

# Estrutura da linguagem R

Prof. Dr. Wagner Hugo Bonat

# Estrutura e objetivos do módulo

# Estrutura do módulo

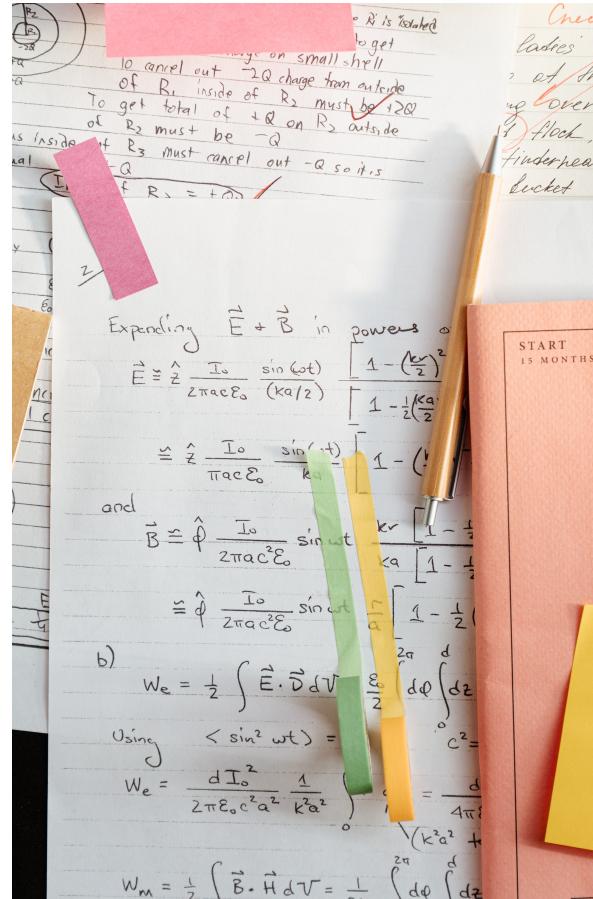
- ▶ Conhecendo a linguagem R.
- ▶ Primeiros passos.
- ▶ Arquivos, pacotes e documentação.
- ▶ Operações aritméticas e lógicas.
- ▶ Estruturas de dados.
- ▶ Vetores atômicos.
- ▶ Matrizes, listas e tabelas.
- ▶ Estruturas de programação.
- ▶ Estruturas de controle.
- ▶ Estruturas de repetição.
- ▶ Funções.
- ▶ Funções avançadas.



<https://www.pexels.com/photo/top-view-of-people-at-the-meeting-3184287/>

# Objetivos do módulo

- ▶ Instalar e configurar o ambiente de trabalho R.
- ▶ Entender onde o R trabalha e como interage com o computador.
- ▶ Dominar a instalação de pacotes adicionais.
- ▶ Saber consultar a vasta documentação do R.
- ▶ Entender as principais estruturas de dados.
- ▶ Dominar as principais estruturas de programação.
- ▶ Fazer funções para automação de tarefas.
- ▶ Compreender a necessidade e as principais técnicas para tratamento de exceções.



# Conhecendo a linguagem R

# Conhecendo a linguagem



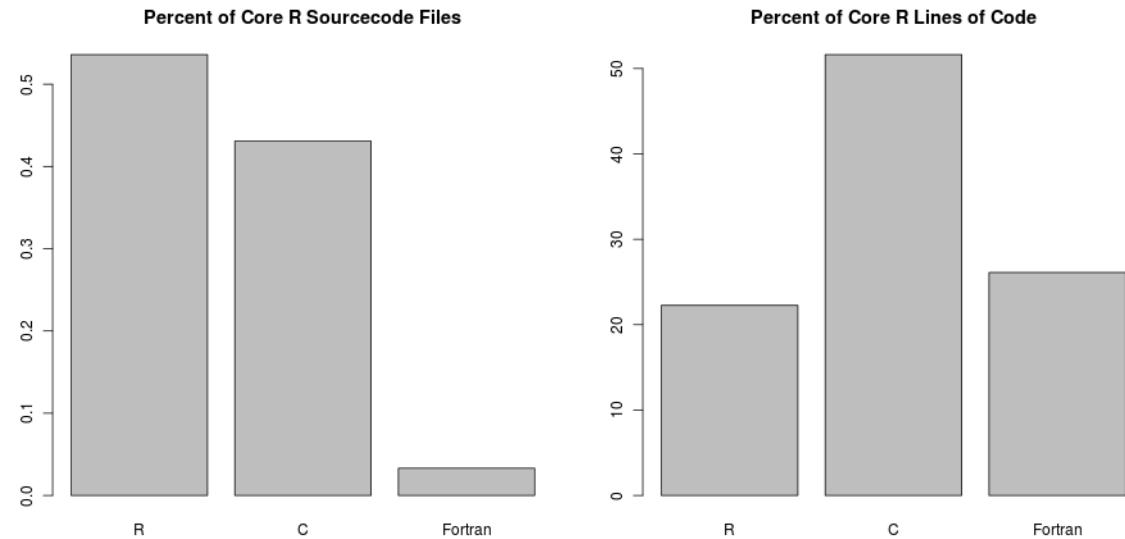
Logo do R.

- ▶ É uma linguagem de programação **livre** e de código **aberto** (um projeto GNU).
- ▶ É um ambiente de software para **computação estatística e gráficos**.
- ▶ Linguagem de **propósito específico**.
- ▶ <https://www.r-project.org/about.html>.

# Características

O R fornece amplo ferramental de estatística além de:

- ▶ Inúmeros recursos gráficos.
- ▶ Extensível: coleção de mais de 18 mil **pacotes** oficiais no [CRAN](#).
- ▶ Interoperabilidade & Interconectividade: **APIs/drivers** para outros softwares/linguagens.
- ▶ Multiplataforma: Linux, Mac OS, Windows, Android.
- ▶ Escalabilidade e código compilado: C e Fortran.
- ▶ Boa parte do código fonte do *R base* está em C e Fortran.



<https://librestats.wordpress.com/2011/08/27/how-much-of-r-is-written-in-r/>

# O Projeto R



[Home]

**Download**

CRAN

**R Project**

About R

Logo

Contributors

What's New?

Reporting Bugs

Conferences

Search

Get Involved: Mailing Lists

Developer Pages

R Blog

# The R Project for Statistical Computing

## Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

## News

- [R version 4.1.0 \(Camp Pontanezen\) prerelease versions](#) will appear starting Saturday 2021-04-17. Final release is scheduled for Tuesday 2021-05-18.
- [R version 4.0.5 \(Shake and Throw\)](#) has been released on 2021-03-31.
- Thanks to the organisers of useR! 2020 for a successful online conference. Recorded tutorials and talks from the conference are available on the [R Consortium YouTube channel](#).
- [R version 3.6.3 \(Holding the Windsock\)](#) was released on 2020-02-29.
- You can support the R Foundation with a renewable subscription as a [supporting member](#)

<https://www.r-project.org/>

# Como obter o R?



[CRAN](#)  
[Mirrors](#)  
[What's new?](#)  
[Task Views](#)  
[Search](#)

*About R*  
[R Homepage](#)  
[The R Journal](#)

*Software*  
[R Sources](#)  
[R Binaries](#)  
[Packages](#)  
[Other](#)

*Documentation*  
[Manuals](#)  
[FAQs](#)  
[Contributed](#)

## The Comprehensive R Archive Network

### Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

### Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2021-03-31, Shake and Throw) [R-4.0.5.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

