Introdução ao Software R

Universidade Estadual de Santa Cruz

Ivan Bezerra Allaman

06/12/2021

CRONOGRAMA

- Primeiro dia
  - Um breve histórico
  - Pontos fortes
  - Pontos fracos
  - Porque alguns acham o R um bicho papão
  - Entendendo como o R funciona
  - Instalando o R no Linux (Distribuição Debian 10)
  - Instalando o R no Windows
  - Instalando o RStudio no Linux (Distribuição Debian 10)
  - Instalando o Tinn-R no Windows
  - Mão na massa - Quebrando o gelo com o R
- Segundo dia
  - Apresentação de algumas potencialidades do R
  - Operadores de atribuição, matemáticos e lógicos
  - Funções elementares
  - Instalar e carregar pacotes, documentação e ajuda

- Terceiro dia
  - Vetores
  - Matrizes
  - Arrays
- Quarto dia
  - Data frames
  - Listas
  - Tabelas
  - Como criar funções
  - Condicionais e loop
- Quinto dia
  - Gráficos
  - Importação de dados e exportação de objetos
  - BÔNUS!

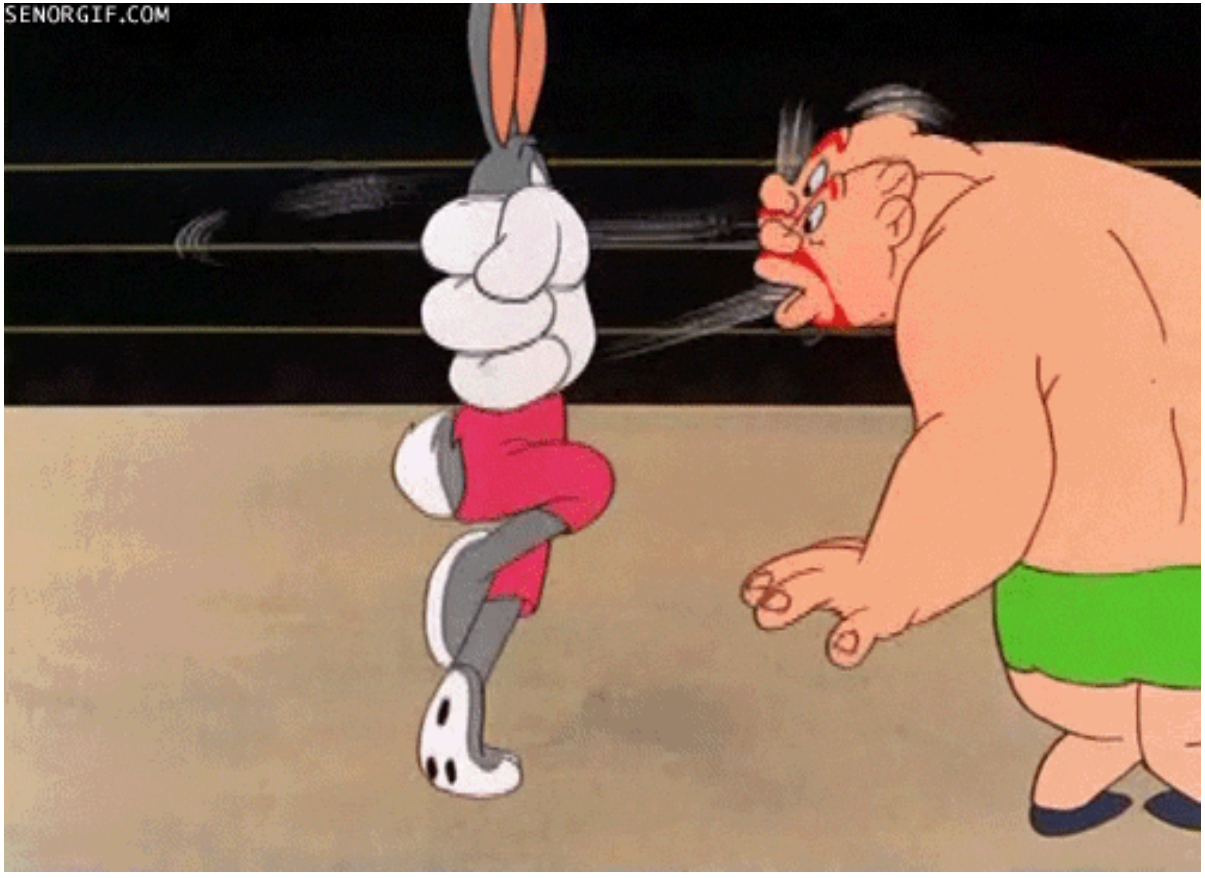

YO IT'S 1ST DAY OF
SCHOOL!

Instruções para um bom andamento do curso

- Não seja ancioso! Aperte um botão ou execute um comando apenas quando o instrutor falar.

- Não copie o código que será fornecido junto ao material! Tente você mesmo digitar as linhas de comando e prover comentários com suas próprias palavras.

