MAC 113 — Introdução à Ciência da Computação

Aula 27

Nelson Lago

1°/2025

Previously on MAC113...

ggplot2

- **Em um gráfico, representamos dados com formas geométricas (pontos, linhas, caixas, barras...) e seus atributos estéticos (posição, forma, tamanho, cor...)**
  - Em um gráfico de linha, a forma é a linha e o atributo estético usado geralmente é a sua posição no plano cartesiano (coordenadas x, y)
- **Gráficos com `ggplot2` são construídos em camadas**
  - Adicionamos camadas com "`+`"
  - A primeira "camada" é a área (vazia) do gráfico: `ggplot(df)`
    - » *(um "atalho" comum é acrescentar o dataframe com os dados aqui também)*
  - As próximas camadas são as formas geométricas: `geom_point()`, `geom_line()` etc.
    - » *E, para cada uma, incluímos as definições de como os atributos estéticos representam os dados:* `aes()`
  - As próximas camadas definem os demais aspectos visuais, como o sistema de coordenadas, as escalas dos eixos, títulos, legendas etc.

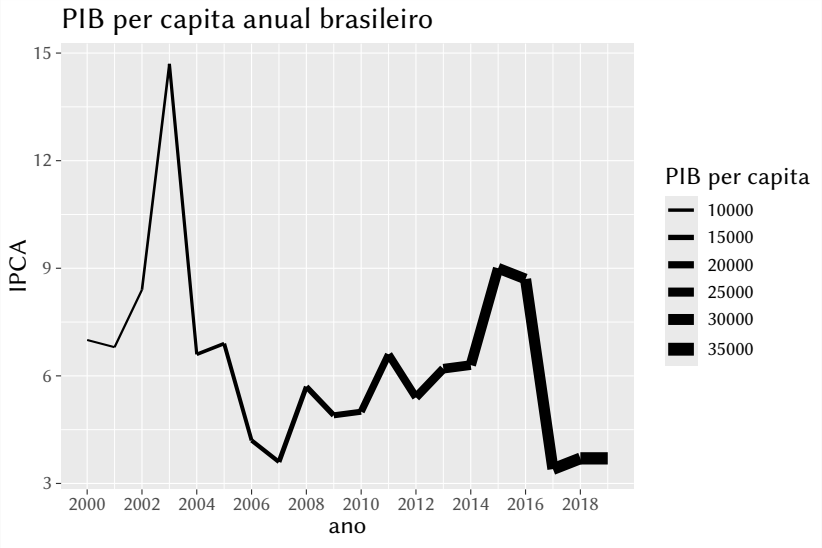
ggplot2

- **Para cada forma geométrica, é preciso definir como o dado é representado por aquela forma**
  - Ou seja, qual atributo estético representa qual dado e de que maneira
  - No caso de linhas, pontos etc., geralmente queremos representar os dados pela posição das linhas/pontos no sistema de coordenadas:
    `geom_line(aes(x=blah, y=bleh))`
  - Mas a espessura e a cor da linha também podem ser usadas para representar algum dado:
    `geom_line(aes(x=blah, y=bleh, color=blih, linewidth=bloh))`

ggplot2

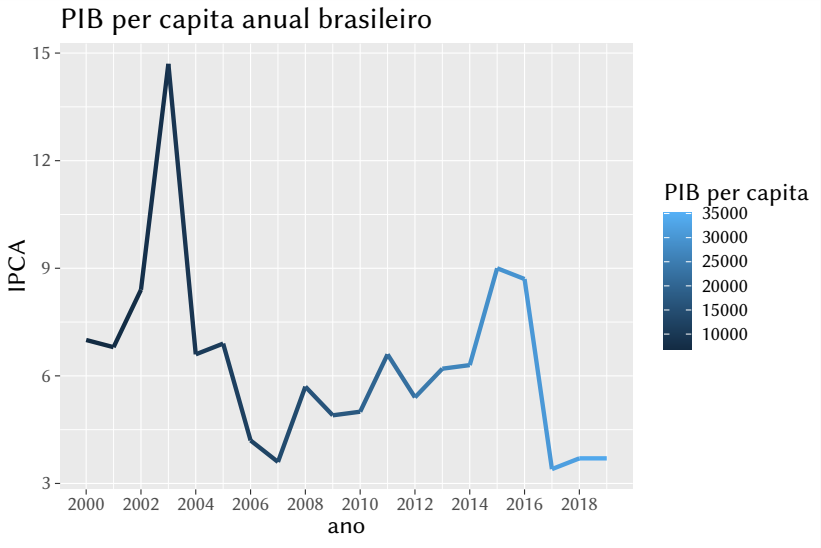
```
library(tidyverse)
endereço <- 'http://www.ime.usp.br/~lago/dadinhos/renda_vs_inflacao.csv'
df <- read_delim(endereço, delim=",", col_types = "idd")
names(df)[2] <- "PIBpc"
p <- ggplot(df) +
  geom_line(aes(x=ano, y=IPCA, linewidth=PIBpc)) +
  labs(title="PIB per capita anual brasileiro",
       linewidth="PIB per capita") +
  scale_x_continuous(breaks=seq(2000, 2019, 2))
print(p)
```

