Capítulo 2 - Essencial

2.1 Instalação e Interfaces

Importante! Sempre instalar o R antes do RStudio, pois o Rstudio é apenas uma interface para o R, portanto, precisa encontrar a instalação do R para poder ser instalado.

Windows

R

1. Entrar no site https://cran.r-project.org/;
2. Selecionar o link `Download R for Windows`;
3. Em `Subdrectories`, selecionar o link `base`;
4. Nesta página estará em destaque o link da última versão disponível do R para windows. Clicar no link e fazer o dwonload;
5. Após o download concluído, abrir o `.exe` e seguir o procedimento de qualquer instalação. Ou entrar neste link e baixar a versão 3.3.2 do R para windows.

RStudio

1. Entrar no site https://www.rstudio.com/;
2. Na aba `Products`, selecionar `RStudio`;
3. Na opção `Desktop`, clicar em `DOWNLOAD RSTUDIO DESKTOP`;
4. Nesta página é mostrada a última versão do integrated development environment(IDE). Na parte `Installers`, selecionar o link do instalador para `windows`;
5. Após o download concluído, abrir o `.exe` e seguir o procedimento de qualquer instalação. Ou entrar neste link e baixar RStudio 1.0.44 - Windows Vista/7/8/10.

Linux

R

1. Entrar no site https://cran.r-project.org/;
2. Selecionar o link `Download R for Linux`;
3. Selecionar conforme sua distribuição linux: `debian`, `redhat`, `suse`, `ubuntu`;

debian

1. Esta página contém as informações do R em relação a esta distribuição;
2. Para a instalação, executar no terminal:

```
apt-get install r-base r-base-dev
```

redhat

1. Selecionar `README`;
2. Esta página contém as informações do R em relação a esta distribuição;
3. Para a instalação, executar no terminal:

```
sudo yum install R
```

suse

1. Esta página contém as informações do R em relação a esta distribuição;
2. Para a instalação, executar no terminal:

```
VERSION=$(grep VERSION /etc/SuSE-release | sed -e 's/VERSION = //')
zypper addrepo -f \
http://download.opensuse.org/repositories/devel\:/languages\:/R\:/patched/openSUSE_$VERSION/ \
R-base
```

3. Nesta mesma página existem outras formas de instalação.

ubuntu

1. Esta página contém as informações do R em relação a esta distribuição;
2. Para a instalação, executar no terminal:

```
sudo apt-get install r-base r-base-core
```

RStudio

1. Entrar no site https://www.rstudio.com/;
2. Na aba `Products`, selecionar `RStudio`;
3. Na opção `Desktop`, clicar em `DOWNLOAD RSTUDIO DESKTOP`;
4. Nesta página é mostrada a última versão do integrated development environment(IDE). Na parte `Installers`, selecionar o link do instalador para sua distribuição Linux.

debian e ubuntu

1. 32 bits - https://download1.rstudio.org/rstudio-1.0.44-i386.deb
2. 64 bits - https://download1.rstudio.org/rstudio-1.0.44-amd64.deb

redhat e susa

1. 32 bits - https://download1.rstudio.org/rstudio-1.0.44-i686.rpm
2. 64 bits - https://download1.rstudio.org/rstudio-1.0.44-x86_64.rpm

---

MAC

R

1. Entrar no site https://cran.r-project.org/;
2. Selecionar o link `Download R for (Mac) OS X`;
3. Nesta página estão os links dos pacotes para Mac OS X 10.6 ou maior.

Versões mais antigas

1. Mac OS 8.6 a 9.2 - https://cran.r-project.org/bin/macos/
2. Mac OS X até 10.5 e Power PC - https://cran.r-project.org/bin/macosx/old

RStudio

1. Entrar no site https://www.rstudio.com/;
2. Na aba `Products`, selecionar `RStudio`;
3. Na opção `Desktop`, clicar em `DOWNLOAD RSTUDIO DESKTOP`;
4. Nesta página é mostrada a última versão do integrated development environment(IDE). Na parte `Installers`, selecionar o link do instalador para `Mac OS X 10.6+`.
5. Ou entrar neste link e baixar RStudio 1.0.136 - Mac OS X 10.6+ (64-bit).

OBS: Caso sua distribuição não esteja nesses links, pode ser necessário compilar o código fonte que pode ser baixado aqui.

2.2 Objetos e suas Classes

Objetos

O R é uma linguagem baseada em objetos, ou seja, tudo que é usado no R está guardado na memória do computador como um objeto. O R não acessa diretamente a memória do computador.

Para armazenar algo em um objeto é utilizado o operador de atribuição, um `<` seguido de um `-`.

Nomes para objetos

- Podem ser formados por letras, números, `"_"` e `"."`;
- Não podem começar com número e/ou ponto;
- Não podem conter espaços;
- Evite usar acentos;
- Procure utilizar nomes curtos e o mais intuitivos possível;
- O R é *case sensitive* (diferencia letras maiúsculas e minúsculas).

  `num`≠`Num`≠`NUM`

Objetos básicos do R

Vetores

O tipo mais básico de objeto do R. Pode ser uma sequencia de itens, mas todos devem ter o mesmo tipo (todos numéricos, todos caracteres):

```
# Exemplo de vetores numéricos
f <- 14
f                     # f é um vetor de um único item
```

```
## [1] 14
```

```
g <- c(1, 2, 3, 4, 5) # função `c()` combina todos seus argumentos
g                     # g é um vetor de 5 posições
```

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

```
#-----------------------------------------------------------------------
# Exemplo de vetor de caracteres
h <- LETTERS[1:10]# LETTERS[1:10] atribui ao vetor as 10 primeiras
                  # letras do alfabeto, maiúscolas.
                  # letters[1:10] atribuiria as mesmas letras, mas
                  # minúscolas
h                 # h é um vetor de caracteres
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

```
#-----------------------------------------------------------------------
# Exemplo de vetor recebendo operação matamática
i <- 25 * 12
i # i receberá o resultado da operação atribuída a ele
```

```
## [1] 300
```

```
#-----------------------------------------------------------------------
# Para ver a classe de um objeto, masta utilizar `class(objeto)`
class(h)
```

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

```
class(i)
```

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

```
#-----------------------------------------------------------------------
# Para ver a estrutura de um objeto, usar `str(objeto)`
str(h)
```

```
##  chr [1:10] "A" "B" "C" "D" "E" "F" "G" "H" "I" ...
```

```
str(i)
```

```
##  num 300
```

Operações com vetores

Como o R tem a característica de poder vetorizar a maioria das suas funções, permitindo que funções possam ser aplicadas a um vetor, retornando um vetor de resultados.

```r
# Criando vetor v1 numérico
v1 <- c(4, 8, 12)
# aplicando a função `log()`, que neste caso calculará o logarítmo
# na base 10 de cada valor do vetor individualmente e retornará
# um vetor de resultados
log(v1)
```

```
## [1] 1.386294 2.079442 2.484907
```

```r
#-----------------------------------------------------------------
# Também podem ser feitas operações aritméticas em vetores e entre
# vetores.
#-----------------------------------------------------------------
# Multiplicando v1 por 2
v1 * 2
```

```
## [1]  8 16 24
```

```r
#-----------------------------------------------------------------
# Criando vetor v2 numérico
v2 <- c(10, 9, 8)
# Somando os dois vetores
v1 + v2
```

```
## [1] 14 17 20
```

```r
# Note que o vetor de resultados é (10+4, 8+9 e 12+8)
```

Regra da Reciclagem

O R reutiliza valores de vetores com menos elementos até que sejam utilizados todos os valores do vetor com mais elementos quando faz operações entre vetores de tamanhos diferentes.