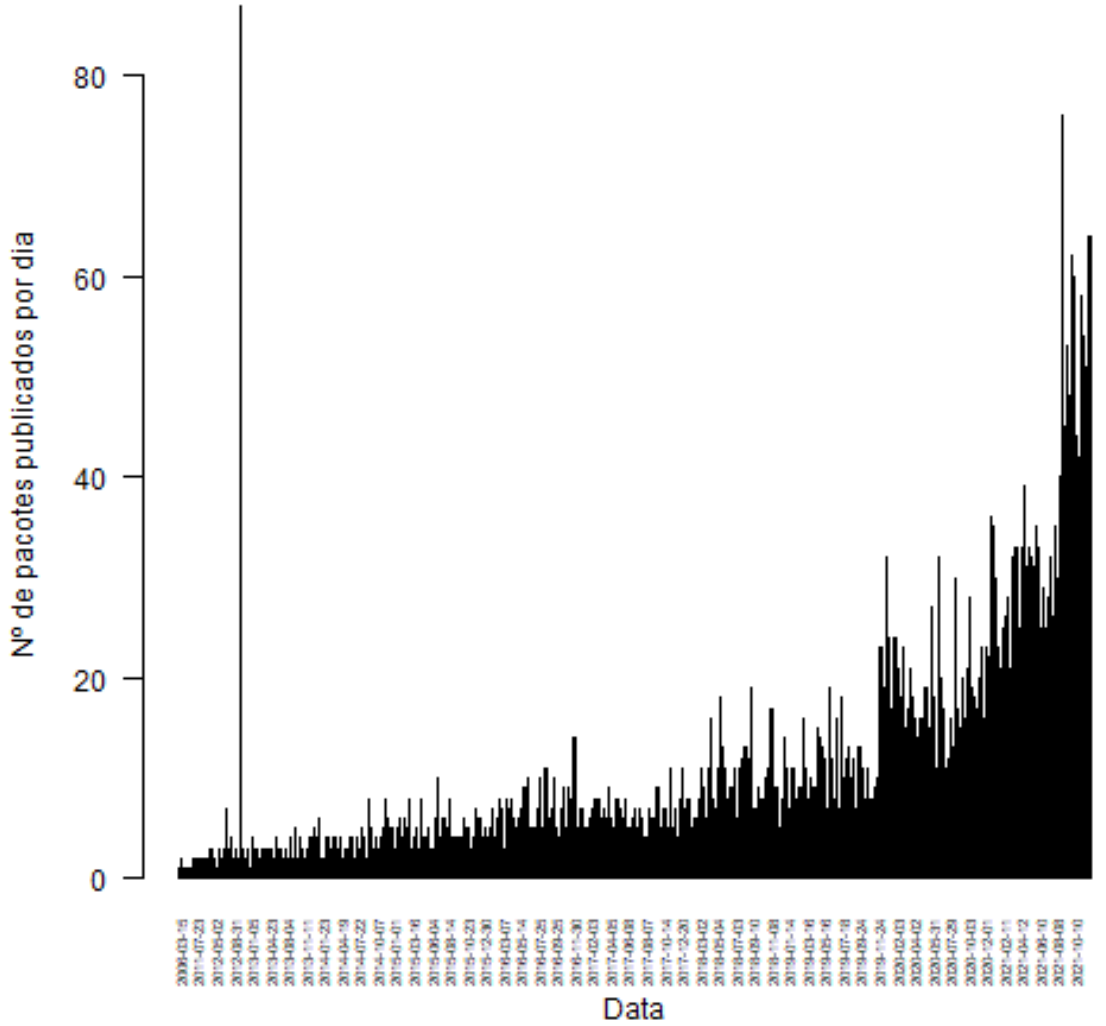
- Seja resiliente!

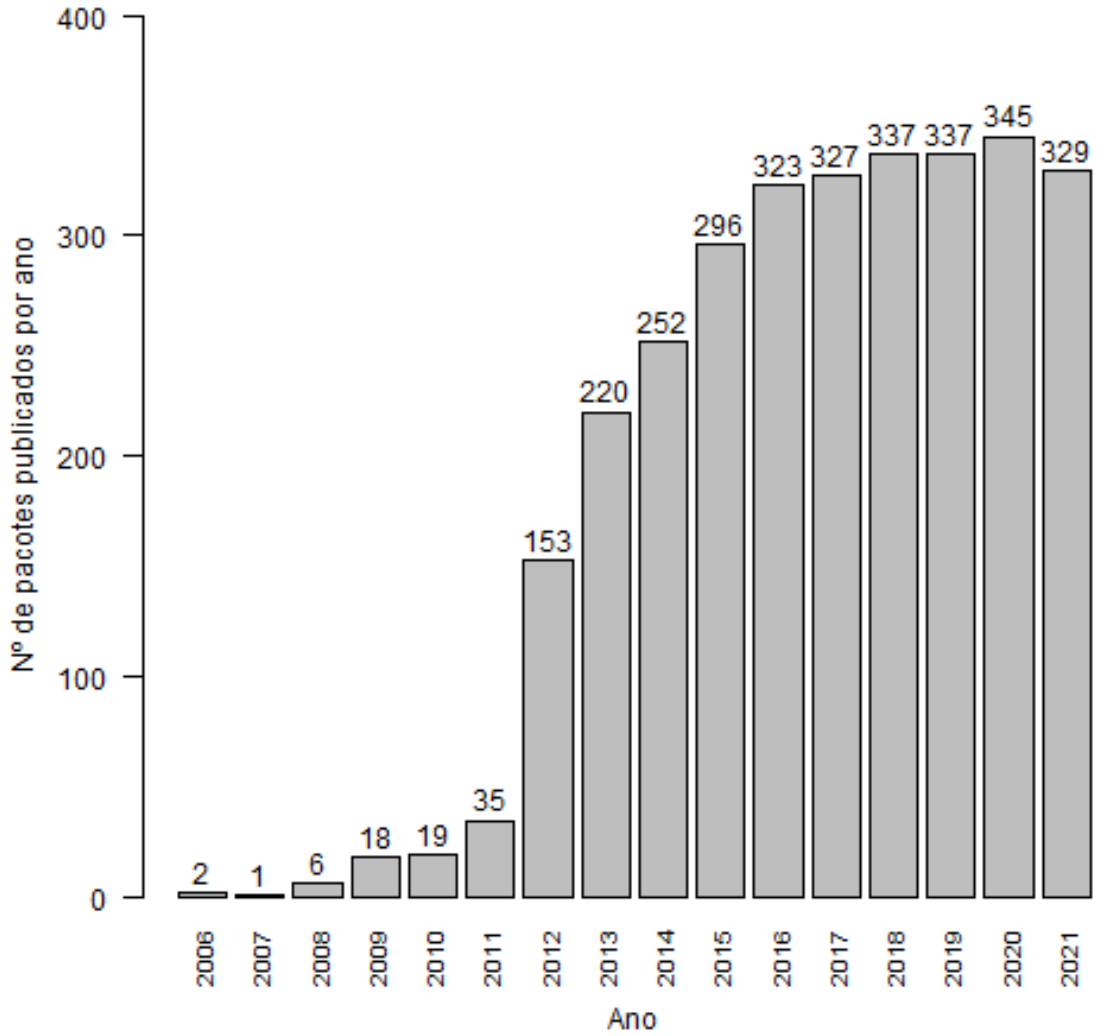


- Continue praticando sempre!