ggplot2


PIB per capita anual brasileiro
IPCA
15
12
9
6
3
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
ano
PIB per capita
10000
15000
20000
25000
30000
35000

ggplot2

```
library(tidyverse)
endereço <- 'http://www.ime.usp.br/~lago/dadinhos/renda_vs_inflacao.csv'
df <- read_delim(endereço, delim=",", col_types = "idd")
names(df)[2] <- "PIBpc"
p <- ggplot(df) +
  geom_line(aes(x=ano, y=IPCA, color=PIBpc), linewidth=2) +
  labs(title="PIB per capita anual brasileiro",
       color="PIB per capita") +
  scale_x_continuous(breaks=seq(2000, 2019, 2))
print(p)
```


PIB per capita anual brasileiro
IPCA
15
12
9
6
3
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
ano
PIB per capita
35000
30000
25000
20000
15000
10000

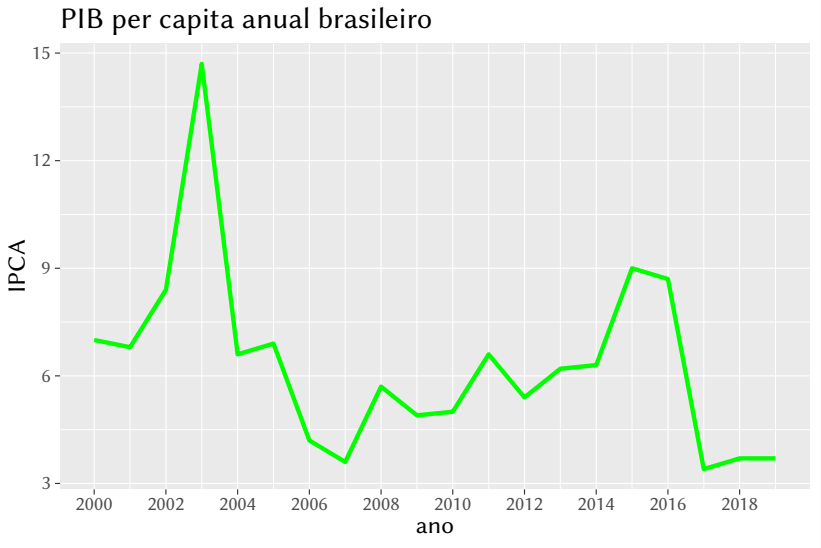
ggplot2

- **Para cada forma geométrica, é preciso definir como o dado é representado por aquela forma**
  - Ou seja, qual atributo estético representa qual dado e de que maneira
  - No caso de linhas, pontos etc., geralmente queremos representar os dados pela posição das linhas/pontos no sistema de coordenadas:
    `geom_line(aes(x=blah, y=bleh))`
  - Mas a espessura e a cor da linha também podem ser usadas para representar algum dado:
    `geom_line(aes(x=blah, y=bleh, color=blih, linewidth=bloh))`
  - Isso é totalmente diferente de
    `geom_line(aes(x=blah, y=bleh), color="green", linewidth=2))`

ggplot2

```
library(tidyverse)
endereço <- 'http://www.ime.usp.br/~lago/dadinhos/renda_vs_inflacao.csv'
df <- read_delim(endereço, delim=",", col_types = "idd")
names(df)[2] <- "PIBpc"
p <- ggplot(df) +
  geom_line(aes(x=ano, y=IPCA), color="green", linewidth=2) +
  labs(title="PIB per capita anual brasileiro") +
  scale_x_continuous(breaks=seq(2000, 2019, 2))
print(p)
```