<https://cran.r-project.org/>

# Pacotes agrupados por tarefas/áreas

Topics	
<a href="#">Bayesian</a>	Bayesian Inference
<a href="#">ChemPhys</a>	Chemometrics and Computational Physics
<a href="#">ClinicalTrials</a>	
<a href="#">Cluster</a>	
<a href="#">Databases</a>	
<a href="#">DifferentialEquations</a>	
<a href="#">Distributions</a>	
<a href="#">Econometrics</a>	
<a href="#">Environmetrics</a>	
<a href="#">ExperimentalDesign</a>	
<a href="#">ExtremeValue</a>	
<a href="#">Finance</a>	
<a href="#">FunctionalData</a>	
<a href="#">Genetics</a>	
<a href="#">Graphics</a>	
<a href="#">HighPerformanceComputing</a>	
<a href="#">Hydrology</a>	
<a href="#">MachineLearning</a>	
<a href="#">MedicalImaging</a>	
<a href="#">MetaAnalysis</a>	
<a href="#">MissingData</a>	Missing Data
<a href="#">ModelDeployment</a>	Model Deployment with R
<a href="#">Multivariate</a>	Multivariate Statistics
<a href="#">NaturalLanguageProcessing</a>	Natural Language Processing
<a href="#">NumericalMathematics</a>	Numerical Mathematics
<a href="#">OfficialStatistics</a>	Official Statistics & Survey Methodology
<a href="#">Optimization</a>	Optimization and Mathematical Programming
<a href="#">Pharmacokinetics</a>	Analysis of Pharmacokinetic Data
<a href="#">Phylogenetics</a>	Phylogenetics, Especially Comparative Methods
<a href="#">Psychometrics</a>	Psychometric Models and Methods
<a href="#">ReproducibleResearch</a>	Reproducible Research
<a href="#">Robust</a>	Robust Statistical Methods
<a href="#">SocialSciences</a>	Statistics for the Social Sciences
<a href="#">Spatial</a>	Analysis of Spatial Data
<a href="#">SpatioTemporal</a>	Handling and Analyzing Spatio-Temporal Data
<a href="#">Survival</a>	Survival Analysis
<a href="#">TeachingStatistics</a>	Teaching Statistics
<a href="#">TimeSeries</a>	Time Series Analysis
<a href="#">WebTechnologies</a>	Web Technologies and Services
<a href="#">gR</a>	gRaphical Models in R

<https://cran.r-project.org/web/views/>

# Download e instalação do R

## Roteiro

Instalar a linguagem R.

- ▶ R é a **linguagem de programação**.
- ▶ Faz o processamento/análise de dados.

Downloads do R:

<http://cran.r-project.org/>.

## Links para downloads

### ▶ Windows

- ▶ Baixar o `*.exe` e instalar com qualquer outro software.
- ▶ Instruções em: `base` para Windows.
- ▶ Print screens do processo aqui: [r-installation-screenshots](#).

### ▶ Mac OS X

- ▶ Seguir instruções no CRAN: [Mac OS X](#).
- ▶ Tutorial de instalação (en): [install-r-and-rstudio-on-mac](#)

### ▶ GNU Linux

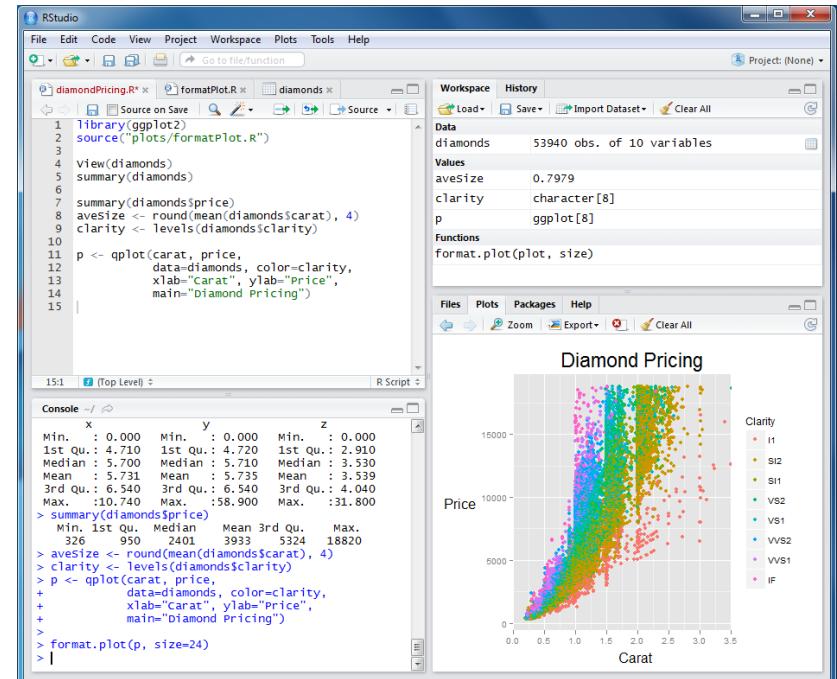
- ▶ Conforme a distribuição, seguir instruções em: [GNU Linux](#).

# IDEs/Editores para R

## Roteiro

Instalar um editor/IDE de sua preferência.

- ▶ **IDEs:** ambiente integrado de desenvolvimento.
  - ▶ RStudio.
  - ▶ RCode.
  - ▶ Tinn-R.
- ▶ **Editores:** são gerais e podem ser habilitadas para as linguagens usando plugins.
  - ▶ GNU Emacs, Spacemacs ou Doom-Emacs usando ESS.
  - ▶ Vim.
  - ▶ Visual Studio Code (VS Code).



Print screen do RStudio.

# Primeiros passos com o R

# Usando o R pela primeira vez

- ▶ **CLI** - *Command Line Interface.*
- ▶ **REPL** - *Read, Eval, Print and Loop.*



A sensação de abrir o R pela primeira vez.

# Instruções na linguagem R

## Modos de uso

### ► Modo **REPL**

- Script com instruções (receita).
- Instruções avaliadas no console (interpretador).
- Analista supervisiona o processo.
- Salva/duplica/modifica o script conforme necessidade.

### ► Modo **Batch**

- Script poder ser executado sem supervisão.
- Usado em ambientes de produção ou simulação computacional.
- Exemplo: \$ Rscript -f minha\_simulação.R.

## Comentários e instruções

- Tudo que vem após # é comentário.
- Eles documentam o que o código faz.

```
# Faz uma soma.  
2 + 2
```

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

```
# Meu índice de massa corporal.  
83/1.85^2
```

```
## [1] 24.25128
```

```
# Quantos segundos tem um dia?  
24 * 60 * 60
```

```
## [1] 86400
```

# InSTRUÇÕES e comentÁRIOS

## Faça comentários relevantes

- ▶ Evite comentários óbvios.
- ▶ Evite comentários ambíguos.
- ▶ É melhor errar pelo excesso.

## InSTRUÇÕES

- ▶ Podem ocupar uma única linha.
- ▶ Podem ocupar várias linhas.
- ▶ Uma linha pode ter várias instruções.

## Recomendações

- ▶ Evite ultrapassar 72 ou 80 caracteres.
- ▶ Mantenha o código devidamente indentado (`CTRL + i`).
- ▶ Evite muitas instruções em uma linha.

# Uma única linha.

```
2 + 2 + 7 + 5
```

```
## [1] 16
```

# Em várias linhas.

```
2 +  
 2 +  
 7 +  
 5
```

```
## [1] 16
```

# Várias em uma linha.

```
2 + 2; 7 + 5
```

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

```
## [1] 12
```

# Área ou espaço de trabalho

## Criação de objetos

- Ao fazer **atribuições**, são criados objetos na área de trabalho.
- Usa-se `<-` para atribuir algo para um objeto (`alt` + `-`)
- Objetos podem ser reusados para criar outros (essa é a intenção).
- Objetos podem ser sobreescritos e apagados

```
peso <- 83      # Meu peso (kg).
altura <- 1.80 # Minha altura (m)
imc <- peso/altura^2
imc
## [1] 25.61728

# Lista objetos na área de trabalho.
ls()
## [1] "altura" "imc"     "peso"

# Apaga objetos.
rm(peso, altura)
ls()
## [1] "imc"
```

# Espaços de trabalho

Onde o R procura por objetos?

- ▶ `ls()` mostra o conteúdo do `.GlobalEnv`, a área de trabalho.
- ▶ Mas existem objetos em outros ambientes ou espaços.
- ▶ Esses são os espaços dos pacotes.
- ▶ Quando não encontra no `.GlobalEnv`, ele vai para o próximo espaço de trabalho.
- ▶ `search()` retorna a lista de espaços de trabalho.
- ▶ Cada pacote tem o seu espaço (*namespace*).

```
# Mas `women` não está no `GlobalEnv`.
# De onde veio?
women

# Mostra o .GlobalEnv.
ls()

# Mostra os demais espaços.
search()

# Lista o conteúdo de um espaço.
ls("package:datasets")

# O que acontece se eu fizer?
women <- c("Gertrude Mary Cox",
        "Florence Nightingale David")

women           # Está no .GlobalEnv.
datasets::women # Está no pacote.
```

# Diretório de trabalho

- ▶ **Diretório de trabalho** é o local no sistema operacional para onde o R está apontando.
- ▶ Isto é, de onde ele lê e escreve arquivos por padrão.
- ▶ Você pode definir por comandos (recomendável) ou usando *RStudio IDE > Session > Setting Working Directory*.

```
# Mostra o atual diretório de trabalho.  
getwd()  
  
# Troca por outro endereço.  
setwd("~/Downloads")  
  
# Lista o conteúdo do diretório de trabalho.  
dir()
```

# Arquivos, pacotes e documentação do R

# Arquivos da linguagem R

Existem arquivos relacionados à linguagem R que servem para melhorar a experiência do usuário.