Um breve histórico


R – 1.0.0
2000/02/29
BD Ripley


Nº de pacotes publicados por dia
0
20
40
60
80
Data


Nº de pacotes publicados por ano
0
100
200
300
400
2
1
6
18
19
35
153
220
252
296
323
327
337
337
345
329
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
Ano

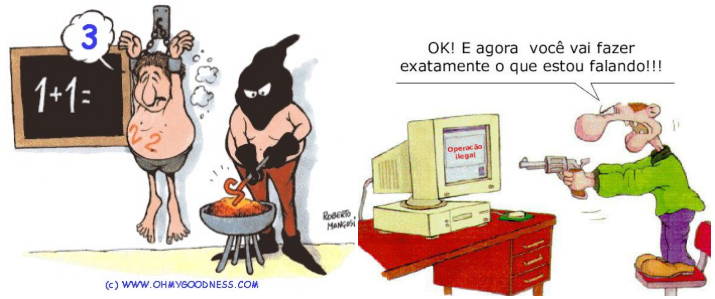
Pontos fortes

1. Código Aberto
2. Suporte exemplar para transformação de dados
3. Variedade de pacotes
4. Altamente compatível
5. Plataforma independente
6. Relatórios automatizados e atraentes, sejam eles estáticos ou dinâmicos
7. Estatística

Pontos fracos

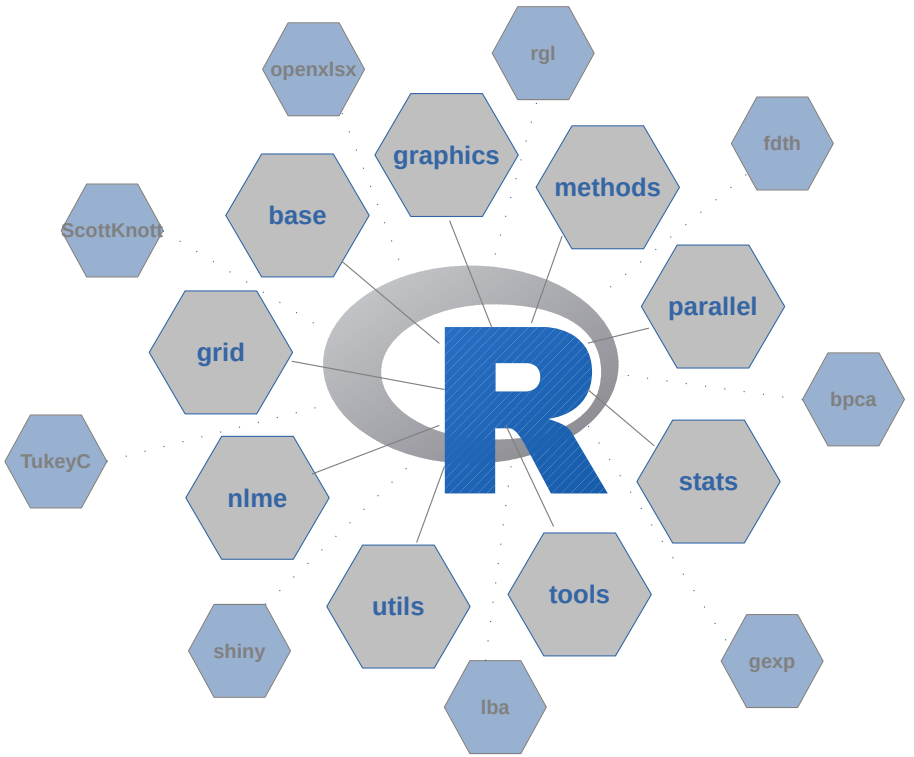
1. Tratamento de dados
2. Linguagem complicada

Porque alguns acham o R um bicho papão

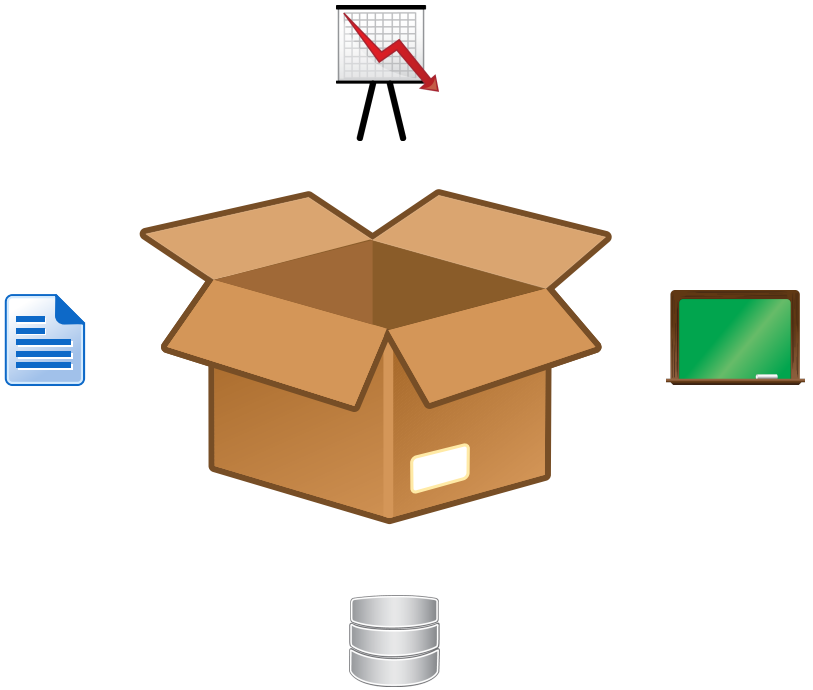


Entendendo como o R funciona

- O R funciona basicamente por pacotes



- E o que é um pacote?
  - é um conjunto de códigos, dados e documentação.



Instalando o R no Windows 10

Instalando o Tinn-R no Windows 10

Instalando o R no Linux (Distribuição Debian 10)

Quebrando o gelo


DAY TWO

Apresentação de algumas potencialidades do R

```
# Apresentação de algumas funcionalidades do R

```{r}
# ------------------------------------------------------------------------------
# Alguns exemplos dos recursos gráficos básicos
# ------------------------------------------------------------------------------
demo(graphics)   # Recursos gráficos genéricos

demo(persp)      # Recursos gráficos 3D

# ------------------------------------------------------------------------------
# Alguns exemplos dos recursos gráficos mais avançados
# ------------------------------------------------------------------------------
demo(plotmath)   # Recursos para escrever equações,
                 # fórmulas e símbolos dentro de gráficos

library(lattice)

demo(lattice)

# rgl (3d dinâmico)
library(rgl)    # Atenção!!!! não vem instalado!!!
demo(abundance) # Recursos gráficos 3D dinâmico (interagir com o mouse)
demo(bivar)     # Recursos gráficos 3D dinâmico (interagir com o mouse)

# ------------------------------------------------------------------------------
# Recursos gráficos interativos para ensino
# ------------------------------------------------------------------------------
library(rpanel) # Atenção!!!! não vem instalado!!!
demo(rp.gulls)

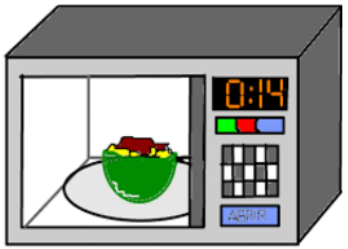
library(asbio) # Atenção!!!! não vem instalado!!!!
anm.ci.tck()    # Intervalo de confiança
```
```

