

# Predavanje 07 – Funkcije

## Funkcije

Funkcije kot smo jih spoznali dosedaj so ukazi, ki izvršijo neko določeno nalogo. Kot smo opazili v R-ju obstajajo mnogi paketi, kot sta na primer *openxlsx* in *ggplot2*, ki razširijo delovanje R-ja z dodajanjem novih funkcij. V R-ju si lahko napišemo tudi svoje funkcije in si s tem olajšamo delo.

## Definicija funkcije

Za definicijo funkcije potrebujemo štiri stvari:

- 1) Ime funkcije: Ime s katerim bomo klicali funkcijo.
- 2) Vhod: Katere vhodne parametre potrebuje funkcija za svoje izvajanje.
- 3) Izhod: Kaj nam funkcija na koncu izvajanja vrne.
- 4) Jedro funkcije: Blok ukazov, ki se izvedejo, da iz vhoda dobimo izhod.

Vhod je lahko tudi prazen. Primer je funkcija `getwd()`, ki le izpiše trenutni direktorij na konzolo. Izhod je tudi lahko prazen oz. vrne vrednost **NULL**. Izhod vedno vrne samo en vrednost.

Funkcijo v R-ju definiramo s privzeto besedo `function` na naslednji način:

```
imeFunkcije <- function (<parametri>) {  
  #Blok kode, ki naj se izvede  
  return()  
}
```

Napišimo svojo prvo funkcijo, ki vrne niz "I love R!". Funkcijo moramo najprej pognati, kot ostale ukaze in jo nato lahko uporabljamo enako, kot že vrgajene funkcije.

```
mnenje <- function () {  
  return("I love R!")  
}
```

Poženimo funkcijo:

```
mnenje()
```

```
## [1] "I love R!"
```

Kot naj je že znano lahko izhod shranimo v spremenljivko.

```
izhod <- mnenje()  
izhod
```

```
## [1] "I love R!"
```

## Dodajanje vhodnih parametrov

V prejšnjem primeru funkcija `hello()` nima nobenega vhoda. Napišimo sedaj funkcijo, `bmi()`, ki ima dva vhodna parametra in sicer sta to višina v m in teža v kg. Na izhodu pa nam vrne indeks telesne teže.

```
bmi <- function (visina, teza) {
  return(teza / (visina / 100) ^ 2)
}
```

Testirajmo funkcijo:

```
bmi(170, 70)
```

```
## [1] 24.22145
```

Izhod funkcije lahko shranimo v spremenljivko:

```
bmi_rezultat <- bmi(170, 70)
bmi_rezultat
```

```
## [1] 24.22145
```

Poglejmo si to še na podatkih iz drugega predavanja:

```
df <- read.csv("./data_raw/osebe.csv")
df
```

```
##      spol visina teza imena
## 1      f    179   75  Micka
## 2      m    185   89  Marko
## 3      m    183   70 Gregor
## 4      m    172   80  Tomaz
## 5      f    174   58   Ana
## 6      m    185   86  Peter
## 7      f    193   73  Mojca
## 8      f    169   63  Katja
## 9      m    173   72   Anze
## 10     f    168   70   Alja
```

Ali funkcija deluje na stolpcih? Testirajmo:

```
bmi(df$visina, df$teza)
```

```
## [1] 23.40751 26.00438 20.90239 27.04164 19.15709 25.12783 19.59784 22.05805
## [9] 24.05693 24.80159
```

Funkcija deluje na vseh podatkovnih tipih, za katere R lahko izračuna jedro funkcije.

V našem primeru bi klic `bmi(df$visina, df$spol)` vrnil napako:

```
bmi(df$visina, df$spol)
```

```
## Warning in Ops.factor(teza, (visina/100)^2): '/' not meaningful for factors
## [1] NA NA NA NA NA NA NA NA NA NA
```

V stolpcu `spol` niso številke, zato ne more deliti vektorjev.