- ▶ **.Rhistory:**

Arquivo texto que salva o histórico de instruções executadas.

- ▶ **.RData:**

Arquivo binário que salva os objetos da área de trabalho.

- ▶ Serve para restaurá-los.

- ▶ Útil quando o processamento é demorado.

- ▶ **.Rproj:**

Arquivo que define configurações do projeto (para RStudio IDE apenas).

- ▶ Eles existem no diretório de trabalho.

- ▶ Uso pode ser configurado em *RStudio IDE > Tools > Global Options*.

- ▶ **.Rprofile:**

Arquivo de configurações lido no início das seções.

- ▶ Carregar pacotes muito usados e configurar opções.

- ▶ Mensagem de boas-vindas e diagnóstico do sistema.

- ▶ Pode ser definido por projeto ou por usuário.

# Instalação de pacotes

## Pacotes

- ▶ Pacotes são coleções de funções e conjuntos de dados organizados e documentados.
- ▶ Um pacote contém código R e eventualmente códigos de outras linguagens.
- ▶ Pacotes podem depender de *libs* do sistema operacional.

## Formas de instalação

- ▶ Pacotes podem ser instalados de repositórios: CRAN, Bioconductor, MRAN, etc.
- ▶ De arquivos de instalação **.\*.tar.gz**.
- ▶ De repositórios Git: GitHub, GitLab, etc.
- ▶ Para mais, visite r-packages-guide.

```
# Para instalar um pacote do repositório.  
install.packages("tidyverse")  
  
# Para carregar o pacote e usá-lo.  
library(tidyverse)  
  
# Para ver o conteúdo dele.  
ls("package:tidyverse")  
  
# Documentação do pacote.  
help(package = "tidyverse")  
  
# Para ver onde foi instalado.  
system.file(package = "tidyverse")  
  
# Os caminhos para endereços de instalação.  
.libPaths()  
  
# Para remover o pacote da sessão.  
detach("package:tidyverse", unload = TRUE)  
  
# Funções relacionadas a pacotes.  
apropos("package")
```

# Documentação interna

## A documentação do R

- ▶ Consiste de documentação de objetos e funções.
- ▶ Tutoriais chamados de vinhetas (*vignettes*).
- ▶ Existem funções específicas para a consulta destes.
- ▶ Pode-se procurar na web também.

```
# Duas formas iguais de chamar  
# a documentação.  
?women  
help(women)  
  
# Procura por ocorrências de `women`.  
help.search("women")  
  
# Objetos que batem com o termo.  
apropos("tukey")  
  
# Exibe as vinhetas de um pacote.  
browseVignettes(package = "survival")  
  
# Procura pelo termo no  
# r-project.org.  
RSiteSearch("spider plot")
```

# Campos da documentação

## **Cabeçalho**

Indica o pacote.

## **Título**

Resumo do que são os objetos documentados.

## **Description**

Descrição do que o objeto é/faz.

## **Usage**

Como usar ou formas de montar as instruções.

## **Arguments**

Quais os argumentos formais da função.

## **Value**

O que a função retorna.

## **Details**

Detalhes adicionais de implementação.

## **Note**

Notas adicionais sobre uso e afins.

## **See Also**

Referências para documentação relacionada.

## **References**

Referências bibliográficas.

## **Authors**

Autores da função.

## **Examples**

Exemplos de uso.

# Guia de sobrevivência no R

## 4 principais funções para o iniciante

1. `str()`
2. `ls()`
3. `apropos()`
4. `help()`

## Outras que serão úteis

- |                                   |                           |
|-----------------------------------|---------------------------|
| 1. <code>help.search()</code>     | 1. <code>args()</code>    |
| 2. <code>help.start()</code>      | 2. <code>class()</code>   |
| 3. <code>RSiteSearch()</code>     | 3. <code>methods()</code> |
| 4. <code>browseVignettes()</code> | 4. <code>find()</code>    |
| 5. <code>vignette()</code>        | 5. <code>example()</code> |
|                                   | 6. <code>demo()</code>    |

Kit de autoaprendizado do   
@omegadatascience



Kit de sobrevivência no R. Aprenda a usá-lo e tudo será mais fácil.

# Operações aritméticas e lógicas

# Operações aritméticas

## As operações básicas

```
2 + 3      # Soma.  
2 - 3      # Subtração.  
2 * 3      # Multiplicação.  
2/3        # Divisão.  
2^3         # Potenciação.  
2^(1/3)     # Radiciação.  
10 %% 3    # Resto.  
10 %/% 3   # Parte inteira.
```

## Logarítmo

```
exp(2)          # Exponencial neperiano.  
log(10)        # Neperiano.  
log10(10)      # Base 10.  
log2(10)        # Base 2.  
log(10, base = 5) # Base qualquer.
```

## Trigonométricas

```
sin(3) # Seno.  
cos(3) # Cosseno.  
tan(3) # Tangente.  
  
asin(0.5) # Arco seno.  
acos(0.5) # Arco cosseno.  
atan(0.5) # Arco tangente.
```

## Arredondamento

```
round(pi, digits = 5)  
floor(pi)  # Inteiro logo baixo.  
ceiling(pi) # Inteiro logo acima.  
trunc(pi)  # Trunca em relação ao zero.
```

# Operações lógicas

## Comparações de valor

```
x <- 3
y <- 4

2 == 2 # Igualdade.
2 != 2 # Desigualdade.
x <= y # Outros operadores:
       # "<", ">", and ">=".

(2 < 5) & (7 >= 3) # Operador AND.
(2 < 5) | (7 >= 3) # Operador OR.
!(2 < 5)             # Operador NOT.

"a" == "b" # Compara strings.
"a" < "b" # Ordem alfanumérica.
"1" < "a"
```

## Tipos especiais

- ▶ `NA`: para valores ausentes.
- ▶ `NULL`: para objetos vazios.
- ▶ `Inf` e `-Inf`: para infinitos.
- ▶ `NaN`: para resultados não razoavelmente definidos.

```
5 + NA          # O resultado é NA.
is.na(5 + NA)  # Verifica se é NA.

10 + NULL       # Retorna objeto vazio.
is.null(NULL)  # Verifica se é nulo.

5/0            # Infinito.
is.finite(5/0) # Verifica se é finito.

0/0            # Valor indeterminado.
is.nan(0/0)    # Verifica se é not a number.
```

# Cartões de referência do R

1. R Reference Card
2. R Reference Card · Statistics
3. RStudio IDE

# Estruturas de dados

# Tipos de objetos

- ▶ Praticamente tudo em R são objetos.
- ▶ Objetos têm duas características: tipo (*type*) e classe (*class*).
- ▶ O tipo de um objeto diz o que é possível fazer com ele.
- ▶ A classe diz qual métodos podemos aplicar em um objeto.
- ▶ Precisamos conhecer as principais estruturas e suas características.
- ▶ Principais tipos de objetos em R.
  - ▶ Caracteres (`character`);
  - ▶ Números reais (`double` e `numeric`);
  - ▶ Números inteiros (`integer`);
  - ▶ Lógico (`logical`);
  - ▶ Complexo (`complex`).
  - ▶ Raw (`raw`).

```
inteiro <- 10L
real <- 10
is.numeric(inteiro)
## [1] TRUE

is.numeric(real)
## [1] TRUE

is.double(inteiro)
## [1] FALSE

is.double(real)
## [1] TRUE

is.integer(inteiro)
## [1] TRUE

is.integer(real)
## [1] FALSE
```

# Estruturas de dados

- ▶ Conjuntos de dados são combinações destes tipos de objetos.
- ▶ Como organizar tais dados é o que chamamos de **estruturas de dados**.
- ▶ O R vem equipado com diversas estruturas para trabalhar com dados.
- ▶ Essas estruturas são chamadas de classes `class()`.
- ▶ Estruturas mais importantes em R
  - ▶ `atomic vector` (vetores atômicos).
  - ▶ `matrix` (matrizes).
  - ▶ `list` (listas).
  - ▶ `data.frame` (tabelas).

