MAC 113 — Introdução à Ciência da Computação

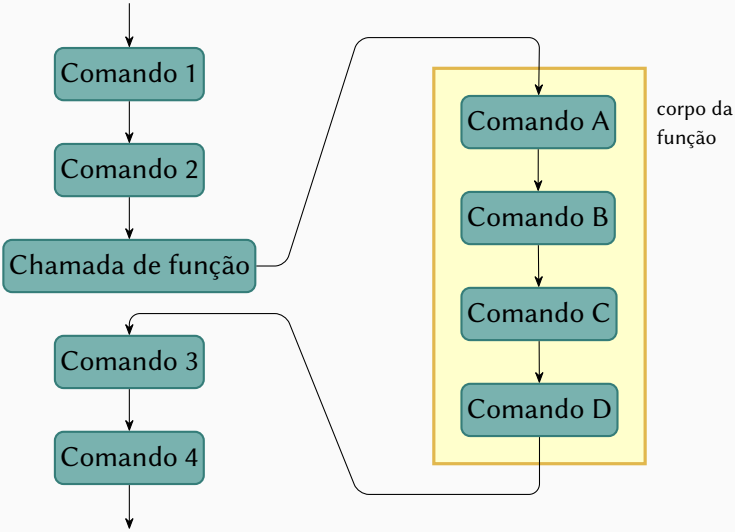
Aula 17

Nelson Lago

1°/2025

Previously on MAC113...

Funções são “desvios”



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 – 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) 😕

Funções – escopo

- **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**

Criando vectors

- `vec <- c(1, 2, 3, 4)` **(numeric)**
- `vec <- 1:4` **(integer)**
- `vec <- integer(4)` **(integer)**
- `vec <- rep(3.14, 4)` **(numeric)**
  - `numeric(4)` é o mesmo que `rep(0, 4)`
- `vec <- vector_maior[1:4]` **(herdado)**
- `vec <- c(um_vector, outro_vector)` **(herdado)**
  - Exemplo: `vec <- c(1:2, 3:4)`

podemos usar mais variáveis no laço além da variável de controle

Indicadores de passagem

Dada uma lista com a idade dos alunos de MAC113, imprima os elementos da lista, um em cada linha e diga se há algum menor de idade na turma

```
idades <- c(18, 18, 19, 18, 20, 18, 19, 17, 18, 19)
menores <- FALSE
i <- 1
while (i <= length(idades)) {
    if (idades[i] < 18) { menores <- TRUE }
    cat("Idade:", idades[i], "\n")
    i <- i + 1
}
if (menores) {
    cat("Há menores de idade na turma", "\n")
}
```

Indicadores de passagem

- **Chamamos uma variável similar a `menores` do exemplo anterior de indicador de passagem**
- **O uso de indicadores de passagem é um padrão de programação bastante comum**
  - Padrões de programação são técnicas (ou truques!) úteis e, portanto, comuns, porém não óbvios
- **Um indicador de passagem é usado para indicar se "alguma coisa" aconteceu durante o laço**
  - Quando usamos um indicador de passagem, não estamos preocupados com **qual** elemento do laço causou o evento
  - Na verdade, pode haver um ou mais elementos responsáveis por essa mudança; o valor do indicador de passagem é alterado no máximo uma vez

Indicadores de passagem

- **O indicador de passagem é diferente da variável de controle do laço**
  - A variável de controle **sempre** muda de valor **no mínimo** uma vez (na última repetição) para indicar o fim do laço
  - O indicador de passagem muda de valor **zero ou uma vezes** e a mudança pode acontecer em qualquer uma das iterações
    - » *(o comando que altera o valor do indicador de passagem pode ser executado mais de uma vez, mas o valor da variável só é efetivamente alterado na primeira)*

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?!**
  1. Não é preciso reescrever condições redundantes
  2. 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

```r
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

And now for something completely different

Prelúdio

O cursor

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

```
Olá!
Tudo 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

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
```

Exercício

Escreva um programa que imprime o nome de todas as cartas do baralho (“ás de paus, 2 de paus,..., 10 de copas, valete de copas...”)

```
main <- function() {
    naipes <- c("ouros", "copas", "espadas", "paus")
    nums <- c("ás", as.character(2:10), "valete", "dama", "rei")
    i <- 1
    while (i <= length(naipes)) {
        j <- 1
        while (j <= length(nums)) {
            cat(nums[j], " de ", naipes[i], ", ", sep="")
            j <- j + 1
        }
        i <- i + 1
        cat("\n")
    }
    cat("coringa vermelho, coringa preto\n")
}
main()
```

Exercício

Leia um número natural e faça a fatoração desse número em primos, indicando os fatores repetidos

```
n <- as.integer(readline("Digite um inteiro positivo: "))
divisor <- 2
while (n > 1) {
    while (n %% divisor > 0) {
        divisor <- divisor + 1
    }
    cat(divisor, "")
    n <- n / divisor
}
cat("\n")
```

Exercício

Dados os números n e m, imprima um retângulo $n \times m$ com o caracter "#". Por exemplo, para 4 e 5:

```
####
####
####
####
####
```

```
n <- as.integer(readline("Digite o número de colunas: "))
m <- as.integer(readline("Digite o número de linhas: "))
linha <- 1
while (linha <= m) {
    coluna <- 1
    while (coluna <= n) {
        cat("#")
        coluna <- coluna + 1
    }
    cat("\n")
    linha <- linha + 1
}
```

Exercício

Dados os números n e m, imprima o contorno de um retângulo $n \times m$ com o caracter "#". Por exemplo, para 4 e 5:

```
####
#  #
#  #
#  #
####
```

```r
n <- as.integer(readline("Digite o número de colunas: "))
m <- as.integer(readline("Digite o número de linhas: "))
linha <- 1
while (linha <= m) {
    coluna <- 1
    while (coluna <= n) {
        if (linha == 1 || linha == m || coluna == 1 || coluna == n) {
            cat("#")
        } else {
            cat(" ")
        }
        coluna <- coluna + 1
    }
    cat("\n")
    linha <- linha + 1
}
```