Contents

Uvod - osnove jezika R

U ovom dijelu proći ćemo kroz osnove programskog jezika R. Obradit ćemo osnove njegove sintakse te tipove varijabli. Ovdje ćemo proći kroz neke jako bazične stvari. Većina "naprednijih" stvari provlačit će se kroz ostatak radionice.

Osnovne matematičke operacije

Za početak, pogledat ćemo kako možemo izvršavati jednostavne naredbe direktno u R konzoli. Kad god u R konzolu unesemo naredbu koju R smatra valjanom te pritisnemo `Enter` (također poznat kao `Return`), R će u konzoli izbaciti rezultat izvršavanja te naredbe. Na primjer, ako u konzolu unesemo `2 + 2`, R će izbaciti rezultat te operacije.

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

Isto možemo napraviti s dijeljenjem (`/`), množenjem (`*`), oduzimanjem (`-`) i potenciranjem (`^`).

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

2 * 3
## [1] 6

5 - 1
## [1] 4

3^2
## [1] 9
```

Poseban operator koji nekad zna biti koristan je *modulo* - `%%` - koji vraća ostatak dijeljenja dvaju brojeva. Na primjer:

```
# (Linije koje sadrže komentare započinju sa znakom #. R ne interpretira te linije.)
# Obično dijeljenje:
5 / 2
## [1] 2.5

# Modulo
5 %% 2
## [1] 1
```

Naravno, kao i u pješačkoj matematici, i u R-u je potrebno paziti na grupiranje matematičkih izraza.

`x + y / z` je $x + (y/z)$ je $x + \frac{z}{y}$

`(x + y) / z` je $(x + y)/z$ je $\frac{x+y}{z}$.

Funkcije

Sa samim matematičkim operacijama nećemo daleko doći. R ima i funkcije - operacije koje primaju parametre (eng. *argument*) i vraćaju neke vrijednosti.

Funkcije u R-u imaju opći oblik `funkcija(argument1, argument2, ... , argumentN)`. Prva funkcija koju ćemo pogledati, a koja nadopunjava matematičke operacije s kojima smo započeli je `sqrt`, kojom možemo dobiti korijen nekog broja.

```
sqrt(4)
## [1] 2
```

Druga, koja se u R-u javlja **jako** često, je `c` (što je, prema Adleru [2012] skraćeno za *combine*). `c` uzima N argumenata i spaja ih u **vektor**.

```
# navodnici su bitni! ali mogu biti jednostruki ili dvostruki,
# bitno je samo da je riječ omeđena jednakim parom
# npr 'a' je oke, "a' nije oke, ali zato "a" je
print(c('patka', "krava", 'pile', "pesa"))
## [1] "patka" "krava" "pile"  "pesa"

print(c(5, 4, 3, 2, 1))
## [1] 5 4 3 2 1
```

Koristeći `c`, stvorili smo dva vektora. Vektori spadaju među osnovne strukture podataka u R-u. Vektori mogu sadržavati proizvoljan broj elemenata **istog tipa**. O tipovima ćemo pričati malo kasnije.

Sada ćemo se pozabaviti varijablama.

Varijable

Kad god smo dosad izvršavali neke funkcije, baratali smo konkretnim vrijednostima (npr. `2 + 2`), a rezultati su ostali lebdjeti u eteru, nedostupni običnim ljudima.

Kako bismo mogli baratati proizovljnim vrijednostima te kako bismo rezultati izvukli iz etera, uvodimo **varijable**. Varijablu imenujemo (eng. *declare*) tako što neki poluproizvoljan naziv spojimo s nekom vrijednosti, koristeći operator `<-`. Na pimjer:

```
a <- 2
```

Ako sad u konzolu unesemo `a`, konzola će nam vratiti vrijednost te varijable. Isti rezultat dobili bismo ako bismo `a` iskoristili kao argument `print` funkcije (`print(a)`).

