

Estrutura de dados

As referências para esta aula são: Irizarry (2019), Boehmke (2016), Wickham, Grolemund, et al. (2017), Zamora Saiz et al. (2020) e Wickham (2019) e Grolemund (2014)

Introdução

- No nosso cotidiano, lidamos com diferentes tipos de informações o tempo todo, mesmo sem perceber.
 - Quando fazemos compras, lidamos com itens (texto), quantidades (números inteiros) e preços (números decimais)
 - Imagine que você registra suas viagens, anotando a distância percorrida (km), o tempo gasto (horas) e se houve ou não acidentes
- Como analistas de dados, o “ingrediente” básico para qualquer estudo são os dados que consideramos.
- Os dados podem ser apresentados como:
 - **números**, que podem ser inteiros ou decimais;
 - **palavras**, seja uma única palavra, uma frase inteira ou até mesmo uma sequência de caracteres, como um código ou uma senha.
 - **sim/não**, uma informação dicotômica.
- Uma vez que cada informação é compreendida, pode haver uma estrutura subjacente nos dados, como uma **sequência de valores**, uma **tabela cruzada** entre duas variáveis ou uma **lista** de categorias ordenadas para determinados indivíduos.
- Compreender essas estruturas é o primeiro passo essencial para a análise de dados.
- No contexto da linguagem de programação R, **objetos e vetores** são conceitos fundamentais relacionados à manipulação de dados;

Objetos

Considere este código:

```
x <- 1
```

- Esse código está fazendo duas coisas:
 - Ele está criando um objeto.
 - E está vinculando esse objeto a um nome, **x**.
- O **nome** faz uma referência a um valor
- Um objeto é simplesmente um valor que é guardado dentro do **nome**;
- Para criar um objeto, escolha um **nome** e use <- ou = para guardar a informação dentro do nome.
- O sinal <- cria uma vinculação do nome no lado esquerdo para o objeto no lado direito.
- O “endereço” de memória do objeto, ou seja, o local na memória onde o objeto é armazenado é obtido

```
lobstr::obj_addr(x)
```

```
[1] "0x203cbf57ba8"
```

- Outros operadores importantes são:

```
x == y    # Igual a
x != y    # Diferente de
x < y     # Menor que
x > y     # Maior que
x <= y    # Menor ou igual a
x >= y    # Maior ou igual a
```

```
a <- 10
b <- 5
soma <- a + b      # Adição
subtracao <- a - b  # Subtração
multiplicacao <- a * b # Multiplicação
divisao <- a / b    # Divisão
modulo <- a %% b    # Resto da divisão
potencia <- a ^ b   # Potência
```

- Considere os exemplos abaixo

```
a <- 2  
a = 2  
print(a)
```

```
[1] 2
```

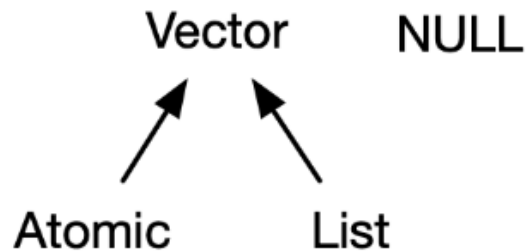
```
b <- 1:6  
print(b)
```

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

- Observação: quando o objeto for criado, ele aparecerá no painel *Environment*
- Ao escolher nomes para objetos em R, é importante seguir algumas regras e boas práticas para garantir a clareza, consistência e evitar conflitos com palavras reservadas.
- Aqui estão algumas restrições para nomear objetos em R:
 - **Sintaxe básica:**
 - * Os nomes de objetos devem começar com uma letra.
 - * Podem conter letras, números e pontos (.), mas **não podem começar com um número ou conter espaços**.
 - * Evite o uso de caracteres especiais, como @, \$, %, &, etc.
 - **Palavras reservadas:**
 - * Evite usar nomes que são palavras reservadas em R, pois isso pode causar conflitos. Alguns exemplos de palavras reservadas incluem **if**, **else**, **while**, **function**, **for**, **in**, **TRUE**, **FALSE**, entre outras.
 - **Observação:**
 - * O R diferencia letras **maiúsculas e minúsculas**, isto é, **a** é considerado um objeto diferente de **A**
 - * Escolha nomes descritivos que forneçam informações sobre o propósito ou conteúdo do objeto.
 - * No R, uma base de dados é representada por objetos chamados de *data frames*
 - * Exemplos de nomes aceitáveis: `idade`, `nomeVariavel`, `meuVetor`, `resultado_final` e `dados_do_paciente`

