Εισαγωγή στην R

Δρ. Σωτήριος Δ. Νικολόπουλος

Big Data & Analytics

Πανεπιστήμιο Πελοποννήσου

Τμήμα Λογιστικής & Χρηματοοικονομικών

s.nikolopoulos@go.uop.gr

Ενότητες

Ενότητες Παρουσίασης

Εισαγωγή

Η R δεν είναι απλά μια γλώσσα προγραμματισμού, αλλά και ένα περιβάλλον λογισμικού.

Είναι ευρέως διαδεδομένη και χρησιμοποιείται κυρίως για στατιστικούς υπολογισμούς, για την παραγωγή γραφικών απεικονίσεων και για την επεξεργασία και ανάλυση των δεδομένων κατά την Εξόρυξη Δεδομένων.

Οι βασικότεροι λόγοι για τους οποίους έγινε τόσο δημοφιλής η R είναι η ευκολία στην εκμάθησή της, η συμβατότητά της με τα πιο διαδεδομένα λειτουργικά συστήματα (Linux, Mac OS και Windows), το ότι διαθέτει έναν μεγάλο αριθμό έτοιμων πακέτων με καλογραμμένα εγχειρίδια χρήσης, και τέλος το γεγονός ότι είναι δωρεάν διαθέσιμη.

Στην κονσόλα της R, ο χρήστης μπορεί να πληκτρολογήσει διάφορες εκφράσεις.

Στις περισσότερες γλώσσες προγραμματισμού υπάρχουν μεταβλητές και τύποι μεταβλητών.

Ωστόσο, η R "βλέπει" τα πάντα ως αντικείμενα (object), τα οποία ανήκουν σε μια κλάση (class).

Με απλά λόγια, για την R τα αντικείμενα είναι οι μεταβλητές, ενώ η κλάση είναι ο τύπος τους.

Στην R δεν απαιτείται η ρητή δήλωση της κλάσης στην οποία ανήκουν τα αντικείμενα.

Αυτή καθορίζεται αυτόματα από την τιμή που θα ανατεθεί στο αντικείμενο. Η ανάθεση τιμής γίνεται με τον τελεστή <− ή με τον τελεστή =

Η R έχει πέντε βασικές ή ατομικές (atomic) κλάσεις αντικειμένων:

1. χαρακτήρας (character)
2. αριθμητικός – πραγματικοί αριθμοί (numeric)
3. ακέραιος (integer)
4. σύνθετος (complex)
5. λογικός (logical – True/False)

Η R χρησιμοποιεί, επίσης, βασικές **δομές δεδομένων ως κλάσεις αντικειμένων**.

Η βασικότερη δομή είναι το διάνυσμα (vector).

Ένα διάνυσμα μπορεί να περιέχει αντικείμενα του ίδιου μόνο τύπου.

Η δημιουργία ενός διανύσματος είναι εφικτή, χρησιμοποιώντας είτε τη συνάρτηση **c** από την αγγλική λέξη concatenate (= συνένωση), είτε τη συνάρτηση **vector**.

Οι αριθμοί αντιμετωπίζονται γενικά ως αριθμητικά αντικείμενα, δηλαδή ως πραγματικοί αριθμοί.

Στην περίπτωση που θέλουμε να ορίσουμε ρητά έναν αριθμό ως ακέραιο, θα πρέπει μετά τον αριθμό να ακολουθεί το επίθεμα L.

Αξίζει να αναφέρουμε ότι υπάρχουν και ειδικές τιμές, όπως η τιμή Inf, η οποία αναπαριστά το άπειρο, και η τιμή **NaN** (Not a Number) η οποία αναπαριστά μη ορισμένη τιμή.

Τύποι Δεδομένων

Βασικές κλάσεις αντικειμένων της R.

```
> x <- "Hello World!"
> class(x)
[1] "character"
> y <- 3.14
> class(y)
[1] "numeric"
> z <- 15L
> class(z)
[1] "integer"
> c <- 5 + 2i
> class(c)
[1] "complex"
> t <- TRUE
> class(t)
[1] "logical"
```

Τύποι Δεδομένων

Παράδειγμα χρήσης των attributes.

Κάθε αντικείμενο έχει συγκεκριμένες ιδιότητες, όπως:

1. names,
2. dim,
3. class,
4. length,
5. και άλλες ιδιότητες ορισμένες από τον χρήστη.

```
> x <- list(age=c(10, 21 ,33), weight=c(30, 66, 80))
> names(x)
[1] "age"    "weight"
> length(x)
[1] 2
```

Τύποι Δεδομένων

```
> x <- c("This", "is", "a", "character", "vector")
> x
[1] "This"      "is"        "a"         "character" "vector"
> y <- c(1, 2, 3, 5, 7)
> y
[1] 1 2 3 5 7
> class(x)
[1] "character"
> class(y)
[1] "numeric"
```

Διανύσματα και Λίστες

Όπως ήδη αναφέρθηκε, η R υποστηρίζει βασικές δομές δεδομένων ως κλάσεις αντικειμένων.

Η βασικότερη δομή είναι το διάνυσμα.

Ο πιο εύκολος τρόπος δημιουργίας ενός διανύσματος είναι με χρήση της συνάρτησης c, από την αγγλική λέξη concatenate (= συνένωση).