## ▶ Exemplos

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

matriz <- matrix(c(1,0,0,1),2, 2)

lista <- list("a" = 5, "b" = 5L,
             "c" = "Letra")

tab <- data.frame("Nome" = c("Bent", "Jon"),
                  "Idade" = c(3, 4))
```

# Vetores atômicos

# Vetores atômicos

- Quando combinamos objetos de um único tipo temos os **vetores atômicos**.
- Exemplo

```
vetor_numerico <- c(1, 5, 11, 33)
```

- Como nomear objetos em R?
  - Um nome pode consistir de letras, números, e \_.
  - Não pode conter palavras reservadas: `TRUE`, `FALSE`, `NULL`, `if` entre outras.
  - Veja todas as palavras reservadas usando `? Reserved`.

- Vetor de caracteres

```
vetor_caracter <- c("hello", "world")
```

- Vetor lógico

```
vetor_logico <- c(TRUE, TRUE, FALSE)
```

- Vetor combinado

```
vetor_combinado <- c(vetor_numerico,  
                      vetor_caracter,  
                      vetor_logico,  
                      "boo")
```

- Coerção entre objetos

```
class(vetor_combinado)
```

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

# Como consultar o tipo de um objeto?

- ▶ Para consultar o tipo de um objeto usamos `is.*`.
- ▶ Exemplos

```
# Funções que começam com is.  
#apropos("^is\\\.")  
is.integer(1)  
is.numeric(1)  
is.integer(1L)  
is.numeric(1L)  
is.character("Curitiba")  
y <- factor(c("Solteiro", "Casado"))  
is.factor(y)  
is.character(y)  
is.logical(c(TRUE, FALSE))
```

- ▶ Outra opção é usar a função `typeof()`.

```
x <- "caracter"  
typeof(x)
```

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

```
x <- 10  
typeof(x)
```

```
## [1] "double"
```

```
x <- 10L  
typeof(x)
```

```
## [1] "integer"
```

```
x <- TRUE  
typeof(x)
```

```
## [1] "logical"
```

# Conversão e classe de objetos

- ▶ O R é uma linguagem automaticamente tipificada.
- ▶ Pode ser necessário converter um objeto para diferentes tipos.
- ▶ Exemplo

```
x <- 10
is.numeric(x)

## [1] TRUE

x <- as.logical(x)
is.numeric(x)

## [1] FALSE
```

- ▶ A classe (`class()`) de um objeto orienta que tipo de métodos podemos usar.
- ▶ Métodos são funções genéricas do R que atuam de forma diferente de acordo com o tipo do objeto.

```
# Um vetor numérico.
x <- c(1, 2, 3)
class(x)

## [1] "numeric"

methods(class = "numeric")

## [1] all.equal      as.data.frame as.Date      as.
## [5] as.POSIXlt     as.raster      coerce      Ops
## see '?methods' for accessing help and source code
```

# Objetos e atributos

- É possível criar objetos com atributos.
- Exemplo

```
# Numérico mas com valores nomeados.  
notas <- c("João" = 7.8,  
         "Bianca" = 10,  
         "Eduarda" = 8.5)  
class(notas)  
## [1] "numeric"  
  
attributes(notas)  
## $names  
## [1] "João"     "Bianca"    "Eduarda"  
  
names(notas)  
## [1] "João"     "Bianca"    "Eduarda"
```

- Tamanho do vetor

```
length(notas)
```

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

# Criando sequências estruturadas

- ▶ Função `seq()` cria sequências estruturadas.

```
# Sequencia de 1 a 7  
1:7
```

```
## [1] 1 2 3 4 5 6 7
```

```
# Sequencia de 1 a 10 de 2 em 2  
seq(from = 1, to = 10, by = 2)
```

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

```
# Sequencia de 1 a 20 de tamanho 7  
seq(from = 1, to = 20, length.out = 7)
```

```
## [1] 1.000000 4.166667 7.333333 10.500000 13.666667 16.833333  
## [7] 20.000000
```

```
# Sequencia começando em 1 de tamanho sete de 2 em 2.  
seq(from = 1, by = 2, length.out = 7)
```

```
## [1] 1 3 5 7 9 11 13
```

# Criando repetições estruturadas

- ▶ Função `rep()` cria repetições estruturadas.

```
# Repita o 0 5 vezes  
rep(0, 5)
```

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

```
# Repita a sequencia 1 a 3, 2 vezes  
rep(1:3, times = 2)
```

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

```
# Repita a sequencia 1 a 3, cada número 2 vezes  
rep(1:3, each = 2)
```

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

# Gerando valores aleatórios

- Dado um vetor de elementos podemos usar a função `sample()` para obter uma amostra aleatória.

```
# Retire uma amostra aleatório de tamanho 10 de uma sequencia de 1 a 20 sem reposição.  
sample(1:20, size = 10, replace = FALSE)
```

```
## [1] 5 20 11 8 17 19 10 12 9 6
```

```
# Retire uma amostra de tamanho 10 da sequencia "a", "b", "c" de tamanho 10 com reposição.  
sample(c("a", "b", "c"), size = 10, replace = TRUE)
```

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

- Outra opção é usar distribuições de probabilidade.

```
# 10 amostras da distribuição uniforme entre 0 e 1  
runif(n = 5, min = 0, max = 1)
```

```
## [1] 0.7970946 0.3066262 0.9527793 0.3280883 0.4354850
```

```
# 10 amostras da distribuição Normal com média 1.80 e desvio-padrão 0.1  
rnorm(n = 5, mean = 1.80, sd = 0.1)
```

```
## [1] 1.774603 1.762540 1.642769 1.918804 1.753407
```

# Selecionando elementos de um vetor

- ▶ Podemos selecionar os elementos de um vetor usando: alguma característica, posição ou nome.
- ▶ Exemplo

```
## Criando vetor numérico nomeado.  
notas <- c("João" = 7.8,  
         "Bianca" = 10,  
         "Eduarda" = 8.5,  
         "Felipe" = 7.0,  
         "Márcia" = 6.5)
```

```
notas[1]      # A posição 1.
```

```
## João  
## 7.8
```

```
notas[5]      # A posição 5.
```

```
## Márcia  
## 6.5
```

```
notas[1:2]    # Um intervalo.
```

```
## João Bianca  
## 7.8 10.0
```

```
notas[c(1, 3)] # Um conjunto.
```

```
## João Eduarda  
## 7.8 8.5
```

```
notas[-1]     # Remove.
```

```
## Bianca Eduarda Felipe Márcia  
## 10.0 8.5 7.0 6.5
```

# Seleção com máscara lógica

- ▶ Seleção com máscara lógica

```
# Alunos com nota maior que 7.0
```

```
mask <- notas > 7.0
```

```
mask
```

```
## João Bianca Eduarda Felipe Márcia
## TRUE TRUE TRUE FALSE FALSE
```

```
notas[mask]
```

```
## João Bianca Eduarda
## 7.8 10.0 8.5
```

```
# Alunos com nota maior que 9.0
```

```
notas[notas > 9.0]
```

```
## Bianca
## 10
```

# Modificando vetores

```
# Atribui nota para um aluno.
```

```
notas["João"] <- 0
```

```
notas
```

```
## João Bianca Eduarda Felipe Márcia
```

```
## 0.0 10.0 8.5 7.0 6.5
```

```
# Atribui nota "desconhecida" para aluno.
```

```
notas["Felipe"] <- NA
```

```
notas
```

```
## João Bianca Eduarda Felipe Márcia
```

```
## 0.0 10.0 8.5 NA 6.5
```

# Removendo e adicionando componentes de um vetor

## ► Removendo componentes

```
# Remove elemento do vetor.  
notas <- notas[-4]  
notas  
  
## João Bianca Eduarda Márcia  
## 0.0 10.0 8.5 6.5
```

## ► Adicionando componentes

```
append(notas, value = c("Carlos" = 9.0))  
## João Bianca Eduarda Márcia Carlos  
## 0.0 10.0 8.5 6.5 9.0  
  
append(notas, value = c("Simone" = 7.2),  
      after = 0)  
## Simone João Bianca Eduarda Márcia  
## 7.2 0.0 10.0 8.5 6.5  
  
novas_notas <- c(notas,  
                  c("Pedro" = 8.0,  
                    "Luana" = 8.3))  
novas_notas  
## João Bianca Eduarda Márcia Pedro Luana  
## 0.0 10.0 8.5 6.5 8.0 8.3
```

# Matrizes, listas e tabelas

# Matrizes

- Matrizes são um simples extensão dos vetores.
- São vetores com dimensão (número de linhas e colunas).
- Todos os elementos de uma matriz devem ser do mesmo tipo.
- Definindo uma matriz

```
matriz <- matrix(data = c(1, 0, 0, 1),  
                  nrow = 2, ncol = 2)  
matriz
```

```
## [,1] [,2]  
## [1,] 1 0  
## [2,] 0 1
```

- Acessando a dimensão de uma matriz

```
dim(matriz)  
## [1] 2 2
```

- Matrizes são por *default* preenchidas por colunas.

