MAC 113 — Introdução à Ciência da Computação

Aula 23

Nelson Lago

1°/2025

Previously on MAC113...

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

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

Dada uma lista de alunos, faça um programa que imprime uma tabela com as notas de todos os alunos

```
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)
)
disciplinas <- c("matemática", "português", "física", "história")
cat(format("Nome", width=15), format(disciplinas, width=10, justify="right"), "\n")
for (aluno in alunos) {
    notas <- aluno[disciplinas]
    cat(format(aluno$nome, width=15),
        paste(format(unlist(notas), width=10, justify="right")))
    cat("\n")
}
```

```
Nome            matemática  português     física   história
Alan Turing            9.7        1.4        9.2        8.7
Ada Lovelace           9.8        1.2       10.0        9.2
```

And now for something completely different

Tabelas

Paciente	Colesterol (mg/dL)	Idade	Quarto	SUS
Fulano de Tal	164	49	132-A	S
Ciclano de Tal	227	83	231-B	N
Média/Total	195,6	66	–	1

- **Por que usamos tabelas?**
- **Provavelmente, várias razões** 😜
- **Uma delas é que se trata de uma visualização dupla**
- **Podemos olhar cada linha ou cada coluna**

Tabelas

- **Em tabelas desse tipo**
  - Cada linha corresponde a uma abstração (“pessoa”, “produto” etc.)
  - Cada coluna corresponde a uma coleção de valores de um mesmo tipo (“preço”, “colesterol”, “idade” etc.)
- **Podemos armazenar cada linha como uma list**
  - Mas processar as colunas (por exemplo, para calcular a média) é trabalhoso
- **Podemos armazenar cada coluna como um vector e usar os índices para identificar as linhas**
  - Mas já vimos antes que isso é um tanto trabalhoso também
- **Podemos usar uma list em que cada elemento é um vector correspondente a uma coluna**
  - Mas isso é apenas uma variação da ideia anterior

Precisamos de algo novo!

Algo que se assemelhe a uma **tabela**

Dataframes

(na verdade, **tibbles**)

Dataframes

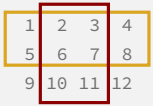
- **Dataframes são bastante similares a lists**
- **Cada item é um vector que representa uma coluna**

```
library(tibble)
#df <- data.frame(nome=c("Zé", "Fê", "Má", "Lú"), ID=c(1,2,3,4), nota=c(7, 8.5, 8.3, 9.1))
df <- tibble(nome=c("Zé", "Fê", "Má", "Lú"), ID=c(1,2,3,4), nota=c(7, 8.5, 8.3, 9.1))
cat("Alunos:", df$nome, "\n")
cat("Média da turma:", mean(df$nota), "\n")
cat("Lista de chamada:\n")
cat(paste(df$nome, " (", df$ID, ") ______", sep=""), sep="\n")
```

```
Alunos: Zé Fê Má Lú
Média da turma: 8.225
Lista de chamada:
Zé (1) ______
Fê (2) ______
Má (3) ______
Lú (4) ______
```

Recortes com dataframes

- **A maneira mais razoável de nos referirmos a elementos em uma tabela é através de suas coordenadas**



- **tabela**$_{l,c}$
- **tabela**$_{1-2,2-3}$

```
2 3
6 7
```

Dataframes

```r
library(tibble)
#df <- data.frame(nome=c("Zé", "Fê", "Má", "Lú"), ID=c(1,2,3,4), nota=c(7, 8.5, 8.3, 9.1))
df <- tibble(nome=c("Zé", "Fê", "Má", "Lú"), ID=c(1,2,3,4), nota=c(7, 8.5, 8.3, 9.1))
cat("Boletim de notas:\n")
for (i in seq_len(nrow(df))) {
  line <- df[i,]
  cat(unlist(line[c("nome", "nota")]), "\n")
}
```

```
Boletim de notas:
Zé 7
Fê 8.5
Má 8.3
Lú 9.1
```

Exercício – Lista de preços

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

```
library(tibble)
#prods <- data.frame(
prods <- tibble(
    nome=c("Desentortador de banana", "Cola para dedos"),
    código=c(1234, 4311),
    preço=c(120, 17),
    estoque=c(30, 83),
    fornecedor=c("Falida SA", "Descolada Ltda")
)
cat("Lista de preços:\n")
cat(paste(prods$nome, ": R$", prods$preço, ",00", sep=""), sep="\n")
```

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

```
library(tibble)
#prods <- data.frame(
prods <- tibble(
    nome=c("Desentortador de banana", "Cola para dedos"),
    código=c(1234, 4311),
    preço=c(120, 17),
    estoque=c(30, 83),
    fornecedor=c("Falida SA", "Descolada Ltda")
)
cat("Estoque:\n")
cat(paste(prods$nome, ": ", prods$estoque, " unidades em estoque. Fornecedor: ",
        prods$fornecedor, sep=""), "", sep="\n")
```