Διανύσματα και Λίστες

Παράδειγμα δημιουργίας διανύσματος

```
> x <- c("This", "is", "a", "character", "vector")
> x
[1] "This"      "is"        "a"         "character" "vector"
> y <- c(1, 2, 3, 5, 7)
> y
[1] 1 2 3 5 7
> class(x)
[1] "character"
> class(y)
[1] "numeric"
```

Εναλλακτικά, μπορεί να χρησιμοποιηθεί η συνάρτηση vector.

Γενικά, η δεικτοδότηση των διάφορων δομών στην R ξεκινούν από το 1 και όχι από το 0.

Αρχικοποίηση λογικού διανύσματος μήκους 5

```
> x <- vector(mode="logical", length=5)
> x
[1] FALSE FALSE FALSE FALSE FALSE
> x[1] <- TRUE
> x
[1]  TRUE FALSE FALSE FALSE FALSE
```

Διανύσματα και Λίστες

Τα διανύσματα ανήκουν στα ατομικά (atomic) αντικείμενα.

Αυτό σημαίνει ότι τα επιμέρους αντικείμενα του διανύσματος θα πρέπει να ανήκουν στην ίδια κλάση.

Σε περίπτωση που αυτό δεν ισχύει, τότε η R **δεν θα εκτυπώσει σφάλμα** αλλά θα κάνει τροποποίηση κλάσης, ώστε όλα τα αντικείμενα να ανήκουν στην ίδια κλάση.

Για τις βασικές κλάσεις, η σειρά βαρύτητας είναι χαρακτήρας, αριθμός, λογικός.

```
> x <- c("Hello World!", 1, TRUE)
> x
[1] "Hello World!" "1"           "TRUE"
> y <- c(1, TRUE, FALSE)
> y
[1] 1 1 0
```

Διανύσματα και Λίστες (Παράδειγμα δημιουργίας λίστας.)

Σε περίπτωση που θέλουμε να αποφύγουμε την αυτόματη μετατροπή, μπορούμε είτε να ορίσουμε ρητά το πώς αυτή θα γίνει με τις συναρτήσεις

- as.integer,
- as.integer,
- as.numeric,
- as.logical κοκ,

είτε να χρησιμοποιήσουμε μια άλλη δομή, τη λίστα (list).

Διανύσματα και Λίστες (Παράδειγμα δημιουργίας λίστας.)

Μια λίστα, όπως και το διάνυσμα, είναι ένα σύνολο από αντικείμενα, που όμως μπορούν να ανήκουν σε διαφορετική κλάση.

Για τη δημιουργία λίστας χρησιμοποιούμε τη συνάρτηση **list**.

```
> x <- list("Hello World!", 2015, TRUE, 3.14)
> x
[[1]]
[1] "Hello World!"
[[2]]
[1] 2015
[[3]]
[1] TRUE
[[4]]
[1] 3.14
```

Μητρώα

Ένα μητρώο (matrix) είναι ουσιαστικά πολλά διανύσματα ενωμένα.

Δηλαδή, είναι μια ειδική δομή, η οποία έχει ως επιπλέον ιδιότητα (attribute) τη διάσταση (dimension).

Με απλά λόγια, ένα μητρώο είναι ένα **δισδιάστατο διάνυσμα**, και από τον ορισμό του είναι ατομικό, δηλαδή τόσο οι γραμμές όσο και οι στήλες του μητρώου θα πρέπει να περιέχουν αντικείμενα της ίδιας κλάσης.

Υπάρχουν αρκετοί τρόποι δημιουργίας ενός μητρώου.

Ένας από αυτούς είναι δημιουργώντας ένα διάνυσμα και στη συνέχεια θέτοντας τις διαστάσεις με τη συνάρτηση dim.

Μητρώα

Παράδειγμα δημιουργίας μητρώου από διάνυσμα.

```
> mat <- c(1, 3, 2, 4)
> dim(mat) <- c(2, 2)
> mat
[,1] [,2]
[1,]    1    2
[2,]    3    4
```

Μητρώα

Παράδειγμα δημιουργίας μητρώου με χρήση της συνάρτησης matrix.

Εναλλακτικά, μπορούμε να δημιουργήσουμε μητρώο μόνο με χρήση της συνάρτησης **matrix**, θέτοντας όμως κάποια επιπλέον ορίσματα.

```
> temp <- c(1, 2, 3, 7, 8, 9)
> mat <- matrix(temp, nrow=2, ncol=3, byrow=TRUE)
> mat
[,1] [,2] [,3]
[1,]    1    2    3
[2,]    7    8    9

"# Default byrow=FALSE"
> mat <- matrix(temp, nrow=2, ncol=3)
> mat
[,1] [,2] [,3]
[1,]    1    3    8
[2,]    2    7    9
```

Μητρώα

Ένας ακόμα τρόπος είναι δένοντας υπάρχοντας διανύσματα είτε κατά γραμμές με χρήση της συνάρτησης rbind είτε κατά στήλες με χρήση της συνάρτησης **cbind**.

```
> t1 <- c(1, 2, 3)
> t2 <- c(7, 8, 9)
# Γραμμές
> rbind(t1, t2)
[,1] [,2] [,3]
t1    1    2    3
t2    7    8    9

# Στήλες
> cbind(t1, t2)
t1 t2
[1,]  1  7
[2,]  2  8
```

Παράγοντες και Κατηγορικά Δεδομένα

Οι παράγοντες (factors) παρέχουν έναν εύκολο τρόπο αναπαράστασης και διαχείρισης κατηγορικών (nominal) δεδομένων.