ggplot2


PIB per capita anual brasileiro
IPCA
15
12
9
6
3
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
ano

ggplot2

- **Para cada forma geométrica, é preciso definir como o dado é representado por aquela forma**
  - Ou seja, qual atributo estético representa qual dado e de que maneira
  - No caso de linhas, pontos etc., geralmente queremos representar os dados pela posição das linhas/pontos no sistema de coordenadas:
    `geom_line(aes(x=blah, y=bleh))`
  - Mas a espessura e a cor da linha também podem ser usadas para representar algum dado:
    `geom_line(aes(x=blah, y=bleh, color=blih, linewidth=bloh))`
  - Isso é totalmente diferente de
    `geom_line(aes(x=blah, y=bleh), color="green", linewidth=2)`
    » *Veja todos os atributos possíveis com* `vignette('ggplot2-specs')`

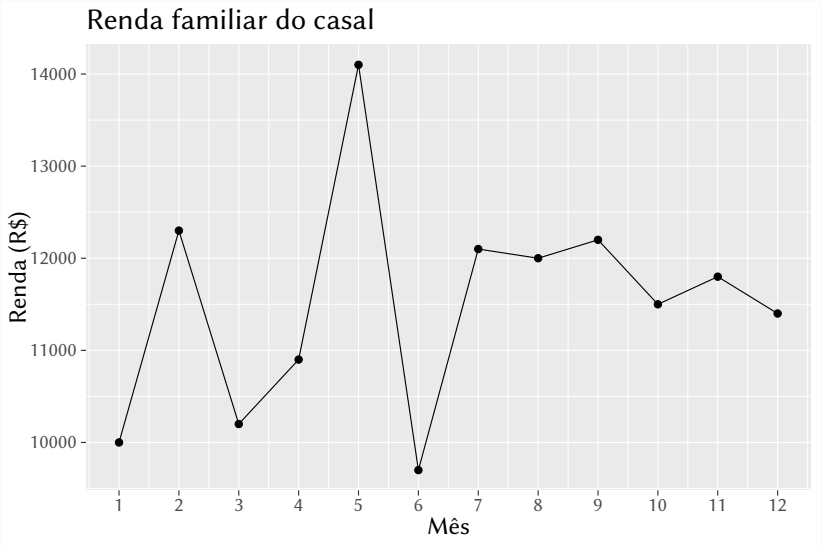
Exercício – funções vetoriais

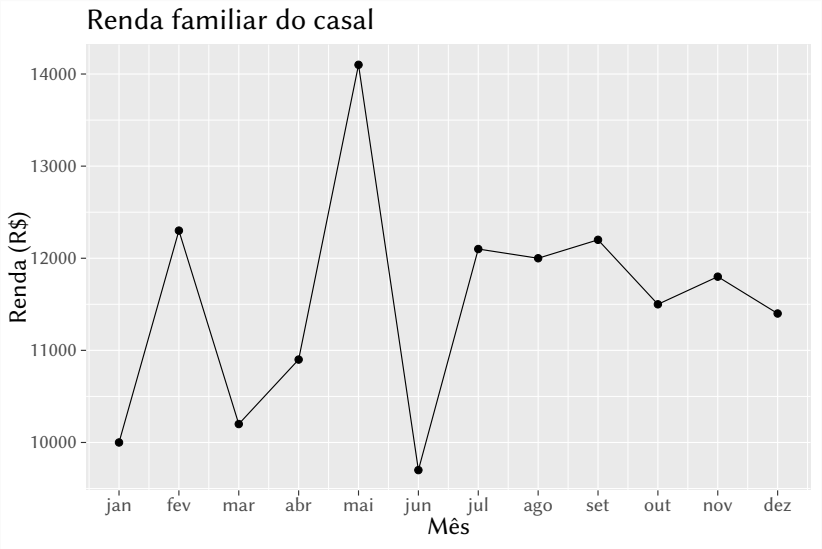
Os rendimentos de cada um dos cônjuges de um casal de *freelancers* no ano passado foi

```
cônjuge1 <- c(4700, 6100, 5300, 5800, 7700, 4600, 5600, 7300, 6900, 6100, 5800, 5800)
cônjuge2 <- c(5300, 6200, 4900, 5100, 6400, 5100, 6500, 4700, 5300, 5400, 6000, 5600)
```

