Εισαγωγή στην R

Ειαγωγή δεδομένων

Τα δεδομένα στην R συνήθως τα παίρνουμε από εξωτερικές πηγές, πχ αρχεία τύπου CSV ή XLSX, λήψη δεδομένων από βάσεις δεδομένων, κλπ. Εδώ θα εξετάσουμε μερικές απλές περιπτώσεις όπου τα δεδομένα εισάγονται απ' ευθείας από τον χρήστη. Από τους πολλές δομές δεδομένων που υποστηρίζει η R θα ασχοληθούμε με τις 3 βασικότερες:

1. Βαθμωτές μεταβλητές
2. Διανύσματα τιμών
3. Πίνακες τιμών

1.1 Βαθμωτές μεταβλητές

Ας υποθέσουμε για παράδειγμα πως θέλουμε να εκχωρήσουμε την τιμή 5 στην μεταβλητή χ, να πως μπορούμε να το κάνουμε:

```
x = 5
```

Αν και το παραπάνω είναι μια απολύτως έγκυρη παράσταση εκχώρησης τιμής στην R χρησιμοποιούμε έναν διαφορετικό τελεστή εκχώρησης τιμών:

```
x <- 5
```

Δηλαδή ένα βέλος που δείχνει προς τα αριστερά, συνδυασμός δύο χαρακτήρων, - και <.

Στην R έχουμε επίσης και αυτή τη δυνατότητα:

```
5 -> x
```

Δηλαδή τιμή με βέλος που δείχνει δεξιά προς το όνομα της μεταβλητής. Η χρήση αυτού του τρόπου είναι σπάνια, παρόλο την εγκυρότητα που έχει. Επίσης, επειδή δεν είναι και πολύ καλή ιδέα να ανακατεύουμε τους τρόπους με τους οποίους γράφουμε κώδικα, ούτε και είναι σωστό να γράφουμε κώδικα με τρόπο εντελώς διαφορετικό από άλλες γλώσσες προγραμματισμού, στη συνέχεια αυτού του βιβλίου οι παραστάσεις εκχώρησης τιμής θα δηλώνονται πάντοτε ως:

```
x <- 5
```

Από τη στιγμή που σε μια μεταβλητή έχει εκωρηθεί τιμή μπορούμε να τη χρησιμοποιήσουμε, πχ

```
x + 1
```

```
## [1] 6
```

ή

```
2/x
```

```
## [1] 0.4
```

Φυσικά μπορούμε να χρησιμοποιήσουμε την μεταβλητή όσες φορές θέλουμε. Επίσης, αν κάτι τέτοιο είναι επιθυμητό, μπορούμε να αλλάξουμε την τιμή της:

```
x <- 10
```

οπότε:

```
x + 1
```

```
## [1] 11
```

ή ακόμα και να διαγράψουμε τη μεταβλητή από την μνήμη:

```
rm(x)
```

οπότε η προσπάθεια χρησιμοποίησης της θα προκαλέσει μήνυμα σφάλματος.

Προς το παρόν είδαμε την πολύ απλή περίπτωση μιας ακέραιας τιμής σε μεταβλητή. Τέτοιες τιμές μπορεί να είναι δεκαδικές, πχ:

```
x <- 0.52
```

ή

```
x <- 12.091
```

Προσοχή, η τελεία υποδηλώνει το δεκαδικό μέρος του αριθμού. Δεν πρέπει να γίνεται σύγχυση με τους δεκαδικούς αριθμούς που χρησιμοποιούμε στην Ελλάδα όπου η υποδιαστολή είναι το κόμμα. Πολλές φορές, ιδιαίτερα σε μεγάλους αριθμούς, μπορεί να δούμε το διαχωρισμό χιλιάδων. Πχ, στην Ελλάδα γράφουμε 3.952,34 και εννοούμε τον αριθμό: τρεις χιλιάδες εννιακόσια πενήντα δύο και τριάντα τέσσερα εκατοστά. Τον αριθμό αυτό θα τον εισάγουμε ως:

```
x <- 3952.34
```

Δηλαδή χωρίς τον διαχωρισμό χιλιάδων και με τελεία ως διαχωριστικό δεκαδικών ψηφίων.

1.1.1 Αριθμοί με επιστημονική γραφή

Στην R μπορούμε να εισάγουμε αριθμούς με επιστημονική γραφή, όπως πχ στο MS Excel, ή σε άλλες γλώσσες προγραμματισμού. Για παράδειγμα, τον αριθμό 100 μπορούμε να τον εισάγουμε ως:

```
1E+2
```

```
## [1] 100
```

είτε ως:

```
1E2
```

```
## [1] 100
```

είτε ως

```
1e2
```

```
## [1] 100
```

Ενώ τον αριθμό 580.1 μπορούμε να τον εισάγουμε ως:

```
5.801e2
```

```
## [1] 580.1
```

Υπενθυμίζεται πως κάθε πραγματικός αριθμός μπορεί να γραφεί με τη μορφή:

$$m \times 10^n 1 \leq m < 10$$

δηλαδή ως αριθμός μεταξύ 1 και 10 πολλαπλασιαμένος με ακέραια δύναμη του 10. Αυτό ισχύει τόσο για αριθμούς μεγαλύτερους του 10 όσο και για αριθμούς μικρότερους της μονάδας. Πχ, αριθμός 0.054 μπορεί να γραφεί ως:

```
5.4e-2
```

```
## [1] 0.054
```

εφόσον ισχύει: $0.054 = 5.4 \times 10^{-2}$. Ας δούμε μερικά παραδείγματα:

```
2e4
```

```
## [1] 20000
```

```
5.5e6
```

```
## [1] 5500000
```

```
1.17e-3
```

```
## [1] 0.00117
```

Αξίζει να σημειωθεί πως ενώ στην τυπική επιστημονικής γραφής η βάση είναι πάντοτε μετά 1 και 10 στην R μπορούμε να χρησιμοποιήσουμε οποιαδήποτε άλλη βάση, Πχ

```
0.04e4
```

```
## [1] 400
```

για να δηλώσουμε τον αριθμό 400, ή:

```
40000e-2
```

```
## [1] 400
```

για να δηλώσουμε και πάλι τον αριθμό 400. Ο σωστός τρόπος είναι:

```
4e2
```

```
## [1] 400
```

Παρόλο λοιπόν που έχουμε την ευχέρεια να δηλώσουμε τον ίδιο αριθμό πρακτικά με πολλούς τρόπους πάντοτε προτιμούμε τον τυπικό, όπως ορίζουν οι κανονισμοί.

Αν χρειαστεί να εκχωρήσουμε μια τέτοια τιμή σε μια μεταβλητή, δεν έχει σημασία πως ακριβώς θα το κάνουμε. Πχ οι παραστάσεις:

```
x <- 518.4
x <- 5.184e22
```

είναι ισοδύναμες. Πράξεις με την μεταβλητή x θα αποδώσουν το ίδιο αποτέλεσμα και στις δύο περιπτώσεις.

1.1.2 Τιμές κειμένου και αλφαριθμητικών

Σε μερικές περιπτώσεις είναι πιθανό να χρειαστεί να εκχωρήσουμε μια τιμή που είναι λέξη, ή απλά μια αλληλουχία χαρακτήρων. Σε αυτή την περίπτωση θα πρέπει να χρησιμοποιήσουμε εισαγωγικά, είτε διπλά:

```
city <- "Αθήνα"
```

είτε απλά:

```
city <- 'Αθήνα'
```

Με μόνη προϋπόθεση να είναι ο ίδιος τύπος εισαγωγικών και από τις δύο πλευρές. Ανάμιξη μπορεί να γίνει, όταν το εισαγωγικό είναι μέρος του λεκτικού. Πχ

```
w <- "σ' αγαπώ"
```

Αλλά και σε αυτή την περίπτωση αρχή και τέλος έχουν τον ίδιο τύπου εισαγωγικού.

1.1.3 Χρήσιμες μαθηματικές συναρτήσεις

Πολύ συχνά χρειάζεται να εφαρμόσουμε γνωστές μαθηματικές συναρτήσεις, όπως για παράδειγμα η τετραγωνική ρίζα, ο λογάριθμος, το ημίτονο κα. Ας δούμε μέσα από μερικά παραδείγματα τις πλέον συνηθισμένες περιπτώσεις.

Τετραγωνική ρίζα με συνάρτηση sqrt, για παράδειγμα:

```
sqrt(2)
```

```
## [1] 1.414214
```

Δεκαδικός λογάριθμος, για παράδειγμα:

```
log10(2)
```

```
## [1] 0.30103
```

Νεπέριος λογάριθμος (με βάση το e):

```
log(2)
```

```
## [1] 0.6931472
```

Η R έχει επίσης διαθέσιμη τη συνάρτηση log2, το λογάριθμο με βάση το 2:

```
log2(2)
```

```
## [1] 1
```

Αντίστοιχα η συνάρτηση exp υπολογίζει την ύψωση στη βάση των φυσικών λογαρίθμων:

```
exp(2)
```

```
## [1] 7.389056
```

ή, αν θέλουμε να υπολογίσουμε το e (τη βάση των φυσικών λογαρίθμων):

```
exp(1)
```

```
## [1] 2.718282
```

Η λογαρίθμηση και η ύψωση στη βάση των φυσικών λογαρίθμων αν εφαρμοστούν διαδοχικά αλληλοεξουδετερώνονται, όπως ακριβώς γνωρίζουμε από τα μαθηματικά:

```
log(exp(2))
```

```
## [1] 2
```

```
exp(log(2))
```

```
## [1] 2
```

Η R διαθέτει επίση μία ενσωματωμένη σταθερά που αντιστοιχεί στον αριθμό π:

```
pi
```

```
## [1] 3.141593
```

Αν χρειαστούμε τον αριθμό π τον εισάγουμε πάντοτε με αυτό τον τρόπο και ποτέ με κάποια άλλη προσέγγιση, πχ 3.14159. Για παράδειγμα το εμβαδόν κύκλου με ακτίνα 3 είναι:

```
pi * 3^2
```

```
## [1] 28.27433
```

Οι τριγωνομετρικές συναρτήσεις έχουν το ίδιο όνομα όπως και άλλα περιβάλοντα (Excel, matlab, κα). Για παράδειγμα το ημίτονο του $\pi/4$ είναι:

```
sin(pi/4)
```

```
## [1] 0.7071068
```

το συνημίτονο του $\pi/4$ είναι:

```
cos(pi/4)
```

```
## [1] 0.7071068
```

και η εφαπτομένη είναι:

```
tan(pi/4)
```

```
## [1] 1
```

Όπως σε όλα τα υπολογιστικά περιβάλλοντα το όρισμα μιας τριγωνομετρικής συνάρτησης είναι σε ακτίνια, όχι σε μοίρες. Αν για παράδειγμα θέλουμε να υπολογίσουμε το συνημίτονο των 60 μοιρών, ο παρακάτω τρόπος είναι λάθος

```
cos(60)
```

```
## [1] -0.952413
```