```r
# Criando vetor v3 numérico
v3 <- c(3, 4, 5, 6, 7, 8)
# Somando os dois vetores
v1 + v3
```

```
## [1]  7 12 17 10 15 20
```

```r
# Note que o vetor de resultados é (4+3, 8+4, 12+5, 4+6, 8+9 e 12+8)
#-------------------------------------------------------------------------
# Criando um vetor v4 numérico
v4 <- c(10, 15, 20, 22, 16)
# Somando os dois vetores
v1 + v4
```

```
## Warning in v1 + v4: comprimento do objeto maior não é múltiplo do
## comprimento do objeto menor
```

```
## [1] 14 23 32 26 24
```

```r
# Note que o vetor de resultados é (4+10, 8+15, 12+20, 4+22 e 8+16)
# O R vai calcular e dar um resultado, mas mostrará uma mensagem de aviso
# no console dizendo que o vetor maior não é múltiplo do menor
```

Matrizes

Matrizes no R não passam de vetores, mas com duas dimensões e, assim como os vetores, devem ter todos os seus elementos do mesmo tipo.

```
# Criando um vetor m1 numérico
m1 <- c(1, 2, 3, 4)
m1
```

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

```
# Transformando em uma matriz, utilizando a função `dim()`, que
# mostra quantas dimensões tem um objeto, caso tenha uma só,
# retornará `NULL`
dim(m1) <- c(2, 2)
# Aqui usa-se a função `dim()` para atribuir uma nova dimensão
# para m1, transformando-o em uma matriz e o c(2, 2) significa
# que a matriz terá 2 linhas e 2 colunas respectivamente
# Dessa maneira a matriz será preenchida por colunas, seguindo
# a ordem da sequência numérica
m1
```

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

```
# Caso o número de linha e colunas não bata com a quantidade de
# elementos, a matriz não será criada e surgirá uma mensagem de
# erro no console
#-----------------------------------------------------------------------
# Existe forma mais simples de criar uma matriz
# usando a função `matrix()`
# criar m1 novamente usando `matrix()`
m1 <- matrix(c(1, 2, 3, 4), 2, 2)
# "matrix(sequencia.de.elementos, nr.de.linhas, nr.de.colunas)
# Dessa maneira a matriz será preenchida por colunas, seguindo
# a ordem da sequência numérica
m1
```

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

```
# Caso o número de linhas e colunas não bata com a quantidade de
# elementos, a matriz não será criada e surgirá uma mensagem de
# erro no console
#-----------------------------------------------------------------------
# Criando a m2 com a mesma sequência numérica, mas preenchendo
# a matriz por linhas
m2 <- matrix(c(1, 2, 3, 4), 2, 2, byrow = TRUE)
# com esse argumento a mais `byrow = TRUE` é garantido que a
# matriz será preenchida por linhas
m2
```

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

```r
#-----------------------------------------------------------------------
# Note a diferença entre m1 e m2 que foram criadas com a mesma
# sequência
m1
```

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

```r
m2
```

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

Também é possível dar nomes as linhas e colunas das matrizes para tornar mais fácil o entendimento das mesmas.

```r
# criando uma matriz m3
m3 <- matrix(c(92, 70, 40, 88, 70, 75), 2, 3)
m3
```

```
##      [,1] [,2] [,3]
## [1,]   92   40   70
## [2,]   70   88   75
```

```r
#-----------------------------------------------------------------------
# Dando nomes as linhas de m3
rownames(m3) <- c("aluno1", "aluno2")
# Dando nomes as colunas de m3
colnames(m3) <- c("P1", "P2", "P3")
m3
```

```
##        P1 P2 P3
## aluno1 92 40 70
## aluno2 70 88 75
```

```r
#-----------------------------------------------------------------------
# Para ver a classe de um objeto, basta utilizar `class(objeto)`
class(m3)
```

```
## [1] "matrix"
```

```r
#-----------------------------------------------------------------------
# Para ver a estrutura de um objeto, usar `str(objeto)`
str(m3)
```

```
##  num [1:2, 1:3] 92 70 40 88 70 75
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:2] "aluno1" "aluno2"
##   ..$ : chr [1:3] "P1" "P2" "P3"
```

Operações com matrizes

No R é possível fazer todas as operações matriciais, desde que as matrizes obedeçam as condições para tal.

Matrizes também se utilizam da regra da reciclagem.

```r
# Operação matriz/número
m1 * 3
```

```
##      [,1] [,2]
## [1,]    3    9
## [2,]    6   12
```

```
#-----------------------------------------------------------------
# Multiplicando m1 por v1
m1 * v1
```

```
## Warning in m1 * v1: comprimento do objeto maior não é múltiplo do
## comprimento do objeto menor
```

```
##      [,1] [,2]
## [1,]    4   36
## [2,]   16   16
```

```
# Note a aplicação da regra da reciclagem, onde v1 tem menos
# elementos que m1, então o R vai multiplicando na odem vetorial
# default, que é por colunas
# Resultado:
# | (1 * 4)  (3 * 12) |
# | (2 * 8)  (4 * 4)  |
#-----------------------------------------------------------------
# Multiplicando m2 por m3
m2 %*% m3
```

```
##       P1  P2  P3
## [1,] 232 216 220
## [2,] 556 472 510
```

```
# `%*%` é um operador especial para multiplicação entre matrizes
#-----------------------------------------------------------------
# Função `t()`, para obter a transposta de uma matriz
t(m3)
```

```
##    aluno1 aluno2
## P1     92     70
## P2     40     88
## P3     70     75
```

```
#-----------------------------------------------------------------
# Função `det()`, para obter o determinante de uma matriz quadrada
det(m2)
```

```
## [1] -2
```

```
#-----------------------------------------------------------------
# Função `solve()`, para obter a inversa de uma matriz
solve(m2)
```

```
##      [,1] [,2]
## [1,] -2.0  1.0
## [2,]  1.5 -0.5
```

```
#-----------------------------------------------------------------
# Também pode ser usada para resolver sistemas lineares
# Usando m2 como matriz dos coeficientes e criando um vetor
# `resp` que será o vetor resultado do sistema
resp <- c(7, 15)
```

O sistema será:

$$\begin{cases} x + 2y = 7 \\ 3x + 4y = 15 \end{cases}$$

```
# Resolvendo o sistema com `solve(matriz.dos.coeficientes, vetor.resposta)`
solve(m2, resp)
```

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

```
# No vetor dos resultados estão os valores de x e y, respectivamente
```

Data frames

São parecidos com matrizes, mas podem armazenar dados de tipos diferentes. Podendo ser vistos também como uma tabela de dados onde as linhas são as observações e as colunas as variáveis. Só podem ser criados se todas as colunas tiverem a mesma quantidade de elementos.