Vetores

- Um vetor é uma estrutura de dados unidimensional que pode conter ou não elementos de um único tipo
 - Os vetores podem ser subdivididos em : **vetores atômicos** e **listas**
 - Eles diferem quanto aos tipos de seus elementos:
 - para **vetores atômicos**, todos os elementos devem ter o mesmo tipo;
 - **para listas**, os elementos podem ter tipos diferentes.
 - Os elementos de um vetor são acessados por índices.
-



Vetores atômicos

- Existem quatro tipos principais de vetores atômicos: lógico, inteiro, *double* e caractere (que contém strings);
- Vetores inteiros e *double* são conhecidos como vetores numéricos;
- Para criar vetores use a função `c()` (*combine*)
- Para saber o tipo de vetor, você pode utilizar a função `typeof()`. Para saber seu comprimento a função `length()`.
- Você pode **testar** se um vetor é de um determinado tipo com uma função `is.*()`

Lógico

```
lgl_var <- c(TRUE, FALSE)# lógico  
lgl_var <- c(T, F)# lógico  
typeof(lgl_var)# verificar o tipo
```

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

```
is.logical(lgl_var) # testar se o vetor é do tipo lógico
```

```
[1] TRUE
```

```
is.integer(lgl_var)
```

```
[1] FALSE
```

```
length(lgl_var)
```

```
[1] 2
```

Double

```
dbl_var <- c(1, 2.5, 4.5)#forma decimal  
dbl_var <- c(1.23e4)##forma científica  
typeof(dbl_var)
```

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

```
is.double(dbl_var) # testar se o vetor é do tipo Double
```

```
[1] TRUE
```

```
is.character(lgl_var)
```

```
[1] FALSE
```

```
length(dbl_var)
```

```
[1] 1
```

- Existem três valores especiais exclusivos para Double: Inf, -Inf e NaN (Not a Number)

```
dbl_var <- c(Inf,-Inf,NaN)  
typeof(dbl_var)
```

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

Inteiro

- Os inteiros são escritos de forma semelhante aos Double, mas devem ser seguidos por L

```
int_var <- c(1L, 6L, 10L)# inteiro  
typeof(int_var)
```

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

```
is.integer(int_var)
```

```
[1] TRUE
```

```
is.character(int_var)
```

```
[1] FALSE
```

```
length(int_var)
```

```
[1] 3
```

```
int_var_1 = 1:10 # criando uma sequência de inteiros de 1 a 10  
typeof(int_var_1)
```

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

Caractere (string)

- As strings são colocadas entre " "

```
chr_var <- c("Leonardo", "Nascimento")
chr_var <- c("Ótimo", "Bom", "Ruim")
chr_var <- c("Masculino", "Feminino")
is.character(chr_var)
```

```
[1] TRUE
```

```
typeof(chr_var)
```

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

Observações

- Em um vetor, cada valor ocupa uma posição específica determinada pela ordem em que os elementos foram adicionados durante a criação do vetor.
- Essa ordem é crucial para acessar cada valor de maneira individual dentro do vetor.

```
meu_vetor = c(1,2,10,4,5)
meu_vetor[3]
```

```
[1] 10
```