Διαθέτουν επίπεδα (levels), τα οποία ουσιαστικά είναι οι πιθανές τιμές που μπορούν να πάρουν.

Η δημιουργία ενός παράγοντα γίνεται με χρήση της συνάρτησης **factor**.

Η σειρά των επιπέδων σε κάποιες περιπτώσεις παίζει ρόλο.

Επίσης, σε κάποιες περιπτώσεις είναι χρήσιμο να θέσουμε το όρισμα **levels** της συνάρτησης **factor**, περιορίζοντας έτσι τις επιτρεπτές τιμές.

Σε αυτή την περίπτωση, τιμές, οι οποίες δεν αναφέρονται στο όρισμα levels, απορρίπτονται και θεωρούνται ελλιπείς τιμές.

Παράγοντες και Κατηγορικά Δεδομένα

Παράδειγμα δημιουργίας παράγοντα (factor).

```
> factor(c("Yes", "No", "No", "Yes"))
[1] Yes No  No  Yes
Levels: No Yes
> f <- factor(c("Yes", "No", "No", "Yes"), levels=c("Yes"))
> f
[1] Yes  <NA> <NA> Yes
Levels: Yes
```

Ελλιπείς Τιμές

Παράδειγμα ελλιπών τιμών.

Υπάρχει ένας ειδικός τύπος δεδομένων για την αναπαράσταση των ελλιπών τιμών.

Οι ελλιπείς τιμές στην R συμβολίζονται είτε ως **NA**, είτε ως **NaN** για μη προκαθορισμένες μαθηματικές πράξεις.

Για τον έλεγχο ελλιπών τιμών υπάρχουν οι συναρτήσεις **is.na** και **is.nan**, αντίστοιχα.

Η τιμή **NA** ανήκει στην αριθμητική κλάση, ενώ η τιμή NaN στη λογική κλάση.

```
> x <- NA
> is.na(x)
[1] TRUE
> y <- 0/0
> y
[1] NaN
> is.nan(y)
[1] TRUE
```

Πλαίσια Δεδομένων

Τα πλαίσια δεδομένων (data frames) χρησιμοποιούνται για την αποθήκευση δεδομένων σε μορφή πίνακα.

Έχουν παρόμοια δομή με τα μητρώα, καθώς είναι και αυτά δισδιάστατα.

Ωστόσο, σε αντίθεση με τα μητρώα,μπορούν, όπως και οι λίστες, να έχουν διαφορετικό τύπο δεδομένων σε κάθε στήλη τους.

Ένας λογικός περιορισμός που θέτουν είναι ότι κάθε στήλη μπορεί να περιέχει μόνο αντικείμενα της ίδιας κλάσης.

Για τη δημιουργία ενός πλαισίου δεδομένων χρησιμοποιούμε τη συνάρτηση **data.frame**.

Πλαίσια Δεδομένων

Παράδειγμα δημιουργίας πλαισίου δεδομένων (data frame).

```
> x <- c("maria", "giwrgos", "giannhs")
> y <- c(15, 16, 20)
> z <- c(FALSE, FALSE, TRUE)
> dfr <- data.frame(username=x, age=y, adult=z)
> dfr
username age adult
1    maria  15 FALSE
2  giwrgos  16 FALSE
3  giannhs  20  TRUE
> dfr[1,]
username age adult
1    maria  15 FALSE
```

Πλαίσια Δεδομένων

Παράδειγμα δημιουργίας πλαισίου δεδομένων (data frame).

```
> dfr[1,]
username age adult
1     maria  15 FALSE
> dfr[,1]
[1] "maria"   "giwrgos" "giannhs"
> dfr$age
[1] 15 16 20
```

Βασικές Εργασίες

Μια από τις βασικότερες εργασίες είναι η ανάγνωση δεδομένων από κάποιο αρχείο.

Οι πιο διαδεδομένοι τρόποι ανάγνωσης αρχείων είναι με χρήση των συναρτήσεων **read.table** και **read.csv**.

Μερικά από τα βασικότερα ορίσματα τους είναι τα εξής:

- **file**, το όνομα του αρχείου,
- **header**, λογικό όρισμα, το οποίο σηματοδοτεί αν το αρχείο έχει γραμμή κεφαλίδας ή όχι,
- **sep**, αλφαριθμητικό, το οποίο ορίζει με τι διαχωρίζονται οι στήλες, π.χ. κόμμα, κενό κ.λπ.,
- **colClasses**, διάνυσμα χαρακτήρων με τις κλάσεις κάθε στήλης του συνόλου δεδομένων,
- **nrows**, αριθμός γραμμών, που θα διαβαστούν – η προεπιλογή είναι να διαβαστεί ολόκληρο το αρχείο,

- **comment.char**, αλφαριθμητικό, που ορίζει τον χαρακτήρα σχολίων στο αρχείο,
- **skip**, αριθμός γραμμών, που θα παραληφθούν από την αρχή του αρχείου,
- **stringsAsFactors**, λογικό όρισμα - τα αντικείμενα κλάσης χαρακτήρα να κωδικοποιηθούν ως παράγοντες. Η προεπιλεγμένη τιμή είναι TRUE.

```
> dat <- read.csv("/home/sotnik/records.csv", stringsAsFacto
> dat
X   ID age height weight gender
1    1 343     15    160     50 female
2    2 548     17    170     70   male
3    3 736     20    165     55 female
4    4 955     21    190     86   male
5    5 230     23    170     60 female
6    6 697     22    183     64   male
7    7 129     16    158     68 female
8    8 853     19    173     63   male
9    9 923     17    183    121 female
10  10 101     16    180     58   male
```

Παραγωγή ακολουθιών

Η παραγωγή ακολουθιών είναι μια απλή, αλλά αρκετά σημαντική εργασία, αφού θέτει τα θεμέλια για πιο σημαντικές εργασίες, όπως η αναφορά σε υποσύνολο μιας δομής και η διανυσματοποίηση.

