Funkce v R
a
R jako programovací jazyk
—
Úvod do jazyka R

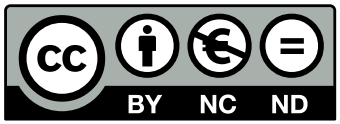
Lubomír Štěpánek[1, 2]

[1]Oddělení biomedicínské statistiky & výpočetní techniky
Ústav biofyziky a informatiky
1. lékařská fakulta
Univerzita Karlova v Praze

ČVUT
FBMI

[2]Katedra biomedicínské informatiky
Fakulta biomedicínského inženýrství
České vysoké učení technické v Praze

(2017) Lubomír Štěpánek, CC BY-NC-ND 3.0 (CZ)



Dílo lze dále svobodně šířit, ovšem s uvedením původního autora a s uvedením původní licence. Dílo není možné šířit komerčně ani s ním jakkoliv jinak nakládat pro účely komerčního zisku. Dílo nesmí být jakkoliv upravováno. Autor neručí za správnost informací uvedených kdekoliv v předložené práci, přesto vynaložil nezanedbatelné úsilí, aby byla uvedená fakta správná a aktuální, a práci sepsal podle svého nejlepšího vědomí a svých „nejlepších“ znalostí problematiky.

Obsah

Vestavěné funkce

- většina funkcionality R je dána vestavěnými funkcemi
- ty jsou optimalizované, odladěné
- je-li to možné, je vhodné je preferovat před uživatelem definovanými funkcemi

Uživatelem definované funkce

- pro unikátnější operace je možné definovat vlastní funkce
- obecná syntaxe

```
1   nazevFunkce <- function(
2       argument_1,
3       argument_2,
4       ...
5   ){
6
7       # '''
8       # komentář
9       # '''
10
11      procedura s argumenty
12      return(vystup)
13
14  }
```

Uživatelem definované funkce

- například

```
1  sectiCtverce <- function(a, b){
2
3    # '''
4    # vrací součet čtverců čísel "a" a "b"
5    # '''
6
7    return(a ^ 2 + b ^ 2)
8
9  }
```

Scoping

- scopingem rozumíme prostor, ze kterého se v dané situaci a za daných okolností vybírají proměnné k výpočtu
- v R obecně *lexikální scoping*
  - proměnné použité (ale nedefinované) ve funkcích se vybírají z mateřského prostředí (o jeden level výš)
- například

```
1  x <- 1
2  f <- function(){x}
3  g <- function(){
4          x <- 0
5          f()
6      }
7  g()   # 1
```

Scoping

- další příklad

```
1        x <- 1
2        f <- function(x){return(2 * x)}
3
4        f(x = 5)   # 10
5
6        x          # 1
```

- proměnné jsou tedy vnímány jako zástupné symboly, podobně jako ve středoškolské matematice
  - uživatelsky velmi přívětivé

Lokální, globální proměnné

- koncept globálních proměnných se v R vůbec nedoporučuje používat, protože porušuje paradigma lexikálního scopingu
- přesto je možné zavést globální proměnnou přiřazením typu „«-“

```
1 x <<- 1   # x je globální proměnnou
```

- ostatní proměnné jsou „lokální“

Podmínka *když*

- též zvaná *if-statement*
- rozhodovací ovládací prvek založený na pravdivosti, či nepravdivosti nějakého výroku
- obecná syntaxe

```
1 ||      if(výrok){
2 ||        procedura v případě, že výrok je TRUE
3 ||      }else{
4 ||        procedura v případě, že výrok je FALSE
5 ||      }
```

Podmínka *když*

- například

```
1  if(x == 1){
2    print("x je rovno 1")
3  }
4
5  # anebo
6  if(x == 1){
7    print("x je rovno 1")
8  }else{
9    print("x není rovno 1")
10 }
```

for() cyklus

- ovládací struktura, smyčka pro opakování procedury stejného charakteru
- vhodná tehdy, **víme-li** dopředu počet opakování (iterací) dané procedury
- obecná syntaxe

```
1        for(indexový prostor){
2
3          procedura pro každý prvek indexového
4          prostoru
5
6        }
```

for() cyklus

- například

```
1   for(i in 1:5){
2
3     print(i)
4
5   }
6
7   # anebo
8   for(my_letter in letters){
9
10    print(
11      paste(my_letter, "je fajn", sep = " ")
12    )
13
14  }
```

while() cyklus

- ovládací struktura, smyčka pro opakování procedury stejného charakteru
- vhodná tehdy, **nevíme-li** dopředu počet opakování (iterací) dané procedury
- obecná syntaxe

```
index <- 1
while(výrok){

  procedura pro každý index,
  dokud je výrok TRUE

  index <- index + 1

}
```

while() cyklus

- například

```
1   i <- 1
2   while(i <= 5){
3     print(i)
4     i <- i + 1
5   }
6
7   # anebo
8   my_letters <- letters
9   while(length(my_letters) > 0){
10
11    print(
12      paste(my_letters[1], "je fajn", sep = " ")
13    )
14    my_letters <- my_letters[-1]
15
16  }
```

repeat-until cyklus

- ovládací struktura, smyčka pro opakování procedury stejného charakteru
- vhodná tehdy, **nevíme-li** dopředu počet opakování (iterací) dané procedury
- prakticky ekvivalentní `while()` cyklu
- obecná syntaxe

```
1   index <- 1
2   while(TRUE){
3
4     if(zastavovací podmínka){break}
5
6     procedura pro každý index
7
8     index <- index + 1
9
10  }
```

repeat-until cyklus

- například

```
1  i <- 1
2  while(TRUE){
3    if(i == 5){break}
4    print(i)
5    i <- i + 1
6  }
7
8  # anebo
9  my_letters <- letters
10 while(TRUE){
11   if(length(my_letters) == 0){break}
12   print(
13     paste(my_letters[1], "je fajn", sep = " ")
14   )
15   my_letters <- my_letters[-1]
16 }
```

