



ME115 - Linguagem R

Parte 02

1º semestre de 2023

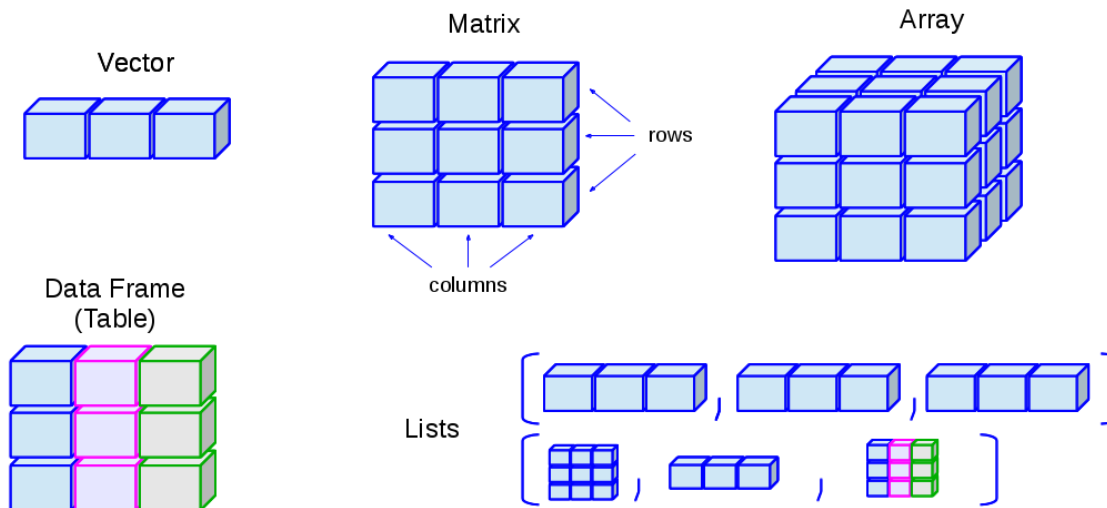
Conteúdo

Nessa aula, trabalharemos os seguintes tópicos:

- Indexação de vetores, matrizes, listas e data frames;
- Operadores relacionais e lógicos;
- Controle de fluxo: `if/else` e `ifelse()`;
- Blocos de Repetição: `for`; `while`.

Estruturas de Dados no R

Na aula passada estudamos as estruturas de dados no R e como criá-las.



Hoje iremos explorar como obter elementos dessas estruturas (indexação) e também funções úteis para trabalhar com cada uma delas.

Fonte: <http://venus.ifca.unican.es>

Vetores

Vetores: uma sequência de elementos do mesmo tipo de dados.

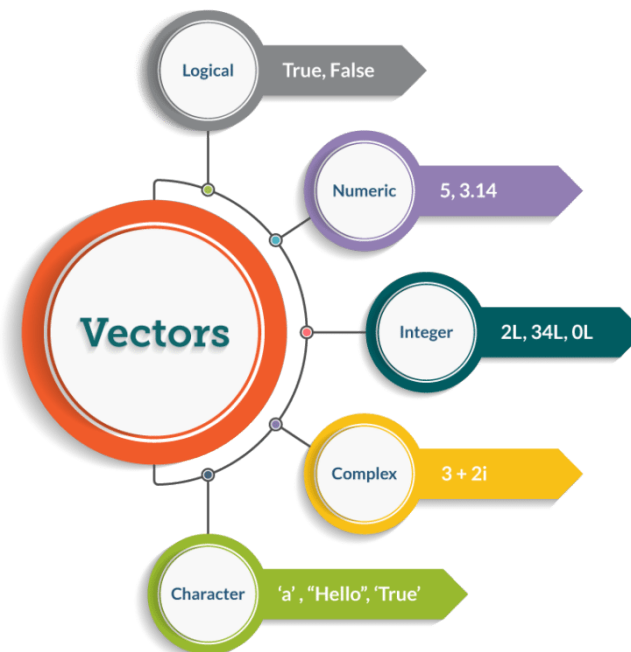
Já vimos que vetores podem ser criados usando:

- `c()`: para concatenar elementos ou subvetores
- `rep()`: para repetir elementos ou padrões
- `seq()` ou `n:m`: para gerar sequências

Exemplo:

```
vec.num <- c(2, 0, 1, 0, 4, 2, 5, 0)
vec.num
```

```
## [1] 2 0 1 0 4 2 5 0
```



Vetores

Mais exemplos:

```
vec.char <- c("Itália", "Japão", "EUA", "Brasil")  
vec.char
```

```
## [1] "Itália" "Japão"  "EUA"    "Brasil"
```

```
vec.seq <- seq(2, 20, 2) # números pares de 2 a 20  
vec.seq
```

```
## [1] 2 4 6 8 10 12 14 16 18 20
```

```
vec.logic <- rep(c(TRUE, FALSE), times = c(3, 2))  
vec.logic
```

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

Nomeando vetores

Pode ser útil nomear os elementos de um vetor. Já fizemos isso na aula prática:

```
codes <- c(italy = 380, canada = 124, egypt = 818)
codes
```

```
##   italy canada  egypt
##   380     124    818
```

O objeto `codes` continua sendo um vetor numérico (verifique com `class()`), mas com nomes:

```
names(codes)
```

```
## [1] "italy"  "canada" "egypt"
```