```
matriz <- matrix(c(1,2,3,4,5,6,7,8),  
                  nrow = 4, ncol = 2)  
matriz
```

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

- Podemos preencher por linhas.

```
matriz <- matrix(c(1,2,3,4,5,6,7,8),  
                  nrow = 4, ncol = 2,  
                  byrow = TRUE)  
matriz
```

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

# Acessando elementos de uma matriz

- Acesso por posição (linha e coluna).

```
# Linha 1 Coluna 2  
matrix[1,2]
```

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

```
# Linha 3 Coluna 1  
matrix[3,1]
```

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

```
# Linhas 1 e 2 Coluna 2  
matrix[1:2, 2]
```

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

```
# Linhas 2 e 3 e Colunas 1 e 2  
matrix[2:3, 1:2]
```

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

- Seleção condicional por máscara lógica.

```
## Elementos maiores do que 4  
matrix[matrix > 4]
```

```
## [1] 5 7 6 8
```

```
## Máscara lógica  
matrix > 4
```

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

# Listas

- A estrutura `list` é a mais flexível em R.
- Pode armazenar diferentes tipos de objetos.

```
minha_lista <- list(10, "Dez", TRUE, 1+10i)
minha_lista

## [[1]]
## [1] 10
##
## [[2]]
## [1] "Dez"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+10i
```

- Acessando elementos de uma lista.

```
# Primeiro elemento
minha_lista[[1]]
## [1] 10

# Quarto elemento
minha_lista[[4]]
## [1] 1+10i

# Primeiro e quarto elemento
minha_lista[c(1,4)]
## [[1]]
## [1] 10
##
## [[2]]
## [1] 1+10i
```

# Listas nomeadas

- ▶ Os elementos da lista podem ser nomeados.

```
minha_lista <- list("Números" = c(10, 100),
                     "Caracter" = c("Dez", "Cem",
                     "Logico" = TRUE,
                     "Complexo" = 1+10i)

minha_lista

## $Números
## [1] 10 100
## $Caracter
## [1] "Dez" "Cem"
##
## $Logico
## [1] TRUE
##
## $Complexo
## [1] 1+10i
```

- ▶ Acesso pelo nome.

```
minha_lista$Logico
```

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

```
minha_lista$Números
```

```
## [1] 10 100
```

- ▶ Acessar os atributos de uma lista.

```
attributes(minha_lista)
```

```
## $names
## [1] "Números"  "Caracter"  "Logico"    "Complexo"
```

# Tabelas

- ▶ Classe mais utilizada para representar conjuntos de dados.
- ▶ Similar a uma planilha eletrônica.
- ▶ Um `data.frame` tem diversos atributos: `rownames()`, `colnames()`, `names()` e `dim()`.
- ▶ É aconselhável usar a convenção: cada linha representa uma observação (registro) e cada coluna uma variável.
- ▶ O R permite ler conjuntos de dados de diversos formatos.
- ▶ O R já vem equipado com conjuntos de dados para fins didáticos.
- ▶ Conjunto de dados `iris`.

```
str(iris)
```

```
## 'data.frame': 150 obs. of 5 variables:
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

# Classe factor

- ▶ A classe `factor` é similar a `character` no entanto permite definir ordem.
- ▶ Por exemplo,

```
fator <- factor(c("alta", "baixa", "baixa", "media",
                  "alta", "media", "baixa", "media", "media"),
                  levels = c("baixa", "media", "alta"),
                  ordered = TRUE)
```

- ▶ Se fizermos uma tabela, por exemplo, a ordem será respeitada

```
table(fator)

## # fator
## baixa media alta
##      3     4     2
```

# Criando e inspecionando um `data.frame`

- ▶ Podemos criar um `data.frame` diretamente em R.

```
dados <- data.frame(Letras = letters[1:6],  
                     Numeros = 1:6,  
                     Logico = rep(c(TRUE, FALSE),  
                                  each = 3))  
  
dados
```

	Letras	Numeros	Logico
## 1	a	1	TRUE
## 2	b	2	TRUE
## 3	c	3	TRUE
## 4	d	4	FALSE
## 5	e	5	FALSE
## 6	f	6	FALSE

- ▶ Diversas opções de inspeção.
- ▶ `head()` - mostra as seis primeiras linhas.
- ▶ `tail()` - mostra as seis últimas linhas.
- ▶ `dim()` - número de linhas e de colunas.
- ▶ `nrow()` - número de linhas.
- ▶ `ncol()` - número de colunas.
- ▶ `str()` - estrutura do `data.frame`.
- ▶ `names()`, `colnames()` e `rownames()` nome das linhas e colunas.

# Acessando elementos de um `data.frame`

- ▶ O acesso é parecido com objetos da classe `matrix`.

```
# Primeira linha todas as colunas  
iris[1,]
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1          5.1         3.5         1.4         0.2  setosa
```

```
# Segunda linha colunas de 1 a 3  
iris[2,1:3]
```

```
## Sepal.Length Sepal.Width Petal.Length  
## 2          4.9         3           1.4
```

```
# Linhas 1 e 5 todas as colunas  
iris[c(1, 5),]
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1          5.1         3.5         1.4         0.2  setosa  
## 5          5.0         3.6         1.4         0.2  setosa
```

- ▶ Acesso pelo nome da coluna.

```
iris$Sepal.Length
```

# Resumo

- ▶ Os principais tipos de dados são: `character`, `numeric`, `integer`, `logical`, `complex` e `raw`.
- ▶ As principais estruturas de dados são: `vector`, `matrix`, `list` e `data.frame`.
- ▶ Lembre-se:
  - ▶ Vetores: conjunto de elementos unidimensional, todos do mesmo tipo.
  - ▶ Matrizes: conjunto de elementos bidimensional (linha e coluna), todos do mesmo tipo.
  - ▶ Listas: caso especial de vetor em que cada elemento pode ser uma estrutura diferente.
  - ▶ Data frames: tabela. Cada coluna é um vetor, portanto os elementos dentro da coluna são de mesmo tipo mas os tipos entre colunas podem ser diferentes.

# Estruturas de programação

# Tipos de estruturas de programação

Existem 2 tipos básicos de estruturas de programação que estão presentes nas linguagens de programação.

## Estruturas de controle

- ▶ Elas permitem fazer **execução condicional ou seletiva** de instruções conforme o resultado de condições que são testadas.
- ▶ Os tipos mais comuns são:
  - ▶ IF-ELSE.
  - ▶ SWITCH ou CASES.

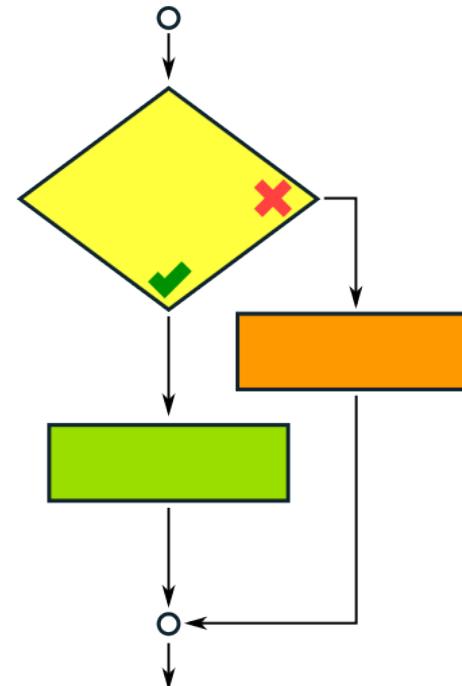
## Estruturas de repetição

- ▶ Também chamados de loops ou estruturas de iteração que permitem executar instruções um certo número de vezes até que uma condição seja atingida.
- ▶ Os tipos mais comuns são:
  - ▶ FOR ou FOREACH.
  - ▶ WHILE.
  - ▶ REPEAT ou DO-UNTIL.