```
print(a)
## [1] 2
```

Vrijednosti varijablama možemo pridavati (eng. *assign*) i koristeći znak `=` (razlikovati od `==`!), no to se **ne preporučuje**. Osim toga, vrijednosti možemo pridavati i s lijeva na desno, koristeći `->`:

```
3 -> b
b
## [1] 3
```

Imena varijabli mogu sadržavati slova, brojeve, točke (`.`) i underscoreove (čija hrvatska imena ne znam; `_`). Imena varijabli ne mogu započinjati s točkom koju prati broj. Na primjer:

```r
.3 <- 5
## Error in 0.3 <- 5: invalid (do_set) left-hand side to assignment
```

Također, imena varijabli ne mogu biti izrazi koji su rezervirani u samom programskom jeziku, kao što je `for` (koji se koristi za iniciranje petlji).

```r
for <- 5
## Error: <text>:1:5: unexpected assignment
## 1: for <-
##          ^
```

Funkcija koja zna biti zgodna kod imenovanja varijabli je **`exists`**, koja kao argument prima izraz u navodnicima (što je poznato kao **string**) te vraća `TRUE` ili `FALSE` ovisno o tom je li objekt istog imena pronađen ili ne.

```r
# varijabla koju smo ranije stvorili
exists('a')
## [1] TRUE

# ključna riječ, definirana u R-u, ne smijemo koristiti
exists('for')
## [1] TRUE

# ključna riječ, definirana u R-u, ne smijemo koristiti
exists('if')
## [1] TRUE

# ime koje nije iskorišteno
exists('Denis')
## [1] FALSE
```

Sad kad znamo kako varijablama pripisati vrijednosti, možemo spremiti vektore koje smo ranije napravili koristeći `c`. Neovisno o tome što *možemo* koristiti svašta za imena varijabli, trebali bismo se truditi imena učiniti smislenima. Dugoročno, to će nas poštedjeti puno mentalnog (a nekad i R-ovskog) napora. Također, savjetovao bih da izbjegavate korištenje "hrvatskih" znakova (č, ć, ž, š, đ) u svom kodu; korištenje tih znakova može izazvati snažne glavobolje.

```r
domace_zivotinje <- c('patka', 'krava', 'pile', 'pesa')

brojevi.5.do.1 <- c(5, 4, 3, 2, 1)
```

Kao i kad smo varijabli `a` pripisali vrijednost `2`, ni sada ne dobivamo nikakav output u konzoli. Ali možemo koristiti `print` ili samo upisati ime varijable u konzolu kako bismo dobili njenu vrijednost.

```r
print(domace_zivotinje)
## [1] "patka" "krava" "pile"  "pesa"

print(brojevi.5.do.1)
## [1] 5 4 3 2 1
```

Sad kad smo svoje vektore pripisali varijablama, možemo dohvaćati pojedine vrijednosti iz njih. Na primjer, ako želimo dohvatiti prvu vrijednost iz vektora `domace_zivotinje`, možemo učiniti ovo:

```r
domace_zivotinje[4]
## [1] "pesa"
```

`4` je, u ovom slučaju, **indeks**. U R-u, za razliku od većine drugih programskih jezika, indeksiranje započinje s `1`, a ne s `0`.

Za dohvaćanje trećeg elementa iz vektora `brojevi.5.do.1` izvršili bismo:

```
brojevi.5.do.1[3]
## [1] 3
```

Zadnji element možemo dohvatiti pomoću funkcije `length`, koja vraća duljinu vektora, tj. broj elemenata koji se u njemu nalaze. Na primjer:

```
length(domace_zivotinje)
## [1] 4
```

Budući da broj elemenata ujedno označava i posljednji element, možemo učiniti sljedeće:

```
# dohvaćanje pomoću indeksa
domace_zivotinje[4]
## [1] "pesa"