Nomeando vetores

Também podemos atribuir nomes usando a função `names()`:

```
codes <- c(380, 124, 818)
country <- c("italy", "canada", "egypt")
names(codes) <- country
codes
```

```
##  italy canada  egypt
##   380    124    818
```

Coerção ou conversão forçada

Se tentarmos misturar tipos diferentes de dados em um vetor, o R irá forçar todos os elementos a serem do mesmo tipo, tentando *adivinhar* qual é o tipo mais apropriado.

```
x <- c(1, "canada", 3)
x; class(x)
```

```
## [1] "1"      "canada" "3"
```

```
## [1] "character"
```

Outro exemplo:

```
v <- c(TRUE, 5, FALSE)
v; class(v)
```

```
## [1] 1 5 0
```

```
## [1] "numeric"
```

CUIDADO: Veja que isso não retornou nenhum alerta ou mensagem de erro!!!

Coerção ou conversão forçada

O R também oferece funções para mudar de um tipo para outro. Veja a função `as.character()`:

```
x <- 1:5  
y <- as.character(x)  
y
```

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

Pode-se reverter isso com a função `as.numeric()`:

```
as.numeric(y)
```

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

Veja também: `as.integer()`, `as.logical()`, `as.complex()`.

Selecionar elementos de um vetor

Considere o vetor **x** do exemplo anterior:

```
x <- c(2, 0, 1, 0, 4, 2, 5, 0)
```

O colchete ([]) é usado para indexar e acessar elementos de um vetor.

Podemos selecionar os elementos por **posição**. Alguns exemplos:

- Selecionar o elemento da 4a posição: `x[4]`
- Selecionar todos os elementos, exceto o da 4a posição: `x[-4]`
- Selecionar os elementos das posições 2 ao 4: `x[2:4]`
- Selecionar os elementos das posições 1 e 5: `x[c(1, 5)]`
- Selecionar todos os elementos, exceto os das posições 1 e 5: `x[-c(1, 5)]`

Selecionar elementos de um vetor

Se os elementos tiverem nomes, podemos acessar os elementos pelos nomes:

```
codes <- c(italy = 380, canada = 124, egypt = 818)
codes["canada"]
```

```
## canada
##      124
```

```
codes[c("egypt", "italy")]
```

```
## egypt italy
##    818    380
```

Operadores Relacionais

Operadores relacionais testam uma condição e sempre retornam **TRUE** ou **FALSE**. **Exemplo:** considere o vetor $x = c(2, 0, 1)$.

Operador	Descrição	Exemplo	Comparação	Resposta
<code>==</code>	igual	<code>x == 2</code>	x igual a 2?	TRUE FALSE FALSE
<code>!=</code>	diferente	<code>x != 2</code>	x diferente de 2?	FALSE TRUE TRUE
<code><</code>	menor	<code>x < 2</code>	x menor que 2?	FALSE TRUE TRUE
<code><=</code>	menor ou igual	<code>x <= 2</code>	x menor ou igual a 2?	TRUE TRUE TRUE
<code>></code>	maior	<code>x > 2</code>	x maior que 2?	FALSE FALSE FALSE
<code>>=</code>	maior ou igual	<code>x >= 2</code>	x maior ou igual a 2?	TRUE FALSE FALSE

Selecionar elementos de um vetor

Além de selecionar pela posição, podemos selecionar os elementos testando certas condições através dos operadores relacionais.

```
x <- c(2, 0, 1, 0, 4, 2, 5, 0)
```

Veja alguns exemplos:

- Selecionar todos os elementos iguais a 2: `x[x == 2]`
- Selecionar elementos positivos, ou seja, maiores que 0: `x[x > 0]`
- Selecionar elementos menores ou iguais a 2: `x[x <= 2]`
- Selecionar todos os elementos no conjunto 1, 2, 5: `x[x %in% c(1, 2, 5)]`

Note que as condições dentro de `[]` retornam valores `TRUE` ou `FALSE`. O valor selecionado é aquele onde `TRUE` ocorre.

Operadores Relacionais

Os operadores relacionais também podem ser usados para comparar elemento a elemento de dois vetores de mesmo comprimento.

Teste você mesmo o exemplo abaixo:

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

```
y <- c(1, 4, 6)
```

```
x == y
```

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

```
x < y
```

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

```
x >= y
```

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

Operadores Lógicos

Operador	Descrição
!	negativa de uma condição
& ou &&	operador lógico E
ou	operador lógico OU
xor	mutuamente exclusivo (ou um ou outro)

Exemplo: Selecionar os elementos de x tal que $0 \leq x < 2$.

```
x <- c(2, 0, 1, 0, 4, 2, 5, 0)
x[x >= 0 & x < 2]
```

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

Vetores - Funções úteis

Considere o seguinte vetor **x** como exemplo:

```
x <- c(2, 0, 1, 0, 4, 2, 5, 0)
```

- Número de elementos: `length(x)`
- Vetor com os elementos ordenados: `sort(x)`
- Vetor em ordem reversa: `rev(x)`
- Contagem da frequência de cada valor: `table(x)`
- Extrair os valores únicos: `unique(x)`

Teste essas funções você mesmo para ver os resultados.

