα συνοδεύει το πακέτο και τις συναρτήσεις του.
- data, για δεδομένα.
- src, όπου θα τοποθετηθεί πηγαίος κώδικας C/C++ ή Fortran ο οποίος θα μεταγλωττιστεί.
- exec, όπου μπορεί να τοποθετηθεί πηγαίος κώδικας άλλων γλωσσών.
- inst, όπου μπορούν να τοποθετηθούν αρχεία τα οποία θα αντιγραφούν χωρίς περαιτέρω επεξεργασία. Όταν το πακέτο εγκατασταθεί, θα αντιγραφούν στο φάκελο εγκατάστασης του πακέτου[418].
- tests, όπου ορίζονται οι πρόσθετοι έλεγχοι που θα πρέπει να γίνονται κατά τη δημιουργία του πακέτου.
- vignettes, για έγγραφα που αφορούν τη δημιουργία εκτεταμένης τεκμηρίωσης (vignette).

Τα ονόματα των φακέλων είναι case-sensitive άρα δεν εξομοιώνονται τα κεφαλαία με τους αντίστοιχους μικρούς χαρακτήρες σε αυτά. Όλοι οι παραπάνω φάκελοι είναι προαιρετικοί. Αν δεν απαιτούνται από το συγκεκριμένο πακέτο, δεν χρειάζεται να υπάρχουν.

---

[416] βλ.§1.5 Χρήση και διαχείριση πακέτων.
[417] βλ.§3.1.7 Έργο (project).
[418] Καθώς η θέση τους μπορεί να ποικίλει ανάλογα με το σύστημα στο οποίο εγκαταστάθηκε το πακέτο, η διαδρομή επιστρέφεται από τη συνάρτηση system.file. Π.χ. η system.file("afolder","afile", package="apackage") επιστρέφει τη διαδρομή για το afile στον φάκελο afolder του inst που περιέχεται στον φάκελο εγκατάστασης του πακέτου apackage (τα afile, afolder, apackage είναι υποθετικά ονόματα).

Ο φάκελος του πακέτου μπορεί επίσης να περιέχει αρχεία ή φακέλους που δεν εμπλέκονται στην τελική δημιουργία του πακέτου. Αυτά μπορούν να εξαιρεθούν από τη διαδικασία. Για να γίνει αυτό, προστίθεται ένα αρχείο κειμένου με όνομα .Rbuildignore. Μέσα στο αρχείο αυτό σημειώνεται (με μορφή κανονικής έκφρασης[419]) οτιδήποτε στον φάκελο δεν αφορά τη δημιουργία του πακέτου, π.χ. μία γραμμή με το κείμενο ^paper εξαιρεί οτιδήποτε ονομάζεται paper, ενώ μία γραμμή με το κείμενο ^Meta$ οτιδήποτε τελειώνει σε Meta.

8.3 Αρχείο DESCRIPTION

Όπως αναφέρθηκε και παραπάνω, οι φάκελοι δεν απαιτούνται για να δημιουργηθεί το πακέτο. Το μόνο που απαιτείται για να δημιουργηθεί ένα πακέτο είναι ένα φάκελος που περιέχει ένα αρχείο κειμένου με όνομα DESCRIPTION. Το αρχείο αυτό καταγράφει με συγκεκριμένο τρόπο, το όνομα του πακέτου που θα δημιουργηθεί και άλλα στοιχεία που ίσως απαιτούνται. Ας δούμε ένα παράδειγμα από ένα πακέτο με όνομα nnlib2Rcpp [95]:

```
Package: nnlib2Rcpp
Type: Package
Title: Tools to define and create neural networks
Version: 0.1.9
Author: Vasilis Nikolaidis [aut, cph, cre]
(<https://orcid.org/0000-0003-1471-8788>)
Maintainer: Vasilis Nikolaidis <vnikolaidis@us.uop.gr>
Description: Contains versions of Autoencoder, BP, LVQ, MAM NN
and a module to define custom neural networks.
LinkingTo: Rcpp
Imports: Rcpp , methods
License: MIT + file LICENSE
Authors@R: person(given = "Vasilis", family = "Nikolaidis",
email = "vnikolaidis@us.uop.gr", role = c("aut", "cph", "cre"),
comment = c(ORCID = "0000-0003-1471-8788"))
URL: https://github.com/VNNikolaidis/nnlib2Rcpp
BugReports: https://github.com/VNNikolaidis/nnlib2Rcpp/issues
Encoding: UTF-8
Suggests: R.rsp
VignetteBuilder: R.rsp
```

Εδώ, μεταξύ άλλων, καταγράφεται το όνομα του πακέτου (Package), η έκδοση (Version), οι δημιουργοί και οι συντηρητές του (Author, Maintainer), μια περιγραφή (Description), τα πακέτα που χρησιμοποιεί (Imports), η άδεια χρήσης (License), τα δευτερεύοντα πακέτα που χρησιμοποιούνται εντός του πακέτου π.χ. σε παραδείγματα (Suggests), ο τρόπος που θα δημιουργηθούν τα συνοδευτικά έγγραφα (VignetteBuilder) κ.α.

8.4 Χτίσιμο ενός πακέτου

Για να περιγραφεί η διαδικασία «χτισίματος» (build) του πακέτου από τον φάκελο του, παρακάτω γίνεται η δημιουργία ενός - ουσιαστικά κενού - πακέτου με όνομα PARADEIGMA0. Στις οδηγίες για το παράδειγμα αυτό, θα αναφέρουμε τη διαδρομή προς τον φάκελο που γίνεται η εργασία ως Ψ, ενώ τη διαδρομή για την εγκατάσταση της ίδιας της R, ως φάκελο Ω. Τα βήματα είναι:

(α) Στον φάκελο Ψ (που εργαζόμαστε) δημιουργούμε έναν νέο υποφάκελο με όνομα PARADEIGMA0. Το όνομα δεν χρειάζεται να είναι αυτό του πακέτου, αλλά συνήθως επιλέγεται να είναι.

(β) Μέσα στον φάκελο PARADEIGMA0 του Ψ δημιουργούμε το μόνο απαραίτητο αρχείο, το DESCRIPTION (χωρίς προέκταση ονόματος, όχι π.χ. DESCRIPTION.txt). Με έναν οποιονδήποτε editor κειμένου όπως π.χ. το Σημειωματάριο (Notepad) ή τον editor του RStudio επεξεργαζόμαστε τα περιεχόμενα του. Ένα ελάχιστο περιεχόμενο για το DESCRIPTION θα ήταν κάτι όπως το ακόλουθο:

```
Package: PARADEIGMA0
```

[419] βλ.§2.3.3.2 Κανονικές εκφράσεις (regular expressions).

```
Type: Package
Title: Example of a package
Version: 0.0.1
Author: Vasilis Nikolaidis [aut]
Maintainer: Vasilis Nikolaidis <vnnikolaidis@gmail.com>
Description: A package build for nothing.
Encoding: UTF-8
License: GPL-3
```

(γ) Το πακέτο τώρα μπορεί να δημιουργηθεί με εντολές εκτός της R. Για το περιβάλλον των Windows, οι εντολές μπορούν να δοθούν από τη γραμμή εντολών (Command Prompt). Εκεί πρέπει πρώτα να γίνει αλλαγή του φακέλου στο Ψ (αντί του Ψ θα δοθεί η διαδρομή):

```
cd Ψ
```

Μετά πρέπει να εκτελεστεί η εντολή (όπου Ω η διαδρομή για την εγκατάσταση της η ίδιας της R, εφόσον δεν υπάρχει στο PATH):

```
Ω/bin/R.exe - CMD build PARADEIGMA0
```

Αν όλα πάνε καλά, το παραπάνω θα δημιουργήσει (στο Ψ) το πακέτο, χρησιμοποιώντας τα στοιχεία του DESCRIPTION. Εδώ, θα παράγει ένα αρχείο με όνομα PARADEIGMA0_0.0.1.tar.gz χρησιμοποιώντας όσα ορίστηκαν ως Package και Version. Το πακέτο αυτό μπορεί ήδη να εγκατασταθεί στην R. Όμως πριν εγκατασταθεί ένα πακέτο, καλό είναι να έχουν γίνει κάποιοι βασικοί έλεγχοι. Αυτό γίνεται με την εντολή (πάλι από τη γραμμή εντολών των Windows):

```
Ω/bin/R.exe - CMD check PARADEIGMA0_0.0.1.tar.gz
```

ή την αυστηρότερη παραλλαγή της (απαραίτητη αν στόχος είναι να διατεθεί μέσω CRAN):

```
Ω/bin/R.exe - CMD check --as-cran PARADEIGMA0_0.0.1.tar.gz
```

Εφόσον διορθωθούν τα λάθη (Errors) και προαιρετικά, οι προειδοποιήσεις (Warnings) και οι σημειώσεις (Notes) που αναφέρθηκαν κατά τον παραπάνω έλεγχο τότε το πακέτο είναι έτοιμο να εγκατασταθεί. Αυτό γίνεται μέσα από την R, με τη γνωστή συνάρτηση install.packages[420]:

```
install.packages("Ψ/PARADEIGMA0_0.0.1.tar.gz", repos = NULL,
type = "source")
```

Εναλλακτικά, μπορεί να εγκατασταθεί μέσα από το σχετικό εργαλείο εγκατάστασης πακέτων του RStudio.

8.5 Παραδείγματα δημιουργίας πακέτου

Ένα κενό πακέτο δεν είναι ιδιαίτερα χρήσιμο. Παρακάτω ακολουθούν παραδείγματα δημιουργίας ενός πακέτου που έχει κάποια περιεχόμενα, ακολουθώντας διαφορετικές προσεγγίσεις για να γίνει αυτό. Σε κάθε περίπτωση, ο κώδικας R του πακέτου στο παράδειγμα θα ορίζει (μόνο) μια συνάρτηση, την parad1 και κάποια δεδομένα, που στην προκειμένη περίπτωση είναι ένα διάνυσμα integer με όνομα dedom1. Η parad1 θα δημιουργείται με την εντολή:

```
parad1 <- function(x=0) { x + 10 }
```

ενώ το dedom1 με την εντολή:

```
dedom1 <- c(1:5,5:1)
```

Παρακάτω παρουσιάζονται κάποιοι τρόποι δημιουργίας του πακέτου αυτού.

8.5.1 Από αντικείμενα που ήδη υπάρχουν

Ο γρηγορότερος τρόπος για να δημιουργηθεί ένα πακέτο είναι από αντικείμενα που ήδη έχουν δημιουργηθεί και ανατεθεί σε μεταβλητές του Global Environment. Για παράδειγμα, ξεκινώντας από ένα κενό Global Environment, αν εκτελεστούν οι δύο εντολές που αναφέρθηκαν παραπάνω, τότε θα οριστούν οι μεταβλητές dedom1 και parad1. Τα αντικείμενα αυτά μπορούν να αποτελέσουν το περιεχόμενο ενός πακέτου με όνομα PARADEIGMA0 χρησιμοποιώντας τη συνάρτηση **package.skeleton** του προ-εγκατεστημένου πακέτου ‘utils’:

```
package.skeleton(name = "PARADEIGMA0")
```

Το παρακάτω θα δημιουργήσει έναν φάκελο με όνομα PARADEIGMA0 (μέσα στον τρέχοντα φάκελο εργασίας). Εκεί υπάρχει ένα αρχείο DESCRIPTION που μπορεί να τροποποιηθεί, ένας φάκελος data όπου είναι

[420] βλ.§1.5.3 Εγκατάσταση και διαχείριση πρόσθετων πακέτων.

αποθηκευμένο το dedom1 (ως αρχείο .rda[421], στο αρχείο dedom1.rda) και ένας φάκελος R όπου είναι αποθηκευμένο το parad1 (ως σενάριο R[422], στο αρχείο parad1.R). Τέλος, υπάρχει ένας φάκελος man που περιέχει τρία αρχεία τεκμηρίωσης, ένα για κάθε αντικείμενο (dedom1.Rd και parad1.Rd) και ένα για το πακέτο γενικότερα (PARADEIGMA1-package.Rd). Τα αρχεία αυτού του τύπου (.Rd) περιγράφονται περαιτέρω παρακάτω.

8.5.1.1 Αρχεία τεκμηρίωσης (.Rd)

Το πρότυπο που ακολουθεί η R για τα αρχεία της βασικής τεκμηρίωσης βασίζεται στο LaTeX. Σε αυτό έχουν προστεθεί ετικέτες προσαρμοσμένες για τον σκοπό της τεκμηρίωσης του αντικειμένου, του κειμένου δηλαδή που θα εμφανίζεται όταν καλείται η συνάρτηση help για αυτό.

Κατά τη δημιουργία του πακέτου, τα αρχεία «βοήθειας» (.rda) που δημιουργήθηκαν μέσα στον φάκελο man περιέχουν ένα προσχέδιο περιεχομένων. Οι δημιουργοί του πακέτου προσαρμόζουν το προσχέδιο αυτό στις ανάγκες τους. Παρακάτω προσαρμόστηκε η «βοήθεια» για τη συνάρτηση parad1 του πακέτου, δηλαδή το αρχείο parad1.Rd. Στόχος είναι να εξηγεί ότι η συνάρτηση προσθέτει 10 στο x, ότι το x πρέπει να είναι αριθμητικό, κλπ.

Τα αρχικά \name και \alias είναι όνομα και συνώνυμα που θα οδηγούν στη βοήθεια αυτή αν αναζητηθούν από τη συνάρτηση help. Ακολουθεί τίτλος (\title) και περιγραφή (\description), κλπ.

```
\name{parad1}
\alias{parad1}
\title{A function that adds 10}
\description{What more to say about a function that adds 10?}
\usage{parad1(x = 0)}
\arguments{
    \item{x}{a \code{numeric} or any other object that allows
addition.}
}
\value{Returns object of same type as x.}
\examples{
parad1(100:120)
parad1(as.complex(2:4))
}
```

Στο τμήμα \usage περιγράφεται ο τρόπος χρήσης της parad1 ενώ στο \arguments περιγράφονται οι παράμετροι (εδώ της μοναδικής παραμέτρου x). Το \value περιγράφει το αντικείμενο που επιστρέφει η συνάρτηση, ενώ το τελευταίο τμήμα (\examples) περιέχει κώδικα R, δηλαδή παραδείγματα χρήσης της συνάρτησης που μπορούν να εκτελεστούν (και θα εκτελεστούν αν κληθεί η συνάρτηση example για το αντικείμενο αυτό).

8.5.1.2 Αρχείο NAMESPACE

Όπως αναφέρθηκε σε άλλη ενότητα[423], όταν συνδέεται ένα πακέτο στην R (με συνάρτηση όπως η library), δημιουργεί (και χρησιμοποιεί) το δικό του environment όπου τοποθετούνται οι μεταβλητές για τα αντικείμενα που δημιουργεί. Στα αντικείμενα αυτά δεν υπάρχει πρόσβαση εκτός του πακέτου, εκτός αν έχουν εξαχθεί στον χώρο ονομάτων (namespace) που το πακέτο ορίζει. Κατά τη δημιουργία του πακέτου, στο αρχείο NAMESPACE ορίζονται τα αντικείμενα που θα υπάρχουν στο namespace του πακέτου, άρα αντικείμενα στα οποία θα έχουν πρόσβαση οι χρήστες του πακέτου και ο κώδικάς τους.

Ήδη, κατά τη δημιουργία του πακέτου με την package.skeleton, έγινε αυτόματα ένας ορισμός αντικειμένων που εξάγονται, θα μπορούν να αναζητηθούν μέσα στο namespace του πακέτου και να χρησιμοποιηθούν από τους χρήστες του. Για το παράδειγμα, ο αυτόματος αυτός ορισμός έγινε με βάση τα υπάρχοντα αντικείμενα, δημιουργώντας την παρακάτω καταχώρηση στο αρχείο NAMESPACE:

```
export("dedom1", "parad1")
```

---

[421] βλ.§9.1.2 Αποθήκευση και ανάκληση αντικειμένων
[422] βλ.§3.1 Σενάρια (R-script).
[423] βλ.§2.2.3 Ο ρόλος των περιβαλλόντων.

Προφανώς, οι δημιουργοί του πακέτου μπορούν να επιλέξουν να μην υπάρχει πρόσβαση από τους χρήστες σε κάποια από τα αντικείμενα, διαγράφοντάς αυτά από τη λίστα στο παραπάνω export, ή να προσθέσουν (εφόσον υπάρχουν) ονόματα άλλων αντικειμένων που πρέπει να εξαχθούν. Επίσης τα αντικείμενα που εξάγονται μπορούν να οριστούν με κανονική έκφραση[424] όπως στο παρακάτω:

```
exportPattern("^[[:alpha:]]+")
```

Με έναν ορισμό όπως ο παραπάνω, επιλέγεται να εξάγεται στο namespace οποιοδήποτε αντικείμενο υπάρχει στο πακέτο αυτό.

8.5.1.3 Ολοκλήρωση της διαδικασίας

Αφού γίνουν οι αλλαγές στα προαναφερθέντα αρχεία και οι αλλαγές που ενδεχομένως προτείνει το αρχείο Read-and-delete-me (το οποίο μετά θα πρέπει να διαγραφεί), το πακέτο μπορεί να δημιουργηθεί εφαρμόζοντας όσα αναφέρθηκαν στην §8.4 Χτίσιμο ενός πακέτου.

8.5.2 Από αρχεία R

Η διαδικασία δημιουργίας ενός πακέτου συνήθως περιλαμβάνει αρχεία που υπάρχουν ήδη. Σε αυτή την περίπτωση οι δημιουργοί του πακέτου πρέπει να τα τοποθετήσουν κατάλληλα σε έναν φάκελο που θα βασίζεται στη δομή που περιγράφεται παραπάνω (βλ.§8.2 Δομή φακέλου για δημιουργία πακέτου).

Για το παράδειγμά μας, ο φάκελος αρκεί να περιέχει έναν υποφάκελο R με το σενάριο που ορίζει τη συνάρτηση parad1 και έναν υποφάκελο data με το αρχείο .rda που περιέχει το dedom1. Η υπόλοιπη διαδικασία θα είναι παρόμοια με όσα αναφέρθηκαν στην προηγούμενη ενότητα. Όμως, ειδικά αν τα αρχεία υπάρχουν ήδη, η διαδικασία δημιουργίας ενός πακέτου μπορεί να υποβοηθηθεί με εργαλεία που παρέχει το πακέτο 'devtools'. Εγκατάσταση του πακέτου 'devtools' προσθέτει και άλλα πακέτα που αυτό χρησιμοποιεί όπως τα 'roxygen2' [96], 'usethis' [97] και 'testthat' [98]. Αφού γίνει η εγκατάσταση του 'devtools', τα βήματα έχουν ως εξής:

(α) Συνδέουμε στην R το πακέτο 'devtools':

```
library("devtools")
```

(β) Αφού ορίσουμε ως τρέχοντα φάκελο εργασίας τη θέση όπου θα δημιουργηθεί ο φάκελος του πακέτου, εκτελούμε την παρακάτω εντολή (εδώ το νέο πακέτο θα ονομάζεται PARADEIGMA2):

```
create("PARADEIGMA2")
```

Η συνάρτηση **create** παρέχεται από το 'devtools'. Η συνάρτηση αυτή όχι μόνο δημιουργεί τα βασικά στοιχεία του φακέλου που απαιτείται για δημιουργία του πακέτου αλλά και ένα έργο[425] για τον σκοπό αυτό, κάτι που διευκολύνει την εργασία ανάπτυξης του πακέτου μέσα από το RStudio.

(γ) Γίνονται όσες αλλαγές χρειάζονται στο DESCRIPTION (βλ.§8.3 Αρχείο DESCRIPTION).

(δ) Ακολούθως, αντιγράφουμε τα αρχεία στις κατάλληλες θέσεις μέσα στον νέο φάκελο. Έχοντας τα αρχεία, μπορεί να γίνει αξιοποίηση του πακέτου 'roxygen2' (που το 'devtools' χρησιμοποιεί) για αυτόματη δημιουργία τεκμηρίωσης από ειδικά σχόλια στον πηγαίο κώδικα. Ας υποθέσουμε για παράδειγμα ότι τα σχόλια στον ορισμό της συνάρτησης parad1 στο αρχείο parad1.R έχουν τη μορφή:

```
#' A function that adds 10
#'
#' @description What more to say about a function that adds 10?
#' @param x a \code{numeric} or any other object that allows
addition.
#' @return Returns object of same type as x.
#' @examples
#' parad1(100:120)
#' parad1(as.complex(2:4))
#' @export
parad1 <- function(x=0) { x + 10 }
```

Τα ειδικά αυτά αρχεία αφορούν τη δημιουργία της τεκμηρίωσης, ενώ το τελευταίο (@export) την εξαγωγή του αντικειμένου parad1 στο αρχείο NAMESPACE. Η εντολή που θα τα χρησιμοποιήσει είναι:

[424] βλ.§2.3.3.2 Κανονικές εκφράσεις (regular expressions).
[425] βλ.§3.1.7 Έργο (project).

```
document("PARADEIGMA2")
```

Η συνάρτηση **document** (του ‘devtools’) θα δημιουργήσει ένα αρχείο τεκμηρίωσης ίδιο με το προαναφερθέν parad1.Rd (βλ.§8.5.1.1 Αρχεία τεκμηρίωσης (.Rd)) και θα προσθέσει το parad1 στο αρχείο NAMESPACE (βλ.§8.5.1.2 Αρχείο NAMESPACE).

(ε) Το πακέτο μπορεί πλέον να ελεγχθεί και να εγκατασταθεί με τις συναρτήσεις **check** και **install** του ‘devtools’, όπως π.χ.

```
check("PARADEIGMA2")
install("PARADEIGMA2")
```

Επίσης πρέπει να αναφερθεί ότι το πακέτο ‘testthat’ (που εγκαθίσταται μαζί με το ‘devtools’) επιτρέπει μεταξύ άλλων τη δημιουργία τεστ ορθής λειτουργίας των συναρτήσεων (unit testing). Έτσι, κατά την εκτέλεση του ελέγχου από την check, θα εκτελούνται και έλεγχοι που έχουν οριστεί από τους δημιουργούς του πακέτου και θα ελέγχουν αν τα αποτελέσματα εφαρμογής των συναρτήσεων είναι τα αναμενόμενα.

8.5.3 Από το RStudio

Το RStudio διευκολύνει σημαντικά τη διαδικασία με έναν οδηγό δημιουργίας πακέτων. Αυτό ενεργοποιείται από την επιλογή μενού File/New Project, καθώς όλο το απαραίτητο υλικό του πακέτου θα δημιουργηθεί μέσα σε ένα νέο «έργο»[426]. Αφού επιλεγεί η διαδρομή του φακέλου για το έργο, εμφανίζονται επιλογές για δημιουργία είτε απλού πακέτου R, είτε πακέτου που χρησιμοποιεί άλλες δυνατότητες (όπως το πακέτο ‘Rcpp’ για ενσωμάτωση κώδικα C++[427]), είτε του πακέτου ‘devtools’ για πρόσθετα εργαλεία ανάπτυξης είτε άλλων πακέτων όπως το ‘roxygen’ και το ‘testthat’ που αναφέρθηκαν παραπάνω. Ο οδηγός επίσης επιτρέπει την εισαγωγή υπαρχόντων αρχείων ή τη δημιουργία ενός κενού πακέτου. Σε κάθε περίπτωση, μετά τη δημιουργία του σχετικού έργου, η δημιουργία, ο έλεγχος και η εγκατάσταση του πακέτου μπορεί να γίνει με τα σχετικά εργαλεία του μενού Build (ή της καρτέλας Build) στο RStudio.