Ο πιο απλός τρόπος παραγωγής ακολουθιών είναι με χρήση του τελεστή ”**:**”.

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> y <- -5:5
> y
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

Παραγωγή ακολουθιών

Ένας άλλος τρόπος παραγωγής ακολουθιών είναι με χρήση της συνάρτησης **seq**.

Η συνάρτηση seq δέχεται τα ακόλουθα ορίσματα:

- **from**, η αρχή της ακολουθίας,
- **to**, ο μέγιστος αριθμός, και συνεπώς το τέλος, της ακολουθίας,
- **by**, το βήμα αύξησης της ακολουθίας – προεπιλεγμένη τιμή το 1,
- **length**, αν δοθεί στη θέση του by, χωρίζει το διάστημα from-to σε τόσα διαστήματα και επιστρέφει τις τιμές των άκρων.

Παραγωγή ακολουθιών

Δημιουργία ακολουθίας με χρήση του τελεστή ":".

```
> x <- seq(from=2, to=10, by=3)
> x
[1] 2 5 8
> x <- seq(from=2, to=10, by=2)
> x
[1]  2  4  6  8 10
> y <- seq(from=2, to=10, length=4)
> y
[1]  2.000000  4.666667  7.333333 10.000000
```

Παραγωγή ακολουθιών

Δημιουργία ακολουθίας με χρήση της συνάρτησης rep.

Τέλος, μια ακόμα χρήσιμη συνάρτηση για παραγωγή ακολουθιών με συγκεκριμένο μοτίβο είναι και η συνάρτηση **rep**, η οποία δέχεται τα ακόλουθα ορίσματα:

- **x**, το αντικείμενο που θα χρησιμοποιηθεί για τη δημιουργία της ακολουθίας.
- **times**, φορές επανάληψης του αντικειμένου.

```
> x <- rep(1:3, 4)
> x
[1] 1 2 3 1 2 3 1 2 3 1 2 3
> x <- rep("hello", 5)
> x
[1] "hello" "hello" "hello" "hello" "hello"
```

Αναφορά σε Υποσύνολα Δομών

Τα διανύσματα, τα μητρώα και οι λίστες παρέχουν έναν τρόπο ομαδοποίησης των δεδομένων.

Ωστόσο, σε αρκετές περιπτώσεις ο χρήστης χρειάζεται να χρησιμοποιήσει ένα υποσύνολο μόνο αυτών των δεδομένων.

Υπάρχουν 3 διαφορετικοί τελεστές για την αναφορά σε ένα υποσύνολο μιας δομής.

Ο πιο απλός τελεστής για αναφορά σε υποσύνολο δομής είναι η μονή τετραγωνική παρένθεση “**[**”.

Μπορεί να έχει ως περιεχόμενο έναν αριθμό ή μια ακολουθία αριθμών για την αναφορά ενός ή περισσότερων στοιχείων της δομής.

Το αποτέλεσμα που επιστρέφεται ανήκει πάντα στην ίδια κλάση με αυτή στην οποία ανήκε το αρχικό αντικείμενο.

Αναφορά σε Υποσύνολα Δομών

Παράδειγμα αναφοράς σε υποσύνολο δομής με τον τελεστή “[“.

```
> x <- seq(1, 15, 2)
> x
[1] 1 3 5 7 9 11 13 15
> x[1:3]
[1] 1 3 5
> class(x[1:3])
[1] “”numeric
>
> y <- list“(”Hello, “”Planet, “Earth”!)
> y[c(1,3)]
[[1]]
[1] “”Hello
[[2]]
[1] “Earth”!
```

Αναφορά σε Υποσύνολα Δομών

Ένας άλλος τελεστής για αναφορά υποσυνόλου είναι οι διπλές τετραγωνικές παρενθέσεις ”[[”.

Ο συγκεκριμένος τελεστής χρησιμοποιείται για λίστες και πλαίσια δεδομένων.

Μπορεί να χρησιμοποιηθεί για αναφορά ενός μόνο στοιχείου της δομής και το αποτέλεσμα που επιστρέφεται ανήκει στην κλάση, στην οποία ανήκει το στοιχείο, όπου γίνεται η αναφορά..

Αναφορά σε Υποσύνολα Δομών

Παράδειγμα αναφοράς σε υποσύνολο δομής με τον τελεστή "[[".

```
> y <- list("Hello", "Planet", "Earth!")
> y[[1]]
[1] "Hello"
> class(y[[1]])
[1] "character"
> y[[c(1,3)]]
Error in y[[c(1, 3)]] : subscript out of bounds
```

Αναφορά σε Υποσύνολα Δομών

Ο τρίτος τελεστής είναι το δολάριο ”$” και μπορεί να γίνει αναφορά σε στοιχεία λίστας ή πλαισίου δεδομένων με ονόματα.

