# ME115 - Linguagem R

Atividade Prática 02 - Gabarito

1º semestre de 2023

## Introdução

Nessa atividade, exploraremos as seguintes operações:

1. indexação de matrizes
2. operações lógicas do tipo `>`, `<`, `&` e `||`, `all`, `any`
3. indexação de `data.frame`
4. Operadores lógicos: no contexto de indexação, comparação entre objetos como mecanismo de parada em blocos de repetição
5. Controle de fluxo: `if/else`; `ifelse`
6. Blocos de repetição: `for`; `while`

## Atividade

1. Considere o código a seguir para criar a matrix

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

```
X <- matrix(data=seq(1, 9), nrow=3, ncol=3)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Essa forma de escrever a matriz não coloca os elementos na posição certa. Podemos usar o argumento `byrow` da função para colocar os elementos na posição certa.

```
X <- matrix(data=seq(1, 9), nrow=3, ncol=3, byrow=T)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

Para acessar os elementos de uma matriz usamos o indexador "`[r, l]`", onde o elemento `r` (à esquerda da vírgula) representa a linha e o `l` (à direita) a coluna. A seguir experimente os códigos que ilustram como indexar matrizes:

```
X[1, 2]      # retorna o elemento da linha 1 e coluna 2
X[2, 1]      # retorno o elemento da linha 2 e coluna 1
X[1,]        # retorna os elementos da linha toda
X[,1]        # retorna os elementos da coluna toda
```

```
X[, 1:2]      # retorna colunas 1 e 2
X[c(1,3), ]   # retorna linhas 1 e 3
diag(X)       # retorna os elementos da diagonal principal da matrix
```

Considere a matrix $X$. Indexando elemento e elemento, calcule:

(a) a soma dos elementos da linha 2.

**Solução:**

```
sum(X[2,])
```

```
## [1] 15
```

(b) a soma dos elementos da coluna 3.

**Solução:**

```
sum(X[, 3])
```

```
## [1] 18
```

(c) a soma dos elementos da diagonal de $X$.

**Solução:**

```
sum(diag(X))
```

```
## [1] 15
```

2. Crie a matrix $Y$ de tamanho $10 \times 10$ que contenha os valores de 1 a 100, dispostos por linha. Imprima na tela os seguintes elementos:

**Solução:**

```
Y <- matrix(1:100, nrow = 10, byrow = TRUE)
even <- seq(2, 10, by=2)
```

(a) Elementos das colunas pares de $Y$.

**Solução:**

```
Y[, even]
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    2    4    6    8   10
##  [2,]   12   14   16   18   20
##  [3,]   22   24   26   28   30
##  [4,]   32   34   36   38   40
##  [5,]   42   44   46   48   50
##  [6,]   52   54   56   58   60
##  [7,]   62   64   66   68   70
##  [8,]   72   74   76   78   80
##  [9,]   82   84   86   88   90
## [10,]   92   94   96   98  100
```

(b) Elementos das linhas pares de $Y$.

**Solução:**

```
Y[even,]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]   11   12   13   14   15   16   17   18   19    20
```

```
## [2,]   31   32   33   34   35   36   37   38   39    40
## [3,]   51   52   53   54   55   56   57   58   59    60
## [4,]   71   72   73   74   75   76   77   78   79    80
## [5,]   91   92   93   94   95   96   97   98   99   100
```

(c) Elementos das linhas e colunas pares de $Y$.

**Solução:**

```
Y[even, even]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   12   14   16   18   20
## [2,]   32   34   36   38   40
## [3,]   52   54   56   58   60
## [4,]   72   74   76   78   80
## [5,]   92   94   96   98  100
```

3. O objeto `data.frame` pode ser pensado como uma matriz. Neste caso, sua indexação pode ser feita através do indexador `[,]`. Execute o código abaixo para carregar o *data frame* `murders` do pacote `dslabs`.

```
library(dslabs)
data(murders)
```

A seguir, responda:

(a) O que representa o comando `murders[1, 1]`?

**Solução:** Representa a primeira linha e primeira coluna do data frame, que é o estado Alabama.

(b) O que representa o comando `murders[1, ]`?

**Solução:** Seleciona a primeira linha do data frame, que representa os dados do Alabama.

(c) O que representa o comando `murders[, 1]`?

**Solução:** Seleciona a primeira coluna do data frame, que representa os estados.

(d) O que representa o comando `murders[1:2, ]`?