γιατί έτσι έχουμε πάρει ως αποτέλεσμα το συνημίτονο των 60 ακτινίων. Για να πετύχουμε το σκοπό μας θα πρέπει να κάνουμε τη μετατροπή:

$$rad = \frac{deg \times \pi}{180}$$

οπότε στην R θα γράψουμε:

```
cos(60*pi/180)
```

```
## [1] 0.5
```

1.1.4 Στρογγυλοποίηση

Πολλές φορές θα χρειαστούμε στρογγυλοποίηση των αποτελεσμάτων των υπολογισμών μας, είτε σε σταθερό πλήθος δεκαδικών ψηφίων, είτε σε δεκάδες, εκατοντάδες, κλπ. Ο τρόπος που το πετυχαίνουμε αυτά είναι με τη συνάρτηση round. Στο σημείο αυτό θα εξετάσουμε ένα απλό παράδειγμα το αποτέλεσμα της διαίρεσης 1500/7:

```
1500/7
```

```
## [1] 214.2857
```

Αν θέλαμε στρογγυλοποίηση με ένα δεκαδικό ψηφίο θα γράφαμε:

```
round(1500/7, 1)
```

```
## [1] 214.3
```

Ενώ με δύο δεκαδικά ψηφία θα γράφαμε:

```
round(1500/7, 2)
```

```
## [1] 214.29
```

Τέλος μπορεί να επιθυμούμε στρογγυλοποίηση σε ακέραιο, χωρίς δεκαδικά ψηφία:

```
round(1500/7, 0)
```

```
## [1] 214
```

Το ίδιο αποτέλεσμα μπορούμε επίσης να πετύχουμε και ως:

```
as.integer(1500/7)
```

```
## [1] 214
```

Αν χρειαστούμε στρογγυλοποίηση σε δεκάδες, τότε θα γράψουμε:

```
round(1500/7, -1)
```

```
## [1] 210
```

Δηλαδή θα δώσουμε ως δεύτερο όρισμα αρνητικό ακέραιο αριθμό. Ενώ αν χρειαστούμε στρογγυλοποίηση σε εκατοντάδες θα γράψουμε:

```
round(1500/7, -2)
```

```
## [1] 200
```

Γενικά, ο κανόνας που ισχύει για τη συνάρτηση round στην R (ακριβώς όπως και αλλού, Excel, Python, Matlab, κα) είναι πως η στρογγυολοποίηση γίνεται στο ψηφίο

$$10^{-k}$$

Για παράδειγμα αν k=1, τότε:

```
k <- 1
10 ^ -k

## [1] 0.1
```

Δηλαδή η στρογγυλοποίηση γίνεται σε πολλαπλάσια του 0.1, στο πρώτο δεκαδικό ψηφίο. Αντίστοιχα, αν k = -3:

```
k <- -3
10 ^ -k

## [1] 1000
```

Η στρογγυλοποίηση θα γίνει σε πολλαπλάσια του 1000, δηλαδή σε χιλιάδες, Για παράδειγμα:

```
round(1500/7, -3)

## [1] 0
```

Το αποτέλεσμα είναι 0, τι σημαίνει αυτό; Πολύ απλά πως η πλησιέστερη χιλιάδα είναι η μηδενική. Αν θυμηθούμε το αποτέλεσμα:

```
1500/7

## [1] 214.2857
```

Είναι εύκολο να καταλάβουμε πως ανάμεσα στο 0 και το 1000 το 214.2857 είναι περισσότερο κοντά στο 0.

Άλλες φορές πάλι είναι η επιθυμητή η στρογγυλοποίηση είναι προς τα πάνω (ceil, ταβάνι) είτε προς τα κάτω (floor, πάτωμα). Να πως μπορούμε να πετύχουμε κάτι τέτοιο:

```
1500/7

## [1] 214.2857

ceiling(1500/7)

## [1] 215

floor(1500/7)

## [1] 214
```

1.1.5 Ακέραια διαίρεση

Ένα ιδιαίτερο θέμα, σε όλα τα περιβάλλοντα προγραμματισμού, είναι η ακέραια διαίρεση. Η διαίρεση δηλαδή στην οποία το ενδιαφέρον είναι στο

ακέραιο μέρος το αποτελέσματος. Για παράδειγμα, η διαίρεση 17/7 έχει ως αποτέλεσμα:

```
17 / 7
```

```
## [1] 2.428571
```

Αν θέλαμε να έχουμε ως αποτέλεσμα τον αριθμό 2, χωρίς το δεκαδικό μέρος, τότε θα έπρεπε να γράψουμε τη διαίρεση ως εξής:

```
17 %/% 7
```

```
## [1] 2
```

Άλλοτε πάλι είναι επιθυμητό να λάβουμε το υπόλοιπο της ακέραιας διαίρεσης. Αυτό μπορούμε να το πετύχουμε με:

```
17 %% 7
```

```
## [1] 3
```

Το αποτέλεσμα είναι το 3, επειδή ισχύει

$$17 = 7 \times 2 + 3$$

```
# 17 = 7*2 + 3
```

Ακέραια διαίρεση στην R μπορούμε να έχουμε και με δεκαδικούς αριθμούς. Για παράδειγμα $\frac{10}{2.5}$:

```
10 / 2.5
```

```
## [1] 4
```

```
10 %/% 2.5
```

```
## [1] 4
```

Και στις δύο περιπτώσεις έχουμε ως αποτέλεσμα το 4, Ενώ στη διαίρεση $\frac{11}{2.5}$:

```
11 / 2.5
```

```
## [1] 4.4
```

```
11 %/% 2.5
```

```
## [1] 4
```

Το αποτέλεσμα διαφοροποιείται, στην απλή διαίρεση το αποτέλεσμα είναι το 4.4, στην ακέραια διαίρεση το αποτέλεσμα είναι το 4.

Όπως και σε πολλά άλλα υπολογιστικά συστήματα έτσι και στην R οι πράξεις με δεκαδικούς αριθμούς δεν είναι απόλυτα ακριβείς. Η ακρίβεια τους εξαρτάται τόσο από το σύστημα Η/Υ που χρησιμοποιούμε, όσο και από τον τρόπο αναπαράστασης αριθμών κινητής υποδιαστολής. Για παράδειγμα:

```
1 %/% 0.2
```

```
## [1] 4
```

Θα περιμέναμε 5 αλλά το αποτέλεσμα είναι 4. Παρόμοια:

```
2 %/% 0.2
```

```
## [1] 9
```

Δε συμβαίνει σε όλες τις περιπτώσεις περιπτώσεις, Για παράδειγμα:

```
3 %/% 0.3
```

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

Συμβαίνει όμως σε δυνάμεις του 2 διαιρεμένες με δυνάμεις του 10:

```
40 %/% 0.4
```

```
## [1] 99
```

Σημειώνουμε πως $0.4 = 2^2/10$ ή για παράδειγμα:

```
800 %/% 0.08
```

```
## [1] 9999
```

όπου $0.08 = 2^3/10$ Χρειάζεται ιδιαίτερη προσοχή αν τύχει να δουλεύουμε με ακέραια διαίρεση.

1.1.6 Λογικές τιμές

Η R έχει επίσης δύο λογικές τιμές για την αναπαράσταση των αληθών και ψευδών παραστάσεων, το TRUE και το FALSE. Οι λογικές αυτές τιμές αντιστοιχούν στις ακέραιες τιμές 1 και 0 αντίστοιχα. Για παράδειγμα, μπορούμε να προσθέσουμε μία τιμή :

```
TRUE + 1
```

```
## [1] 2
```

```
FALSE + 1
```

```
## [1] 1
```

το αποτέλεσμα είναι 2 και 1 αντίστοιχα επειδή $1+1=2$ και $0+1=1$. Επίσης μπορούμε να πολλαπλασιάσουμε με TRUE και FALSE ακριβώς όπως μπορούμε να πολλαπλασιάσουμε με 1 και 0:

```
5 * TRUE
```

```
## [1] 5
```

```
5 * FALSE
```

```
## [1] 0
```

1.1.7 Λογικοί έλεγχοι και παραστάσεις σύγκρισης

Ο πλέον συνηθισμένος λόγος χρήσης λογικών τιμών είναι ως αποτέλεσμα λογικών ελέγχων. Για παράδειγμα, είναι το 5 μεγαλύτερο του 0;

```
5 > 0
```

```
## [1] TRUE
```

Είναι, οπότε το αποτέλεσμα του λογικού ελέγχου είναι TRUE. Είναι το 5 μικρότερο του 0;

```
5 < 0
```

```
## [1] FALSE
```

Όχι, οπότε το αποτέλεσμα του λογικού ελέγχου είναι FALSE.

Τέτοιες συγκρίσεις μπορούν φυσικά να γίνουν ανάμεσα σε μεταβλητές και σταθερές. Για παράδειγμα:

```
x <- 5
x > 0
```

```
## [1] TRUE
```

```
x < 0
```

```
## [1] FALSE
```

ή ακόμα ανάμεσα σε δύο μεταβλητές:

```
x <- 5
a <- 0
x > a
```

```
## [1] TRUE
```

```
x < a
```

```
## [1] FALSE
```

Με το σύμβολο της ανισότητας μπορούμε να συνδυάσουμε και την ισότητα, πχ μεγαλύτερο ή ίσο

```
5 >= 5
```

```
## [1] TRUE
```

ή μικρότερο ή ίσο:

```
5 <= 5
```

```
## [1] TRUE
```

Τέλος αν θέλουμε να ελέγξουμε ώς προς την ισότητα θα γράψουμε:

```
5 == 5
```

```
## [1] TRUE
```

Προσοχή στο διπλό σύμβολο ίσον. Είναι πολύ συχνό λάθος να το παραλείπουμε, κάτι που οδηγεί σε τελείως άλλο αποτέλεσμα. Για παράδειγμα αυτό είναι σωστό:

```
x <- 5
x == 4
```

```
## [1] FALSE
```

ενώ αυτό είναι λάθος:

```
x <- 5
x = 4
```

Εδώ η παράσταση χ=4 σημαίνει την εκχώρηση της τιμής 4 στη μεταβλητή x, κάτι που οδηγεί στην αντικατάσταση της προηγούμενης τιμής. Έτσι αντί για λογικό έλεγχο έχουμε κάνει δήλωση εκχώρησης τιμής.

1.1.8 Ημερομηνίες και ημερολογιακά δεδομένα

Τα ημερολογιακά δεδομένα είναι από μόνα τους ένα ολόκληρο κεφάλαιο. Εδώ θα δούμε μια πολύ σύντομη εισαγωγή για τη χρήση τους. Οι ημερομηνίες στην R δηλώνονται με την ISO μορφή: Έτος-Μήνας-Ημέρα και με χρήση της συνάρτησης as.Date(). Πχ

```
as.Date("2021-10-04")
```

```
## [1] "2021-10-04"
```

Για να δηλώσουμε την ημερομηνία 4/10/2021. Ας θυμηθούμε πως διαφορετικά κράτη και οργανισμοί χρησιμοποιούν διαφορετικά σύμβολα, πχ στη Γερμανία θα γράφαμε 4.10.2021 και στις ΗΠΑ 10/04/2021. Υπάρχει πολύ μεγάλη ποικιλία για το πως μπορεί να γραφεί μια ημερομηνία. Αργότερα θα μάθουμε τρόπους να μετατρέπουμε τη μία γραφή στην άλλη, καλό είναι ωστόσο να θυμόμαστε πως ο σωστός τρόπος να γραφεί μια ημερομηνία είναι ο ISO.

