MAC 113 — Introdução à Ciência da Computação

Aula 22

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")
frases <- paste("Eu gosto de", frutas, sep=": ")
cat(frases[2], "\n")
cat(paste("Eu gosto de", frutas, sep=": "), sep="\n")
```

```
Eu gosto de: maçã
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 – 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")
    }
}
```

Comandos básicos com vectors e lists

- `vec <- 1:10`
- `vec <- rep(3.14, 4)`
- `vec <- c("a", "b", "c")`
- `lista <- list(a=1, b=2, c=3)`
- `length(blah)` (list ou vector)
- `tudo_junto <- c(blah, bleh)`
  (lists ou vectors)
- `vec[X]`
- `lista[[X]]` (numérico)
- `blah[X:Y]`
  (numérico – lists ou vectors)
- `lista[[X]]` (character)
- `lista$X`
  (character, atalho sem aspas)
- `for (item in blah)`
  (list ou vector)
- `names(lista)`
  - `for (nome in names(lista))`
- `unlist(lista)`
- `vec[X] <- valor` (atribui)
- `vec <- append(vec, valor)`
  (acrescenta)
- `lista$nome <- valor`
  (atribui ou acrescenta)

Exercício – Boletim escolar 1/2

No colégio Paçei Direto, os alunos são registrados com seus nomes, números de identificação e as notas das disciplinas, desta forma:

```
aluno <- list(nome="Fulano de Tal", ID=1234, matemática=9.2,
              português=8.4, física=4.8, história=6.2)
```

Exercício – Boletim escolar 2/2

Dada uma lista de alunos, faça um programa que imprime o boletim escolar de cada um

```
library(glue)
alunos <- list(
  list(nome="Alan Turing", ID=1234, matemática=9.7,
       português=1.4, física=9.2, história=8.7),
  list(nome="Ada Lovelace", ID=4321, matemática=9.8,
       português=1.2, física=10, história=9.2)
)
for (aluno in alunos) {
  cat(glue("Boletim do aluno {aluno$nome}:"), "\n")
  cat("matemática:", aluno$matemática, "\n")
  cat("português:", aluno$português, "\n")
  cat("física:", aluno$física, "\n")
  cat("história:", aluno$história, "\n")
}
```

Mais sobre fatiamento

- **`vec[2:7]` → recorta o vector (o resultado é um pedaço do vector)**
- **`lista[2:7]` → recorta a lista (o resultado é um pedaço da lista, incluindo os nomes)**
- **`lista[2]` → recorta a lista (o resultado é uma lista com um único elemento e seu nome)**
- **`vec[2]` → recorta o vector (o resultado é um "vector" com apenas um elemento, mas isso é quase a mesma coisa que o elemento em si)**
- **`lista[[2]]` → extrai o elemento em si da lista (sem nome etc.)**
  - É mais ou menos `unlist(lista[2])`
  - Só funciona para um elemento, não para recortes com vários elementos
- **`vec[[2]]` → idem para vectors, mas é mais ou menos redundante com `vec[2]`**

Mais sobre fatiamento

- **PERA**
- **`vec[2:7]` → `2:7` não é um vector?!?!**
- **Sim!**
- **Para recortar um vector, você usa um vector de inteiros com os índices dos elementos que você quer**
  - `pontas <- vec[c(1, length(vec))]`

Escreva uma função que recebe um *vector* e devolve outro *vector*, igual ao primeiro, mas na ordem inversa

```
inverte_vec <- function(v) {
    return(v[length(v):1])
}
```

- **Evidentemente, o mesmo funciona para listas**
  - `l <- list(a=1, b=2, c=3)`
    `pontas <- l[c('a', 'c')]` ou `pontas <- l[c(1, 3)]`

Exercício – Boletim escolar

Dada uma lista de alunos, faça um programa que imprime o boletim escolar de cada um

```
library(glue)
alunos <- list(
  list(nome="Alan Turing", ID=1234, matemática=9.7,
       português=1.4, física=9.2, história=8.7),
  list(nome="Ada Lovelace", ID=4321, matemática=9.8,
       português=1.2, física=10, história=9.2)
)
for (aluno in alunos) {
  cat(glue("Boletim do aluno {aluno$nome}:"), "\n")
  notas <- aluno[c("matemática", "português", "física", "história")]
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
  cat("Média de todas as disciplinas:", mean(unlist(notas)), "\n")

}
```

Exercício – Lista de preços

Dada uma lista de produtos, faça um programa que imprime a lista de preços

```
prods <- list(
  list(nome="Desentortador de banana", código=1234,
       preço=120, estoque=30, fornecedor="Falida SA"),
  list(nome="Cola para dedos", código=4311,
       preço=17, estoque=83, fornecedor="Descolada Ltda")
)
cat("Lista de preços:\n")
for (prod in prods) {
    cat(prod$nome, ": R$", prod$preço, ",00", "\n", sep="")
}
```

Exercício – Controle de estoque

Dada uma lista de produtos, faça um programa que imprime a quantidade de cada produto em estoque e o nome do fornecedor

```
prods <- list(
  list(nome="Desentortador de banana", código=1234,
       preço=120, estoque=30, fornecedor="Falida SA"),
  list(nome="Cola para dedos", código=4311,
       preço=17, estoque=83, fornecedor="Descolada Ltda")
)
cat("Estoque:\n")
for (prod in prods) {
    cat(prod$nome, ": ", prod$estoque, " unidades em estoque. Fornecedor: ",
        prod$fornecedor, "\n", sep="")
}
```

Exercício – Controle de estoque

Modifique o programa anterior para apenas imprimir os itens que têm menos de 50 unidades em estoque

```
prods <- list(
  list(nome="Desentortador de banana", código=1234,
       preço=120, estoque=30, fornecedor="Falida SA"),
  list(nome="Cola para dedos", código=4311,
       preço=17, estoque=83, fornecedor="Descolada Ltda")
)
cat("Estoque:\n")
for (prod in prods) {
  if (prod$estoque < 50) {
    cat(prod$nome, ": ", prod$estoque, " unidades em estoque. Fornecedor: ",
        prod$fornecedor, "\n", sep="")
  }
}
```