MAC 113 — Introdução à Ciência da Computação

Aula 20

Nelson Lago

1°/2025

Previously on MAC113...

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**
- **Duas 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(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
```

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

```r
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

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

```r
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

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

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

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

Funções vetoriais

- Quando queremos fazer algo envolvendo vetores como um todo, funções vetoriais são muito úteis
- Tão úteis que R possui várias funções vetoriais prontas!

```
x <- c(1, 5, 3)
cat(max(x), mean(x), "\n")
```

```
5 3
```

- Tão úteis que, em R, muitas funções e operadores que podem processar dados "simples" são também funções vetoriais!

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
z <- x + y
cat(z, "\n")
```

```
5 7 9
```

paste()

- **Uma função vetorial interessante é `paste()`**
  - Ela processa os elementos de um vetor e gera um vetor de strings correspondente

```
frutas <- c("caqui", "maçã", "abacate")
cat(paste("Eu gosto de", frutas), "\n")
```

```
Eu gosto de caqui Eu gosto de maçã Eu gosto de abacate
```

Exercício – cartas do baralho

Usando a função `paste()`, 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() {
    cartas <- c()
    nums <- c("ás", as.character(2:10), "valete", "dama", "rei")
    naipes <- c("ouros", "copas", "paus", "espadas")
    for (naipe in naipes) {
        cartas <- c(cartas, paste(nums, "de", naipe))
    }
    cartas <- c(cartas, "coringa vermelho", "coringa preto")
    cat(cartas, "\n")
}
main()
```

paste()

- **paste()** pode processar mais de um vector por vez

```
nums <- 1:10
dobros <- nums * 2
cat(paste(dobros, "é o dobro de", nums, "\n"), sep="")
```

```
2 é o dobro de 1
4 é o dobro de 2
6 é o dobro de 3
8 é o dobro de 4
10 é o dobro de 5
12 é o dobro de 6
14 é o dobro de 7
16 é o dobro de 8
18 é o dobro de 9
20 é o dobro de 10
```

- **Mas o que acontece se os vectors não são do mesmo tamanho?!**

Exercício – funções vetoriais

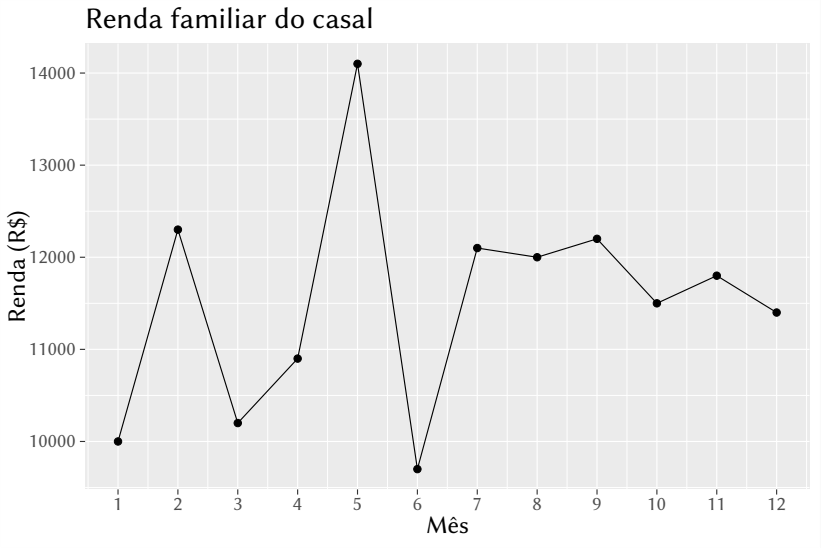
Os rendimentos de cada um dos cônjuges de um casal de *freelancers* no ano passado foi

```
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(ggplot2)
library(scales)
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=breaks_pretty(n=12))
print(p)
```

Renda familiar

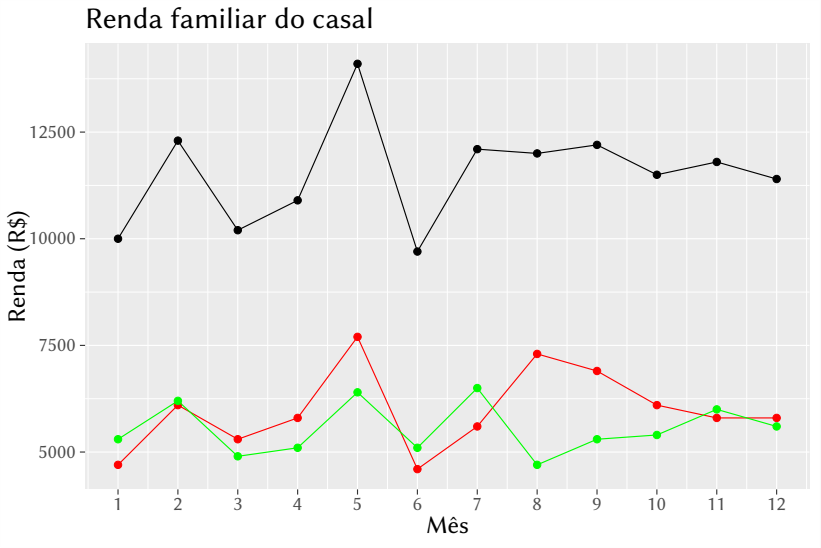


Exercício – funções vetoriais

Inclua as rendas individuais no gráfico anterior