Ας δούμε το πλεονέκτημα να χρησιμοποιούμε σωστά μια ημερομηνία με τη συνάρτηση as.Date σε σχέση με ένα ισοδύναμο λεκτικό. Πχ,

```
d <- as.Date("2021-10-04")
f <- "2021-10-04"
```

Στις μεταβλητές d και f έχουν εκχωρηθεί τιμές που αντιστοιχούν σε μια συγκεκριμένη ημερομηνία. Ωστόσο με την μεταβλητή d μπορούμε να εκτελέσουμε πράξεις, πχ να υπολογίσουμε ποια είναι η επόμενη ημερομηνία:

```
d + 1
```

```
## [1] "2021-10-05"
```

ή να υπολογίσουμε ποια ήταν η ημερομηνία 45 ημέρες πριν:

```
d - 45
```

```
## [1] "2021-08-20"
```

Αντίθετα, κάτι τέτοιο δεν είναι δυνατό με την μεταβλητή f:

```
f + 1
```

```
## Error in f + 1: non-numeric argument to binary operator
```

Αντίστοιχα αν έχουμε δύο ημερομηνίες μπορούμε εύκολα να υπολογίσουμε τη διαφορά τους:

```
d1 <- as.Date("2021-06-21")
d2 <- as.Date("2021-10-04")
d2 - d1
```

```
## Time difference of 105 days
```

Μια άλλη χρήσιμη συνάρτηση ημερομηνίας είναι η τρέχουσα ημερομηνία, δηλαδή η ημερομηνία που δείχνει το ρολόι του υπολογιστή στον οποίο εργαζόμαστε:

```
Sys.Date()
```

```
## [1] "2022-11-01"
```

1.1.9 Μετατροπή τύπου δεδομένων

Είναι δυνατό να μετατρέψουμε ένα τύπο δεδομένων σε άλλο. Πχ αριθμό σε κείμενο, κείμενο σε αριθμό, κείμενο σε ημερομηνία, ημερομηνία σε αριθμό, κλπ. Όλες αυτές οι διεργασίες γίνονται με συναρτήσεις που ξεκινούν με as. όπως ακριβώς η as.Date που είδαμε παραπάνω.

Για παράδειγμα, το λεκτικό 10 μπορεί να μετατραπεί στον αριθμό 10:

```
x <- "10"
as.numeric(10)
```

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

Επίσης ένας αριθμός μπορεί να μετατραπεί σε κείμενο με τη συνάρτηση as.character:

```
as.character(3.14)
```

```
## [1] "3.14"
```

Το λεκτικό “2021-10-04” μπορεί να μετατραπεί στην ημερομηνία 4 Οκτωβρίου 2021:

```
d <- "2021-10-04"
as.Date(d)
```

```
## [1] "2021-10-04"
```

ή μια ημερομηνία μπορεί να μετατραπεί σε ακέραιο αριθμό:

```
as.integer(d)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA
```

Τι σημαίνει αυτό; Τι σημαίνει ο αριθμός 18904 για την ημερομηνία 4 Οκτωβρίου 2021; Αυτό έχει να κάνει με τον τρόπο που οι υπολογιστές αποθηκεύουν και χειρίζονται τις ημερομηνίες, δεν είναι κάτι ιδιαίτερο της R. Κάθε ημερομηνία αποθηκεύεται ως ακέραιος αριθμός, ξεκινώντας από την πρωτοχρονιά του 1970:

```
as.integer(as.Date("1970-01-01"))
```

```
## [1] 0
```

Από εκεί και μετά για κάθε μέρα που περνάει προσθέτουμε τη μονάδα. Έτσι Από τις 1/1/1970 έως και 4/10/2021 έχουν περάσει 18904 ημέρες. Μπορούμε να το διαπιστώσουμε και ως εξής:

```
as.integer(as.Date("2021-10-04")) - as.integer(as.Date("1970-01-01"))
```

```
## [1] 18904
```

Μερικές φορές θα χρειαστεί να ελέγξουμε αν μια μεταβλητή έχει ένα ορισμένο τύπο δεδομένων, πχ ακέραιο, αριθμό, ημερομηνία, κείμενο, κα. Αυτό μπορούμε να το κάνουμε με συναρτήσεις που αρχίζουν με το is. και αποδίδουν TRUE ή FALSE. Για παράδειγμα:

```
x <- 10
is.numeric(x)
```

```
## [1] TRUE
```

```
is.character(x)
```

```
## [1] FALSE
```

Μπορούμε επίσης να ελέγξουμε αν ένας αριθμός είναι ακέραιος ή όχι, για παράδειγμα:

```
is.integer(x)
```

```
## [1] FALSE
```

Ίσως αυτό δεν το περιμέναμε. Τι συμβαίνει, δεν είναι ακέραιος αριθμός το 10; Στην R όχι, δεν είναι. Όλοι οι αριθμοί καταχωρούνται αυτόματα ως δεκαδικοί. Γι' αυτό και ο έλεγχος is.integer αποδίδει FALSE. Αν ωστόσο χρειαζόμαστε ακέραιο αριθμό μπορούμε να κάνουμε την μετατροπή:

```
x <- as.integer(x)
```

Και να το διαπιστώσουμε:

```
is.integer(x)
```

```
## [1] TRUE
```

Φυσικά, η μεταβλητή x συνεχίζει να είναι αριθμητική μεταβλητή:

```
is.numeric(x)
```

```
## [1] TRUE
```

Επομένως: μια ακέραια μεταβλητή είναι πάντοτε και αριθμητική, ωστόσο μια αριθμητική μεταβλητή μπορεί να είναι ή να μην είναι ακέραια.

1.2 Διανυσματικές μεταβλητές

Στην προηγούμενη ενότητα εξετάσαμε την απλή περίπτωση όπου μια μεταβλητή έχει μία τιμή. Είναι ωστόσο δυνατό μια μεταβλητή να έχει περισσότερες από μία τιμές, να είναι δηλαδή διανυσματική. Κάτι τέτοιο στην R γίνεται με τη συνάρτηση c. Για παράδειγμα, η μεταβλητή x έχει 5 τιμές:

```
x <- c(1, 2, 3, 4, 5)
```

Από άλλες γλώσσες προγραμματισμού ίσως να γνωρίζουμε πως απαιτούνται ειδικές διαδικασίες για την εκχώρηση τιμών σε διανυσματικές μεταβλητές. Στην R κάτι τέτοιο δεν χρειάζεται. Μόλις έχουμε ορίσει ένα διάνυσμα πέντε θέσεων και έχουμε εκχωρήσει τις αντίστοιχες τιμές.

Για τη μεταβλητή x μπορούμε τώρα να υπολογίσουμε ποσότητες όπως είναι το πλήθος των στοιχείων, δηλαδή το μήκος του διανύσματος:

```
length(x)
```

```
## [1] 5
```

και εφόσον το διάνυσμα έχει αριθμητικές τιμές, μπορούμε επίσης να υπολογίσουμε: το άθροισμα των τιμών $ _{i=1}^{N} x_i$

```
sum(x)
```

```
## [1] 15
```

τη μέση τιμή

$$ \mu = \frac{1}{N} \sum_{i=1}^{N} x_i $$

```
mean(x)
```

```
## [1] 3
```

τη διακύμανση

$$ Var(X) = \sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2 $$

```
var(x)
```

```
## [1] 2.5
```

την τυπική απόκλιση

$$ \sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2 } $$

```
sd(x)
```

```
## [1] 1.581139
```

Όλες οι παραπάνω συναρτήσεις δέχονται ως όρισμα ένα διάνυσμα με πολλές τιμές και επιστρέφουν μία τιμή, πχ τη διακύμανση. Αν ωστόσο εφαρμόσουμε συναρτήσεις που δέχονται μία τιμή και επιστρέφουν μία άλλη, για παράδειγμα η τετραγωνική ρίζα, τότε θα πάρουμε ως αποτέλεσμα ένα διάνυσμα τιμών. Για παράδειγμα:

```
sqrt(x)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

Εδώ έχουμε ένα διάνυσμα πέντε τιμών, οπότε με την εφαρμογή της τετραγωνικής ρίζας θα πάρουμε ως απάντηση ένα διάνυσμα πέντε τιμών επίσης. Το ίδιο θα συμβεί αν εφαρμόσουμε οποιαδήποτε άλλη αλγεβρική παράσταση, για παράδειγμα πρόσθεση αριθμητικής σταθεράς σε διάνυσμα:

```
x + 1
```

```
## [1] 2 3 4 5 6
```

πολλαπλασιασμός αριθμητικής σταθεράς με διάνυσμα:

```
2*x
```

```
## [1]  2  4  6  8 10
```

ή συνδυασμός:

```
2*x + 1
```

```
## [1]  3  5  7  9 11
```

Ακόμα μπορούμε να υψώσουμε σε δύναμη όλα τα στοιχεία, για παράδειγμα:

```
x**2
```

```
## [1]  1  4  9 16 25
```

όπου το διπλό σύμβολο του αστερίσκου είναι ο τελεστής ύψωσης σε δύναμη, το ίδιο με το σύμβολο ^ (SHIFT+6):

```
x ^ 2
```

```
## [1]  1  4  9 16 25
```

Αν έχουμε δύο διανύσματα με ίδιο μήκος (ίδιο πλήθος στοιχείων) τότε μπορούμε να κάνουμε οποιαδήποτε αλγεβρική πράξη μεταξύ τους, για παράδειγμα αν:

```
x <- c(1, 2, 3, 4, 5)
y <- c(5, 8, 6, 1, 2)
```

τότε είναι εφικτές όλες οι παρακάτω αλγεβρικές παραστάσεις:

```
x + y
```

```
## [1]  6 10  9  5  7
```

```
x - y
```

```
## [1] -4 -6 -3  3  3
```

```
x * y
```

```
## [1]  5 16 18  4 10
```

```
x / y
```

```
## [1] 0.20 0.25 0.50 4.00 2.50
```

```
x ^ y
```

```
## [1]   1 256 729   4  25
```

για τις οποίες πρέπει να σημειώσουμε πως ακολουθούν τον κανόνα στοιχείο προς στοιχείο.

1.2.1 Στοιχεία ενός διανύσματος

Ας υποθέσουμε πως έχουμε το παρακάτω διάνυσμα τιμών:

```
x <- c(4, -1, 3)
```

το οποίο έχει μήκος:

```
length(x)
```

```
## [1] 3
```

Για να πάρουμε την τιμή της πρώτης θέσης του διανύσματος μπορούμε να γράψουμε:

```
x[1]
```

```
## [1] 4
```

ενώ για να πάρουμε την τιμή της τρίτης θέσης (τελευταίας) θα γράψουμε:

```
x[3]
```

```
## [1] 3
```

Στο συγκεκριμένο παράδειγμα έχουμε ένα πολύ μικρό σε μήκος διάνυσμα και είναι προφανώς το μήκος του. Σε άλλες περιπτώσεις ενδέχεται κάτι τέτοιο να μην είναι τόσο προφανές. Έτσι, ο πλέον ενδεδειγμένος τρόπος να δούμε το τελευταίο στοιχείο ενός διανύσματος είναι να εκμεταλλευτούμε τη συνάρτηση length:

```
x[length(x)]
```

```
## [1] 3
```

δηλαδή να υπολογίσουμε το μήκος (ακέραιος αριθμός) και να το εισάγουμε ως θέση του διανύσματος.

Αν χρειαστεί να προσθέσουμε ένα στοιχείο στο τέλος του διανύσματος, για παράδειγμα την τιμή 2 ως τέταρτο στοιχείο, τότε μπορούμε να γράψουμε:

```
x[4] <- 2
```

Παρόμοια με την προηγούμενη παρατήρηση σχετικά με το μήκος του διανύσματος και σε αυτή την περίπτωση μπορούμε να εκμεταλλευτούμε τη συνάρτηση length. Η επόμενη από το τέλος θέση του διανύσματος μπορεί να υπολογιστεί ως:

```
length(x)+1
```

```
## [1] 4
```

Επομένως, αν θέλουμε να προσθέσουμε μία ακόμα τιμή μπορούμε ορθότερα να γράψουμε:

```
x[length(x)+1] <- 2
```

η οποία θα προσθέσει μία τιμή οποιοδήποτε και αν είναι το μήκος του. Για παράδειγμα αυτή τη στιγμή το διάνυσμα έχει μήκος:

```
length(x)
```

```
## [1] 4
```

Αν προσθέσουμε μία πλέον τιμή στο τέλος (πχ το -5) δεν χρειάζεται παρά να γράψουμε ακριβώς το ίδιο:

```
x[length(x)+1] <- -5
```

Και έτσι έχουμε τώρα το διάνυσμα:

```
x
```

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

Είναι δυνατό να λάβουμε περισσότερες από μία τιμές ενός διανύσματος. Για παράδειγμα τις 2 πρώτες:

```
x[1:2]
```

```
## [1]  4 -1
```

η τις θέσεις 2 και 3:

```
x[2:3]
```

```
## [1] -1  3
```

Εδώ έχουμε εκμεταλλευτεί τον τελεστή αλληλουχίας, την άνω και κάτω τελεία. Για παράδειγμα η έκφραση 1:2 δίνει την αλληλουχία ακέραιων αριθμών που ξεκινά από το ένα, μεταβάλλεται κατά μία μονάδα και φτάνει έως το δύο. Αντίστοιχα, η παράσταση 3:7 θα αποδώσει το διάνυσμα αριθμών:

```
2:7
```

```
## [1] 2 3 4 5 6 7
```

Ένα άλλο θέμα που θα εξετάσουμε είναι να βρούμε τις τελευταίες θέσεις ενός διανύσματος, για παράδειγμα τις δύο τελευταίες θέσεις. Αν θυμηθούμε πως η τελευταία θέση ενός διανύσματος είναι η

```
length(x)
```

```
## [1] 5
```

τότε η προτελευταία είναι η

```
length(x) - 1
```

```
## [1] 4
```

επομένως στη περίπτωση του διανύσματος x οι δύο τελευταίες θέσεις είναι:

```
(length(x)-1):length(x)
```

```
## [1] 4 5
```

Προσοχή, οι παρενθέσεις στην παράσταση (length(x)-1) είναι απαραίτητες. Χωρίς τις παρενθέσεις θα παίρναμε ένα τελείως διαφορετικό αποτέλεσμα:

```
length(x) - 1:length(x)
```

```
## [1] 4 3 2 1 0
```

που αντιστοιχεί στην αφαίρεση των αριθμών 1 έως 5 από το 5, οι παρενθέσεις αλλάζουν την προτεραιότητα των πράξεων: πρώτα η αφαίρεση και μετά η αλληλουχία τιμών. Οι τιμές που αντιστοιχούν σε αυτές τις θέσεις είναι:

```
x[(length(x)-1):length(x)]
```

```
## [1]  2 -5
```

Σταδιακά παρατηρούμε πως οι υπολογισμοί γίνονται όλα και περισσότερο πολύπλοκοι. Για τη διευκόλυνση μας, την ταχύτητα ανάπτυξης κώδικα και την ορθότητα των υπολογισμών η R έχει δύο ενσωματωμένες συναρτήσεις για την εύρεση των αρχικών και τελικών τιμών διανυσμάτων (και πινάκων δεδομένων όπως θα δούμε αργότερα), τις συναρτήσεις head (τιμές από την αρχή) και tail (τιμές από το τέλος). Για παράδειγμα, οι δύο πρώτες τιμές του διανύσματος:

```
head(x, 2)
```

```
## [1]  4 -1
```

και οι δύο τελευταίες τιμές:

```
tail(x, 2)
```

```
## [1]  2 -5
```

Φυσικά ο αριθμός 2 (πλήθος τιμών) μπορεί να αντικατασταθεί με άλλη επιθυμητή τιμή, για παράδειγμα οι 3 πρώτες τιμές του διανύσματος:

```
head(x, 3)
```

```
## [1]  4 -1  3
```

Μέχρι τώρα εξετάσαμε περιπτώσεις όπου οι τιμές που αναζητούσαμε ήταν σε συνεχείς (διπλανές) θέσεις. Αν χρειαστεί να επιλέξουμε τιμές που δεν είναι σε διπλανές θέσεις θα πρέπει να εισάγουμε ως διάνυσμα τις θέσεις. Για

παράδειγμα, για να βρούμε τις τιμές του διανύσματος στις 1 και 4 θα γράψουμε:

```
x[c(1, 4)]
```

```
## [1] 4 2
```

ή για να βρούμε τις τιμές στις θέσεις 1, 2 και 4 θα γράψουμε:

```
x[c(1, 2, 4)]
```

```
## [1]  4 -1  2
```

1.3 Λίστες

Ενώ τα διανύσματα είναι αντικείμενα που περιέχουν ομοειδεί δεδομένα, οι λίστες μπορεί να περιέχουν ετερογενή δεδομένα, για παράδειγμα μίξη αριθμητικών και λεκτικών δεδομένων. Αυτό δεν είναι υποχρεωτικό, αλλά μπορεί κάλλιστα να συμβεί.

Ας δούμε ένα παράδειγμα μιας λίστας η οποία περιέχει μόνο αριθμητικά στοιχεία, πολύ παρόμοια σαν ένα διάνυσμα:

```
list(4, -1, 3)
```

```
## [[1]]
## [1] 4
##
## [[2]]
## [1] -1
##
## [[3]]
## [1] 3
```

Παρατηρούμε πως η συνάρτηση που δημιουργεί τη λίστα είναι η list. Αν θέλουμε μπορούμε να αποθηκεύσουμε της τιμές σε μεταβλητή:

```
x <- list(4, -1, 3)
```

Αυτή η συγκεκριμένη λίστα είναι πολύ παρόμοια με ένα διάνυσμα:

```
is.list(x)
```

```
## [1] TRUE
```

```
is.vector(x)
```

```
## [1] TRUE
```

Για παράδειγμα μπορούμε να υπολογίσουμε το μήκος της

```
length(x)
```

```
## [1] 3
```

Αλλά δεν μπορούμε να υπολογίσουμε το άθροισμα των στοιχείων της:

```
sum(x)

## Error in sum(x): invalid 'type' (list) of argument
```

ούτε μπορούμε να εκτελέσουμε αλγεβρικές πράξεις, για παράδειγμα:

```
x + 1

## Error in x + 1: non-numeric argument to binary operator
```

Μπορούμε να έχουμε πρόσβαση στα στοιχεία μιας λίστα ως εξής, για παράδειγμα το πρώτο στοιχείο μια λίστας:

```
x[1]

## [[1]]
## [1] 4
```

Παρατηρούμε πως είναι επίσης λίστα, αυτό είναι φανερό από τις διπλές αγκύλες. Αν θέλαμε να πάρουμε την κυριολεκτική τιμή θα γράφαμε:

```
x[[1]]

## [1] 4
```

οπότε τώρα έχουμε την τιμή 4, με την οποία μπορούμε να εκτελέσουμε αλγεβρικές πράξεις, για παράδειγμα:

```
x[[1]] + 1

## [1] 5
```

Τα στοιχεία μιας λίστας μπορεί να είναι επώνυμα:

```
x <- list(a = 4, b = -1, c = 3)
```

οπότε η πρόσβαση σε αυτά μπορεί να γίνεται με τη χρήση του συμβόλου $, για παράδειγμα:

```
x$a

## [1] 4

x$b

## [1] -1

x$c

## [1] 3
```

Μία λίστα μπορεί να περιέχει ετερογενή δεδομένα, σε αντίθεση με ένα διάνυσμα. Το παρακάτω παράδειγμα αναφέρεται σε μία λίστα με 3 στοιχεία:

```
x <- list(a = c(1, 2, 3), b = 5, c = "Athens")
```

Το πρώτο στοιχείο (a) είναι ένα διάνυσμα:

```
x$a
```

```
## [1] 1 2 3
```

Το δεύτερο στοιχείο (b) της λίστας είναι ένας αριθμός:

```
x$b
```

```
## [1] 5
```

Ενώ το τρίτο στοιχείο (c) της λίστας είναι ένα λεκτικό:

```
x$c
```

```
## [1] "Athens"
```

Ας εξετάσουμε κάποια επιπλέον τρόπους πρόσβασης στα στοιχεία μιας λίστας. Το πλήθος των στοιχείων μιας λίστας μπορεί να βρεθεί ως εξής:

```
length(x)
```

```
## [1] 3
```

Το πρώτο στοιχείο μπορεί να βρεθεί ως:

```
x[1]
```

```
## $a
## [1] 1 2 3
```

Αξίζει να προσέξουμε η επιστρεφόμενη τιμή είναι επίσης μια λίστα. Αν θέλουμε να απαλλαγούμε από τη δομή της λίστας και να πάρουμε αυτό που περιέχεται σε αυτή τη θέση της λίστας θα πρέπει να γράψουμε:

```
x[[1]]
```

```
## [1] 1 2 3
```

Δηλαδή να χρησιμοποιήσουμε διπλές αγκύλες. Το ίδιο αποτέλεσμα μπορούμε να πετύχουμε με την συνάρτηση unlist():

```
unlist(x[1])
```

```
## a1 a2 a3
##  1  2  3
```

Εδώ πήραμε ως αποτέλεσμα ένα επώνυμο διάνυσμα, a1, a2, a3, επειδή το πρώτο στοιχείο είχε το όνομα a (χ$α). Αν δεν θέλουμε τα ονόματα τότε θα πρέπει να δώσουμε την επιλογή:

```r
unlist(x[1], use.names = FALSE)
```

```
## [1] 1 2 3
```

Η χρήση ή όχι ονομάτων στα στοιχεία ενός διανύσματος δεν επηρεάζει σε τίποτα τις πράξεις με διανύσματα. Επίσης, δεν είναι υποχρεωτική. Χρησιμοποιούμε επώνυμες τιμές στα διανύσματα όταν θέλουμε να αποθηκεύσουμε κάποια επιπλέον πληροφορία για τις τιμές, ή όταν θέλουμε να έχουμε πρόσβαση στα στοιχεία του διανύσματος με βάση το όνομα.

Ας επιστρέψουμε στη λίστα x. Η πρόσβαση στο στοιχείο 0 μιας λίστας δεν είναι δυνατή, θα επιστρέψει μήνυμα σφάλματος:

```r
x[[0]]
```

```
## Error in x[[0]]: attempt to select less than one element in
get1index <real>
```

Το ίδιο θα συμβεί αν ζητήσουμε πρόσβαση στο τέταρτο στοιχείο, το οποίο βέβαια δεν υπάρχει, η λίστα έχει μόνο 3 στοιχεία:

```r
x[[4]]
```

```
## Error in x[[4]]: subscript out of bounds
```

Για να βρούμε το τελευταίο στοιχείο μιας λίστας μπορούμε να γράψουμε:

```r
x[length(x)]
```

```
## $c
## [1] "Athens"
```

ή, αν δεν θέλουμε λίστα ως επιστρεφόμενη τιμή:

```r
x[[length(x)]]
```

```
## [1] "Athens"
```

Αν ωστόσο ζητήσουμε ένα στοιχείο μιας λίστας με όνομά του, ενώ αυτό το στοιχείο δεν υπάρχει, για παράδειγμα:

```r
x$d
```

```
## NULL
```

θα πάρουμε την τιμή NULL ως ως επιστρεφόμενη τιμή.

Σε αρκετές περιπτώσεις μπορεί να έχουμε περιπτώσεις με εμφωλευμένες λίστες, δηλαδή ένα στοιχείο μια λίστας να είναι επίση λίστα. Ας δούμε το παρακάτω παράδειγμα:

```r
x <- list(x1 = 5, x2 = 4, x3 = 10)
y <- list(y1 = "Athens", y2 = "Rome")
a <- list(num = x, city = y)
```

Η μεταβλητή a είναι λίστα με δύο στοιχεία τα οποία είναι επίσης λίστες. Σε αυτή την περίπτωση χρειάζεται ιδιαίτερη προσοχή αν θέλουμε να έχουμε επιμέρους πρόσβαση στα στοιχεία της *λίστας μέσα στη λίστα*. Ας δούμε για παράδειγμα την τιμή 10 και ας εξετάσουμε τη θέση της.

Στην τρίτη θέσητου πρώτου στοιχείου της λίστας a:

```r
a[[1]][[3]]
```

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

Στην τρίτη θέση της εσωτερικής λίστας num:

```r
a$num[[3]]
```

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

Στη θέση x3 του πρώτου στοιχείου της λίστας a:

```r
a[[1]]$x3
```

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

Στη θέση x3 της εσωτερικής λίστας num:

```r
a$num$x3
```

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

Και οι τέσσερις παραπάνω τρόποι είναι απολύτως ισοδύναμοι μεταξύ τους, δίνουν ακριβώς το ίδιο αποτέλεσμα. Ποιον τρόπο θα επιλέξουμε εξαρτάται από τις συνθήκες του προβλήματος και από τις επιλογές μας.

1.4 Πίνακες δεδομένων data frames

1.5 Μήτρες

Οι διατάξεις (arrays) είναι διανυσματικές δομές δεδομένων. Μπορεί να είναι 1, 2 ή περισσότερων διαστάσεων. Οι διαστάσεις καθορίζονται από τις ανάγκες του χρήστη. Συνήθως δεν χρησιμοποιούμε διατάξεις με περισσότερες από 2 διαστάσεις. Υπάρχουν ωστόσο περιπτώσεις όπου μπορούμε να έχουμε πολλές διαστάσεις.

Μια διάταξη γραμμή τριών στοιχείων, για παράδειγμα:

```r
array(0, dim = c(1, 3))
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
```

Μια διάταξη στήλη τριών στοιχείων, για παράδειγμα:

```
array(0, dim = c(3, 1))
```

```
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
```

Μια διάταξη δύο διαστάσεων, με 3 γραμμές και 3 στήλες:

```
array(0, dim = c(3, 3))
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0
```

Στα παραδείγματα αυτά τα στοιχεία της διάταξης είναι παντού μηδέν, αυτό τα δίνουμε ως απλό παράδειγμα, δεν σημαίνει πως είναι πάντοτε μηδέν. Για παράδειγμα, μια διάταξη 3x3 με στοιχεία τυχαίους αριθμούς της ομοιόμορφης κατανομής (0, 1):

```
array(runif(9), dim = c(3, 3))
```

```
##           [,1]      [,2]       [,3]
## [1,] 0.3392495 0.3805874 0.19066675
## [2,] 0.4024695 0.9090835 0.17280578
## [3,] 0.2345134 0.4121510 0.06428593
```

Ή, μια διάταξη 2x4, με τους αριθμούς 1 έως 8:

```
array(1:8, dim = c(2, 4))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

Παρόμοια δομή με τις διατάξεις είναι η μήτρες (matrix). Οι μήτρες (matrix) της R αντιστοιχούν σε αυτό που γνωρίζουμε ως μήτρα από την άλγεβρα. Ο ορισμός τους είναι παρόμοιος με τις διατάξεις.
Για παράδειγμα μια μήτρα 1x3:

```
matrix(0, nrow = 1, ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
```

Ή μια μήτρα 3x1:

```
matrix(0, nrow = 3, ncol = 1)
```

```
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
```

Ή μια μήτρα 3x3:

```
matrix(0, nrow = 3, ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0
```

Στον ορισμό της μήτρας υπάρχει μια παράμετρος (byrow) η οποία καθορίζει τον τρόπο με τον οποίο συμπληρώνονται οι τιμές. Για παράδειγμα, στη μήτρα 2x4:

```
matrix(1:8, nrow = 2, ncol = 4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

Έχει εφαρμοστεί η προεπιλογή byrow=FALSE:

```
matrix(1:8, nrow = 2, ncol = 4, byrow = FALSE)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

Δηλαδή τα στοιχεία της μήτρας συμπληρώνονται κατά στήλη. Αν θέλουμε η συμπλήρωση να γίνει κατά γραμμές θα γράψουμε:

```
matrix(1:8, nrow = 2, ncol = 4, byrow = TRUE)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
```

Η πρόσβαση στα στοιχεία διάταξης ή μήτρας γίνεται με τους αριθμούς γραμμής και στήλης. Για παράδειγμα έστω η μήτρα:

```
m <- matrix(1:8, nrow = 2, ncol = 4)
m
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

Η πρώτη στήλη της μήτρας είναι:

```
m[, 1]
```

```
## [1] 1 2
```

Ενώ η δεύτερη γραμμή της μήτρας είναι:

```
m[2, ]
```

```
## [1] 2 4 6 8
```

Δηλαδή ο πρώτος αριθμός (πριν το κόμμα) δηλώνει την γραμμή και ο δεύτερος αριθμός (μετά το κόμμα) δηλώνει τον αριθμό της στήλης. Εάν πριν το κόμμα δεν υπάρχει αριθμός τότε λαμβάνονται όλες οι γραμμές, ενώ εάν δεν υπάρχει αριθμός μετά το κόμμα τότε λαμβάνονται όλες οι στήλες.

Μπορούμε να έχουμε πρόσβαση σε συγκεκριμένο στοιχείο μιας μήτρας, για παράδειγμα:

```
m[2, 2]
```

```
## [1] 4
```

Μπορούμε να ελέγξουμε αν μια μεταβλητή (έα αντικείμενο) είναι μήτρα ως εξής:

```
is.matrix(m)
```

```
## [1] TRUE
```

Ενώ με τη συνάρτηση as.matrix() μπορούμε να επιχειρήσουμε να μετατρέψουμε ένα αντικείμενο σε μήτρα:

```
as.matrix(m)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

Εδώ βέβαια το m είναι μήτρα, οπότε η απόπειρα είναι οπωσδήποτε επιτυχής.

Αν χρειαστεί να εξάγουμε τις διαστάσεις μιας διάταξης (array) ή μιας μήτρας μπορούμε να χρησιμοποιήσουμε τη συνάρτηση dim():

```
dim(m)
```

```
## [1] 2 4
```

η οποία θα επιστρέψει ένα διάνυσμα δύο τιμών με το πλήθος γραμμών και στηλών αντίστοιχα. Εναλλακτικά, μπορούμε να βρούμε το πλήθος γραμμών:

```
nrow(m)
```

```
## [1] 2
```

και το πλήθος στηλών:

```
ncol(m)
```

```
## [1] 4
```

1.6 Πλαίσια δεδομένων, data frames

Τα πλαίσια δεδομένων είναι η πλέον συνηθισμένη δομή δεδομένων στην R. Είναι σχεδόν αδύνατο να κάνει κανείς κάποια σταστιστική ανάλυση χωρίς να χρησιμοποιήσει πλαίσια δεδομένων.

Ένα πλαίσιο δεδομένων είναι στην ουσία μια λίστα από διανύσματα ίσου μεγέθους. Τα χαρακτηριστικά ενός πλαισίου δεδομένων μοιράζονται λοιπόν ανάμεσα στα χαρακτηριστικά της λίστας και του διανύσματος, Για να εξερευνήσουμε τις ιδιότητες τους θα χρησιμοποιήσουμε το ποιο γνωστό πλαίσιο δεδομένων της R, το mtcars:

```
mtcars
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear
carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4
4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4
4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4
1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3
1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3
2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3
1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3
4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4
2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4
2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4
4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4
4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3
3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3
3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3
3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3
```

```
4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3
4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3
4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4
1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4
2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4
1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3
1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3
2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3
2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3
4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3
2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4
1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5
2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5
2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5
4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5
6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5
8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4
2
```

Λεπτομέρειες για το οποίο μπορούμε να μάθουμε ως εξής:

```
?mtcars
```

Περιέχει πληροφορίες σχετικά με αυτοκίνητα (από το περιοδικό Motor trend USA, 1974), συγκριτικά στοιχεία για 32 αυτοκίνητα που κυκλοφορούσαν στην αγορά των ΗΠΑ το 1974. Τον τύπο του αντικειμένου mtcars μπορούμε να τον δούμε ως εξής:

```
typeof(mtcars)
```

```
## [1] "list"
```

ενώ την κλάση:

```r
class(mtcars)
```

```
## [1] "data.frame"
```

Όπως και σε άλλες περιπτώσεις μπορούμε να εξετάσουμε αν κάποιο αντικείμενο είναι data frame με τη συνάρτηση is.data.frame():

```r
is.data.frame(mtcars)
```

```
## [1] TRUE
```

Ως λίστα, το πλαίσιο δεδομένων έχει ονόματα στηλών:

```r
colnames(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"
"gear"
## [11] "carb"
```

και ονόματα γραμμών:

```r
rownames(mtcars)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"

##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"

##  [7] "Duster 360"          "Merc 240D"           "Merc 230"

## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"

## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac
Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"

## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"

## [22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"

## [25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"

## [28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"

## [31] "Maserati Bora"       "Volvo 142E"
```

Η συνάρτηση names() επιστρέψει επίσης τα ονόματα στηλών:

```r
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"
"gear"
## [11] "carb"
```

Είναι δηλαδή ισοδύναμη με τη συνάρτηση colnames(). Αξίζει να σημειωθεί πως ενώ τα ονόματα γραμμών είναι υποχρεωτικό να υπάρχουν τα ονόματα στηλών είναι προαιρετικά.

Ως πίνακας το πλαίσιο δεδομένων έχει διαστάσεις:

```
dim(mtcars)
```

```
## [1] 32 11
```

```
nrow(mtcars)
```

```
## [1] 32
```

```
ncol(mtcars)
```

```
## [1] 11
```

Η πρόσβαση στα στοιχεία ενός πλαισίου δεδομένων μπορεί να γίνει είτε αριθμητικά, είτε με ονόματα στηλών (λίστα) είτε συνδυαστικά. Για παράδειγμα, τα δεδομένα της στήλης *mpg* (miles per galon), μπορούν να εξαχθούν ως η πρώτη στήλη του πλαισίου δεδομένων:

```
mtcars[, 1]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4
17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0
30.4 15.8 19.7
## [31] 15.0 21.4
```

είτε ως

```
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4
17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0
30.4 15.8 19.7
## [31] 15.0 21.4
```

Οι δύο τρόποι είναι ισοδύναμοι μεταξύ τους και επιστρέφουν το ίδιο αποτέλεσμα.