Escreva um programa que calcula a renda familiar mensal e gera o gráfico correspondente

```
library(tidyverse)
meses <- c("jan", "fev", "mar", "abr", "mai", "jun", "jul", "ago", "set", "out", "nov", "dez")
renda <- cônjuge1 + cônjuge2
p <- ggplot(NULL, aes(x=1:12, y=renda)) +
     geom_line() +
     geom_point() +
     labs(title="Renda familiar do casal", x="Mês", y="Renda (R$)") +
     scale_x_continuous(breaks=1:12, labels=meses)
print(p)
```


Renda familiar do casal
Renda (R$)
14000
13000
12000
11000
10000
1
2
3
4
5
6
7
8
9
10
11
12
Mês


Renda familiar do casal
Renda (R$)
14000
13000
12000
11000
10000
jan
fev
mar
abr
mai
jun
jul
ago
set
out
nov
dez
Mês

Outras coisas sobre R e dataframes

- **Para salvar um dataframe em um arquivo:**
  - `write_delim(df, file="blah.csv", delim=",")`
- **Para salvar o último gráfico exibido com ggplot2 em um arquivo:**
  - `ggsave("arquivo.pdf", width=6, height=4)` (polegadas)
- **Para ler diretamente arquivos do excel ou libreoffice:**
  - biliotecas `readxl` e `readODS` (*mas prefira usar CSV!*)
- **Bibliotecas muito úteis e populares:**
  - `purrr` (operações vetoriais com lists/vectors)
  - `dplyr` (operações vetoriais com dataframes/tibbles)
  - `stringr` (operações com textos)
    - » *carregue todas (além de `ggplot2`, `tibble`...) com `library(tidyverse)`*
- **Para acessar a documentação**
  - `?nome-da-função` (sobre a função)
  - `vignette("package")` (sobre a package)
  - `??alguma-coisa` (busca "alguma-coisa" em toda a documentação)

O operador `%in%`

- Como saber se existe um elemento dentro de um vector?
- `if (valor %in% vector) { … }`

Recortes com filtros

- **Dado um dataframe `df` (ou um vector/lista), podemos fazer um "recorte" usando coordenadas: `df[linhas,colunas]`**
  - `linhas` e `colunas` podem ser valores únicos ou vectors
- **Também podemos usar vectors do tipo `LOGICAL`**

```
vec <- 1:5
cat(vec[c(TRUE, FALSE, FALSE, TRUE, FALSE)], "\n")
1 4
grandes <- vec > 3
cat(grandes, "\n")
FALSE FALSE FALSE TRUE TRUE
cat(vec[grandes], "\n")
4 5
cat(vec[vec > 3], "\n")
4 5
```

Recortes com filtros

```r
library(tibble)
df <- tibble(nome=c("Fulano", "Ciclano", "Beltrano"), idade=c(17, 25, 19))
cat("Idade do Fulano:", unlist(df[df$nome == "Fulano","idade"]), "\n")
```

Idade do Fulano: 17

```r
library(tibble)
df <- tibble(nome=c("Fulano", "Ciclano", "Beltrano"), idade=c(17, 25, 19))
cat("Maiores de idade:", unlist(df[df$idade>=18,]), "\n")
maiores <- df[df$idade>=18,]
cat("Maiores de idade:", paste(maiores$nome, ": ", maiores$idade, sep=""), sep="\n")
```