# dohvacanje pomocu funkcije length()
domace_zivotinje[length(domace_zivotinje)]
## [1] "pesa"

# iskoristit ćemo priliku i pokazati kako možemo usporediti dvije vrijednosti
domace_zivotinje[4] == domace_zivotinje[length(domace_zivotinje)]
## [1] TRUE
```

Ovo funkcionira jer evaluiranje, odnosno izvršavanje koda `length(domace_zivotinje)` kao rezultat vraća brojku 4. Također, vidjeli smo da možemo koristiti `==` kako bismo provjerili jesu li dva objekta, odnosno dvije varijable jednake. Na primjer

```
2 + 2 == 4
## [1] TRUE

4 == 4
## [1] TRUE

4 == 5
## [1] FALSE
```

Treba voditi računa o tome da su `==` i `=` **vrlo različiti**!

Tipovi varijabli

R razlikuje nekoliko osnovnih tipova podataka: - `character` : "stringovi", tj. tekstualni podaci. Npr. `'patka'` - `integer` : cijeli brojevi. Npr. `1` - `numeric` : realni brojevi. Npr. `1.161` - `logical` : logičke vrijednosti. Postoje ukupno dvije - `TRUE` (može se kratiti u `T`) i `FALSE` (može se kratiti u `F`)

Pogledat ćemo nekoliko primjera ovih tipova, te vidjeti kako možemo provjeriti kojeg je neka varijabla ili vrijednost tipa.

```
# character
'susjed'
## [1] "susjed"

# da bismo provjerili je li neka vrijednost character, koristimo is.character()
is.character('susjed')
## [1] TRUE
is.character(domace_zivotinje[4])
## [1] TRUE
```

```r
is.character(1)
## [1] FALSE
```

Kod `integer` i `numeric` tipova postoje neke specifičnosti.

```r
# integer
1
## [1] 1

# za provjeravanje koristimo is.integer()
is.integer(1)
## [1] FALSE
```

Pozivanje funkcije `is.integer()` s vrijednosti `1` vraća `FALSE`. To je zato jer R brojeve automatski sprema kao `numeric`.

```r
# za provjeravanje je li objekt numeric koristimo is.numeric()
is.numeric(1)
## [1] TRUE
```

Kako bismo natjerali R da nam da `integer` vrijednost, možemo staviti `L` na kraj broja:

```r
is.integer(1L)
## [1] TRUE
```

Ovo je zgodno znati jer se može dogoditi da funkcija traži `integer`, ali odbija prihvatiti (recimo) `5` kao odgovarajuću vrijednost.

```r
# na kraju, numeric
is.numeric(1.5115)
## [1] TRUE
```

Za pisanje decimalnih brojeva **moramo koristiti točku**.

```r
is.numeric(1,4141)

1,5151 + 1
## Error: <text>:3:2: unexpected ','
## 2:
## 3: 1,
##     ^
```

Posljednji tip je `logical`:

```r
TRUE == T
## [1] TRUE
FALSE == F
## [1] TRUE

is.logical(TRUE)
## [1] TRUE

is.logical(F)
## [1] TRUE
```

Za kraj, pogledat ćemo output različitih `is.` funkcija kad im damo različite vrijednosti.

```r
is.logical(1)
## [1] FALSE
```

```r
is.numeric(1L)
## [1] TRUE

is.character(02918)
## [1] FALSE

is.integer(151518)
## [1] FALSE
```

Primjećujemo da se `1L` tretira i kao `numeric` tip.