Για την κλάση του αποτελέσματος που επιστρέφεται ισχύει η ίδια λογική με αυτή του τελεστή [[.

```
> y <- list(age=c(15, 16, 28), height=c(1.60, 1.68, 1.76))
> y$age
[1] 15 16 28
> y$height
[1] 1.60 1.68 1.76
> class(y)
[1] "list"
> y$age
[1] 15 16 28
> class(y$age)  [1] "numeric"
```

Αναφορά σε Υποσύνολα Δομών

Παράδειγμα χρήσης του ορίσματος drop σε αναφορά στοιχείου μητρώου.

Ειδικά στα μητρώα, όταν γίνεται αναφορά σε ένα μόνο στοιχείο, το αποτέλεσμα θεωρείται ως ένα διάνυσμα μήκους 1, αντί για ένα μητρώο διαστάσεων 1×1.

Με χρήση του ορίσματος **drop** μπορούμε να αλλάξουμε αυτή τη συμπεριφορά.

```
> x <- matrix(1:4, nrow=2, ncol=2, byrow=TRUE)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> class(x[1,1])
[1] "integer"
> class(x[1,1, drop=FALSE])
[1] "matrix" "array"
```

Αναφορά σε Υποσύνολα Δομών

Παράδειγμα χρήσης του ορίσματος exact για μερικό ταίριασμα ονόματος.

Οι τελεστές [[και $ επιτρέπουν μερικό ταίριασμα ονομάτων.

Αυτό είναι εφικτό με χρήση του ορίσματος **exact**.

```
> y <- list(age=c(15, 16, 28), height=c(1.60, 1.68, 1.76))
> y
$age
[1] 15 16 28
$height
[1] 1.60 1.68 1.76
> y["age"]]
[1] 15 16 28
> y["a", exact=FALSE]]
[1] 15 16 28
```

Αναφορά σε Υποσύνολα Δομών

Παράδειγμα αφαίρεσης ελλιπών τιμών.

Τέλος, μια από τις πιο χρήσιμες εφαρμογές της αναφοράς υποσυνόλου δομής είναι η εξαγωγή των θέσεων με ελλιπείς τιμές NA.

```
> y <- c(15, 20, 45, NA, NA, 50)
> y
[1] 15 20 45 NA NA 50
> i <- is.na(y)
> i
[1] FALSE FALSE FALSE  TRUE  TRUE FALSE
> y[!i]
[1] 15 20 45 50
```

Διανυσματοποίηση

Ένα από τα χαρακτηριστικά της R που την κάνουν να ξεχωρίζει, είναι ότι επιτρέπει την εκτέλεση πράξεων μεταξύ διανυσμάτων και μητρώων.

Η μετατροπή πράξεων σε πράξεις διανυσμάτων/μητρώων λέγεται διανυσματοποίηση (vectorization).

Σε αρκετές περιπτώσεις, πράξεις μεταξύ διανυσμάτων ή/και μητρώων μπορούν να αντικαταστήσουν τη χρήση βρόχων επανάληψης.

Έτσι, ο κώδικας μπορεί να γίνει συνοπτικός και αρκετά πιο ευανάγνωστος.

Ο κυριότερος λόγος που θέλουμε να έχουμε όσο το δυνατόν περισσότερη διανυσματοποίηση είναι ότι οι πράξεις μεταξύ διανυσμάτων εκτελούνται πολύ πιο γρήγορα απ' ό,τι αν γίνονταν οι επιμέρους πράξεις μία-μία με βρόχο επανάληψης.

Αυτό συμβαίνει, διότι κατά τις πράξεις μεταξύ διανυσμάτων, οι πράξεις εκτελούνται παράλληλα.

Διανυσματοποίηση

Σύγκριση χρόνου πρόσθεσης διανυσμάτων με βρόχο επανάληψης και διανυσματοποίηση.

```
> x <- rnorm(10000000)
> y <- rnorm(10000000)
> z <- vector(mode="numeric", length=10000000)
> start <- proc.time()
> for (i in 1:10000000){
  }
proc.time()-start
user    system   elapsed
15.23    0.03     15.29

> z[i] <- x[i] + y[i]
> proc.time()-start
user  system elapsed
0.129    0.006    2.659
```

Δομές Ελέγχου

Εκτέλεση υπό συνθήκη: if-else

Η R διαθέτει βασικές δομές ελέγχου, όπως η υπό συνθήκη εκτέλεση και οι βρόχοι επανάληψης.

Αυτές οι δομές ελέγχου είναι πολύ απλές, αλλά συγχρόνως χρήσιμες.

Εκτέλεση υπό συνθήκη: if-else Αυτή είναι και η πιο βασική δομή ελέγχου.

Η συντριπτική πλειοψηφία των προγραμμάτων σε κάποιο σημείο θα χρειαστεί να χρησιμοποιήσει υπό συνθήκη εκτέλεση.

Ουσιαστικά, γίνεται έλεγχος μιας έκφρασης και αν αυτή αποτιμηθεί ως αληθής (True), τότε εκτελείται ένα κομμάτι κώδικα. Σε αντίθετη περίπτωση, υπάρχουν 3 πιθανά σενάρια.

