MAC 113 — Introdução à Ciência da Computação

Aula 12

Nelson Lago

1°/2025

Previously on MAC11X...

Indentação

```
main <- function() {
    …
    while (condição) {
        …
        if (condição) {
            …
        } else {
            …
        }
        …
    }
    …
}
```

- A **condição** do `if`/`while` fica entre parênteses
- O **código** a ser executado ou não fica entre chaves
- Os **parâmetros** das funções ficam entre parênteses
- O **código** das funções fica entre chaves
- Em geral, o que vai entre parênteses é "curto", mas o que vai entre chaves pode ser bastante longo (várias linhas)
- É muito fácil para um humano se confundir com as chaves!
- A solução é a **indentação**: qualquer código entre chaves é deslocado para a direita
  - Qualquer tamanho, mas uma boa opção para **R** são 4 espaços

Cuidado com o amigo da onça!

- **Talvez você tenha percebido que às vezes é possível omitir `return()` ...**
- **Talvez você tenha percebido que às vezes é possível omitir as chaves com `if` / `else` / `while` ...**
- **Talvez você tenha percebido que às vezes é possível omitir o `"\n"` no final do comando `cat()` ...**

Cuidado!

Não caia em tentação!

Quando você menos esperar, isso vai causar problemas!

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

Nomes e memória

- **Em R, só existe quem tem nome:**

```
saudação <- "Olá!"
saudação <- "Oi!"
```

- **O texto `"Olá!"` não existe mais em nenhum lugar da memória após a segunda linha ser executada!**

```
x <- 1
x <- x + 1
```

- **O número `1` não existe mais em nenhum lugar da memória após a segunda linha ser executada!**

Nomes e memória

- **Em R, só existe quem tem nome:**

```
x <- 1
y <- x
x <- x + 1
```

- **O número `1` *continua existindo* na memória após a terceira linha ser executada**
  - Mas o que acontece na segunda linha?!?!?
    - » *Temos duas "cópias" do número `1` na memória?*
- **DEPENDE**
  - Em R, sim!
  - Na terceira linha, voltamos a ter apenas uma cópia
    - » *(R na verdade evita copiar quando não é necessário, mas do ponto de vista do programador é "como se" sempre houvesse a cópia)*

Variáveis e memória

O que são “variáveis”?

- **Nomes que usamos para acessar dados armazenados em algum lugar na memória do computador?**
- **Conceitos abstratos que representam uma informação relevante para a computação que estamos realizando?**

Ambos

- **Mas cada linguagem dá mais ênfase a este ou àquele ponto de vista**

Variáveis e memória

- **Em python**
  - `x = x + 1` não altera o número `1` armazenado na memória, mas sim o valor associado à variável `x` (o número `2` "nasce" e recebe o nome `x`, enquanto o número `1` é "jogado fora")
    - » *Pensando em variáveis como conceitos, isso faz sentido: o valor da variável é o conceito, e ele foi modificado; o número armazenado é* ***outro*** *número*
- **Em outras linguagens, como R e C, as coisas são diferentes:**
  - `x <- y` guarda duas "cópias" do mesmo número em lugares diferentes da memória
  - `x <- x + 1` procura o lugar na memória em que a variável `x` está armazenada e altera o valor que está ali
    - » *Pensando em variáveis como nomes para um dado na memória, isso faz sentido: o conteúdo daquele espaço de memória foi modificado*

Escopo

E por que eu preciso me preocupar com isso?

```
metade <- function(x) {
    x <- x / 2
    return (x)
}
x <- 10
cat(metade(x), "é a metade de", x, "\n")
```

- **Na chamada a `metade()`, o número `10`, que está armazenado em `val`, é copiado para o nome `x`**
  - O nome `x` só existe dentro da função `metade()`!
  - Quando a função termina, o nome `x` e seu valor deixam de existir
  - Quando o valor de `x` é modificado dentro da função, isso não altera nenhuma variável fora da função
    - » *E isso continua valendo mesmo que, "por coincidência", o nome "de fora" e o nome "de dentro" sejam iguais*