NULL

- Embora não seja um vetor, NULL está intimamente relacionado aos vetores e geralmente desempenha a função de um vetor genérico de comprimento zero.

```
vetor_null <- c(NULL)
vetor_null
```

```
NULL
```

```
typeof(vetor_null)
```

```
[1] "NULL"
```

NA

- Em R, “NA” (*Not Available*) é usado para representar valores ausentes ou desconhecidos.
- A maioria dos cálculos envolvendo um valor faltante retornará outro valor faltante.

```
meu_vetor <- c(10, NA)  
2*meu_vetor
```

```
[1] 20 NA
```

- Para verificar se um valor é “NA”, você pode usar a função `is.na()`.

```
x <- c(1, 2, 3, NA, 5)  
is.na(x)
```

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

- Muitas funções em R têm maneiras para lidar com valores ausentes. Por exemplo, algumas funções têm argumentos como `na.rm` para remover NAs durante cálculos

```
notas_alunos <- c(10, 7, NA, 8, 8.5)  
mean(notas_alunos, na.rm = TRUE) # média
```

```
[1] 8.375
```

- Você pode substituir valores NA por outros valores usando a função `is.na()` e indexação

```
meu_vetor <- c(1, 2, 3, 4, NA)  
meu_vetor[is.na(meu_vetor)] <- 0
```


Coerção

- Para vetores atômicos, o tipo é uma propriedade de todo o vetor
- Todos os elementos devem ser do mesmo tipo.
- Quando você tenta combinar tipos diferentes, eles serão forçados em uma ordem fixa: caractere → double → inteiro → lógico.

```
y1 <- c(1L,"leonardo") # inteiro, character
y1
```

```
[1] "1"          "leonardo"
```

```
typeof(y1)
```

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

```
y2 <- c(5.5,10L) # double, inteiro
y2
```

```
[1] 5.5 10.0
```

```
typeof(y2)
```

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

Listas

- As listas são um avanço em complexidade em relação aos vetores atômicos: cada elemento pode ser de qualquer tipo
- Você constrói listas com a função `list()`

```
l1 <- list(
  1:3, # criando
  "a",
  c(TRUE, FALSE, TRUE),
  c(2.3, 5.9)
)
print(l1)
```

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

```
[[2]]  
[1] "a"
```

```
[[3]]  
[1] TRUE FALSE TRUE
```

```
[[4]]  
[1] 2.3 5.9
```

```
typeof(l1)
```

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

- Para acessar um elemento da lista usamos `[[]]`

```
l1 <- list(  
  1:3,  
  "a",  
  c(TRUE, FALSE, TRUE),  
  c(2.3, 5.9)  
)  
l1[[4]]
```

```
[1] 2.3 5.9
```

```
l1[[4]][1]
```

```
[1] 2.3
```

- Você pode testar uma lista com `is.list()` e forçar uma lista com `as.list()`

```
minha_lista = list(1:3)  
print(minha_lista)
```

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

```
is.list(minha_lista)
```

```
[1] TRUE
```

```
vec <- c(1,2,3)
is.list(vec)
```

```
[1] FALSE
```

```
as.list(vec)
```

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

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

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

Você pode transformar uma lista em um vetor atômico com `unlist()`.

```
minha_lista = list(1:3,4:10)
print(minha_lista)
```

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

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

```
unlist(minha_lista)
```

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

- As regras para o tipo resultante são complexas, não estão bem documentadas e nem sempre são equivalentes ao que você obterá com `c()`.