Exercício – Notas

Dada uma lista de alunos com suas notas, crie um programa que imprime as médias dos alunos para cada disciplina

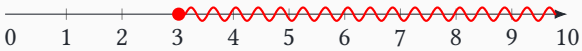
```
library(glue)
library(tibble)
alunos <- list(
  list(nome="Alan Turing", ID=1234, matemática=9.7,
       português=1.4, física=9.2, história=8.7),
  list(nome="Ada Lovelace", ID=4321, matemática=9.8,
       português=1.2, física=10, história=9.2)
)
tmp <- as_tibble(alunos[[1]])
for (i in seq_len(length(alunos[-1])) + 1) {tmp <- rbind(tmp, alunos[[i]]) }
alunos <- tmp
disciplinas <- c("matemática", "português", "física", "história")
cat("Médias:\n")
cat(format(disciplinas, width=10, justify="right"), "\n")
for (col in alunos[disciplinas]) {
  cat(format(mean(col), width=10), "")
}
```

And now for something slightly different

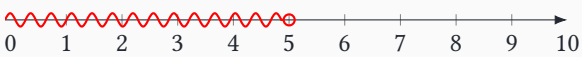
Booleanos e conjuntos

Às vezes vale a pena entender o operador “and” como “intersecção de conjuntos”
(e o operador “or” como “união de conjuntos”)

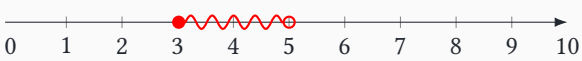
$3 \leq$ nota



nota $<$ 5



$3 \leq$ nota && nota $<$ 5



Álgebra booleana

Propriedades Comutativas	A and B = B and A	A or B = B or A
Propriedades Distributivas	A and (B or C) = (A and B) or (A and C)	A or (B and C) = (A or B) and (A or C)
Propriedades Associativas	(A or B) or C = A or (B or C)	(A and B) and C = A and (B and C)
Propriedades Idempotentes	A and A = A	A or A = A
Dupla Negação	not not A = A	
Elementos Absorventes	A or True = True	A and False = False
Elementos Neutros	A or False = A	A and True = A
Leis de De Morgan	not (A or B) = (not A) and (not B)	not (A and B) = (not A) or (not B)

Álgebra booleana

- **pizza**
- **sushi**
- **moqueca**
- **hambúrguer**

- **Sou guloso**
  - pizza **ou** hambúrguer
- **Sou alérgico a peixes**
  - **nem** sushi **nem** moqueca

Equivalentes! Qual usar?

O que facilita o entendimento

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca** — `(! sushi) && (! moqueca)`
- **não quero se for (sushi ou moqueca)** — `! (sushi || moqueca)`

- **Topa jantar e depois cinema?**
  - Não tenho grana para tudo isso!
    - » `! (jantar && cinema)`
  - Tem que abrir mão de (pelo menos) um deles
    - » `(! jantar) || (! cinema)`

Exercício

Leia duas datas fornecidas pelo usuário (para cada data você deve ler três valores: dia, mês e ano) e informe qual dessas datas é a mais recente

```
dia1 <- as.integer(readline("Digite o primeiro dia: "))
mês1 <- as.integer(readline("Digite o primeiro mês: "))
ano1 <- as.integer(readline("Digite o primeiro ano: "))
dia2 <- as.integer(readline("Digite o segundo dia: "))
mês2 <- as.integer(readline("Digite o segundo mês: "))
ano2 <- as.integer(readline("Digite o segundo ano: "))
…
```

Exercício

```
…
if (ano1 == ano2) {
    if (mês1 == mês2) {
        if (dia1 == dia2) {
            cat("As datas são iguais\n")
        } else if (dia1 > dia2) {
            cat("A primeira data é mais recente\n")
        } else {
            cat("A segunda data é mais recente\n")
        }
    } else if (mês1 > mês2) {
        cat("A primeira data é mais recente\n")
    } else {
        cat("A segunda data é mais recente\n")
    }
} else if (ano1 > ano2) {
    cat("A primeira data é mais recente\n")
} else {
    cat("A segunda data é mais recente\n")
}
```

Exercício

```
…
if (ano1 == ano2 && mês1 == mês2 && dia1 == dia2) {
    cat("As datas são iguais\n")
} else if ( (ano1 > ano2) || (ano1 == ano2 && mês1 > mês2) || (ano1 == ano2 && mês1 == mês2 && dia1 > dia2) ) {
    cat("A primeira data é mais recente\n")
} else {
    cat("A segunda data é mais recente\n")
}
```