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

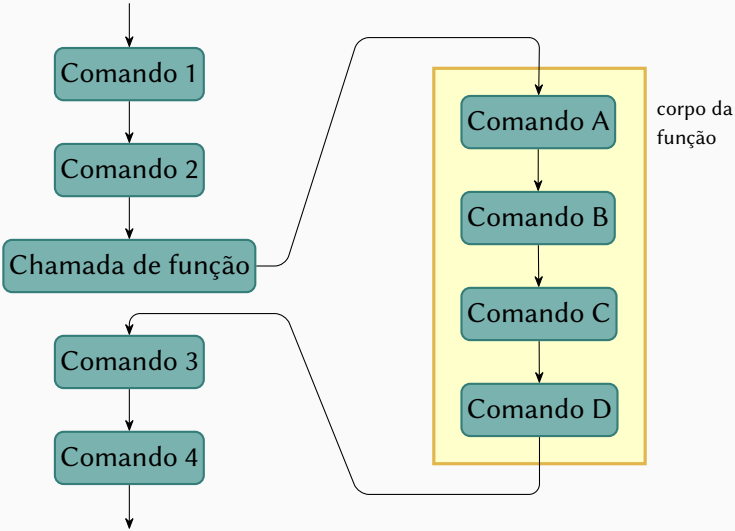
Funções

- **Não confunda `cat()` e `return`!**
  - `cat()` e `readline()` ⟶ comunicação com o "mundo"
  - `return` e os parâmetros das funções ⟶ comunicação entre partes diferentes do programa

```
metade <- function(x) {
    x <- x / 2
    return (x)
}
val <- 10
cat(metade(val), "é a metade de", val, "\n")
```

```
metade <- function(x) {
    y <- x / 2
    cat(y, "é a metade de", x, "\n")
}
metade(10)
```

Funções são “desvios”



Coleções

Há três tipos principais de coleções:

- **Listas de coisas "iguais"**
  - Nomes de pacientes
  - Salários de funcionários
  - PIB anual
  - Objetos para comprar
- **Composição de dados formando uma abstração**

  Pessoa: nome, endereço, estado civil, CPF...
  Pessoa: nome, colesterol, cor do cabelo...

  País: capital, PIB per capita, área cultivável...
  Personagem: inteligência, força, destreza...
- **Tabelas (mistura dos dois)**

Coleções

A primeira coleção que vamos ver é o vector:

```
cores <- c("vermelho", "azul", "amarelo")
```

Um vector é um conjunto ordenado:

```
cat(cores[1], "\n")
cat(cores[3], "\n")
```

```
vermelho
amarelo
```

- **Todos os elementos de um vector precisam ser do mesmo tipo**
  - Usamos vectors para representar coleções de coisas "iguais"

Exercício — sistema de *login*

```
main <- function() {
    uid <- readline("username: ")
    senha <- readline("senha: ")
    nome <- checa_login(uid, senha)
    cat(saudação(nome), "\n")
}
```

```
checa_login <- function(uid, senha) {
    if (uid == "turing" && senha == "tmachine")
        { return("Alan Turing") }
    if (uid == "llace" && senha == "anengine")
        { return("Ada Lovelace") }
    if (uid == "hopper" && senha == "business")
        { return("Grace Hopper") }
    return ("")
}
```

```
saudação <- function(nome) {
    if (nome == "") { return("Login ou senha incorreto") }
    hora <- as.integer(format(Sys.time(), format="%H"))
    if (hora >= 6 && hora < 13)
        { return(glue("Bom dia, {nome}!")) }
    if (hora > 13 && hora < 19)
        { return(glue("Boa tarde, {nome}!")) }
    if (hora >= 19 || hora < 6)
        { return(glue("Boa noite, {nome}!")) }
}
```

Exercício – sistema de *login*

- **Para acrescentarmos mais usuários, é preciso mexer na sequência de comandos "`if`"**
  - Isso não parece muito esperto!
- **Vamos usar vectors!**

```
main <- function() {
    logins <- c("turing", "llace", "hopper")
    senhas <- c("tmachine", "anengine", "business")
    nomes <- c("Alan Turing", "Ada Lovelace", "Grace Hopper")
    uid <- readline("username: ")
    senha <- readline("senha: ")
    nome <- checa_login(uid, senha, logins, senhas, nomes)
    cat(saudação(nome), "\n")
}
```

Exercício – sistema de *login*

```
checa_login <- function(uid, senha, logins, senhas, nomes) {
    i <- 1
    while (i <= length(senhas)) {
        if (uid == logins[i] && senha == senhas[i]) {
            return(nomes[i])
        }
        i <- i + 1
    }
    return("")
}
```