Matrizes

- Uma matriz em R é uma estrutura bidimensional que pode armazenar dados de um único tipo.
- Isso significa que todos os elementos de uma matriz devem ser do mesmo tipo, como números inteiros, *double* ou caracteres.
- Você pode criar uma matriz usando a função **matrix()**. Especifique os dados e o número de linhas e colunas.

```
# Criando uma matriz 2x2
vec1 = c(1,2)
vec2 = c(3,4)
minha_matriz <- matrix(c(vec1,vec2), nrow = 2, ncol = 2)
minha_matriz
```

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

```
is.matrix(minha_matriz)
```

```
[1] TRUE
```

```
vec1 = c(1,2)
vec2 = c(3,4)
minha_matriz <- matrix(c(vec1,vec2), nrow = 2, ncol = 2, byrow = T)
minha_matriz
```

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

```
dim(minha_matriz)
```

```
[1] 2 2
```

```
ncol(minha_matriz)
```

```
[1] 2
```

```
nrow(minha_matriz)
```

```
[1] 2
```

- Os elementos de uma matriz podem ser acessados usando índices de linha e coluna.

```
minha_matriz[1,2] # Acessando o elemento na primeira linha e segunda coluna
```

```
[1] 2
```

- Adicionando nomes

```
rownames(minha_matriz) = c("linha1","linha2")
colnames(minha_matriz) = c("C1","C2")
minha_matriz
```

```
      C1 C2
linha1  1  2
linha2  3  4
```

```
minha_matriz[, "C1"] # acessando a coluna 1
```

```
linha1 linha2
      1      3
```

Array

- Um array em R é uma estrutura de dados multidimensional que pode conter elementos de um único tipo. Diferentemente das matrizes, os arrays podem ter mais de duas dimensões.
- Você pode criar um array usando a função **array()**. Especifique os dados e as dimensões.

```
# Criando um array 2x2x3 - linhas X colunas X camadas
vec1 = c(1L,2L,3L,4L)
vec2 = c(5L,6L,7L,8L)
vec3 = c(9L,10L,11L,12L)
meu_array <- array(c(vec1,vec2,vec3), dim = c(2,2,3))
meu_array
```

```
, , 1
```

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

```
, , 2
```

```
      [,1] [,2]
[1,]     5     7
[2,]     6     8
```

```
, , 3
```

```
      [,1] [,2]
[1,]     9    11
[2,]    10    12
```

```
typeof(meu_array)
```

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

- Usando o argumento `dimnames` para nomear as dimensões

```
dimnames(meu_array) <- list(linhas=c("Linha 1","Linha 2"),
                             Colunas = c("Col 1","Col 2"),
                             Camadas = c("C1","C2","C3"))
```

- Os elementos de um array são acessados usando índices correspondentes às dimensões.

```
meu_array[1,2,2]# linhas X colunas X camadas
```

```
[1] 7
```

```
meu_array[, , 1] # acessando a matriz da primeira camada
```

	Colunas	
linhas	Col 1	Col 2
Linha 1	1	3
Linha 2	2	4

```
meu_array[, , 2] # acessando a matriz da segunda camada
```

	Colunas	
linhas	Col 1	Col 2
Linha 1	5	7
Linha 2	6	8

```
meu_array[1, , ] # acessando a primeira linha de todas as camadas
```

	Camadas		
Colunas	C1	C2	C3
Col 1	1	5	9
Col 2	3	7	11

- Titanic é exemplo de dados que estão organizados no formato *array* de 4 dimensões

No	Nome	Níveis
1	Classe	1 ^a , 2 ^a , 3 ^a , Tripulação
2	Sexo	Masculino, Feminino
3	Idade	Criança, Adulto
4	Sobreviveu	Não, Sim

Titanic

```
, , Age = Child, Survived = No
```

	Sex	
Class	Male	Female
1st	0	0
2nd	0	0
3rd	35	17

```

Crew    0      0

, , Age = Adult, Survived = No

      Sex
Class  Male Female
1st    118      4
2nd    154     13
3rd    387     89
Crew   670      3

, , Age = Child, Survived = Yes

      Sex
Class  Male Female
1st     5      1
2nd    11     13
3rd    13     14
Crew    0      0

, , Age = Adult, Survived = Yes

      Sex
Class  Male Female
1st    57     140
2nd    14      80
3rd    75      76
Crew   192     20

