Insper EDUCAÇÃO EXECUTIVA ONLINE

Live Learning AO VIVO

Programação para Não Programadores

Aula 3

Prof. Magno Severino e Prof. Marina Muradian

20/04/2021

Objetivos de aprendizagem

- Compreender o controle do fluxo de execução de comandos.
- Aplicar comandos de repetição.
- Criar funções personalizadas.

Controle de fluxo e repetição

Por algum motivo particular, você pode querer executar ou não um determinado trecho do seu código. Também, pode querer que um trecho seja executado repetidas vezes.

`if` e `else`

A maneira mais simples de controlar o fluxo de execução é adicionando um valor lógico e executando o comando `if`.

```
if(TRUE) message("Isso é verdadeiro!")

if(FALSE) message("Isso não é verdadeiro!")

if(is.na(NA)) message("O valor está faltando!")

if(runif(1) > 0.5) message("Essa mensagem ocorre com 50% de chance.")

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

Para executar um comando caso a condição do `if` não seja verdadeira, utilize o `else`.

```
if(FALSE)
{
   message("Não será executado...")
} else #o else deve estar na mesma linha que fecha as chaves
{
   message("mas este será.")
}
```

Como R é uma linguagem vetorizada, é possível vetorizar estruturas de controle de fluxo, através da função `ifelse`. Esta função necessita de três argumentos: o primeiro é um vetor de valor lógico, o segundo contém os valores que serão retornados quando a condição lógica é verdadeira (`TRUE`), o terceiro contem os valores que serão retornados quando a condição é falsa (`FALSE`).

```
set.seed(123)
numeros <- sample(1:100, 10)

maiores_50 <- ifelse(numeros > 50, "Maior que 50", "Menor ou igual a 50")

maiores_50 <- factor(maiores_50)
table(maiores_50)
```

```
## maiores_50
##        Maior que 50 Menor ou igual a 50
##                   4                   6
```

Estruturas de repetição

while

O `while` primeiramente checa a condição e, caso verdadeira, a executa. No exemplo abaixo, o `while` será executado somente se `acao` for diferente de "Descansar".

```
acoes <- c( "Aprender a programar em R",
        "Fazer um café",
        "Assistir um filme",
        "Descansar")

acao <- sample(acoes, 1)

print(acao)
```

```
## [1] "Fazer um café"
```

```
while(acao != "Descansar") {
  acao <- sample(acoes, 1)
  print(acao)
}
```

```
## [1] "Assistir um filme"
## [1] "Descansar"
```

for

É usado quando se sabe exatamente quantas vezes o trecho de código deverá ser repetido. O `for` aceita uma variável iterativa bem como um vetor. O loop é repetido, dando ao iterador cada elemento do vetor por vez. O caso mais simples é um vetor contendo inteiros.

```
for(i in 1:5) {
  j <- i ^ 2
  print(paste("j = ", j))
}
```

```
## [1] "j =  1"
## [1] "j =  4"
## [1] "j =  9"
## [1] "j =  16"
## [1] "j =  25"
```

```
for(month in month.name)
{
 print(paste0("The month of ", month))
}
```

```
## [1] "The month of January"
## [1] "The month of February"
## [1] "The month of March"
## [1] "The month of April"
## [1] "The month of May"
## [1] "The month of June"
## [1] "The month of July"
## [1] "The month of August"
## [1] "The month of September"
## [1] "The month of October"
## [1] "The month of November"
## [1] "The month of December"
```

No exemplo abaixo, sorteamos 10 números entre 1 e 100 e, para cada número, se ele for par, calculamos o quadrado, se for impar, somamos 1.

```
set.seed(2)

numeros <- sample(1:100, 10)
resultado <- numeric(length(numeros))
for(i in 1:length(numeros)) {
  if(numeros[i] %% 2 == 0)
    resultado[i] <- numeros[i]^2
  else
    resultado[i] <- numeros[i]+1
}

rbind(numeros, resultado)
```

```
##           [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## numeros     85   79   70    6   32    8   17   93   81    76
## resultado   86   80 4900   36 1024   64   18   94   82  5776
```

Uma alternativa é usar a função `ifelse` para fazer a mesma operação acima.

```
ifelse(numeros %% 2 == 0, numeros^2, numeros+1)
```

```
##  [1]   86   80 4900   36 1024   64   18   94   82 5776
```

Funções

Os tipos e estruturas de variáveis são importantes para armazenamento de dados, funções nos permitem processar os dados.

Criando e chamando funções

Uma função é criada de maneira parecida com a criação de variáveis, atribuindo um nome à função.

```
hipotenusa <- function(x, y) {
  sqrt(x^2 + y^2)
}
```

Aqui, `hipotenusa` é o nome da função criada, `x` e `y` são os argumentos e o conteúdo entre chaves é chamado de corpo da função.

```
hipotenusa(3, 4) #sem nomear, os parametros sao considerados seguindo a ordem de definicao da funcao
```

```
## [1] 5
```

```
hipotenusa(y=24, x=7)
```

```
## [1] 25
```

Pode-se definir valores padrão para argumentos de uma função. Na função abaixo, que calcula a potenciação, caso `exp` não seja definido, a função calculará o quadrado do número dado.

```
potencia <- function(base, exp = 2) {
  base ^ exp
}

potencia(10)
```

```
## [1] 100
```

```
potencia(10, 3)
```

```
## [1] 1000
```

```
potencia(c(1:10, NA))
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100  NA
```

Lista de exercicios

1. O comando abaixo gera amostras aleatórias seguindo a distribuição binomial para simular o lançamento de 100 moeda.

```
set.seed(1)
lancamentos <- rbinom(100, 1, 0.5)
```

Considere que 0 represente "cara" e 1 represente "coroa." Crie uma variavel para armazenar os lançamentos como um fator contendo os níveis "cara" e "coroa".

2. Para fins de exemplo, neste exercício considere apenas uma amostra de 1000 linhas dos dados armazenados no dataframe `flights` do pacote `nycflights`.

```
library(nycflights13)

set.seed(1)

flights <- flights[sample(1:nrow(flights), 1000),]
```

a) De acordo com a descrição do mesmo (veja `?flights`), a coluna `distance` está registrada em milhas. Crie uma função que receba como argumento um número em milhas e converta-o para kilometros (para um resultado aproximado, multiplique o valor de comprimento por 1,609). Em seguida, crie um novo dataframe, `flights2` em que sua coluna distance esteja representada em km.

b) Crie uma sequencia de comandos utilizando a estrutura `for` para classificar cada uma das distâncias obtidas no item anterior em "curta distância" (até 500km), "média distância" (entre 500km e 2000km) e "longa distância" (mais que 2000km). Armazene o resultado obtido em uma nova coluna no datafrarme `flights`. Transforme essa coluna em fator.