Operadores de atribuição, matemáticos e lógicos

Operadores de atribuição

- `<-`
- `->`
- `=`

Entendendo que todo objeto pertence a uma classe



Eletrodoméstico

Alimentos

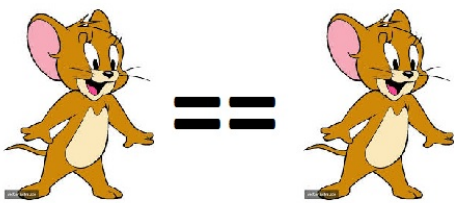
Limpeza

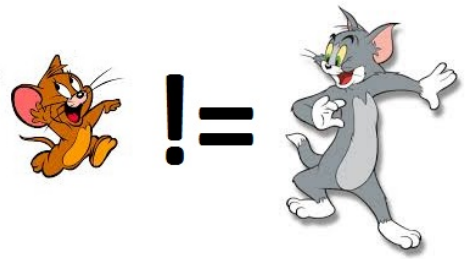
Operadores matemáticos

- `+` soma
- `-` subtração
- `*` multiplicação
- `/` divisão
- `^` potenciação

Operadores relacionais

- `>` maior que
- `<` menor que
- == igual a
- != diferente de
- `>=` maior igual
- <= menor igual
- & e
- | ou





Exercícios

1. Para cada mês do ano de 2017, crie um objeto contendo o valor gasto de energia em sua casa.
2. Some os valores e divida pela quantidade de objetos criados.
3. Some novamente os objetos criados armazenando-os em um novo objeto (passo 1). Crie um outro objeto de modo que contenha a soma de cada objeto ao quadrado (passo 2). Crie novamente outro objeto que contenha a quantidade de meses de 2017 que você estudou (passo 3). Novamente crie outro objeto que seja a razão do objeto ao quadrado criado no passo 1 (numerador) com o objeto criado no passo 3 (denominador) (passo 4). Agora subtraia o objeto criado no passo 2 com o objeto criado no passo 4 e divida pelo objeto criado no passo 3 menos 1. Que medida estatística você calculou?

4 Calcule f(2,1,3), para a função $f(x,y,z)=xyz/(x^2+y^2+z^2)$ (Utilize orientação a objeto!!!!!!!!!)

5 Supondo uma relação linear entre o custo de energia e água da sua casa, utilize 5 meses consecutivos de gastos com energia e água e calcule a covariância entre as duas variáveis (Utilize orientação a objeto!!!!!!!!!!!!)

6 Aproveite os dados do exercício 5 e calcule a correlação linear de pearson (Utilize orientação a objeto!!!!!!!!)

7 Calcule o IMC de uma pessoa que tem 75kg e mede 1,79m. (IMC=peso/alura²) (Utilize orientação a objeto!!!!!!!)

Funções elementares

- `objects()` serve para ver os objetos criados na sessão atual
- `search()` serve para listar dentre outras coisas os paotes carregados na sessão atual
- `rm()` serve para remover um dado objeto criado na sessão atual
- `rm(list = ls())` serve para remover todos os objetos criados de uma única vez
- `print()` serve para imprimir um objeto, uma "string" ou um número.

Exercícios

1. Crie e execute os seguintes objetos: aa = 1, bb = 2, cc = 3, dd = 4, ee = 5. Qual comando você utiliza para ver os objetos criados? Remova os objetos criados e utilize novamente o comando para ver os objetos. Qual é o resultado?

Instalar e carregar pacotes, documentação e ajuda

Instalar e carregar pacotes

- `install.packages()` é a função utilizada para instalar qualquer pacote.
  - o input da função deve ser uma única string.
  - Exemplo: `install.packages('fdth')`

- `library()` ou `require()` são utilizadas para carregar um dado pacote que foi instalado na sessão atual.
  - o input das funções deve ser uma única string.
  - Exemplo: `library('fdth')`
  - a diferença básica entre `library()` e `require()` é que o último é utilizado dentro de funções com a finalidade de checagem, retornando **verdadeiro** ou **falso** caso o pacote exista ou não no sistema.

Documentação

- Manual: pode ser acessado diretamente pelo CRAN.
  - Este documento informa de maneira objetiva quem são os autores, a versão, as funções dentre outras informações.
  - Caso queira algo direto, podes utilizar a função abaixo:

```
manual <- function(package){
  endereco <- paste0('https://cran.r-project.org/web/packages/',
                     package,
                     '/',
                     package,
                     '.pdf')
  browseURL(endereco)
}
manual('fdth')
```

- Vignette: é um documento mais didático que apresenta as finalidades do pacote.
  - Não é obrigatório, ou seja, o pacote pode ter ou não tal documento.
  - A função para acessar tal documento caso haja é `vignette()`.
  - Exemplo: `vignette('fdth')`.

Ajuda

- Quando temos dúvida ou não sabemos ao certo como funciona uma função podemos utilizar a função `help()`.
  - Exemplo: `help(fdt)`.
  - Podemos utilizar a forma contraída `?fdt`.
- Se não sabemos ao certo o nome da função no qual queremos ajuda, podemos utilizar a função `apropos()` para fazer uma busca.
  - Exemplo: `apropos(me)`. Perceba que será retornado todas as funções que tem as letras **me** dos pacotes que estão carregados na sessão atual é claro. Uma busca mais refinada pode ser feito com `apropos(^me)`, ou seja, será retorando todas as funções que **começam** com **me**.

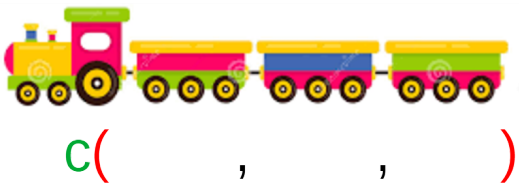
- A função `help.search()` irá buscar na documentação dos pacotes qualquer termo que contenha a palavra buscada retornando os resultados em uma página HTML.
  - Exemplo: `help.search('mea')`.
- A função `help.start()` irá abrir uma página em HTML com várias seções úteis para o usuário, como tutoriais, pacotes instalados na sua máquina entre outros materiais.
- Se precisas de ajuda quanto a pacotes/funções que existem em uma dada área do conhecimento, então precisas instalar o pacote **sos** e executar a função `findFn()`.
  - Exemplo: `findFn('anthropology')`.

