

MAC 113 — Introdução à Ciência da Computação

Aula 18

Nelson Lago

1º/2025



Previously on MAC113...

Funções – escopo

- Funções são **nomes** dados a trechos de código que representam uma **ideia** bem definida e auto-contida
- Uma das vantagens é que elas podem ser usadas em contextos diferentes para realizar a computação correspondente àquela ideia
- Portanto...
- Elas precisam ser capazes de funcionar de maneira independente do contexto

**Cada um com seu cada um e ninguém se mete
no cada um dos outros**

Funções – escopo

```
viés <- 3
fatorial_enviesado <- function(n) {
  fat <- 1
  cat("viés anterior:", viés, "\n")
  viés <- 5
  while (n >= 2) {
    fat <- fat * n
    n <- n - 1
  }
  return (fat + viés)
}
cat(fatorial_enviesado(4), "\n")
cat("viés final:", viés, "\n")
```

Cada um com seu cada um e ninguém se mete
no cada um dos outros
(só às vezes) 🙄

- **Às vezes significa:**

- ▶ **Para ler:** quando a variável não existe no contexto da função, mas existe no contexto global
- ▶ **Para modificar:** quando fazemos uma atribuição com `<<-`

Funções – `main()`

```
main <- function() {  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  cat(fatorial(x), "\n")  
}  
  
fatorial <- function(n) {  
  fat <- 1  
  while (n >= 2) {  
    fat <- fat * n  
    n <- n - 1  
  }  
  return (fat)  
}  
  
main()
```


Embora não seja obrigatório, em geral é uma boa ideia usar uma função `main()`

Escopo no google colab

- O google colab (e o R Studio) é composto por duas partes:
 - 1 Um “editor de texto turbinado”
 - 2 Um ambiente para a execução do programa criado nele
- Cada célula de código é um pedaço de **um mesmo** programa
 - ▶ O ambiente de execução de todas as células é o mesmo
- Quando você inicia uma sessão com um documento que já existe, seu programa está no editor de texto como você deixou
- **MAS**
- O ambiente de execução está “limpo”
 - ▶ Nenhuma função foi definida ou chamada, nenhuma variável foi definida...
- Ao rodar uma célula, o estado do ambiente de execução “incorpora” o que está nela
 - ▶ **Apenas nesta sessão!**
- Variáveis e funções definidas em uma célula “valem” em outras células **se você executá-las primeiro**

Condições mutuamente excludentes

```
if (delta < 0) {  
    cat("não há raízes reais", "\n")  
}  
if (delta == 0) {  
    ...  
    cat("A raiz dupla é", raiz, "\n")  
}  
if (delta > 0) {  
    ...  
    cat(glue("As raízes são {raiz1} e {raiz2}"), "\n")  
}
```

- **Tratamos todos os casos corretamente, mas não deixamos claro que os três casos são mutuamente excludentes (um e apenas um dos casos é executado)**

Condições mutuamente excludentes — else if

```
if (delta < 0) {  
    cat("não há raízes reais", "\n")  
} else if (delta == 0) {  
  
    ...  
    cat("A raiz dupla é", raiz, "\n")  
} else {  
  
    ...  
    cat(glue("As raízes são {raiz1} e {raiz2}"), "\n")  
}
```

- A indentação deixa mais claro que, na verdade, são casos do mesmo “nível” e mutuamente excludentes (**um e apenas um** dos casos é executado)
 - ▶ Mas, na verdade, esse código equivale exatamente ao anterior!

else e else if

- Tudo que pode ser feito com **else** e **else if** pode ser feito com uma sequência de **if**'s
- Então pra que serve essa bagaça?!
 - ❶ Não é preciso reescrever condições redundantes
 - ❷ Deixa mais claro que as condições são mutuamente excludentes
- Cada **if** ou **else if** funciona “excluindo” uma parte das possibilidades
- Os **if**'s ou **else if**'s seguintes se aplicam apenas ao universo de possibilidades que ainda não foram excluídas
- O **else** final pega tudo o que “sobrou”

A ordem faz diferença!