Vetores - Mais Funções

Vamos simular 10 lançamentos de uma moeda honesta e armazenar isso em um vetor:

```
x <- sample(0:1, size = 10, replace = TRUE)
```

Algumas funções úteis são:

- Soma de todos os elementos: `sum(x)`
- Média dos elementos: `mean(x)`
- O maior e menor elemento: `max(x)` e `min(x)`
- O índice do maior e menor valor: `which.max(x)` e `which.min(x)`
- Amplitude dos elementos: `range(x)`



Operações em Vetores

No R, operações aritméticas em vetores ocorrem elemento a elemento.

Exemplo: suponha que tenhamos as seguintes alturas em polegadas:

```
height.in <- c(69, 62, 66, 70, 70, 73, 67, 73, 67, 70)
```

e queremos convertê-las em centímetros.

```
height.cm <- height.in * 2.54  
round(height.cm)
```

```
## [1] 175 157 168 178 178 185 170 185 170 178
```

Veja que cada elemento de `height.in` foi multiplicado por 2.54.

O mesmo acontece se aplicarmos outras operações matemáticas.

Operações em Vetores

Se tivermos dois vetores de mesmo comprimento e queremos somá-los, eles serão somados elemento por elemento:

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} + \begin{pmatrix} d \\ e \\ f \end{pmatrix} = \begin{pmatrix} a + d \\ b + e \\ c + f \end{pmatrix}.$$

Exemplo:

```
x <- c(5, 3, 2)
y <- c(2, 3, 4)
x + y
```

```
## [1] 7 6 6
```

O mesmo se aplica a outras operações matemáticas, como $-$, $*$ e $/$.

Operações em Vetores

Exemplo: Suponha que tenhamos os pesos (kg) e alturas (m) de algumas pessoas e queremos calcular os respectivos IMC:

Índice de Masa Corporal

$$\text{IMC} = \frac{\text{Peso (Kg)}}{\text{Altura (m)}^2}$$

```
peso <- c(85, 75, 63, 68, 54)
altura <- c(1.65, 1.59, 1.68, 1.74, 1.54)
imc <- peso / altura^2
round(imc, 2)
```

```
## [1] 31.22 29.67 22.32 22.46 22.77
```

Quais são os pesos das pessoas com IMC > 30?

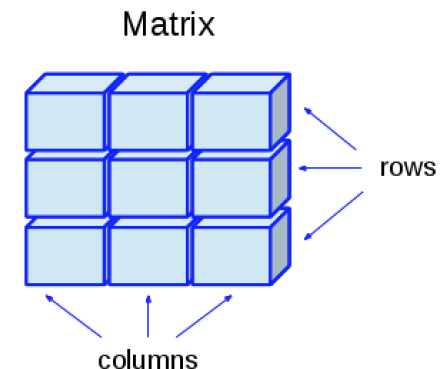
```
peso[imc >= 30]
```

```
## [1] 85
```

Matrizes

Matriz: objeto retangular bidimensional (linhas e colunas) onde as colunas são todas do mesmo tipo.

Podemos criar uma matriz usando a função `matrix()`:



```
m <- matrix(c(2, 4, 3, 1, 5, 7), nrow=2, ncol=3, byrow = TRUE)
```

Para assinalar nomes às colunas e linhas, respectivamente:

```
colnames(m) <- c("C1", "C2", "C3")  
rownames(m) <- c("R1", "R2")
```

```
##      C1 C2 C3  
## R1   2  4  3  
## R2   1  5  7
```

Como criar uma matriz

Também pode-se criar uma matriz à partir de vetores individuais do mesmo tipo e comprimento, usando as funções `rbind()` ou `cbind()`:

```
vetor1 <- c(3, 4, 5)
vetor2 <- c(9, 10, 11)
m2 <- cbind(vetor1, vetor2)
m2
```

```
##      vetor1 vetor2
## [1,]      3      9
## [2,]      4     10
## [3,]      5     11
```

Os nomes das colunas são os nomes dos vetores, mas você pode definí-los de outras formas:

```
colnames(m2) <- c("Coluna1", "Coluna2");m2
```

```
cbind(Coluna1 = vetor1, Coluna2 = vetor2)
```

```
##      Coluna1 Coluna2
## [1,]      3      9
## [2,]      4     10
## [3,]      5     11
```

```
##      Coluna1 Coluna2
## [1,]      3      9
## [2,]      4     10
## [3,]      5     11
```

Selecionar elementos de uma matriz

Assim como em vetores, o operador `[]` é usado para selecionar elementos em matrizes.

Alguns exemplos:

- Selecionar o elemento da 2a linha e 3a coluna:
`m[2, 3]`
- Selecionar o elemento usando o nome da linha e da coluna: `m["row2", "col3"]`
- Selecionar uma linha completa: `m[2,]`
- Selecionar uma coluna completa: `m[, 1]`
- Selecionar todas as colunas exceto a 2 e 3:
`m[, -c(2, 3)]`

	col1	col2	col3
row1	2	4	3
row2	1	5	7

	col1	col2	col3
row1	2	4	3
row2	1	5	7

	col1	col2	col3
row1	2	4	3
row2	1	5	7

	col1	col2	col3
row1	2	4	3
row2	1	5	7

Matrizes - Funções Úteis

Considere a seguinte matrix `m` definida anteriormente.