Exercícios

1. Instale as bibliotecas fdth e openxlsx. Já é possível utilizar as funções dos pacotes? Explique.
2. Quem são os autores e qual é a versão dos pacotes do exercício 1?
3. Acesse o manual dos pacotes do exercício 1 e cite pelo menos duas funções que contenha exemplos. Copie e cole os exemplos abaixo e execute-os.
4. Peça ajuda a função fdt do pacote fdth e escreva para que serve os argumentos k, start, end e h.
5. Acesse a vignette dos pacotes do exercício 1.
6. Procure se existe algum pacote na área de tratamento de efluente. Utilize o termo em inglês "Effluent treatment".

Vetores

- É uma função utilizada para concatenar (agrupar) valores, semelhante a um vagão de trem. Utilizamos a função `c()` para tal finalidade. Cada valor é separado por vírgula.



- Podemos agrupar valores numéricos ou strings, mas, se agruparmos número e string em um mesmo vagão, então o R entenderá que tudo é string. Percebam que toda string deve estar entre **aspas**.

- Há outras funções para criar vetores.
  - A função `:` cria um vetor sequencial com espaçamento 1.
  - Exemplo:

```
1:5
```

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

```
3:12
```

```
##  [1]  3  4  5  6  7  8  9 10 11 12
```

```
31:29
```

```
## [1] 31 30 29
```

- A função `seq()` cria um vetor cujo o espaçamento da sequência pode ser definido pelo usuário.
  - Exemplo:

```
seq(3,18)# o padrão é espaçamento 1
```

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

```
seq(3,18,by=3)# o argumento by defini qual o espaçamento desejado p
```

```
## [1]  3  6  9 12 15 18
```

```
seq(3,18,length=10)# o argumento length define o tamanho da sequênc
```

```
##  [1]  3.000000  4.666667  6.333333  8.000000  9.666667 11.333333
##  [7] 13.000000 14.666667 16.333333 18.000000
```

Indexação

- É o termo usado quando queremos acessar ou colocar um elemento no vetor. Tal acesso é feito indicando a posição do elemento de interesse. No R, a contagem começa no 1, ao contrário de outras linguagens que a contagem começa no 0.
- A indexação em vetores é feita pelo o nome do objeto seguido de colchetes: `objeto[posição]`.
  - Exemplo:

```
trem <- c(1,15,2)
trem[2] # acessando o elemento que está na posição 2.
```

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

```
trem[1] <- 40 # substituindo o elemento na posição 1 por 40.
trem
```

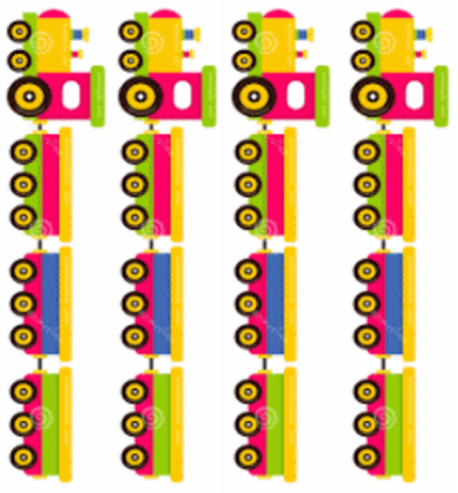
```
## [1] 40 15  2
```

Exercícios

1. Armazene um vetor de -500 a 500, contando de 3 em 3
2. Divida o vetor criado na questão 1 por -3
3. Do vetor criado na questão 1, armazene apenas os negativos
4. Armazene um vetor que vai de 8 a 15, de 3 maneiras diferentes
5. Crie um vetor que armazene o peso de 5 pessoas e os seus respectivos nomes
6. Ainda sobre o vetor criado na questão 5, descubra o se existe alguém com peso maior ou igual a 60kg, e caso, positivo, os armazene em um novo vetor

Matrizes

- É um aglomerado de vetores (vagões) com duas dimensões. A função principal para criar matriz é a `matrix()`.



- Seguindo a mesma ideia dos vetores, só podemos ter matrizes numéricas ou de strings. Caso tenha números e strings misturados, o sistema entenderá que tudo é string.

- Exemplos:

```
mat1 <- matrix(c(2,1,5,6),ncol=2)#o argumento ncol especifica o núm
mat1
```

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

```
mat2 <- matrix(c(2,1,5,6),nrow=2)#o argumento nrow especifica o núm
mat2
```

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

- É possível criar matrizes com um menor controle com as funções `rbind()` e `cbind()`.
  - Com a função `rbind()` podemos formar uma matriz combindando vários vetores do mesmo tamanho. Neste caso, os vetores serão empilhados em linha.
  - Exemplo:

```
vetor1 <- c(4,2,6,10)
vetor2 <- c(21,33,77,90)
rbind(vetor1,vetor2)
```

```
##         [,1] [,2] [,3] [,4]
## vetor1    4    2    6   10
## vetor2   21   33   77   90
```

- Com a função `cbind()` a ideia é a mesma. A diferença é que os vetores serão empilhados como coluna.
  - Exemplo:

```
cbind(vetor1,vetor2)
```

```
##      vetor1 vetor2
## [1,]      4     21
## [2,]      2     33
## [3,]      6     77
## [4,]     10     90
```

Indexação

- Vamos seguir a mesma ideia de vetores. A diferença agora é que teremos duas dimensões.
- Neste caso o objeto é seguido de colchetes no qual deve ser indicado a linha e a coluna: `objeto[linha,coluna]`.
  - Exemplo:

```
mat1[1,2] # irá retornar o elemento que está na primeira linha e segunda
```

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

```
mat1[ ,2] # irá retornar todos os elementos da segunda coluna
```

```
## [1] 5 6
```

```
mat1[2, ] # irá retornar todos os elementos da segunda linha
```

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

```
mat1[1,1] <- 100 # irá substituir o elemento da primeira linha e primeir
```

Exercícios

1. Gere a mesma matriz 3x3, de 3 maneiras diferentes, com valores inseridos por você.
2. Crie uma matriz que indique o número de graduados, pós graduados e doutores de uma universidade, de acordo com as áreas Economia, Sociologia, e Biomedicina.
3. Numa pesquisa, um aluno observou que a massa dos fungos variava de acordo com o pH e tipo substrato utilizado no seu meio. Os dados observados foram:

Substrato	pH.4	pH.6	pH.8
A	0.23	0.68	0.47
B	0.87	0.88	0.75
C	0.11	0.11	0.15

Crie uma matriz com esses dados, nomeie as linhas e colunas, e obtenha o valor médio da massa dos fungos, de acordo com o pH, e de acordo com o tipo de substrato.

4 A partir da matriz obtida na questão 3, obtenha apenas os valores maiores que 0,5 e calcule a média.

5 A partir da matriz obtida na questão 3, altere a massa pesada para o substrato C, no pH=6 para 0,13, supondo que o pesquisador tenha anotado errado o valor. Após alterado, recalcule o valor médio naquela faixa de pH e naquele substrato.