```r
is.integer(1L)
## [1] TRUE

is.numeric(1L)
## [1] TRUE
```

Isto ne vrijedi za, na primjer, `5.911`:

```r
is.integer(5.911)
## [1] FALSE

is.numeric(5.911)
## [1] TRUE
```

Nakon upoznavanja s osnovnim tipovima vrijednosti i varijabli, pogledat ćemo osnovne strukture podataka.

Strukture podataka

Strukture podataka su formati organiziranja, upravljanja i spremanja podataka koji omogućuju efikasno pristupanje podacima i njihovo modificiranje (https://en.wikipedia.org/wiki/Data_structure).

Već smo se upoznali s jednim tipom strukture podataka u R-u, a to je vektor. R ima nekoliko osnovnih struktura podataka. Ovdje ćemo proći kroz one koje se najčešće javljaju. Za ponavljanje, stvorit ćemo novi vektor:

```r
c('vektor', 'od', '4', 'elementa')
## [1] "vektor"   "od"       "4"        "elementa"

# možemo provjeriti je li neki objekt vektor koristeći is.vector
is.vector(c('vektor', 'od', '4', 'elementa'))
## [1] TRUE

# funkcije za provjeravanje tipova varijabli možemo primijeniti i na vektore
is.character(c('vektor', 'od', '4', 'elementa'))
## [1] TRUE
```

data.frame

`data.frame` je vjerojatno najvažnija osnovna struktura (ili barem ona s kojom ćete se najčešće družiti). On odgovara onom što možemo vidjeti u *Data viewu* SPSS-a - sastoji se od redova koji predstavljaju opažanja/sudionike/caseove i stupaca koji predstavljaju varijable. Može sadržavati varijable koje su različitih tipova (za razliku od nekih drugih struktura, koje žele gledati samo jedan tip podataka). `data.frame` možemo stvoriti koristeći istoimenu funkciju:

```
print(data.frame(brojke = c(1, 2, 3, 4, 5),
           'slova' = c('a', 'b', 'de', 'ce', 'fe'),
           'logike'= c(F, F, T, T, F)))
##   brojke slova logike
## 1      1     a  FALSE
## 2      2     b  FALSE
## 3      3    de   TRUE
## 4      4    ce   TRUE
## 5      5    fe  FALSE
```

Pri stvaranju novog `data.framea`, svi redovi moraju imati vrijednosti na svim stupcima jer će se R inače požaliti.

```
data.frame('brojke' = c(1, 2, 3, 4, 5),
           'slova' = c('a', 'b', 'de', 'ce', 'fe'),
           # maknuli smo zadnjji element (F) iz stupca 'logike'
           'logike'= c(F, F, T, T))
## Error in data.frame(brojke = c(1, 2, 3, 4, 5), slova = c("a", "b", "de", : arguments imply differing
```

Tome možemo doskočiti tako što ćemo eksplicitno neku vrijednost proglasiti nedostajućom, što činimo pomoću posebne vrijednosti `NA`:

```
data.frame('brojke' = c(1, 2, 3, 4, 5),
           'slova' = c('a', 'b', 'de', 'ce', 'fe'),
           # umjesto posljednjeg elementa u stupcu 'logike' stavili smo NA
           'logike'= c(F, F, T, T, NA))
##   brojke slova logike
## 1      1     a  FALSE
## 2      2     b  FALSE
## 3      3    de   TRUE
## 4      4    ce   TRUE
## 5      5    fe     NA
```

Proširit ćemo ovaj `data.frame` s još nekim vrijednostima te ga spremiti u varijablu `brojke_i_slova`:

```
brojke_i_slova <-
           # osim korištenjem funkcije c(), vektore brojeva možemo stavarati
           # i pomoću n:m
           data.frame('brojke' = 1:15,
           # pozivajući ugrađenu varijablu _letters_ možemo odabrati prvih 15
           # (malih) slova
           'slova' = letters[1:15],
           'logike'= c(F, F, T, T, NA))

brojke_i_slova
##    brojke slova logike
## 1       1     a  FALSE
## 2       2     b  FALSE
## 3       3     c   TRUE
## 4       4     d   TRUE
## 5       5     e     NA
## 6       6     f  FALSE
## 7       7     g  FALSE
## 8       8     h   TRUE
## 9       9     i   TRUE
```