Είτε να συνεχιστεί η ροή του προγράμματος, είτε να ελεγχθεί κάποια άλλη συνθήκη, ή τέλος να εκτελεστεί ένα άλλο κομμάτι κώδικα, το οποίο θα θέλαμε να εκτελεστεί μόνο στην περίπτωση που η έκφραση είναι ψευδής (False).

Δομές Ελέγχου

Παράδειγμα δομής ελέγχου if-else.

```
> x <- 15
> if (x < 0) {
  print("Negative!")
  }else if (x < 10){
    print("Positive, less than 10!")
    }else{
      print("Number larger than 10!")
      }
[1] "Number larger than 10!"
```

Εκτέλεση κατ' επανάληψη: for, repeat και while

Οι επαναληπτικές δομές έχουν ως στόχο την εκτέλεση ενός κομματιού κώδικα για προκαθορισμένο ή μη προκαθορισμένο αριθμό φορών.

Στην R, ο βρόχος for αρκεί για να καλύψει το μεγαλύτερο μέρος των περιπτώσεων που απαιτούν επαναληπτική εκτέλεση.

Ο βρόχος επανάληψης **for** μπορεί να χρησιμοποιηθεί με δυο τρόπους.

Κατά τον κλασσικό τρόπο, μια μεταβλητή παίρνει τιμές από ένα καθορισμένο εύρος τιμών σε κάθε επανάληψη.

Κατά τον εναλλακτικό τρόπο, μια μεταβλητή παίρνει τιμές από τα στοιχεία μιας συλλογής αντικειμένων.

Εκτέλεση κατ' επανάληψη: for, repeat και while

Παράδειγμα επαναληπτικού βρόχου for.

```
> for(i in 1:10){
    cat(i)
    cat(" ")
  }
1 2 3 4 5 6 7 8 9 10
> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
> for(x in letters){
      cat(x)
      cat(" ")
    }
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Εκτέλεση κατ' επανάληψη: for, repeat και while

Παράδειγμα επαναληπτικού βρόχου while.

Ένας άλλος τρόπος επανάληψης είναι με χρήση του βρόχου **while**.

Σε έναν βρόχο while αρχικά ελέγχεται μια έκφραση.

Όσο (while) η έκφραση είναι αληθής (true), τότε συνεχίζεται να εκτελείται το κομμάτι κώδικα μέσα στον βρόγο, μέχρι η έκφραση να γίνει ψευδής (false).

```
> x <- 1
> while (x < 5){
    print(x)
    x <- x+1
    }
[1] 1
[1] 2
[1] 3
[1] 4
```

Εκτέλεση κατ' επανάληψη: for, repeat και while

Παράδειγμα επαναληπτικού βρόχου repeat.

Μια ακόμα επαναληπτική δομή είναι η repeat.

Ουσιαστικά, πρόκειται για έναν ατέρμονο επαναληπτικό βρόχο.

Ο μόνος τρόπος τερματισμού είναι με χρήση της εντολής break.

```
> x <- 1
> repeat{
    print(x)
    if (x > 3){
      break
    }
    x <- x+1
    }
[1] 1
[1] 2
[1] 3
[1] 4
```

Εντολές next και break

Παράδειγμα εντολής next.

Οι εντολές next και break χρησιμοποιούνται σε συνδυασμό με τις επαναληπτικές δομές.

Η εντολή next χρησιμοποιείται για την παράλειψη μιας επανάληψης ενός βρόχου.

Ουσιαστικά, με την next ο βρόχος προχωράει στην επόμενη επανάληψη, μη εκτελώντας ότι ακολουθεί μετά την εντολή.

```
for(i in 1:100){
    # Προσπέρασε ~ τις 20 πρώτες ~ επαναλήψεις
    if (i <= 20){
        next
    }
}
```

Συναρτήσεις

Οι συναρτήσεις είναι αναπόσπαστο κομμάτι της R.

Σε κάθε ένα από τα πολλά έτοιμα πακέτα που είναι διαθέσιμα, υπάρχουν διαφορετικές συναρτήσεις, οι οποίες χρησιμοποιούνται για διαφορετική λειτουργία η κάθε μια.

Πέρα από τις έτοιμες συναρτήσεις της, η R επιτρέπει στον χρήστη να ορίσει τις δικές του συναρτήσεις.

Οι συναρτήσεις ορίζονται, χρησιμοποιώντας την οδηγία ή δεσμευμένη λέξη function και αποθηκεύονται και αυτές ως αντικείμενα, όπως γίνεται σχεδόν με τα πάντα στην R.

Πιο συγκεκριμένα, οι συναρτήσεις είναι αντικείμενα που ανήκουν στην κλάση ”function”.

Παρακάτω θα δούμε ότι συναρτήσεις μπορούν να χρησιμοποιηθούν ως ορίσματα άλλων συναρτήσεων.

Συναρτήσεις

Ορισμός συνάρτησης από τον χρήστη.

Στο παρακάτω παράδειγμα βλέπουμε τη δήλωση μιας απλής συνάρτησης, η οποία δέχεται έναν αριθμό και εκτυπώνει τόσες φορές ένα μήνυμα.