# Estruturas de controle

# Estrutura IF-ELSE

- ▶ O IF-ELSE permite desviar a execução do código conforme uma condição cujo resultado é lógico, ou seja, TRUE ou FALSE.
- ▶ Se o resultado da condição for TRUE, executase certas instruções.
- ▶ Se o resultado da condição for FALSE, executase outras instruções.



Fluxograma de uma estrutura de controle tipo IF-ELSE.  
Fonte: os autores.

# Exemplo de estrutura IF-ELSE no R

```
if (.condicao1.) {  
  .procedimento1.  
} else if (.condicao2.) {  
  .procedimento2.  
} else {  
  .procedimento_final.  
}
```

Condição que retorna valor lógico: TRUE/FALSE.

Código executado se a condição for TRUE.

Condição avaliada quando a anterior for FALSE

Código executado quando a condição for TRUE.

Condição avaliada quando as anteriores forem FALSE.

```
faltas <- 10  
nota <- 70  
if (nota >= 70 & faltas < 15) {  
  result <- "Aprovado"  
} else if (nota < 70) {  
  result <- "Reprovado por nota"  
} else {  
  result <- "Reprovado por faltas"  
}  
result
```

Exemplo de código com a estrutura de controle tipo IF-ELSE. Fonte: os autores.

# Exercício · Saudação conforme o horário em automação de email marketing

Com base na hora do envio de uma mensagem, use bom dia, boa tarde ou boa noite.

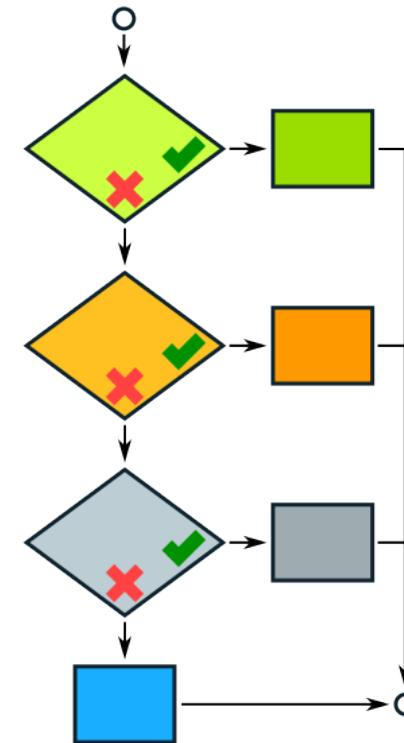
- ▶ Se entre 6h00 e 11h59, então "Bom dia!"
- ▶ Se entre 12h00 e 17h59, então "Boa tarde!"
- ▶ Se entre 18h00 e 22h59, então "Boa noite!"
- ▶ Caso nenhum dos acima, não enviar a mensagem.



Fonte: <https://pegas.io/marketing/marketing-automation/email-marketing/>.

# Estrutura SWITCH

- ▶ O SWITCH nada mais é que um empilhamento de estruturas IF-ELSE.
- ▶ No entanto, ele confere mais clareza para o código e eficiência<sup>1</sup>.
- ▶ O caso mais comum é usar SWITCH para testar um valor contra um conjunto de strings.
- ▶ Geralmente há uma condição final ou de escape.



Fluxograma de uma estrutura de controle tipo SWITCH.  
Fonte: os autores.

# Exemplo de estrutura SWITCH no R

```
switch(string,
  string1 = {
    procedimento1
  },
  string2 = {
    procedimento2
  },
  {
    procedimento_final
  })
  
```

Variável do tipo string.

Código executado se `string == string1`.

Código executado se `string == string2`.

Código executado quando nenhuma das condições anteriores for TRUE.

Sem valor para comparação.

```
animal <- "gato"
som <-
  switch(animal,
    "cachorro" = {
      latir()
    },
    "gato" = {
      miar()
    },
    "vaca" = {
      mugir()
    },
    {
      silencio()
    })
  som
```

Exemplo de código com a estrutura de controle tipo SWITCH. Fonte: os autores.

## Exemplo · Tipos de média

Use `switch()` para aplicar a

- ▶ média aritmética,

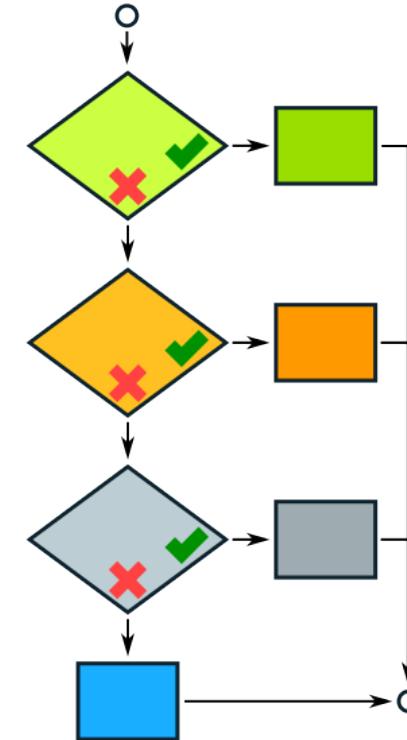
$$m_a = \sum_i \frac{y_i}{n}$$

- ▶ harmônica e

$$m_h = \frac{n}{\sum_i \frac{1}{y_i}}$$

- ▶ geométrica.

$$m_g = \left( \prod_i y_i \right)^{1/n}$$



Fluxograma de uma estrutura de controle tipo SWITCH.  
Fonte: os autores.

# Versões vetoriais

- ▶ A estrutura IF-ELSE apresentada **não é vetorial**, ou seja, no teste da condição são comparados dois valores.
- ▶ Quando a condição tiver que ser testada sobre uma série de valores em um vetor, pode-se usar a função `base::ifelse()` ou `dplyr::if_else()`.
- ▶ A versão vetorial para a estrutura SWITCH é o `dplyr::case_when()`.

*# Notas dos alunos.*

```
notas <- c("João" = 70, "Ana" = 89,  
         "Márcia" = 81, "Tiago" = 65,  
         "Rodrigo" = 35)
```

*# Usando IF-ELSE vetorial.*

```
ifelse(notas >= 70, "Aprovado",  
       ifelse(notas >= 40, "Exame",  
              "Reprovado"))
```

```
##           João          Ana  
## "Aprovado"  "Aprovado"  
##       Márcia        Tiago  
## "Aprovado"      "Exame"  
##       Rodrigo  
## "Reprovado"
```

*# Usando SWITCH vetorial.*

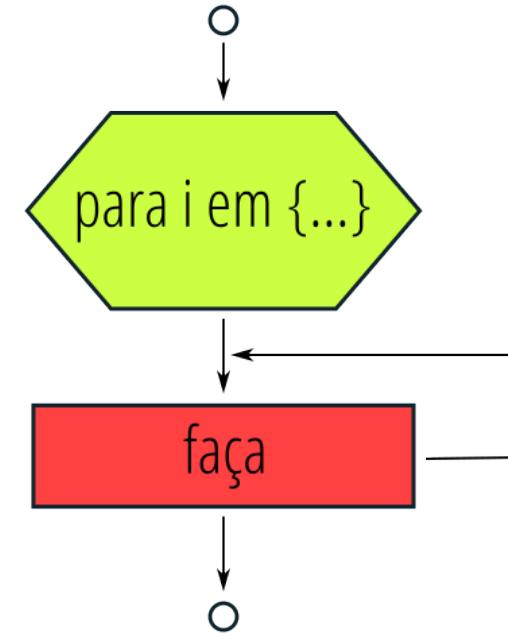
```
dplyr::case_when(notas >= 70 ~ "Aprovado",  
                  notas >= 40 ~ "Exame",  
                  TRUE ~ "Reprovado")
```

```
## [1] "Aprovado"  "Aprovado"  
## [3] "Aprovado"  "Exame"  
## [5] "Reprovado"
```

# Estruturas de repetição

# Estrutura de loop FOR

- ▶ O loop FOR é usado pra executar instruções um número previamente **conhecido** de vezes.
- ▶ O loop itera sobre uma sequência de valores, um vetor ou lista.
- ▶ Pode-se **interromper** a execução do loop ou **saltar** conforme condições que são avaliadas.
- ▶ Deve-se usar o FOR para situações em que deseja fazer a execução percorrendo uma série conhecida de elementos ou número de iterações conhecido.