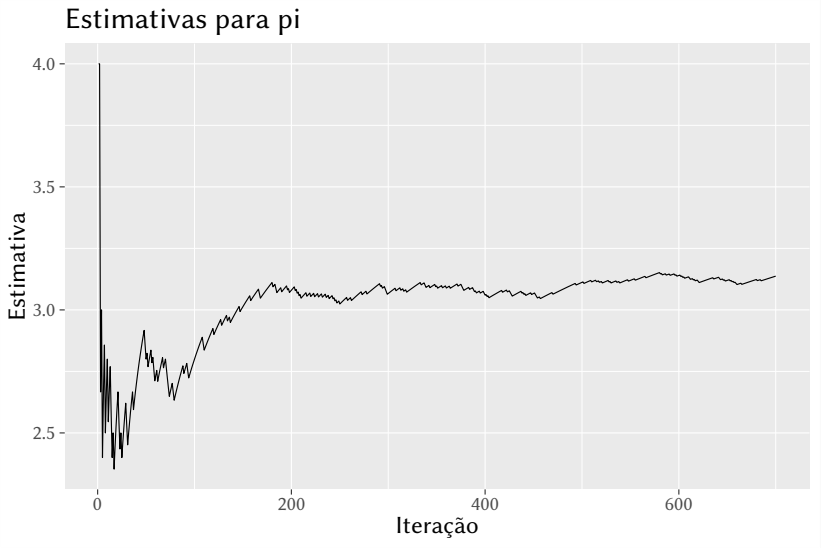
Maiores de idade: Ciclano Beltrano 25 19
Maiores de idade:
Ciclano: 25
Beltrano: 19

And now for more of the same

π e Monte Carlo, parte II

```
main <- function() {
    chutes <- as.integer(readline("Quantas amostras? "))
    dentro <- 0
    estimativas <- numeric(chutes)
    i <- 1
    while (i <= chutes) {
        x <- runif(1, -1, 1)
        y <- runif(1, -1, 1)
        if (x^2 + y^2 <= 1) { # dentro do círculo
            dentro <- dentro + 1
        }
        estimativas[i] <- 4 * dentro / i
        i <- i + 1
    }
    cat("Pi é aproximadamente", 4*dentro/chutes, "\n")
    p <- ggplot(NULL, aes(x=1:chutes, y=estimativas)) +
         geom_line() +
         labs(title="Estimativas para pi", x="Iteração", y="Estimativa")
   print(p)
}
main()
```

π e Monte Carlo, parte II

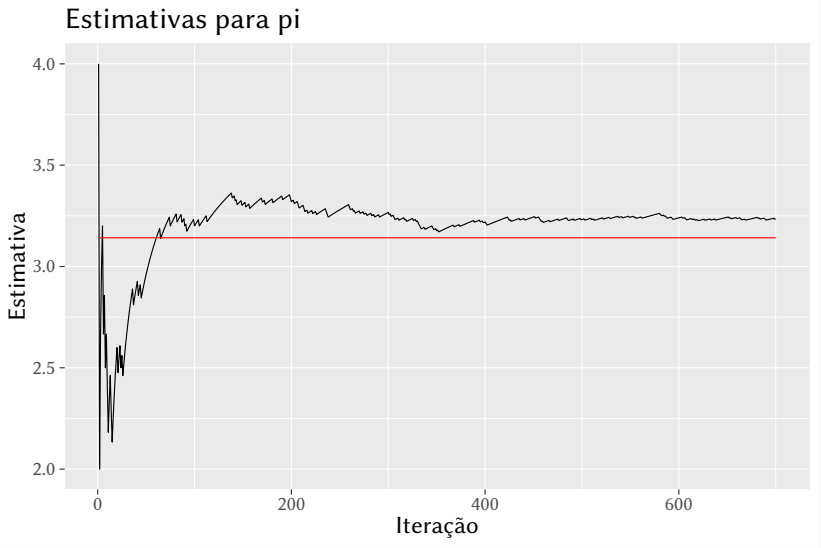


π e Monte Carlo, parte II

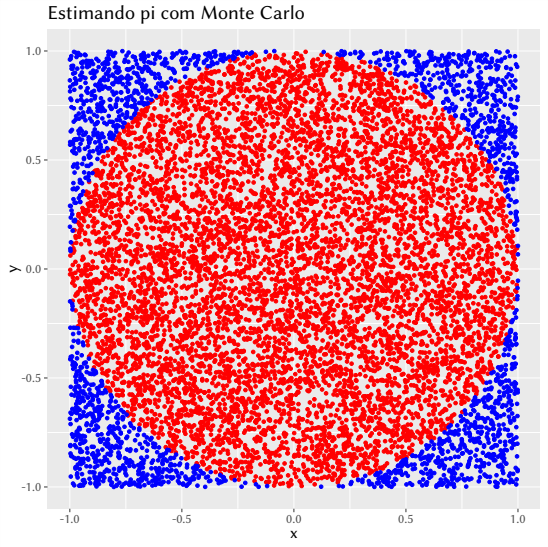
```
main <- function() {
    …

    p <- ggplot(NULL, aes(x=1:chutes, y=estimativas)) +
         geom_line() +
         labs(title="Estimativas para pi", x="Iteração", y="Estimativa") +
         geom_line(aes(x=1:chutes, y=pi), color="red")
   print(p)
}
main()
```