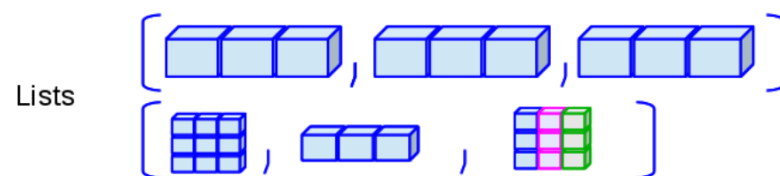
Algumas funções úteis:

- Dimensão da matriz: `dim(m)`
- Número de linhas e colunas: `nrow(x)` e `ncol(x)`
- Transposto da matriz: `t(m)`
- Soma dos elementos das linhas ou colunas: `rowSums(m)` ou `colSums(m)`
- Média dos elementos das linhas ou colunas: `rowMeans(m)` ou `colMeans(m)`
- Adicionar uma linha ou coluna a `m`: `rbind()` ou `cbind()`
- Multiplicação de matrizes: `m %*% n`

	col1	col2	col3
row1	2	4	3
row2	1	5	7

Listas

Listas: coleção de elementos que podem ser de tipos e dimensões diferentes.



Para criar uma lista usamos a função `list()`:

```
registro <- list(nome = "Júlia Alves", ra = 210012, notas = c(95, 82, 91, 97, 93))
registro
```

```
## $nome
## [1] "Júlia Alves"
##
## $ra
## [1] 210012
##
## $notas
## [1] 95 82 91 97 93
```

Indexando Listas

Para extrair os componentes de uma lista, você pode utilizar o operador \$ ou [].

Veja no exemplo abaixo que os três comandos são equivalentes.

```
registro$ra; registro[["ra"]]; registro[[2]]
```

```
## [1] 210012
```

```
## [1] 210012
```

```
## [1] 210012
```

Agora experimente usar colchete simples ([]) e observe a diferença:

```
registro[2]
```

```
## $ra
```

```
## [1] 210012
```

Indexando Listas

O operador `[]` pode ser usado para extrair um elemento de dentro de um outro elemento de uma lista:

```
registro[c(3, 2)] # 2o elemento do 3o elemento da lista
```

```
## [1] 82
```

```
registro[[3]][[2]] # mesmo que o comando acima
```

```
## [1] 82
```

O operador `[]` pode ser usado para extrair vários elementos de uma lista:

```
registro[c(2, 1)]
```

```
## $ra  
## [1] 210012  
##  
## $nome  
## [1] "Júlia Alves"
```

Data Frame

Data Frame: objeto retangular onde as colunas podem ser de tipos diferentes. Nesse sentido, é um caso especial de uma lista, porém os elementos devem ter o mesmo comprimento.

Vamos utilizar aqui o data frame que criamos na aula anterior:

```
nome <- c("João", "Maria", "Tiago", "Ana", "Priscila")
idade <- c(21, 20, 18, 20, 19)
cidade <- c("Campinas", "São Paulo", "Campinas", "Ibaté", "Araraquara")
fumante <- c(FALSE, FALSE, TRUE, FALSE, FALSE)
df <- data.frame(nome, idade, cidade, fumante)
df
```

```
##      nome idade  cidade fumante
## 1  João    21  Campinas  FALSE
## 2  Maria    20  São Paulo  FALSE
## 3  Tiago    18  Campinas   TRUE
## 4   Ana    20    Ibaté    FALSE
## 5 Priscila   19 Araraquara  FALSE
```

nome	idade	cidade	fumante
João	21	Campinas	FALSE
Maria	20	São Paulo	FALSE
Tiago	18	Campinas	TRUE
Ana	20	Ibaté	FALSE
Priscila	19	Araraquara	FALSE

Data Frame - Funções Úteis

Já vimos que podemos verificar a estrutura de um data frame usando `str(df)`.

nome	idade	cidade	fumante
João	21	Campinas	FALSE
Maria	20	São Paulo	FALSE
Tiago	18	Campinas	TRUE
Ana	20	Ibaté	FALSE
Priscila	19	Araraquara	FALSE

As funções para verificar a dimensão de matrizes também funcionam para data frames: `nrow(df)`, `ncol(df)` e `dim(df)`.

Quando o data frame é muito longo, podemos querer apenas dar uma olhada nas primeiras ou últimas linhas usando `head()` e `tail()`, respectivamente.

```
head(df, 2)
```

```
##      nome idade  cidade fumante
## 1  João    21  Campinas  FALSE
## 2  Maria   20  São Paulo  FALSE
```

Se quisermos olhar os nomes das colunas/variáveis: `names(df)`

Indexando Data Frames

Para acessar as colunas ou elementos de um data frame, podemos usar os mesmos comando que funcionam tanto com matrizes quanto listas.