```
library(ggplot2)
library(scales)
renda <- cônjuge1 + cônjuge2
p <- ggplot(NULL, aes(x=1:12)) +
     geom_line(aes(y=renda)) +
     geom_point(aes(y=renda)) +
     geom_line(aes(y=cônjuge1), color="red") +
     geom_point(aes(y=cônjuge1), color="red") +
     geom_line(aes(y=cônjuge2), color="green") +
     geom_point(aes(y=cônjuge2), color="green") +
     labs(title="Renda familiar do casal", x="Mês", y="Renda (R$)") +
     scale_x_continuous(breaks=breaks_pretty(n=12))
print(p)
```

Renda familiar II



Exercício – funções vetoriais

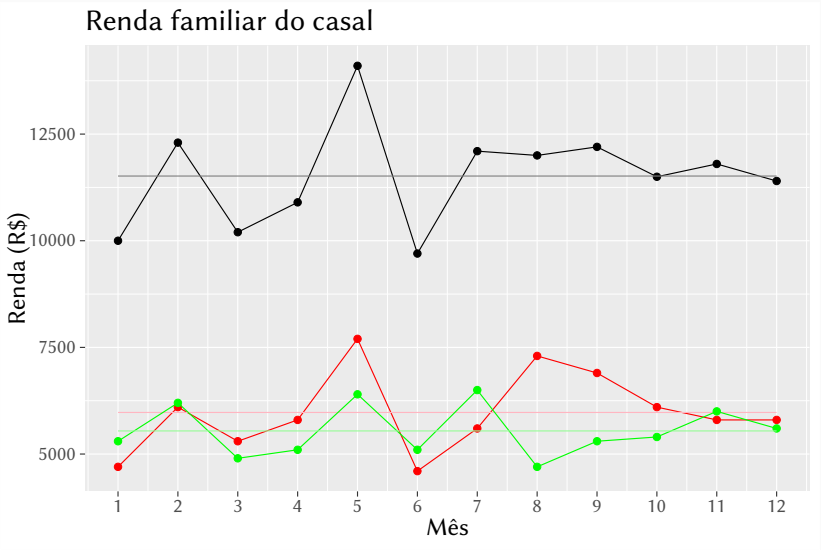
Inclua uma linha para as médias anuais de cada uma das rendas no gráfico anterior

```
library(ggplot2)
library(scales)
renda <- cônjuge1 + cônjuge2
p <- ggplot(NULL, aes(x=1:12)) +
     geom_line(aes(y=renda)) +
     geom_point(aes(y=renda)) +
     geom_line(aes(y=mean(renda)), color="gray50") +
     geom_line(aes(y=cônjuge1), color="red") +
     geom_point(aes(y=cônjuge1), color="red") +
     geom_line(aes(y=mean(cônjuge1)), color="lightpink") +
     geom_line(aes(y=cônjuge2), color="green") +
     geom_point(aes(y=cônjuge2), color="green") +
     geom_line(aes(y=mean(cônjuge2)), color="palegreen") +
     labs(title="Renda familiar do casal", x="Mês", y="Renda (R$)") +
     scale_x_continuous(breaks=breaks_pretty(n=12))
print(p)
```

vector – 12 elementos

valor único

Renda familiar III



Exercício – setlist

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {
    sl <- gera_setlist()
    for (i in 1:5) { imprime(sl) }
}
```

```
imprime <- function(l) {
    cat('setlist - Show da banda "Os Errantes"', "\n")
    cat("\n")
    cat(paste(1:length(l), " - ", l, "\n"), sep="")
}
```

```
gera_setlist <- function() {
    n <- as.integer(readline("Quantas músicas? "))
    setlist <- character(n)
    for (i in 1:n) {
        música <- readline("Qual a próxima música? ")
        setlist[i] <- música
    }
    return(setlist)
}
```

Exercício – tabuada

Imprima as tabuadas de 1 a 10

```
main <- function() {
    for (n in 1:10) {
        cat(format(1:10 * n, width=3), "\n")
    }
}
main()
```

Exercício – quadrados imprecisos

Crie uma função que recebe um vector e devolve outro vector com os quadrados dos elementos do primeiro **acrescidos de um erro aleatório**