Fluxograma de uma estrutura de repetição tipo FOR.  
Fonte: os autores.

# Exemplo de estrutura FOR no R

```
    Variável indexadora.  
    |  
for (.variavel. in .vetor.) {  
    procedimento.          | Vetor ou lista a ser percorrido.  
    if (.condicao_de_salto.) next  
    procedimento.  
    if (.condicao_de_saida.) break  
    procedimento.  
}  
    | Código executado  
    | dentro da estrutura  
    | de repetição.
```

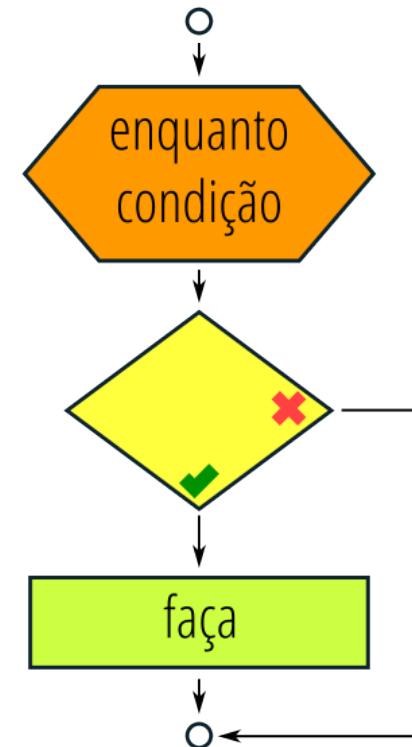
```
tx_juros <- 0.01  
n_meses <- 12           | Importante!  
rend <- numeric(n_meses) | Alocar o vetor do  
                           | tamanho correto.  
rend[1] <- 100  
for (i in 2:n_meses) {  
    rend[i] <- rend[i - 1] * (1 + tx_juros)  
}  
rend
```

```
y <- c(1, 2, 1, 4, 0, NA, 3, 2, 4)  
s <- 0                   | Encerra o loop se a soma  
                           | passar de 10.  
for (i in y) {  
    if (s > 10) break  
    if (is.na(i)) next  
    s <- s + i            | Passa para o próximo valor  
                           | quando encontrar um NA.  
}  
s
```

Exemplo de código com a estrutura de repetição tipo FOR. Fonte: os autores.

# Estrutura WHILE

- ▶ O loop WHILE é usado quando precisa executar instruções um número **desconhecido** de vezes até que uma condição seja atendida.
- ▶ A condição é testada **antes** das instruções serem avaliadas.
- ▶ **Cuidado:** uma condição de parada deve ser declarada para que o loop não seja interminável.



Fluxograma de uma estrutura de repetição tipo WHILE.  
Fonte: os autores.

# Exemplo de estrutura WHILE no R

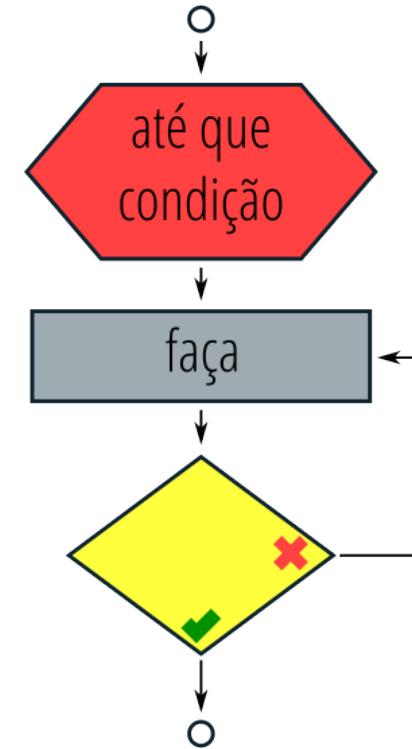
```
while (.condicao.) {  
  .procedimento.  
  if (.condicao_salto.) next  
  .procedimento.  
  if (.condicao_encerra.) break  
  .procedimento.  
}  
  .procedimento.  
  Condicao que mantem a execucao.  
  Condicao que move para o proximo passo.  
  Condicao que interrompe a execucao e sai.  
  Código executado dentro da estrutura de repetição.
```

```
n_numbers <- 12  
total <- 0  
i <- 1L  
while (i < n_numbers) {  
  u <- total + runif(1)  
  if (sum(u) > 4) break  
  total <- u  
  i <- i + 1L  
}  
total
```

Exemplo de código com a estrutura de repetição tipo WHILE. Fonte: os autores.

# Estrutura REPEAT

- ▶ O loop REPEAT é usado quando precisa executar instruções um número **desconhecido** de vezes até que uma condição seja atendida.
- ▶ A condição é testada **em qualquer posição** dentro das instruções a serem executadas.
- ▶ **Cuidado:** uma condição de parada deve ser declarada para que o loop não seja interminável.



Fluxograma de uma estrutura de repetição tipo REPEAT.  
Fonte: os autores.

# Exemplo de estrutura REPEAT no R

```
repeat {  
  ·procedimento·  
  if (.condicao_salto.) next  
  ·procedimento·  
  if (.condicao_encerra.) break  
  ·procedimento·  
}  
  └─ Código executado  
      dentro da estrutura  
      de repetição.
```

Condição que move para o próximo passo.

Condição que interrompe a execução e sai.

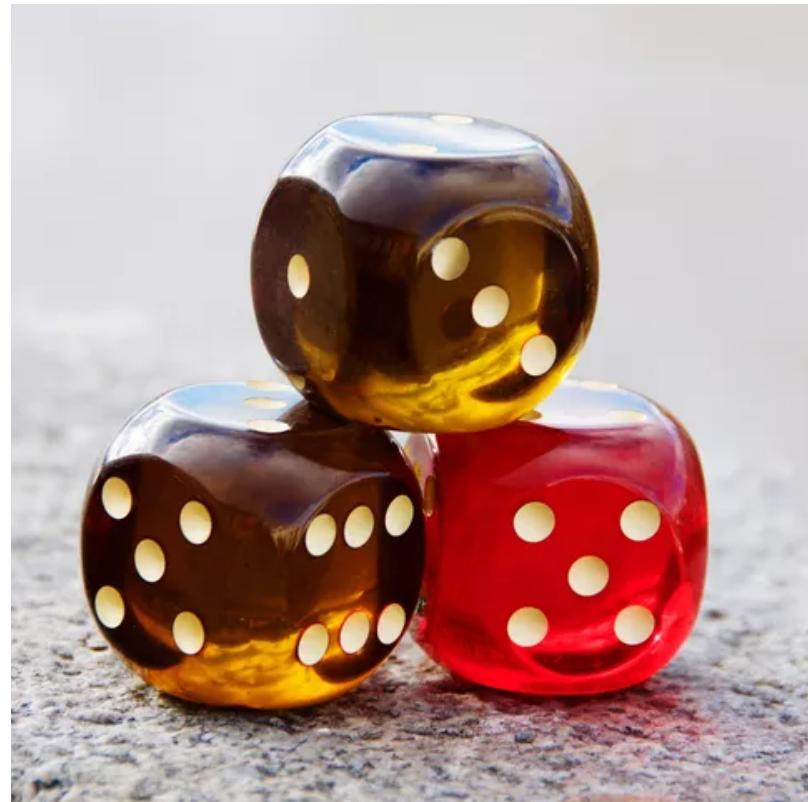
```
total <- 0  
i <- 1L  
repeat {  
  u <- total + runif(1)  
  if (sum(u) > 4) break  
  total <- u  
  i <- i + 1L  
}  
total
```

Exemplo de código com a estrutura de repetição tipo REPEAT. Fonte: os autores.

# Exercício • Lançamento de dados

Considere o seguinte jogo: lançar 3 dados até que os valores das faces sejam uma sequência, por exemplo, 3, 4 e 5 ou 1, 2 e 3.

1. Use um loop WHILE ou o REPEAT para contar quantas tentativas são necessárias até atingir o resultado.
2. Dicas:
  - ▶ use `sample()` para sortear 3 valores do espaço amostral.
  - ▶ em seguida ordene os valores do menor para o maior com `sort()`.
  - ▶ calcule a diferença entre valores consecutivos com `diff()`.