**Solução:** Representa as duas primeiras linhas do data frame e todas as colunas.

(e) O que representa o comando `murders[, 3:4]`?

**Solução:** Representa as colunas 3 e 4 do data frame e todas as linhas.

4. O código abaixo retorna "Nem todos são positivos". Por que?

```
x <- c (1, 2, -3, 4)

if (all (x > 0)) {
  print("Todos são positivos")
  } else {
    print("Nem todos são positivos")
    }
```

**Solução:** Porque ao testar a condição `all(x > 0)` isso nos retorna `FALSE`, ja que na posição 3 do vetor temos um número negativo. Então, no código é executado o que está dentro do `else`, ou seja, "Nem todos são positivos".

(a) Reescreva o código tal que ele retorne "Todos são positivos". **Dica**: utilize o operador de negação `!`.

**Solução:**

```
x <- c (1, 2, -3, 4)

if (!all (x > 0)) {
  print("Todos são positivos")
  } else {
    print("Nem todos são positivos")
    }
```

```
## [1] "Todos são positivos"
```

(b) Reescreva o código tal que ele retorne uma frase "Nem todos são números pares" para o caso de haver algum número ímpar em `x` e caso contrário "Todos são números pares". **Dica**: use `x %% 2`.

**Solução:**

```
x <- c (1, 2, -3, 4)

if (all (x %% 2 == 0)) {
  print("Todos são números pares")
  } else {
    print("Nem todos são números pares")
    }
```

```
## [1] "Nem todos são números pares"
```

(c) Usando a função any (utilize `?any` para saber sobre ela) e o vetor `x`, reescreva o código para que a mensagem "Algum número negativo" seja exibida caso existam números negativos em `x` e "Nenhum número negativo" caso contrário.

**Solução:**

```
x <- c (1, 2, -3, 4)

if (any (x < 0)) {
  print("Algum número negativo")
  } else {
    print("Nem número negativo")
    }
```

```
## [1] "Algum número negativo"
```

(d) Defina um vetor `y` tal que usando o código em (b) ele retorne "Todos são números pares".

**Solução:**

```
y <- seq(2, 12, by = 2)

if (all (y %% 2 == 0)) {
  print("Todos são números pares")
  } else {
    print("Nem todos são números pares")
    }
```

```
## [1] "Todos são números pares"
```

5. Usando o código da aula prática 01 dado a seguir:

```
library(dslabs)
data(murders)
```

```
murder_rate <- murders$total / murders$population*100000
```

Execute o código abaixo, o qual nos diz qual(is) estado(s) tem a taxa de assassinato por 100.000 habitantes menor do que 0.5 pessoas. Veja que o `if` é usado aqui para cobrir também os casos em que nenhum estado apresenta taxa menor que o limite estabelecido.

```
ind <- which(murder_rate < 0.5)

if(length(ind) > 0){
  print(murders$state[ind])
  } else {
    print("No state has murder rate that low")
    }
```

```
## [1] "New Hampshire" "Vermont"
```

A seguir, usando o código fornecido, encontre:

(a) os estados com taxa por 100.000 habitantes maior ou igual a 2 pessoas.

**Solução:**

```
ind <- which(murder_rate >= 2)

if(length(ind) > 0){
  print(murders$state[ind])
  } else {
    print("No state has murder rate that low")
    }
```

```
##  [1] "Alabama"              "Alaska"               "Arizona"
##  [4] "Arkansas"             "California"           "Connecticut"
##  [7] "Delaware"             "District of Columbia" "Florida"
## [10] "Georgia"              "Illinois"             "Indiana"
## [13] "Kansas"               "Kentucky"             "Louisiana"
## [16] "Maryland"             "Michigan"             "Mississippi"
## [19] "Missouri"             "Nevada"               "New Jersey"
## [22] "New Mexico"           "New York"             "North Carolina"
## [25] "Ohio"                 "Oklahoma"             "Pennsylvania"
## [28] "South Carolina"       "Tennessee"            "Texas"
## [31] "Virginia"
```

(b) os estados com taxa por 100.000 habitantes entre 0.5 e 2 pessoas.

**Solução:**

```
ind <- which(murder_rate >= 0.5 & murder_rate <= 2)

if(length(ind) > 0){
  print(murders$state[ind])
  } else {
    print("No state has murder rate that low")
    }
```

```
##  [1] "Colorado"      "Hawaii"        "Idaho"         "Iowa"
##  [5] "Maine"         "Massachusetts" "Minnesota"     "Montana"
##  [9] "Nebraska"      "North Dakota"  "Oregon"        "Rhode Island"
## [13] "South Dakota"  "Utah"          "Washington"    "West Virginia"
## [17] "Wisconsin"     "Wyoming"
```

(c) os estados com taxa por 100.000 habitantes menor do que 0.25 pessoas.