[426] Όπως παραπάνω υποσημείωση 425.
[427] βλ.§7.4 C++.

Αναφορές Κεφαλαίου 8

[92] R Core Team, (2022). Writing R Extensions. https://cran.r-project.org/doc/manuals/R-exts.html

[93] Leisch, F. (2008). Creating R Packages: A Tutorial. Compstat 2008-Proceedings, Heidelberg, Germany.

[94] Wickham, H. (2015). *R Packages*. O'Reilly Media. ISBN 978-1491910597.https://r-pkgs.org/

[95] Nikolaidis, V. N. (2021). The nnlib2 library and nnlib2Rcpp R package for implementing neural networks. *Journal of Open Source Software, τόμ. 6, αρ. 61*, p. 2876. http://doi.org/10.21105/joss.02876.

[96] Wickham, H., Danenberg, P., Csárdi, G., & Eugster, M. (2021). Roxygen2: In-Line Documentation for R. https://CRAN.R-project.org/package=roxygen2

[97] Wickham, H., Bryan, J., & Barrett, M. (2021). Usethis: Automate Package and Project Setup. https://CRAN.R-project.org/package=usethis

[98] Wickham, H. (2011). Testthat: Get Started with Testing. *The R Journal, τόμ. 3, αρ. 1*, pp. 5-10. https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

Κεφάλαιο 9: Δεδομένα και περιεχόμενο

Σύνοψη
Το τελευταίο κεφάλαιο του βιβλίου απασχολείται με την εισαγωγή, δημιουργία και διαχείριση περιεχομένου στην R. Οι δυνατότητες που δίνουν τα πακέτα επέκτασης της R στον τομέα αυτό είναι τεράστιες, συνεχώς αυξάνονται και εξελίσσονται. Το κεφάλαιο αυτό κάνει μια παρουσίαση των βασικών τεχνικών σε θέματα όπως: εισαγωγή και εξαγωγή δεδομένων, δημιουργία γραφημάτων, δημιουργία υπηρεσιών ή εφαρμογών ιστού και δημιουργία δυναμικών εγγράφων.

Προαπαιτούμενη γνώση
Εξοικείωση με τα βασικά στοιχεία προγραμματισμού στην R (Κεφ. 3 και 4), συναρτήσεις (Κεφ.5).

9.1 Εισαγωγή και εξαγωγή δεδομένων

Εφαρμογές ανάλυσης δεδομένων μοιραία εμπλέκουν μεγάλο αριθμό δεδομένων, κάτι που δεν επιτρέπει την εισαγωγή τους από το πληκτρολόγιο. Πολλές συναρτήσεις που παρέχονται από τα πακέτα της R αναλαμβάνουν την εισαγωγή δεδομένων από εξωτερικές πηγές ή την εξαγωγή τους προς κάποιον αποδέκτη.

Χαμηλού επιπέδου συναρτήσεις δημιουργίας μιας σύνδεσης με κάποια πηγή ή αποδέκτη περιλαμβάνουν τις: **file** (για αρχεία), **url** (για web), **socketConnection** (για δικτυακές συνδέσεις) και άλλες συναφείς συναρτήσεις του πακέτου ‘base’[428]. Είτε πρόκειται για ανάγνωση από την πηγή, είτε για εγγραφή προς έναν προορισμό, η σύνδεση πρέπει να ανοίξει, ενώ μετά τη χρήση της να κλείσει. Ένα παράδειγμα εγγραφής σε αρχείο:

```
fc <- file("test1.txt")
open(fc, open = "wt")
writeLines("Κείμενο 1", con=fc)
writeLines("Κείμενο 2", con=fc)
close(fc)
```

Εδώ, δημιουργείται μια σύνδεση προς ένα αρχείο με όνομα “test1.txt”. Εφόσον δεν έχει οριστεί διαδρομή, το αρχείο θα δημιουργηθεί στον τρέχοντα φάκελο εργασίας429. Ο ορισμός του ονόματος και της διαδρομής του αρχείου, σε αυτό και σε άλλα παραδείγματα που ακολουθούν, θα μπορούσε να γίνει διαδραστικά με συναρτήσεις όπως η file.choose[430], ενώ όνομα για ένα προσωρινό αρχείο μπορεί να παραχθεί με τη συνάρτηση **tempfile**. Ακολούθως, με τη συνάρτηση open, η σύνδεση με το αρχείο ανοίγει για έξοδο κειμένου (“wt” σημαίνει write και text mode) και χρησιμοποιείται η writeLines[431] για την εγγραφή γραμμών κειμένου σε αυτό. Τέλος, το αρχείο κλείνει με τη συνάρτηση close. Το επόμενο παράδειγμα διαβάζει δύο γραμμές κειμένου από το αρχείο σε μεταβλητές x1 και x2. Η διαδικασία είναι παρόμοια, αλλά το αρχείο ανοίγει σε mode ανάγνωσης (“rt” σημαίνει read σε text mode) και χρησιμοποιείται η readLines432 για ανάγνωση του κειμένου (η παράμετρος n ορίζει τον αριθμό γραμμών που θα εισαχθούν):

```
fc <- file("test1.txt")
open(fc, open = "rt")
x1 <- readLines(con=fc,n = 1)
x2 <- readLines(con=fc,n = 1)
close(fc))
```

Αντί της writeLine για εγγραφή δεδομένων ως κείμενο θα μπορούσε να χρησιμοποιηθεί η cat, ενώ αντί της readLine, για ανάγνωση δεδομένων από πηγές κειμένου θα μπορούσε να χρησιμοποιηθεί η scan. Αυτές και άλλες συναρτήσεις (που περιγράφονται στη §2.3 Συναρτήσεις κειμένου και ο βασικός τύπος character) έχουν παράμετρο για προσδιορισμό της σύνδεσης που θα χρησιμοποιηθεί για κείμενο αντί του Console. Αν τα δεδομένα πρέπει να διαχειριστούν ως δυαδικές τιμές (binary) και όχι ως κείμενο, το πακέτο ΄base’ διαθέτει τις συναρτήσεις **readBin** και **writeBin** για ανάγνωση και εγγραφή αντίστοιχα.

---
[428] Βλ. help(connections).
[429] βλ.§3.1.6 Φάκελοι αρχείων και ο τρέχων φάκελος εργασίας.
[430] βλ.§3.3 Στοιχεία διεπαφής χρήστη (user interface).
[431] βλ.§2.3.2 Έξοδος και εμφάνιση κειμένου.
[432] βλ.§2.3.4 Είσοδος κειμένου.

Συναρτήσεις όπως αυτές που αναφέρθηκαν παραπάνω και παρέχονται από το ‘base’, επιτρέπουν τον χειρισμό συνδέσεων σε βασικό (χαμηλού-επιπέδου) επίπεδο. Όμως σπάνια υπάρχει ανάγκη χρήσης τους, καθώς άλλα πακέτα περιέχουν ισχυρότερες, παραμετροποιήσημες, ειδικού-σκοπού συναρτήσεις ανάγνωσης και εγγραφής δεδομένων που συνήθως επαρκούν. Επιπροσθέτως, το RStudio απλοποιεί τη δημιουργία συνδέσεων και την εισαγωγή δεδομένων από διάφορες πηγές, εγκαθιστώντας τα απαραίτητα πακέτα και δημιουργώντας μέσω οδηγών (παράθυρων wizard) τις απαραίτητες εντολές R (που μπορούν να χρησιμοποιηθούν και αργότερα για τον ίδιο σκοπό, εντός ή εκτός RStudio). Στο RStudio αυτό γίνεται από την καρτέλα Environment[433] που παρέχει την επιλογή Import Dataset[434] για την εισαγωγή δεδομένων από αρχεία κειμένου, αρχείων XML, αρχείων λογιστικών φύλλων Excel, αρχείων στατιστικών πακέτων SPSS, SAS, Stata, κλπ.). Αν χρειάζεται, το RStudio καθοδηγεί και την εγκατάσταση των απαραίτητων πακέτων που παρέχουν τις συναρτήσεις, όπως του πακέτου ‘readxl’ [99] για ανάγνωση αρχείων Excel[435]. Με την εκτέλεση του οδηγού (wizard) το RStudio συνθέτει τις κατάλληλες εντολές χρησιμοποιώντας π.χ. για αρχεία κειμένου, συναρτήσεις όπως οι **read.table** ή **read.delim** κλπ. του πακέτου ‘utils’ (βλ. παρακάτω) ή τις αντίστοιχες **read_table**, **read_delim** κλπ. του πακέτου ‘readr’ [100] (μέρος του ‘tidyverse’ [57]). Με παρόμοιο τρόπο γίνεται και η δημιουργία συνδέσεων με εξωτερικές πηγές όπως οι βάσεις δεδομένων. Το RStudio υποβοηθά την παραμετροποίηση της σύνδεσης από την καρτέλα Connections[436] και δημιουργεί τις κατάλληλες εντολές χρησιμοποιώντας συναρτήσεις πακέτων όπως το ‘DBI’ [101], εντολές που μπορούν να εκτελεστούν άμεσα, να ενσωματωθούν σε σενάρια R κλπ. Για επεξεργασία των δεδομένων από τις συνδέσεις αυτές μπορούν να χρησιμοποιηθούν συναρτήσεις του ίδιου του πακέτου ‘DBI’, ή άλλων πακέτων όπως το ‘dbplyr’ [102] που επιτρέπει τη χρήση συναρτήσεων του πακέτου ‘dplyr’ [55][437] σε βάσεις δεδομένων.

Τα εργαλεία αυτά, οι σχετικοί οδηγοί του RStudio, όπως και πολλά από τα πακέτα πρόσβασης σε εξωτερικά δεδομένα που έχουν δημιουργηθεί (και συνεχώς δημιουργούνται ή επικαιροποιούνται) για την R δεν θα αναλυθούν στην ενότητα αυτή. Παρακάτω γίνεται μόνο μία εισαγωγική παρουσίαση κάποιων θεμάτων σχετικών με εξωτερικές πηγές δεδομένων με χρήση επιλεγμένων παραδειγμάτων.

9.1.1 Δεδομένα σε κείμενο

Για ανάγνωση και εγγραφή δεδομένων, το προ-εγκατεστημένο πακέτο ‘utils’ παρέχει διάφορες συναρτήσεις (με πρόθεμα read και write στο όνομά τους). Κάποιες από τις συναρτήσεις αυτές αφορούν ανάγνωση και εγγραφή δεδομένων σε μορφή πίνακα (tabular) όπως αντικείμενα matrix[438] και data.frame[439] σε αρχεία απλού κειμένου (με την κωδικοποίηση των χαρακτήρων να μπορεί να οριστεί μέσω των παραμέτρων τους).

Η χρήση αρχείων κειμένου ως πηγή δεδομένων έχει κάποια θετικά στοιχεία, μεταξύ αυτών:

- Είναι ο πλέον γενικός τρόπος να αποθηκευτούν δεδομένα χωρίς καμία εξάρτηση από συγκεκριμένα προγράμματα για την επεξεργασία τους.
- Τα προγράμματα που χειρίζονται δεδομένα (μεταξύ αυτών τα προγράμματα υπολογιστικών φύλλων, όπως το Excel, τα συστήματα διαχείρισης βάσεων δεδομένων κλπ.) παρέχουν δυνατότητα εισαγωγής και εξαγωγής δεδομένων από και προς αρχεία κειμένου.
- Πολλές διαδικτυακές πηγές δεδομένων παρέχουν δεδομένα σε αρχεία κειμένου.
- Είναι αναγνώσιμα από τον άνθρωπο και μπορεί να γίνει επεξεργασία τους με συνήθεις editors κειμένου όπως π.χ. το Σημειωματάριο (Notepad) στα Windows.
- Είναι ο προτεινόμενος τρόπος να αποθηκεύονται δεδομένα σε συστήματα βασισμένα σε unix.

Υπάρχουν όμως και αρνητικά στοιχεία. Η αποθήκευση ως κείμενο δεν είναι ιδιαίτερα αποδοτική όσον αφορά τον απαιτούμενο αποθηκευτικό χώρο και χρόνο επεξεργασίας. Επίσης, οι διάφορες συμβάσεις που χρησιμοποιούνται κατά την εγγραφή κειμένου (π.χ. τρόπος καταγραφής χιλιάδων ή δεκαδικών ψηφίων σε αριθμούς, κωδικοποιήσεις χαρακτήρων κ.α.) μπορεί να είναι άγνωστες κατά την ανάγνωση, προκαλώντας προβλήματα. Τα αρχεία κειμένου δεν είναι σχεδιασμένα για να αποθηκεύουν δεδομένα, είναι ο πλέον απλός και γενικός τρόπος να αποθηκευτεί οποιοδήποτε κείμενο. Άρα τα αρχεία αυτά δεν επιβάλλουν κάποια

[433] βλ.§1.4.2.7 Η καρτέλα Environment.
[434] Η σχετική επιλογή μενού είναι File/Import Dataset.
[435] Τα πακέτα που αναφέρονται στην ενότητα αυτή είναι διαθέσιμα στο CRAN.
[436] βλ.§1.4.2.5 Η καρτέλα Connections.
[437] Για το πακέτο ‘dplyr’ βλ. και §5.4 Η συναρτησιακή προσέγγιση στον πραγματικό κόσμο.
[438] βλ.§4.1.3.1 Ο τύπος matrix (πίνακας 2 διαστάσεων).
[439] βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων).

συγκεκριμένη δομή στα περιεχόμενα τους. Για να διευκολυνθεί ο εντοπισμός και η ανάκτηση δεδομένων από ένα αρχείο κειμένου, πρέπει το κείμενο να ακολουθεί κάποιους κανόνες που θα είναι γνωστοί κατά την ανάκτηση των δεδομένων από αυτό.

Έτσι, για δεδομένα σε μορφή πίνακα, η πρώτη γραμμή (το header ή heading στην ορολογία της R) μπορεί να περιέχει τα ονόματα των πεδίων (στηλών, μεταβλητών) ενώ κάθε επόμενη γραμμή θα αφορά μια εγγραφή χωρισμένη σε ομοειδή πεδία. Το παρακάτω είναι ένα παράδειγμα δεδομένων όπου κάθε γραμμή περιέχει τα ίδια πεδία (περιοχή, όνομα, επώνυμο, διεύθυνση, υπόλοιπο).

```
151 ΝΙΚΟΣ ΑΛΕΞΙΟΥ ΦΙΛΕΛΛΗΝΩΝ 12 213.4
2197 ΚΩΣΤΑΣ ΠΑΥΛΟΥ ΛΥΚΟΥΡΓΟΥ 12 42333.3
031 ΓΙΩΡΓΟΣ ΔΗΜΗΤΡΙΟΥ ΜΠΟΥΛΟΥΚΟΥ 8 2
```

Όμως, που τελειώνουν τα δεδομένα κάθε πεδίου; Οι δύο συνήθεις τρόποι να διαχωριστούν τα πεδία κάθε εγγραφής όταν αποθηκεύονται σε αρχεία κειμένου είναι:

(α) Αποθηκεύοντας κάθε εγγραφή ως κείμενο με πεδία σταθερού μήκους. Σε αρχεία κειμένου όπου τα πεδία των δεδομένων είναι σταθερού μήκους, κάθε πεδίο καταλαμβάνει προκαθορισμένο αριθμό χαρακτήρων (ενώ το υπόλοιπο γεμίζει π.χ. με κενά). Παράδειγμα πιθανού περιεχομένου σε αρχείο κειμένου που περιέχει πεδία σταθερού μήκους είναι:

```
 151   ΝΙΚΟΣ        ΑΛΕΞΙΟΥ          ΦΙΛΕΛΛΗΝΩΝ 12        213.4
2197   ΚΩΣΤΑΣ       ΠΑΥΛΟΥ           ΛΥΚΟΥΡΓΟΥ 12       42333.3
 031   ΓΙΩΡΓΟΣ      ΔΗΜΗΤΡΙΟΥ        ΜΠΟΥΛΟΥΚΟΥ 8           2.0
```

Η εγγραφή τέτοιων δεδομένων μπορεί να γίνει με χρήση της συνάρτησης sprintf[440] σε συνδυασμό με τη writeLines, ενώ η ανάγνωση με συναρτήσεις όπως η **read.fwf** και **read.fortran** του πακέτου 'utils'.

(β) Αποθηκεύοντας κάθε εγγραφή ως κείμενο με πεδία μεταβλητού μήκους. Σε αρχεία κειμένου όπου τα δεδομένα είναι μεταβλητού μήκους κάποιος χαρακτήρας επιλέγεται και χρησιμοποιείται ως διαχωριστικό (delimiter ή separator) των πεδίων. Πολύ συχνά ο χαρακτήρας αυτός είναι το κόμμα, το tab, ή το κενό. Ένα παράδειγμα πιθανού περιεχομένου αν αποθηκευτεί ο παραπάνω πίνακας σε αρχείο κειμένου με πεδία μεταβλητού μήκους (εδώ με διαχωριστικό πεδίων το @) είναι:

```
151@ΝΙΚΟΣ@ΑΛΕΞΙΟΥ@ΦΙΛΕΛΛΗΝΩΝ 12@213.4
2197@ΚΩΣΤΑΣ@ΠΑΥΛΟΥ@ΛΥΚΟΥΡΓΟΥ 12@42333.3
031@ΓΙΩΡΓΟΣ@ΔΗΜΗΤΡΙΟΥ@ΜΠΟΥΛΟΥΚΟΥ 8@2
```

Τα αρχεία που χρησιμοποιούν τον χαρακτήρα κόμμα (,) για διαχωρισμό ονομάζονται comma-separated values (αρχεία csv). Για τα αρχεία αυτά μπορούν να χρησιμοποιηθούν συναρτήσεις όπως οι **read.csv** και **write.csv** (ή **read.csv2** και **write.csv2** για χρήση άλλου χαρακτήρα αν το κόμμα θα είναι το σύμβολο δεκαδικού μέρους των αριθμών στο αρχείο). Αν χρησιμοποιείται ο χαρακτήρας tab για τον διαχωρισμό, τα αρχεία ονομάζονται tab-delimited. Η γενική συνάρτηση για ανάγνωση αρχείων κειμένου όπου έχει οριστεί κάποιος χαρακτήρας διαχωρισμού των πεδίων είναι η **read.delim** (και **read.delim2**), ενώ για εγγραφή η **write.delim**. Όλες βασίζονται στις **read.table** και **write.table** του πακέτου 'utils'. Ακολουθεί ένα παράδειγμα χρήσης τους, όπου θα αποθηκευτούν (και μετά θα ανακληθούν) τα δεδομένα στο παρακάτω data.frame:

```
d1 <- data.frame(
          Μα = c("α", "β", "α", "γ", "β", "β"),
          Μβ = c(2.2, 1.9, 12,  2.5,  23, 31),
          Μγ = c(  1,   1,  3,   NA,   5,  5))
```

Η εντολή για να αποθηκευτεί σε αρχείο csv με όνομα "test2.txt" χωρίς ονόματα ή αρίθμηση γραμμών (καθώς στο d1 δεν έχουν οριστεί ονόματα για τις γραμμές) είναι:

```
write.csv(d1, file='test2.txt', row.names = F)
```

Τα περιεχόμενα του αρχείου 'test2.txt' που δημιουργήθηκε είναι:

```
"Μα","Μβ","Μγ"
"α",2.2,1
"β",1.9,1
"α",12,3
"γ",2.5,NA
"β",23,5
"β",31,5
```

Η ανάγνωση του αρχείου 'test2.txt' σε data.frame με όνομα d2 μπορεί να γίνει με εντολή όπως η:

440 βλ.§2.3.1 Σύνθεση κειμένου.

```
d2<-read.csv('test2.txt')
```
Και τα δεδομένα στο αντικείμενο d2 είναι πλέον ίδια με αυτά του αρχικού d1:
```
all.equal(d1,d2)
[1] TRUE
```
Στο αρχείο, ο διαχωρισμός έγινε με κόμμα. Επειδή σε πολλές χώρες (όπως και στην Ελλάδα) είναι σύνηθες να χρησιμοποιείται το κόμμα ως σύμβολο δεκαδικού, η παρακάτω είναι μια παραλλαγή του προηγουμενου παραδείγματος, όπου ορίζουμε επιπρόσθετα το κείμενο που θα χρησιμοποιηθεί ως ένδειξη της ειδικής τιμής NA και την κωδικοποίηση χαρακτήρων σε UTF-8. Για την εγγραφή:
```
write.csv2(d1, file='test2.txt', row.names = F,
                  na="Δ/Υ", fileEncoding = "UTF-8")
```
Τα νέα περιεχόμενα του αρχείου 'test2.txt' που δημιουργήθηκε είναι:
```
"Mα";"Mβ";"Mγ"
"α";2,2;1
"β";1,9;1
"α";12;3
"γ";2,5;Δ/Υ
"β";23;5
"β";31;5
```
Ενώ η ανάγνωση του αρχείου 'test2.txt' στο d2 μπορεί να γίνει με εντολή όπως η:

```
d2<-read.csv2(file='test2.txt',dec=',',header = T,
                 na="Δ/Υ", fileEncoding = "UTF-8")
```
Τα παραπάνω είναι εφαρμόσιμα σε αντικείμενα matrix και data.frame. Οι διάφοροι άλλοι τύποι πινακοειδών (tabular) αντικειμένων όπως οι tibble, data.table και Matrix (βλ.§4.3 Άλλοι τύποι αντικειμένων) παρέχουν τις δικές τους μεθόδους ανάγνωσης και αποθήκευσης σε αρχεία, οι οποίες συχνά έχουν βελτιωμένα χαρακτηριστικά σε σχέση με τις συναρτήσεις που αναφέρθηκαν εδώ.