```
# Criando um Data frame
# Função `data.frame(nome.da.coluna1 = elementos, nome.da.coluna2 = elementos)`
da <- data.frame(t1 = c(16, 25, 72, 85), t2 = c(79, 81, 55, 68),
                 t3 = c(55, 69, 100, 25))
da
```

```
##   t1 t2  t3
## 1 16 79  55
## 2 25 81  69
## 3 72 55 100
## 4 85 68  25
```

```
#-------------------------------------------------------------------------------
# Para colocar nomes nas linhas, utilizar a função `row.names()`
row.names(da) <- c("Maria", "José", "Lauro", "Lurdes")
da
```

```
##        t1 t2  t3
## Maria  16 79  55
## José   25 81  69
## Lauro  72 55 100
## Lurdes 85 68  25
```

```
#-------------------------------------------------------------------------------
# Também é possível já criar o data frame com os nomes das linhas
# `row.names = ` é um argumento da função `data.frame()`
# Criando um data frame da2 com os mesmos dados, mas com os nomes
# das linhas
da2 <- data.frame(t1 = c(16, 25, 72, 85), t2 = c(79, 81, 55, 68),
                  t3 = c(55, 69, 100, 25),
                  row.names = c("Maria", "José", "Lauro", "Lurdes"))
#-------------------------------------------------------------------------------
# Note como o resultado é o mesmo
da
```

```
##        t1 t2  t3
## Maria  16 79  55
## José   25 81  69
## Lauro  72 55 100
```

```
## Lurdes 85 68  25
```

```
da2
```

```
##        t1 t2  t3
## Maria  16 79  55
## José   25 81  69
## Lauro  72 55 100
## Lurdes 85 68  25
```

```
#-------------------------------------------------------------------------------
# Acrescentando uma coluna na tabela
# (quantidade de elementos = quantidade de linhas do data frame)
# Criar uma coluna com as médias entre t1, t2 e t3 de cada pessoa
# data.frame$nome.da.nova.coluna <- elementos.da.nova.coluna
# Função `apply(X, MARGIN, FUN)`, que serve para aplicar uma funçao
# a um conjunto de dados, vetor, matriz ou data frame.
# X = conjunto.de.dados
# MARGIN = 1(linha) ou 2(coluna)...
# FUN = nome.da.função
da2$Media <- apply(da2, 1, mean)
# Aqui está sendo dito que da2 vai receber uma nova coluna
# com a média dos vetores formados pelos valores das linhas
da2
```

```
##        t1 t2  t3    Media
## Maria  16 79  55 50.00000
## José   25 81  69 58.33333
## Lauro  72 55 100 75.66667
## Lurdes 85 68  25 59.33333
```

```
#-------------------------------------------------------------------------------
# Podem ser acrescentadas linhas ao data frame
# (quantidade de elementos = quantidade de colunas do data frame)
# Criar uma linha com uma nova pessoa e suas notas
# data.frame["nome.da.nova.coluna",] <- elementos.da.nova.coluna
da["Elvira",] <- c(50, 95, 75)
# Aqui está dizendo que da vai receber uma linha e as
# colunas desta linha vao receber, respectivamente, 50, 95 e 75
da
```

```
##        t1 t2  t3
## Maria  16 79  55
## José   25 81  69
## Lauro  72 55 100
## Lurdes 85 68  25
## Elvira 50 95  75
```

```
#-------------------------------------------------------------------------------
# Colunas podem ser apagadas de um data frame
# Remover a coluna t3 de da
# data.frame <- data.frame[, -número.da.coluna.a.ser.apagada]
da <- da[, -3]
# Aqui está dizendo para remover todas as linhas da terceira
# coluna do data frame
da
```

```
##        t1 t2
```

```
## Maria  16 79
## José   25 81
## Lauro  72 55
## Lurdes 85 68
## Elvira 50 95
```

```
#-------------------------------------------------------------------------------
# Linhas podem ser removidas de um data frame
# Remover a linha "Maria" de da
# data.frame <- data.frame[-número.da.linha.a.ser.apagada,]
da <- da[-1,]
# Aqui está dizendo para remover todas as colunas da primeira
# linha do data frame
da
```

```
##        t1 t2
## José   25 81
## Lauro  72 55
## Lurdes 85 68
## Elvira 50 95
```

```
#-------------------------------------------------------------------------------
# Para ver a classe de um objeto, basta utilizar `class(objeto)`
class(da2)
```

```
## [1] "data.frame"
```

```
class(da)
```

```
## [1] "data.frame"
```

```
#-----------------------------------------------------------------------
# Para ver a estrutura de um objeto, usar `str(objeto)`
str(da2)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ t1   : num  16 25 72 85
##  $ t2   : num  79 81 55 68
##  $ t3   : num  55 69 100 25
##  $ Media: num  50 58.3 75.7 59.3
```

```
str(da)
```

```
## 'data.frame':    4 obs. of  2 variables:
##  $ t1: num  25 72 85 50
##  $ t2: num  81 55 68 95
```

Listas

É uma coleção de objetos ordenados, onde cada objeto é um elemento da lista. Estes objetos não precisam ter o mesmo tipo ou mesma quantidade de elementos. Os elementos da lista vem numerados, mas podem ter nomes atribuídos a eles. Uma lista pode ser entendida como um armário onde cada objeto está em uma gaveta.

```
# Criando uma lista "Aluno" utilizando a função `list()`
# Onde nome.da.lista <- list(nome.do.elemento = conteúdo do elemento, ...)
aluno <- list(nr = 3654, nome = "Flávio Costa",
```

```
             notas = c(76, 58, 80))
aluno
```

```
## $nr
## [1] 3654
##
## $nome
## [1] "Flávio Costa"
##
## $notas
## [1] 76 58 80
```

```
# Note que cada elemento da lista é mostrado separadamente
# E que os elementos são todos de tipos diferentes
# nr = numérico, nome = string e notas = vetor numérico
#-------------------------------------------------------------------------------
# Também pode ser criada com objetos já existentes
l1 <- list(vet = v1, mat = m1, d.f = da)
l1
```

```
## $vet
## [1]  4  8 12
##
## $mat
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## $d.f
##        t1 t2
## José   25 81
## Lauro  72 55
## Lurdes 85 68
## Elvira 50 95
```

```
# Utilizando objetos de exemplos anteriores, foi criada uma lista
# com três elementos:
# vet = vetor, mat = matriz e d.f = data frame
#-------------------------------------------------------------------------------
# Para ver a classe de um objeto, basta utilizar `class(objeto)`
class(l1)
```

```
## [1] "list"
```

```
#-----------------------------------------------------------------
# Para ver a estrutura de um objeto, usar `str(objeto)`
str(l1)
```

```
## List of 3
##  $ vet: num [1:3] 4 8 12
##  $ mat: num [1:2, 1:2] 1 2 3 4
##  $ d.f:'data.frame': 4 obs. of  2 variables:
##   ..$ t1: num [1:4] 25 72 85 50
##   ..$ t2: num [1:4] 81 55 68 95
```

Arrays

São como matrizes, mas com mais de duas dimensões, ou seja, são vetores de 3 a X dimensões.