Mostrando dados

Faça um programa que imprime os 7 primeiros quadrados perfeitos

```
Os primeiros quadrados perfeitos:
1
4
9
16
25
36
49
```

```
main <- function() {
    quadrados <- integer(7)
    i <- 1
    while (i <= 7) {
        quadrados[i] <- i^2
        i <- i + 1
    }
    cat("Os primeiros quadrados perfeitos: ", "\n")
    i <- 1
    while (i <= length(quadrados)) {
        cat(quadrados[i], "\n")
        i <- i + 1
    }
}
main()
```

Mostrando dados

Faça um programa que imprime os 7 primeiros quadrados perfeitos

```
Os primeiros quadrados perfeitos:
1 4 9 16 25 36 49
```

```
main <- function() {
    quadrados <- integer(7)
    i <- 1
    while (i <= 7) {
        quadrados[i] <- i^2
        i <- i + 1
    }
    cat("Os primeiros quadrados perfeitos: ", "\n")
    cat(quadrados, "\n")
}
main()
```

Mostrando dados – ggplot2

```
library(ggplot2)
main <- function() {

    quadrados <- integer(7)
    i <- 1
    while (i <= 7) {
        quadrados[i] <- i^2
        i <- i + 1
    }
    p <- ggplot(NULL, aes(x=1:7 , y=quadrados)) +
         geom_line() +
         geom_point() +
         labs(title="Primeiros quadrados perfeitos", x="x", y="x^2")
    print(p)
}
main()
```

Mostrando dados – ggplot2

Lendo dados

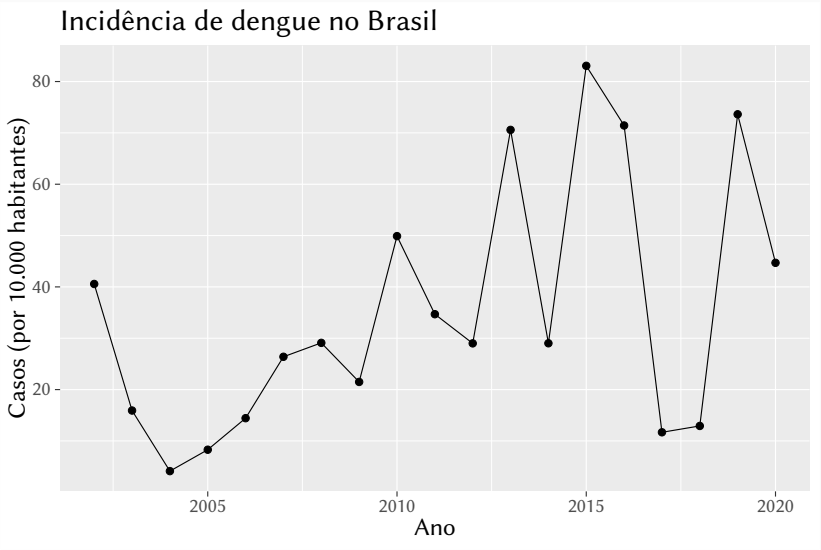
```
main <- function() {
    tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/casos-dengue.csv")
    casos <- tmp[[1]] # transforma em vector

    maior <- casos[1]
    i <- 1
    while (i <= length(casos)) {
        if (casos[i] > maior) {
            maior <- casos[i]
        }
        i <- i + 1
    }
    cat("O maior número de casos em um ano foi", maior, "por dez mil habitantes")
}
main()
```

Tudo em todo lugar ao mesmo tempo

```
main <- function() {
  tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/casos-dengue.csv")
  casos <- tmp[[1]]
  tmp <- read.csv("http://www.ime.usp.br/~lago/dadinhos/anos-dengue.csv")
  anos <- tmp[[1]]
  p <- ggplot(NULL, aes(x=anos, y=casos)) +
       geom_line() +
       geom_point() +
       labs(title="Incidência de dengue no Brasil", x="Ano",
            y="Casos (por 10.000 habitantes)")
  print(p)
}
main()
```

matplotlib


Incidência de dengue no Brasil
Casos (por 10.000 habitantes)
80
60
40
20
2005
2010
2015
2020
Ano