MAC 113 — Introdução à Ciência da Computação

Aula 21

Nelson Lago

1°/2025

Previously on MAC113...

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()
```

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

- **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")
)
itens <- list("Nome", "ID", "endereço", "curso")
for (aluno in alunos) {

  cat(paste(itens, aluno, sep=": "), sep="; ")
  cat("\n")
}
```

Abstrações

AAAAAHHHH!!!!!!

- **Uma lista em que cada elemento é uma outra lista?!**
  - Sim 😃
- **Em geral, fazemos vectors para percorrer todos os elementos**
  - Lista de preços, lista dos vértices de um polígono, lista de operações bancárias...
- **MAS**
- **Aqui estamos usando a lista para representar uma pessoa, então não faz sentido percorrer a lista**
  - Não é uma "lista de itens", mas sim as diferentes informações associadas a uma pessoa (é uma abstração: podemos "esquecer" que é uma lista)
    - » *(você **pode** percorrer a lista, mas em geral não é isso que você quer)*
- **Mas faz sentido percorrer a lista de pessoas**

Abstrações

- **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
- **Isso é uma convenção**
- **MAS**

Abstrações

- É meio besta usar números para representar conceitos como “nome”, “endereço” etc.
- Listas permitem dar **nomes** a cada um dos seus índices (“posições”)

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

```
mano <- list(nome="Alan Turing", ID=1234,
             end="Rua dos bobos, 0", curso="análise de algoritmos")
cat(glue("Nome: {mano$nome}; ID: {mano$ID}; "))
cat(glue("endereço: {mano$end}; curso: {mano$curso}"), "\n")
```

```
mano <- list(Nome="Alan Turing", ID=1234,
             endereço="Rua dos bobos, 0", curso="análise de algoritmos")
cat(paste(names(mano), unlist(mano), sep=": "), sep="; ")
cat("\n")
```

Cuidado!

- **Os nomes são strings (character)**
- **Ou seja, normalmente ficam entre aspas!**
- **MAS, para facilitar, R permite omitir as aspas**
  1. Na definição dos elementos, com "="

     ```
     lista <- list(primeiro=1, segundo=2)
     ```

  2. Ao acessar um elemento usando a notação com "$"

     ```
     nome <- aluno$nome
     ```

Lembre-se:

- **Nomes em listas são opcionais**
- **Eles fazem muito sentido quando usamos uma lista para criar uma abstração**
- **Eles não são muito úteis quando usamos uma lista como uma lista de itens**
  - Nos exemplos anteriores, usamos nomes com a abstração pessoa mas não com a lista de pessoas

Abstrações

- **Em outras linguagens, as coleções mais ou menos similares às listas de R são chamadas dicionários**
  - Dada uma “palavra”, o dicionário fornece a “definição”
- **Nas listas de R, os índices numéricos continuam valendo**
- **Em dicionários de outras linguagens, geralmente não!**
  - Nessas linguagens, não há ordem definida em dicionários
  - Ao percorrer todos os itens do dicionário, eles podem ser processados em qualquer ordem
    - » *(em versões recentes de python, a ordem é definida: os elementos são processados na ordem em que foram inseridos)*

Abstrações

- **Essas são coisas novas?**
- **Não!**
  1. Repetições (`while`, `for`, operações vetoriais)
  2. Coleções (vectors, lists)
  3. Acesso a itens da coleção individualmente (pelo índice numérico ou pelo "nome")
- **Sim!**
  - Abstrações mais poderosas

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
    notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
    cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
}
main()
```

```
Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
```

Exercício – Lista de notas

E para acrescentar um aluno?

```
library(glue)
main <- function() {
    notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
    notas[["Rê"]] <- 7.6
    cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
}
main()
```

```
Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
Rê: 7.6
```

Exercício – Lista de notas

Qual a média da turma?

```
library(glue)
main <- function() {
    notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
    notas$Rê <- 7.6
    cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
    cat("Média da turma:", mean(unlist(notas)), "\n")
}
main()
```

```
Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
Rê: 7.6
Média da turma: 8.1
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {
    usuários <- list(
        list(login="turing", senha="tmachine", nome="Alan Turing"),
        list(login="llace", senha="anengine", nome="Ada Lovelace"),
        list(login="hopper", senha="business", nome="Grace Hopper")
    )
    uid <- readline("username: ")
    senha <- readline("senha: ")
    nome <- checa_login(uid, senha, usuários)
    #cat(saudação(nome), "\n")
    cat("Bem-vindo,", nome, "\n")
}
checa_login <- function(uid, senha, usuários) {
    for (user in usuários) {
        if (uid == user$login && senha == user$senha) {
            return (user$nome)
        }
    }
    return("")
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
    opção <- as.integer(readline("Escolha a opção desejada: "))
    if (opção == 1) {
        nome <- readline("Nome? ")
        fone <- readline("Telefone? ")
        contatos[[nome]] <- fone
    } else if (opção == 2) {
        nome <- readline("Nome? ")
        cat("O telefone de", nome, "é", contatos[[nome]], "\n")
    } else if (opção == 3) {
        cat(paste(names(contatos), unlist(contatos), sep=": "), sep="\n")
    } else if (opção == 4) {
        cat("Obrigado por usar a agenda telefônica!\n")
    } else {
        cat("Oops! Opção inválida, tente novamente\n")
    }
}
```