```
# Criar um array é semelhante a criar uma matriz
# Utilizando a função `array()`
# array(data = x, dim, y)
# data = uma sequência, lista ou vetor que irá ser usada para preencher
# o array
# dim = um vetor de inteiros que vai estipular o indice maximo de cada
# dimensão
# onde x = quantidade de elementos na primeira dimensão
ar1 <- array(data = 1:50, dim = c(2, 5, 5))
ar1
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
##
## , , 2
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   11   13   15   17   19
## [2,]   12   14   16   18   20
##
## , , 3
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   21   23   25   27   29
## [2,]   22   24   26   28   30
##
## , , 4
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   31   33   35   37   39
## [2,]   32   34   36   38   40
##
## , , 5
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   41   43   45   47   49
## [2,]   42   44   46   48   50
```

```
# No exemplo data = uma sequencia de 1 a 50 que irá preencher o array da
# mesma forma que uma matriz é preenchida, por colunas e pode-se
# entender que temos uma matriz de duas linhas e cinco colunas repetida
# cinco vezes, como é mostrado no exemplo
#-----------------------------------------------------------------------
# Para nomear um array é necessário uma lista, onde cada gaveta dela terá
# os nomes dos elementos de cada dimensão do array
# Criando a lista com os nomes
l2 <- list(c("A", "B"), c(LETTERS[3:7]), c(LETTERS[8:12]))
l2
```

```
## [[1]]
## [1] "A" "B"
##
## [[2]]
## [1] "C" "D" "E" "F" "G"
##
## [[3]]
## [1] "H" "I" "J" "K" "L"
```

```
# Nomeando ar1 com a função dimnames(nome.do.objeto) <- lista.com.os.nomes
dimnames(ar1) <- l2
ar1
```

```
## , , H
##
##   C D E F  G
## A 1 3 5 7  9
## B 2 4 6 8 10
##
## , , I
##
##    C  D  E  F  G
## A 11 13 15 17 19
## B 12 14 16 18 20
##
## , , J
##
##    C  D  E  F  G
## A 21 23 25 27 29
## B 22 24 26 28 30
##
## , , K
##
##    C  D  E  F  G
## A 31 33 35 37 39
## B 32 34 36 38 40
##
## , , L
##
##    C  D  E  F  G
## A 41 43 45 47 49
## B 42 44 46 48 50
```

```
#-----------------------------------------------------------------------
# Um array também pode ser nomeado quando criado
# Será um array com nome de três alunos e suas notas em duas provas de cada
# uma das quatro matérias
# Criando a lista com os nomes de cada elemento das dimensões
l3 <- list(c("P1", "P2"),
           c("Português", "Matemática", "História", "Geografia"),
           c("Viviane", "Belmiro", "Isabel"))
l3
```

```
## [[1]]
## [1] "P1" "P2"
##
```

```
## [[2]]
## [1] "Português"  "Matemática" "História"   "Geografia"
##
## [[3]]
## [1] "Viviane" "Belmiro" "Isabel"
```

```
# Criando um vetor com as notas
notas <- c(81, 70, 66, 58, 90, 76, 54, 89,
           79, 56, 45, 70, 92, 86, 70, 64,
           72, 80, 93, 84, 65, 61, 56, 78)
# Criando o array
ar2 <- array(data = notas, dim = c(2, 4, 3), dimnames = l3)
ar2
```

```
## , , Viviane
##
##    Português Matemática História Geografia
## P1        81         66       90        54
## P2        70         58       76        89
##
## , , Belmiro
##
##    Português Matemática História Geografia
## P1        79         45       92        70
## P2        56         70       86        64
##
## , , Isabel
##
##    Português Matemática História Geografia
## P1        72         93       65        56
## P2        80         84       61        78
```

```
#-----------------------------------------------------------------------------
# Para ver a classe de um objeto, basta utilizar `class(objeto)`
class(ar1)
```

```
## [1] "array"
```

```
class(ar2)
```

```
## [1] "array"
```

```
#-----------------------------------------------------------------------
# Para ver a estrutura de um objeto, usar `str(objeto)`
str(ar1)
```

```
##  int [1:2, 1:5, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : chr [1:2] "A" "B"
##   ..$ : chr [1:5] "C" "D" "E" "F" ...
##   ..$ : chr [1:5] "H" "I" "J" "K" ...
```

```
str(ar2)
```

```
##  num [1:2, 1:4, 1:3] 81 70 66 58 90 76 54 89 79 56 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : chr [1:2] "P1" "P2"
##   ..$ : chr [1:4] "Português" "Matemática" "História" "Geografia"
```

```
##   ..$ : chr [1:3] "Viviane" "Belmiro" "Isabel"
```

2.3 Funções e Argumentos

Função é um comando que executa alguma tarefa específica neste formato:

`função(argumento)`

função é onde vai o nome da função e **argumento** são valores ou métodos utilizados pela função.

Ex: Na função `mean()` que calcula a média aritmética de uma sequência numérica.

```
# Criando um objeto "a" que vai receber a sequência numérica
# utilizando a função "c()" que combina todos os argumentos
a <- c(1, 2, 3, 4, 5)
# "a" é um objeto com uma sequência numérica
a
```

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

```
# Usando a função mean para tirar a média aritmética da
# sequência
mean(a)
```

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

Para os argumentos, existe uma particularidade. Usando como exemplo a função `plot()`:

```
# Criar dois objetos para serem usados pela função `plot()`
adubo <- c(1:10)
cres <- c(1, 4, 5, 3, 2, 3, 5, 5, 1, 4)
# plot(x = , y = , xlab = "", ylab = "")
# x é a variável independente
# y é a variável dependente
# xlab e ylab são os nomes dados aos respectivos eixos na tabela
#-----------------------------------------------------------------
# Função `plot()` com todos argumentos explicitados
plot(x = adubo, y = cres, xlab = "Adubo", ylab = "Crescimento")
# Função `plot()` com x e y não explicitados, pois estão na ordem
# default(valor padrão), logo nao precisam ser explicitados
plot(adubo, cres, xlab = "Adubo", ylab = "Crescimento")
# Função `plot()` com todos os argumentos explicitados, mas com
# as ordens trocadas
plot(y = cres, x = adubo, ylab = "Crescimento", xlab = "Adubo")
# Das 3 maniras obten-se o mesmo resultado
```

OBS: Como os argumentos `xlab` e `ylab` são parâmetros genéricos para gráficos e não exclusivos da função `plot()`, devem ser sempre explicitados.

Criar Funções

No R as funções são objetos e podem ser manipulados de forma semelhante. Tem três característica principais: nome da função, lista de argumentos e e o corpo da função e são criadas usando a função `function()`com

esta estrutura:

```
nome.da.função <- function(lista de argumentos){
            corpo.da.função
}
```

Nome da função: O nome desejado para a função,. Recomenda-se um nome intuitivo, o mais curto possível e verificar se o nome já não é de outra função utilizando `help(nome.da.função)`, `?nome.da.função` ou digitando o nome da função na busca da aba `Help` do RStudio.

Argumentos: Os argumentos da função são os valores e parâmetros dados que serão utilizados pela função a ser criada, também podem ter valores por "default" (que são os valores que a função vai usar caso não sejam mudados ou fornecidos). Devem ser colocados como argumento da função `function()` e separados por vírgulas `,`. Podem ter qualquer nome, pois serão utilizados apenas dentro da função, mas recomenda-se nomes o mais intuitivos e curtos possíveis.

Corpo da função: Conjunto de instruções da linguagem R que realizarão os processamentos e é delimitado por duas chaves `{`<corpo.da.função>`}` e identados(como se fossem parágrafos, para que o R saiba que são comandos a serem executados dentro da função). O valor retornado pela função é o resultado do último comando do corpo da função ou através da função `return()`.Se, em algum momento do processamento da função for executado o `return()`, o processamento será interrompido e a função retornará o valor que estiver no `return()`.

Vejamos um exemplo simples da criação de uma função que recebe como argumento uma temperatura em graus Celsius e transforma em graus Fahrenheit.