## Preverjanje pravilnosti vhoda

Vsaka funkcija ima predpostavko, da na vhod dobi točno določen tip podatkov. V primeru, da želimo to kodo deliti ali preprečiti tudi nadalnje napake v svojih skriptah je priporočljivo preveriti vrednosti in tip vhoda.

Napišimo sedaj funkcijo, `hipotenuza()`, ki ima dva vhodna parametra in sicer sta to dolžini katet pravokotnega trikotnika. Na izhodu pa nam vrne dolžino hipotenuze  $c = \sqrt{a^2 + b^2}$ .

```
hipotenuza <- function (a, b) {
  c <- sqrt(a * a + b * b)
  return(c)
}
```

Testirajmo funkcijo:

```
hipotenuza(1, 1)
```

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

```
hipotenuza(3, 7)
```

```
## [1] 7.615773
```

```
hipotenuza(-3, 2) #Vrne odgovor, čeprav je nesmiselen vhod
```

```
## [1] 3.605551
```

```
hipotenuza("a", "b") #Vrne napako, ker je tip vhoda napačen
```

```
## Error in a * a: non-numeric argument to binary operator
```

Funkcija je za lastno uporabo smiselna, če si sami pazimo na to, kaj podamo na vhod. Napišimo sedaj še funkcijo `kateta()`, ki sprejme eno dolžino katete in dolžino hipotenuze ter vrne dolžino druge katete. Dolžino manjkajoče katete izračunamo kot  $b = \sqrt{c^2 - a^2}$ .

```
kateta <- function (a, c) {
  b <- sqrt(c * c - a * a)
  return(b)
}
```

Testirajmo funkcijo:

```
kateta(1, sqrt(2))
```

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

```
kateta(3, 7)
```

```
## [1] 6.324555
```

```
kateta(7, 3) #Vrne opozorilo, da funkcija ne zna narediti izračuna
```

```
## Warning in sqrt(c * c - a * a): NaNs produced
```

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

```
kateta(-1, 2) #Vrne odgovor, čeprav je nesmiselen vhod
```

```
## [1] 1.732051
```

```
kateta("a", "b") #Vrne napako, ker je tip vhoda napačen
```

```
## Error in c * c: non-numeric argument to binary operator
```

V primeru `kateta(7, 3)` dobimo le opozorilo, čeprav je izračun vrednost **NaN** (Not A Number). Kar je R-jev znak za definicijo neizračunljivih vrednosti, kot naprimer  $0/0$ .

Poskušajmo to funkcijo razširiti in popraviti, tako da bo zaznala te težave. V prvem primeru smo funkciji na vhod podali kateto, ki je daljša od hipotenuze, kar ni smiselno. V tem primeru še vedno lahko predpostavimo, da je prvi parameter hipotenuza in še vedno opravimo izračun.

```
kateta <- function (a, c) {
  if (c > a) {
    b <- sqrt(c * c - a * a)
  }else{
    warning("Kateta je daljša od hipotenuze. Menjam vrednosti.")
    b <- sqrt(a * a - c * c)
  }
  return(b)
}
```

```
kateta(1, sqrt(2))
```

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

```
kateta(3, 7)
```

```
## [1] 6.324555
```

```
kateta(7, 3) #Vrne pravilo vrednost in opozorilo
```

```
## Warning in kateta(7, 3): Kateta je daljša od hipotenuze. Menjam vrednosti.
```

```
## [1] 6.324555
```

V zadnjem primeru nam sedaj funkcija naredi pravilni izračun, hkrati pa si še izpišemo opozorilo s funkcijo `warning()`. Funkcija `warning()` nam opozorila izpiše na koncu izvajanja funkcije, lahko pa si jih ogledamo tudi, če v konzolo napišemo `warning()`.