6 Faça a multiplicação das matrizes A e B, e descubra se det(A)*det(B) é igual a det(A*B).

$$A = \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$$ $$B = \begin{bmatrix}-1 & 3\\4 & 2\end{bmatrix}$$

7 De acordo com a matriz obtida a partir da multiplicação de A*B na questão 6, obtenha a diagonal, a matriz inversa (se possível), e a matriz transposta.

Arrays

- Um array é um aglomerado de vetores ou matrizes em três dimensões. A classe array generaliza as classes vetores e matrizes.
- A função utilizada para criar um array é `array()`.
  - O primeiro argumento é o `data` e serve como input dos dados que é um vetor.
  - O segundo argumento é o `dim` que especifica as dimensões do array. Este argumento tem comprimento máximo de 3.
  - Exemplo: `dim(2,4,3)` indica 2 linhas, 4 colunas e 3 dimensões.
- Vejam algums exemplos abaixo.

```
array(data = c(20,2,10), dim = c(3,1,1)) # vetor coluna com 3 linha
```

```
## , , 1
##
##      [,1]
## [1,]   20
## [2,]    2
## [3,]   10
```

```r
array(data = c(20,2,10), dim = c(1,3,1)) # vetor linha com 3 coluna
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]   20    2   10
```

```r
array(data = c(20,2,10,1), dim = c(2,2,1))# matrix com 2 linhas e 2
```

```
## , , 1
##
##      [,1] [,2]
## [1,]   20   10
## [2,]    2    1
```

```r
array(data = c(20,2,10,1,22,4,5,99),dim=c(2,2,2))# duas matrizes co
```

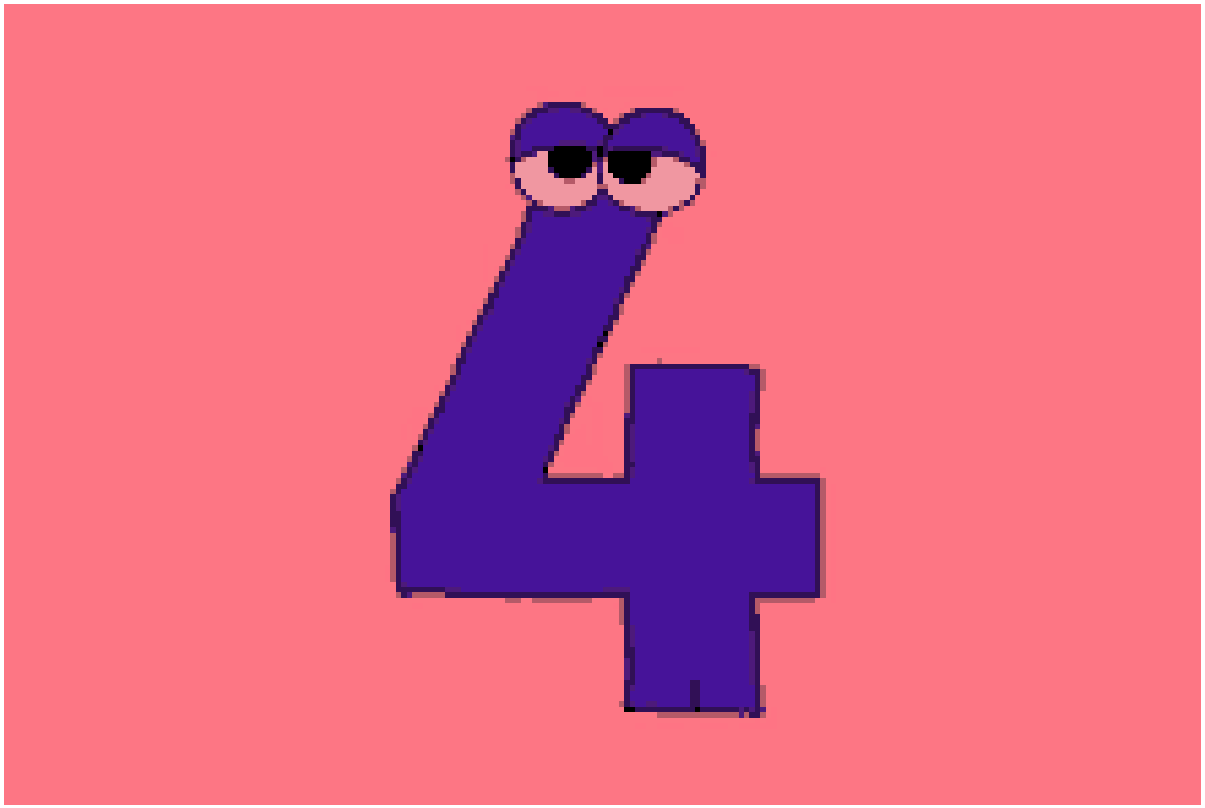
```
## , , 1
##
##      [,1] [,2]
## [1,]   20   10
## [2,]    2    1
##
## , , 2
##
##      [,1] [,2]
## [1,]   22    5
## [2,]    4   99
```

Indexação

- Segue ainda a mesma ideia de vetores e matrizes. No entanto, deve ser indicado três dimensões ao invés de duas de matrizes e uma de vetor.
- Neste caso o objeto é seguido de colchetes com a indicação das três dimensões:`objeto[linha,coluna,terceira_dimensão]`.
  - Exemplo:

```
dad <- array(data=1:24,dim=c(3,2,4))
dad[3,2,2] # acessando a terceira linha, segunda coluna e segunda d
```

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



Data frames

- É uma classe bem semelhante as matrizes. A diferença é que ela é mais flexível, pois adimite em cada coluna números e strings, sem alterar a classe original da coluna.
- Devido a sua flexibilidade, grande parte dos dados importados são convertidos para classe `data.frame`.
- A função utilizada é `data.frame()`.
  - Exemplo:

```
idade <- c(25,20,30,33,27)
sexo <- c('masculino','feminino','feminino','masculino','feminino')
dados <- data.frame(Idade = idade, Sexo = sexo)
dados
```

```
##   Idade      Sexo
## 1    25 masculino
## 2    20  feminino
## 3    30  feminino
## 4    33 masculino
## 5    27  feminino
```

- É possível renomear as colunas e linhas com as funções `names()` e `row.names` respectivamente.
  - Exemplo:

```
names(dados) <- c("idadezinha", "sexozinho")
row.names(dados) <- c("joão","ivan","maria","pedro","marcelo")
dados
```

```
##         idadezinha sexozinho
## joão            25 masculino
## ivan            20  feminino
## maria           30  feminino
## pedro           33 masculino
## marcelo         27  feminino
```