Exercício

- No Brasil, a idade mínima para trabalhar é 16 anos
- A partir dos 14 anos, é possível trabalhar apenas como jovem aprendiz
- O trabalho noturno só é permitido para maiores de 18 anos

```
idade <- as.integer(readline("Qual sua idade? "))
if (idade >= 18) {
  cat("Você pode trabalhar em qualquer atividade", "\n")
} else if (idade >= 16) {
  cat("Você não pode trabalhar no período noturno", "\n")
} else if (idade >= 14) {
  cat("Você só pode trabalhar como jovem aprendiz", "\n")
} else {
  cat("Vai estudar, moleque!", "\n")
}
```

- As condições “já vistas” ficam implícitas
 - A ordem faz diferença

O cursor

```
cat("Olá!", "\n")  
cat("Tudo bem?", "\n")
```

Olá bem?

- A tela tem um **cursor**: o lugar em que vai aparecer a próxima coisa que for impressa
 - ▶ “O lugar em que está a ponta do lápis”
- **cat()** escreve a mensagem e deixa o cursor no final do texto que foi impresso
- **"\n"** move o cursor para o início da linha seguinte
 - ▶ `cat("blah", "\n")`
- **"\n"** é um texto como qualquer outro
 - ▶ `cat("blah", "\n")`
 - ▶ `cat("blah\n")`

Brincando com o cursor

- **cat()** não separa os itens com um espaço, mas sim com o que é definido por **sep**
 - ▶ `cat("nome", "RG", "CPF", sep="|")`
» *nome|RG|CPF*
- se você não definir **sep**, R utiliza um espaço

“São 14:16:02h”

```
horas <- "14"  
minutos <- "16"  
segundos <- "02"  
cat("São ")  
cat(horas, minutos, segundos, sep=":")  
cat("h\n")
```


Repetições encaixadas

- Repetições usam uma variável de controle
- Quando essa variável corresponde a um conjunto, as repetições permitem trabalhar sobre os elementos desse conjunto
 - ▶ De maneira unidimensional!
 - ▶ Números naturais, nomes de pessoas, notas de alunos...
- Como trabalhar com conjuntos de maneira bidimensional?
 - ▶ Pontos de um plano, tabelas...
- Repetições encaixadas

Repetições encaixadas

```
while (condição1) {  
    while (condição2) {  
        ...  
    }  
}
```

- A cada “rodada” do laço mais externo, o laço interno é executado várias vezes (até sua condição deixar de ser verdadeira)
- Para isso funcionar, condição2 deve voltar a ser verdadeira a cada nova rodada do laço mais externo
- **Dois variáveis de controle**
 - ▶ Um olho no peixe, outro no gato 🐟🐱

Exemplo — tabuada

```
tabuada_do <- 1
while (tabuada_do <= 10) {
  n <- 1
  while (n <= 10) {
    cat(tabuada_do * n, " ")
    n <- n + 1
  }
  cat("\n")
  tabuada_do <- tabuada_do + 1
}
```

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

Exemplo — tabuada

```
tabuada_do <- 1
while (tabuada_do <= 10) {
  n <- 1
  while (n <= 10) {
    cat(format(tabuada_do * n, width=3), " ")
    n <- n + 1
  }
  cat("\n")
  tabuada_do <- tabuada_do + 1
}
```