**Solução:**

```
ind <- which(murder_rate < 0.25)

if(length(ind) > 0){
  print(murders$state[ind])
  } else {
    print("No state has murder rate that low")
    }
```

```
## [1] "No state has murder rate that low"
```

6. O controle de fluxo `ifelse` trabalha em vetores, como exemplificado no código a seguir.

```
a <- c(0, 1, 2, -4, 5)
ifelse(a > 0, 1/a, NA)
```

```
## [1]  NA 1.0 0.5  NA 0.2
```

A seguir,

(a) Crie uma sequência de inteiros -10 a 120 e armazene. Utilizando `ifelse`, retorne a raiz quadrada dos elementos que são maiores ou iguais a zero e NA caso contrário.

**Solução:**

```
x <- -10:120
ifelse(x >= 0, sqrt(x), NA)
```

```
## Warning in sqrt(x): NaNs produzidos
```

```
##   [1]       NA       NA       NA       NA       NA       NA       NA
##   [8]       NA       NA       NA 0.000000 1.000000 1.414214 1.732051
##  [15] 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000 3.162278
##  [22] 3.316625 3.464102 3.605551 3.741657 3.872983 4.000000 4.123106
##  [29] 4.242641 4.358899 4.472136 4.582576 4.690416 4.795832 4.898979
##  [36] 5.000000 5.099020 5.196152 5.291503 5.385165 5.477226 5.567764
##  [43] 5.656854 5.744563 5.830952 5.916080 6.000000 6.082763 6.164414
##  [50] 6.244998 6.324555 6.403124 6.480741 6.557439 6.633250 6.708204
##  [57] 6.782330 6.855655 6.928203 7.000000 7.071068 7.141428 7.211103
##  [64] 7.280110 7.348469 7.416198 7.483315 7.549834 7.615773 7.681146
##  [71] 7.745967 7.810250 7.874008 7.937254 8.000000 8.062258 8.124038
##  [78] 8.185353 8.246211 8.306624 8.366600 8.426150 8.485281 8.544004
##  [85] 8.602325 8.660254 8.717798 8.774964 8.831761 8.888194 8.944272
##  [92] 9.000000 9.055385 9.110434 9.165151 9.219544 9.273618 9.327379
##  [99] 9.380832 9.433981 9.486833 9.539392 9.591663 9.643651 9.695360
## [106] 9.746794 9.797959 9.848858 9.899495 9.949874 10.000000 10.049876
## [113] 10.099505 10.148892 10.198039 10.246951 10.295630 10.344080 10.392305
## [120] 10.440307 10.488088 10.535654 10.583005 10.630146 10.677078 10.723805
## [127] 10.770330 10.816654 10.862780 10.908712 10.954451
```

(b) Utilizando a sequência criada em (a) e o comando `ifelse`, retorne 1 se o número for par e 0 caso contrário.

**Solução:**

```
ifelse(x %% 2 == 0, 1, 0)
```

```
##   [1] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

```
##  [38] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
##  [75] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
## [112] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

(c) Utilizando a sequencia criada em (a) e o comando `ifelse`, retorne o próprio número se este for divisível por 3 e NA caso contrário.

**Solução:**

```
ifelse(x %% 3 == 0, x, NA)
```

```
##   [1]  NA  -9  NA  NA  -6  NA  NA  -3  NA  NA   0  NA  NA   3  NA  NA   6  NA
##  [19]  NA   9  NA  NA  12  NA  NA  15  NA  NA  18  NA  NA  21  NA  NA  24  NA
##  [37]  NA  27  NA  NA  30  NA  NA  33  NA  NA  36  NA  NA  39  NA  NA  42  NA
##  [55]  NA  45  NA  NA  48  NA  NA  51  NA  NA  54  NA  NA  57  NA  NA  60  NA
##  [73]  NA  63  NA  NA  66  NA  NA  69  NA  NA  72  NA  NA  75  NA  NA  78  NA
##  [91]  NA  81  NA  NA  84  NA  NA  87  NA  NA  90  NA  NA  93  NA  NA  96  NA
## [109]  NA  99  NA  NA 102  NA  NA 105  NA  NA 108  NA  NA 111  NA  NA 114  NA
## [127]  NA 117  NA  NA 120
```