Dodajmo še preverjanje negativnih vrednosti. V tem primeru, bi želeli, da se funkcija zaustavi in nam vrne napako. To lahko naredimo z ukazom `stop()`, ki zaustavi funkcijo in nam vrne željeno sporočilo napake.

```
kateta <- function (a, c) {
  if (a <= 0 | c <= 0) {
    stop("Negativne vrednosti stranic!")
  }
  if (c > a) {
    b <- sqrt(c * c - a * a)
  }else{
    warning("Kateta je daljša od hipotenuze. Menjam vrednosti.")
    b <- sqrt(a * a - c * c)
  }
  return(b)
}
```

Testirajmo:

```
kateta(1, sqrt(2))
```

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

```
kateta(3, 7)
```

```
## [1] 6.324555
```

```
kateta(7, 3) #Vrne pravilno vrednost in opozorilo
```

```
## Warning in kateta(7, 3): Kateta je daljša od hipotenuze. Menjam vrednosti.
```

```
## [1] 6.324555
```

```
kateta(-1, 2) #Vrne napako skupaj z našim opisom napake
```

```
## Error in kateta(-1, 2): Negativne vrednosti stranic!
```

V zadnjem primeru nam R vrne našo napako in njen pripadajoči opis, da lahko v tem primeru hitreje odkrijemo kaj je izvor napake, če se nam to pripeti v prihodnosti.

Nekatere napake R zazna že sam, tako nam klic `kateta("a", "b")` že sam vrne napako, lahko pa ga poskušamo zaznati tudi sami. Za to lahko uporabimo funkcijo `is.numeric()`.

```
kateta <- function (a, c) {  
  if (!(is.numeric(a) & is.numeric(c))) {  
    stop("Na vhodu morajo biti številke.")  
  }  
  if (a <= 0 | c <= 0) {  
    stop("Negativne vrednosti stranic!")  
  }  
  if (c > a) {  
    b <- sqrt(c * c - a * a)  
  }else{  
    warning("Kateta je daljša od hipotenuze. Menjam vrednosti.")  
    b <- sqrt(a * a - c * c)  
  }  
  return(b)  
}
```

Testirajmo:

```
kateta(1, sqrt(2))
```

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

```
kateta(3, 7)
```

```
## [1] 6.324555
```

```
kateta(7, 3) #Vrne pravilno vrednost in opozorilo
```

```
## Warning in kateta(7, 3): Kateta je daljša od hipotenuze. Menjam vrednosti.
```

```
## [1] 6.324555
```

```
kateta(-1, 2) #Vrne napako skupaj z našim opisom napake
```

```
## Error in kateta(-1, 2): Negativne vrednosti stranic!
```

```
kateta("a", "b") #Vrne napako, ker je tip vhoda napačen
```

```
## Error in kateta("a", "b"): Na vhodu morajo biti številke.
```

Obstaja še ena možnost za napako in sicer, če pozabimo podati vhodne parametre. V tem primeru, dobimo napako, ki nam sporoči, da ta parameter manjka.

```
kateta(1)
```

```
## Error in kateta(1): argument "c" is missing, with no default
```

Če želimo, da se naša funkcija lahko kliče tudi z manj parametri lahko dodamo privzete vrednosti tem parametrom. Pazimo, da so vsi obvezni parametri na začetku. Edina sprememba sedaj je v glavi funkcije `function (a = 1, c = 2)`.

```
kateta <- function (a = 1, c = 2) {
  if (!(is.numeric(a) & is.numeric(c))) {
    stop("Na vhodu morajo biti številkke.")
  }
  if (a <= 0 | c <= 0) {
    stop("Negativne vrednosti stranic!")
  }
  if (c > a) {
    b <- sqrt(c * c - a * a)
  } else {
    warning("Kateta je daljša od hipotenuze. Menjam vrednosti.")
    b <- sqrt(a * a - c * c)
  }
  return(b)
}
```

```
kateta(0.5) #izračun za klica kateta(0.5, 2)
```

```
## [1] 1.936492
```

```
kateta() #izračun za klica kateta(0.5, 1)
```

```
## [1] 1.732051
```