```
# cel_far é o nome da função
# celsius é o argumento que será usado pela função cel_far
cel_far <- function(celsius){
     res <- (9/5) * celsius + 32
     res  # Esta é a última linha de processamento, portanto
          # a função retornará o valor contido no objeto `res`
}
# Um exemplo da utilização da função `cel_far()`
cel_far(25)
```

```
## [1] 77
```

OBS: Ao criar uma função, utiliza-se o mesmo procedimento de criação de um objeto, Utilizando `Ctrl + Enter`, que processa a linha e pula para a próxima, no console aparecerá um `+`, que significa q o R está em modo de espera(comandos incompletos), então deve-se continuar utilizando `Ctrl + Enter` nas próximas linhas. A função só será criada ao se processar o `}`.

Exemplo de como aparece no console:

```
> cel_far <- function(celcius){
+ res <- (9/5) * celcius + 32
+ res
+ }
```

Exemplo de criação de função com interrupção no `return()` em uma função que calcule raiz quadrada:

```
raiz <- function(num){
    if (num < 0)        # if é uma função que verifica a condição em seu
       return(NULL)     # argumento, no caso, se o número for menor que 0
    res <- sqrt(num)    # a função será interrompida e o valor `NULL`(Nulo)
    res                 # do return será retornado, caso contrário, o
}                       # processamento segue normalmente
#-------------------------------------------------------------------------
```

```r
# Exemplo com número negativo informado
raiz(-3)
```

```
## NULL
```

```r
# Exemplo com número positivo
raiz(4)
```

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

Ajuda

O R tem funções de ajuda. Existem algumas maneiras de acessá-la.

Caso não saiba o nome da função.

Existem 3 formas de encontrar a função que fará aquilo que você deseja. Por exemplo, tentar descobrir como calcular logaritmo no R:

1. Usando palavras chave como argumento para a função `help.search(palavra.chave)`:

```r
# Maneira errada
help.search("logarítmo")
# O argumento deve estar entre aspas (""), pois se trata de uma palavra
```

Note que assim não encontrá nenhum resultado, pois o R foi desenvolvido na língua inglesa, então a busca deve ser feita com palavras em inglês.

```r
# Maneira correta
help.search("logarithm")
# Lembrando que o argumento deve estar entre aspas (""), pois se trata
# de uma palavra
```

Assim o R irá procurar dentro dos arquivos de help funções para calcular logaritmos. Uma janela irá se abrir com as opções.

2. Nas versões mais atuais do R pode-se usar simplesmente `??palavra.chave`:

```r
# Lembrando que a pesquisa deve ser feita com palavras em inglês
??logarithm
# Obten-se o mesmo resultado da função help.search()
```

3. Também é possível buscar ajuda no site do R, pela internet com a função `RSiteSearch(palavra.chave)`:

```r
# Só funcionará se o computador estiver conectado a internet
RSiteSearch("logarithm")
```

Caso saiba o nome da função.

1. Usando a função `help(nome.da.função)`:

```r
# Agora se usa como argumento o nome da função sem aspas
help(log)
# vai abrir uma pagina de ajuda na aba `Viewer`
```

2. Usando `?nome.da.função`

```r
# Obten-se o mesmo resultado da função help()
?log
```

Busca de exemplos

Para obter somente exemplos de alguma função, utilizar a função `example(nome.da.função)`:

```r
# Vai mostrar todos os exemplos da função contidos na página de ajuda
example(log)
```

```
##
## log> log(exp(3))
## [1] 3
##
## log> log10(1e7) # = 7
## [1] 7
##
## log> x <- 10^-(1+2*1:9)
##
## log> cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
##              x
##  [1,] 1e-03 9.995003e-04 9.995003e-04 1.000500e-03 1.000500e-03
##  [2,] 1e-05 9.999950e-06 9.999950e-06 1.000005e-05 1.000005e-05
##  [3,] 1e-07 1.000000e-07 1.000000e-07 1.000000e-07 1.000000e-07
##  [4,] 1e-09 1.000000e-09 1.000000e-09 1.000000e-09 1.000000e-09
##  [5,] 1e-11 1.000000e-11 1.000000e-11 1.000000e-11 1.000000e-11
##  [6,] 1e-13 9.992007e-14 1.000000e-13 9.992007e-14 1.000000e-13
##  [7,] 1e-15 1.110223e-15 1.000000e-15 1.110223e-15 1.000000e-15
##  [8,] 1e-17 0.000000e+00 1.000000e-17 0.000000e+00 1.000000e-17
##  [9,] 1e-19 0.000000e+00 1.000000e-19 0.000000e+00 1.000000e-19
```

Pesquisa dos argumentos de uma função

Quando o interesse é ver os argumentos de uma função, utilizar a função `args(nome.da.função)`:

```r
# Mostra os argumentos da função com seus valores em "default", que são
# os valores que a função vai usar caso nao sejam mudados ou fornecidos
args(log)
```

```
## function (x, base = exp(1))
## NULL
```

2.4 Indexação e Seleção Condicional

Indexação

Indexação é a forma usada no R para selecionar `subsets`(sub-conjuntos).

Existem três operadores usados para tal:

- O operador `[ ]` retorna sempre um elemento do mesma classe do objeto original, podendo ser utilizado para selecionar múltiplos elementos de um objeto e o valor dentro do operador `[ ]` é chamado de Índice;
- O operador `[[ ]]` usado para extrair elementos de uma `lista` ou `data.frame`. Este elemento não precisa ser da mesma classe do objeto original;
- O operador `$` é usado para extrair elementos nomeados e é similar ao `[[ ]]`.

Vetores

```
# Utilizando o vetor `g` de exemplos anteriores
g
```

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

```
# Utilizando o operador `[ ]` para acessar o indice 3 do vetor
g[3]
```

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

```
# Neste caso o R retornarou o valor que está na terceira posição do vetor
# Extraindo o valor da terceira posição do vetor de caracteres `h`
h[3]
```

```
## [1] "C"
```

```
# ------------------------------------------------------------------------------
# No caso de ser fornecido um índice não condizente com uma posição
# do vetor, será retornado `NA`
g[10]
```

```
## [1] NA
```

```
h[15]
```

```
## [1] NA
```

```
#-------------------------------------------------------------------------------
# Para acessar múltiplos elementos, usa-se a função `c()`
g[c(1,2,4)]
```