9.1.2 Αποθήκευση και ανάκληση αντικειμένων

Μπορεί να μη θεωρείτε τα αντικείμενα R ως δεδομένα, αλλά σίγουρα υπάρχουν αντικείμενα που περιέχουν δεδομένα. Ένα ή περισσότερα αντικείμενα μπορούν να αποθηκευτούν σε ένα αρχείο της R (συνήθως με όνομα που έχει επέκταση .rds) με τη συνάρτηση save και να ανακληθούν με τη load (ή με απευθείας άνοιγμα του αρχείου από το RStudio). Για παράδειγμα (χρησιμοποιώντας το d1 που ορίστηκε παραπάνω):
```
save(d1, "test3.rda")
```
Η εντολή θα αποθηκεύσει το αρχείο στον τρέχοντα φάκελο εργασίας. Τα ονόματα των αντικειμένων είναι αποθηκευμένα στο αρχείο, οπότε τα αντικείμενα θα επαναδημιουργηθούν αυτόματα αν εκτελεστεί η εντολή:
```
load("test3.rda")
```
Επίσης, για τοπική αποθήκευση σε αρχείο ενός αντικειμένου οποιουδήποτε τύπου, μπορεί να χρησιμοποιηθεί η συνάρτηση **saveRDS** του πακέτου 'base'. Έτσι (χρησιμοποιώντας πάλι το d1 που ορίστηκε παραπάνω):
```
saveRDS(d1, "test4.RDS")
```
Η ανάκληση του αντικειμένου από το αρχείο σε άλλη μεταβλητή d3 γίνεται με τη **readRDS**:
```
d3<- readRDS("test4.RDS")
```
Όπως αναφέρει η τεκμηρίωση[441], οι συναρτήσεις αυτές δεν συστήνονται για διαμοιρασμό του αντικειμένου σε άλλους υπολογιστές (άρα είναι για τοπική αποθήκευση). Μια εναλλακτική επιλογή είναι να αποθηκευτεί η πλήρης δομή του αντικειμένου ως κείμενο με τη συνάρτηση **dput**[442] και προσδιορισμό του αρχείου προορισμού:
```
dput(d1,"test5.txt")
```
Με την περιγραφή του αντικειμένου αποθηκευμένη στο αρχείο, το αντικείμενο μπορεί να δημιουργηθεί ξανά, χρησιμοποιώντας τη συνάρτηση **dget**:
```
d4<-dget("test5.txt")
```

---

[441] βλ. help(readRDS).

[442] βλ.§2.7 Οι συναρτήσεις-βοηθήματα: str, summary και dput.

9.1.3 Ανάγνωση δεδομένων web

Για πηγές από το web, ένας προφανής - χαμηλού επιπέδου - τρόπος εισαγωγής τους είναι η δημιουργία μιας σύνδεσης url και η χρήση τεχνικών όπως αυτές που αναφέρθηκαν στην αρχή της ενότητας (§9.1 Εισαγωγή και εξαγωγή δεδομένων):

```
wc <- url("https://www.uop.gr/")
open(wc, open = "rt")
p <- readLines(con=wc, n = 20)
close(wc)
```

Το παραπάνω, διαβάζει 20 γραμμές από τον html κώδικα της καθορισμένης στο url σελίδας στη μεταβλητή character με όνομα p. Όμως, για ουσιαστικότερη εκμετάλλευση των δεδομένων απαιτούνται πρόσθετα πακέτα όπως το ‘rvest’ [103], που παρέχει συναρτήσεις για τον εντοπισμό συγκεκριμένων τμημάτων της σελίδας, εξαγωγή του κειμένου κλπ. Πληθώρα άλλων πακέτων δίνουν πρόσβαση σε ιστοσελίδες ή άλλες διαδικτυακές πηγές δεδομένων. Ενδεικτικά αναφέρουμε το ‘rtweet’ για αναζήτηση και ανάκτηση περιεχομένου των αναρτήσεων στο κοινωνικό δίκτυο tweeter, ή το ‘quantmod’ [104] για ανάκτηση δεδομένων οικονομικού ενδιαφέροντος από διάφορες σχετικές διαδικτυακές πηγές. Ένα παράδειγμα με το πακέτο ‘quantmod’ είναι η παρακάτω εντολή που χρησιμοποιεί τη συνάρτηση του πακέτου **getSymbols** για να διαβάσει ως χρονοσειρά (αντικείμενο ts[443]) τον πληθυσμό της Ελλάδας από την πηγή FRED (St. Louis Federal Reserve Bank's FRED system). Μετά, εμφανίζει τη χρονοσειρά σε διάγραμμα (με τη συνάρτηση plot):

```
getSymbols("POPTOTGRA647NWDB",  src = "FRED")
plot(POPTOTGRA647NWDB,main="Population in Greece")
```

Αντίστοιχα το επόμενο παράδειγμα διαβάζει δεδομένα για τη μετοχή της εταιρίας Apple:

```
getSymbols("AAPL", src = "yahoo", from = start)
candleChart(AAPL,theme='white', type='candles')
```

Εδώ πηγή είναι το Yahoo, ενώ το γράφημα που ακολουθεί δημιουργείται με τη συνάρτηση **candleChart**, μία από τις υποστηρικτικές συναρτήσεις που παρέχεται από το ίδιο πακέτο.

9.1.4 Ένα εργαλείο για πολλές μορφές δεδομένων

Ένα πακέτο που διευκολύνει την εισαγωγή και την εξαγωγή δεδομένων σε διάφορες μορφές (format) είναι το ‘rio’ [105]. Το πακέτο αυτό παρέχει τρείς συναρτήσεις **import**, **convert** και **export**, οι οποίες -χωρίς ιδιαίτερη παραμετροποίηση- αναλαμβάνουν την ανάγνωση, τη μετατροπή και την αποθήκευση από και προς πληθώρα διαφορετικών πηγών και μορφών[444]. Το πακέτο συγκεντρώνει τις δυνατότητες που παρέχουν άλλα πακέτα τα οποία χρησιμοποιεί και τις προσαρμόζει σε ένα κοινό πρότυπο εντολών. Δύο παραδείγματα εντολών με συναρτήσεις του πακέτου:

```
d7<-import('https://www.w3schools.com/xml/simple.xml')
```

Στο παραπάνω παράδειγμα γίνεται πρόσβαση σε αρχείο xml του οποίου τα περιεχόμενα ανακτώνται, αναλύονται και επιστρέφονται ως data.frame. Ένα παράδειγμα εξαγωγής δεδομένων:

```
export(d1,file = "test.xlsx")
```

Εδώ ένα data.frame (στη περίπτωση αυτή, το iris) εξάγεται σε αρχείο Excel το οποίο θα δημιουργηθεί στον τρέχοντα φάκελο εργασίας.

9.2 Γραφικές παραστάσεις

Οι πολυδιαφημισμένες δυνατότητες της R στον τομέα των γραφημάτων οφείλονται (και αυτές) στον τεράστιο αριθμό πακέτων επέκτασης που διατίθενται για τον σκοπό αυτό. Τα γραφήματα είναι σημαντικό εργαλείο για τον αναλυτή δεδομένων και έτσι υποστηρίζονται εκτενώς από την R και το οικοσύστημά της. Στην ενότητα αυτή γίνεται μια σύντομη εισαγωγή σε κάποια βασικά πακέτα γραφικών παραστάσεων, ξεκινώντας από τα προ-εγκατεστημένα ‘graphics΄,΄lattice’.

[443] βλ.§4.3.3 Οι τύποι ts και xts (χρονοσειρές).
[444] βλ. ενότητα Details στο help("import", package="rio").

9.2.1 Το υπόβαθρο: πακέτα ‘grDevices’ και ‘grid’

Αν και η συνήθης χρήση πακέτων όπως τα προαναφερθέντα δεν το απαιτεί, καλό είναι να γίνει μια αναφορά σε ένα πακέτο που χρησιμοποιούν, το πακέτο ‘grDevices’. Εδώ ορίζονται «συσκευές γραφικών» (graphics devices) στις οποίες θα απεικονίσουν τα γραφήματα τους τα άλλα πακέτα. Δεν πρόκειται για πραγματικές συσκευές αλλά για εικονικές συσκευές που παρέχουν το υπόβαθρο απεικόνισης γραφικών. Τέτοιες συσκευές μπορεί να είναι παράθυρα στο ΛΣ του υπολογιστή, αρχεία (pdf, ps, bmp, png, jpg, svg κ.α. - για μια πλήρη λίστα, βλ. help(Devices)). Άλλα πακέτα μπορούν να ορίσουν νέους τύπους συσκευών, πέραν όσων παρέχει το ‘grDevices’. Αυτό κάνει και το RStudio ώστε τα γραφήματα να εμφανίζονται στην καρτέλα Plots[445], με τα νέα να υπερκαλύπτουν τα προηγούμενα χωρίς να τα διαγράφουν και να επιτρέπεται (με κουμπιά της καρτέλας αυτής) η επιλογή του γραφήματος που εμφανίζεται, η δυνατότητα zoom, εξαγωγής σε αρχείο εικόνας, διαγραφή ενός ή όλων των γραφημάτων κ.α.

Ο έλεγχος των «συσκευών» απεικόνισης γραφικών γίνεται με συναρτήσεις που έχουν πρόθεμα dev του πακέτου ‘grDevices’. Καθώς είναι δυνατόν πολλές «συσκευές» να είναι ανοιχτές ταυτόχρονα, μια ορίζεται ως τρέχουσα, στην οποία θα καταλήγουν οι εντολές γραφικών. Τον αριθμό της επιστρέφει η συνάρτηση **dev.cur**

```
> dev.cur()
null device
          1
```

Αν μόλις έχει ξεκινήσει η R (ή το RStudio) οπότε δεν έχει χρειαστεί ακόμα να δημιουργηθεί κάποια συσκευή γραφικών, η συνάρτηση dev.cur θα επιστρέψει τη null device με αριθμό 1. Αυτό σημαίνει πως δεν υπάρχουν συσκευές. Ας δημιουργήσουμε τώρα μερικές:

```
> dev.new()                # μια νέα συσκευή (παράθυρο)
> dev.new()                # μια ακόμα νέα συσκευή
> dev.new()                # μια τρίτη νέα συσκευή
> dev.new()                # μια τέταρτη νέα συσκευή
> dev.list()
windows windows windows windows
      2       3       4       5
```

Εκτός της dev.new που λειτουργεί σε όλα τα ΛΣ, υπάρχουν και εξειδικευμένες συναρτήσεις για καθένα από αυτά. Για παράδειγμα, η **windows** λειτουργεί σε Microsoft Windows και δημιουργεί μια συσκευή (παράθυρο) με συγκεκριμένα χαρακτηριστικά, π.χ. windows(100,200) για παράθυρο με ελάχιστες διαστάσεις 100x200. Η συνάρτηση **dev.list** επιστρέφει τους αριθμούς των ενεργών συσκευών γραφικών. Στο RStudio η λίστα συσκευών θα περιέχει και συσκευές που αυτό δημιουργεί[446]. Εδώ οι συσκευές αυτές είναι τέσσερα παράθυρα που άνοιξαν στην οθόνη και τους ανατέθηκαν οι αριθμοί 2, 3, 4 και 5. Μερικές εντολές που μπορούν να εκτελεστούν μετά τις παραπάνω:

Δοκιμάστε:	**Σχόλιο**
`dev.set(5)`	Ορισμός τρέχουσας συσκευής, εδώ το παράθυρο #5.
`plot(0:10,10:20)`	Εκτέλεση εντολών γραφικών, θα γίνουν στην τρέχουσα συσκευή (#5) .
`msgWindow("maximize",which=5)`	Μεγιστοποίηση του παράθυρου #5.

Με τα παραπάνω, αρχικά θα δημιουργηθεί ένα γράφημα (με τη γνωστή συνάρτηση plot του ‘graphics’, βλ. παρακάτω). Αυτή θα «ζωγραφιστεί» στην τρέχουσα συσκευή που ορίστηκε με την **dev.set**, δηλαδή το παράθυρο #5. Το παράθυρο αυτό αμέσως μετά δέχεται εντολή («μήνυμα») να μεγιστοποιηθεί, με τη συνάρτηση **msgWindow**.

Δοκιμάστε:	**Σχόλιο**
`dev.set(5)`	Ορισμός τρέχουσας συσκευής, εδώ το παράθυρο #5.
`dev.copy(which = 4)`	Αντιγραφή από την τρέχουσα συσκευής στη συσκευή #4.
`dev.off(3)`	Διαγραφή της συσκευής #3.

Με τις εντολές αυτές, η συνάρτηση **dev.copy** αντιγράφει στο #4 τα περιεχόμενα του #5 (που επιλέχτηκε ξανά ως τρέχον για την περίπτωση που αυτό είχε αλλάξει). Επίσης η αχρησιμοποίητη έως τώρα συσκευή

[445] βλ.§1.4.2.3 Οι καρτέλες Plots και Viewer.

[446] Αυτό μπορεί να αποφευχθεί με παραλλαγή της εντολής: dev.new(noRStudioGD = F).

(παράθυρο) #3 θα κλείσει με τη συνάρτηση **dev.off**. Για να κλείσουν όλες οι συσκευές μπορεί να εκτελεστεί μία εντολή όπως:

```
while(dev.off()>1){}
```

Το παραπάνω κλείνει την τρέχουσα συσκευή μέχρι η νέα τρέχουσα (που επιστρέφει το dev.off) να είναι η #1, η ειδική null συσκευή που δείχνει ότι καμία συσκευή δεν είναι διαθέσιμη. Τέτοιες εντολές επιτρέπουν τον έλεγχο της συσκευής στην οποία θα επιδράσουν οι όποιες εντολές γραφικών εκτελεστούν, άρα μπορούν να δημιουργηθούν και υποτυπώδη animation όπως:

```
dev.new()
d <- dev.cur()
for (i in c(1:20, 20:1))
{
  x <- seq(-i, i, .2)
  y <- 3 * x * cos(x)
  dev.set(d)
  barplot(y, col = "blue")
  Sys.sleep(.1)      # αναμονή .1 sec
}
dev.off()
```

Στον παραπάνω κώδικα η ψευδαίσθηση της κίνησης προκαλείται από την επιλογή (με την εντολή dev.set(d)) της ίδιας συσκευής (παραθύρου) για την εμφάνιση μιας νέας παραλλαγής του γραφήματος (που εδώ δημιουργεί η barplot του ‘graphics’, βλ. παρακάτω), αντικαθιστώντας το προηγούμενο.

Όμως πιο σημαντικό είναι ότι εκτός από παράθυρα, οι συσκευές γραφικών μπορεί να αφορούν έξοδο γραφικών με άλλους τρόπους. Για παράδειγμα οι συναρτήσεις **pdf**, **bmp** κλπ. δημιουργούν συσκευές γραφικών για έξοδο σε αρχεία (εδώ Adobe Acrobat και Windows bitmap αντίστοιχα). Οι παρακάτω εντολές δημιουργούν ένα plot σε νέο παράθυρο (το οποίο μετά κλείνουν):

```
dev.new()
plot(0:10,10:20)
dev.off()
```

Οι επόμενες, δημιουργούν το ίδιο plot, αλλά σε συσκευή που οδηγεί σε ένα αρχείο pdf (με όνομα “test.pdf”):

```
pdf(file = "test.pdf")
plot(0:10,10:20)
dev.off()
```

Το ίδιο γίνεται παρακάτω, σε ένα αρχείο bitmap (με όνομα “test.bmp” και την εικόνα να έχει τις συγκεκριμένες διαστάσεις):

```
bmp(file = "test.bmp", width=640, height=480)
plot(0:10,10:20)
dev.off()
```

Είτε είναι παράθυρα, είτε άλλης μορφής εικονικές συσκευές γραφικών, η εκτέλεση των εντολών που τις χρησιμοποιούν (όπως εδώ η plot) θα έχει το ίδιο αποτέλεσμα. Εκτός από τις συσκευές, το ‘grDevices’ παρέχει και πολλές συναρτήσεις που βοηθούν τη δημιουργία γραφικών. Εδώ ορίζονται ονόματα για τα χρώματα (βλ. συνάρτηση **colors**), παρέχονται λειτουργίες χειρισμού raster, μετατροπής εκφράσεων σε κείμενο προς εμφάνιση (βλ. help(plotmath)) κ.α. Άλλα παραδείγματα σχετικά με το πακέτο ‘grDevices’ αναφέρονται στο [106].

Το δεύτερο προ-εγκατεστημένο πακέτο που παρέχει υπόβαθρο για δημιουργία γραφικών είναι το ‘grid’. Στο πακέτο αυτό ορίζονται χαμηλού-επιπέδου αντικείμενα γραφικών, τα επονομαζόμενα “grob” (graphics object) που αντιστοιχούν σε στοιχειώδη γραφικά στοιχεία. Ως αντικείμενα, ένα grob μπορεί να αποθηκευτεί σε μεταβλητή και να γίνει χειρισμός του με τον συνήθη τρόπο, όπως ένα οποιοδήποτε αντικείμενο της R (βλ. και [60]). Τα γραφικά στοιχεία που προσφέρει είναι γραμμές, κύκλοι, καμπύλες Bezier, παραλληλόγραμμα, άξονες, κείμενο κ.α. Για αυτά, το πακέτο υποστηρίζει σελίδες γραφικών ενώ κάθε αντικείμενο μπορεί να ανήκει σε μια δενδροειδή ιεραρχία και να ανατίθεται σε ένα viewport. Τα αντικείμενα “grob” δημιουργούνται με συναρτήσεις όπως οι **rectGrob**, **polygonGrob**, **circleGrob**, **bezierGrob**, **textGrob** κλπ. αλλά δεν απεικονίζονται έως να κληθεί με αυτά η συνάρτηση **grid.draw**. Για άμεση απεικόνιση μπορούν να χρησιμοποιηθούν οι αντίστοιχες

συναρτήσεις **grid.rect**, **grid.polygon**. **grid.circle**, **grid.bezier**, **grid.text** κλπ. Ένα εισαγωγικό παράδειγμα[447] παρακάτω, εφαρμόζει και τις δύο προσεγγίσεις:

```
grid.newpage()

g1 <- polygonGrob( name = "g1",
    x = c(0.1, 0.2, 0.3),
    y = c(0.1, 0, 0.3),
    gp = gpar(col = "black", fill = "green", lty = 4))

g2 <- rectGrob( name = "g2",
  x = 0.5, y = 0.5, width = 0.5, height = 0.3,
  gp = gpar(fill = "yellow"))

g3 <- circleGrob( name = "g3", x = 0.7, y = 0.7,
  r = seq(0.1, 0.3, 0.01),  gp = gpar(col = "red"))

grid.draw(g1)
grid.draw(g3)
grid.draw(g2)

grid.circle(x = 1, y = 0, r = 0.2, gp = gpar(fill = "blue"))
grid.circle(x = 0, y = 1, r = 0.2, gp = gpar(fill = "blue"))
grid.lines(c(1,0), c(0,1), gp = gpar(lty = "dotdash", lwd = 5))
grid.text("Γειά σου κόσμε!", x=.1, y=.3, gp = gpar(col = "Red"))
grid.bezier(x = c(0.4, 0.3, 0.8, 0.4), y = c(1, 0.4, 0.4, 0))
```

Τα αντικείμενα στις μεταβλητές g1, g2 και g3 είναι κλάσης "grob" και εμφανίζονται όταν κληθεί η grid.draw με αυτά. Οι επόμενες εντολές grid.circle κλπ. έχουν άμεσο αποτέλεσμα. Οι γραφικές παράμετροι των αντικειμένων (χρώμα, γραμματοσειρά, στυλ γραμμών κλπ.) που χρησιμοποιούνται για την απεικόνιση ορίζονται στην παράμετρο gp όπως επιστρέφονται από τη συνάρτηση gpar. Εδώ χρησιμοποιούνται ίδιας μορφής παράμετροι με αυτές της συνάρτησης par (βλ. επόμενη §9.2.2 Πακέτο 'graphics').

Το 'grid' αποτελεί βάση για άλλα πακέτα γραφικών, υψηλότερου επιπέδου όπως το δημοφιλές ggplot2[448]. Λεπτομερής περιγραφή των λειτουργιών του πακέτου 'grid' υπάρχει στην τεκμηρίωση (βλ. vignette(package='grid')) και ενδιαφέροντα παραδείγματα στα [60] και [107].

9.2.2 Πακέτο 'graphics'

Το βασικό πακέτο γραφημάτων που έρχεται με κάθε εγκατάσταση της R είναι το 'graphics. Παρέχει συναρτήσεις υψηλότερου επιπέδου λειτουργικότητας όπως η **plot**[449] **(**που έχει ήδη χρησιμοποιηθεί σε κάποια παραδείγματα σε άλλες ενότητες του βιβλίου). Οι συναρτήσεις του 'graphics' συνήθως καλούνται για να δημιουργήσουν πλήρη γραφήματα. Για να δημιουργηθεί ένα γράφημα πρέπει πρώτα να προετοιμαστεί ένα υπόβαθρο για αυτό. Ευτυχώς, αυτό γίνεται αυτόματα και εσωτερικά από τις ίδιες τις συναρτήσεις γραφημάτων (όπως η plot). Όμως αξίζει να δούμε λίγο τον μηχανισμό, χρησιμοποιώντας ενδεικτικά κάποιες συναρτήσεις του πακέτου:

```
par(bg="sienna")         # Ορίζει χρώμα υποβάθρου.
frame()                  # ή plot.new() για νέο γράφημα.
grid(nx = 20, ny=5)      # Πλέγμα 20 γραμμών στο x, 5 στο y.
plot.window(c(-10,10),   # ορισμός συστήματος συντεταγμένων,
            c(0,10))     # αν δεν γίνει τα όρια είναι 0 έως 1.
axis(side=4)             # Ζωγραφίζει έναν άξονα δεξιά.
lines(c(-5, 0, 4, 1),    # Ζωγραφίζει γραμμές που ενώνουν τις
      c(10, 0, 1, 5),    # θέσεις (-5,10),(0,0),(4,1) και
```

[447] Το παράδειγμα απαιτεί να συνδεθεί πρώτα το πακέτο 'grid', με εντολή όπως η library(grid).
[448] βλ.§9.2.4 Πακέτο 'ggplot2'.
[449] Σε κάποιες εκδόσεις της R η plot έχει μεταφερθεί στο πακέτο 'base'.

```
       col='blue')        # (1,5), χρώματος μπλε.
points(c(5,8),c(2,4),    # Ζωγραφίζει κίτρινα σημεία στις
       type="b",          # θέσεις (5,2) και (8,4) ενωμένα με
       col="yellow")      # γραμμή (τύπου "b").
text(7,7,"Γειά κόσμε!", # Ζωγραφίζει κείμενο στη θέση (7,7)
       col="white")       # χρώματος λευκού.
rect(-9,9,-7,5)           # Ζωγραφίζει παραλληλόγραμμο.
par(bg="white")           # Επαναφέρει το λευκό υπόβαθρο για
                          # τα γραφήματα που ακολουθούν.
```

Η πρώτη εντολή στο παράδειγμα χρησιμοποιεί τη συνάρτηση **par**. Με τη συνάρτηση αυτή ορίζονται ρυθμίσεις που θα χρησιμοποιούν οι συναρτήσεις του ‘graphics’ για τη μορφή (εμφάνιση) των γραφικών (χρώματα, γραμματοσειρές, κλπ.). Κάποιες από αυτές τις παραμέτρους επιτρέπεται να οριστούν και από τις άλλες συναρτήσεις του πακέτου (καθώς καλούν εσωτερικά την par μεταφέροντας σε αυτή σχετικές παραμέτρους). Στο παράδειγμα, η παράμετρος bg ορίζει το χρώμα του υποβάθρου (background color). Άλλες συνήθεις παράμετροι είναι οι col (για το χρώμα που θα χρησιμοποιηθεί όταν ζωγραφίζεται ένα σχήμα), pty (περιοχή απεικόνισης, τετράγωνη ή μέγιστη), mar (για τα περιθώρια ή margin που θα υπάρχουν στον χώρο απεικόνισης), pch (χαρακτήρες ή κωδικοί συμβόλων για τα σημεία), lwd (πάχος γραμμών), new (ρυθμίζει αν τα επόμενα γραφικά θα υπερκαλύπτουν τα υπάρχοντα), ask (σταματά τη δημιουργία νέων γραφημάτων μέχρι να πάρει εντολή από τον χρήστη) κ.α. Παρατηρήστε ότι η εντολή με τη συνάρτηση **lines** που ακολουθεί παρακάτω (και ζωγραφίζει γραμμές) χρησιμοποίησε την παράμετρο col (της par) για ορισμό του χρώματος της γραμμής. Για περισσότερες παραμέτρους του par, βλ. help("graphical parameters") ή help(par).