```
> myPrinter <- function(x){
+   for (i in seq_len(x)){
+     print("Hello World!")
+   }
+ }
> myPrinter(3)
[1] "Hello World!"
[1] "Hello World!"
[1] "Hello World!"
```

Συναρτήσεις

Χρήση του ορίσματος ... σε συνάρτηση.

Στην περίπτωση που ο αριθμός των ορισμάτων μια συνάρτησης δεν είναι σταθερός, χρησιμοποιούμε το όρισμα "...", το οποίο μεταφράζεται ως 1 ή περισσότερα ορίσματα ακολουθούν.

Το όρισμα "..." χρησιμοποιείται ως έχει μέσα στη συνάρτηση, όπως φαίνεται στο παρακάτω παράδειγμα.

Ένα θέμα που προκύπτει από τη χρήση του ορίσματος "..." είναι ότι όσα ορίσματα ακολουθούν μετά από αυτό θα πρέπει να αναφερθούν ρητά κατά το πέρασμα τιμών στη συνάρτηση.

```
> myPrinter <- function(..., mes){
+     print(sum(...))
+     print(mes)
+ }
> myPrinter(3, 5, 11, mes = "Hi!")
[1] 19
[1] "Hi!"
```

Κανόνες Εμβέλειας

Οι κανόνες εμβέλειας είναι το βασικό χαρακτηριστικό της R που τη διαφοροποιεί από τον πρόγονό της, τη γλώσσα προγραμματισμού S.

Αυτοί οι κανόνες χρησιμοποιούνται για να προκαθοριστεί ποια τιμή θα πάρει μια ελεύθερη μεταβλητή (free variable), η οποία ορίζεται και χρησιμοποιείται πρώτη φορά μέσα σε μια συνάρτηση.

Η R χρησιμοποιεί λεξικολογική εμβέλεια (lexical scoping).

Σύμφωνα με τους κανόνες της λεξικογραφικής εμβέλειας, οι τιμές των ελεύθερων μεταβλητών αναζητούνται στο ίδιο περιβάλλον, στο οποίο ορίστηκε και η συνάρτηση μέσα στην οποία ορίζονται.

Κανόνες Εμβέλειας

Η R χρησιμοποιεί την έννοια του περιβάλλοντος (environment). Ένα περιβάλλον είναι μια συλλογή από ζεύγη (σύμβολο, τιμή), για παράδειγμα το σύμβολο x έχει τιμή 5. Κάθε περιβάλλον έχει ένα γονικό περιβάλλον, με εξαίρεση το κενό περιβάλλον.
Για τον συσχετισμό τιμών και ελεύθερων μεταβλητών ακολουθείται η ακόλουθη διαδικασία αναζήτησης:

- Αν η τιμή ενός συμβόλου δεν βρεθεί στο περιβάλλον στο οποίο ορίστηκε η συνάρτηση, τότε η αναζήτηση συνεχίζεται στο γονικό περιβάλλον.
- Η αναζήτηση συνεχίζεται στη γονική ιεραρχία μέχρι το ανώτατο επίπεδο, στο οποίο βρίσκεται το καθολικό περιβάλλον (global environment), το οποίο ουσιαστικά είναι ο χώρος εργασίας (workspace) ή ο χώρος ονόματος του πακέτου (package namespace).
- Μετά το περιβάλλον στο ανώτατο επίπεδο, η αναζήτηση συνεχίζεται μέχρι να φτάσει στο άδειο περιβάλλον.
- Αν μια τιμή συμβόλου δεν έχει βρεθεί, πριν την άφιξη στο επίπεδο με

Επαναληπτικές Συναρτήσεις

Η δημιουργία βρόχων for και while είναι χρήσιμη και εύκολη, όχι όμως όταν πρόκειται να έχουμε πολλά επίπεδα εμφωλευμένων βρόχων.

Η R παρέχει κάποιες έτοιμες συναρτήσεις, οι οποίες υλοποιούν κατά κάποιο τρόπο τους βρόχους αυτούς με έναν πιο συμπαγή τρόπο. περιβάλλον. η R παράγει σφάλμα.

^^ |

Επαναληπτικές Συναρτήσεις

lapply

Η **lapply** υπολογίζει το αποτέλεσμα μιας συνάρτησης πάνω στο κάθε στοιχείο μιας λίστας. Τα βασικά βήματα που εκτελεί η συγκεκριμένη συνάρτηση είναι τα εξής:

- κάνει ένα πέρασμα της λίστας, στοιχείο προς στοιχείο,
- εφαρμόζει τη συνάρτηση σε κάθε στοιχείο της λίστας, και
- επιστρέφει μια λίστα.

Με την συνάρτηση **str**, μπορούμε να δούμε το πλήθος και τη σειρά των ορισμάτων οποιασδήποτε συνάρτησης.

Επαναληπτικές Συναρτήσεις

lapply

Με την συνάρτηση **str**, μπορούμε να δούμε το πλήθος και τη σειρά των ορισμάτων οποιασδήποτε συνάρτησης. Όπως θα δούμε στην συνέχεια, η συνάρτηση lapply παίρνει 3 ορίσματα ως είσοδο:

- **X**, η λίστα πάνω στα στοιχεία της οποίας θα εφαρμοστεί η συνάρτηση **FUN**,
- **FUN**, η συνάρτηση που θα εφαρμοστεί ή το όνομα της συνάρτησης,
- **...**, περιέχει τα ορίσματα που θα περαστούν στη **FUN**.

Επαναληπτικές Συναρτήσεις

Παράδειγμα χρήσης της lapply.

