MAC 113 — Introdução à Ciência da Computação

Aula 15

Nelson Lago

1°/2025

Previously on MAC113...

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")
cores[2] <- "verde"
cat(cores[2], "\n")
```

```
vermelho
amarelo
verde
```

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

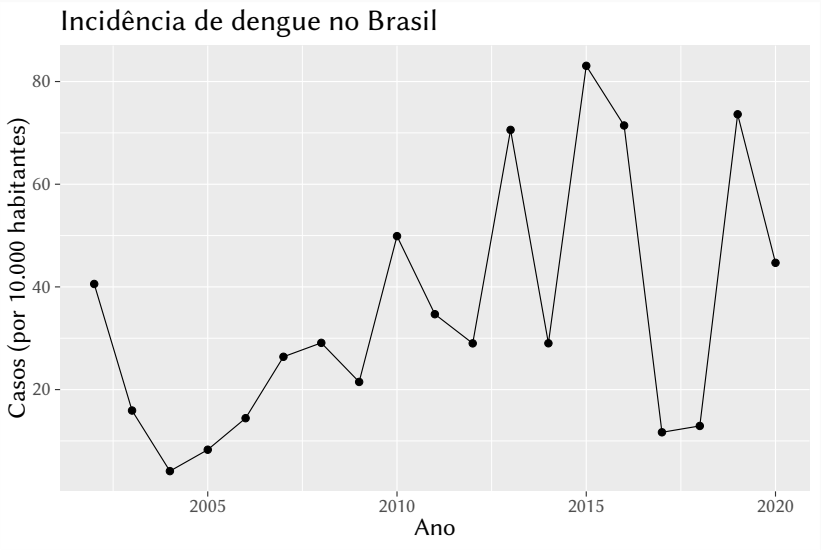
Criando vectors

- `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)`

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

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")
}
```

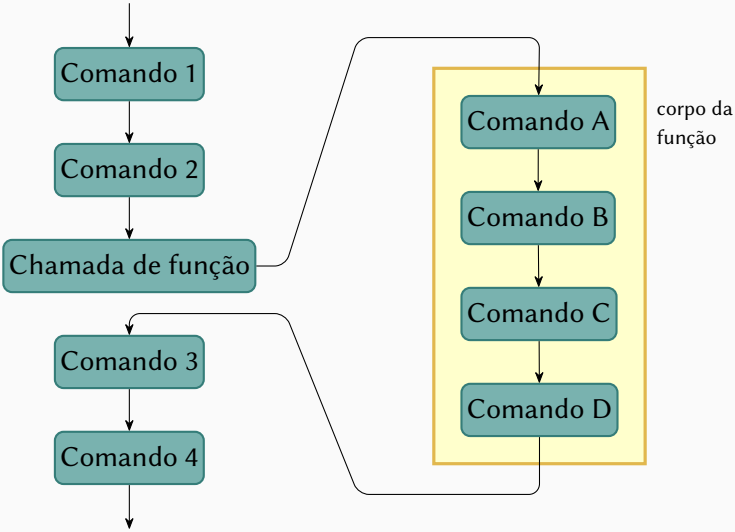
Indicadores de passagem

- **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)*

Funções são “desvios”



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

```
fatorial <- function(n) {
    fat <- 1
    while (n >= 2) {
        fat <- fat * n
        n <- n - 1
    }
    return (fat)
}
fatorial(4)
cat(fat, "\n")
```

Funções – escopo

```
fatorial <- function(n) {
    fat <- 1
    while (n >= 2) {
        fat <- fat * n
        n <- n - 1
    }
    return (fat)
}
cat(fatorial(4), "\n")
```

Funções – escopo

```
n <- 4
fatorial <- function(n) {
    fat <- 1
    while (n >= 2) {
        fat <- fat * n
        n <- n - 1
    }
    return (fat)
}
cat(fatorial(), "\n")
```

Funções – escopo

```
n <- 5
fatorial <- function(n) {
    fat <- 1
    while (n >= 2) {
        fat <- fat * n
        n <- n - 1
    }
    return (fat)
}
cat(fatorial(n), "\n")
cat(n, "\n")
```

Funções – escopo

```
fatorial_enviesado <- function(n) {
    fat <- 1
    viés <- 5

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

Funções – escopo

```
viés <- 3
fatorial_enviesado <- function(n) {
    fat <- 1
    viés <- 5

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

Funções – escopo

```
viés <- 3
fatorial_enviesado <- function(n) {
    fat <- 1


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

Funções – escopo

```
viés <- 3
fatorial_enviesado <- function(n) {
    fat <- 1


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

AAAAHHHH!!!!