7. O comando `for` é um comando de repetição. Considere os códigos a seguir como exemplos.

```
for(i in 1:5){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
for(i in seq(1, 10, by=2)){
     print(i)
  }
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
```

```
for(i in c("a", "b", "c")){
     print(i)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
```

```
## bloco de repetição aninhado

for(i in c("a", "b", "c")){
  for(j in seq(1, 10, by=2)){
    print(i)
    print(j)
    }
  }
## o resultado é extenso e por isso não foi apresentado
```

```r
soma <- 0
for (i in 1:10){
  soma <- soma + i
}

print(soma)
```

```
## [1] 55
```

A seguir,

(a) Utilizando o *data frame* `murders`, considere a taxa de mortes por 100000 habitantes. Calcule a média da taxa de mortes e armazene na variável `media`.

**Solução:**

```r
media <- mean(murder_rate)
```

(b) Calcule a média da taxa de mortes para a região Sul (*South*) utilizando um bloco de `for`, como nos exemplos, e guarde na variável `media.south`.

**Solução:**

```r
soma.south <- 0
n.south <- 0

for (i in 1:nrow(murders)){
  if(murders$region[i] == "South"){
    soma.south <- soma.south + murder_rate[i]
    n.south <- n.south + 1
  }
}
media.south <- soma.south/n.south
```

**Nota:** o código acima pode ser substituído por uma linha:

```r
mean(murder_rate[murders$region == "South"])
```

(c) Calcule a média da taxa de mortes para regiões que não sejam a região "South" ou "North Central" utilizando um único bloco de repetição `for` e guarde na variável `media.outras`. **Dica**: use o código do item (b) com operador de negação `!`.

**Solução:**

```r
soma.outras <- 0
n.outras <- 0

for (i in 1:nrow(murders)){
if(!(murders$region[i] == "South" | murders$region[i] == "North Central")){
            soma.outras <- soma.outras + murder_rate[i]
            n.outras <- n.outras + 1
  }
}
media.outras <- soma.outras/n.outras
```

(d) Imprima na tela o valor das variáveis em (a), (b) e (c).

**Solução:**

```r
print(c(Media = media, "Media South" = media.south,
"Media Outras" = media.outras))
```

```
##        Media  Media South Media Outras
##     2.779125     4.417012     1.839169
```

8. Considere o exemplo de bloco de repetição aninhado do exercício anterior. A seguir, calcule a seguinte expressão

$$m = \frac{1}{100} \sum_{i=1}^{10} \sum_{j=1}^{10} i \times j$$

e imprima na tela o valor final de $m$.

**Solução:**

```r
m <- 0
for (i in 1:10){
  for (j in 1:10){
    m <- m + i * j
  }
}
m <- m/100
m
```

```
## [1] 30.25
```

**Nota:** O código acima pode ser resumido em uma linha:

```r
mean(outer(1:10, 1:10))
```

```
## [1] 30.25
```

9. Considere bloco de repetição usando o comando `while`.

```r
## note que no bloco de repetição while temos que incrementar manualmente
ind <- 1
while (ind < 100) {
  print(ind);
  ind <- ind + 1
}
```

(a) Considere a sequência `-108:88` e armazene-a em `s`. Utilizando o bloco de repetição `while`, faça a soma dos valores de `s` até que um valor positivo em `s` seja encontrado.

**Solução:**

```r
s <- -108:88

i <- 1
soma <- 0
while (s[i] <= 0) {
  soma <- soma + s[i]
  i <- i + 1
  }
soma
```

```
## [1] -5886
```

**Nota:** O código acima pode ser resumido em uma linha:

```
sum(s[s <= 0])
```

```
## [1] -5886
```

(b) Usando `s` e o bloco de repetição `while`, faça a soma dos valores de `s` a partir do valor 88 até que um valor negativo de `s` seja encontrado. **Dica**: você pode usar decremento de um índice.

**Solução:**

```
i <- length(s)
soma <- 0

while (s[i] >= 0) {
  soma <- soma + s[i]
  i <- i - 1
  }
soma
```

```
## [1] 3916
```

**Nota:** O código acima pode ser resumido em uma linha:

```
sum(s[s >= 0])
```

```
## [1] 3916
```

## Agradecimentos

O material foi produzido pela Profa. Tatiana Benaglia para o curso de ME115.