```
## 10     10     j     NA
## 11     11     k  FALSE
## 12     12     l  FALSE
## 13     13     m   TRUE
## 14     14     n   TRUE
## 15     15     o     NA
```

Primjećujemo da se vrijednosti iz stupca `logike` ponavljaju. To funkcionira zato što je broj redova `data.framea` djeljiv s brojem elemenata koje smo stavili u vektor `logike` pri definiranju `data.framea` (imamo 15 redova, a vektor `logike` ima 5 elemenata).

Sad kad smo proširili `brojke_i_slova`, pogledat ćemo kako možemo pristupati vrijednostima u `data.frameu`. Elementima možemo pristupati korištenjem uglatih zagrada, kao i kod vektora. Pritom treba imati na umu da je `data.frame` **dvodimenzionalni objekt**, zbog čega traži **dva indeksa** odvojena zarezom - **prvi** se odnosi na **redove**, a **drugi** na **stupce**.

Ako jedan od indeksa izostavimo, ali stavimo zarez, R će vratiti sve elemente na odgovarajućem mjestu, odnosno vratit će sve redove ako izostavimo prvi indeks i sve stupce ako izostavimo drugi indeks.

```
# svi stupci prvog  reda
brojke_i_slova[1, ]
##   brojke slova logike
## 1      1     a  FALSE

# svi redovi prvog stupca
brojke_i_slova[, 1]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

Ovdje također možemo koristiti `n:m` sintaksu za dohvaćanje raspona vrijednosti. Na primjer, da bismo dohvatili prva tri reda i sve stupce `brojki_i_slova`, napravili bismo sljedeće:

```
# prva tri reda, svi stupci
brojke_i_slova[1:3, ]
##   brojke slova logike
## 1      1     a  FALSE
## 2      2     b  FALSE
## 3      3     c   TRUE
```

Za dohvaćanje vrijednosti koje nisu uzastopne, možemo koristiti funkciju `c`, koju također možemo kombinirati s `n:m` sintaksom:

```
# prva tri reda i redovi 7 do 12, te stupci 1 i 3
brojke_i_slova[c(1:3, 7:12), c(1, 3)]
##    brojke logike
## 1       1  FALSE
## 2       2  FALSE
## 3       3   TRUE
## 7       7  FALSE
## 8       8   TRUE
## 9       9   TRUE
## 10     10     NA
## 11     11  FALSE
## 12     12  FALSE
```

Stupcima možemo pristupati i pomoću njihovih imena:

```
brojke_i_slova[1:3, c('logike', 'brojke')]
##   logike brojke
```

```
## 1  FALSE      1
## 2  FALSE      2
## 3   TRUE      3
```

Naposljetku, **jednom** određenom stupcu možemo pristupiti koristeći `$` operator:

```
brojke_i_slova$logike
##  [1] FALSE FALSE  TRUE  TRUE    NA FALSE FALSE  TRUE  TRUE    NA FALSE
## [12] FALSE  TRUE  TRUE    NA
```

Prije nego što prijeđemo na sljedeću strukturu podataka, upoznat ćemo se s funkcijom `str` (structure). To je funkcija koja kao input prima neki objekt i vraća prikaz njegove strukture. Primjerice, možemo pogledati kakva je struktura našeg `data.framea` `brojke_i_slova`.

```
str(brojke_i_slova)
## 'data.frame':    15 obs. of  3 variables:
##  $ brojke: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ slova : Factor w/ 15 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ logike: logi  FALSE FALSE TRUE TRUE NA FALSE ...
```

R nas informira da je `brojke_i_slova` objekt tipa `data.frame` te da sadrži 15 redova (`15 obs.`) i 3 varijable. Uz svaku varijablu naveden je njen tip te je prikazano prvih nekoliko elemenata.