Συναρτήσεις όπως οι **points**, **lines**, **rect** κ.α. επιτρέπουν αρκετά χαμηλού επιπέδου χειρισμό των γραφικών του πακέτου ‘graphics’. Για παράδειγμα, ο παρακάτω κώδικας δημιουργεί έναν χώρο όπου ο χρήστης μπορεί να ζωγραφίσει ένα spline, επιλέγοντας με το ποντίκι 5 σημεία που θα το ορίζουν. Η συνάρτηση **locator** κάνει λήψη συντεταγμένων από το ποντίκι και τις επιστρέφει ως αντικείμενο list[450]:

```
old_par <- par()         # Οι τρέχουσες ρυθμίσεις γραφικών.
par(pty="s",             # Νέες ρυθμίσεις: Τετράγωνος
    mar=c(0,0,0,0),      # χώρος ζωγραφικής, περιθώρια
    bg="yellow")         # και χρώμα υποβάθρου.
frame()                  # Νέο γράφημα.
plot.window(c(0,9),      # ορισμός εύρους συντεταγμένων,
            c(0,9))      # αν δεν γίνει τα όρια είναι 0 έως 1.
points(                  # Μία σειρά από τελείες (pch=".")
  expand.grid(0:9,0:9),# στις θέσεις (0,0),(0,1),...(9,8),
  pch=".")               # (9,9) για διακοσμητικούς λόγους.
p<-locator(5)            # Λήψη 5 σημείων με κλικ στο γράφημα.
points(p$x,p$y,          # Εμφάνιση των σημείων που δόθηκαν
       col="red",        # εδώ ως κόκκινοι σταυροί (+).
       pch=3)
lines(                   # Υπολογισμός σημείων καμπύλης spline,
  spline(p$x,p$y,        # βάσει των σημείων που δόθηκαν και
    method="natural")) # εμφάνιση τους ενωμένα με τη lines.
par(old_par)             # επαναφορά των γραφικών ρυθμίσεων.
```

Ας προχωρήσουμε τώρα σε υψηλότερου επιπέδου λειτουργίες γραφημάτων, δηλαδή αυτές που συνήθως χρησιμοποιούνται. Η πλέον συνήθης τέτοια συνάρτηση είναι η plot. Η plot είναι generic συνάρτηση, άρα το αποτέλεσμά της ποικίλει ανάλογα με τον τύπο του αντικειμένου που επεξεργάζεται. Η προεπιλεγμένη (default) συνάρτηση χειρισμού της plot (plot.default)[451], δέχεται συντεταγμένες σημείων και εμφανίζει τα σημεία σε γράφημα διασποράς (scatter plot). Αν π.χ. δοθεί ένα διάνυσμα αριθμών, θα εμφανίσει σημεία στις θέσεις (n,x) όπου n ο δείκτης ενός στοιχείου, x η τιμή του. Αν δοθούν δύο διανύσματα αριθμών στις πρώτες δύο παραμέτρους της θα εμφανίσει σημεία στις θέσεις (x,y), όπου x η τιμή ενός στοιχείου στο πρώτο διάνυσμα και

[450] βλ.§4.2.1 Ο τύπος list (λίστα).

[451] βλ. help(plot). Generic είναι και άλλες συναρτήσεις γραφικών παραστάσεων. Για τον ρόλο των generic συναρτήσεων, βλ. Κεφάλαιο 6 και ειδικότερα §6.3 Κλάσεις S3.

y η τιμή του αντίστοιχου στοιχείου στο δεύτερο. Συντεταγμένες σημείων μπορούν να δοθούν και με διάφορους άλλους τρόπους. Ακολουθεί ένα παράδειγμα:

Δοκιμάστε:	**Σχόλιο**
`x <- -10:10`	Δημιουργία x στο διάστημα από -10 έως 10 με βήμα 0.1.
`y <- x^2`	Υπολογισμός y (εδώ είναι το τετράγωνο των αντίστοιχων x).
`plot(y)`	Εμφάνιση σημείων στις θέσεις (δείκτης σημείου, τιμή στο y).
`plot(x,y,main="Γράφημα", pch=3)`	Με '+' στις θέσεις (τιμή στο x, τιμή στο y) και τίτλος.
`plot(x,y,type='l',col="blue",lwd=5)`	Με γραμμή που ενώνει τα σημεία (x,y), μπλε, πάχους 5.
`grid(nx = 20, ny=20)`	Πλέγμα 20x20 προστίθεται στο προηγούμενο γράφημα.

Οι παράμετροι main και type ανήκουν στην plot. Η type ορίζει τον τύπο του γραφήματος. Εδώ, η τιμή 'l' σημαίνει γραμμή, το προεπιλεγμένο 'p' σημεία, το 'h' ιστόγραμμα κλπ. Υπάρχουν και άλλοι διαθέσιμοι τύποι που περιγράφονται στην τεκμηρίωση της συνάρτησης[452]. Η παράμετρος main δέχεται το κείμενο που θα εμφανιστεί ως τίτλος στο γράφημα. Άλλες παράμετροι της plot που χρησιμοποιούνται συχνά είναι η sub για το κείμενο στον δευτερεύοντα τίτλο (υπότιτλο), οι xlab και ylab για ορισμό του κειμένου στους άξονες x και y, οι xlim και ylim για τα όρια των αξόνων. Όπου υπάρχει ανάγκη εμφάνισης μαθηματικών εκφράσεων και συμβόλων (σε τίτλους, λεζάντες κλπ.), το 'graphics' αξιοποιεί τη δυνατότητα μετατροπής αντικειμένων expression σε τέτοια (αυτή παρέχεται από το 'grDevices', για τις συμβάσεις που εφαρμόζονται βλ. help(plotmath)). Έτσι,. αν π.χ. η παράμετρος main οριστεί ως main=expression(4*chi^3) θα εμφανιστεί στον κύριο τίτλο του γραφήματος το κείμενο $4\chi^3$. Η plot και άλλες συναρτήσεις γραφημάτων του 'graphics' επιτρέπουν πρόσθετες παραμέτρους που ελέγχουν την εμφάνιση του γραφήματος, όπως π.χ. οι pch, col και lwd στο παράδειγμα παραπάνω και τις περνούν στη συνάρτηση par την οποία καλούν εσωτερικά.

Ως generic, η plot μπορεί να υποστηρίζεται από διάφορους τύπους αντικειμένων. Στο επόμενο παράδειγμα καλείται η plot για ένα αντικείμενο function. Επιπρόσθετα, ως τίτλος (main) του plot επιλέχτηκε το σώμα της συνάρτησης, αφού η συγκεκριμένη συνάρτηση έχει νόημα να γραφεί με μαθηματικά σύμβολα):

Δοκιμάστε:	**Σχόλιο**
`f<-function(x) 2*sqrt(x)`	Μια συνάρτηση.
`plot(f, from=0, to=50, main=body(f))`	Η γραφική της παράσταση στην περιοχή από 0 έως 5.

Ως προεπιλογή, κάθε plot ξεκινά ένα νέο γράφημα (καλώντας τη συνάρτηση plot.new) και ορίζει το εύρος των συντεταγμένων σε αυτό (με την plot.window). Αφού ζωγραφιστεί το γράφημα, μπορούν να προστεθούν και άλλα γραφικά στοιχεία σε αυτό (όπως έγινε παραπάνω με τη συνάρτηση grid). Αυτός είναι και ένας απλός τρόπος εμφάνισης πολλών δεδομένων στο ίδιο plot:

Δοκιμάστε:	**Σχόλιο**
`x<-seq(-10,10,.1)`	Δημιουργία x στο διάστημα από -10 έως 10 με βήμα 0.1.
`y1<-3*x*cos(x)`	Υπολογισμός τιμών y 1ης σειράς, y1.
`y2<-10*cos(x)`	Υπολογισμός τιμών y 2ης σειράς, y2.
`y3<-3*sin(x)`	Υπολογισμός τιμών y 3ης σειράς, y3.
`plot( x,y1,type="l",ylab="f(x)" )`	Γράφημα (γραμμή) για τα x,y1 (και ετικέτα άξονα y).
`lines(x,y2,type="l",col="blue",lty=2)`	Προσθήκη γραμμής για τα x,y2 (μπλε, διακεκομμένη).
`lines(x,y3,type="l",col="green")`	Προσθήκη γραμμής για τα x,y3 (πράσινη).

Προφανώς μπορούν να προστεθούν και άλλα είδη γραφικών στοιχείων στο γράφημα. Π.χ. με τη συνάρτηση text μπορεί να προστεθεί κείμενο ενώ με τη συνάρτηση axis ένας άξονας. Το παρακάτω συνεχίζει από το προηγούμενο:

Δοκιμάστε:	**Σχόλιο**
`text(-7,20,"3xcos(x)")`	Προσθήκη κειμένου στη θέση (-7,20).
`text(-1,-10,"-10cos(x)",col="blue")`	Προσθήκη μπλε κειμένου στη θέση (-1,-10).
`text(3,5,"3sin(x)",col="green")`	Προσθήκη πράσινου κειμένου στη θέση (3,5).
`axis(3,-8:8)`	Άξονας στο πάνω μέρος, ticks στις θέσεις -8, -7,…,6 ,7 , 8.

[452] βλ. help(plot.default).

Η συνάρτηση **legend** προσθέτει υπόμνημα (λεζάντα) στο γράφημα. Εδώ ορίζεται το κείμενο και τα χαρακτηριστικά των στοιχείων που θα εμφανιστούν στη λεζάντα, ενώ η παράμετρος cex (της par) ρυθμίζει τον βαθμό μεγέθυνσης των συμβόλων και του κειμένου που θα εμφανιστεί:

```
legend("topright",
        c("3xcosx","-10cosx","3sinx"),
        lty=c(1,2,1),
        col=c("black","blue","green"),
        cex=0.8)
```

Ακολουθούν κάποια ενδεικτικά παραδείγματα δημιουργίας γραφημάτων με το πακέτο ΄graphics’:

(α) Γράφημα διασποράς XY (scatter plot).

Δοκιμάστε:	**Σχόλιο**
`x<-c(10, 3, 13, 30, 23, 11, 40)`	Διάνυσμα με τιμές x για 6 σημεία.
`y<-c(20, 3, -3, 30,  2, 12,  9)`	Διάνυσμα με τιμές y των σημείων.
`s<-letters[1:6]`	Διάνυσμα με 6 γράμματα, τα “a”,”b”,”c”,”d”,”e” και “f”.
`h<-hcl.colors(6)`	Διάνυσμα με τιμές για 6 χρώματα.
`plot(x,y,pch=s,col=hcl.colors(6))`	Διασπορά, με διαφορετικά γράμματα και χρώματα ανά σημείο.
`identify(x,y)`	Εντοπισμός σημείων με το ποντίκι, βλ. παρακάτω.

Σε διαδραστικές συσκευές (όπως τα παράθυρα), η συνάρτηση **identify** επιτρέπει να εντοπιστούν ποια σημεία εμφανίζονται, κάνοντας κλικ σε αυτά με το ποντίκι. Στα Windows, η διαδικασία σταματά πιέζοντας το πλήκτρο Esc.

(β) Γράφημα διασποράς με formula. Η σχετική συνάρτηση χειρισμού της plot (plot.formula) χρησιμοποιεί το formula για προσδιορισμό των μεταβλητών που θα εμφανιστούν, με την πηγή τους να ορίζεται στην παράμετρο data (εδώ το data.frame iris[453]):

```
plot(Petal.Length~Petal.Width, data=iris)
```

(γ) Γραφική παράσταση καμπύλης. Αντίθετα με την plot, η **curve** μπορεί να δεχτεί απευθείας την έκφραση που θα απεικονίσει (ως αντικείμενο expression, call ή και function που χρησιμοποιεί x):

Δοκιμάστε:	**Σχόλιο**
`curve(2*sqrt(x),0,10,n=6000)`	Εμφάνιση καμπύλης, υπολογισμένη για 6000 x από 0 έως 10.
`curve((2*sin(x))^2,0,10,add = T)`	Προσθέτει σε υπάρχον γράφημα τη συγκεκριμένη καμπύλη.

(δ) Γράφημα στηλών ή ράβδων (bar plot) με τη συνάρτηση **barplot**.

Δοκιμάστε:	**Σχόλιο**
`d<-c(4, 3, 12, 6)`	Διάνυσμα με 4 τιμές (ύψη ή height των στηλών/ραβδων).
`n<-c("CH","SP","IT","DE")`	Διάνυσμα με ονόματα για τις τιμές.
`barplot(d)`	Γράφημα κάθετων στηλών.
`barplot(d,horiz=TRUE)`	Γράφημα οριζοντίων ράβδων.
`barplot(d,names=n)`	Γράφημα κάθετων στηλών με ονόματα.
`barplot(d,col=gray.colors(2))`	Με 2 διαφορετικά γκρι να εναλλάσσονται ως χρώμα στήλης.
`dc<-c("red","red","blue","red")`	Ορισμός διανύσματος με ονόματα χρωμάτων.
`barplot(d,col=dc)`	Γράφημα κάθετων στηλών με τα συγκεκριμένα χρώματα.
`bp<-barplot(d,ylim=c(0,20))`	Με ορισμό του εύρους του άξονα y και ανάθεση σε bp.
`text(x = bp[,1],y=d+1,labels = n)`	Προσθήκη λεζάντας ακριβώς πάνω από κάθε στήλη.

Η τελευταία εντολή προσθήκης κειμένου χρησιμοποιεί την επιστρεφόμενη από τη barplot θέση του μέσου κάθε στήλης (ή ράβδου) που δημιουργήθηκε, κάτι που βοηθά την πρόσθεση άλλων γραφικών στοιχείων πάνω σε αυτές. Επίσης, στη συνάρτηση barplot τα δεδομένα μπορούν να είναι σε πίνακα. Στο παράδειγμα που ακολουθεί δημιουργούνται κατάλληλοι πίνακες από διανύσματα (με τη συνάρτηση cbind) και τροφοδοτούνται στην barplot. Το αποτέλεσμα εδώ είναι δύο διαγράμματα στο ίδιο plot (ένα για κάθε στήλη στα δεδομένα).

```
d1<-c(10,13,20,23,11,40)          # 1η σειρά δεδομένων.
```

[453] βλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων.

```
d2<-c(22,13,14,15,25,22)           # 2η σειρά δεδομένων.
n<-c("Αρτα","Πάτρα","Καλαμάτα",    # Ονόματα για τα δεδομένα
     "Βόλος","Θεσ/νικη",
     "Ηράκλειο")
dc<-c("blue","blue","red",         # Χρώματα για τα δεδομένα
      "blue","blue","blue")
barplot(cbind(d1,d2),
        names=cbind(n,n),
        col=cbind(dc,dc),
        beside=TRUE)
```

(ε) Ιστόγραμμα (histogram) με τη συνάρτηση **hist**. Το ιστόγραμμα απεικονίζει τη συχνότητα ανά περιοχή τιμών, πόσα δηλαδή από τα δεδομένα βρίσκονται σε κάθε περιοχή τιμών (bin).

```
d<-c(170,154,171,160,169,152,163,165,
     166,150,172,165,175,182,166,169)
hist(d, main = "Συχνότητα")     # Εμφάνιση του γραφήματος
hist(d,breaks=seq(150,200,10))  # σε bins 150,160,170,…,200
```

Η συνάρτηση hist επιστρέφει μία λίστα που αν αποθηκευτεί σε μεταβλητή μπορεί να εμφανιστεί μέσω plot. Στη λίστα καταγράφεται η συχνότητα ανά bin, τα όρια των bin (breaks) κ.α.

```
print(hist(d))
```

Υπάρχουν διάφοροι τρόποι να υπολογιστούν τα όρια των περιοχών τιμών που θα χρησιμοποιήσει η hist. Για περισσότερα βλ. help(hist). Τόσο η hist όσο και οι υπόλοιπες συναρτήσεις γραφημάτων που προσφέρει το πακέτο, έχουν μεγάλες δυνατότητες προσαρμογής μέσω των παραμέτρων τους. Επίσης πολλές από αυτές υποστηρίζουν παραμέτρους όπως η add (βλ. παράδειγμα της curve παραπάνω) ώστε να μπορούν να συνδυαστούν πολλαπλά γραφήματα στην ίδια απεικόνιση. Το πακέτο επίσης περιέχει τις συναρτήσεις **layout** και **split.screen** για ορισμό περιοχών στη συσκευή απεικόνισης (παράθυρο κλπ.) στις οποίες θα εμφανίζονται διαφορετικά γραφήματα. Ακολουθεί ένα παράδειγμα της split.screen που εδώ χωρίζει την επιφάνεια σε 4 «οθόνες» και εμφανίζει ένα διαφορετικό γράφημα σε καθεμία:

```
x<-c(150,154,171,160,169,152,163,165)    # μερικά δεδομένα
y<-c( 66, 50, 72, 65, 75, 82, 66, 69)    # κι άλλα δεδομένα

split.screen(c(2,2))       # χωρισμός σε 4 (2x2) οθόνες

screen(1)                  # Εμφάνιση επόμενου στην «οθόνη» 1
hist(x,main="Οθόνη 1")

screen(2)                  # Εμφάνιση επόμενου στην «οθόνη» 2
barplot(y,col=hcl.colors(6),main="Οθόνη 2")

screen(3)                  # Εμφάνιση επόμενου στην «οθόνη» 3
plot(x,y,pch=3,col=hcl.colors(6),main="Οθόνη 3")

screen(4)                  # Εμφάνιση επόμενου στην «οθόνη» 4
m<-lm(y~x)
plot(x,m$fitted.values,ylim=c(min(y),max(y)),main="Οθόνη 4")

close.screen(all = TRUE)   # Τέλος κατάστασης split.screen
```

Το επόμενο είναι ένα παράδειγμα χρήσης layout, όπου απεικονίζονται τρία γραφήματα, εδώ τοποθετημένα το ένα κάτω από το άλλο. Το matrix (εδώ 3 γραμμές x 1 στήλη) έχει τη σειρά με την οποία θα εμφανίζονται τα γραφήματα στις περιοχές του layout. Η αναλογία ύψους για την καθεμία περιοχή ορίζεται στην παράμετρο height. Η συνάρτηση **layout.show** δείχνει τη θέση των περιοχών αυτών στη συσκευή. Στα επόμενα βήματα ζωγραφίζονται τα γραφήματα με δεδομένα από το iris[454]: ορίζονται περιθώρια (με τη συνάρτηση par), δημιουργείται ένα θηκόγραμμα (συνάρτηση **boxplot**) στην περιοχή 1, ένα γράφημα διασποράς (με plot) στην

[454] βλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων.

περιοχή 2, και ένα ιστόγραμμα (συνάρτηση hist) στην περιοχή 3. Στο τελευταίο γράφημα προστίθεται η πυκνότητα (προσεγγισμένη από τα δεδομένα με τη συνάρτηση **density** και απεικονισμένη μέσω της lines).

```
layout(mat=matrix(nrow=3,ncol=1,c(1,2,3)),  height=c(1,4,2))
layout.show(n = 3)
par(mar=c(1, 4, 1, 1))
boxplot(iris$Petal.Length, horizontal=TRUE,
        outline=TRUE, frame=FALSE, col = "gray")
plot(iris$Petal.Length,iris$Petal.Width,
     ylab="Πλάτος Πέταλου")
hist(iris$Petal.Length, col = "grey", freq = F,
     main="Μήκοςς Πέταλου")
lines(density(iris$Petal.Length),lwd=3)
```

Η υποστήριξη στην τεκμηρίωση για κάθε συνάρτηση του πακέτου ‘graphics’ είναι εκτενής και περιέχει παραδείγματα που αναδεικνύουν πολλές από τις δυνατότητες τους. Αξίζει να εκτελέσετε τα παραδείγματα με τη συνάρτηση example, π.χ. με την εντολή example(pairs) για τη συνάρτηση **pairs** (απεικόνιση ανά ζεύγος μεταβλητών ενός matrix ή data.frame) ή τα αντίστοιχα των **pie** (γράφημα πίτας), boxplot (θηκόγραμμα ή box-and-whisker plot), **image** (χρωματική απεικόνιση τιμών σε ένα matrix), **contour** (εμφάνιση ορίων) ή **persp** (απεικόνιση, με προοπτική, επιφάνειας πάνω στο επίπεδο x–y), **matplot** (εμφάνιση δεδομένων στις στήλες ενός πίνακα με αυτές ενός άλλου) και άλλων συναρτήσεων του πακέτου.

9.2.3 Πακέτο ‘lattice’

Στην ενότητα αυτή γίνεται μια σύντομη παρουσίαση του ‘lattice’ [21], ενός άλλου προ-εγκατεστημένου πακέτου γραφημάτων. Τα δεδομένα στα παραδείγματα[455] που ακολουθούν προέρχονται από το data.frame iris[456]. Η συνάρτηση **xyplot** δημιουργεί γραφήματα διασποράς:

```
p <- xyplot (Petal.Width ~ Petal.Length,
             data = iris, groups = Species,
             auto.key = T, xlab = "Μήκος",
             ylab = "Πλάτος")
```

Οι συναρτήσεις γραφημάτων του lattice είναι generic. Ο συνήθης τύπος που χρησιμοποιούν είναι formula[457] οπότε το εμπλεκόμενο αντικείμενο έχει τη μορφή y ~ x | b1 * b2, όπου y, x οι μεταβλητές που θα απεικονιστούν (από το αντικείμενο που ορίζεται ως data), ενώ b1, b2 είναι μεταβλητές που ορίζουν συνθήκες. Η παραπάνω εντολή ζητά να απεικονιστεί γράφημα διασποράς για σημεία με y το Petal.Width του iris, x το αντίστοιχο Petal.Length, ομαδοποιημένα χρωματικά ως προς τη μεταβλητή Species (παράμετρος groups), με αυτόματη λεζάντα (παράμετρος auto.key) και τις δοθείσες ετικέτες στον άξονα x (παράμετρος xlab) και y (παράμετρος ylab). Οι συναρτήσεις επιστρέφουν αντικείμενα, οπότε τα γραφήματα μπορούν να ανατεθούν σε μεταβλητές, όπως έγινε παραπάνω στη μεταβλητή p. Τα αντικείμενα είναι τύπου list που κληρονομούν την κλάση "trellis"[458] η οποία ορίζεται στο πακέτο. Έτσι μπορεί να γίνει πρόσβαση και επεξεργασία των στοιχείων τους (άρα και του γραφήματος που θα προκύψει) ενώ μπορεί να γίνει εμφάνιση του γραφήματος σε ύστερο χρόνο με χρήση των συναρτήσεων plot και print, καθώς η κλάση "trellis" παρέχει μεθόδους χειρισμού των συναρτήσεων αυτών.