Propriedade distributiva: (A and B) or (A and C) = A and (B or C)

```
…
if (ano1 == ano2 && mês1 == mês2 && dia1 == dia2) {
    cat("As datas são iguais\n")
} else if ( (ano1 > ano2) || (ano1 == ano2 && (mês1 > mês2 || (mês1 == mês2 && dia1 > dia2))) ) {
    cat("A primeira data é mais recente\n")
} else {
    cat("A segunda data é mais recente\n")
}
```

Exercício

```
…
if (ano1 == ano2 && mês1 == mês2 && dia1 == dia2) {
    cat("As datas são iguais\n")
} else if (  ano1 > ano2
           ||  ano1 == ano2 && mês1 > mês2
           ||  ano1 == ano2 && mês1 == mês2 && dia1 > dia2  ) {

    cat("A primeira data é mais recente\n")
} else {
    cat("A segunda data é mais recente\n")
}
```

```
…
if (ano1 == ano2 && mês1 == mês2 && dia1 == dia2) {
    cat("As datas são iguais\n")
} else if (  ano1 > ano2
           ||  ano1 == ano2 && (mês1 > mês2 ||  mês1 == mês2 && dia1 > dia2)   ) {

    cat("A primeira data é mais recente\n")
} else {
    cat("A segunda data é mais recente\n")
}
```

Exercício — condições

Dados os inteiros a e b, quais expressões são equivalentes a `a >= b`?

- `b <= a` ✔
- `a > b && a == b` ✘
- `a > b || a == b` ✔
- `a > b + 1` ✘
- `a > b - 1` ✔
- `a + 1 > b` ✔
- `! (a < b)` ✔
- `a < b == FALSE` ✔
- `! (a <= b || a != b)` ✘
- `! (a <= b && a != b)` ✔
- `(a %/% 2) >= (b %/% 2)` ✘

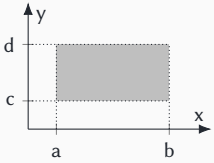
Exercício — condições

Dada a nota de um aluno em uma variável real `n`, selecione todas as expressões equivalentes a `n < 3.0 || n >= 5.0`:

- `n >= 3.0 && n < 5.0` ✘
- `! (n >= 3.0 && n < 5.0)` ✔
- `! (n >= 3.0 || n < 5.0)` ✘
- `n < 3.0 || n > 5.0 || n == 5.0` ✔
- `(n -4.0)**2 > 1.0 || n == 5.0` ✔
- `n - 4.0 >= 1.0 && n - 4.0 < -1.0` ✘
- `(n - 4.0)**2 > 1.0 && n - 4.0 != 1.0` ✘
- `n - 4.0 >= 1.0 || n - 4.0 < -1.0` ✔

Exercício — condições

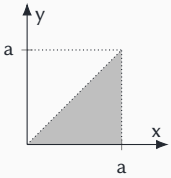
Dadas as coordenadas reais x e y de um ponto, selecione todas expressões que geram **TRUE** se esse ponto está na região sombreada da figura ao lado e **FALSE** caso contrário. A região sombreada não inclui as linhas de fronteira.



- `x > a && x < b && y > c && y < d` ✔
- `(x<b && y<d) || ! (x<=a && y<=c)` ✘
- `(x<b && y<d) && ! (x<=a || y<=c)` ✔
- `! (x<=a && x>=b && y<=c && y>=d)` ✘
- `x > a || x < b || y > c || y < d` ✘
- `! (x<=a && x>=b) || ! (y<=c && y>=d)` ✘
- `! (x<=a || x>=b) && ! (y<=c || y>=d)` ✔
- `! (x<=a || x>=b || y<=c || y>=d)` ✔

Exercício — condições

Dadas as coordenadas reais x e y de um ponto, selecione todas expressões que geram **TRUE** se esse ponto está na região sombreada da figura ao lado e **FALSE** caso contrário. A região sombreada não inclui as linhas de fronteira.



- `x < a || (0 < y && y < x)` ✘
- `! (x >= a && y <= 0 && x <= y)` ✔
- `x >= a || y <= 0 || x <= y` ✘
- `x < a && y > 0 && ! (x <= y)` ✘
- `y < x && ! (x >= a || 0 >= y)` ✔
- `! (x >= a || y <= 0 || x <= y)` ✔
- `x < a && y > 0 && x > y` ✔
- `x > 0 || x < a || y < a || y > 0 || x > y` ✘
- `x < a || y > 0 || x > y` ✘
- `x>0 && x<a && y<a && y>0 && x>y` ✔