π e Monte Carlo, parte II



π e Monte Carlo, parte III



π e Monte Carlo, parte III

```
chutes <- as.integer(readline("Amostras? "))
x_círculo <- numeric(chutes)
y_círculo <- numeric(chutes)
x_quadrado <- numeric(chutes)
y_quadrado <- numeric(chutes)
dentro <- 0
fora <- 0
i <- 1
while (i <= chutes) {
    x <- runif(1, -1, 1)
    y <- runif(1, -1, 1)
    if (x^2 + y^2 <= 1) {
        dentro <- dentro + 1
        x_círculo[dentro] <- x
        y_círculo[dentro] <- y
    } else {
        fora <- fora + 1
        x_quadrado[fora] <- x
        y_quadrado[fora] <- y
    }
    i <- i + 1
}
```

```
x_círculo <- x_círculo[1:dentro]
y_círculo <- y_círculo[1:dentro]
x_quadrado <- x_quadrado[1:fora]
y_quadrado <- y_quadrado[1:fora]
p <- ggplot(NULL) +
  geom_point(aes(x=x_círculo, y=y_círculo),
             color="red", size=1) +
  geom_point(aes(x=x_quadrado, y=y_quadrado),
             color="blue", size=1) +
       labs(title="Estimativa de pi",
            x="x", y="y")
print(p)
```

π e Monte Carlo, parte IV

```
library(tidyverse)
chutes <- as.integer(readline("Amostras? "))
pontos <- tibble(x=runif(chutes, -1, 1), y=runif(chutes, -1, 1))
dentro <- pontos[pontos$x^2 + pontos$y^2 <= 1,]
fora <- pontos[pontos$x^2 + pontos$y^2 > 1,]
cat("O valor aproximado de pi é", 4 * nrow(dentro)/chutes, "\n")
p <- ggplot(NULL) +
     geom_point(aes(x=dentro$x, y=dentro$y), color="red", size=1) +
     geom_point(aes(x=fora$x, y=fora$y), color="blue", size=1) +
     labs(title="Estimativa de pi", x="x", y="y")
print(p)
```

Exercício — aniversários

```
checa_nivers_repetidos <- function(pessoas) {
    nivers <- runif(pessoas, 1, 365)
    if (length(unique(nivers)) < length(nivers)) { return (TRUE) }
    return (FALSE)
}
```

And now for something completely different

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
2. **Definir como esse problema pode ser solucionado (*algoritmo*)**
   - Sequência finita de passos bem definidos baseados em um conjunto limitado (vocabulário) de operações possíveis
     - » *O algoritmo é abstrato (como a planta de um prédio ou uma receita de bolo)*
3. ***Implementar* o algoritmo em uma linguagem de programação**
   - Gerando um *programa* que pode ser *executado* para solucionar o problema
4. **Testar o programa**

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**
2. **"Faz alguma coisa" com esses dados**
   - Gerando um resultado
3. **"Faz alguma coisa" com esse resultado**
   - Mostra para o usuário
   - **Utiliza como dado para fazer outra coisa**

Mas como criar o algoritmo “certo”?

Algoritmos

1. **Usando ou adaptando um algoritmo que já existe**
2. **Adaptando ideias de outros contextos**
3. **Conhecendo técnicas comuns (ou seja, estudo e prática)**

Busca linear

Escreva uma função que recebe um vector de números `vec` e um número `n` e informa se o número está ou não no vector (sem usar `%in%`)

```
pertence <- function(vec, n) {
    for (val in vec) {
        if (val == n) { return (TRUE) }
    }
    return (FALSE)
}
```

Busca linear

Escreva uma função que recebe um vector ordenado de números `vec` e um número `n` e informa se o número está ou não no vector

```
pertence <- function(vec, n) {
    for (val in vec) {
        if (val > n) { return (FALSE) }
        else if (val == n) { return (TRUE) }
    }
    return (FALSE)
}
```

Mas é assim que procuramos uma palavra em um dicionário?!?!