Στην επόμενη παραλλαγή του παραδείγματος το Species χρησιμοποιείται ως συνθήκη για την επιλογή του χώρου απεικόνισης των δεδομένων (άρα να είναι διαφορετικό για κάθε είδος iris στα δεδομένα), αφού το γράφημα έχει χωριστεί σε τρία μέρη (με την παράμετρο layout που υποστηρίζουν οι συναρτήσεις του πακέτου):

```
xyplot (Petal.Width ~ Petal.Length | Species,
        data = iris,
        layout=c(3,1),
        groups = Species)
```

Η συνάρτηση **cloud** δημιουργεί γραφήματα διασποράς τριών αξόνων:

```
cloud(Petal.Width ~ Petal.Length * Sepal.Length,
```

[455] Τα παραδείγματα απαιτούν να συνδεθεί πρώτα το πακέτο ‘lattice’, με εντολή όπως η library(lattice).
[456] βλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων.
[457] βλ.§4.3.4 Αντικείμενα τύπου language (expression, call, name και formula).
[458] Η "trellis" είναι κλάση S3, βλ.§6.3 Κλάσεις S3.

```
        data = iris, auto.key = T, group = Species)
```
Η επόμενη εντολή (συνάρτηση **histogram**) δημιουργεί ιστόγραμμα της μεταβλητής Petal.Width του iris:
```
histogram(~ Petal.Width, data = iris)
```
Ενώ παρακάτω δημιουργείται το διάγραμμα πυκνότητας για τα ίδια δεδομένα:
```
x<-densityplot(~ Petal.Width, data = iris, plot.points = FALSE
```
Για περισσότερα παραπέμπουμε στην εκτενή τεκμηρίωση του πακέτου (help("lattice")) καθώς και στις πολλές διαδικτυακές πηγές που παρουσιάζουν τις δυνατότητες του πακέτου αυτού.

9.2.4 Πακέτο 'ggplot2'

Το πακέτο ΄ggplot2' [108] είναι μια ιδιαίτερα διαδεδομένη συλλογή συναρτήσεων γραφημάτων. Βασισμένο στο 'grid'[459], το 'ggplot2' αποτελεί με τη σειρά του βάση για πολλά άλλα πακέτα επέκτασης της R. Το πακέτο μπορεί να εγκατασταθεί μόνο του ή ως μέρος του 'tidyverse' [57] το οποίο το συμπεριλαμβάνει[460]. Το πακέτο εφαρμόζει μια «γραμματική γραφικών» (grammar of graphics) για την προβολή των δεδομένων σε οπτικά αντικείμενα. Ας δημιουργήσουμε ένα απλό γράφημα διασποράς με τα δεδομένα από το data.frame iris[461], όπως έγινε και για τα παραδείγματα στα προηγούμενα πακέτα γραφημάτων. Για γρήγορη δημιουργία γραφημάτων το πακέτο παρέχει τη συνάρτηση **qplot** (ή **quickplot**). Εδώ, ένα γράφημα διασποράς:
```
qplot(Petal.Width, Petal.Length,data=iris,colour = Species)
```
Όμως η βασική συνάρτηση γραφημάτων στο 'ggplot2' είναι η συνάρτηση **ggplot**. Ακολουθεί ένα παράδειγμα όπου, για να φανεί καλύτερα ο μηχανισμός, η δημιουργία του γραφήματος θα γίνει σε βήματα:
```
p <- ggplot(data = iris)
plot(p)
```
Με την πρώτη εντολή δημιουργείται και ανατίθεται σε μεταβλητή p ένα αντικείμενο το οποίο ανήκει σε διάφορες κλάσεις του πακέτου οι οποίες παρέχουν λειτουργικότητα γραφημάτων[462]. Έτσι, το αντικείμενο αυτό είναι το υπόβαθρο για ένα οποιοδήποτε γράφημα. Εδώ απλά έγινε ορισμός των δεδομένων που θα χρησιμοποιηθούν (το iris) αλλά αυτό δεν είναι υποχρεωτικό.

Η μέθοδος που χειρίζεται τη generic συνάρτηση plot (όπως και την print), θα εμφανίσει το γράφημα. Όμως το γράφημα θα εμφανιστεί κενό καθώς δεν έχουν οριστεί βασικά στοιχεία του όπως οι άξονες (άρα και οι μεταβλητές) που θα απεικονιστούν. Αυτό γίνεται με ένα αντικείμενο «αισθητικής προβολής» (aesthetic mapping ή απλώς aes) που δημιουργείται με τη συνάρτηση **aes** και ακολούθως προστίθεται στο p:
```
a <- aes(Petal.Width, Petal.Length)
p <- p + a
plot(p)
```
Τώρα το γράφημα εμφανίζει τους άξονες που ορίστηκαν μέσω της aes, αλλά δεν εμφανίζει ακόμα δεδομένα. Για να εμφανιστούν πρέπει να προστεθεί στο γράφημα και μια «γεωμετρία». Με την παρακάτω εντολή προστίθεται «γεωμετρία» που αντιστοιχεί σε διαγράμματα διασποράς, δηλαδή αυτή που δημιουργεί η **geom_point**:
```
g <- geom_point()
p <- p + g
plot(p)
```
Η plot (ή η print) εμφανίζοντας το p παράγει τώρα ένα γράφημα διασποράς με δεδομένα. Πάντως όλα τα παραπάνω συνδυάζονται συνήθως σε μία εντολή όπως:
```
ggplot(iris, aes(Petal.Width,Sepal.Length)) +
  geom_point()
```
Κάθε γράφημα έχει μόνο ένα αντικείμενο aesthetics, οπότε αν προστεθεί νέο θα αντικαταστήσει όποιο υπάρχει ήδη. Το παρακάτω, ορίζει ένα γράφημα βασισμένο στο p αλλά με άλλη μεταβλητή στον άξονα y και χρωματισμό (καθώς και διαχωρισμό των δεδομένων) βάσει του είδους (μεταβλητή Species):
```
p + aes(Petal.Width, Sepal.Length, colour = Species)
```

---

[459] βλ.§9.2.1 Το υπόβαθρο: πακέτα 'grDevices' και 'grid'.

[460] Τα πακέτα αυτά μπορούν να εγκατασταθούν από το CRAN, βλ. §1.5 Χρήση και διαχείριση πακέτων. Πριν τη χρήση τους πρέπει να συνδεθούν με την R με εντολή όπως η library(ggplot2).

[461] βλ. Παράρτημα Π.2 Το iris και άλλα σύνολα δεδομένων.

[462] Είναι κλάσεις S3, βλ.§6.3 Κλάσεις S3. Το αντικείμενο είναι βασισμένο σε λίστα ενώ οι κλάσεις που κληρονομεί μπορούν να εμφανιστούν με την εντολή class(p).

Με τα διάφορα δομικά στοιχεία που παρέχει το ‘ggplot2’ υπάρχει μεγάλη ευελιξία στη δημιουργία γραφημάτων. Εδώ έχουμε μια παραλλαγή του p όπως η προηγούμενη, αλλά έχει προστεθεί και μια ακόμα γεωμετρία, αυτή που παράγει θηκογράμματα (ή box-and-whisker plot). Το νέο γράφημα θα εμφανιστεί ταυτόχρονα με το γράφημα διασποράς που υπάρχει ήδη στο p.

```
p +
  aes(Petal.Width, Sepal.Length, colour = Species) +
  geom_boxplot(orientation = "y")
```

Η επόμενη εντολή χρησιμοποιεί τη συνάρτηση **stat_smooth** για να προσθέσει μια γραμμή τάσης στα δεδομένα (που θα υπολογίσει εφαρμόζοντας την lm με linear fitting στα δεδομένα).

```
p + stat_smooth(method = lm)
```

Με το παρακάτω, εμφανίζεται ένα γράφημα στήλης που απεικονίζει των αριθμό δειγμάτων ανά είδος (στο συγκεκριμένο dataset είναι όλα 50):

```
ggplot(iris,aes(Species)) + geom_bar()
```

Το επόμενο εμφανίζει ένα γράφημα πυκνότητας για τις τιμές της μεταβλητής Petal.Width ανά είδος (iris). Η παράμετρος alpha ρυθμίζει τον βαθμό διαφάνειας στα γραφικά στοιχεία μιας «γεωμετρίας»:

```
ggplot(iris,aes(x = Petal.Width , fill = Species) ) +
  geom_density(alpha = 0.4)
```

Ενώ το επόμενο, εμφανίζει ένα ιστόγραμμα όπου το εύρος τιμών της μεταβλητής Petal.Width έχει χωριστεί σε 6 bin, πάνω από το οποίο τοποθετείται γράφημα στήλης που καταγράφει τον αριθμό δειγμάτων ανά τιμή στην ίδια μεταβλητή:

```
ggplot(iris,
       aes(y = Petal.Width, fill=Species, color=Species)) +
  geom_histogram(bins=6,fill=NA) + geom_bar()
```

Στο επόμενο παράδειγμα, δημιουργείται ένα γράφημα όπου εμφανίζονται διασπορές από δυο διαφορετικά σύνολα δεδομένων (το iris και το sqiris που κατασκευάστηκε υψώνοντας τις αριθμητικές στήλες του iris στο τετράγωνο):

```
sqiris = iris; sqiris[1:4] = sqiris[1:4]^2

p <- ggplot() +
      aes(Petal.Width,Sepal.Length,colour=Species) +
      geom_point(data = iris) +
      geom_point(data = sqiris)

plot(p)
```

Το παρακάτω μεγεθύνει μια συγκεκριμένη περιοχή του p, με x από 1 έως 10 και y από 0 έως 60:

```
p + coord_cartesian(xlim=c(1,10), ylim=c(0, 60))
```

Το επόμενο, αλλάζει το σύστημα συντεταγμένων σε πολικές (εδώ η γωνία είναι βασισμένη στο y, άρα το Sepal.Length βάσει του aes):

```
p + coord_polar(theta='y')
```

Ενώ με το παρακάτω, αλλάζουν οι τίτλοι του γραφήματος και των αξόνων:

```
p + ggtitle("Δύο Διασπορές", subtitle="Βασισμένες στο iris") +
    xlab("Πλάτος Πετάλου") +
    ylab("Μήκος Σέπαλου")
```

Τέλος, η επόμενη εντολή αλλάζει την εικόνα του γραφήματος εφαρμόζοντας ένα «θέμα» (theme):

```
p + theme_light()
```

Στα theme που παρέχονται μπορούν να προστεθούν νέα. Ο τρόπος περιγράφεται στο [60], ενώ πληθώρα άλλων πηγών παρέχουν παραδείγματα και οδηγίες χρήσης του ευρέως χρησιμοποιούμενου αυτού πακέτου.

Στην πληθώρα των πακέτων που βασίζονται ή χρησιμοποιούν το ‘ggplot2’ παρέχονται πολύ ενδιαφέρουσες πρόσθετες δυνατότητες. Για παράδειγμα, το πακέτο ‘ggmap’ [109] επιτρέπει τη δημιουργία γεωγραφικών χαρτών ως γραφήματα ‘ggplot2’ (αντικείμενα αντίστοιχα αυτών που επιστρέφει η συνάρτηση ggplot) στα οποία μπορούν να προστεθούν «γεωμετρίες» και γενικότερα να γίνει χειρισμός τους με τρόπους που αναφέρθηκαν παραπάνω. Πολλά πακέτα προσθέτουν τύπους γραφημάτων ή παρέχουν συναρτήσεις για συνήθη γραφήματα, απλοποιώντας έτσι τις εντολές που απαιτούνται. Παράδειγμα των τελευταίων είναι το πακέτο ‘simplevis’ [110] που παρέχει συναρτήσεις δημιουργίας γραφημάτων, συνδυάζοντας τις δυνατότητες

του ‘ggplot2’ και άλλων πακέτων. Τέλος, κάποια πακέτα προσθέτουν διάδραση στα γραφήματα ‘ggplot2’, κάτι που παρουσιάζεται εκτενέστερα στην επόμενη ενότητα (§9.2.5 Διαδραστικά γραφήματα).

9.2.5 Διαδραστικά γραφήματα

Τα πακέτα που αναφέρονται ενδεικτικά στην ενότητα αυτή αφορούν τη δημιουργία γραφημάτων τα οποία αλληλεπιδρούν με τους χρήστες τους. Τα πακέτα που παρουσιάζονται παρακάτω δημιουργούν περιεχόμενο το οποίο είναι έτοιμο για χρήση στο web, ενσωμάτωση σε άλλες ιστοσελίδες κλπ. Αν η ανάπτυξή τους γίνεται στο RStudio, το αποτέλεσμα θα εμφανίζεται στην καρτέλα Viewer[463] η οποία παρέχει τη δυνατότητα εξαγωγής του γραφικού ως αυτόνομη σελίδα web (επιλογή Export/Save as Web Page). Πακέτα για διαδραστικά γραφήματα[464] περιλαμβάνουν τα ‘echarts4r’ [111] (βασισμένο στο via Apache ECharts), ‘highcharter’ [112] (βασισμένο στο Highcharts Javascript charting library), ‘ggiraph’ [113] και ‘plotly’ [114] (για την ομώνυμη βιβλιοθήκη διαγραμμάτων). Ακολουθεί μια σύντομη παρουσίαση για τα δύο τελευταία, καθώς συνδέονται και με το πακέτο ‘ggplot2’ που παρουσιάστηκε στην προηγούμενη ενότητα[465].

Το πακέτο ‘ggiraph’ μετατρέπει γραφήματα ‘ggplot2’ σε διαδραστικά στοιχεία web, στα οποία μπορεί να εμφανίζονται πληροφορίες, να γίνεται επιλογή, να εκτελούνται εντολές κλπ. Η μετάβαση γίνεται σε δυο βήματα. Γίνεται δημιουργία του γραφήματος ‘ggplot2’, αλλά αντί της όποιας «γεωμετρίας» χρησιμοποιείται μια με ίδιο όνομα στο οποίο έχει προστεθεί το κείμενο “_interactive”, π.χ. geom_point_interactive αντί του geom_point. Ακολούθως, δημιουργείται το διαδραστικό αντικείμενο με τη συνάρτηση **girafe.** Το αντικείμενο αυτό μπορεί να εμφανιστεί (με print), να εξαχθεί σε ιστοσελίδα, να συνεργαστεί με άλλα στοιχεία, να ενταχθεί σε δυναμικές web σελίδες βασισμενες σε Shiny (το πακέτο δίνει ειδική υποστήριξη για αυτό) κλπ. Ένα ενδιαφέρον χαρακτηριστικό του ‘ggiraph’ είναι η δυνατότητα επιλογής δεδομένων στο γράφημα αλλά και εκτέλεσης εντολών αν υπάρξει δράση, όπως κλικ με ποντίκι, σε αυτά. Αυτό γίνεται προσθέτοντας στα δεδομένα μια στήλη με εντολές Javascript που αντιστοιχούν σε κάθε στοιχείο και θα εκτελούνται αν υπάρξει δράση του χρήστη σε αυτό, ενώ η στήλη αυτή προστίθεται ως παράμετρος του aes με όνομα onclick). Ένα απλό παράδειγμα μετατροπής γραφήματος ‘ggplot2’ σε στοιχείο web ακολουθεί[466]:

```
iris2 <- iris
iris2$RowNumber <-rownames(iris2)

p <- ggplot(iris2) +
  aes(Petal.Width,
      Sepal.Length,
      tooltip = paste("Φυτό",RowNumber,"Είδος:",Species,
                      "Επιφάνεια=",Petal.Width*Petal.Length),
      data_id = RowNumber) +
  geom_point_interactive(size=2)

g <- girafe(ggobj = p,
            options = list(opts_selection(type = "multiple",
                                          only_shiny=FALSE)))

print(g)
```

Εδώ προστέθηκε στο γράφημα ένα tooltip που θα εμφανίζει τον αριθμό και είδος του φυτού όταν περνά από πάνω του ο δείκτης του ποντικιού (tooltip). Για τον λόγο αυτό έγινε μια μικρή παρέμβαση στα δεδομένα, προσθέτωντας μια στήλη που περιέχει τον αριθμό γραμμής κάθε εγγραφής. Μετά, σε μεταβλητή p ανατέθηκε το γράφημα που δημιουργήθηκε με τον συνήθη τρόπο για γραφήματα ‘ggplot2’. Η μόνη (προαιρετική) παρέμβαση είναι η προσθήκη στο aes στοιχείων για χρήση σε διαδραστικό mode, εδώ π.χ. δημιουργείται το κείμενο για το tooltip. Επίσης, αντί του geom_point χρησιμοποιήθηκε το geom_point_interactive. Οι επόμενες εντολές δημιουργούν το διαδραστικό web γράφημα (συνάρτηση girafe) και το εμφανίζουν.

---

[463] βλ.§1.4.2.3 Οι καρτέλες Plots και Viewer.

[464] Το πακέτα που αναφέρονται είναι διαθέσιμα στο CRAN, βλ. §1.5 Χρήση και διαχείριση πακέτων.

[465] βλ.§9.2.4 Πακέτο ‘ggplot2’.

[466] Για να λειτουργήσει το παράδειγμα πρέπει πρώτα να συνδεθούν τα πακέτα ‘ggplot2’ και ‘ggiraph’.

Το πακέτο ‘plotly’ είναι μια ισχυρότατη βιβλιοθήκη δημιουργίας γραφημάτων με πληθώρα σχετικών συναρτήσεων, αλλά επιτρέπει και αυτό τη μετατροπή γραφημάτων ‘ggplot2’. Το παρακάτω[467] δημιουργεί το αντίστοιχο διαδραστικό web γράφημα από ένα τέτοιο αντικείμενο, παρόμοιο δηλαδή με αυτό που είχαμε στο προηγούμενο παράδειγμα ενώ χρησιμοποιεί και τα ίδια δεδομένα (iris2):

```
p <- ggplot(iris2) +
  aes(Petal.Width,Sepal.Length,
      group = 1,
      text = paste("Φυτό",RowNumber,"Είδος:",Species,
                   "Επιφάνεια=",Petal.Width*Petal.Length)) +
    geom_point()

ggplotly(p, tooltip = "text")
```

Το γράφημα δημιουργήθηκε κανονικά με τη συνάρτηση ggplot του ‘ggplot2’, ενώ εδώ, προαιρετικά, προστέθηκε το text για τη δημιουργία του κειμένου στο tooltip. Η τελική μετατροπή σε web γράφημα έγινε με τη συνάρτηση **ggplotly** του ‘plotly’. Το γράφημα πλέον παρέχει τη δυνατότητα zoom σε κάποια περιοχή, εμφανίζει τον αριθμό, το είδος και άλλα στοιχεία του αντικειμένου μέσω tooltip, παρέχει τη δυνατότητα επιλογής κ.α. Όμως, όπως αναφέρθηκε ήδη, το πακέτο ΄plotly’ είναι κυρίως μια ισχυρή βιβλιοθήκη γραφημάτων. Για παράδειγμα, η επόμενη εντολή αξιοποιεί τις συναρτήσεις **plot_ly** και **add_markers** για να περάσει τα δεδομένα σε ένα γράφημα (αντικείμενο plotly) και μετά να προσθέσει τα σημεία. Το αποτέλεσμα είναι ένα γράφημα διασποράς, τριών αξόνων, στο οποίο μπορεί να γίνει περιστροφή, zoom κλπ.

```
plot_ly(iris, x = ~Petal.Length,
              y = ~Petal.Width,
              z = ~Sepal.Width) %>%
  add_markers(color = ~Species)
```

Τα επόμενα δύο παράδειγμα δημιουργούν ιστογράμματα. Στο πρώτο, ξεκινώντας από το αντικείμενο plotly προσθέτει ένα ιστόγραμμα και κάποιο κείμενο.

```
plot_ly(iris, y = ~Petal.Width) %>%
  add_histogram(nbinsy = 30) %>%
    add_text(x=18,y=2.3,text="Πλάτος Πετάλου")
```

Με παρόμοιο τρόπο, δημιουργείται ένα ιστόγραμμα δυο μεταβλητών, όπου η συχνότητα καθορίζει το χρώμα που εμφανίζεται:

```
plot_ly(iris, x = ~Petal.Length, y = ~Petal.Width) %>%
  add_histogram2d(nbinsx = 30,nbinsy = 30)
```

Με την ίδια ευκολία κατασκευάζεται μεγάλη ποικιλία γραφημάτων. Οι πηγές υλικού για το plotly είναι πολλές και περιλαμβάνουν το [115].

9.3 Εφαρμογές web

Προφανής χώρος εφαρμογής των διαδραστικών γραφημάτων της προηγούμενης ενότητας είναι οι εφαρμογές ιστού. Παρακάτω γίνεται μια σύντομη εισαγωγή σε δύο πακέτα που σχετίζονται με εφαρμογές web, τα ‘beakr’ και ‘shiny’

9.3.1 Πακέτο ‘beakr’

Το ‘beakr’ [116] αφορά δημιουργία υπηρεσιών web (web service) όπου η επεξεργασία θα γίνεται με κώδικα R. Το πακέτο δίνει έμφαση στην απλούστευση της υλοποίησης τέτοιων υπηρεσιών. Τα βασικά στοιχεία του πακέτου είναι μια συνάρτηση δημιουργίας του εξυπηρετητή (server, με τη συνάρτηση **newBeakr**), τρεις συναρτήσεις χειρισμού αιτημάτων που δημιουργούν αντικείμενα τα οποία μπορούν να προστεθούν στον server (συναρτήσεις **httpPOST**, **httpGET** και **httpPUT**, για αιτήματα POST, GET και PUT αντίστοιχα) και μια συνάρτηση προσαρμογής συναρτήσεων R για χρήση ως middleware στις παραπάνω υπηρεσίες (συνάρτηση **decorate**). Επιπροσθέτως, το πακέτο παρέχει βοηθητικές συναρτήσεις, μεταξύ αυτών συναρτήσεις χειρισμού του server όπως οι **stopServer**, **stopAllServers** κ.α. Ακολουθεί ένα παράδειγμα[468]:

[467] Για να λειτουργήσουν τα παραδείγματα που ακολουθούν πρέπει πρώτα να συνδεθεί το πακέτο ‘plotly’.
[468] Τα παραδείγματα που ακολουθούν απαιτούν να έχει πρώτα συνδεθεί το πακέτο, με εντολή όπως η library(‘beakr’).

```
newBeakr() %>%
httpGET("/", function(req, res, err) { "Welcome!"}) %>%
listen()
```

Το παραπάνω, δημιουργεί έναν web server στον οποίο μπορεί να γίνει πρόσβαση με οποιονδήποτε browser και εκτελείται τοπικά στην προεπιλεγμένη διεύθυνσή http://127.0.0.1:25118/ . Αν γίνει πρόσβαση στη διεύθυνση αυτή, εμφανίζεται το κείμενο “Welcome!”. Αυτό έγινε με τρία βήματα: (α) δημιουργήθηκε ο server με τη συνάρτηση newBeakr, (β) με τη συνάρτηση httpGET προστέθηκε μια υπηρεσία χειρισμού GET στο “/” η οποία εκτελεί συνάρτηση R (middleware) που επιστρέφει το κείμενο και (γ) ο server τέθηκε σε κατάσταση αναμονής αιτημάτων με τη συνάρτηση **listen**.