```

Data frame

- No R, um data frame é uma estrutura de dados bidimensional semelhante a uma tabela em um banco de dados relacional ou a uma planilha.
- Cada coluna em um data frame pode conter dados de diferentes tipos, tornando-os especialmente úteis para representar conjuntos de dados complexos.
- Você pode criar um data frame manualmente usando a função `data.frame()`.

```

meu_data_frame <- data.frame(
  Nome = c("Alice","Leo","Vitor"),
  Idade = c(25,30,22),

```



```

    Nota = c(85, 92, 78)
)
str(meu_data_frame) # informações sobre as colunas

```

```

'data.frame':   3 obs. of  3 variables:
 $ Nome : chr  "Alice" "Leo" "Vitor"
 $ Idade: num  25 30 22
 $ Nota : num  85 92 78

```

```
meu_data_frame
```

	Nome	Idade	Nota
1	Alice	25	85
2	Leo	30	92
3	Vitor	22	78

```

dados_climaticos <- data.frame(
  Dia = seq(from = as.Date("2023-01-01"), by = "days", length.out = 5),
  Temperatura = c(25.3, 24.5, 22.0, 26.8, 23.5),
  Umidade = c(65, 70, 75, 60, 80),
  VelocidadeVento = c(10, 12, 8, 15, 9)
)

# Exibindo o data frame
print(dados_climaticos)

```

	Dia	Temperatura	Umidade	VelocidadeVento
1	2023-01-01	25.3	65	10
2	2023-01-02	24.5	70	12
3	2023-01-03	22.0	75	8
4	2023-01-04	26.8	60	15
5	2023-01-05	23.5	80	9

```
head(dados_climaticos,3)
```

	Dia	Temperatura	Umidade	VelocidadeVento
1	2023-01-01	25.3	65	10
2	2023-01-02	24.5	70	12
3	2023-01-03	22.0	75	8

- Você pode acessar uma coluna específica usando o nome da coluna.
- Você também pode acessar elementos por índices de linha e coluna

```
dados_climaticos$Temperatura
```

```
[1] 25.3 24.5 22.0 26.8 23.5
```

```
dados_climaticos[,2]
```

```
[1] 25.3 24.5 22.0 26.8 23.5
```

```
dados_climaticos[1:3,2]
```

```
[1] 25.3 24.5 22.0
```

- Outra maneira de acessar as informações é usando a função `subset()`

```
subset(dados_climaticos, Umidade > 70)
```

	Dia	Temperatura	Umidade	VelocidadeVento
3	2023-01-03	22.0	75	8
5	2023-01-05	23.5	80	9

Geração de sequências

- Existem alguns comandos no R que são especialmente úteis para criar vetores de números;
- A função `seq()` gera uma sequência de números com uma progressão aritmética especificada.

```
# Gerar uma sequência de números de 1 a 21 com incrementos de 2
seq(from = 1, to = 21, by = 2)
```

```
[1] 1 3 5 7 9 11 13 15 17 19 21
```

```
# Gerar uma sequência de números de 0 a 21 com 15 valores igualmente espaçados
seq(0, 21, length.out = 15)
```

```
[1] 0.0 1.5 3.0 4.5 6.0 7.5 9.0 10.5 12.0 13.5 15.0 16.5 18.0 19.5 21.0
```

- A função `rep()` permite repetir constantes especificadas em longos vetores

```
# Replica os valores de 1 a 4 (seq inteira) um número especificado de vezes
rep(1:4, times = 2)
```

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

```
# Replica cada número da seq de acordo com o vetor definido no argumento times
rep(1:4, times = c(1,2,3,10))
```

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

```
# Replica os valores de 1 a 4 de forma agrupada (cada valor repetido antes de passar para o próximo)
rep(1:4, each = 2)
```

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

```
# Replica os valores de 1 a 4 de forma agrupada (cada valor repetido duas vezes) até atingir o comprimento desejado
rep(1:4, each = 2, length.out = 16)
```