## Vračanje vrednosti funkcije

Vračanje vrednosti funkcije se lahko zgodi kjerkoli v funkciji. Po ukazu `return()` se izvajanje funkcije takoj zaključi. To lahko uporabimo za implementacijo svoje funkcije, ki lahko shrani naše podatke v *csv* ali *xlsx* obliki. Ta funkcija si bo pomagala tudi z ukazom `paste()`, ki združi vse parametre v en niz in med njih vrine znak podan s parametrom `sep`. Primer:

```
paste("A", "B", sep = "-")
```

```
## [1] "A-B"
```

```
paste("A", 2, 3, sep = ".")
```

```
## [1] "A.2.3"
```

Napišimo sedaj željeno funkcijo:

```
shraniPodatke <- function (podatki, datoteka, koncnica = "csv") {
  library(openxlsx)
  celotno_ime <- paste(datoteka, koncnica, sep = ".")
  if (koncnica == "csv") {
    write.csv2(podatki, file = celotno_ime)
    return()
  }
  if (koncnica == "xlsx") {
    write.xlsx(podatki, celotno_ime)
    return()
  }
  warning("Datoteka ni shranjena. Neznana končnica.")
  return()
}
```

```
wd <- getwd()
datoteka <- paste(wd, "osebe", sep = "/")
shraniPodatke(df, datoteka)

## NULL

shraniPodatke(df, datoteka, koncnica = "xlsx")

## NULL
```

## Območje spremenljivk

V prejšnjih funkcijah smo uporabljali spremenljivke `a`, `b` in `c`. Te spremenljivke so dostopne samo znotraj funkcije in enaka imena spremenljivk lahko uporabljamo zunaj funkcije. Prikažimo na preprostem primeru, ki uporablja dve spremenljivki `x` in `y`.

```
podvoji <- function (x) {
  print(x)
  print(y)
  y <- x * 2
  print(x)
  print(y)
  return (y)
}
x <- 7
y <- 8
print("Pred klicem")

## [1] "Pred klicem"

print(x)

## [1] 7

print(y)

## [1] 8

print("Med klicem")

## [1] "Med klicem"

x <- podvoji(x)

## [1] 7
## [1] 8
## [1] 7
## [1] 14

print("Po klicu")

## [1] "Po klicu"

print(x)

## [1] 14

print(y)

## [1] 8
```

Pred klicem funkcije vidimo, da sta vrednosti  $x = 7$  in  $y = 8$ , kar je pričakovano. Takoj znotraj funkcije podvoji sta vrednosti še vedno enaki  $x = 7$  in  $y = 8$ . Tukaj je vredost  $y$  dosegljiva čeprav je nismo podali funkciji. Funkcija lahko dostopa tudi do vseh spremenljivk, ki so nastale v workspacu, ki je to funkcijo klical. Čeprav to deluje se odsvetuje uporaba teh spremenljivk, ker je koda v takem primeru zelo nerazumljiva.