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100



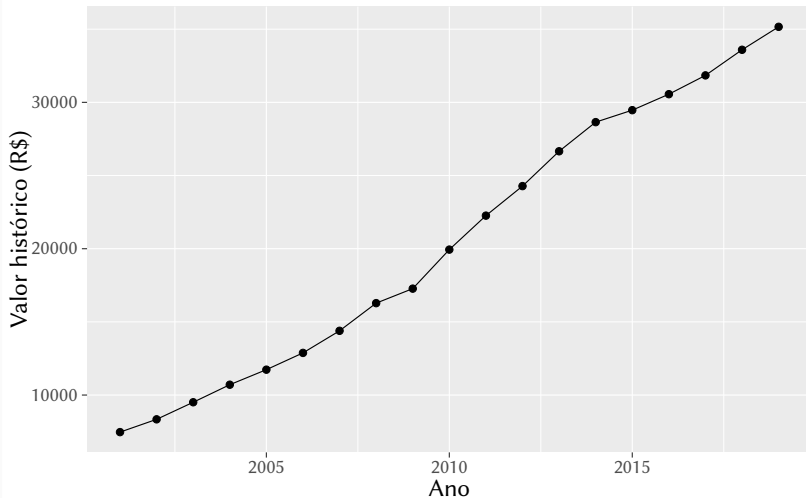
PIB per capita anual

- Gráfico com o PIB per capita anual brasileiro (2000 a 2019)

```
main <- function() {  
  tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/anos-pib.csv")  
  anos <- tmp[[1]]  
  tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/pib-per-capita.csv")  
  pib <- tmp[[1]]  
  p <- ggplot(NULL, aes(x=anos, y=pib)) +  
    geom_line() +  
    geom_point() +  
    labs(title="PIB per capita anual brasileiro", x="Ano",  
          y="Valor histórico (R$)")  
  print(p)  
}  
main()
```

PIB per capita anual

PIB per capita anual brasileiro

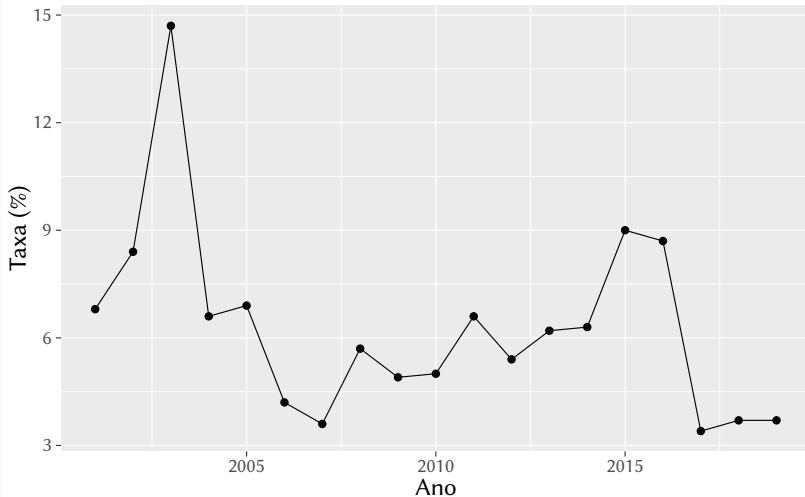


- Gráfico com o IPCA anual brasileiro (2000 a 2019)

```
main <- function() {  
  tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/anos-pib.csv")  
  anos <- tmp[[1]]  
  tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/ipca-anual.csv")  
  ipca <- tmp[[1]]  
  p <- ggplot(NULL, aes(x=anos, y=ipca)) +  
    geom_line() +  
    geom_point() +  
    labs(title="IPCA anual brasileiro", x="Ano",  
          y="Taxa (%)")  
  print(p)  
}  
main()
```

PIB per capita anual

IPCA anual brasileiro



PIB per capita anual corrigido

- Gráfico com o PIB per capita anual brasileiro (2000 a 2019) **com valores corrigidos (equivalentes a reais em 2020)**
 - ▶ Considere que a inflação “acontece” de uma só vez, em 31 de dezembro

```
main <- function() {  
  tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/anos-pib.csv")  
  anos <- tmp[[1]]  
  tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/pib-per-capita.csv")  
  pib <- tmp[[1]]  
  tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/ipca-anual.csv")  
  ipca <- tmp[[1]]  
  pib <- pib_atualizado(pib, ipca)  
  p <- ggplot(NULL, aes(x=anos, y=pib)) +  
    geom_line() +  
    geom_point() +  
    labs(title="PIB per capita anual brasileiro", x="Ano",  
          y="Valor histórico atualizado (R$ em 2020)")  
  print(p)  
}
```