Vidimo da R varijablu `slova` tretira kao `Factor`, a ne kao `character`. `factor` je tip koji R koristi za reprezentiranje kategoričkih varijabli. `factor` možemo stvoriti pozivanjem istoimene funkcije:

```
print(factor(c('pas', 'mačka', 'mačka', 'pas', 'pas')))
## [1] pas   mačka mačka pas   pas
## Levels: mačka pas
```

Ispod sadržaja ovog vektora navedene su razine koje on može poprimiti (`levels`). Razine možemo dohvatiti i pozivom funkcije `levels`:

```
zvijeri <- factor(c('pas', 'mačka', 'mačka', 'pas', 'pas'))
levels(zvijeri)
## [1] "mačka" "pas"
str(zvijeri)
##  Factor w/ 2 levels "mačka","pas": 2 1 1 2 2
```

Faktori se koriste i za reprezentiranje rangiranih podatka. U tom slučaju potrebno je funkciji `factor` dodati argument `ordered` i dati mu vrijednost `TRUE`. Osim toga, potrebno je eksplicirati koji je redoslijed razina.

```
prihodi <- factor(c('srednji', 'niski', 'visoki', 'visoki', 'srednji'),
                  ordered = T, levels = c('niski', 'srednji', 'visoki'))
print(prihodi)
## [1] srednji niski   visoki  visoki  srednji
## Levels: niski < srednji < visoki
str(prihodi)
##  Ord.factor w/ 3 levels "niski"<"srednji"<..: 2 1 3 3 2
```

U kodu gore, `ordered` i `levels` su imenovani (named) argumenti. Imenovani argumenti se razlikuju od pozicijskih (eng. *positional*) po tome što nije bitno kojim ih redoslijedom postavljamo. Isti efekt dobili bismo da smo prvo napisali `levels`, a potom `ordered`.

Nekad se možemo izvući bez da argumente imenujemo, ali nekad će R baciti error, tako da tu razliku treba imati na umu. Sad kad smo se upoznali s faktorima, prijeći ćemo na liste.

list

Lista je uređeni skup elemenata. Listu možemo definirati koristeći funkciju `list`:

```
print(list('franz', 'liszt'))
## [[1]]
## [1] "franz"
##
## [[2]]
## [1] "liszt"
```

Objekti u listi ne moraju biti istog tipa. Na primjer, možemo napraviti listu koja sadrži jedan `character`, jedan `integer` i jedan `numeric`.

```
spisak <- list('franz', 1L, 3.14)
print(spisak)
## [[1]]
## [1] "franz"
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] 3.14
```

Brojevi u dvostrukim uglatim zagradama (`[[n]]`) daju nam do znanja da lista ima 3 elementa. To možemo potvrditi pozivom funkcije `str` na `spisku`.

```
str(spisak)
## List of 3
##  $ : chr "franz"
##  $ : int 1
##  $ : num 3.14
```

Ovdje vidimo i da `spisak` sadrži elemente različitih tipova. Liste možemo puniti raznolikim objektima, čak i drugim listama.

```
raznoliki_objekti <- list(# pojedine elemente listi možemo i imenovati
    imena = c('Ramiro', 'Zorro', 'Vladimir'),
    brojevi = c(3.61, 4.15, 7.151, 20:25),
    inception = list(glumci = c('Leonardo di Caprio', 'ostali'),
                     broj_kamera = 5))

str(raznoliki_objekti)
## List of 3
##  $ imena    : chr [1:3] "Ramiro" "Zorro" "Vladimir"
##  $ brojevi  : num [1:9] 3.61 4.15 7.15 20 21 ...
##  $ inception:List of 2
##   ..$ glumci     : chr [1:2] "Leonardo di Caprio" "ostali"
##   ..$ broj_kamera: num 5
```

Imenovanim elementima listi možemo pristupati isto kao i stupcima `data.framea`:

```
print(raznoliki_objekti$imena)
## [1] "Ramiro"   "Zorro"    "Vladimir"

print(raznoliki_objekti[2])
```

```
## $brojevi
## [1]  3.610  4.150  7.151 20.000 21.000 22.000 23.000 24.000 25.000
```