Três dados. Fonte:  
<https://www.thoughtco.com/probabilities-for-rolling-three-dice-3126558>

# E se quisermos saber

Em média, quantos lançamentos são necessários?

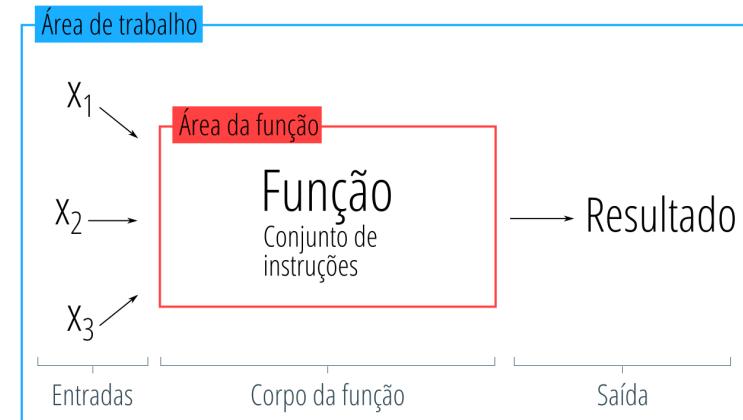
## Refaça n-vezes

- ▶ Para isso, pode-se usar o loop FOR com um loop WHILE dentro.
- ▶ Cada ciclo do FOR determina o número de tentativas usadas no WHILE.
- ▶ Depois é só calcular a média.
- ▶ No entanto, para situações como essa de execuções independentes das mesmas instruções, pode-se usar a `replicate()`.

# Funções

# O que são funções?

- ▶ Funções **encapsulam uma tarefa** composta de várias instruções.
- ▶ Funções pegam valores de **entrada** e geram valores de **saída** conforme as instruções nela implementadas.
- ▶ Funções permitem partitionar um grande número de instruções em funções que atuam como **partes de um algoritmo**.
- ▶ Funções permitem **reuso** de código de uma forma enxuta.
- ▶ Permitem **preservar** o estado das variáveis na área de trabalho.



Fluxograma de uma estrutura de repetição tipo FOR.  
Fonte: os autores.

# Anatomia de uma função no R

```
Nome da função           Lista de parâmetros
  [imc <- function(peso, altura) {
    Corpo da função
      [imc <- peso/altura^2
       limits <- c(0, 18.5, 25, 30, Inf)
       labels <- c("Magreza", "Adequado",
                  "Pré-obeso", "Obesidade")
       classif <- labels[findInterval(imc, vec = limits)]
       return(list(IMC = imc, Classificacao = classif))
    }
  ]
  Retorno da função
```

Anatomia de uma função no R. Fonte: os autores.

- ▶ **Nome da função:** usado para chamá-la.
- ▶ **Lista de parâmetros:** também chamado de lista de **argumentos formais** que passa **argumentos atuais** ou valores como variáveis de entrada.
- ▶ **Corpo da função:** contém a série de instruções que são realizadas.
- ▶ **Retorno da função:** retorna o que a função determinou com as instruções nela implementadas.

# Cuidados para implementar funções

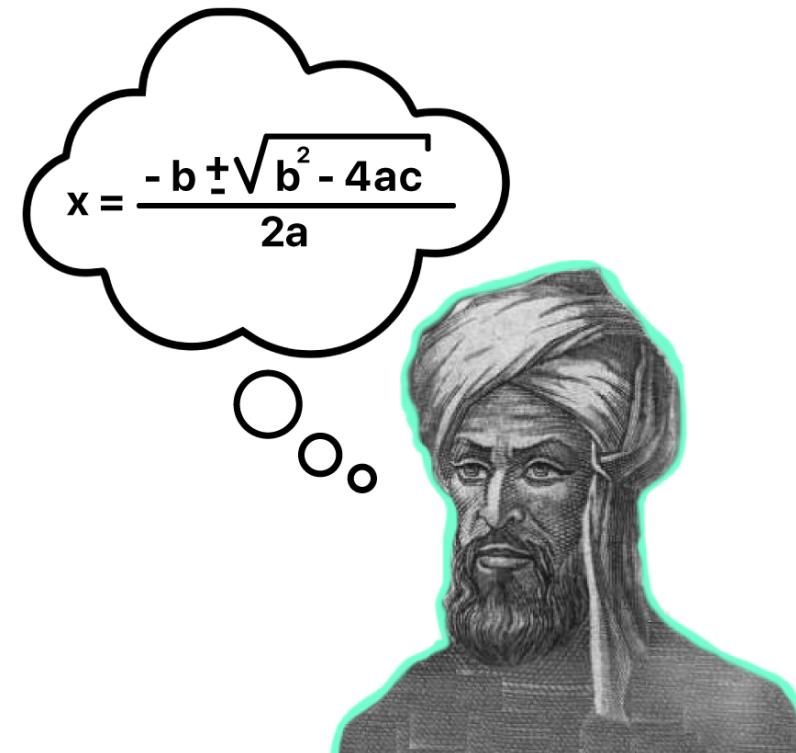
- ▶ **Conheça o procedimento** que deseja encapsular numa função.
- ▶ Escolha um **nome apropriado** para a função, preferencialmente, um verbo ou composto por verbo.
- ▶ Use nomes apropriados para os argumentos da função.
  - ▶ Argumentos podem ser vetores, matrizes, listas, etc, e até outras funções.
  - ▶ Argumentos podem ter valores default.
- ▶ Coloque o resultado da função como valor de retorno.
- ▶ Se for necessário retornar mais de um objeto, use uma lista.
- ▶ Ao programar, faça os devidos **tratamentos de exceção**.
- ▶ Coloque **mensagem descritivas** para erros, avisos, comentários, etc.
- ▶ Modifique variáveis **fora do escopo** da função apenas quando não houver outra solução.

## Exercício · Fórmula de Báskara

Faça uma função que retorne as raízes de uma parábola com equação

$$ax^2 + bx + c,$$

usando a fórmula de Báskara.



A fórmula de Báskara. Fonte:  
<https://blog.professorferretto.com.br/formula-de-bhaskara/>

# Tratamento de exceções

- ▶ Quando se faz funções para usos gerais e uma maior audiência, é importante fazer os **tratamentos de exceção** e fazer a função comunicar de forma clara com o usuário.
- ▶ Tratar exceções é prever desvios de uso ou de processamento causados, por exemplo, por:
  - ▶ Variáveis com tipos incorretos.
  - ▶ Objetos de classe incorreta.
  - ▶ Indeterminação nos resultados.
  - ▶ Falha de convergência.
  - ▶ Conexão/rede não disponível.
  - ▶ Arquivos/objetos não encontrados.
  - ▶ Etc.
- ▶ `cat()` e `print()`:  
Usa-se para imprimir conteúdo no console.
- ▶ `message()`:  
Usa-se para imprimir mensagens **neutras** para o usuário.
- ▶ `warning()`:  
Usa-se para imprimir mensagens de **aviso**. Ela causa a saída `Warning`.
- ▶ `stop()`:  
Usa-se para imprimir mensagens de **erro**. Ela causa a saída `Error` e interrompe a execução do código.

# Exercício • Lançamento de dados

Reconsidere o seguinte jogo: lançar 3 dados até que os valores das faces sejam uma sequência, por exemplo, 3, 4 e 5 ou 1, 2 e 3.

1. Use um loop WHILE ou o REPEAT para contar quantas tentativas são necessárias até atingir o resultado.
2. Dicas:
  - ▶ use `sample()` para sortear 3 os valores do espaço amostral.
  - ▶ em seguida ordene os valores do menor para o maior com `sort()`.
  - ▶ calcule a diferença entre valores consecutivos com `diff()`.
3. Encapsule tudo dentro de uma função de forma a generalizar o seu jogo.

# Considerações finais

## Funções

- ▶ Criar funções para **encapsular tarefas repetidas** compostas de várias instruções.
- ▶ Possui nome, argumentos, instruções e retorno.
- ▶ Usar nomes apropriados para a função e argumentos.
- ▶ Fazer tratamentos de exceção.

## Aspectos avançados

- ▶ Argumentos com valores default.
- ▶ Tratando exceções.
- ▶ Funções sem argumentos.
- ▶ Lazy evaluation.
- ▶ Uso dos ....