Μπορούμε επίσης να εξάγουμε συγκεκριμένη γραμμή του πλαισίου δεδομένων, παράδειγμα τη δεύτερη γραμμή:

```
mtcars[2, ]
```

```
##               mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

ή τις γραμμές 1 έως 3:

```
mtcars[1:3, ]
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

ή τις γραμμές 1 και 3 (χωρίς τη γραμμή 2)

```
mtcars[c(1, 3), ]
```

```
##             mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21.0   6  160 110 3.90 2.62 16.46  0  1    4    4
## Datsun 710 22.8   4  108  93 3.85 2.32 18.61  1  1    4    1
```

Το ίδιο μπορούμε να κάνουμε και με τις στήλες, για παράδειγμα τις στήλες 3, 4, 6, και τις γραμμές 1 έως 5:

```
mtcars[1:5, c(3, 4, 6)]
```

```
##                   disp  hp    wt
## Mazda RX4          160 110 2.620
## Mazda RX4 Wag      160 110 2.875
## Datsun 710         108  93 2.320
## Hornet 4 Drive     258 110 3.215
## Hornet Sportabout  360 175 3.440
```

Βέβαια, για τις στήλες μπορούμε να χρησιμοποιήσουμε ονόματα:

```
mtcars[1:5, c("disp", "hp", "wt")]
```

```
##                   disp  hp    wt
## Mazda RX4          160 110 2.620
## Mazda RX4 Wag      160 110 2.875
## Datsun 710         108  93 2.320
## Hornet 4 Drive     258 110 3.215
## Hornet Sportabout  360 175 3.440
```

Είτε για τις γραμμές, είτε για τις στήλες μπορούμε να επιλέξουμε διαφορετική σειρά εμφάνισης από αυτή του πλαισίου δεδομένων:

```
mtcars[5:1, c("hp", "disp", "wt")]
```

```
##                    hp disp    wt
## Hornet Sportabout 175  360 3.440
## Hornet 4 Drive    110  258 3.215
## Datsun 710         93  108 2.320
## Mazda RX4 Wag     110  160 2.875
## Mazda RX4         110  160 2.620
```

Αξίζει να σημειωθεί πως όλες αυτές οι ενέργειες επιστρέφουν επίσης πλαίσιο δεδομένων, η επιλογή συγκεκριμένων γραμμών ή/και στηλών δεν αλάζει την κλάση του αντικειμένου, για παράδειγμα:

```r
x <- mtcars[5:1, c("hp", "disp", "wt")]
class(x)

## [1] "data.frame"
```

Μπορούμε να δημιουργήσουμε ένα πλαίσιο δεδομένων ως εξής:

```r
data.frame(x = c("a", "b", "c"), y = c(5, 7, 2))

##   x y
## 1 a 5
## 2 b 7
## 3 c 2
```

δηλαδή δίνοντας ως ορίσματα στη συνάρτηση data.frame() τα διανύσματα που αποτελούν τις στήλες του πλαισίου δεδομένων. Κάτι τέτοιο είναι δυνατό να γίνει και σταδιακά:

```r
x <- c("a", "b", "c")
y <- c(5, 7, 2)
data.frame(x, y)

##   x y
## 1 a 5
## 2 b 7
## 3 c 2
```

Μπορούμε φυσικά να αποθηκεύσουμε το αποτέλεσμα δημιουργίας του πλαισίου δεδομένων σε νέα μεταβλητή για τη χρησιμοποιήσουμε αργότερα:

```r
df <- data.frame(x, y)
```

Μπορούμε να επιβεβαίωσουμε την κλάση:

```r
class(df)

## [1] "data.frame"
```

και τη δομή του πλαισίου δεδομένων:

```r
str(df)

## 'data.frame':    3 obs. of  2 variables:
##  $ x: chr  "a" "b" "c"
##  $ y: num  5 7 2
```

Αν έχουμε ένα πλαίσιο δεδομένων μόνο με αριθμητικές τιμές, για παράδειγμα:

```r
df <- data.frame(x = c(5, 4, 6), y = c(-1, -2, 3))
df

##   x  y
## 1 5 -1
```

```
## 2 4 -2
## 3 6  3
```

τότε μπορούμε να υπολογίσουμε τα αθροίσματα κατά στήλη:

```
colSums(df)

##  x  y
## 15  0
```

ή τα αθροίσματα κατά στήλη:

```
rowSums(df)

## [1] 4 2 9
```

Καθώς επίσης, μέση τιμή κατά στήλη:

```
colMeans(df)

## x y
## 5 0
```

και μέση τιμή κατά γραμμή

```
rowMeans(df)

## [1] 2.0 1.0 4.5
```

Πολλές φορές θα δούμε να υπολογίζονται αθροίσματα κατά γραμμή μετά αό αναστροφή του πλαισίου δεδομένων (γραμμές σε ατήλες):

```
t(df)

##   [,1] [,2] [,3]
## x    5    4    6
## y   -1   -2    3
```

Οπότε το άθροισμα κατά γραμμή:

```
colSums(t(df))

## [1] 4 2 9
```

Αν χρειαστεί να υπολογίσουμε το άθροισμα των στοιχείων μόνο μιας συγκεκριμένης στήλης, για παράδειγμα της x, τότε θα γράψουμε:

```
sum(df$x)

## [1] 15
```

Ενώ αντίστοιχα αν χρειαστούμε τη μέση τιμή των στοιχείων μιας συγκεκριμένης στήλης, για παράδειγμα της y, τότε θα γράψουμε

```
mean(df$y)
```

```
## [1] 0
```

Μερικές φορές χρειάζεται να κατασκευάσουμε πλαίσια δεδομένων (ή μήτρες ανάλογα με την περίπτωση) με κάποια συγκεκριμένη δομή. Ας δούμε μερικά τυπικά παραδείγματα.

Τους αριθμούς από το 1 έως το 5 και τα τετράγωνά τους. Αυτό μπορεί να γίνει ως:

```
df <- data.frame(x = 1:5)
df$y <- df$x ** 2
```

Οπότε έχουμε:

```
df

##   x  y
## 1 1  1
## 2 2  4
## 3 3  9
## 4 4 16
## 5 5 25
```

Δηλαδή κατασκευάζουμε ένα πλαίσιο δεδομένων με μία στήλη (x) που περιέχει τους αριθμούς 1 έως 5. Στη συνέχεια προσθέτουμε μια επιπλέον στήλη (df$y ¿υψώνοντας στο τετράγωνο τα στοιχεία της πρώτης στήλης¿x). Ισοδύναμα μπορούμε να πετύχουμε το ίδιο αποτέλεσμα ως εξής:

```
x <- 1:5
y <- x ** 2
df <- data.frame(x, y)
df

##   x  y
## 1 1  1
## 2 2  4
## 3 3  9
## 4 4 16
## 5 5 25
```

Ποιον τρόπο να προτιμήσουμε; Δεν υπάρχει ιδιαίτερος λόγος να είμαστε αυστηροί ως προς επιλογή. Ο πρώτος είναι περισσότερο οικονομικός: δεν φορτώνει τη μνήμη με ενδιάμεσες μεταβλητές που δεν χρειαζόμαστε. Ο δεύτερος είναι ίσως περισσότερο κατανοητός λόγω της βηματικής κατασκευής.

Ας δούμε ένα ακόμα παράδειγμα. Ένα πλαίσιο δεδομένων που περιέχει το ημίτονο και το συνημίτονο των αριθμών από το 0 έως το 2π ανά $\pi/6$ Να πως μπορούμε να πετύχουμε κάτι τέτοιο:

```
df <- data.frame(x = seq(from = 0, to = 2*pi, by = pi/4))
df$sin <- sin(df$x)
df$cos <- cos(df$x)
```

Οπότε έχουμε:

```
df
```

```
##           x           sin           cos
## 1 0.0000000  0.000000e+00  1.000000e+00
## 2 0.7853982  7.071068e-01  7.071068e-01
## 3 1.5707963  1.000000e+00  6.123234e-17
## 4 2.3561945  7.071068e-01 -7.071068e-01
## 5 3.1415927  1.224647e-16 -1.000000e+00
## 6 3.9269908 -7.071068e-01 -7.071068e-01
## 7 4.7123890 -1.000000e+00 -1.836970e-16
## 8 5.4977871 -7.071068e-01  7.071068e-01
## 9 6.2831853 -2.449294e-16  1.000000e+00
```

Είναι πιθανό να χρειαστεί να μετατρέψουμε ένα πλαίσιο δεδομένων σε μήτρα. Αυτό μπορούμε να το πετύχουμε ως εξής:

```
as.matrix(df)
```

```
##                x           sin           cos
##  [1,] 0.0000000  0.000000e+00  1.000000e+00
##  [2,] 0.7853982  7.071068e-01  7.071068e-01
##  [3,] 1.5707963  1.000000e+00  6.123234e-17
##  [4,] 2.3561945  7.071068e-01 -7.071068e-01
##  [5,] 3.1415927  1.224647e-16 -1.000000e+00
##  [6,] 3.9269908 -7.071068e-01 -7.071068e-01
##  [7,] 4.7123890 -1.000000e+00 -1.836970e-16
##  [8,] 5.4977871 -7.071068e-01  7.071068e-01
##  [9,] 6.2831853 -2.449294e-16  1.000000e+00
```

Ενώ αν έχουμε μία μήτρα:

```
m <- as.matrix(df)
class(m)
```

```
## [1] "matrix" "array"
```

Μπορούμε να τη μετατρέψουμε σε πλαίσιο δεδομένων:

```
df <- as.data.frame(m)
class(df)
```

```
## [1] "data.frame"
```

Όπως και σε άλλες περιπτώσεις το αν είναι κάποιο μεταβλητή πλαίσιο δεδομένων μπορούμε να το ελέγξουμε με τη συνάρτηση is.data.frame():

```
is.data.frame(df)
```

```
## [1] TRUE
```

```
is.data.frame(m)
```

```
## [1] FALSE
```

1.7 Ειδικές τιμές

1.7.1 Άπειρο

Είναι γνωστό πως η διαίρεση $1/0$ δεν είναι δυνατή. Στα περισσότερα υπολογιστικά περιβάλλοντα θα πάρουμε ένα μήνυμα σφάλματος. Σε κάποιες γλώσσες προγραμματισμού θα διακοπεί η ροή του προγράμματος. Στην R θα πάρουμε το αποτέλεσμα:

```
1 / 0
```

```
## [1] Inf
```

που αντιστοιχεί σε:

$$\lim_{x \to 0} \frac{1}{x} = \infty$$

Δεν έχουμε δηλαδή διακοπή της ροής του προγράμματος, αλλά παίρνουμε ως απάντηση την ειδική τιμή Inf. Η τιμή αυτή είναι αριθμός, δεν είναι λεκτικό. Μπορούμε να το ελέγξουμε ως εξής:

```
x <- Inf
is.numeric(x)
```

```
## [1] TRUE
```

Αν επίσης χρειαστεί να ελέγξουμε αν κάποια τιμή είναι άπειρο ή όχι μπορούμε να εκτελέσουμε την εντολή:

```
is.infinite(x)
```

```
## [1] TRUE
```

ενώ υπάρχει επίσης η δυνατότητα να ελέγξουμε αν κάποια τιμή είναι πεπερασμένη:

```
is.finite(x)
```

```
## [1] FALSE
```

Πέρα από τη διαίρεση με το μηδέν, η τιμή άπειρο μπορεί να προκύψει και ως αποτέλεσμα έγκυρων παραστάσεων με τη μαθηματική έννοια. Για παράδειγμα αν εξετάσουμε το αποτέλεσμα της πράξης

```
100 ^ 100
```

```
## [1] 1e+200
```