Varování

- textová hláška vrácená funkcí nebo procedurou
- nejde o maligní chybu

```
log(-5)  # NaN; In log(-5) : NaNs produced
```

- lze zavést i vlastní, například

```
logaritmuj <- function(x){
  # vrací přirozený logaritmus čísla "x"
  if(x <= 0){
    print(
      "x je nekladné, bude vráceno NaN"
    )
  }
  return(suppressWarnings(log(x)))
}

logaritmuj(-5)  # NaN; x je nekladné, bude vráceno NaN
```

Chyby

- ochranný mechanismus funkcí, který zabrání dalšímu provádění kódu

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

- lze zavést i vlastní, například

```
sectiCtverce <- function(a, b){
  # '''
  # vrací součet čtverců čísel "a" a "b"
  # '''
  if(!is.numeric(a)){stop("a musí být číslo!")}
  if(!is.numeric(b)){stop("b musí být číslo!")}
  return(a ^ 2 + b ^ 2)
}

sectiCtverce(1, 2)    # 5
sectiCtverce(1, "2")  # Error: b musí být číslo!
```

Rodina funkcí apply()

- jde o funkce dobře optimalizované tak, že v rámci svého vnitřního kódu „co nejdříve“ volají C++ ekvivalenty R-kové funkce
- díky tomu jsou exekučně rychlé
- nejužitečnější je `apply()` a `lapply()`

```
1  # vrací průměry nad všemi sloupci "mtcars"
2  x <- apply(mtcars, 2, mean)
3
4  # méně šikovně to samé
5  x <- NULL
6  for(i in 1:dim(mtcars)[2]){
7    x <- c(x, mean(mtcars[, i]))
8  }
9  names(x) <- colnames(mtcars)
```

Funkce apply()

- vrací vektor výsledků funkce `FUN` nad maticí či datovou tabulkou `X`, kterou čte po řádcích (`MARGIN = 1`), nebo sloupcích (`MARGIN = 2`)
- syntaxe je `apply(X, MARGIN, FUN, ...)`

```
1  apply(mtcars, 2, mean)
2
3  my_start <- Sys.time()
4  x <- apply(mtcars, 2, mean)
5  my_stop <- Sys.time(); my_stop - my_start # 0.019s
6
7  my_start <- Sys.time()
8  x <- NULL
9  for(i in 1:dim(mtcars)[2]){
10   x <- c(x, mean(mtcars[, i]))
11   names(x)[length(x)] <- colnames(mtcars)[i]
12 }
13 my_stop <- Sys.time(); my_stop - my_start # 0.039s
```

Funkce `lapply()`

- vrací list výsledků funkce `FUN` nad vektore či listem `X`
- syntaxe je `lapply(X, FUN, ...)`
- **skvěle se hodí pro přepis `for()` cyklu do vektorizované podoby!**
- vhodná i pro adresaci v listu

```
1   set.seed(1)
2   my_long_list <- lapply(
3       sample(c(80:120), 100, TRUE),
4       function(x) sample(
5           c(50:150), x, replace = TRUE
6       )
7   )   # list vektorů náhodné délky
8       # generovaných z náhodných čísel
9
10  lapply(my_long_list, "[[", 14)
11      # z každého prvku listu (vektoru)
12      # vybírám jen jeho 14. prvek
```

Náhrada for cyklu funkcí lapply()

- obě procedury jsou ekvivalentní stran výstupu, `lapply()` je významně rychlejší

```
1  # for cyklus
2  x <- NULL
3  for(i in 1:N){
4    x <- c(x, FUN)
5  }
6
7  # lapply
8  x <- unlist(
9    lapply(
10     1:N,
11     FUN
12   )
13 )
```

Náhrada for cyklu funkcí lapply()

```r
# for cyklus
my_start <- Sys.time()

for_x <- NULL
for(i in 1:100000){for_x <- c(for_x, i ^ 5)}

my_stop <- Sys.time(); my_stop - my_start # 18.45s

# lapply
my_start <- Sys.time()

lapply_x <- unlist(lapply(
  1:100000, function(i) i ^ 5
))

my_stop <- Sys.time(); my_stop - my_start # 0.10s
```

Literatura


ZVÁRA, Karel. *Základy statistiky v prostředí R*.
Praha, Česká republika: Karolinum, 2013. ISBN 978-80-246-2245-3.


WICKHAM, Hadley. *Advanced R*.
Boca Raton, FL: CRC Press, 2015. ISBN 978-1466586963.

Děkuji za pozornost!

lubomir.stepanek@lf1.cuni.cz
lubomir.stepanek@fbmi.cvut.cz