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

```
# Replica as letras A e B (seq inteira) duas vezes
rep(LETTERS[1:2], times = 2)
```

```
[1] "A" "B" "A" "B"
```

- Alternativamente, você pode usar o comando `sample()` para gerar uma amostra aleatória com reposição (o mesmo elemento pode ser selecionado mais de uma vez) ou sem reposição (cada elemento é escolhido apenas uma vez)

```
# Selecionar (sem reposição) 3 números de 1 a 5.
sample(1:5, 3, replace = FALSE, prob = NULL)
```

```
[1] 5 3 4
```

```
# Selecionar (com reposição) 3 números de 1 a 5
sample(1:5, 3, replace = TRUE, prob = NULL)
```

```
[1] 4 3 4
```

```
# Selecionar (com reposição) 3 números de 1 a 5 considerando as chances (probabilidades)
sample(1:5, 3, replace = TRUE, prob = c(0.30,0.40,0.15,0.10,0.05))
```

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

```
# Gera uma sequência de 100 valores, escolhendo aleatoriamente entre 0 e 1, com reposição  
sample(c(0,1), 10, replace = TRUE)
```

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

Atributos

- Além dos próprios elementos, os vetores podem ter atributos que fornecem informações adicionais sobre os dados



Nomes

- Você pode atribuir nomes a cada elemento do vetor usando a função **names()**. Isso facilita a referência a elementos específicos pelo nome.

```
meu_vetor <- c(1, 2, 3)
names(meu_vetor) <- c("primeiro", "segundo", "terceiro")
meu_vetor
```

```
primeiro  segundo terceiro
      1         2         3
```

```
attributes(meu_vetor)
```

```
$names
[1] "primeiro" "segundo" "terceiro"
```

- Formas alternativas para nomear um vetor

```
x1 <- c(a = 1, b = 2, c = 3)
x1
```

```
a b c
1 2 3
```

```
x2 <- setNames(1:3, c("a", "b", "c"))
x2
```

```
a b c
1 2 3
```

Dimensão

- Em R, vetores podem ter atributos de dimensão, que são comumente associados a matrizes.

```
meu_vetor <- 1:5
meu_vetor
```

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

```
dim(meu_vetor) <- c(5, 1) # linha x coluna
meu_vetor
```

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

```
attributes(meu_vetor)
```

```
$dim
[1] 5 1
```

Classe

- A classe de um objeto é uma propriedade que indica a natureza ou tipo do objeto em termos de programação orientada a objetos
- A classe é uma parte fundamental em R, pois determina como o objeto será tratado em operações específicas e quais métodos (funções associadas) podem ser aplicados a ele.
- Aqui estão algumas das classes mais comuns em R:
 1. **numeric:** Números reais (*double*).
 2. **integer:** Números inteiros.
 3. **logical:** Valores lógicos (TRUE ou FALSE).
 4. **character:** Strings de caracteres.
 5. **factor:** Fatores, usados para representar variáveis categóricas com níveis
 6. **Date:** Representação de datas.
 7. **POSIXct e POSIXlt:** Representação de datas e horas.
 8. **data.frame:** Uma estrutura bidimensional que pode conter colunas de diferentes classes.
 9. **matrix:** Uma estrutura bidimensional que contém elementos de uma única classe.
 10. **array:** Uma estrutura de dados multidimensional que pode conter elementos de uma única classe.

11. **list:** Uma estrutura que pode conter elementos de diferentes classes e até outras listas.

12. **function:** Funções.