Busca binária

Quando procuramos em um dicionário:

- **Abrimos o dicionário "em algum lugar perto do meio"**
  - Ih, é antes ⟶ abre "em algum lugar do meio" entre o começo e a página atual
  - Ih, é depois ⟶ abre "em algum lugar do meio" entre a página atual e o fim
- **No caso do dicionário, sabemos o mínimo ("a") e o máximo ("z") da lista, então não abrimos "no meio", mas tentamos "chutar" uma página próxima**
- **Numa lista de números genérica, não temos essa "dica", então o melhor é olhar o "meio" mesmo**

Busca binária

Vamos procurar pelo número 23

Busca binária

Escreva uma função que recebe um vector ordenado de números vec e um número n e informa se o número está ou não no vector

```
pertence <- function(vec, n) {
    while (length(vec) > 1) {
        meio <- length(vec) %/% 2
        if (n == vec[meio]) { return (TRUE) }
        if (n < vec[meio]) { resto <- seq_len(meio - 1) }
        else { resto <- seq_len(length(vec) - meio) + meio }
        vec <- vec[resto]
    }
    if (length(vec) == 1) { return (n == vec[1]) }
    return (FALSE)
}
```

Exercício — adivinhando números 3/4

Escreva um programa que tenta adivinhar um número de 1 a 10 escolhido pelo usuário: o usuário responde “<”, “>” ou “=” para indicar se o número correto é maior, menor ou igual ao número gerado pelo computador (repetições até atingir um resultado)

```
library(glue)
main <- function() {
    cat("Escolha um número de 1 a 10, vou tentar adivinhá-lo!", "\n")
    resposta <- "x" # qualquer coisa diferente de "="
    while (resposta != "=") {
        chute <- sample(1:10, 1)
        resposta <- readline(glue('Será {chute}? responda com "<", ">" ou "=" '))
    }
    cat("Aêh, sou vidente!", "\n")
}
main()
```

Exercício — adivinhando números 4/4

Melhore o programa anterior, fazendo uso das informações de maior/menor fornecidas pelo usuário

```
cat("Escolha um número de 1 a 10, vou tentar adivinhá-lo!", "\n")
resposta <- "x" # qualquer coisa diferente de "="
menor <- 1
maior <- 10
while (resposta != "=") {
    chute <- sample(menor:maior, 1)
    resposta <- readline(glue('Será {chute}?; responda com "<", ">" ou "=" '))
    if (resposta == ">") {
        menor <- chute + 1
    }
    if (resposta == "<") {
        maior <- chute - 1
    }
}
cat("Aêh, sou vidente!", "\n")
```

Divisão e conquista

Divisão e conquista

Você sabe somar mais de dois números de uma vez?

$$1 + 2 + 3 + 4 = (1 + 2) + 3 + 4 = 3 + 3 + 4 = (3 + 3) + 4 = 6 + 4 = 10$$

Imagine que o computador só é capaz de fazer uma soma de cada vez; faça uma função que soma uma lista de números

```
soma <- function(v) {
    s <- 0
    for (n in v) { s <- s + n }
    return (s)
}
```

Pega um por um e soma

(como no exemplo $1 + 2 + 3 + 4$ acima)

Divisão e conquista

Mas podemos pensar um pouco diferente:

```
somatório <- function(l) {
    if (length(l) == 0) { return (0) }
    if (length(l) == 1) { return (l[1]) }
    if (length(l) == 2) { return (l[1] + l[2]) }
    if (length(l) >= 3) { return (l[1] + somatório(l[2:length(l)])) }
}
```

$$1 + 2 + 3 + 4 = 1 + (2 + 3 + 4) = 1 + (2 + (3 + 4)) = 1 + (2 + 7) = 1 + 9 = 10$$

A soma de n números é a soma do primeiro com a soma do restante

Divisão e conquista

```
somatório <- function(l) {
    if (length(l) == 0) { return (0) }
    resto <- seq_len(length(l) - 1) + 1
    return (l[1] + somatório(l[resto]))
}
```

```
somatório <- function (l) {
    resto <- seq_len(length(l) - 1) + 1
    return (l[1] + somatório(l[resto]))
}
```

- **Não dá para fazer chamadas recursivas para sempre!**
  - É preciso haver ao menos um caso em que **não** fazemos a chamada recursiva
  - É preciso garantir que **sempre** vamos chegar nesse caso