```
## [1] 1 2 4
```

```
h[c(1,2,4)]
```

```
## [1] "A" "B" "D"
```

```
#-------------------------------------------------------------------------------
# Também pode ser utilizada qualquer função de gerar sequências
g[1:4]
```

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

```
h[1:4]
```

```
## [1] "A" "B" "C" "D"
```

```
# Selecionar os elementos com índice ímpar usando a função `seq()`
g[seq(1, 5, by = 2)]
```

```
## [1] 1 3 5
```

```r
h[seq(form = 1, to = 10, by = 2)]
```

```
## Warning: In seq.default(form = 1, to = 10, by = 2) :
##  extra argument 'form' will be disregarded
```

```
## [1] "A" "C" "E" "G" "I"
```

```r
# A função `seq(from = inicio.da.sequência, to = fim.da.sequência,
#               by = valor.do.incremento)`
# Nos exemplos acima são feitas sequências com o tamanho dos objetos
# com a função `seq()`, onde `from` e `to` são os argumentos que
# delimitam o intervalo da sequência e o argumento `by` serve para
# "pular" valores desta sequência.
# No caso do vetor `g` foi pedido uma sequência de 1 a 5, mas que
# só fossem retornados os valores de dois em dois, logo a função
# `seq()` retornou os valores 1, 3 e 5 para serem usados pelo
# operador `[ ]`, que por sua vez retorná os valores contidos nos
# índices 1, 3 e 5 do vetor `g`
# Também é possível criar a sequência em um objeto a parte
ind <- seq(1, 10, by = 2)
h[ind]
```

```
## [1] "A" "C" "E" "G" "I"
```

```r
# Note que o resultado é o mesmo do exemplo anterior
#-------------------------------------------------------------------------------
# Pode-se selecionar elementos exceto os que estão no índice
# utilizando o sinal `-`
g[-4]
```

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

```r
h[-5]
```

```
## [1] "A" "B" "C" "D" "F" "G" "H" "I" "J"
```

```r
# Também em seqências
g[-c(1, 3, 5)]
```

```
## [1] 2 4
```

```r
h[-ind]
```

```
## [1] "B" "D" "F" "H" "J"
```

Vetores nomeados

Quando um vetor tem seus elementos nomeados é possível realizar a indexação usando estes nomes

```r
# Dando nome aos elementos do vetor `g`
names(g) <- letters[1:length(g)]
# A função `names(objeto)` é uma função genérica para nomear vetores
# usando outro vetor
# `letters[1:length(g)]` vai gerar um vetor com a sequência entre 1
# e o tamanho do vetor `g`, pois a função `length(objeto)` retorna
# o tamanho do objeto usado como argumento
# O comando acima diz para o R nomear o cvetor `g` com as letras
# minúscolas de "a" até o tamanho do vetor `g`
g
```

```
## a b c d e
## 1 2 3 4 5
```

```
# Fazendo a busca pelo nome
g["d"]
```

```
## d
## 4
```

```
# Lembrando que, como os nomes são caracteres, devem estar entre `""`
```

Acrescentar, modificar e remover elementos de um vetor

Utilizando o operador [ ] é possível acrescentar, modificar e remover elementos de um vetor.

```
# Acrescentando um elemento ao vetor `g`
# Basta utilizar `vetor[índice.não.existente] <- valor`
g[6] <- 20
g
```

```
##  a  b  c  d  e
##  1  2  3  4  5 20
```

```
#-------------------------------------------------------------------------------
# Caso seja colocado um índice o qual deixe um intervalo entre
# o tamanho do vetor e o índice, os elementos desse intervalo
#serão preenchidos automáticamente com `NA`
g[15] <- 42
g
```

```
##  a  b  c  d  e
##  1  2  3  4  5 20 NA NA NA NA NA NA NA NA 42
```

```
#-------------------------------------------------------------------------------
# Usando seqências
g[16:20] <- 14
g
```

```
##  a  b  c  d  e
##  1  2  3  4  5 20 NA NA NA NA NA NA NA NA 42 14 14 14 14 14
```

```
#-------------------------------------------------------------------------------
# Nomeando o vetor `g`
names(g) <- letters[1:length(g)]
g
```

```
##  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t
##  1  2  3  4  5 20 NA NA NA NA NA NA NA NA 42 14 14 14 14 14
```

```
#-------------------------------------------------------------------------------
# Modificando valores
# O processo é parecdido, mas com a diferença que se utiliza
# índices já existentes no vetor
g[14] <- 31
g
```

```
##  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t
```

```
##  1  2  3  4  5 20 NA NA NA NA NA NA NA 31 42 14 14 14 14 14
```

```
#-------------------------------------------------------------------------------
# Usando sequências
g[16:20] <- c(12, 23, 8, 14, 57)
g
```

```
##  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t
##  1  2  3  4  5 20 NA NA NA NA NA NA NA 31 42 12 23  8 14 57
```

```
#-------------------------------------------------------------------------------
# Usando o nome
g["b"] <- 99
g
```

```
##  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t
##  1 99  3  4  5 20 NA NA NA NA NA NA NA 31 42 12 23  8 14 57
```

```
#-------------------------------------------------------------------------------
# Para retirar elementos se utiliza `-` com uma estrutura
# um pouco diferente
# `vetor <- vetor[-índice.do.elemento.a.ser.retirado]`
g <- g[-14]
g
```

```
##  a  b  c  d  e  f  g  h  i  j  k  l  m  o  p  q  r  s  t
##  1 99  3  4  5 20 NA NA NA NA NA NA NA 42 12 23  8 14 57
```

```
length(g) # Verificando o novo tamanho do vetor
```

```
## [1] 19
```

```
#-------------------------------------------------------------------------------
# Usando sequência
g <- g[-c(14, 15, 16, 17, 18, 19)]
g
```

```
##  a  b  c  d  e  f  g  h  i  j  k  l  m
##  1 99  3  4  5 20 NA NA NA NA NA NA NA
```

```
length(g) # Verificando o novo tamanho do vetor
```

```
## [1] 13
```

Matrizes

```
# Assim como para vetores, para matrizes é usado o operador `[ ]`
# Mas como a matriz tem duas dimensões da seguinte maneira:
# matriz [índice.da.linha, índice.da.coluna]
# Utilizando a matriz `m3` de exemplos anteriores
m3
```

```
##        P1 P2 P3
## aluno1 92 40 70
## aluno2 70 88 75
```

```r
# Acessando o elemento da linha 2 e coluna 3
m3[2, 3]
```

```
## [1] 75
```

```r
#-------------------------------------------------------------------------------
# Para os elementos da linha 2 e colunas 1 e 2, usa-se uma sequência
m3[2, c(1, 2)]
```

```
## P1 P2
## 70 88
```

```r
#-------------------------------------------------------------------------------
# Acessando todos os elementos da primeira linha
m3[1, c(1, 2, 3)]
```

