Introdução ao R: aula 6

Alexandre Rademaker

EMAp/FGV

February 5, 2011

Blocos

```
> a <- {
      x <- 10
      y <- 20
 }
> a
```

```
[1] 20
```

Atribuições retornam o valor atribuído. Blocos retornam o valor da última expressão. Em R, um bloco é também uma expressão assim como atribuição. Por isso também podemos escrever:

```
> z <- w <- 10
> c(z, w)
```

```
[1] 10 10
```

Condicional

```
> x <- 3
> if (x > 2) y <- 2 * x else y <- 3 * x
> if (x > 2) {
      y <- 2 * x
  } else {
      y <- 3 * x
  }
```

Condicional

```
> x <- rnorm(5, sd = 10)
> ifelse(x < 0, "-", "+")
[1] "+" "-" "+" "-" "-"
```

Condicional

```
> vs <- rnorm(10)
> switch(1, mean(vs), median(vs), sum(vs))

[1] -0.1160390

> semaforo <- "verde"
> switch(semaforo, verde = "continua", amarelo = "acelera",
      vermelho = "para")

[1] "continua"
```

Repetição

```
> Fibonacci <- numeric(12)
> Fibonacci[1] <- Fibonacci[2] <- 1
> for (i in 3:12) Fibonacci[i] <- Fibonacci[i -
    2] + Fibonacci[i - 1]
> Fibonacci
 [1]   1   1   2   3   5   8  13  21  34  55  89
[12] 144
```

Repetição

```
> x <- rnorm(10)
> k <- 0
> for (v in x) {
      if (v > 0)
          y <- v
      else y <- 0
      k <- k + y
 }
> k
[1] 3.170696
> k <- sum(x[x > 0])
```

Repetição

```
> Fib1 <- 1
> Fib2 <- 1
> Fibonacci <- c(Fib1, Fib2)
> while (Fib2 < 100) {
      Fibonacci <- c(Fibonacci, Fib2)
      oldFib2 <- Fib2
      Fib2 <- Fib1 + Fib2
      Fib1 <- oldFib2
 }
> Fibonacci
 [1]  1  1  1  2  3  5  8 13 21 34 55 89
```

Repetição

```
> texto <- c()
> repeat {
      cat("Introduza uma frase ? (frase vazia termina) ")
      fr <- readLines(n = 1)
      if (fr == "")
          break
      else texto <- c(texto, fr)
 }
```

Repetição

```
> repeat {
      cat("Introduza um nro positivo ? (zero termina) ")
      nro <- scan(n = 1)
      if (nro < 0)
          next
      if (nro == 0)
          break
      pos <- c(pos, nro)
 }
```

Funções

```
> hipotenusa <- function(a, b) {
      h <- sqrt(a^2 + b^2)
      h
  }
> hipotenusa(4, 3)

[1] 5

> hipotenusa(c(4, 5, 6), c(3, 4, 5))

[1] 5.000000 6.403124 7.810250
```

Usando o `edit()`.

λ: funções sem nome!

```
> (function(x) {
      y <- 10
      x * y
  })(10)
```

```
[1] 100
```

Funções: argumentos

```
> fx <- function(a, b = 10) sqrt(a^2 + b^2)
> fx(12)

[1] 15.6205

> fx <- function(a, b = 10) c(a, b, sqrt(a^2 +
      b^2))
> fx(10)

[1] 10.00000 10.00000 14.14214

> fx(b = 1, a = 12)

[1] 12.00000  1.00000 12.04159
```

Escopo

```
> x <- 5
> f <- function() {
      x <- 6
      x
  }
> f()
[1] 6
> x
[1] 5
```

Escopo

```
> x <- 5
> f <- function() print(x)
> f()
```

```
[1] 5
```

```
> rm(x)
```

O que ocorre se executarmos `f()` agora?

Lazy evaluation

```
> f1 <- function(a1, a2 = sqrt(a1)) {
      a1 <- a1^2
      a2
  }
> f1(4)
```

```
[1] 4
```

Lazy evaluation

```
> f2 <- function(a1, a2 = sqrt(a1)) {
      z <- a2/a1
      a1 <- a1^2
      a2
  }
> f2(4)
```

```
[1] 2
```

Exercício

Escrever a função para cálculos das raizes de uma função quadrática.
Testes:

```
> args(quadratic)

function (a, b, c)
NULL

> quadratic(1, -5, 6)

     [,1] [,2]
[1,]    2    3
```

Exercício: vetores como argumentos

Mas a função também deve receber listas de valores em cada argumento:

```
> quadratic(1, c(-5, 1), 6)
```

```
                  [,1]                [,2]
[1,]  2.0+0.000000i  3.0+0.000000i
[2,] -0.5-2.397916i -0.5+2.397916i
```

Exercício: resposta

```
> quadratic
function (a, b, c)
{
    rad <- b^2 - 4 * a * c
    if (is.complex(rad) || all(rad >= 0)) {
        rad <- sqrt(rad)
    }
    else {
        rad <- sqrt(as.complex(rad))
    }
    cbind(-b - rad, -b + rad)/(2 * a)
}
```