Por exemplo, para selecionar a primeira coluna, os seguintes comandos são equivalentes:

nome	idade	cidade	fumante
João	21	Campinas	FALSE
Maria	20	São Paulo	FALSE
Tiago	18	Campinas	TRUE
Ana	20	Ibaté	FALSE
Priscila	19	Araraquara	FALSE

```
df$nome  
df[[1]]  
df[, 1]  
df[, "nome"]
```

```
## [1] "João"      "Maria"     "Tiago"     "Ana"       "Priscila"
```

Indexando Data Frames como Listas

```
df[1] # Obter a primeira coluna
```

```
##      nome
## 1  João
## 2  Maria
## 3  Tiago
## 4   Ana
## 5 Priscila
```

nome	idade	cidade	fumante
João	21	Campinas	FALSE
Maria	20	São Paulo	FALSE
Tiago	18	Campinas	TRUE
Ana	20	Ibaté	FALSE
Priscila	19	Araraquara	FALSE

```
df[c(1, 3)] # Obter a primeira e terceira colunas
```

```
##      nome      cidade
## 1  João  Campinas
## 2  Maria São Paulo
## 3  Tiago  Campinas
## 4   Ana    Ibaté
## 5 Priscila Araraquara
```

Indexando Data Frames como Listas

```
df[-3] # omitir a terceira coluna
```

```
##      nome idade fumante
## 1   João    21   FALSE
## 2   Maria    20   FALSE
## 3   Tiago    18    TRUE
## 4    Ana    20   FALSE
## 5 Priscila   19   FALSE
```

nome	idade	cidade	fumante
João	21	Campinas	FALSE
Maria	20	São Paulo	FALSE
Tiago	18	Campinas	TRUE
Ana	20	Ibaté	FALSE
Priscila	19	Araraquara	FALSE

Indexando Data Frames como Matrizes

```
df[2, ] # obter a 2a linha, mantendo todas as colunas
```

```
##      nome idade   cidade fumante  
## 2 Maria    20 São Paulo  FALSE
```

```
df[, 3] # obter a 3a coluna  
df[, "cidade"]
```

```
## [1] "Campinas" "São Paulo" "Campinas" "Ibaté" "Araraquara"
```

```
df[1:2, ] # obter as linhas 1 e 2, mantendo todas as colunas
```

```
##      nome idade   cidade fumante  
## 1 João    21 Campinas  FALSE  
## 2 Maria   20 São Paulo  FALSE
```

Subconjuntos de Data Frames

Existe uma forma mais conveniente para obter um subconjunto dos dados, através da função `subset()` da base do R.

Por exemplo, selecionar apenas as pessoas de Campinas.

nome	idade	cidade	fumante
João	21	Campinas	FALSE
Maria	20	São Paulo	FALSE
Tiago	18	Campinas	TRUE
Ana	20	Ibaté	FALSE
Priscila	19	Araraquara	FALSE

```
subset(df, cidade == "Campinas")
```

```
##      nome idade  cidade fumante
## 1  João    21 Campinas  FALSE
## 3  Tiago    18 Campinas   TRUE
```

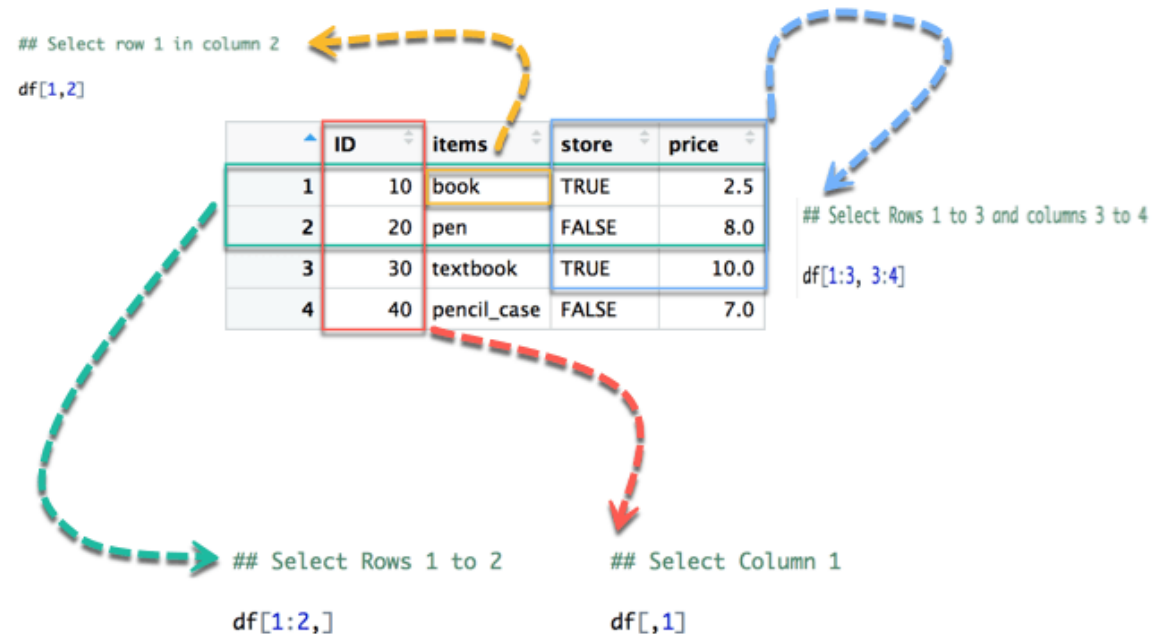
Essa função tem três argumentos: `subset(df, select, subset)`.

select: o nome de uma ou mais colunas;

subset: expressão lógica que seleciona linhas.

Manipulação de Data Frames

Hoje vimos como indexar data frames usando a base do R e também algumas funções que são úteis. O intuito é para vocês se familiarizarem com esse tipo de objeto.