- Em R, a função **class()** é usada para obter a classe de um objeto

```
vec1 = c(1,2)
vec2 = c(3,4)
class(vec1)
```

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

```
minha_matriz <- matrix(c(vec1,vec2), nrow = 2, ncol = 2)
minha_matriz
```

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

```
class(minha_matriz)
```

```
[1] "matrix" "array"
```

- Você pode atribuir uma classe a um vetor usando a função **class()**. Isso é comumente usado em programação orientada a objetos em R.

```
vec <- c(1,2,3,4)
class(vec)
```

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

```
class(vec)<- "minha_classe"
class(vec)
```

```
[1] "minha_classe"
```

Exemplo

- Criando um objeto da classe Pessoa

```
peessoa1 <- structure(  
  c("Leonardo"),  
  idade = 31,  
  last_name = "Nascimento",  
  class = "Pessoa"  
)  
peessoa1
```

```
[1] "Leonardo"  
attr(,"idade")  
[1] 31  
attr(,"last_name")  
[1] "Nascimento"  
attr(,"class")  
[1] "Pessoa"
```

```
class(peessoa1)
```

```
[1] "Pessoa"
```

```
attr(peessoa1,"idade") # selecionado o atributo idade
```

```
[1] 31
```

```
names(peessoa1)
```

```
NULL
```

Atividades

Questão 1

```
#'Criar uma classe simples chamada "Produto" que representa um produto.  
#'Considere os atributos: preço e validade.  
#'Utilizar a função structure para criar objetos da classe "Produto".
```



```
#' Considere o exemplo para trabalhar com datas

exem_data = as.Date("2023-12-31") # ano-mês-dia
data_formatada = format(exem_data, format = "%d/%m/%Y") # dia-mês-ano
data_formatada
```

Questão 2

```
#'Criar uma classe simples chamada "Livro" que representa um livro com atributos: autor e an
#'Utilizar a função structure para criar objetos da classe "Livro" com informações fictícias
```

Questão 3

```
## letra a)-----
#' Crie um data.frame contendo as variáveis: nome e nota
#' escolha os nomes e valores
#' 5 linhas

## letra b)-----
#' Adicione um atributo chamado "disciplina", representando o nome da disciplina e
#' Exiba o atributo "disciplina"

## letra c)-----
#'Modifique o valor do atributo "disciplina" para representar outra disciplina.

## letra d)-----

#'Adicione o atributo "semestre" e Exiba o atributo "semestre"
```

Questão 4

1. Crie um data frame com as seguintes informações:
 1. Coluna 1: nome de duas cidades, cada uma repetindo 5 vezes.
 2. Coluna 2: variável temperatura, escolha o valor da temperatura para as cidades

3. Para cada cidade, calcule a média (`mean()`) e o desvio padrão (`sd()`) da temperatura.
4. Para cada cidade, use a função `summary()`
5. Transforme o data frame em uma matriz. O que acontece?

Questão 5

Crie uma matriz 3×5 contendo todos os números consecutivos entre 16 e 30, organize por colunas.

Questão 6

Use o data frame CO2 da biblioteca `dataset` para realizar as seguintes tarefas:

1. Acessar o help `?datasets::CO2` para conhecer a base de dados
2. Observe o data frame para identificar as diferentes variáveis incluídas
3. Calcule a média e o desvio padrão das taxas de absorção de CO2 (*uptake*) em Quebec.
4. Calcule as concentrações mínima e máxima de CO2 para as plantas que estão sob o tratamento *chilled*.

Boehmke, Bradley C. 2016. *Data wrangling with R*. Springer.

Grolemund, Garrett. 2014. *Hands-on programming with R: Write your own functions and simulations*. " O'Reilly Media, Inc."

Irizarry, Rafael A. 2019. *Introduction to data science: Data analysis and prediction algorithms with R*. Chapman; Hall/CRC.

Wickham, Hadley. 2019. *Advanced r*. chapman; hall/CRC.

Wickham, Hadley, Garrett Grolemund, et al. 2017. *R for data science*. Vol. 2. O'Reilly Sebastopol, CA.

Zamora Saiz, Alfonso, Carlos Quesada González, Lluís Hurtado Gil, e Diego Mondéjar Ruiz. 2020. *An Introduction to Data Analysis in R: Hands-on Coding, Data Mining, Visualization and Statistics from Scratch*. Springer International Publishing.