Indexação

- Pode ser feita de três modos distintos:
  - Da mesma forma como matriz, ou seja, por meio de colchetes indicando a linha e a coluna.
  - Exemplo: `objeto[linha,coluna]`.
  - Utilizando `$`. Neste caso a coluna é acessada pelo nome.
  - Exemplo: `objeto$nome`.
  - Utilizando dois colchetes. Veremos mais adiante que esta forma é utilizado em listas.
  - Exemplo: `objeto[[dimensão]]`.

```
dados[4,2] # acessando o elemento da quarta linha e segunda coluna.
```

```
## [1] "masculino"
```

```
dados$idadezinha # acessando a coluna idadezinha
```

```
## [1] 25 20 30 33 27
```

```
dados[[2]] # acessando a segunda coluna
```

```
## [1] "masculino" "feminino"  "feminino"  "masculino" "feminino"
```

```
dados[['sexozinho']] # acessando a segunda coluna
```

```
## [1] "masculino" "feminino"  "feminino"  "masculino" "feminino"
```

- O usuário tem a opção de usar a função `attach()` ao invés de indexação para ter acesso direto as colunas do dataframe.
  - Exemplo:

```
attach(dados)
idadezinha
```

```
## [1] 25 20 30 33 27
```

```
sexozinho
```

```
## [1] "masculino" "feminino"  "feminino"  "masculino" "feminino"
```

- Observação importante: ao terminar de analisar o dataframe com *attach*, utilize a função `detach()` para desfixar o dataframe e evitar qualquer confusão caso trabalhe com outros dataframes.
  - Exemplo:

```
detach(dados)
```

- Outra forma de ter acesso as colunas do dataframe sem indexação é por meio da função `with()`.
  - Exemplo:

```
with(dados, idadezinha)
```

```
## [1] 25 20 30 33 27
```

```
with(dados, sexozinho)
```

```
## [1] "masculino" "feminino"  "feminino"  "masculino" "feminino"
```

```
with(dados, mean(idadezinha))
```

```
## [1] 27
```

Exercícios

1 Os dados a seguir são uma parte dos inscritos no curso "Introdução ao Software R" período 2021/2. Crie um dataframe. Qual é a idade mínima, máxima, primeiro quartil, terceiro quartil, média e mediana?

Idade	Sexo
30	Feminino
20	Masculino
25	Masculino
35	Feminino
24	Masculino
25	Masculino
36	Masculino
23	Masculino
45	Masculino
26	Masculino
25	Feminino

Listas

- É classe mais flexível (opinião minha) do R.
- É possível criarmos objetos com o número de dimensões que quisermos e, em cada dimensão, é possível armazenar diferentes classes de objetos como vetores, matrizes, dataframes, tabelas, etc.

- A função utilizada é `list()`.
  - Exemplo:

```
vetor <- c(2,10,7)
dataframe <- data.frame(var1 = rnorm(4),var2 = rpois(4,lambda=10))
matriz <- matrix(rnorm(20),ncol=5)
escalar <- "Bora vascão"

mylista <- list(vetor,dataframe,matriz,escalar)
mylista
```

```
## [[1]]
## [1]  2 10  7
##
## [[2]]
##         var1 var2
## 1  1.7247198    2
## 2  0.3328630    2
## 3 -0.5381865   13
## 4 -0.4053418   12
##
## [[3]]
##            [,1]       [,2]        [,3]       [,4]       [,5]
## [1,] -0.7676145 -1.0805556  0.35550437  0.1217304  1.4727074
## [2,] -0.5413055 -1.0822497 -1.97216285  0.7206983 -0.6413169
## [3,] -1.3324869 -1.8528108  0.43521907 -0.7564931 -0.1494978
## [4,]  0.4687019  0.6116946 -0.06626898  0.5337566  0.6852557
##
## [[4]]
## [1] "Bora vascão"
```

- É possível nomear/renomear cada slot(andar/gaveta) da lista por meio da função `names()`.
  - Exemplo:

```
names(mylista) <- c("vetor","dataframe","matriz","escalar")
mylista
```

```
## $vetor
## [1]  2 10  7
##
## $dataframe
##         var1 var2
## 1  1.7247198    2
## 2  0.3328630    2
## 3 -0.5381865   13
## 4 -0.4053418   12
##
## $matriz
##            [,1]       [,2]        [,3]       [,4]       [,5]
## [1,] -0.7676145 -1.0805556  0.35550437  0.1217304  1.4727074
## [2,] -0.5413055 -1.0822497 -1.97216285  0.7206983 -0.6413169
## [3,] -1.3324869 -1.8528108  0.43521907 -0.7564931 -0.1494978
## [4,]  0.4687019  0.6116946 -0.06626898  0.5337566  0.6852557
##
## $escalar
## [1] "Bora vascão"
```

Indexação

- Pode ser feita de duas formas:
  - Utilizando `$`, ou seja, `objeto$nome`.
  - Utilizando dois colchetes `objeto[[dimensão]]`.
  - Exemplo:

```
mylista[[1]]
```

```
## [1]  2 10  7
```

```
mylista$vetor
```

```
## [1]  2 10  7
```

Tabelas

- Uma das formas de criar tabelas é por meio da função `table()`.
- Fazendo uma tabela simples.
  - Exemplo:

```
sexo  <- c("Feminino","Masculino","Masculino","Feminino","Masculino
table(sexo)
```

```
## sexo
##  Feminino Masculino
##         3         8
```

- Fazendo uma tabela de dupla entrada.
  - Exemplo:

```
ocupacao <- c('gradUESC','gradUESC','posUESC','posUESC','gradUESC',
dad2 <- data.frame(Ocupacao = ocupacao, Sexo = sexo)
table(dad2)
```

```
##            Sexo
## Ocupacao    Feminino Masculino
##   gradUESC         1         4
##   posOUTRA         1         1
##   posUESC          1         2
##   profUESC         0         1
```

- Fazendo uma tabela multidimensional.
  - Exemplo:

```
sistemaoperacional <- c('Windows','Windows','Windows','Windows','Wi
dad3 <- data.frame(Ocupacao = ocupacao, Sexo = sexo, SO = sistemaop
table(dad3)
```

```
## , , SO = Windows
##
##           Sexo
## Ocupacao   Feminino Masculino
##   gradUESC        1         4
##   posOUTRA        1         1
##   posUESC         1         2
##   profUESC        0         0
##
## , , SO = Windows e Linux
##
##           Sexo
## Ocupacao   Feminino Masculino
##   gradUESC        0         0
##   posOUTRA        0         0
##   posUESC         0         0
##   profUESC        0         1
```