θα δούμε έναν πολύ μεγάλο αριθμό. Πόσο μεγάλο αριθμό ωστόσο μπορεί να χειριστεί η R ή γενικά ο υπολογιστής; Όχι *άπειρα* μεγάλο. Για παράδειγμα:

```
200 ^ 200
```

```
## [1] Inf
```

Αυτή η παράσταση είναι απολύτως έγκυρη με τη μαθηματική έννοια. Ωστόσο στο υπολογιστικό περιβάλλον που χρησιμοποιούμε δεν είναι δυνατή η αναπαράσταση ενός τόσο μεγάλου αριθμού. Το πόσο μεγάλος είναι ο αριθμός που μπορεί να χειριστεί ένα υπολογιστικό περιβάλλον, ή ένας μεταγλωττιστής μιας γλώσσας προγραμματισμού ποικίλει και εξαρτάται από διάφορους παράγοντες. Στον υπολογιστή όπου γράφεται αυτό το βιβλίο η ποσότητα αυτή είναι:

```
.Machine$double.xmax
```

```
## [1] 1.797693e+308
```

Δηλαδή ένας αριθμός περίπου $\approx 1.8\,10^{308}$, πολύ μεγάλος μεν, όχι άπειρο δε. Για παράδειγμα:

```
10^308 * 2
```

```
## [1] Inf
```

Δηλαδή ενώ εκτελούμε μια μαθηματικά έγκυρη πράξη παίρνουμε ως αποτέλεσμα *άπειρο* ενώ γνωρίζουμε πως το αποτέλεσμα είναι πεπερασμένος αριθμός. Άπειρο λοιπόν μπορεί να προκύψει και από έγκυρες μαθηματικά παραστάσεις για απλά ο υπολογιστής δεν έχει τη δυνατότητα να αναπαριστά μεγάλους αριθμούς πέρα από κάποιο όριο. Αν και έχει εξαιρετικά απίθανο και σπάνιο να χρειαστούμε τόσο μεγάλους αριθμούς στην καθημερινή εργασία μας είναι πιθανό κάτι τέτοιο να προκύψει ως ενδιάμεσο αποτέλεσμα άλλων πράξεων. Προσοχή λοιπόν.

Μαζί με την τιμή του *άπειρα θετικά*:

```
Inf
```

```
## [1] Inf
```

μπορούμε να έχουμε και την τιμή *άπειρα αρνητικά*:

```
-Inf
```

```
## [1] -Inf
```

Οι τιμές αυτές μπορούν να συμμετέχουν μέσα σε παραστάσεις λογικών ελέγχων. Για παράδειγμα:

```
Inf > 0
```

```
## [1] TRUE
```

```
-Inf > 0
```

```
## [1] FALSE
```

```
Inf > -Inf
```

```
## [1] TRUE
```

Επίσης είναι έγκυρος ο παρακάτω έλεγχος:

```
x == Inf
```

```
## [1] TRUE
```

αν και είναι πάντοτε προτιμότερο να χρησιμοποιούμε τη συνάρτηση is.infinite αν θέλουμε να ελέγξουμε για το *άπειρο* στην τιμή μιας μεταβλητής. Για παράδειγμα:

```
x <- Inf
y <- -Inf
is.infinite(x)
```

```
## [1] TRUE
```

```
is.infinite(y)
```

```
## [1] TRUE
```

Σε απλές αλγεβρικές παραστάσεις όπου συμμετέχει η τιμή Inf το αποτέλεσμα θα είναι Inf, για παράδειγμα:

```
x <- Inf
x + 1
```

```
## [1] Inf
```

```
2 * x
```

```
## [1] Inf
```

Αλλά αν διαιρέσουμε θα πάρουμε ως αποτέλεσμα το μηδέν:

```
1 / x
```

```
## [1] 0
```

Ενδιαφέρον έχει επίσης το αποτέλεσμα διαίρεσης με το μηδέν:

```
x / 0
```

```
## [1] Inf
```

ενώ ακόμα μεγαλύτερο ενδιαφέρον έχει το αποτέλεσμα:

```
x * 0
```

```
## [1] NaN
```

```
x / x
```

```
## [1] NaN
```

Το NaN είναι μια άλλη ειδική τιμή την οποία θα εξετάσουμε αμέσως παρακάτω, σημαίνει not a number (μη αριθμητικό αποτέλεσμα, αδύνατος υπολογισμός).

Οι παραπάνω έλεγχοι και υπολογισμοί μπορούν επίσης να εφαρμοστούν εκεί όπου η τιμή Inf είναι μέλος ενός διανύσματος τιμών, για παράδειγμα:

```
x <- c(1.5, Inf, -Inf, 5)
is.numeric(x)
```

```
## [1] TRUE
```

```
x == Inf
```

```
## [1] FALSE  TRUE FALSE FALSE
```

```
x == -Inf
```

```
## [1] FALSE FALSE  TRUE FALSE
```

```
x > 0
```

```
## [1]  TRUE  TRUE FALSE  TRUE
```

ή ας δούμε τις παρακάτω παραστάσεις με το διάνυσμα τιμών x:

```
# expressions with vectors
x + 1
```

```
## [1]  2.5  Inf -Inf  6.0
```

```
2 * x
```

```
## [1]    3  Inf -Inf   10
```

```
1 / x
```

```
## [1] 0.6666667 0.0000000 0.0000000 0.2000000
```

```
x / 0
```

```
## [1]  Inf  Inf -Inf  Inf
```

```
x * 0
```

```
## [1]   0 NaN NaN   0
```

```
x / x
```

```
## [1]   1 NaN NaN   1
```

1.7.2 NaN, not a number

Η ειδική αυτή τιμή (μη αριθμητικό αποτέλεσμα) μπορεί να προκύψει από παραστάσεις όπως:

```
0 / 0
```

```
## [1] NaN
```

```
Inf / Inf
```

```
## [1] NaN
```

```
Inf - Inf
```

```
## [1] NaN
```

Οποιαδήποτε παραστάση περιλαμβάνει NaN έχει ως αποτέλεσμα NaN:

```
NaN + 1
```

```
## [1] NaN
```

```
NaN / 5
```

```
## [1] NaN
```

NaN αποτελέσματα μπορεί να προκύψουν από φαινομενικά *αθώες* παραστάσεις, για παράδειγμα:

```
x <- 1
y <- 3
2 ^ ((y - 3*x) / (x - 1))
```

```
## [1] NaN
```

```
(3*x - y) / (x - 1)
```

```
## [1] NaN
```

όπου προσπαθούμε να αναπαράγουμε τους τύπους

$$2^{\frac{y-3x}{x-1}}, \frac{3*x-y}{x-1}$$

Οι μαθηματικές αυτές παραστάσεις είναι βέβαια έγκυρες, αλλά δεν ορίζονται παντού.

Για να ελέγξουμε την ύπαρξη μιας NaN τιμής θα χρειαστούμε τη συνάρτηση is.nan, για παράδειγμα:

```
x <- NaN
is.nan(x)

## [1] TRUE
```

Κάτι που δεν μπορούμε να το κάνουμε με τη συνάρτηση is.finite:

```
is.finite(x)

## [1] FALSE

is.infinite(x)

## [1] FALSE
```

Δηλαδή η τιμή NaN δεν είναι ούτε άπειρη ούτε πεπερασμένη! Αλλά είναι αριθμητική:

```
is.numeric(x)

## [1] TRUE
```

Παρόλο που είναι NaN! Αυτό αποτελεί μια *παραξενιά* της R (έχει και πολλές άλλες). Το γενικό συμπέρασμα είναι για το κάθε πράγμα ελέγχουμε με την ειδική για το σκοπό συνάρτηση και δεν προσπαθούμε να βγάλουμε έμεσα συμπεράσματα. Για παράδειγμα, είναι λάθος να χρησιμοποιήσουμε τον τελεστή ισότητας:

```
x == NaN

## [1] NA
```

όπου θα πάρουμε μια άλλη ειδική τιμή, την NA (not available, μη διαθέσιμη).

Η τιμή NaN μπορεί να υπάρξει ως στοιχείο σε ένα διάνυσμα. Σε αυτή τη περίπτωση τα αποτελέσματα πράξεων με διάνυσμα θα είναι ανάλογα με όσα έχουν ήδη ειπωθεί. Για παράδειγμα:

```
x <- c(1, NaN, 5)
x + 1

## [1]   2 NaN   6

2 * x

## [1]   2 NaN  10

1 / x

## [1] 1.0 NaN 0.2
```

```
x / 0
```

```
## [1] Inf NaN Inf
```

```
x * 0
```

```
## [1]   0 NaN   0
```

```
x / x
```

```
## [1]   1 NaN   1
```

1.7.3 Μη διαθέσιμη τιμή

Η μη διαθέσιμη τιμή (Not Available, NA) είναι τιμή η οποία μπορεί να υπάρξει μεν, ωστόσο δεν είναι διαθέσιμη κάποιο συγκεκριμένη στιγμή. Η πλέον συνηθισμένη τέτοια περίπτωση είναι στο λεγόμενο *κενό στα δεδομένα*, για παράδειγμα ανάγνωση δεδομένων από αρχείο Excel όπου ένα κελί δεν έχει συμπληρωμένη τιμή. Η τιμή μπορεί γενικά να υπάρχει, αλλά δεν υπάρχει τη στιγμή που διαβάζουμε το αρχείο. Η τιμή NA είναι διαφορετική από την NaN, δηλαδή την τιμή που λόγους που σχετίζονται με μαθητικούς όρους και κανόνες δεν υπάρχει (όπως για παράδειγμα είναι η διαίρεση με το μηδέν). Η τιμή γράφεται ως:

```
NA
```

```
## [1] NA
```

Με κεφαλαία λατινικούς χαρακτήρες.