Το επόμενο παράδειγμα για το πακέτο ‘beakr’ στοχεύει στην ανάδειξη χειρισμού αιτημάτων υποβολής δεδομένων με GET και POST. Το παράδειγμα αφορά μια υπηρεσία που προσθέτει δύο αριθμούς. Για να γίνει αυτό ευκολότερο χρειαζόμαστε μια υποτυπώδη σελίδα που θα δέχεται τα δεδομένα σε μια φόρμα και θα τα υποβάλει (submit) στον server. Μια τέτοια σελίδα ανατίθεται παρακάτω στη μεταβλητή my_page:

```
my_page <-
'<!DOCTYPE html><html lang="el" >
<head><title>Beakr example</TITLE></head>
<body BGCOLOR="F4F4F4">
Add numbers!
</p>
<form action="/add" method="GET">
<input type = "text", name="x", value=""> +
<input type = "text", name="y", value="">
<input type="submit" value="Add (GET)" />
</form>
</p>
<form action="/addP" method="POST">
<input type = "text", name="x", value=""> +
<input type = "text", name="y", value="">
<input type="submit" value="Add (POST)" />
</form>
</p>
<a href="http://127.0.0.1:25118">Reload Page</a></p>
</p></p></body></html>'
```

Η υπηρεσία που παρέχεται είναι η πρόσθεση δύο αριθμών. Παρακάτω δημιουργείται η σχετική συνάρτηση:

```
f_add <- function(x,y)
{
  x<-as.numeric(x)
  y<-as.numeric(y)
  return(x+y)
}
```

Τώρα μπορεί να δημιουργηθεί ο server:

```
a_server <- newBeakr()
a_server %>%
  httpGET ("/",    function(req, res, err) { my_page }) %>%
  httpGET ("/add", decorate(f_add)) %>%
  httpPOST("/addP",decorate(f_add)) %>%
  handleErrors() %>%
  listen(host = "127.0.0.1", port = 25118)
```

Με τις παραπάνω εντολές ενεργοποιείται ο server (της μεταβλητής a_server) και απαντά στις διευθύνσεις:

127.0.0.1:25118, όπου η σχετική συνάρτηση απλώς επιστρέφει τη σελίδα my_page.

127.0.0.1:25118/add, που χειρίζεται υποβολή δύο αριθμών x και y με GET. Π.χ. για να προστεθούν οι αριθμοί 10 και 5 η κλήση είναι: http://127.0.0.1:25118/add?x=10&y=5).

127.0.0.1:25118/addP, που είναι ίδιο με το προηγούμενο για υποβολή μέσω POST.

Οι δύο τελευταίες υπηρεσίες καλούν ως middleware τη συνάρτηση f_add (ορίστηκε παραπάνω) αφού αυτή έχει προετοιμαστεί μέσω της συνάρτησης decorate του πακέτου.

9.3.2 Πακέτο 'shiny'

Το 'shiny' είναι ένα ιδιαίτερα διαδεδομένο πακέτο δημιουργίας εφαρμογών με δυναμικά αποκρινόμενες (reactive) ιστοσελίδες. Επιτρέπει τον ορισμό με κώδικα R της μορφής και συμπεριφοράς των στοιχείων του «πελάτη» (client ή ui, δηλαδή της σελίδας) και των λειτουργιών του εξυπηρετητή (server). Το'shiny' υποστηρίζεται από το RStudio με βοηθήματα δημιουργίας των σχετικών αρχείων (μενού New File/Shiny Web App), εκτέλεσης της εφαρμογής με εμφάνιση στην καρτέλα Viewer[469], δημοσίευσης (deploy) της εφαρμογής σε άλλους servers κ.α. Επίσης πολλά πακέτα παρέχουν υποστήριξη για τις εφαρμογές 'shiny'. Οι εφαρμογές 'shiny' μπορούν να επεκταθούν με χρήση διαφόρων τεχνικών που εφαρμόζονται στο web (CSS, htmlwidgets κ.α), να εξυπηρετήσουν ιστοσελίδες ή να αποτελέσουν τμήμα εγγράφων R Markdown[470] ή άλλων εφαρμογών παρέχοντας δυναμικό περιεχόμενο.

Μια εφαρμογή 'shiny' αποτελείται από δύο μέρη, το ui (user interface) που περιγράφει τη σελίδα και τον server που εξυπηρετεί τις ανάγκες του ui τροφοδοτώντας το με δεδομένα, αποτελέσματα, γραφήματα κλπ. Αυτά τα δύο μέρη ορίζονται είτε στο ίδιο είτε σε διαφορετικά σενάρια R[471]. Το ui δημιουργείται με συναρτήσεις όπως οι **fillPage**, **fixedPage**, **fluidPage** κ.α. που καλούνται με ορίσματα τα στοιχεία που θα αποτελούν τη σελίδα. Τα στοιχεία αυτά δημιουργούνται επίσης με συναρτήσεις του πακέτου 'shiny', όπως οι **textInput** (για δημιουργία ενός textbox που θα δέχεται κείμενο), **numericInput** (για δημιουργία ενός textbox που θα δέχεται αριθμούς), **textOutput** και **verbatimTextOutput** για δημιουργία στη σελίδα ενός στοιχείου εμφάνισης κειμένου, καθώς και πολλές άλλες. Το αποτέλεσμα είναι ένα αντικείμενο (κλάση shiny.tag.list) που θα χρησιμοποιηθεί για τη δημιουργία της σελίδας. Στο παράδειγμα[472] που ακολουθεί δημιουργείται μια σελίδα η οποία δέχεται δυο αριθμούς και εμφανίζει το άθροισμά τους (παρόμοιο με το παράδειγμα της προηγούμενης ενότητας). Ο υπολογισμός του αθροίσματος γίνεται στον server.

Το πρώτο μέρος του κώδικα R, αμέσως παρακάτω, ορίζει το ui. Εδώ ορίζεται απλώς η διεπαφή με τους χρήστες, η σελίδα δηλαδή που θα εμφανιστεί, τα στοιχεία που θα περιέχει και – εμμέσως - οι μεταβλητές που συνδέονται με αυτά:

```
my_ui <-
  fluidPage(title = "Shiny example",
    titlePanel("Πρόσθεση:"),
    numericInput(inputId = "x", label = "Αριθμός #1", "0"),
    numericInput(inputId = "y", label = "Αριθμός #2", "0"),
    div(style = "background-color: lightgrey",
              HTML("<b>Απάντηση")),
    textOutput(outputId = "d"),
    verbatimTextOutput(outputId = "v"))
```

Ο κώδικας αυτός δημιουργεί μια απλή σελίδα, ορίζει τον τίτλο της, προσθέτει δύο στοιχεία εισόδου αριθμών (με τη συνάρτηση numericInput, ονομασμένα x και y αντίστοιχα) ενώ λίγο παρακάτω προσθέτει δύο στοιχεία εξόδου (εδώ κειμένου, με τις συναρτήσεις textOutput και verbatimTextOutput, ονομασμένα d και v). Η εντολή αναθέτει το αποτέλεσμα στη μεταβλητή my_ui. Εμφάνιση (print) της μεταβλητής my_ui δίνει τα στοιχεία html που ορίστηκαν:

```
> my_ui
<div class="container-fluid">
  <h2>Πρόσθεση:</h2>
  <div class="form-group shiny-input-container">
    <label  class="control-label"  id="x-label"  for="x">Αριθμός
#1</label>
    <input id="x" type="number" class="form-control" value="0"/>
  </div>
```

469 βλ.§1.4.2.3 Οι καρτέλες Plots και Viewer.
470 βλ.§9.4 Δυναμικά έγγραφα.
471 βλ.§3.1 Σενάρια (R-script).
472 Το παράδειγμα απαιτεί να έχει πρώτα συνδεθεί το πακέτο, με εντολή όπως η library('shiny').

```
    <div class="form-group shiny-input-container">
      <label class="control-label" id="y-label" for="y">Αριθμός
#2</label>
      <input id="y" type="number" class="form-control" value="0"/>
    </div>
    <div style="background-color: lightgrey"><b>Απάντηση</div>
    <div id="d" class="shiny-text-output"></div>
    <pre class="shiny-text-output noplaceholder" id="v"></pre>
</div>
```

Όπως αναφέρθηκε ήδη, η επεξεργασία θα γίνεται στον server. Για να οριστεί η συμπεριφορά του server, αρκεί να δημιουργηθεί μια συνάρτηση όπως η παρακάτω:

```
my_server <- function(input, output)
{
  output$v <- renderText(input$x + input$y)

  output$d <- renderText(
    paste("Το αποτέλεσμα της πρόσθεσης",
          input$x,"συν",input$y,"είναι:") )
}
```

Η συνάρτηση δέχεται τρεις παραμέτρους. Η πρώτη (input) αφορά τη μεταφορά δεδομένων από τα στοιχεία του ui σε αντικείμενα R στον server, η δεύτερη αφορά τη μεταφορά αντικειμένων R από τον server προς τα στοιχεία του ui και η τρίτη (session, που εδώ παραλήφθηκε) ελέγχει τη συνεδρία. Τα input και output περιέχουν αντικείμενα αντιστοιχισμένα με στοιχεία του ui βάσει του ονόματος (id) των τελευταίων. Έτσι εδώ, το input περιέχει μεταβλητές με όνομα x και y που αντιστοιχούν στα στοιχεία με inputID x και y στο ui, ενώ το output περιέχει μεταβλητές d και v που αντιστοιχούν στα στοιχεία με outputID d και v. Συγκεκριμένα, ο κώδικας του server στο παράδειγμα ορίζει ότι το στοιχείο εξόδου με όνομα v του ui θα δέχεται ένα κείμενο που δημιουργείται από το άθροισμα των x και y. Τα τελευταία δόθηκαν ως είσοδος στα αντίστοιχα στοιχεία εισόδου του ui και πέρασαν στον server μέσα από την παράμετρο input. Με ανάλογο τρόπο θα αλλάζει και το στοιχείο εξόδου d. Αλλαγές στο ui προκαλούν «μηνύματα» που σηματοδοτούν ότι πρέπει να υπολογιστούν νέες τιμές εξόδου ώστε να ενημερωθούν τα περιεχόμενα των στοιχείων του ui που ενδεχομένως επηρεάζονται. Από τον τύπο των στοιχείων εξόδου που εμπλέκονται εδώ, αυτό γίνεται με reactive τρόπο, χωρίς να χρειάζεται να ανανεωθεί ολόκληρη η σελίδα αλλά μόνο τα στοιχεία του ui που εξαρτώνται από τις αλλαγές στην είσοδο.

Τέλος, για να δημιουργηθεί η εφαρμογή (ui και server) εκτελείται η συνάρτηση **shinyApp** στην οποία ορίζονται τα δύο μέρη της εφαρμογής, άρα για το παράδειγμά αυτό, με εντολή όπως:

```
shinyApp(my_ui, my_server)
```

Το πακέτο ‘shiny’ έχει υιοθετηθεί ως βάση σε πληθώρα εφαρμογών web, «πινάκων ελέγχου» (dashboard) κλπ. Επιτρέπει να γίνεται επεξεργασία των δεδομένων με R, τα αποτελέσματα της οποίας να τροφοδοτούν με δυναμικό περιεχόμενο τους τελικούς χρήστες, συμπεριλαμβανομένων (διαδραστικών ή μη) γραφημάτων που δημιουργούνται από τα πακέτα της γλώσσας. Παραδείγματα τέτοιων εφαρμογών υπάρχουν στα [117] [118], ενώ υπάρχει και σημαντικός αριθμός από σχετικές πηγές ή βιβλία όπως τα [119] [120] κ.α.

9.4 Δυναμικά έγγραφα

Η ενότητα αυτή ασχολείται με τη δημιουργία εγγράφων τα οποία ενσωματώνουν και εκτελούν κώδικα. Τέτοια έγγραφα βρίσκουν πολλές εφαρμογές, π.χ. τη δημιουργία εγγράφων που επεξηγούν με πιο πλήρη τρόπο κάποια μεθοδολογία που εφαρμόζεται, τη δημιουργία αναφορών που είναι επαναλαμβανόμενες και στις οποίες εκτελείται η ίδια διαδικασία επεξεργασίας με διαφορετικά δεδομένα, την υποβοήθηση δημιουργίας επιστημονικών ή επαγγελματικών εγγράφων όπου η επεξεργασία πρέπει να είναι αναπαράξιμη (reproducible) ή/και ελέγξιμη κλπ.

Έχει ήδη αναφερθεί ένας απλός τρόπος δημιουργίας αναφορών με ενσωματωμένο κώδικα R στην §3.1.5 Εργαλείο αναφοράς (report) όπου έγινε και μια πρώτη αναφορά στο RMarkdown, το ευρύτερο πλαίσιο για τη δημιουργία τέτοιων αναφορών. Οπως και για άλλα χρήσιμα εργαλεία του οικοσυστήματος της R, το RStudio υποβοηθά τη χρήση του RMarkdown διευκολύνοντας τη δημιουργία, την προεπισκόπηση και την τελική δημοσίευση ή εξαγωγή τέτοιων εγγράφων, καθώς και την εγκατάσταση των απαιτούμενων πακέτων ‘rmarkdown’ [121], ‘knitr’ [122] [123] [124] και πακέτων που αυτά χρησιμοποιούν.

Το RMarkdown είναι επέκταση του Markdown, μιας γλώσσας περιγραφής μορφοποιημένου κειμένου με απλό, αναγνώσιμο από τον άνθρωπο, τρόπο. Το τελικό, μορφοποιημένο έγγραφο δημιουργείται με πηγή ένα αρχείο απλού κειμένου στο οποίο έχουν χρησιμοποιηθεί κωδικοί μορφοποίησης που ορίζει το Markdown (και οι οποίοι είναι επίσης απλό κείμενο). Τα αρχεία Markdown (με επέκταση .md) μετατρέπονται στη συνέχεια σε διάφορες μορφές τελικών εγγράφων. Κατά τη μετατροπή εφαρμόζεται η μορφοποίηση που έχει οριστεί στο αρχείο Markdown. Πηγές για το συντακτικό της Markdown υπάρχουν πολλές[473] και δεν θα παρουσιαστεί περισσότερο σε αυτή την ενότητα, με εξαίρεση του απλού παραδείγματος Markdown που ακολουθεί:

```
Αυτό είναι ένα έγγραφο κειμένου που περιέχει κωδικούς Markdown.
Για παράδειγμα έχει ένα

## Τίτλο ενότητας
ενώ στο κείμενο μπορούν να γίνουν μορφοποιήσεις, π.χ. πλάγια
(όπως *αυτό*) ή έντονη γραφή (όπως **αυτό**), δείκτες (όπως
^αυτό^) και πολλές άλλες. Μπορούν να οριστούν πίνακες, λίστες,
σύνδεσμοι, εξισώσεις, μορφοποίηση για κώδικα ή κείμενο που
μεταφέρεται αυτολεξεί (quote) κλπ.
```

Κάποια από τα σύμβολα στο παραπάνω κείμενο είναι οδηγίες μορφοποίησης. Για παράδειγμα το ## οδηγεί σε μορφοποίηση επικεφαλίδας (δευτέρου μεγέθους), το κείμενο ανάμεσα σε ** θα εμφανιστεί με πλάγια γραφή κλπ. Αν θεωρήσουμε πως το παραπάνω βρίσκεται σε ένα αρχείο με όνομα "παράδειγμα.md", μπορεί να μετατραπεί σε άλλη μορφή (προεπιλεγμένη είναι η html) με εντολή όπως:

```
rmarkdown::render("παράδειγμα.md")
```

Το RMarkdown επεκτείνει το Markdown επιτρέποντας κάποια τμήματα του τελικού εγγράφου να δημιουργούνται από κώδικα R. Αυτό μπορεί να γίνει σε συνεργασία με κώδικα C++[474], Python[475], εντολές κελύφους Bash κ.α. Οι εντολές θα εκτελεστούν κατά τη μετάφραση του αρχείου RMarkdown (επέκταση .Rmd) σε τελικό έγγραφο ενω τα αποτελέσματα της εκτέλεσης θα εμπλουτίσουν το τελικό έγγραφο. Η τελική μορφή του εγγράφου μπορεί να είναι κείμενο, pdf, html, docx, ppt[476] κ.α. Πακέτα της R και τεχνικές που αναφέρθηκαν σε άλλες ενότητες μπορούν να αξιοποιηθούν εσωτερικά του αρχείου. Για παράδειγμα, εφόσον η τελική μορφή το υποστηρίζει, μπορεί να γίνει χρήση του 'shiny'[477] για δημιουργία διαδραστικών εγγράφων.

Ακολουθεί παράδειγμα εγγράφου γραμμένο σε RMarkdown. Το πρώτο μέρος (ανάμεσα στα ---) είναι ειδικό τμήμα του εγγράφου (ονομάζεται YAML header) όπου ορίζονται στοιχεία του αλλά και επιλογές που αφορούν τον τρόπο μετατροπής του στην τελική μορφή. Ακολουθεί ένα τμήμα κώδικα. Ένα τέτοιο τμήμα του εγγράφου αποκαλείται chunk και ορίζεται ανάμεσα σε ```. Εδώ έχουμε ένα chunk με κώδικα R, το όνομά του είναι 'προετοιμασία' και ζητείται να μην εμφανιστεί ο κώδικας αυτός στο τελικό έγγραφο (echo=FALSE). Οι εντολές στο συγκεκριμένο chunk είναι προαιρετικά βήματα προετοιμασίας για τον κώδικα που θα ακολουθήσει σε επόμενα chunks. Έτσι εδώ οι εντολές ορίζουν πως ο κώδικας των chunks γενικά, δεν θα πρέπει να εμφανίζεται στο τελικό κείμενο (αυτό γίνεται μέσω του αντικειμένου **opts_chunk** του πακέτου 'knitr'). Η επόμενη εντολή (συνάρτηση **Sys.setlocale**[478]) βεβαιώνει ότι το σύστημα θα χρησιμοποιήσει Ελληνικές ρυθμίσεις, ενώ η τελευταία (συνάρτηση **library**) συνδέει ένα πακέτο που θα χρησιμοποιηθεί παρακάτω.

Ακολουθεί κάποιο κείμενο Markdown που θα εμφανιστεί στο τελικό έγγραφο και ένα ακόμα chunk κώδικα R (με όνομα μέρος_α) που περιέχει κώδικα ο οποίος θα εκτελεστεί. Με την ίδια δομή δημιουργείται και το υπόλοιπο έγγραφο. Η πλήρης μετατροπή του αρχείου σε τελικό έγγραφο δεν επηρεάζεται από το τρέχον Global Environment[479], ούτε το επηρεάζει. Κατά τη διαδικασία αυτή, ο κώδικάς μέσα στα chunks ορίζει και αναζητά μεταβλητές σε ένα environment (κοινό για όλα τα chunks) το οποίο δημιουργείται για τον σκοπό της μετατροπής. Έτσι τα chunk που ακολουθούν έχουν πρόσβαση σε αντικείμενα που δημιουργήθηκαν σε προηγούμενα.

```
---
title: "Δοκιμαστικό Έγγραφο"
author: "Β. Νικολαΐδης"
```

---

[473] Το συντακτικό του Markdown περιγράφεται και στη σχετική με το RMarkdown βιβλιογραφία (βλ. παρακάτω).
[474] βλ.§7.4 C++.
[475] βλ.§7.3 Python
[476] Άλλες μορφές εκτός του html απαιτούν εγκατάσταση πρόσθετου λογισμικού ή/και πακέτων.
[477] βλ.§9.3.2 Πακέτο 'shiny'
[478] βλ. Παράρτημα Π.1.1 Προβλήματα που σχετίζονται με τα Ελληνικά.
[479] βλ.§2.2.2 Το καθολικό περιβάλλον (Global Environment).

````
date: "4/4/2022"
---

```{r προετοιμασία}
knitr::opts_chunk$set(echo = FALSE)
Sys.setlocale(category = "LC_ALL", locale = "Greek")
library(ggplot2)
```

### Παράδειγμα
Αυτό είναι κείμενο που θα μπει στο τελικό έγγραφο. Παρακάτω
δημιουργούνται δύο μεταβλητές και μια συνάρτηση. Ο κώδικας
αυτός θα εμφανιστεί στο τελικό κείμενο λόγω του echo=TRUE στον
ορισμό του chunk.

```{r μέρος_α, echo=TRUE}
g<-function(x) 2*x
x<-c(1,2,7,11,13,21,22)
y<-g(x)
```

Περισσότερο κείμενο και μετά ένα chunk κώδικα. Εδώ ο κώδικας
δεν εμφανίζεται στο τελικό κείμενο. Ο λόγος είναι ότι το
προεπιλεγμένο echo για τα chunks έχει οριστεί FALSE (βλ.
knitr::opts_chunk$set στην αρχή). Επίσης εδώ δεν εμφανίζεται
ούτε το αποτέλεσμα της print, καθώς include=FALSE στον ορισμό
του chunk.

```{r μέρος_β, include=FALSE}
d<-data.frame(x,y)
rownames(d)<-paste("Περίπτωση_",1:7,sep = "")
print(d)
```

Και κάποια αποτελέσματα. Παρατηρήστε το include=TRUE ώστε να
εμφανιστούν στο τελικό έγγραφο:

```{r εμφάνισε_το_d, include=TRUE}
print(d)
```

ή ακόμα καλύτερα:

```{r εμφάνισε_καλύτερα_το_d, include=TRUE}
knitr::kable(d)
```

καθώς και ένα γράφημα (εδώ μέσω του πακέτου 'ggplot2' που
συνδέθηκε σε προηγούμενο chunk)

```{r pressure, echo=FALSE}
qplot(x,y)
```

Ενώ σημαντικό είναι ότι αποτελέσματα μπορούν να ενσωματωθούν
στο κείμενο, π.χ. το `r sum(y)` εμφανίζει το άθροισμα του y.
````

Στο παράδειγμα αυτό, το τελευταίο μέρος του εγγράφου χρησιμοποιεί την συνάρτηση **kable** του ‘knitr’ για βελτίωση της εμφάνισης των πινάκων στο τελικό έγγραφο (εδώ εμφανίζει το data.frame με όνομα d), ενώ αμέσως μετά ενσωματώνει στο έγγραφο ένα γράφημα με αποτελέσματα. Επίσης, το τμήμα `r sum(y)` στην τελευταία γραμμή δείχνει τον τρόπο εκτέλεσης μιας εντολής R (εδώ της εντολής sum(y)) εντός του κειμένου. Στο τελικό έγγραφο το τμήμα αυτό θα αντικατασταθεί από το αποτέλεσμα (στην περίπτωση αυτή, τον αριθμό 154).

Στα πακέτα που επεκτείνουν τις δυνατότητες του RMarkdown περιλαμβάνονται τα ‘officer’ [125] και ‘officedown’ [126] τα οποία επιτρέπουν χειρισμό των στοιχείων εντός εγγράφων Microsoft Word και Powerpoint. Για περισσότερα σχετικά με RMarkdown, το σημαντικό αυτό εργαλείο δημιουργίας εγγράφων, βλ. το υλικό που παρέχεται από το RStudio [20] καθώς και τα [65], [127], [128] και [129].

Αναφορές Κεφαλαίου 9

[20] The RStudio Team, «RStudio». Ηλεκτρονικό. Προσπελάστηκε Φεβρουάριος, 2022, από https://www.rstudio.com/

[21] Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. New York: Springer.