```
## P1 P2 P3
## 92 40 70
```

```r
# Todos os elementos da segunda coluna
m3[c(1, 2), 2]
```

```
## aluno1 aluno2
##     40     88
```

```r
#-------------------------------------------------------------------------------
# Existe uma maneira mais fácil de acessar todos os elementos de uma
# ou mais linhas e colunas de uma matriz
# `matriz[índice.da.linha,]` ou `matriz[, índice.da.coluna]`
# Assim estará sendo pedido ao R que retorne todos os elementos de
# uma ou mais linhas ou colunas
# Repetindo os exemplos anteriores
# Acessando todos os elementos da primeira linha
m3[1,]
```

```
## P1 P2 P3
## 92 40 70
```

```r
# Todos os elementos da segunda coluna
m3[, 2]
```

```
## aluno1 aluno2
##     40     88
```

```r
# Note que os resultados são os mesmos
#-------------------------------------------------------------------------------
# Acessando todos os elementos de mais de uma linha ou coluna
m3[, c(2, 3)]
```

```
##        P2 P3
## aluno1 40 70
## aluno2 88 75
```

```r
#-------------------------------------------------------------------------------
# Note que o R sermpre retorna em formato de vetor
# Caso seja necessário que seja retoirnado sem perder o formato de matriz
# utiliza-se o argumento `drop = FALSE` dentro do operador `[ ]`, assim
# o R manterá a estrutura de matriz na resposta
m3[2, c(1, 2)]
```

```
## P1 P2
## 70 88
```

```
m3[2, c(1, 2), drop = FALSE]
```

```
##        P1 P2
## aluno2 70 88
```

```
m3[c(1, 2), 2]
```

```
## aluno1 aluno2
##     40     88
```

```
m3[c(1, 2), 2, drop = FALSE]
```

```
##        P2
## aluno1 40
## aluno2 88
```

```
# Note a diferença nos resultados
```

Matrizaes nomeadas

Quando matrizes estão nomeadas, pode-se acessar seus elementos usando os nomes.

```
# Acessando todas as notas de `aluno1`
m3["aluno1",]
```

```
## P1 P2 P3
## 92 40 70
```

```
# Acessando P2 para todas as linhas
m3[, "P2"]
```

```
## aluno1 aluno2
##     40     88
```

```
# Pode-se usar os dois tipos de índices para indexar
m3["aluno2", c(1, 2), drop = FALSE]
```

```
##        P1 P2
## aluno2 70 88
```

Acrescentar e remover linhas e colunas a uma matriz

Para acrescentar linhas ou colunas em uma matriz, usa-se as funções `rbind()` para linhas e `cbind()` para colunas, já para apagar linhas e colunas, o processo é similar ao de vetores com o operador `[ ]`.

```
# Acrescentando uma linha na matriz `m3` usando `rbind()`
# `matriz <- rbind(matriz, valores)` OBS: a quantidade de valores
# deve ser a mesma do número de colunas da matriz
m3 <- rbind(m3, c(72, 86, 64))
m3
```

```
##        P1 P2 P3
## aluno1 92 40 70
## aluno2 70 88 75
##        72 86 64
```

```r
#-------------------------------------------------------------------------------
# Para acrescentar uma coluna, usa-se a função `cbind()`
# `matriz <- cbind(matriz, valores)` OBS: a quantidade de valores
# deve ser a mesma do número de linhas da matriz
m3 <- cbind(m3, c(25, 37, 50))
m3
```

```
##        P1 P2 P3
## aluno1 92 40 70 25
## aluno2 70 88 75 37
##        72 86 64 50
```

```r
#-------------------------------------------------------------------------------
# Para remover lionhas e colunas de uma matriz o processo é similar
# ao com vetores, usando `-`
# Removendo a terceira linha da matriz `m3`
m3 <- m3[-3,]
m3
```

```
##        P1 P2 P3
## aluno1 92 40 70 25
## aluno2 70 88 75 37
```

```r
# `matriz <- matriz[-índice.da.linha,]`
#-------------------------------------------------------------------------------
# Removendo a quarta coluna da matriz `m3`
m3 <- m3[, -4]
m3
```

```
##        P1 P2 P3
## aluno1 92 40 70
## aluno2 70 88 75
```

```r
# `matriz <- matriz[, -índice.da.coluna]`
```

Data frames

```r
# Como o data frame tem duas dimensões, segue a mesma lógica das matrizes
# Usando o operador `[ ]`
# Elemento da primeira linha e segunda coluna
da[1, 2]
```

```
## [1] 81
```

```r
# Todos elementos da segunda coluna
da[, 2]
```

```
## [1] 81 55 68 95
```

```r
# Os três primeiros elementos da primeira coluna
da[1:3, 1]
```

```
## [1] 25 72 85
```

```r
# Também pode ser acessado pelos nomes
da["Lurdes", "t2"]
```

```
## [1] 68
```

```
#-------------------------------------------------------------------------------
# Pode ser usado o operador `[[ ]]` que retorna uma coluna do data frame
# data.frame[[índice.da.coluna]]
da[[2]]
```

```
## [1] 81 55 68 95
```

```
da[["t1"]]
```

```
## [1] 25 72 85 50
```

```
# Para acessar o terceiro elemento da segunda coluna com o formato
# data.frame[[índice.da.coluna]][índice.do.vetor]
# O operador `[[ ]]` extrai a colunda do data frame e retorna um vetor
# com os elementos desta coluna, então basta acessar o elemento do
# vetor seguindo a indexação de vetor
da[["t2"]][3]
```

```
## [1] 68
```

```
# Os dois últimos elementos da coluna 1
da[[1]][c(3, 4)]
```

```
## [1] 85 50
```

```
#-------------------------------------------------------------------------------
# Existe a opção de usar o operador `$` que tem a mesma função do `[[ ]]`
# mas somente com o nome da coluna
# data.frame$nome.da.coluna[índice]
# Os mesmos exemplos anteriores com o operador `$`
da$t2
```

```
## [1] 81 55 68 95
```

```
da$t1
```

```
## [1] 25 72 85 50
```

```
da$t2[3]
```

```
## [1] 68
```

```
da$t1[c(3, 4)]
```

```
## [1] 85 50
```

```
# Note como os resultados são os mesmos
```

Lista

```
# Para indexação de listas é semelhante ao data frame usando `[[ ]]` e `$`
# Usando a lista `l1` dos exemplos anteriores
l1
```

```
## $vet
## [1]  4  8 12
##
```

```
## $mat
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## $d.f
##        t1 t2
## José   25 81
## Lauro  72 55
## Lurdes 85 68
## Elvira 50 95
```

```
#-------------------------------------------------------------------------------
# A diferença da lista para o data frame é que não é acessado uma coluna e sim
# um dos componentes da lista com `[[ ]]` ou `$` usando o nome destes componentes
# que podem ser vetores, data frames, matrizes, etc e então acessar os elementos
# destes componentes conforme o tipo
# `l1` é composta por um vetor, uma matriz e um data frame
l1[[1]]
```

```
## [1]  4  8 12
```

```
l1[["mat"]]
```

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

```
l1$d.f
```

```
##        t1 t2
## José   25 81
## Lauro  72 55
## Lurdes 85 68
## Elvira 50 95
```

```
#-------------------------------------------------------------------------------
# Acessando elementos dos componentes da lista
# Extraindo os elementos 1 e 2 do primeiro componente de `l1` que é um vetor
l1[["vet"]][c(1, 2)]
```