Funções – escopo

```
viés <- 3
fatorial_enviesado <- function(n) {
    fat <- 1


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

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")
```

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")
```

AAAAHHHH!!!!

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()`

```
cat(fatorial(4), "\n")

fatorial <- function(n) {
    fat <- 1
    while (n >= 2) {
        fat <- fat * n
        n <- n - 1
    }
    return (fat)
}
```

Funções – main()

```
fatorial <- function(n) {
    fat <- 1
    while (n >= 2) {
        fat <- fat * n
        n <- n - 1
    }
    return (fat)
}

cat(fatorial(4), "\n")
```

Funções – main()

```
fatorial <- function(n) {
    fat <- 1
    while (n >= 2) {
        fat <- fat * n
        n <- n - 1
    }
    return (fat)
}

main <- function() {
    x <- as.integer(readline("Digite um inteiro positivo: "))
    cat(fatorial(x), "\n")
}

main()
```

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**

Exercício

Coleções

Escreva um programa que processa duas listas de números ordenadas e gera uma nova lista com a união das duas listas, também ordenada

```
mescla <- function(v1, v2) {
    tudo <- integer(length(v1) + length(v2))
    i <- 1
    j <- 1
    k <- 1
    while (i <= length(v1) && j <= length(v2)) {
        if (v1[i] < v2[j]) {
            tudo[k] <- v1[i]
            i <- i + 1
        } else {
            tudo[k] <- v2[j]
            j <- j + 1
        }
        k <- k + 1
    }
    while (i <= length(v1)) {
        tudo[k] <- v1[i]
        i <- i + 1
        k <- k + 1
    }
    while (j <= length(v2)) {
        tudo[k] <- v2[j]
        j <- j + 1
        k <- k + 1
    }
    return (tudo)
}
main <- function() {
    v1 <- c(3, 6, 7, 10, 23)
    v2 <- c(4, 5, 11, 21, 24, 27, 29, 30)
    cat(mescla(v1, v2), "\n")
}
```

And now for something slightly different

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

É só usar `else`!

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

- **Oops!!!!!!**

Condições mutuamente excludentes

```
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")
    }
}
```

- **Tratamos todos os casos corretamente, mas não deixamos claro que os três casos são mutuamente excludentes**
  - É um tanto estranho que `delta == 0` seja um "sub-caso" do `else` anterior
  - Parece que há algo de especial no caso `delta < 0`

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

- **Dados x e y, diga se eles são iguais ou qual deles é maior**
  - Quantos casos possíveis?
  - Os casos são mutuamente excludentes?

```
if (x < y) {
    cat("x é menor do que y.", "\n")
} else if (x > y) {
    cat("x é maior do que y.", "\n")
} else {
    cat("x e y são iguais.", "\n")
}
```

a condição é implícita (“o que sobrou”)

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

Exercício – escolhas

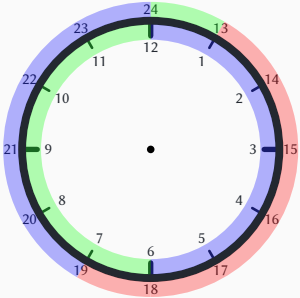
Faça um programa que pergunta ao usuário qual seu animal preferido entre gato, cachorro e passarinho e coleta uma resposta “a”, “b” ou “c”. Conforme a resposta, o programa imprime a onomatopeia correspondente

```
main <- function() {
    escolha <- readline("Qual seu animal preferido (a: gato; b: cachorro; c: passarinho)? ")
    if (escolha == "a") {
        cat("Miau!", "\n")
    } else if (escolha == "b") {
        cat("Auau!", "\n")
    } else if (escolha == "c") {
        cat("Piupiu!", "\n")
    } else {
        cat("Não sei que animal é esse!", "\n")
    }
}
main()
```

“o que sobrou”

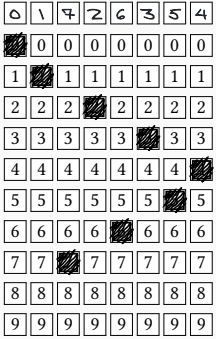
Revisitando o exercício de *login*

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



Prova – número USP

Número USP



Utilize caneta azul ou preta e preencha completamente a quadrícula.
Exemplo: ▨ **Não use** ☒

Turma: (somente um número; consulte a pessoa responsável se não souber)

11 12 20 21 22

⟵ Marque as quadrículas ao lado para formar o seu número USP e escreva seu nome completo em letra legível na linha pontilhada abaixo. **Se seu número possui menos que 8 dígitos complete com zeros à esquerda.**

Nome:

.....