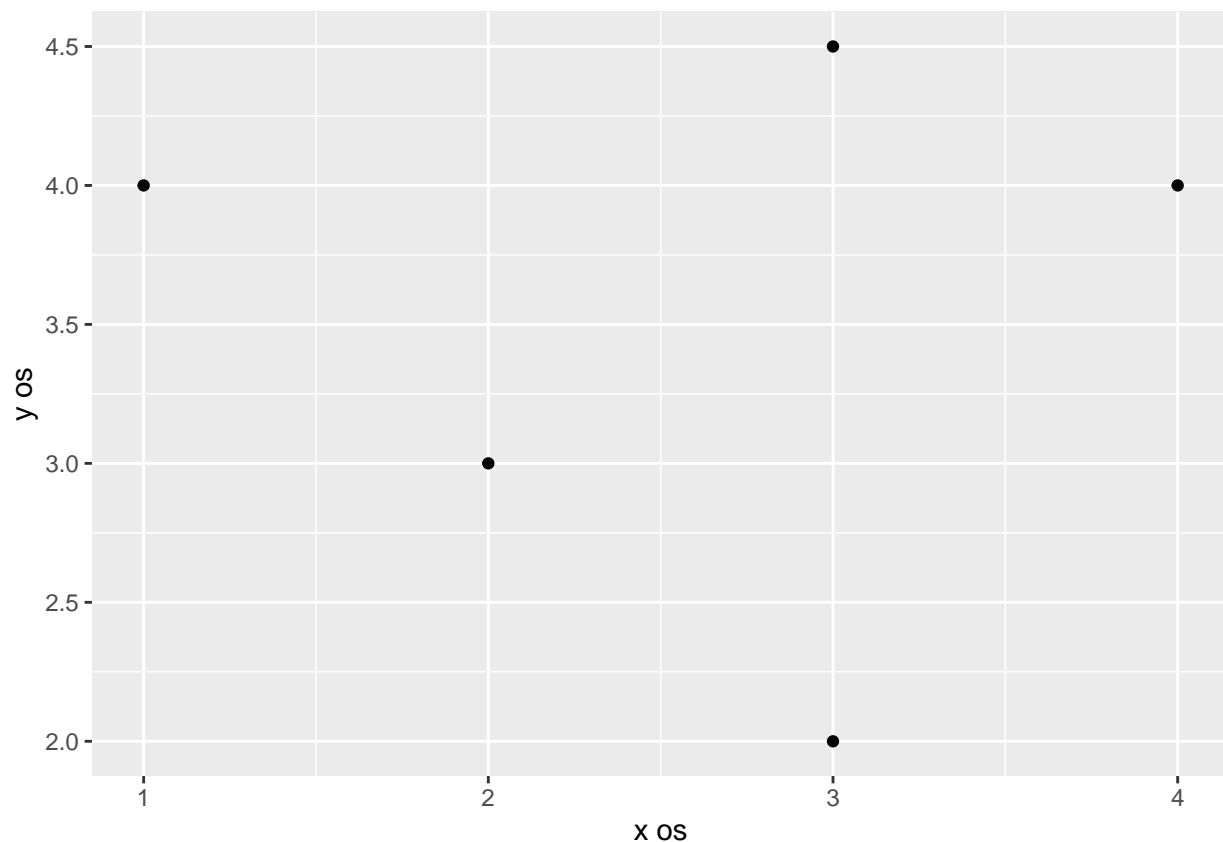
Pri drugem izpisu se je vrednost  $y$  spremenila na 14. Po klicu pa spet velja  $y = 8$ . To je zato, ker ima vsaka funkcija svoje delovno območje in naredi kopijo spremenljivke  $y$ . Dobra lastnost tega je, da vsaka funkcija deluje lokalno in z ukazi znotraj funkcije nikoli ne boste spremenili vrednosti zunaj nje. Edina sprememba, ki se na koncu pozna je  $x = 14$ , saj to vrednost funkcija dejansko vrne in jo spremenljivki  $x$  priredimo zunaj funkcije.

## Domača naloga

- 1) Napišite funkcijo `razsevni_diagram(x, y, labelx, labely)`, ki zna z uporabo funkcije `ggplot()` izrisati razsevni diagram teh točk. Če label ne podamo sta privzeti imeni  $x$  in  $y$ .

Primer:

```
razsevni_diagram(c(1, 2, 3, 4, 3), c(4, 3, 2, 4, 4.5), "x os", "y os")
```



- 2) Dopolnite funkcijo `hipotenuza(a,b)` tako, da bo varnejša. Preverite, da je na vhodu res pozitivno število.
- 3) Napišite funkcijo `nalozi_priljubljene_pakete()`, s katero boste lahko v prihodnje lažje naložili vse pakete, ki jih uporabljate. Npr.: *ggplot2*, *openxlsx*, *tidyr*