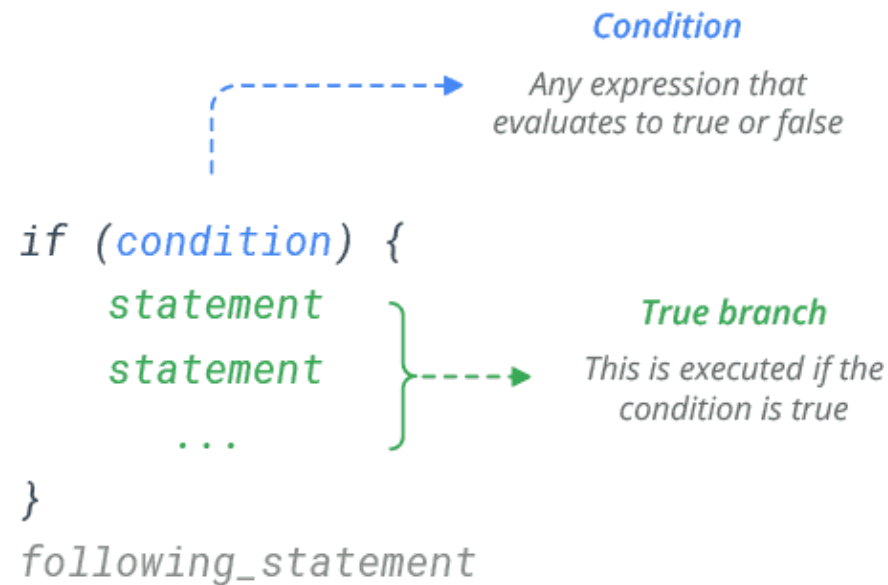


Trabalharemos muito mais com data frames nesse curso e exploremos também o pacote `dplyr`, criado para manipulação de bancos de dados.

Controle de Fluxo

Controle de Fluxo: **if**

if: executa um bloco de comandos se a condição é verdadeira.



Fonte: <https://www.learnbyexample.org/r-if-else-elseif-statement/>

Controle de Fluxo: `if`

Para testar uma condição, utilizaremos os operadores relacionais vistos anteriormente.

Vejamos um exemplo:

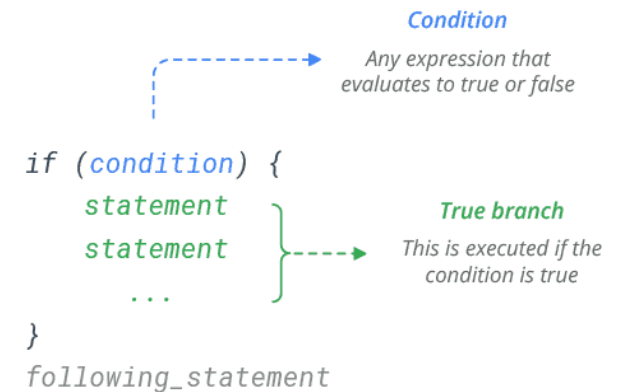
```
x <- 7
y <- 5
if(x > y) {
  print("x é maior que y")
}
```

```
## [1] "x é maior que y"
```

Como só há uma linha de comando dentro do `if`, as chaves não são essenciais.

```
if(x > y) print("x é maior que y")
```

```
## [1] "x é maior que y"
```



if dentro de if

Você pode colocar um `if` dentro de outro `if` para testar mais de uma condição e retornar resultados diferentes.

```
x <- 7
y <- 5
z <- 2
if(x > y) {
  print("x é maior que y")
  if(x > z) print("x é maior que y e z")
}
```

```
## [1] "x é maior que y"
## [1] "x é maior que y e z"
```

Ou você pode utilizar operadores lógicos e juntar todos os `ifs` em um só.

if com mais de uma condição

Para agrupar duas ou mais condições em um único `if`, use os operadores lógicos que vimos anteriormente.

```
x <- 7
y <- 5
z <- 2
if (x > y && x > z) {
  print("x é maior que y e z")
}
```

```
## [1] "x é maior que y e z"
```


Controle de Fluxo: **if/else**

if: executa um bloco de comandos se a condição é **VERDADEIRA**.

else: executa um bloco de comandos se a condição é **FALSA**.

```
if (condition) {  
    statement  
    statement  
    ...  
} else {  
    statement  
    statement  
    ...  
}  
following_statement
```

True branch
This is executed if the condition is true

False branch
This is executed if the condition is false

Fonte: <https://www.learnbyexample.org/r-if-else-elseif-statement/>

Controle de Fluxo: `if/else`

Vejamos um exemplo simples de comparação:

```
x <- 7; y <- 5
if(x > y) {
  print("x é maior que y")
} else {
  print("y é maior que x")
}
```

```
## [1] "x é maior que y"
```

`if/else` em uma única linha:

`variable <- if (condition) Statement else Statement;`

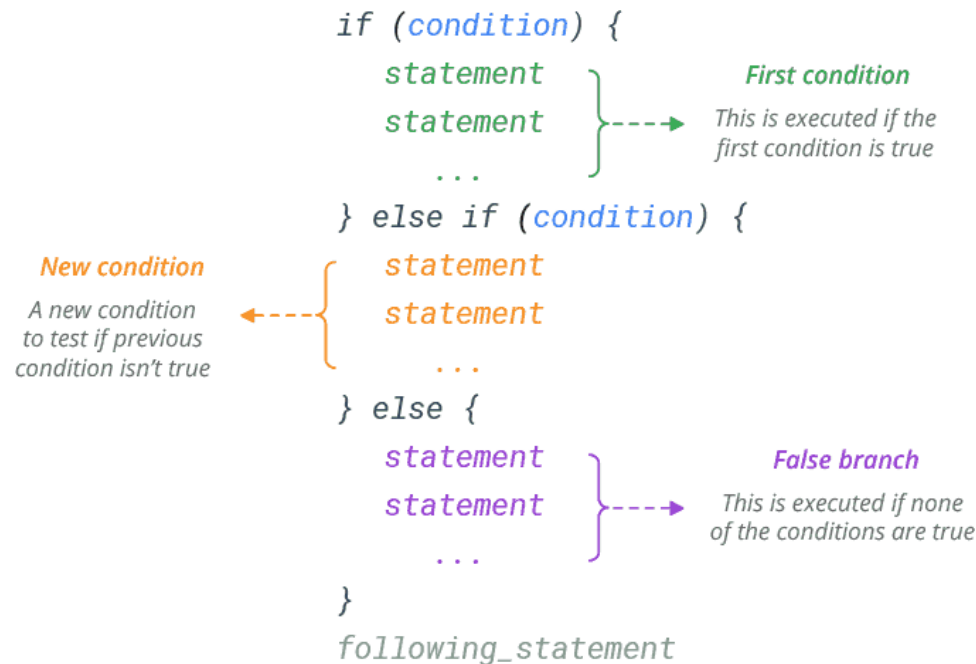
True branch
Execute this if the condition is true

False branch
Execute this if the condition is false

```
if (x > y) print("x é maior que y") else print("y é maior que x")
```

Controle de Fluxo: **else if**

else if: usado para especificar uma nova condição a ser testada, caso a primeira seja **FALSA**.



Fonte: <https://www.learnbyexample.org/r-if-else-elseif-statement/>

Controle de Fluxo: **else if**

Veja um exemplo:

```
x <- 5
y <- 5
if(x > y) {
  print("x é maior que y")
} else if(x < y) {
  print("y é maior que x")
} else {
  print("x e y são iguais")
}
```

```
## [1] "x e y são iguais"
```

No R, você pode usar quantos **else if** você achar necessário. Porém, não é melhor prática quando você quer tomar uma série de decisões. Veja a função **switch()**.

A função `ifelse()`

No R, os controles de fluxo não são operações vetoriais, ou seja, eles conseguem lidar apenas com um valor por vez.

Veja o que acontece quando a condição é checada em um vetor:

```
v <- 1:6
if(v %% 2 == 0) {
  print("ímpar")
} else {
  print("par")
}
```

```
## Error in if (v%%2 == 0) {: the condition has length > 1
```

A função `ifelse()`

`ifelse()`: função que testa a condição para cada elemento de um vetor e retorna um vetor de mesmo comprimento, preenchido com os valores especificados se “VERDADEIRO” ou “FALSO”.

Sintaxe:

```
ifelse (condition, TrueVector, FalseVector)
```

Condition

Condition is checked for every element of a vector

True branch

Select element from this if the condition is true

False branch

Select element from this if the condition is false

```
v <- 1:6  
ifelse(v %% 2 == 0, "par", "ímpar")
```

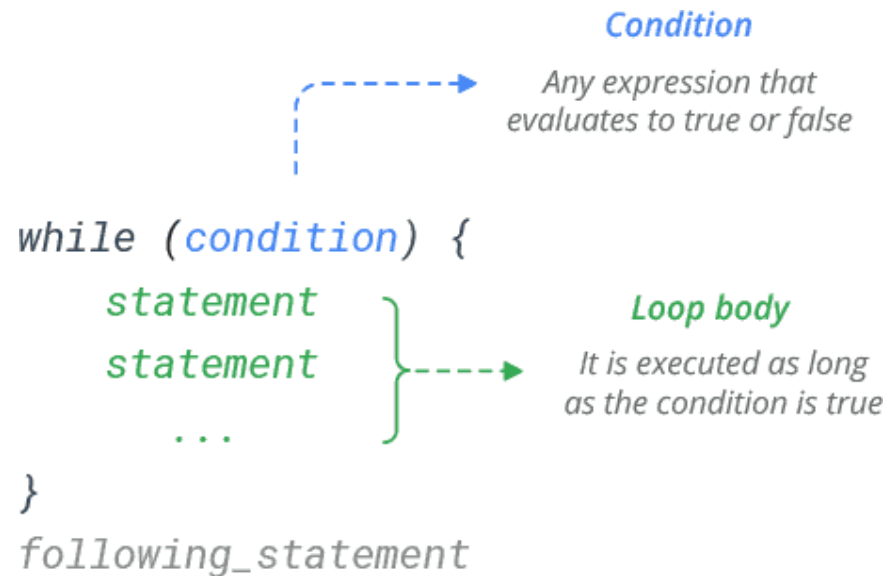
```
## [1] "ímpar" "par" "ímpar" "par" "ímpar" "par"
```

Blocos de Repetição

Blocos de Repetição: **while**

while: usado quando você quer repetir uma tarefa indefinidamente, até que uma condição seja satisfeita.

