3 Programowanie w R

3.1 Funkcje

- Korzystając z programu R, bardzo szybko odczuwa się potrzebę użycia pewnych fragmentów kodu wielokrotnie, choć być może dla różnych danych.
- Tak jak listy grupują obiekty (być może różnych typów), tak funkcje zbierają określone wyrażenia służące np. do obliczenia pewnych wartości dla zadanych danych.
- Dodatkową zaletą stosowania funkcji jest możliwość dzielenia długiego kodu na łatwiejsze do opanowania części.

- Tworzenie obiektów typu funkcja odbywa się według następującej składni

```
function(lista parametrów) ciało funkcji
```

gdzie `ciało funkcji` jest wyrażeniem do wykonania na obiektach określonych przez `listę parametrów` .

- Wartość obliczonego wyrażenia jest wynikiem działania funkcji. Takim wynikiem może być jeden i tylko jeden obiekt, np. lista.
- Parametrów może być jednak wiele. `lista parametrów` to ciąg oddzielonych przecinkami elementów postaci:
  - `nazwa` parametru (pod taką nazwą będzie dostępny w funkcji obiekt przekazany przy wywołaniu),
  - `nazwa = wyrażenie` (parametr z wartością domyślną),
  - `...` - parametr specjalny, który pozwala przekazać dowolną liczbę argumentów w grupie.

```
szescian <- function(x) x^3 # funkcje zazwyczaj się nazywa
szescian(2)
```

```
## [1] 8
```

```r
szescian_2 <- function(x, y) {
  x3 <- x^3
  y3 <- y^3
  return(c(x3, y3))
}
szescian_2(2, 3) # lub szescian_2(x = 2, y = 3)
```

```
## [1]  8 27
```

```r
szescian_3 <- function(x = 2, y = 2) {
  x3 <- x^3
  y3 <- y^3
  return(c(x3, y3))
}
szescian_3()
```

```
## [1] 8 8
```

```r
szescian_3(y = 3)
```

```
## [1]  8 27
```

```r
str(lapply(list(1, 2, 3), function(x) x^3))
```

```
## List of 3
##  $ : num 1
##  $ : num 8
##  $ : num 27
```

```r
szescian_4 <- function(x) {
    if (!is.numeric(x)) {
      stop("non-numeric argument x")
    }
    x^3
}
szescian_4(-3)
## [1] -27
szescian_4("a")
## Error in szescian_4("a") : non-numeric argument x
```

3.2 Instrukcje warunkowe

- Wyrażenie warunkowe `if` ma następującą składnię:

```r
if (warunek) wyrazenieTRUE else wyrazenieFALSE
```

- Przykładowo:

```r
if (is.numeric("wyrazenie")) {
  print("wyrazenieTRUE")
} else {
  print("wyrazenieFALSE")
}
```

```
## [1] "wyrazenieFALSE"
```

- W celu agregacji wektorów logicznych, można wykorzystać dwie ważne funkcje agregujące wartości logiczne: `all()` i `any()`, które zwracają wartość `TRUE` wtedy i tylko wtedy, gdy odpowiednio wszystkie lub co najmniej jedna wartość w wektorze równa jest `TRUE`.

```r
x <- 1:5
any(x < 2)
```

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

```
all(x < 2)
```

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

```
all(x == seq(1, 5))
```

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

3.3 Pętle

- Pętle umożliwiają wielokrotne wykonywanie tego samego wyrażenia (choć zapewne na różnych obiektach). W programie R mamy do dyspozycji pętle:
  - `while`
  - `repeat`
  - `for`
- Składnia pętli `while` jest następująca:

```
while (warunek) wyrazenie
```

- Zadaniem pętli `while` jest obliczanie `wyrazenia` dopóty, dopóki `warunek` jest spełniony.

```
i <- 1
while (i <= 3) {
  print(i)
  i <- i + 1
}
```

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

- Aby pętla nie wykonywała się nieskończoną liczbę razy, zazwyczaj `warunek` będzie konstruowany na danych odczytywanych z pewnego obiektu, który jest modyfikowany za pomocą `wyrazenia` .
- Może się zdarzyć, że `warunek` testowy nigdy nie będzie spełniony i wtedy liczba wykonanych obrotów pętli będzie równa zeru.

- Pętla `repeat` zachowuje się tak jak `while` z `warunkiem` testowym na stałe ustawionym na `TRUE` . Zatem należy zawsze pamiętać o wywołaniu `break` , o ile chcemy doczekać wyniku.

```
i <- 0
repeat {
  i <- i + 1
  print(i)
  if (i == 3) break
}
```

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

- Pętla `for` jest chyba najczęściej stosowaną pętlą w programie R. Szczególnie nadaje się ona do ,,przechodzenia" po elementach wektora atomowego lub listy bądź też wykonywania ciągu wyrażeń zadaną liczbę razy. Jej składnia jest następująca:

```
for (nazwa in wektor) wyrazenie
```

- W pętli `for` każdą kolejną (od pierwszej do ostatniej) wartość `wektora` związujemy z podaną `nazwą` i obliczamy `wyrażenie` . Pętla ta wykonuje się zawsze dokładnie `length(wektor)` razy, o ile nie użyte zostało wyrażenie `break` .

```
for (i in 1:3) print(i)
```

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

3.4 Zadania

Zadanie 1. Oblicz iloczyn elementów dowolnego wektora `x` za pomocą pętli `while`, `repeat` i `for` (każdej z osobna).

```
# dla
x <- 1:5
```

```
## [1] 120
```

Zadanie 2. Ile liczb postaci $\binom{n}{r}$ jest większych od miliona dla $1 \leqslant r \leqslant n \leqslant 100$?

```
## [1] 4075
```

Zadanie 3. Napisz funkcję, która sprawdza czy wektor jest palindromem.

```
# dla
x <- c(1, 2, 3, 3, 2, 1)
```

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

```
# dla
x <- c(1, 2, 3, 3, 2, 2)
```

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

Zadanie 4. Napisz funkcję zamieniającą miarę kąta podaną w stopniach na radiany. Sprawdź działanie tej funkcji dla kątów o mierze: 0°, 30°, 45°, 60°, 90°. Następnie przygotuj ramkę danych, w której zebrane będą informacje o wartościach funkcji sinus, cosinus, tangens i cotangens dla kątów o takich miarach.

```
## [1] 0.0000000 0.5235988 0.7853982 1.0471976 1.5707963
```

```
##         sin          cos           tg          ctg
## 1 0.0000000 1.000000e+00 0.000000e+00          Inf
## 2 0.5000000 8.660254e-01 5.773503e-01 1.732051e+00
## 3 0.7071068 7.071068e-01 1.000000e+00 1.000000e+00
## 4 0.8660254 5.000000e-01 1.732051e+00 5.773503e-01
## 5 1.0000000 6.123234e-17 1.633124e+16 6.123234e-17
```

Zadanie 5. Napisz funkcję, której argumentem będzie wektor liczbowy a wynikiem wektor zawierający trzy najmniejsze i trzy największe liczby w tym wektorze. W przypadku argumentu krótszego niż trzy liczby, funkcja ma zwracać komunikat o błędzie z komentarzem ,,za krótki argument".

```
# dla
x <- c(2, 6, 1, 5, 7, 3, 4)
```

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

```
# dla
x <- c(2, 6)
## Error in command 'extreme_3(x)': za krótki argument
```