[55] Wickham, H., François, R., Henry, L., Müller, K. (2021). dplyr: A Grammar of Data Manipulation. https://CRAN.R-project.org/package=dplyr και https://dplyr.tidyverse.org/

[57] Wickham H. et al. (2019). Welcome to the tidyverse. *Journal of Open Source Software, τόμ. 4, αρ. 43*, p. 1686.

[60] Peng, R. D., Kross, S., & Anderson, B. (2020). *Mastering Software Development in R*. Ηλεκτρονικό. https://bookdown.org/rdpeng/RProgDA/

[65] Wickham, H., & Grolemund, G. (2016). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data, 1st Edition.* O'Reilly Media, Inc. ISBN 978-1491910399.

[99] Wickham, H., & Bryan, J. (2022). Readxl: Read Excel Files. https://CRAN.R-project.org/package=readxl

[100] Wickham, H., Hester, J., & Bryan, J. (2022). Readr: Read Rectangular Text Data. https://CRAN.R-project.org/package=readr

[101] R Special Interest Group on Databases (R-SIG-DB), Wickham, H., & Muller, K. (2021). DBI: R Database Interface. https://CRAN.R-project.org/package=DBI

[102] Wickham, H., Girlich, M., & Ruiz, E. (2021). dbplyr: A 'dplyr' Back End for Databases. https://CRAN.R-project.org/package=dbplyr

[103] Wickham, H. (2021). Rvest: Easily Harvest (Scrape) Web Pages. https://CRAN.R-project.org/package=rvest

[104] Ryan, J. A., & Ulrich, J. M. (2020). Quantmod: Quantitative Financial Modelling Framework. https://CRAN.R-project.org/package=quantmod και http://www.quantmod.com/

[105] Chan Chung-hong, Chan Geoffrey CH, Leeper Thomas J., Becker Jason (2021). Rio: A Swiss-army knife for data file I/O. https://CRAN.R-project.org/package=rio

[106] Peng, R. D. (2016). *Exploratory Data Analysis with R*. lulu.com. ISBN 978-1365060069. https://bookdown.org/rdpeng/exdata/

[107] Zhou, L., & Braun, W. J. (2010). Fun with the R Grid Package. *Journal of Statistics Education, τόμ. 16, αρ. 3*. http://jse.amstat.org/v18n3/zhou.pdf

[108] Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. New York: Springer-Verlag. ISBN 978-3-319-24277-4.

[109] Kahle, D., & Wickham, H. (2013). ggmap: Spatial Visualization with ggplot2. *The R Journal, τόμ. 5, αρ. 1*, pp. 144--161. https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf

[110] Hodge, D. (2022). Simplevis: Simple 'ggplot2' and 'leaflet' Visualisation with Less Brainpower. https://CRAN.R-project.org/package=simplevis

[111] Coene, J. (2022). Echarts4r: Create Interactive Graphs with 'Echarts JavaScript' Version 5. https://echarts4r.john-coene.com/ και https://github.com/JohnCoene/echarts4r

[112] Kunst, J. (2022). Highcharter: A Wrapper for the 'Highcharts' Library. https://jkunst.com/highcharter/ και https://github.com/jbkunst/highcharter

[113] Gohel, D., & Skintzos, P. (2022). ggiraph: Make 'ggplot2' Graphics Interactive. https://CRAN.R-project.org/package=ggiraph

[114] Sievert, C. (2020). *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC. ISBN 9781138331457.

[115] Plotly, «Plotly Open Source Graphing Libraries», Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://plotly.com/r/

[116] Martin, H., & Callahan, J. (2021). Beakr: A Minimalist Web Framework for R. https://CRAN.R-project.org/package=beakr

[117] Appsilon, «R Shiny in Government – Top 7 Dashboards You Should See,» 5 4 2022., Ηλεκτρονικό. Προσπελάστηκε Απρίλιος 11, 2022, από https://appsilon.com/r-shiny-in-government-examples/

[118] RStudio, «Shiny Gallery», Ηλεκτρονικό. Προσπελάστηκε Απρίλιος, 2022, από https://shiny.rstudio.com/gallery/

[119] Wickham, H. (2021). *Mastering Shiny*. O'Reilly Media ISBN 978-1492047384. https://mastering-shiny.org/index.html

[120] Fay, C., Rochette, S., Guyader, V., & Girard, C. (2021). *Engineering Production-Grade Shiny Apps*. Chapman and Hall/CRC. ISBN 9780367466022. https://engineering-shiny.org/

[121] Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., & Iannone, R. (2022). Rmarkdown: Dynamic Documents for R. https://rmarkdown.rstudio.com

[122] Xie, Y. (2022). Knitr: A General-Purpose Package for Dynamic Report Generation in R. https://CRAN.R-project.org/package=knitr

[123] Xie, Y. (2015). *Dynamic Documents with R and knitr, 2nd Edition*. Chapman and Hall/CRC. ISBN 978-1498716963.

[124] Xie, Y. (2014). Knitr: A Comprehensive Tool for Reproducible Research in R. *Implementing Reproducible Computational Research*. Victoria Stodden, Friedrich Leisch, Roger D. Peng, Επιμ., Chapman and Hall/CRC.

[125] Gohel, D. (2022). Officer: Manipulation of Microsoft Word and PowerPoint Documents. https://CRAN.R-project.org/package=officer

[126] Gohel, D., & Noam, R. (2022). Officedown: Enhanced 'R Markdown' Format for 'Word' and 'PowerPoint'. https://CRAN.R-project.org/package=officedown

[127] Douglas, A., Roos, D., Mancini, F., Couto, A., & Lusseau, D. (2022). An Introduction to R. https://intro2r.com/

[128] Xie, Y., Allaire, J., & Grolemund, G. (2018). *R Markdown: The Definitive Guide*. Chapman and Hall/CRC. ISBN: 9781138359338. https://bookdown.org/yihui/rmarkdown

[129] Xie, V., Dervieux, C., & Riederer, E. (2020). *R Markdown Cookbook*. Chapman and Hall/CRC. ISBN 9780367563837. https://bookdown.org/yihui/rmarkdown-cookbook

Παραρτήματα

Π.1 Επίλυση προβλημάτων μετά την εγκατάσταση των R και RStudio

Π.1.1 Προβλήματα που σχετίζονται με τα Ελληνικά

Η R «τρέχει» σε πολλούς διαφορετικούς τύπους υπολογιστικών συστημάτων τα οποία μπορεί να έχουν διαφορές στις τοπικές ρυθμίσεις για τη γλώσσα, την απεικόνιση αριθμών κλπ. Αυτό μπορεί να προκαλέσει προβλήματα όταν γίνεται αποθήκευση Ελληνικών χαρακτήρων μέσα σε αρχεία ή χρήση τους σε ονόματα μεταβλητών ή και αρχείων. Για παράδειγμα, η χρήση ελληνικών ονομάτων μεταβλητών ή αρχείων μπορεί να μην υποστηρίζεται σωστά σε κάποιους συνδυασμούς τοπικών ρυθμίσεων του λειτουργικού συστήματος ή/και locale της R (που είναι οι αντίστοιχες επιλογές που χρησιμοποιεί η R για τοπικές ρυθμίσεις). Παρόμοια προβλήματα παρουσιάζονται και στο RStudio. Σε μερικά locale της R (όπως το English_United States.1252 που φορτώνεται εξ ορισμού αν ο υπολογιστής είναι ρυθμισμένος σε αντίστοιχη γλώσσα), κάποιοι ελληνικοί χαρακτήρες σε ονόματα μεταβλητών μετατρέπονται σε λατινικούς[480] (π.χ. το όνομα μεταβλητής 'α' μετατρέπεται σε 'a'), άλλοι παραμένουν χωρίς αλλαγή (π.χ. το όνομα 'β' γίνεται δεκτό χωρίς αλλαγή), ενώ άλλοι επιστρέφουν μήνυμα λάθους (π.χ. το όνομα 'γ΄). Μπορείτε να αποφύγετε κάποια τέτοια προβλήματα:

- Υιοθετήστε μόνο αγγλικά (λατινικούς χαρακτήρες) στον κώδικα, στα ονόματα μεταβλητών, στα δεδομένα, στα σχόλια, στα ονόματα αρχείων κλπ.
- Αν ο κώδικας R που δημιουργείτε περιέχει ελληνικά, για την αποθήκευσή του σε αρχεία επιλέξτε μια κωδικοποίηση χαρακτήρων που να υποστηρίζει τα Ελληνικά όπως η UTF-8. Στο RStudio, θα βρείτε τη σχετική ρύθμιση στο μενού Tools / Global Options / Code / Saving / Default Text Encoding.

Αν ο κώδικας περιέχει ελληνικούς χαρακτήρες (σε ονόματα μεταβλητών κλπ.) θα πρέπει επίσης να ενημερώσετε την R ώστε να χρησιμοποιήσει ελληνικές τοπικές ρυθμίσεις (locale) στην τρέχουσα συνεδρία. Για να δείτε το τρέχον locale στο οποίο είναι ρυθμισμένη η R, εκτελέστε την εντολή Sys.getlocale(). Εφόσον υπάρχουν ελληνικοί χαρακτήρες σε κώδικα, σε διαδρομές προς αρχεία ή φακέλους που χρησιμοποιεί κλπ., θα πρέπει το locale που επιστρέφει η παραπάνω εντολή να είναι "Greek". Για να αλλάξετε (αν χρειάζεται) το local σε "Greek" εκτελείτε την εντολή:

```
Sys.setlocale(category = "LC_ALL", locale = "Greek")
```

Η εντολή αυτή ρυθμίζει την R ώστε να χρησιμοποιήσει Ελληνικό locale για τη γλώσσα, τις ημερομηνίες, τα ονόματα (μεταβλητών, αρχείων, φακέλων) κλπ. (αντί του "English_United States.1252" ή άλλων αντίστοιχων που βρίσκουμε συχνά ως προεπιλογή). Η ρύθμιση αυτή ισχύει για την τρέχουσα συνεδρία με την R. Υπάρχουν τρόποι να γίνει μόνιμη αλλαγή του locale που θα χρησιμοποιεί η R (σε κάθε νέα συνεδρία), αλλά για τις ανάγκες του βιβλίου αυτού συνιστούμε μόνο τη χρήση των παραπάνω τεχνικών (και συγκεκριμένα την αποφυγή χρήσης ελληνικών χαρακτήρων σε ονόματα μεταβλητών, φακέλων, αρχείων κ.α.)[481]. Τέλος για να δείτε τα σύμβολα που χρησιμοποιεί η R αυτή τη στιγμή για αριθμούς κλπ., εκτελέστε Sys.localeconv(). Με την τελευταία μπορείτε και να αλλάξετε τα σύμβολα αυτά, π.χ. το σύμβολο των δεκαδικών ψηφίων, όμως η συγκεκριμένη αλλαγή δεν συνιστάται[482].

Π.1.2 Αδυναμία εγκατάστασης πρόσθετων πακέτων επέκτασης

Μετά την εγκατάσταση της R και του RStudio σε Microsoft Windows, έχει παρατηρηθεί, σε ορισμένες εκδόσεις τους, αδυναμία να γίνει εγκατάσταση νέων πρόσθετων πακέτων από τον χρήστη. Τα πρόσθετα πακέτα συνήθως εγκαθίστανται στον "home folder" (που συμβολίζεται με ~ σε πολλά λειτουργικά συστήματα). Στα Windows εξ ορισμού αυτό είναι το "Έγγραφα" (Documents) του χρήστη, στον φάκελο R που δημιουργείται εκεί. Όμως συγκεκριμένες ιδιαιτερότητες των Windows στον υπολογιστή σας όπως η παρουσία του λογισμικού OneDrive της Microsoft ή οι ελληνικοί χαρακτήρες στη διαδρομή (διεύθυνση) του φακέλου αυτού (η οποία περιλαμβάνει

---

[480] βλ. και help(gettext) σχετικά με την υποστήριξη τοπικών γλωσσών (Native Language Support/NLS).

[481] Για να είναι το βιβλίο αυτό πιο ευανάγνωστο πολλά παραδείγματά του χρησιμοποιούν Ελληνικούς χαρακτήρες (παρά την προτροπή αποφυγής τους). Αν υπάρχει πρόβλημα στα παραδείγματα αυτά, εκτελέστε την παραπάνω εντολή με τη συνάρτηση Sys.setlocale στην αρχή της συνεδρίας σας με την R.

[482] βλ. help(Sys.localeconv) και help(Sys.setlocale).

και το όνομα του χρήστη) μπορεί να προκαλούν προβλήματα. Εναλλακτικά, η R μπορεί να έχει ρυθμιστεί ώστε να χρησιμοποιεί τον φάκελο τις ίδιας της R ως χώρο απόθεσης νέων πακέτων (στον υπολογιστή σας συνήθως βρίσκεται σε θέση όπως το “C:/Program Files/R/R-x.y.z/library” όπου x.y.z ο αριθμός έκδοσης της R), στον φάκελο όμως αυτό συνήθως δεν επιτρέπεται η εγγραφή νέων αρχείων.

Γενικά υπάρχουν διάφορες αιτίες που μπορεί να προκαλέσουν το πρόβλημα όπως και αρκετοί τρόποι επίλυσής του. Ο πιο απλός όμως είναι να δημιουργήσετε έναν φάκελο για να αποθηκεύονται τα πακέτα και να ορίσετε μια “μεταβλητή περιβάλλοντος” στα Windows που θα ονομάζεται R_LIBS_USER και θα περιέχει τη διαδρομή για τον φάκελο αυτό. Δηλαδή:

Βήμα 1ο. Δημιουργήστε φάκελο σε κάποιο δίσκο για να αποθηκεύονται τα πακέτα της R που εγκαθιστάτε εσείς π.χ. στον δίσκο C έναν νέο φάκελο Rlibs (άρα εδώ η διαδρομή είναι C:\Rlibs).

Βήμα 2ο. Θα ορίσετε μια “μεταβλητή περιβάλλοντος” των Windows που να ονομάζεται R_LIBS_USER και να έχει τη διαδρομή για τον φάκελο που φτιάξατε πριν. Δυστυχώς τα ακριβή βήματα αλλάζουν σε διαφορετικές εκδόσεις των Windows αλλά είτε από τις Ρυθμίσεις, είτε με δεξί-κλικ στην Έναρξη και μετά στο Σύστημα, αναζητήστε την επιλογή “Ρυθμίσεις συστήματος για προχωρημένους” και επιλέξτε τη. Ακολούθως επιλέξτε στο παράθυρο που θα ανοίξει τις “Μεταβλητές Περιβάλλοντος” και στο επόμενο παράθυρο (Μεταβλητές Περιβάλλοντος) στον χώρο που αφορά τις μεταβλητές του χρήστη υπάρχει επιλογή “Νέα”. Δώστε ως όνομα της νέας μεταβλητής το R_LIBS_USER ενώ ως τιμή της μεταβλητής, δώστε τη διαδρομή για τον φάκελο που φτιάξατε π.χ. για το παράδειγμα μας: C:\Rlibs. Άλλος τρόπος να ορίσετε τη “μεταβλητή περιβάλλοντος” (αντί του προηγούμενου βήματος) είναι με τη χρήση του Windows PowerShell (υπάρχει με δεξί-κλικ στην Έναρξη) όπου μπορείτε να ορίσετε τη μεταβλητή εκτελώντας εκεί την εντολή:

```
[Environment]::SetEnvironmentVariable("R_LIBS_USER",
"C:\Rlibs", "User")
```

Εναλλακτικά, των παραπάνω μπορείτε να ορίσετε τις διαδρομές που χρησιμοποιεί η R για να εντοπίσει βιβλιοθήκες (δηλαδή συλλογές πακέτων της) μέσω του .libPaths() στο αρχείο .Rprofile ή στο αρχείο Rprofile.site. Τα δυο αυτά αρχεία κειμένου περιέχουν ρυθμίσεις και μπορείτε να τα επεξεργαστείτε π.χ. με το Σημειωματάριο. Το Rprofile.site επηρεάζει όλη την εγκατάσταση της R στον υπολογιστή σας και θα το βρείτε στον κύριο φάκελο της R, σε θέση όπως το “C:/Program Files/R/R-x.y.z/etc/Rprofile.site” όπου x.y.z ο αριθμός έκδοσης. Το .libPaths() είναι ένα διάνυσμα με τις διαδρομές στις οποίες η R αναζητά πακέτα (άρα αν το δώσετε ως εντολή στην R μπορείτε να δείτε ποιες διαδρομές είναι οι τρέχουσες). Για περισσότερα σχετικά με το θέμα αυτό πληκτρολογήστε help(Startup) και help(.libPaths).

Π.1.3 Αλλαγή του προεπιλεγμένου φακέλου εργασίας

Μπορείτε να ορίσετε κάποιον δικό σας φάκελο που θα έχει προεπιλεγεί να χρησιμοποιείται ως φάκελος εργασίας όταν ξεκινά η R (δηλαδή το default working directory). Ο φάκελος αυτός χρησιμοποιείται εξ ορισμού όταν δεν υπάρχει κάποιος συγκεκριμένος προορισμός, π.χ. όταν το RStudio προσπαθεί να αποθηκεύσει προσωρινά για να κάνει source ένα ενεργό έγγραφο R-script το οποίο δεν έχει ακόμα αποθηκευτεί σε κάποιο συγκεκριμένο αρχείο. Εξ ορισμού το RStudio χρησιμοποιεί το “home folder” (που συμβολίζεται με ~). Όπως προαναφέρθηκε, στα Windows ο φάκελος αυτός αντιστοιχεί στα “Έγγραφα” του χρήστη και συγκεκριμένες ιδιαιτερότητες των Windows στον υπολογιστή σας μπορεί να μην επιτρέπουν στο RStudio να χρησιμοποιήσει τον φάκελο αυτό ως τρέχοντα φάκελο εργασίας. Αν λοιπόν θέλετε να αλλάξει ο φάκελος που εξ ορισμού (default) χρησιμοποιεί το RStudio επιλέξτε το μενού Tools / Global Options / General, και στην καρτέλα Basic θα βρείτε την επιλογή Default Working Directory (when not in a project). Χρησιμοποιήστε το κουμπί “Browse” και επιλέξτε τον φάκελο που θέλετε να χρησιμοποιείται από το RStudio ως “default”. Σημείωση: ακόμα και αν έχουν γίνει οι παραπάνω αλλαγές έχουν παρατηρηθεί προβλήματα σε κάποιες εγκαταστάσεις εκδόσεων του RStudio σε Windows, όπου αποτυχαίνει το source ενός προσωρινού R Script (ενός σεναρίου R που δεν έχει αποθηκευτεί, π.χ. untitled1.R). Το πρόβλημα προκύπτει πάλι καθώς δεν επιτρέπεται στο RStudio να αλλάξει τον τρέχοντα φάκελο εργασίας στον φάκελο "Έγγραφα" (με την εντολή setwd(~\), ενώ – παραδόξως - η εντολή επιτρέπεται αν δοθεί απευθείας στην R). Το πρόβλημα σταματά όταν αποθηκευτεί το R Script σε κάποιο συγκεκριμένο αρχείο, οπότε και επιτρέπεται η εκτέλεση του με source.

Π.2 Το iris και άλλα σύνολα δεδομένων

Πέραν των συναρτήσεων και άλλων αντικειμένων, τα πακέτα της R συχνά περιέχουν και δεδομένα. Έτσι, στο πακέτο 'datasets' που είναι μέρος της system library (δηλαδή εγκαθίσταται μαζί με την ίδια την R) υπάρχει ένας αριθμός από ενδεικτικά σύνολα δεδομένων (data sets) κυρίως για χρήση τους σε παραδείγματα. Για να έχει η R πρόσβαση στα σύνολα αυτά πρέπει να συνδεθεί το πακέτο αυτό στην τρέχουσα συνεδρία, κάτι που συνήθως γίνεται αυτόματα ή εκτελώντας την εντολή library(datasets).

Ένα τέτοιο σύνολο δεδομένων είναι το iris, το οποίο φορτώνεται με το πακέτο 'datasets' ή με την εντολή data(iris). Το σύνολο αυτό περιέχει χαρακτηριστικά λουλουδιών από τρία διαφορετικά είδη του φυτού Ίριδα. Συγκεκριμένα, για κάθε δείγμα έχουν καταγραφεί (σε εκατοστά) το μήκος του σέπαλου (sepal length), το πλάτος του σέπαλου (sepal width), καθώς και το μήκος και το πλάτος του πέταλου (petal length και petal width). Τα τρία είδη του φυτού Iris, υπογένος Limniris, που χρησιμοποιήθηκαν είναι τα Iris setosa[483], Iris versicolor, και Iris virginica, ενώ το σύνολο περιέχει 50 περιπτώσεις από κάθε είδος. Τα δεδομένα στο data set iris είναι σε αντικείμενο τύπου data.frame[484] άρα έχει σχήμα πίνακα. Ο πίνακας αυτός περιέχει 150 γραμμές (παρατηρήσεις) 5 στηλών (μεταβλητών), όπου οι 4 πρώτες είναι οι μετρήσεις των προαναφερθέντων χαρακτηριστικών του κάθε δείγματος λουλουδιού, ενώ η 5η είναι factor[485] που καταγράφει το είδος του λουλουδιού στο οποίο έγινε η συγκεκριμένη παρατήρηση. Προφανώς, μπορείτε να δείτε τα δεδομένα γράφοντας iris ή View(iris) ενώ για περισσότερα στοιχεία για τα δεδομένα γράψτε help(iris). Τέλος, για να δείτε τα σύνολα δεδομένων που είναι διαθέσιμα στα εγκατεστημένα πακέτα εκτελέστε την εντολή data()[486].

[483] Π.χ. Για το είδος Iris setosa, βλ. https://en.wikipedia.org/w/index.php?title=Iris_setosa&oldid=1021726181
[484] βλ.§4.2.3 Ο τύπος data.frame (πλαίσιο δεδομένων).
[485] βλ.§4.1.2 Οι τύποι factor και ordered (παράγοντας).
[486] Όπως αναφέρθηκε παραπάνω, η συνάρτηση data επιτρέπει και την εισαγωγή συνόλων δεδομένων από αρχεία ή πακέτα.

Αναφορές – Βιβλιογραφία

[1] R Core Team (2021). R: A Language and Environment for Statistical Computing. Vienna, Austria. https://www.R-project.org/

[2] Crawley, M. J. (2014). *Εισαγωγή στη στατιστική ανάλυση με το R*. Αθήνα: Πασχαλίδης - Broken Hill. ISBN 978-996-371-625-8.

[3] Ανδρουλάκης, Γ., Witte, S. R., Witte, S. J., & Κουνέτας, Κ. (2019). *Στατιστική: Ανάλυση δεδομένων με χρήση της R*. Εκδόσεις Κριτική. ISBN 978-960-586-309-8.

[4] Ιωαννίδης, Δ., & Αθανασιάδης, Ι. (2017). *Στατιστική και μηχανική μάθηση με την R: Θεωρία και εφαρμογές (1η Έκδοση)*. Εκδόσεις Τζιόλα. ISBN 978-960-418-642-6.