Također, možemo dohvatiti više elemenata odjednom.

```
print(raznoliki_objekti[c('imena', 'brojevi')])
## $imena
## [1] "Ramiro"   "Zorro"    "Vladimir"
##
## $brojevi
## [1]  3.610  4.150  7.151 20.000 21.000 22.000 23.000 24.000 25.000

print(raznoliki_objekti[2:3])
## $brojevi
## [1]  3.610  4.150  7.151 20.000 21.000 22.000 23.000 24.000 25.000
##
## $inception
## $inception$glumci
## [1] "Leonardo di Caprio" "ostali"
##
## $inception$broj_kamera
## [1] 5
```

Kad imamo ugniježđene (eng. *nested*) strukture, možemo ulančavati operatore za dohvaćanje kako bismo ušli dublje u strukture.

```
print(raznoliki_objekti$inception$glumci)
## [1] "Leonardo di Caprio" "ostali"

print(raznoliki_objekti['inception']['broj_kamera'])
## $<NA>
## NULL
```

Posljednji komad koda nije vratio rezultat koji smo očekivali (vrijednost koja je spremljena u `broj_kamera` je 5). Kad pogledamo output koji daje `str`, vidimo da `raznoliki_objekti['inception']` vraća listu koja ima jedan element, koji je lista koja ima dva elementa.

```
str(raznoliki_objekti['inception'])
## List of 1
##  $ inception:List of 2
##   ..$ glumci     : chr [1:2] "Leonardo di Caprio" "ostali"
##   ..$ broj_kamera: num 5
```

Kako bismo došli do broja kamera, možemo učiniti sljedeće:

```
raznoliki_objekti['inception']$inception$broj_kamera
## [1] 5
```

Do ove zavrzlame dolazi zbog operatora `[]`, koji vraća listu kad je primijenjen na listi. Ako želimo vratiti samo objekt koji se nalazi pod imenom `inception`, možemo koristiti operator `[[]]`.

```
str(raznoliki_objekti[['inception']])
## List of 2
##  $ glumci     : chr [1:2] "Leonardo di Caprio" "ostali"
##  $ broj_kamera: num 5
```

Sada dobivamo listu (umjesto liste lista), te možemo direktno dohvatiti broj kamera na bilo koji od dosad spomenutih načina. Učinit ćemo to te se pritom osvrnuti na ishode različitih metoda dohvaćanja.

```
str(raznoliki_objekti[['inception']]$broj_kamera)
##  num 5

# ovo je funkcija koja printa "sirove" stringove
# \n označava novi red
# funkciju ovdje koristimo samo za ispisivanje pregrade
cat('==========\n')
## ==========

str(raznoliki_objekti[['inception']]['broj_kamera'])
## List of 1
##  $ broj_kamera: num 5

cat('==========\n')
## ==========

str(raznoliki_objekti[['inception']][['broj_kamera']])
##  num 5

cat('==========\n')
## ==========

str(raznoliki_objekti$inception$broj_kamera)
##  num 5
```

Pogledat ćemo kako se ovi operatori ponašaju na varijabli `glumci`, koja sadrži 2 elementa.

```
str(raznoliki_objekti[['inception']]$glumci)
##  chr [1:2] "Leonardo di Caprio" "ostali"

cat('==========\n')
## ==========

str(raznoliki_objekti[['inception']]['glumci'])
## List of 1
##  $ glumci: chr [1:2] "Leonardo di Caprio" "ostali"

cat('==========\n')
## ==========

str(raznoliki_objekti[['inception']][['glumci']][1])
##  chr "Leonardo di Caprio"

cat('==========\n')
## ==========

str(raznoliki_objekti$inception$glumci[[2]])
##  chr "ostali"
```