- Existe a função `ftable()`. Inclusive, tabelas multidimensionais ficam mais elegantes.
  - Exemplo:

```
ftable(dad3)
```

```
##                     SO Windows Windows e Linux
## Ocupacao Sexo
## gradUESC Feminino            1                0
##          Masculino           4                0
## posOUTRA Feminino            1                0
##          Masculino           1                0
## posUESC  Feminino            1                0
##          Masculino           2                0
## profUESC Feminino            0                0
##          Masculino           0                1
```

- Calculando proporções.

```
tabela <- table(dad2)# vejam que é necessário criar uma tabela primeiro!
prop.table(tabela)# percebam que o input é da classe table.
```

```
##           Sexo
## Ocupacao     Feminino  Masculino
##   gradUESC 0.09090909 0.36363636
##   posOUTRA 0.09090909 0.09090909
##   posUESC  0.09090909 0.18181818
##   profUESC 0.00000000 0.09090909
```

```
prop.table(tabela,margin=1)#tomando a linha como referência
```

```
##           Sexo
## Ocupacao    Feminino Masculino
##   gradUESC 0.2000000 0.8000000
##   posOUTRA 0.5000000 0.5000000
##   posUESC  0.3333333 0.6666667
##   profUESC 0.0000000 1.0000000
```

```
prop.table(tabela,margin=2)#tomando a coluna como referência
```

```
##           Sexo
## Ocupacao    Feminino Masculino
##   gradUESC 0.3333333 0.5000000
##   posOUTRA 0.3333333 0.1250000
##   posUESC  0.3333333 0.2500000
##   profUESC 0.0000000 0.1250000
```

Como criar funções

- Para criar funções basta utilizar `function(argumento1, argumento2, ...)`.
- O número de argumentos fica a critério do usuário.
  - Exemplo:

```
foo <- function(x) x/10 + 1 # função curta
foo(x = 10) # chamada explícita
```

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

```
foo(10) # chamada implícita
```

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

```
foo2 <- function(x,y){ # função com mais de uma linha
     num <- x + 5
     den <- y^2
     res <- num/den
     return(res)
}
foo2(x = 5, y = 2)
```

```
## [1] 2.5
```

```
foo2(2, 5) # se for implícito, mantém-se a ordem dos argumentos, ou
```

```
## [1] 0.28
```

```
foo2(y = 3, x = 1) # posso alterar a ordem dos input desde que eu d
```

```
## [1] 0.6666667
```

- Podemos criar uma função com outras funções prontas.
  - Exemplo:

```
media <- function(x){
      sum(x)/length(x)
}
```

- Podemos ainda aproveitar argumentos de outras funções, desde que, utilizemos a diretiva **...** da seguinte forma: `function(x,...)`.
  - Exemplo:

```
cv1 <- function(x){
 sd(x)/mean(x)
}
vetor <- c(40,3,12,NA,1)
cv1(vetor) # perceba que o resultado será NA.
```

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

```
cv2 <- function(x,...){
 sd(x,...)/mean(x,...)
}
cv2(vetor, na.rm = TRUE)# perceba que o argumento na.rm não foi cri
```

```
## [1] 1.284391
```

- Se fizestes uma função muito utilizada, podes salvar em um arquivo com extensão ".R" e invocá-la por meio da função `source()`.
- Supondo que exista um arquivo "cv.R" no mesmo diretório da sessão atual, então:

```
getwd()# verificando o diretório da sessão atual
```

```
## [1] "C:/Users/ivana/Dropbox/Documentos/UESC/cursos_extensao/R/2021_2/basi
```

```
source('cv.R') # carregando o arquivo
cv(vetor, na.rm=TRUE)
```

```
## [1] 1.284391
```

Condicionais e loop

Condicionais

- As principais funções condicionais são: `if()` (se),`else`(senão),`ifelse()`(se senão).

```
if(condição){
 escreva o código desejado se a condição for satisfeita!
}else{
 escreva o código desejado caso a condição não seja satisfeita!
}
```

- Exemplo:

```
clube = 'curintia'

if(clube == 'flamengo'){
 print('Timeco!!!!!!!!')
}else{
 print('Time bom!!!!')
}
```

```
## [1] "Time bom!!!!"
```

- Podemos aninhar quantos `if(){}else{}` desejarmos.
  - Exemplo:

```
clube <- 'palmeiras'
if(clube == 'flamengo'){
 print('Timeco!!!!!!')
}else if(clube == 'vasco'){
 print('Um dia sobe!!!!')
}else{
 print('Time bom!!!!')
}
```

```
## [1] "Time bom!!!!"
```

- A função `ifelse()` é mais utilizada em situações mais curtas.

```
ifelse(condição, resultado se verdadeiro, resultado se falso)
```

- Exemplo:

```
clube <- 'vasco'
ifelse(clube == 'flamengo','Timeco','Timão')
```

```
## [1] "Timão"
```

Loop

- As funções utilizadas em loop são: `for(){}`, `while()` e `repeat()`.
- Deve-se tomar um cuidado muito grande com as funções `while` e `repeat` para não entra em um **loop infinito**.

```
for(sequêcia a ser percorrida){
   objeto a ser percorrido
}
```

```
while(condição de permanência do loop){
   código desejado
}

repeat{
  código desejado

  if(condição de parada!){
     break() #função de parada!
  }
}
```

- Exemplo:

```
for(i in 1:10){
print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```r
frase <- c("Eu","vou","ganhar","na","mega","da","virada")
for(i in 1:length(frase)){
 print(frase[i])
}
```

```
## [1] "Eu"
## [1] "vou"
## [1] "ganhar"
## [1] "na"
## [1] "mega"
## [1] "da"
## [1] "virada"
```

```r
pandemia <- 2019
while(pandemia < 2021){
  pandemia <- pandemia + 1
  print('Tamo lascado!')
}
```

```
## [1] "Tamo lascado!"
## [1] "Tamo lascado!"
```

```r
pandemia <- 2019
repeat{
 pandemia <- pandemia + 1
 print(paste('Tamo lascado! Ainda em ', pandemia))
 if(pandemia > 2021){
 print(paste('Ebaaaaaa chegou a vacina!!!!!!!Estamos em ', pandemia
 break()
 }
}
```

```
## [1] "Tamo lascado! Ainda em  2020"
## [1] "Tamo lascado! Ainda em  2021"
## [1] "Tamo lascado! Ainda em  2022"
## [1] "Ebaaaaaa chegou a vacina!!!!!!!Estamos em  2023"
```