[5] Ντζούφρας, Ι., & Καρλής, Δ. (2015). *Εισαγωγή στον προγραμματισμό και στη στατιστική ανάλυση με R* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. ISBN 978-960-603-449-7. https://hdl.handle.net/11419/2601

[6] Κολυβά-Μαχαίρα, Φ., Μπόρα-Σέντα, Ε., & Μπράτσας, Χ. (2018). *Στατιστική (2η Έκδοση)*. Εκδόσεις Ζήτη. ISBN 978-960-456-511-5.

[7] Νικολάου, Χ. (2019). *Ανάλυση Δεδομένων με την R*. Εκδόσεις Δίσιγμα. ISBN 978-618-5242-56-5.

[8] Κουνετάς, Κ., & Χατζησταμούλου, Ν. (2015). *Εισαγωγή στην επιχειρησιακή έρευνα και στον γραμμικό προγραμματισμό. Λύσεις προβλημάτων με το πρόγραμμα R* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. ISBN 978-960-603-301-8. https://hdl.handle.net/11419/5699

[9] Βερύκιος, Β., Καγκλής, Β., & Σταυρόπουλος, Η. (2015). *Η επιστήμη των δεδομένων μέσα από τη γλώσσα R* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. ISBN 978-960-603-394-0. https://hdl.handle.net/11419/2965

[10] Σταυρακούδης, Α. (2012). *Εισαγωγή στις υπολογιστικές μεθόδους για τις οικονομικές και επιχειρησιακές σπουδές.* Αθήνα: Κλειδάριθμος. ISBN 978-960-461-511-7.

[11] Κουτσουπιάς, Ν. Δ. (2018). *Πολυμεταβλητή ανάλυση δεδομένων με τη γλώσσα R*. Θεσσαλονίκη: Πανεπιστήμιο Μακεδονίας.

[12] Φουσκάκης, Δ. (2013). *Ανάλυση Δεδομένων με Χρήση της R*. Εκδόσεις Τσότρας.

[13] Θεμιστοκλέους, Χ. (2017). *Πειραματική μεθοδολογία και στατιστική στη γλωσσολογία, με τη χρήση R*. Εκδοτικός Όμιλος Ίων. ISBN 978-960-508-187-4.

[14] Πετράκος, Γ. (2016). *Εφαρμογές της Θεωρίας πιθανοτήτων με τη χρήση της R*. Εκδόσεις Τσότρας.

[15] Arratia, A. (2014). *Computational Finance*. Atlantis Press. ISBN 978-94-6239-069-0. http://doi.org/10.2991/978-94-6239-070-6

[16] Ang, C. S. (2015). *Analyzing Financial Data and Implementing Financial Models Using R*. Springer. ISBN 978-3-319-14074-2. http://doi.org/10.1007/978-3-319-14075-9

[17] Chambers, J. M. (1998). *Programming with Data*. New York: Springer-Verlag,The Green Book. ISBN 978-0-387-98503-9.

[18] Peng, R. D. (2015). *R Programming for Data Science*. Lean Publishing. https://bookdown.org/rdpeng/rprogdatascience/

[19] The R Foundation, «The R Project for Statistical Computing». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://www.r-project.org/

[20] The RStudio Team, «RStudio». Ηλεκτρονικό. Προσπελάστηκε Φεβρουάριος, 2022, από https://www.rstudio.com/

[21] Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. New York: Springer.

[22] De Vries, A. (2020). MiniCRAN: Create a Mini Version of CRAN Containing Only Selected Packages. https://CRAN.R-project.org/package=miniCRAN

[23] Eddelbuettel, D. (2021). Drat: 'Drat' R Archive Template. https://CRAN.R-project.org/package=drat

[24] Wickham, H., Hester, J., Chang, W., & Bryan, J. (2021). Devtools: Tools to Make Developing R Packages Easier. https://CRAN.R-project.org/package=devtools και https://devtools.r-lib.org/

[25] Fox, J. (2017). *Using the R Commander: A Point-and-Click Interface of R*. Boca Raton FL: Chapman

and Hall/CRC Press.

[26] Muenchen, R. A. (2022, February). R Graphical User Interface Comparison. Ηλεκτρονικό. https://r4stats.com/articles/software-reviews/r-gui-comparison/

[27] Cui, B. (2020). DataExplorer: Automate Data Exploration and Treatment. https://CRAN.R-project.org/package=DataExplorer

[28] Baniecki, H., & Przemyslaw, B. (2019, November). ModelStudio: Interactive Studio with Explanations for ML Predictive Models. *Journal of Open Source Software, τόμ. 4, αρ. 43*, p. 1798.

[29] Baruffa, O. et al., «*Big Book of R*». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://www.bigbookofr.com

[30] R-bloggers, «R-bloggers». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://www.r-bloggers.com/

[31] CodeProject, «The Code Project». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://www.codeproject.com/tags/r

[32] Stack Overflow, «The Stack Overflow». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://stackoverflow.com/questions/tagged/r

[33] Wickham, H. (2019). *Advanced R (2nd Editton)*. Chapman and Hall/CRC. https://adv-r.hadley.nz/

[34] Wickham, H. (2010, February). Stringr: modern, consistent string processing. *The R Journal, τόμ. 2, αρ. 2*. https://CRAN.R-project.org/package=stringr

[35] Wickham, H., Hester, J., & Ooms, J. (2021). Xml2: Parse XML. https://CRAN.R-project.org/package=xml2

[36] Benoit, K., Watanabe, K., & Wang, H. (2018). Quanteda: An R package for the quantitative analysis of textual data. *Journal of Open Source Software, τόμ. 3, αρ. 30*, p. 774. https://quanteda.io

[37] Feinerer, I., Hornik, K., & Meyer, D. (2008). Text Mining Infrastructure in R. *Journal of Statistical Software, τόμ. 5, αρ. 25*, pp. 1-54. https://www.jstatsoft.org/v25/i05/

[38] Jockers, M. L. (2015). Syuzhet: Extract Sentiment and Plot Arcs from Text. https://github.com/mjockers/syuzhet

[39] Proellochs, N., & Feuerriegel, S. (2021). SentimentAnalysis: Dictionary-Based Sentiment Analysis. https://CRAN.R-project.org/package=SentimentAnalysis

[40] Friedl, J. E. (2006). *Mastering Regular Expressions, Third Edition*. O'Reilly Media. ISBN 978-0596528126.

[41] Forta, B. (2018). *Learning Regular Expressions*. Addison-Wesley Professional. ISBN 9780134757063.

[42] Van Buuren, S. & Groothuis-Oudshoorn, K. (2011). Mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software, τόμ. 45, αρ. 3*, pp. 1-67. 10.18637/jss.v045.i03.

[43] Kowarik, A. & Templ, M. (2016). Imputation with the R Package VIM. *Journal of Statistical Software, τόμ. 74, αρ. 7*, pp. 1-16. 10.18637/jss.v074.i07.

[44] Chang, W., Luraschi, W., & Mast, T. (2021). Profvis: Interactive Visualizations for Profiling R Code. https://CRAN.R-project.org/package=profvis

[45] Hester, J., & Wickham, H. (2020). Fs: Cross-Platform File System Operations Based on 'libuv'. https://CRAN.R-project.org/package=fs

[46] RStudio Support, «Using RStudio Projects». Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2021, από https://support.rstudio.com/hc/en-us/articles/200526207-Using-RStudio-Projects

[47] Venables, W. N., Smith, D. M., & The R Core Team (2021). An Introduction to R. https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

[48] Grosjean, P., (2022). SciViews-R. https://www.sciviews.org/SciViews-R/

[49] Lawrence, M., & Temple Lang, D. (2010). RGtk2: A Graphical User Interface Toolkit for R. *Journal of Statistical Software, τόμ. 37, αρ. 8*, pp. 1--52. http://www.jstatsoft.org/v37/i08/

[50] R Core Team (2021). Tcltk: Tcl/Tk Interface, R Package version 4.1.2. https://www.R-project.org/

[51] Lawrence, M. F., & Verzani, J. (2012). *Programming Graphical User Interfaces in R*. New York:

Chapman and Hall/CRC. http://doi.org/10.1201/9781315373898
[52] Verzani, J. (2022). gWidgets2: Rewrite of gWidgets API for Simplified GUI Construction. https://CRAN.R-project.org/package=gWidgets2
[53] Grolemund, G., & Wickham, H. (2011). Dates and Times Made Easy with lubridate. *Journal of Statistical Software, τόμ. 40, αρ. 3*, pp. 1-25.
[54] The R Core Team (2021). R Language Definition. https://cran.r-project.org/doc/manuals/R-lang.html
[55] Wickham, H., François, R., Henry, L., Müller, K. (2021). dplyr: A Grammar of Data Manipulation. https://CRAN.R-project.org/package=dplyr και https://dplyr.tidyverse.org/
[56] Wickham, H. (2007). Reshaping Data with the reshape Package. *Journal of Statistical Software, τόμ. 21, αρ. 12*, pp. 1-20. http://www.jstatsoft.org/v21/i12/
[57] Wickham H. et al. (2019). Welcome to the tidyverse. *Journal of Open Source Software, τόμ. 4, αρ. 43*, p. 1686.
[58] Müller, K., & Wickham, H. (2021). Tibble: Simple Data Frames. https://CRAN.R-project.org/package=tibble
[59] Dowle, M., & Srinivasan, A. (2021). Data.table: Extension of `data.frame`. https://CRAN.R-project.org/package=data.table
[60] Peng, R. D., Kross, S., & Anderson, B. (2020). *Mastering Software Development in R*. Ηλεκτρονικό. https://bookdown.org/rdpeng/RProgDA/
[61] Ryan, J. A., & Ulrich, J. M. (2020). Xts: eXtensible Time Series. https://CRAN.R-project.org/package=xts
[62] Chambers, J. M. (2016). *Extending R, 1st Edition*. Chapman and Hall/CRC. ISBN 978-1-4987-7572-4.
[63] Altfeld, J. (2021). TryCatchLog: Advanced 'tryCatch()' and 'try()' Functions. https://CRAN.R-project.org/package=xts
[64] Milton Bache, S., & Wickham, H. (2022). Magrittr: A Forward-Pipe Operator for R. https://CRAN.R-project.org/package=magrittr
[65] Wickham, H., & Grolemund, G. (2016). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data, 1st Edition.* O'Reilly Media, Inc. ISBN 978-1491910399.
[66] Henry, L., & Wickham, H. (2020). Purrr: Functional Programming Tools. https://CRAN.R-project.org/package=purrr
[67] Rodrigues, B. (2020). Modern R with the tidyverse, Leanpub. https://b-rodrigues.github.io/modern_R/
[68] Burns, P. (2012). *The R Inferno, Second Edition*. lulu.com. ISBN 978-1471046520. https://www.burns-stat.com/documents/books/the-r-inferno/
[69] Chambers, J., & Hastie, T. (1992). *Statistical Models in S*. New York: Chapman & Hall/ CRC.The White Book. ISBN 9780203738535.
[70] Chang, W. (2021). R6: Encapsulated Classes with Reference Semantics. https://CRAN.R-project.org/package=R6 και https://r6.r-lib.org/index.html
[71] rForce Team (2022). JRI - Java/R Interface. http://www.rforge.net/JRI/index.html
[72] Eddelbuettel, D., Francois, R., & Bachmeier, L. (2020). RInside: C++ Classes to Embed R in C++ (and C) Applications. https://CRAN.R-project.org/package=RInside
[73] Simon, U. (2021). rJava: Low-Level R to Java Interface. https://CRAN.R-project.org/package=rJava
[74] Fuller, T. P. (2018). rGroovy: Groovy Language Integration. https://CRAN.R-project.org/package=rGroovy
[75] Tcl Core Team, (2022). Tcl Developer Xchange. https://www.tcl.tk/
[76] Welch, B., & Jones, K. (2003). *Practical Programming in Tcl and Tk, 4th edition*. Pearson. ISBN 978-0130385604.
[77] Ousterhout, J. K., & Jones, K. (2009). *Tcl and the Tk Toolkit*. 2nd edition. Addison-Wesley Professional. ISBN: 978-0321336330.
[78] Tierney, L. (2021). tkrplot: TK Rplot. https://CRAN.R-project.org/package=tkrplot

[79] Campelo, F. (2021). tkRplotR: Display Resizable Plots https://CRAN.R-project.org/package=tkRplotR
[80] Python Software Foundation, «Python Language Reference,» 2022. url: http://www.python.org
[81] Van Rossum, G. (1995). Python tutorial, Technical Report CS-R9526. Centrum voor Wiskunde en Informatica (CWI), Amsterdam.
[82] Belopolsky Alexander, Chapman Brad, Cock Peter, Eddelbuettel Dirk, Kluyver Thomas, Moreira Walter, Oget Laurent, Owens John, Rapin Nicolas, Slodkowicz Grzegorz, Smith Nathaniel, Warnes Gregory, the JRI author(s), the R authors et al. Documentation for rpy2. Ηλεκτρονικό. Προσπελάστηκε Απρίλιος, 2022, από https://rpy2.github.io/doc/v2.9.x/html/index.html
[83] Ushey, K., Allaire, J., & Tang, Y. (2022). Reticulate: Interface to 'Python'. https://CRAN.R-project.org/package=reticulate
[84] Eddelbuettel, D., & Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software, τόμ. 40, αρ. 8*, pp. 1-18. http://doi.org/10.18637/jss.v040.i08 .
[85] Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7. http://doi.org/10.1007/978-1-4614-6868-4 .
[86] Eddelbuettel, D., & Balamuta, J. J. (2018). Extending R with C++: A Brief Introduction to Rcpp. *The American Statistician, τόμ. 72, αρ. 1*, pp. 28-36. http://doi.org/10.1080/00031305.2017.1375990.
[87] RTools, Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://cran.r-project.org/bin/windows/Rtools/
[88] Eddelbuettel, D. et al., (2022, January). Rcpp: Seamless R and C++ Integration. Ηλεκτρονικό. https://www.rcpp.org/
[89] Eddelbuettel, D. (2022, February). Rcpp: Seamless R and C++ Integration. Ηλεκτρονικό. https://dirk.eddelbuettel.com/code/rcpp.html
[90] François, R., & Eddelbuettel, D. (2017). Rcpp Quick Reference Guide. https://dirk.eddelbuettel.com/code/rcpp/Rcpp-quickref.pdf
[91] Tsuda, M. E. (2020). Rcpp for everyone. Ηλεκτρονικό. https://teuder.github.io/rcpp4everyone_en/ [Πρόσβαση 2 2022].
[92] R Core Team, (2022). Writing R Extensions. https://cran.r-project.org/doc/manuals/R-exts.html
[93] Leisch, F. (2008). Creating R Packages: A Tutorial. Compstat 2008-Proceedings, Heidelberg, Germany.
[94] Wickham, H. (2015). *R Packages*. O'Reilly Media. ISBN 978-1491910597.https://r-pkgs.org/
[95] Nikolaidis, V. N. (2021). The nnlib2 library and nnlib2Rcpp R package for implementing neural networks. *Journal of Open Source Software, τόμ. 6, αρ. 61*, p. 2876. http://doi.org/10.21105/joss.02876.
[96] Wickham, H., Danenberg, P., Csárdi, G., & Eugster, M. (2021). Roxygen2: In-Line Documentation for R. https://CRAN.R-project.org/package=roxygen2
[97] Wickham, H., Bryan, J., & Barrett, M. (2021). Usethis: Automate Package and Project Setup. https://CRAN.R-project.org/package=usethis
[98] Wickham, H. (2011). Testthat: Get Started with Testing. *The R Journal, τόμ. 3, αρ. 1*, pp. 5-10. https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf
[99] Wickham, H., & Bryan, J. (2022). Readxl: Read Excel Files. https://CRAN.R-project.org/package=readxl
[100] Wickham, H., Hester, J., & Bryan, J. (2022). Readr: Read Rectangular Text Data. https://CRAN.R-project.org/package=readr
[101] R Special Interest Group on Databases (R-SIG-DB), Wickham, H., & Muller, K. (2021). DBI: R Database Interface. https://CRAN.R-project.org/package=DBI
[102] Wickham, H., Girlich, M., & Ruiz, E. (2021). dbplyr: A 'dplyr' Back End for Databases. https://CRAN.R-project.org/package=dbplyr
[103] Wickham, H. (2021). Rvest: Easily Harvest (Scrape) Web Pages. https://CRAN.R-project.org/package=rvest

[104] Ryan, J. A., & Ulrich, J. M. (2020). Quantmod: Quantitative Financial Modelling Framework. https://CRAN.R-project.org/package=quantmod και http://www.quantmod.com/

[105] Chan Chung-hong, Chan Geoffrey CH, Leeper Thomas J., Becker Jason (2021). Rio: A Swiss-army knife for data file I/O. https://CRAN.R-project.org/package=rio

[106] Peng, R. D. (2016). *Exploratory Data Analysis with R*. lulu.com. ISBN 978-1365060069. https://bookdown.org/rdpeng/exdata/

[107] Zhou, L., & Braun, W. J. (2010). Fun with the R Grid Package. *Journal of Statistics Education, τόμ. 16, αρ. 3*. http://jse.amstat.org/v18n3/zhou.pdf

[108] Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. New York: Springer-Verlag. ISBN 978-3-319-24277-4.

[109] Kahle, D., & Wickham, H. (2013). ggmap: Spatial Visualization with ggplot2. *The R Journal, τόμ. 5, αρ. 1*, pp. 144--161. https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf

[110] Hodge, D. (2022). Simplevis: Simple 'ggplot2' and 'leaflet' Visualisation with Less Brainpower. https://CRAN.R-project.org/package=simplevis

[111] Coene, J. (2022). Echarts4r: Create Interactive Graphs with 'Echarts JavaScript' Version 5. https://echarts4r.john-coene.com/ και https://github.com/JohnCoene/echarts4r

[112] Kunst, J. (2022). Highcharter: A Wrapper for the 'Highcharts' Library. https://jkunst.com/highcharter/ και https://github.com/jbkunst/highcharter

[113] Gohel, D., & Skintzos, P. (2022). ggiraph: Make 'ggplot2' Graphics Interactive. https://CRAN.R-project.org/package=ggiraph

[114] Sievert, C. (2020). *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC. ISBN 9781138331457.

[115] Plotly, «Plotly Open Source Graphing Libraries», Ηλεκτρονικό. Προσπελάστηκε Μάρτιος, 2022, από https://plotly.com/r/

[116] Martin, H., & Callahan, J. (2021). Beakr: A Minimalist Web Framework for R. https://CRAN.R-project.org/package=beakr

[117] Appsilon, «R Shiny in Government – Top 7 Dashboards You Should See,» 5 4 2022., Ηλεκτρονικό. Προσπελάστηκε Απρίλιος 11, 2022, από https://appsilon.com/r-shiny-in-government-examples/

[118] RStudio, «Shiny Gallery», Ηλεκτρονικό. Προσπελάστηκε Απρίλιος, 2022, από https://shiny.rstudio.com/gallery/

[119] Wickham, H. (2021). *Mastering Shiny*. O'Reilly Media ISBN 978-1492047384. https://mastering-shiny.org/index.html

[120] Fay, C., Rochette, S., Guyader, V., & Girard, C. (2021). *Engineering Production-Grade Shiny Apps*. Chapman and Hall/CRC. ISBN 9780367466022. https://engineering-shiny.org/

[121] Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., & Iannone, R. (2022). Rmarkdown: Dynamic Documents for R. https://rmarkdown.rstudio.com

[122] Xie, Y. (2022). Knitr: A General-Purpose Package for Dynamic Report Generation in R. https://CRAN.R-project.org/package=knitr

[123] Xie, Y. (2015). *Dynamic Documents with R and knitr, 2nd Edition*. Chapman and Hall/CRC. ISBN 978-1498716963.

[124] Xie, Y. (2014). Knitr: A Comprehensive Tool for Reproducible Research in R. *Implementing Reproducible Computational Research*. Victoria Stodden, Friedrich Leisch, Roger D. Peng, Επιμ., Chapman and Hall/CRC.

[125] Gohel, D. (2022). Officer: Manipulation of Microsoft Word and PowerPoint Documents. https://CRAN.R-project.org/package=officer

[126] Gohel, D., & Noam, R. (2022). Officedown: Enhanced 'R Markdown' Format for 'Word' and 'PowerPoint'. https://CRAN.R-project.org/package=officedown

[127] Douglas, A., Roos, D., Mancini, F., Couto, A., & Lusseau, D. (2022). An Introduction to R. https://intro2r.com/

[128] Xie, Y., Allaire, J., & Grolemund, G. (2018). *R Markdown: The Definitive Guide*. Chapman and Hall/CRC. ISBN: 9781138359338. https://bookdown.org/yihui/rmarkdown

[129] Xie, V., Dervieux, C., & Riederer, E. (2020). *R Markdown Cookbook*. Chapman and Hall/CRC. ISBN 9780367563837. https://bookdown.org/yihui/rmarkdown-cookbook

[130] The R Core Team (2021). R Internals. https://cran.r-project.org/doc/manuals/r-release/R-ints.pdf

Το σύστημα R είναι ένα εργαλείο που χρησιμοποιείται ευρέως σε εφαρμογές στατιστικής, ανάλυσης δεδομένων και μηχανικής μάθησης. Επιπρόσθετα, η επεκτασιμότητα και η ευελιξία της R την έχουν κάνει κατάλληλη για εφαρμογές σε πολλούς άλλους τομείς και γνωστικά πεδία. Η R μπορεί να χρησιμοποιηθεί με διαδραστικό τρόπο, χωρίς προγραμματισμό, αλλά ταυτόχρονα είναι γλώσσα προγραμματισμού και εργαλείο ανάπτυξης λογισμικού. Σε αυτή τη δεύτερη φύση της R (σε αυτήν, της γλώσσας προγραμματισμού), επικεντρώνεται το βιβλίο αυτό. Με έμφαση στη χρήση παραδειγμάτων, το βιβλίο ξεκινά από την αρχική επαφή με τον προγραμματισμό και την R, και προχωρά προοδευτικά σε πιο προχωρημένα θέματα. Αρχικά στοχεύει να παρουσιάσει την R σε νέους προγραμματιστές ή νέους χρήστες της συγκεκριμένης γλώσσας χρησιμοποιώντας απλά, οικεία παραδείγματα γενικού προγραμματισμού και παρακάμπτοντας, σε κάποιο βαθμό, την παρουσίασή της ως επιστημονικού εργαλείου. Εδώ, παρουσιάζονται οι σχετικές έννοιες, τεχνικές και τα εργαλεία που αξιοποιούνται από τους προγραμματιστές, οι βασικές αρχές που διέπουν τη γλώσσα R, οι ιδιαιτερότητές της, οι δυνατότητες που προστίθενται σε αυτήν και παράγονται από το οικοσύστημά της (πηγές, πακέτα, περιβάλλοντα ανάπτυξης εφαρμογών κλπ.). Ακολούθως, στο δεύτερο μέρος του, το βιβλίο προχωρά σε περισσότερο προχωρημένα θέματα με την παρουσίαση κάποιων από τις δυνατότητες της γλώσσας αυτής, οι οποίες επιτρέπουν την ανάπτυξη πιο σύνθετων εφαρμογών και λύσεων.

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: 978-618-5667-90-0	**DOI:** http://dx.doi.org/10.57713/kallipos-100

Το παρόν σύγγραμμα χρηματοδοτήθηκε από το Πρόγραμμα Δημοσίων Επενδύσεων του Υπουργείου Παιδείας.