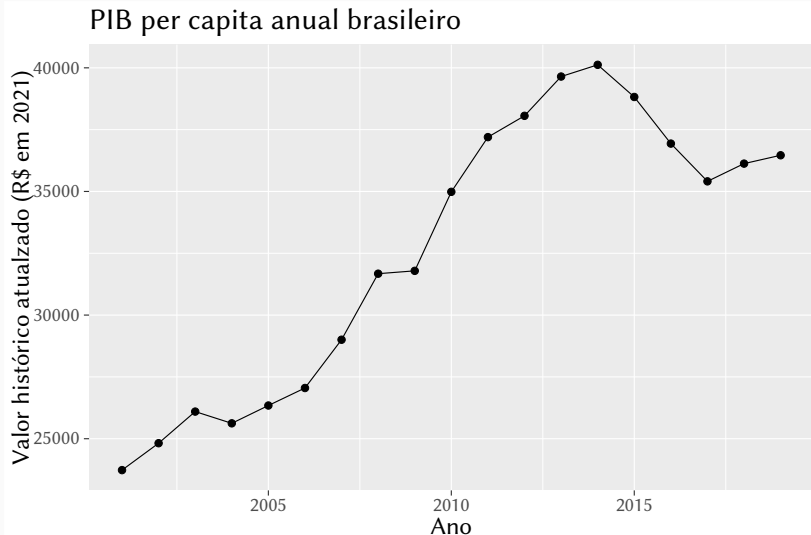
PIB per capita anual corrigido

- Gráfico com o PIB per capita anual brasileiro (2000 a 2019) **com valores corrigidos (equivalentes a reais em 2020)**
 - ▶ Considere que a inflação “acontece” de uma só vez, em 31 de dezembro

```
main <- function() {  
  ...  
}  
pib_atualizado <- function(pib, ipca) {  
  ipca <- ipca_acumulado(ipca)  
  i <- 1  
  while (i <= length(pib)) {  
    pib[i] <- pib[i] * ipca[i]  
    i <- i + 1  
  }  
  return(pib)  
}
```

```
ipca_acumulado <- function(ipca) {  
  i <- 1  
  while (i <= length(ipca)) {  
    ipca[i] <- (100 + ipca[i]) / 100  
    i <- i + 1  
  }  
  i <- 1  
  while (i <= length(ipca)) {  
    j <- i + 1  
    while (j <= length(ipca)) {  
      ipca[i] <- ipca[i] * ipca[j]  
      j <- j + 1  
    }  
    i <- i + 1  
  }  
  return(ipca)  
}
```

PIB per capita anual



And now for something completely different

Tipos de repetição

- **Dois tipos fundamentais de repetição**

- ① Repetições até atingir um resultado

- » *Encontrar o próximo primo*
 - » *Reiniciar o jogo até o usuário escolher “sair”*
 - » ...

- ② Repetições sobre os elementos de um conjunto

- » *Apresentar todos os pixels de uma foto na tela*
 - » *Trocar todas as letras de um texto para maiúsculas*
 - » ...

Repetições com coleções

```
primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
i <- 1
while (i < length(primos)) {
  cat("O número", primos[i], "é primo", "\n")
  i <- i + 1
}
```

- **i é a variável de controle do laço**
- **Mas! Não usamos o valor de i “de verdade”**
 - ▶ Só usamos i para acessar os elementos do vetor

```
primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
for (p in primos) {
  cat("O número", p, "é primo", "\n")
}
```

- **outro jeito de fazer repetições sobre os elementos de um conjunto**

Repetições com coleções

```
primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
for (p in primos) {
  cat("O número", p, "é primo", "\n")
}
```

- **Um laço correto precisa**

- ▶ Inicializar a variável de controle antes do início do laço
- ▶ Verificar a condição adequada a cada iteração para que as repetições aconteçam o número correto de vezes
- ▶ Alterar o valor da variável de acordo com a lógica do programa (no mínimo, na última iteração) para garantir que o laço termine

- **Os laços com for “escondem” esses passos**

- ▶ (R faz automaticamente para você)