Sintaxe:



The diagram illustrates the syntax of a **while** loop. It shows the code structure with annotations for the **Condition** and **Loop body**.

```
while (condition) {  
    statement  
    statement  
    ...  
}  
following_statement
```

Condition
Any expression that evaluates to true or false

Loop body
It is executed as long as the condition is true

Blocos de Repetição: **while**

Exemplo: iterar até que x se torne 0.

```
x <- 5
while (x != 0) {
  print(x)
  x <- x - 1
}
```

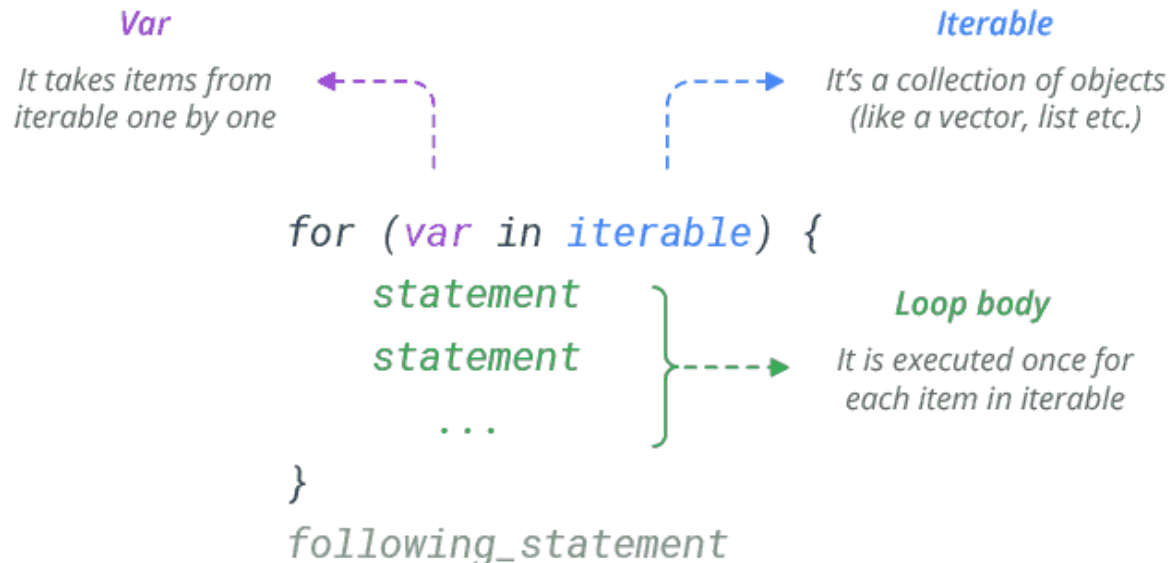
```
## [1] 5
## [1] 4
## [1] 3
## [1] 2
## [1] 1
```

Nota: Pode-se escrever apenas `while(x)` já que o R entende o zero como **FALSO** e qualquer valor diferente de zero como **VERDADEIRO**.

Blocos de Repetição: **for**

O `for` loop do R é um pouco diferente de como normalmente ele é usado em outras linguagens de programação. Você pode iterar não somente sobre uma progressão numérica, mas também sobre os elementos de um vetor ou lista.

Sintaxe:



Blocos de Repetição: **for**

Exemplo: iterar sobre os elementos de um vetor.

```
colors <- c("red", "green", "blue", "yellow")
for (x in colors) {
  print(x)
}
```

```
## [1] "red"
## [1] "green"
## [1] "blue"
## [1] "yellow"
```

Blocos de Repetição: **for**

Exemplo: iterar sobre os elementos de uma lista.

```
mylist <- list(3.14, "Hi", c(1,2,3))  
for (x in mylist) {  
  print(x)  
}
```

```
## [1] 3.14  
## [1] "Hi"  
## [1] 1 2 3
```

Blocos de Repetição: **for** dentro de **for**

Exemplo: Percorrer uma matriz linha por linha e imprimir seus elementos.

```
m <- matrix(1:6, nrow = 2, byrow = FALSE)
```

```
for(i in 1:nrow(m)) {  
  for(j in 1:ncol(m)) {  
    print(m[i, j])  
  }  
}
```

```
## [1] 1  
## [1] 3  
## [1] 5  
## [1] 2  
## [1] 4  
## [1] 6
```

Break e Next

break: usado para quebrar o loop imediatamente quando uma certa condição é detectada. Ele sai do loop naquela iteração e o programa continua depois do loop.

Exemplo: Parar o loop quando chegar na cor `blue`.

```
cores <- c("red", "green", "blue", "yellow")
for (x in cores) {
  if (x == "blue")
    break
  print(x)
}
```

```
## [1] "red"
## [1] "green"
```

Break e Next

`next`: usado para pular aquela iteração quando uma certa condição é detectada e o loop continua com a próxima iteração.

Exemplo: Pular a cor `blue`, mas imprimir as outras cores.

```
cores <- c("red", "green", "blue", "yellow")
for (x in cores) {
  if (x == "blue")
    next
  print(x)
}
```

```
## [1] "red"
## [1] "green"
## [1] "yellow"
```

Referências

Algumas referências utilizadas para a construção desse material:

[Introdução à Ciência de Dados - Rafael A. Irizarry](#)

[R Programming for Data Science - Roger D. Peng](#)

[Curso de R online para iniciantes - Didática Tech](#)

[Learn by example](#)

[Base R - Cheat Sheet](#)

Slides produzidos pela profa. Tatiana Benaglia.