```
## [1] 4 8
```

```
# Extraindo o elemento da primeira linha e segunda coluna da matriz que é
# o segundo componente da lista
l1[[2]][1, 2]
```

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

```
# Acessando a o primeiro elemento da segunda coluna do data frame
l1$d.f$t2[1]
```

```
## [1] 81
```

```
# No exemplo acima o primeiro `$` extrai o data frame da lista e o
# segundo `$` extrai a segunda coluna do data frame em formato de vetor,
# então o primeiro elemento é indexado como em um vetor
```

Seleção condicional

Nos exemplos anteriores era sempre dado um índice já conhecido para buscar um elemento. Já na seleção condicional é feita uma varredura no objeto para um certo elemento ou elementos que obedeçam as condições da expressão condicional usando os seguintes operadores:

- `>` maior que;
- `<` menor que;
- `>=` maior ou igual a;
- `<=` menor ou igual a;
- `==` igual a;
- `!=` diferente de;
- `!` negação lógica;
- `&` e;
- `|` ou;
- `%in%` contido em.

As expressões condicionais podem conter quantos operadores forem necessários

```
# Verificar quais elementos de `g` são maiores que 18
g > 18
```

```
##     a     b     c     d     e     f     g     h     i     j     k     l
## FALSE  TRUE FALSE FALSE FALSE  TRUE    NA    NA    NA    NA    NA    NA
##     m
##    NA
```

```
# Como se trata de uma seleção condicional, é retornado de forma
# binária com `TRUE`(verdade) e `FALSE`(falso)
#-------------------------------------------------------------------------------
# Para serem retornados os índices usa-se a função `which()`
which(g > 18)
```

```
## b f
## 2 6
```

```
# Na função `which(seleção.condicional)` é retornado os índices
# do objeto que atendem as condições e os valores `NA` são omitidos,
# pois são tratados como `FALSE`
#-------------------------------------------------------------------------------
# Se a intenção é extrair os valores dos elementos, usar a indexação com
# a função `which()` ou com a seleção condicional aliada a indexação
g[which(g > 18)]
```

```
##  b  f
## 99 20
```

```
g[g > 18]
```

```
##    b    f <NA> <NA> <NA> <NA> <NA> <NA> <NA>
##   99   20   NA   NA   NA   NA   NA   NA   NA
```

```
# Note que no segundo caso os elementos `NA` não são omitidos do resultado
#-------------------------------------------------------------------------------
# Buscar elementos menores que 5 ou maiores ou iguais a 10
g < 5 | g >= 10 # Retorna `TRUE` e `FALSE`
```

```
##     a     b     c     d     e     f     g     h     i     j     k     l
##  TRUE  TRUE  TRUE  TRUE FALSE  TRUE    NA    NA    NA    NA    NA    NA
```

```
##     m
##    NA
```

```
which(g < 5 | g >= 10) # Retorna os índices dos resultados `TRUE`
```

```
## a b c d f
## 1 2 3 4 6
```

```
g[which(g < 5 | g >= 10)] # Retorna os valores onde o resultado foi `TRUE`
```

```
##  a  b  c  d  f
##  1 99  3  4 20
```

```
#-------------------------------------------------------------------------------
# Caso tente uma condição a qual nenhum elemento obedeça
# Elementos menores que 5 e maiores que 30
g < 5 & g > 30
```

```
##     a     b     c     d     e     f     g     h     i     j     k     l
## FALSE FALSE FALSE FALSE FALSE FALSE    NA    NA    NA    NA    NA    NA
##     m
##    NA
```

```
which(g < 5 & g > 30)
```

```
## named integer(0)
```

```
g[which(g < 5 & g > 30)]
```

```
## named numeric(0)
```

```
# Não funciona, pois não existe nenhum elmento q seja menor que 5 e
# maior que 30 ao memso tempo
#-------------------------------------------------------------------------------
# Pode ser usado um objeto para fazer a busca em outro
# Criando um vetor `j` do mesmo tamanho de `g`
j <- LETTERS[1:length(g)]
j
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M"
```

```
# Fazer a busca dos valores de g onde `j` é igual a "D" ou "K"
j == "D" | j == "K"
```

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

```
g[j == "D" | j == "K"]
```

```
##  d  k
##  4 NA
```

```
g[j %in% c("D", "K")]
```

```
##  d  k
##  4 NA
```

```
#-------------------------------------------------------------------------------
# Usando intervalos
g %in% 3:20
```

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

```
## [12] FALSE FALSE
```

```
which(g %in% 3:20)
```

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

```
g[which(g %in% 3:20)]
```

```
##  c  d  e  f
##  3  4  5 20
```

```
#-------------------------------------------------------------------------------
# Buscar usando `$` no data frame `da` as notas de `t2` maiores que a média 70
da$t2 > 70
```

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

```
which(da$t2 > 70)
```

```
## [1] 1 4
```

```
da$t2[which(da$t2 > 70)]
```

```
## [1] 81 95
```

2.5 Valores Perdidos e Especiais

- `NA` Not Available (não disponível), constante lógica que contém um valor perdido;
- `NULL` Nulo, palavra reservada e geralmente retornada por expressões ou funções com valor indefinido;
- `NaN` Not a number (não é um número), exemplo: $\frac{0}{0}$;
- `-Inf` e `Inf` Infinite (infinito), exemplo: $\frac{1}{0}$;

```
# Para testar se um objeto tem valores `NA`, usa-se a função
# `is.na(objeto)`
is.na(g)
```

```
##     a     b     c     d     e     f     g     h     i     j     k     l
## FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##     m
##  TRUE
```

```
which(is.na(g))
```

```
##  g  h  i  j  k  l  m
##  7  8  9 10 11 12 13
```

```
g[which(is.na(g))]
```

```
##  g  h  i  j  k  l  m
## NA NA NA NA NA NA NA
```

```
#-------------------------------------------------------------------------------
# Também pode ser usado para indexação como no caso do exemplo anterior onde
# foi buscado valores maiores que 18 no vetor `g` e os `NA` também foram
# retornados. Utilizando `ia.na()` precedido do operador `!`
g[ g > 18 & !is.na(g)]
```

```
##  b  f
## 99 20
```

```
#-------------------------------------------------------------------------------
# Paraq fazer operações sem correr o risco de interferência dos `NA`
# de um objeto, existe um argumento `na.rm = TRUE` que pode ser usado
# pela maioria das funções do R e desconsidera os `NA` ao serem executadas
mean(g, na.rm = TRUE)
```

```
## [1] 22
```

```
#-------------------------------------------------------------------------------
# Para verificar se existe algum `NaN` em um objeto usa-se a função
# `is.nan(objeto)`
n <- 0/0
is.nan(n)
```

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

```
n
```

```
## [1] NaN
```

```
#-------------------------------------------------------------------------------
# Para verificar se existe algum `Inf` em um objeto usa-se a função
# `is.infinite(objeto)`
o <- 1/0
p <- -1/0
is.infinite(o)
```

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

```
is.infinite(p)
```

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

```
o
```

```
## [1] Inf
```

```
p
```

```
## [1] -Inf
```