Repetições com coleções

```
primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
for (p in primos) {
  cat("O número", p, "é primo", "\n")
}
```

```
primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
i <- 1
while (i <= length(primos)) {
  p <- primos[i]
  cat("O número", p, "é primo", "\n")
  i <- i + 1
}
```


Repetições com coleções

```
cat("Prepare-se para o grito:", "\n")
n <- 10
while (n > 0) {
  cat(n, "\n")
  n <- n - 1
}
cat("AAAH!!", "\n")
```

- Repetições sobre o conjunto dos números 10–1
- Podemos criar um vector com esses números!

```
cat("Prepare-se para o grito:", "\n")
for (n in 10:1) {
  cat(n, "\n")
}
cat("AAAH!!", "\n")
```

**Tipos de laços diferentes “combinam melhor”
com sintaxes diferentes**

Tipos de repetição

- Quando a quantidade de repetições é desconhecida, **while** é uma boa escolha:
 - ▶ **while** (! achei)
 - ▶ **while** (encontrados < 10)
 - ▶ **while** (usuárioQuerJogar)
- Quando queremos manipular os elementos de uma coleção, **for** (... in ...) é uma boa escolha:
 - ▶ **for** (p in primos)
 - ▶ **for** (canção in canções)
- Quando queremos manipular uma lista pré-definida de números *ou* os *índices* dos elementos de uma coleção, **for** (n in X:Y) é uma boa escolha:
 - ▶ **for** (n in 1:10)
 - ▶ **for** (n 1:length(meu_vec))

Repetições com coleções

- Muitas vezes, estamos interessados apenas nos elementos do conjunto

```
primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
for (p in primos) {
  cat("O número", p, "é primo", "\n")
}
```

- Mas às vezes estamos interessados na posição (índice) dos elementos também

```
primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
for (i in 1:length(primos)) {
  cat(glue("O {i}o primo é {primos[i]}"), "\n")
}
```

Rápido como uma tartaruga

A biblioteca **TurtleGraphics** permite desenhar figuras simples, mas infelizmente tem problemas de compatibilidade com o colab. É possível contornar esse problema com isto:

```
install.packages("TurtleGraphics")
library("TurtleGraphics")
fakedraw <- function() {invisible(NULL)}
environment(fakedraw) <- asNamespace("TurtleGraphics")
assignInNamespace(".turtle_draw", fakedraw, ns="TurtleGraphics")
assignInNamespace(".turtle_undraw", fakedraw, ns="TurtleGraphics")
```

Desenhando um quadrado colorido (com for)

Modifique o programa abaixo para desenhar um quadrado com os lados coloridos usando for

```
main <- function() {  
  turtle_init(300,300)  
  cores <- c("red", "green", "blue", "yellow")  
  
  for (cor in cores) {  
    turtle_col(cor)  
    turtle_forward(100)  
    turtle_right(90)  
  }  
}  
main()
```

Desenhando uma estrela (com for)

Modifique o programa abaixo para desenhar uma estrela de cinco pontas usando for

```
main <- function() {  
  turtle_init(300,300)  
  turtle_right(90)  
  
  for (i in 1:5) {  
    turtle_forward(100)  
    turtle_right(144)  
  
  }  
}  
main()
```

Exercício — elefantes

Escreva um programa que lê um inteiro positivo n e imprime a letra de “Um elefante incomoda muita gente...” n vezes

```
main <- function() {  
  n <- as.integer(readline("Quantas vezes? "))  
  if (n < 1) { return }  
  cat("Um elefante incomoda muita gente\n")  
  cat("2 elefantes incomodam incomodam muito mais\n")  
  if (n < 2) { return }  
  incomodam <- c("incomodam", "incomodam")  
  for (i in 2:n) {  
    cat(i, "elefantes", incomodam, "muita gente\n")  
    incomodam <- c("incomodam", incomodam)  
    cat(i + 1, "elefantes", incomodam, "muito mais\n")  
  }  
}
```

`main()`