Kao što možemo vidjeti, razlika između `[[]]` i `[]` gubi se kad se spustimo na najnižu razinu, tj. na dohvaćanje pojedinih elemenata vektora `glumci`. Osim što vraćaju različite objekte, `[[]]` i `[]` razlikuju se u sljedećem: - `[]` može dohvaćati više elemenata (npr. [1:5]), a `[[]]` samo jedan - `[[]]` može baratati parcijalnim imenima ako se argumentu `exact` da vrijednost `FALSE`, a `[]` nema tu mogućnost

```
str(raznoliki_objekti[['ince', exact = F]])
## List of 2
##  $ glumci     : chr [1:2] "Leonardo di Caprio" "ostali"
##  $ broj_kamera: num 5

str(raznoliki_objekti)
## List of 3
##  $ imena    : chr [1:3] "Ramiro" "Zorro" "Vladimir"
##  $ brojevi  : num [1:9] 3.61 4.15 7.15 20 21 ...
##  $ inception:List of 2
##   ..$ glumci     : chr [1:2] "Leonardo di Caprio" "ostali"
##   ..$ broj_kamera: num 5

# nije ono što bismo očekivali
str(raznoliki_objekti[[1:2]])
##  chr "Zorro"

str(raznoliki_objekti['ince', exact = F])
## Error in raznoliki_objekti["ince", exact = F]: incorrect number of dimensions
```

Ovu ponešto supitlnu i prepredenu razliku se isplati imati a umu jer različite funkcije očekuju različite inpute, a lako im je nesvjesno gurati nešto što ne žele (na primjer listu umjesto `data.framea` koji se nalazi u listi). Posljednja struktura koju ćemo pogledati je matrica.

matrix

Matrica je 2D objekt koji sadrži elemente istog tipa. Možemo je stvoriti koristeći funkciju `matrix`.

```
postava <- matrix(c('Neo', 150, 'Morpheus', 165, 'Agent Smith', 140),
             # broj redova i stupaca matrice
             nrow = 3, ncol = 2,
             # trebaju li se podaci upisivati red po red ili stupac po stupac
             # default je F
             byrow = T)
print(postava)
##      [,1]          [,2]
## [1,] "Neo"         "150"
## [2,] "Morpheus"    "165"
## [3,] "Agent Smith" "140"
```

Dimenzije matrice možemo dohvatiti funkcijom `dim`, koja je primijenjiva i na `data.frame` (ali ne i na liste).

```
dim(postava)
## [1] 3 2
```

Redovima i stupcima matrica možemo dati imena, radi lakšeg orijentiranja:

```
dimnames(postava) <- list(
    # imena redova
    c('lepi', 'pametni', 'zli'),
    # imena stupaca
    c('ime', 'visina'))
print(postava)
##         ime           visina
## lepi    "Neo"         "150"
```

```
## pametni "Morpheus"    "165"
## zli     "Agent Smith" "140"
```

Imena redova možemo dohvatiti funkcijom `rownames`, a imena stupaca funkcijom `colnames`.

```
print(rownames(postava))
## [1] "lepi"    "pametni" "zli"

print(colnames(postava))
## [1] "ime"    "visina"
```

Iste funkcije možemo koristiti i na `data.frameu`, pri čemu je kod njih na raspolaganju i funkcija `names`.

```
names(brojke_i_slova)
## [1] "brojke" "slova"  "logike"

# također, liste

names(raznoliki_objekti)
## [1] "imena"     "brojevi"   "inception"
```

Elementima možemo pristupati pomoću `[]` operatora (po istom principu kao i kod `data.framea`), ali ne i pomoću `$` operatora.

```
print(postava[2:3, 'ime'])
##       pametni           zli
##    "Morpheus" "Agent Smith"
```

Ovime ćemo završiti uvod u R te se baciti na pripremu podataka za obradu.

Reference

Adler, J. (2012). *R in a nutshell: A desktop quick reference*, 2. izdanje. O'Reilly Media, Inc.