```
> str(lapply)
function (X, FUN, ...)
> x <- list(a=rnorm(10), b=rnorm(20), c=rnorm(30))
> lapply(x, mean)
$a
[1] -0.05970574

$b
[1] 0.1117777

$c
[1] -0.03975962
```

Επαναληπτικές Συναρτήσεις

sapply.

Η **sapply** λειτουργεί όπως η **lapply**, αλλά επιχειρεί να απλοποιήσει το αποτέλεσμα που επιστρέφεται.

Δηλαδή, η μόνη ουσιαστική διαφορά τους βρίσκεται στην επιστρεφόμενη τιμή.

Πιο συγκεκριμένα, η **sapply** προσπαθεί να απλοποιήσει το επιστρεφόμενο αποτέλεσμα ως εξής:

- Αν το αποτέλεσμα είναι μια λίστα, της οποία τα στοιχεία έχουν όλα μήκος ίσο με 1, τότε επιστρέφεται ένα διάνυσμα.
- Αν το αποτέλεσμα είναι μια λίστα, της οποία τα στοιχεία είναι όλα διανύσματα ίδιου μήκους (> 1), τότε επιστρέφεται ένα μητρώο.
- Αν όλα τα υπόλοιπα αποτύχουν, τότε επιστρέφει μια λίστα.

Επαναληπτικές Συναρτήσεις

Παράδειγμα χρήσης της slapply.

Συγκρίνοντας τα αποτελέσματα του παρακάτω παραδείγματος με αυτά του παραδείγματος της **lapply**, γίνεται πιο ξεκάθαρη η χρησιμότητα και λόγος ύπαρξης της **sapply**.

```
> str(sapply)
function (X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
> x <- list(a=rnorm(10), b=rnorm(20), c=rnorm(30))
> sapply(x, mean)
a                b                c
-0.002159579  0.282856490 -0.064730499
```

split

Η συνάρτηση split δεν ανήκει στις επαναληπτικές συναρτήσεις.

Η συγκεκριμένη συνάρτηση παίρνει ως όρισμα ένα διάνυσμα ή κάποιο άλλο αντικείμενο και το χωρίζει σε ομάδες βάσει ενός παράγοντα (factor) ή μιας λίστα παραγόντων (list of factors).

Ο λόγος, για τον οποίο αναφέρουμε αυτή τη συνάρτηση μαζί με τις επαναληπτικές συναρτήσεις, είναι ότι ο συνδυασμός της split μαζί με κάποια από τις lapply ή sapply αποτελεί κλασσικό παράδειγμα στην R.

Η βασική ιδέα είναι ότι μπορούμε να πάρουμε μια δομή δεδομένων, να τη χωρίσουμε σε υποσύνολα με βάση κάποια άλλη μεταβλητή, και στη συνέχεια να εφαρμόσουμε κάποια συνάρτηση σε αυτά τα υποσύνολα.

split

Παράδειγμα χρήσης της συνάρτησης split σε συνδυασμό με την sapply.

```
> str(sapply)
function (X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
> x <- list(a=rnorm(10), b=rnorm(20), c=rnorm(30))
> sapply(x, mean)
a           b           c
0.26171869  0.04063794 -0.16428579
> dat <- data.frame(subject=1:6, age=c(15,17,16,20,21,23),
+                   adult=c(FALSE,FALSE,FALSE,TRUE,TRUE,TRUE
> s <- split(dat, dat$adult)
```

split

Παράδειγμα χρήσης της συνάρτησης split σε συνδυασμό με την sapply.

```
> s
$`FALSE`
  subject age adult
1       1  15 FALSE
2       2  17 FALSE
3       3  16 FALSE
$`TRUE`
  subject age adult
4       4  20  TRUE
5       5  21  TRUE
6       6  23  TRUE
```

split

Παράδειγμα χρήσης της συνάρτησης split σε συνδυασμό με την sapply.

```
> sapply(s, function(x){
+    mean(x[["age"]])
+ })
   FALSE     TRUE
16.00000 21.33333
```

tapply

Η συνάρτηση tapply χρησιμοποιείται για την εφαρμογή μιας συνάρτησης πάνω σε ένα υποσύνολο ενός διανύσματος.

Μπορεί να θεωρηθεί ως συνδυασμός των συναρτήσεων split και sapply, αλλά μόνο για διανύσματα.

Τα ορίσματα της tapply είναι τα ακόλουθα:

- **X**, ένα διάνυσμα πάνω στο οποίο θα γίνει ο διαχωρισμός και η εφαρμογή της συνάρτησης,
- INDEX, παράγοντας (factor) ή λίστα παραγόντων (list of factors),
- FUN, η συνάρτηση που θα εφαρμοστεί,
- ..., περιέχει τα υπόλοιπα ορίσματα που θα περαστούν στην FUN,
- simplify, λογικό όρισμα – να απλοποιηθούν τα αποτελέσματα ή όχι;

tapply

Παράδειγμα χρήσης της συνάρτησης tapply.

```
> str(tapply)
function (X, INDEX, FUN = NULL, ..., default = NA, simplify
> x <- c(rnorm(10), rnorm(10), rnorm(10), rnorm(10))
> f <- gl(4, 10)
> f
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
Levels: 1 2 3 4
> tapply(x, f, mean)
          1           2           3           4
-0.15703595 -0.48610828 -0.01035259 -0.36655480
```