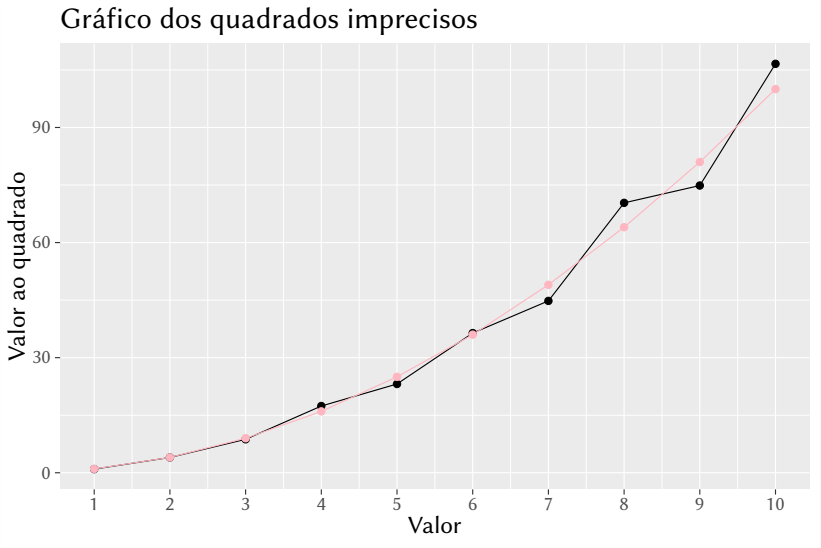
```
quadrados_imprecisos <- function(v) {
    return (v**2 * runif(length(v), 0.9, 1.1))
}
```

Exercício – quadrados imprecisos

Que tal um gráfico?

```
library(ggplot2)
library(scales)
main <- function() {
  vals <- 1:10
  tosco <- quadrados_imprecisos(vals)
  p <- ggplot(NULL, aes(x=vals)) +
    geom_line(aes(y=tosco)) +
    geom_point(aes(y=tosco)) +
    geom_line(aes(y=vals**2), color="lightpink") +
    geom_point(aes(y=vals**2), color="lightpink") +
    labs(title="Gráfico dos quadrados imprecisos", x="Valor", y="Valor ao quadrado") +
    scale_x_continuous(breaks=breaks_pretty(n=10))
  print(p)
}
main()
```

Quadrados imprecisos



And now for something completely different

Abstrações

O que é uma pessoa?

- **Julinho da Adelaide**
- **Anna O.**
- **Paciente 427**
- **Alan Mathison Turing, nascido em Londres em 23 de junho de 1912**
- **...**

Abstrair (para nós): dar atenção a apenas partes de um objeto

Abstrações

- **As linguagens de programação oferecem abstrações básicas**
  - uma soma é uma série de operações que ocorrem no nível elétrico, mas normalmente pensamos apenas em "+"
  - variáveis, funções, coleções...
- **Com base nelas, criamos novas abstrações, que se tornam novos conceitos**
  - Como as peças de um Lego podem ser usadas para construir formas diversas
- **Para objetos computacionais, como por exemplo vectors**
  - Independentes do problema
  - Independentes do conteúdo
- **Para objetos relacionados ao problema a ser resolvido**
  - Variáveis ("`dia`", "`mês`" etc.)
  - Funções ("`campeonato()`", "`bissexto()`" etc.)

Abstrações

- **Qual uma boa abstração para "pessoa" em uma instituição de ensino?**
  - Nome
  - ID (numérico)
  - Endereço
  - Curso
- **Podemos usar variáveis para cada um desses itens...**
- **Mas como representar uma lista de pessoas??**

Abstrações

```r
nomes <- c(…)
ids <- c(…)
ends <- c(…)
cursos <- c(…)
for (i in 1:length(nomes)) {
  cat(glue("Nome: {nomes[i]}; ID: {ids[i]}; "))
  cat(glue("endereço: {ends[i]}; curso: {cursos[i]}"), "\n")
}
```

- Cada aluno é representado pelo **índice** que ele ocupa em cada vector
- Funciona!
- **MAS...**
- Não dá para criar uma única variável para representar um aluno
- Sempre que quisermos manipular os alunos, precisamos processar os quatro vectors de uma vez
- ...
- O ideal é representar todos esses dados como um **conjunto**

Abstrações

- **Que tal ao contrário?**
- **Além de vectors, R possui listas**
  - Também é um conjunto ordenado, mas os elementos de uma lista podem ser de tipos diferentes 👋
- **Um aluno é representado por uma lista (que é um conjunto!)**
  - O primeiro elemento dessa lista é o nome, o segundo é o ID, o terceiro é o endereço e o quarto é o curso

```
mano <- list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos")
cat(glue("Nome: {mano[[1]]}; ID: {mano[[2]]}; "))
cat(glue("endereço: {mano[[3]]}; curso: {mano[[4]]}"), "\n")
```

- **Acabamos de criar um novo conceito (a ideia de "pessoa") com base em uma nova abstração**
  - E não vamos pensar muito no fato de essa abstração ser simplesmente uma lista
- **Mas e agora, como representar uma lista de pessoas??**

Abstrações

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")
)

for (i in 1:length(alunos)) {
  aluno <- alunos[[i]]
  cat(glue("Nome: {aluno[[1]]}; ID: {aluno[[2]]}; "))
  cat(glue("endereço: {aluno[[3]]}; curso: {aluno[[4]]}"), "\n")
}
```