Ας υποθέσουμε πως έχουμε κάνει την παρακάτω εκχώρηση:

```
x <- 10
```

Η μεταβλητή x φαίνεται ως βαθμωτή, αλλά στην ουσία είναι διανυσματική. Όλες οι μεταβλητές στην R είναι διανυσματικές. Έτσι, μπορούμε να γράψουμε:

```
x[1]
```

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

Δηλαδή να ζητήσουμε την τιμή της πρώτης θέσης του διανύσματος x και να πάρουμε ως απάντηση την τιμή 10. Το διάνυσμα x έχει μήκος 1:

```
length(x)
```

```
## [1] 1
```

Τι θα συμβεί αν ζητήσουμε τη δεύτερη θέση του διανύσματος;

```
x[2]
```

```
## [1] NA
```

Η τιμή αυτή δεν υπάρχει, το αποτέλεσμα είναι NA. Προσοχή, δεν παίρνουμε κάποιο μήνυμα σφάλματος, δεν διακόπτεται η ροή του προγράμματος. Κάτι τέτοιο θα ήταν απολύτως αναμενόμενο σε άλλες γλώσσες προγραμματισμού, για παράδειγμα, C, C++, Java, Fortran, Matlab, κα. Στο περιβάλλον της R κάτι τέτοιο δεν είναι σφάλμα, η τιμή της δεύτερης θέσης (και κάθε άλλης θέσης εκτός της πρώτης) είναι απλά μη διαθέσιμη. Μπορούμε να την ορίσουμε αργότερα, για παράδειγμα:

```
x[2] <- 5
```

Οπότε τώρα το διάνυσμα x έχει μήκος 2:

```
length(x)

## [1] 2
```

Και η τιμή της δεύτερης θέσης είναι:

```
x[2]

## [1] 5
```

Αντίστοιχα είναι NA η τιμή της τρίτης θέσης:

```
x[3]

## [1] NA
```

Αντίστοιχες παρατηρήσεις μπορούμε να κάνουμε και σε διανύσματα στα οποία εξαρχής έχουν εκχωρηθεί πολλές τιμές. Για παράδειγμα:

```
x <- c(2, 5, 0)
```

Η τρίτη θέση έχει την τιμή:

```
x[3]

## [1] 0
```

Αλλά η τέταρτη, ή πέμπτη (και κάθε άλλη παρακάτω) έχει τιμή:

```
x[5]

## [1] NA
```

Αν μία τιμή είναι ή όχι NA μπορεί να ελεγχθεί με τη συνάρτηση is.na:

```
x <- NA
is.na(x)

## [1] TRUE
```

```r
x <- 10
is.na(x)

## [1] FALSE
```

Ένας πολύ συχνός λόγος όπου παρατηρούμε τιμές NA είναι η αποτυχία μετατροπής. Για παράδειγμα ας υποθέσουμε πως κάνουμε ανάγνωση δεδομένων από εξωτερικό αρχείο που περιέχει αριθμητικά στοιχεία. Αν κάπου έχει γίνει λάθος και το πρόγραμμα συναντήσει την τιμή 1ο (ένα όμικρον) αντί της κανονικής τιμής 10 τότε:

```r
as.numeric("1o")

## Warning: NAs introduced by coercion

## [1] NA
```

Η R θα αποτύχει να μετατρέψει το 1ο σε αριθμό, στη θέση του θα τοποθετεί η τιμή NA.

Είναι πιθανό να έχουμε τιμές NA σε οποιαδήποτε θέση ενός διανύσματος, στην αρχή, στο τέλος, ή στη μέση. Για παράδειγμα:

```r
x <- c(1, NA, 3)
```

Τότε ο έλεγχος is.na θα αποδώσει

```r
is.na(x)

## [1] FALSE  TRUE FALSE
```

ένα αντίστοιχο διάνυσμα τιμών TRUE/FALSE ανάλογα με την ύπαρξη ή όχι μη διαθέσιμων τιμών NA.

Ενώ ο έλεγχος is.numeric

```r
is.numeric(x)

## [1] TRUE
```

θα επιστρέψει μία μόνο τιμή TRUE/FALSE ανάλογα με το άν το διάνυσμα περιέχει ή όχι αριθμητικές τιμές.

Ιδιαίτερη μεγάλη προσοχή χρειάζεται όταν εφαρμόζουμε άθροισμα ή μέση τιμή σε διάνυσμα που περιέχει τιμές NA, για παράδειγμα:

```r
x <- c(1, NA, 3)
sum(x)

## [1] NA
```

Τι συμβαίνει; Γιατί το άθροισμα δεν είναι το 4; Γιατί πολύ απλά το αποτέλεσμα οποιασδήποτε αλγεβρικής πράξης με NA έχει ως αποτέλεσμα NA, για παράδειγμα:

```
NA + 1
```

```
## [1] NA
```

```
NA * 2
```

```
## [1] NA
```

```
NA - NA
```

```
## [1] NA
```

Οτιδήποτε και να κάνουμε με τιμές NA το αποτέλεσμα είναι NA. Ίσως έχουμε κατά νου αυτό που συμβαίνει σε άλλα περιβάλλοντα, όπως για παράδειγμα τα λογιστικά φύλλα (MS Excel, LibreCalc, googlesheets, κα) όπου σε περίπτωση αθροίσματος ένα κενό κελί παίρνει την τιμή 0. Από στατιστική άποψη κάτι τέτοιο είναι λάθος, δεν μπορούμε να γνωρίζουμε αν η τιμή ενός κενού κελιού είναι 0 ή κάτι άλλο.

Αν ωστόσο θέλουμε να υπολογίσουμε το άθροισμα των τιμών χωρίς να λάβουμε υπόψη τις τιμές NA τότε θα πρέπει να γράψουμε:

```
sum(x, na.rm = TRUE)
```

```
## [1] 4
```

Δηλαδή να ζητήσουμε από τη συνάρτηση sum να απαλείψει από τον υπολογισμό τις τιμές NA. Ακριβώς το ίδιο συμβαίνει και με τη συνάρτηση mean για τον υπολογισμό της μέσης τιμής:

```
mean(x)
```

```
## [1] NA
```

```
mean(x, na.rm = TRUE)
```

```
## [1] 2
```

Αυτό που είδαμε λίγο παραπάνω σχετικά με τις αλγεβρικές πράξεις NA το παρατηρούμε επίσης και σε διανύσματα, για παράδειγμα:

```
x + 1
```

```
## [1]  2 NA  4
```

```
x * 2
```

```
## [1]  2 NA  6
```

οι θέσεις που περιέχουν τιμές NA θα παραμείνουν NA οποιοδήποτε και αν είναι η αλγεβρική παράσταση. Το ίδιο θα συμβεί και σε οποιοδήποτε λογικό έλεγχο:

```
x == 1

## [1]  TRUE    NA FALSE

x > 2

## [1] FALSE    NA  TRUE
```

Οι τιμές NA θα αποδώσουν NA.
Δεν μπορούμε να γνωρίζουμε αν είναι ίσες με τη μονάδα ή μεγαλύτερες του 2.
Το αποτέλεσμα σύγκρισης οτιδήποτε με NA είναι πάντοτε NA.

Αν θα πρέπει να ελέγχξουμε τιμές NA δεν μπορούμε να το κάνουμε με τον τελεστή ισότητας. Αυτό είναι λάθος:

```
x == NA

## [1] NA NA NA
```

Ο σωστός τρόπος είναι με τη συνάρτηση is.na:

```
is.na(x)

## [1] FALSE  TRUE FALSE
```

Μπορούμε να απαλλαγούμε από τις τιμές NA ενός διανύσματος ως εξής:

```
x <- c(1, NA, 3)
y <- na.omit(x)
```

Το διάνυσμα y δεν περιέχει τιμές NA

```
y

## [1] 1 3
## attr(,"na.action")
## [1] 2
## attr(,"class")
## [1] "omit"
```

Έχει αλλάξει βέβαια η κλάση από **vector** σε **omit**. Τα παραπάνω σχόλια μας δείχνουν επίσης πως στη θέση 2 υπήρχε τιμή NA. Αν θέλουμε να μετατρέψουμε σε **vector** το αποτέλεσμα (και να χάσουμε την πληροφορία για το που υπήρχε τιμή NA) θα γράψουμε:

```
x <- c(1, NA, 3)
y <- as.vector(na.omit(x))
```

Και τώρα το διάνυσμα y είναι:

```
y
```

```
## [1] 1 3
```

Το αποτέλεσμα αυτό μπορούμε να το πετύχουμε επίσης με τη χρήση της συνάρτησης na.exclude:

```
na.exclude(x)
```

```
## [1] 1 3
## attr(,"na.action")
## [1] 2
## attr(,"class")
## [1] "exclude"
```

Παρόμοια με πριν αν θέλουμε το αποτέλεσμα να έχει την κλάση **vector** μπορούμε να γράψουμε:

```
y <- as.vector(na.exclude(x))
```

Τώρα το y είναι διάνυσμα:

```
y
```

```
## [1] 1 3
```

Σε κάποιες άλλες περιπτώσεις, κυρίως σε προγραμματιστικές δομές ελέγχου, χρειάζεται να αποφασίσουμε για να το αν θα συνεχίσουμε τη ροή του προγράμματος εφόσον έχουμε συναντήσει τιμές NA, ή αν θα πρέπει να γίνει διακοπή της ροής του προγράμματος. Ας δούμε λοιπόν πως δουλεύουν οι συναρτήσεις na.pass και na.fail.

Αν αποφασίσουμε να συνεχίσουμε τη ροή του προγράμματος:

```
na.pass(x)
```

```
## [1]  1 NA  3
```

Δεν συμβαίνει τίποτα, το πρόγραμμα συνεχίζεται κανονικά. Αν αποφασίσουμε να σταματήσουμε τη ροή του προγράμματος:

```
na.fail(x)
```

```
## Error in na.fail.default(x): missing values in object
```

Το πρόγραμμα θα διακοπεί, δεν θα εκτελεστούν οι εντολές κάτω από την εντολή na.fail(x).

1.7.4 Ελλιπής τιμή

Η ελλιπής τιμή

```
NULL
```

```
## NULL
```

μοιάζει με την μη διαθέσιμη, αλλά δεν ακριβώς το ίδιο. Πρόκειται κυρίως για τιμή αντίστοιχη των τιμών NULL στις βάσεις δεδομένων. Αν μία μεταβλητή έχει την τιμή NULL, τότε μπορεί να ελεχγθεί με τη συνάρτηση is.null :

```
x <- NULL
is.null(x)

## [1] TRUE
```

Ενώ η συνάρτηση is.na θα αποδώσει:

```
is.na(x)

## logical(0)
```

δηλαδή αδυναμία απάντησης, logical(0) είναι το αποτέλεσμα λογικού ελέγχου που δεν μπορεί να επιστρέψει ούτε TRUE ούτε FALSE. Οι έλεγχοι για τον τύπο δεδομένων, για παράδειγμα:

```
is.numeric(x)

## [1] FALSE

is.character(x)

## [1] FALSE
```

θα αποδώσουν FALSE. Όπως είδαμε παραπάνω με τις ειδκές τιμές NA και NaN έτσι και εδώ ο έλεγχος δεν μπορεί να γίνει με τον τελεστή ισότητας:

```
x == NULL

## logical(0)
```

Κάθε αλγεβρική παράσταση που περιέχει τιμή NULL θα επιστρέψει numeric(0):

```
x + 1

## numeric(0)

x / x

## numeric(0)
```

δηλαδή ένα διάνυσμα μηδενικού μήκους.

προσοχή σε διανύσματα που περιέχουν τιμή NULL, ασ δούμε για παράδειγμα την τιμή της δεύτερης θέσης του διανύσματος:

```
x <- c(10, NULL, 30)
x[2]

## [1] 30
```

Γιατί το 30; Γιατί απλά η τιμή NULL είναι σαν να μην υπάρχει. Όπως δεν υπάρχει τιμή στην τρίτη θέση του διανύσματος:

```
x[3]

## [1] NA
```

Δηλαδή το διανύσμα έχει στην πραγκατικότητα δύο θέσεις:

```
length(x)

## [1] 2
```

Ή, το παρακάτω διάνυσμα έχει μήκος 2:

```
x <- c(10, NULL, 30, NULL)
length(x)

## [1] 2
```

NULL τιμές μπορούν να εμφανιστούν επίσης σε αόριστες θέσεις μια λίστας. Για παράδειγμα έστω η λίστα τιμών x που περιέχει τις μεταβλητές a και b:

```
x <- list(a = 10, b = 20)
x

## $a
## [1] 10
##
## $b
## [1] 20
```

Ενώ για παράδειγμα η τιμή της x$a είναι 10:

```
x$a

## [1] 10
```

η τιμή x$c είναι NULL

```
x$c

## NULL
```

εφόσον η μεταβλητή c δεν έχει οριστεί ως μέλος της λίστας x.