MAC 113 — Introdução à Ciência da Computação

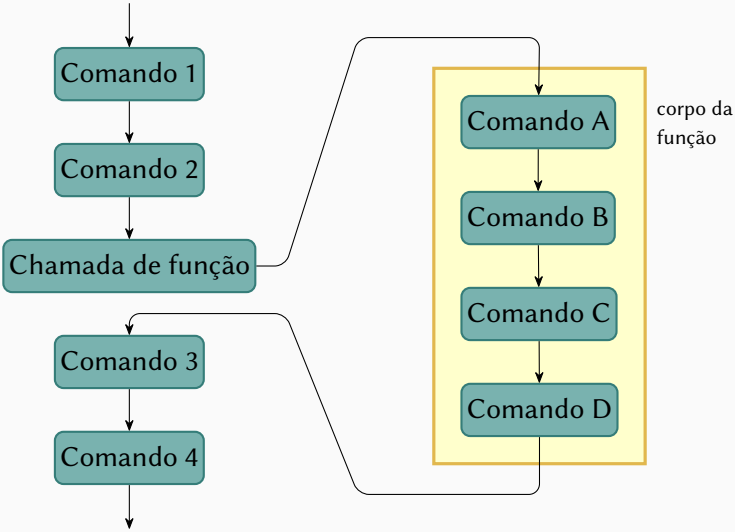
Aula 14

Nelson Lago

1°/2025

Previously on MAC113...

Funções são “desvios”



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"

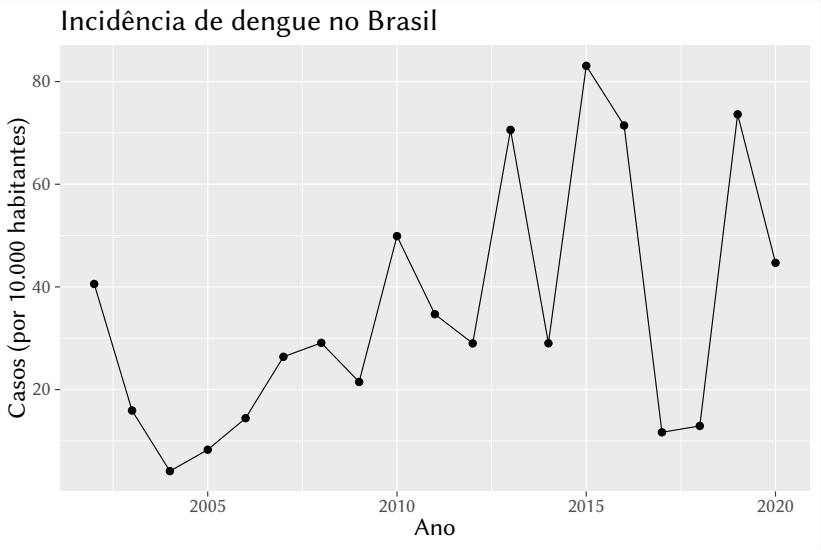
Criando vectors

- `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)*

Exercícios

Extremos

Escreva uma função que recebe uma lista de números ordenada e devolve uma lista com o menor e o maior números da lista dada

```
extremos <- function(v) {
    return (c(v[1],v[length(v)]))
}
```

Sequência de inteiros

Escreva uma função que recebe uma lista de números inteiros e devolve o pedaço da lista entre o primeiro e o último números estritamente positivos (inclusive)

```
sequência_positivos <- function(v) {
    primeiro_positivo <- 0
    último_positivo <- 0
    i <- 1
    while (i <= length(v)) {
        if (v[i] > 0) {
            último_positivo <- i
            if (primeiro_positivo == 0) { primeiro_positivo <- i }
        }
        i <- i + 1
    }
    if (primeiro_positivo == 0) {
        return (c())
    } else {
        return (v[primeiro_positivo:último_positivo])
    }
}
```

Invertendo um *vector*

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

```
inverte_vec <- function(v) {
    v2 <- v # novo vector com o mesmo tamanho e tipo do original
    i <- 1
    while (i <= length(v)) {
        v2[length(v) - i + 1] <- v[i]
        i <- i + 1
    }
    return(v2)
}
```

Exercício — calculando π com Gregory-Leibniz

Dado um inteiro $k \geq 0$, é possível calcular um valor aproximado de π através da expansão de Gregory-Leibniz:

$$\pi \approx \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} \cdots + \frac{(-1)^k 4}{2k+1}$$

Escreva um programa que lê um valor para k e realiza o cálculo acima.

Exercício — calculando π com Gregory-Leibniz

$$\pi \approx \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} \cdots + \frac{(-1)^k 4}{2k+1}$$

```
main <- function() {
    k <- as.integer(readline("Digite o valor de k: "))
    cat("O valor de pi é aproximadamente", acha_pi(k), "\n")
}
acha_pi <- function(k) {
    pi <- 0
    sinal <- 1
    n <- 0
    while (n <= k) {
        pi <- pi + sinal * 4 / (2*n + 1)
        sinal <- sinal * -1
        n <- n + 1
    }
    return(pi)
}
main()
```

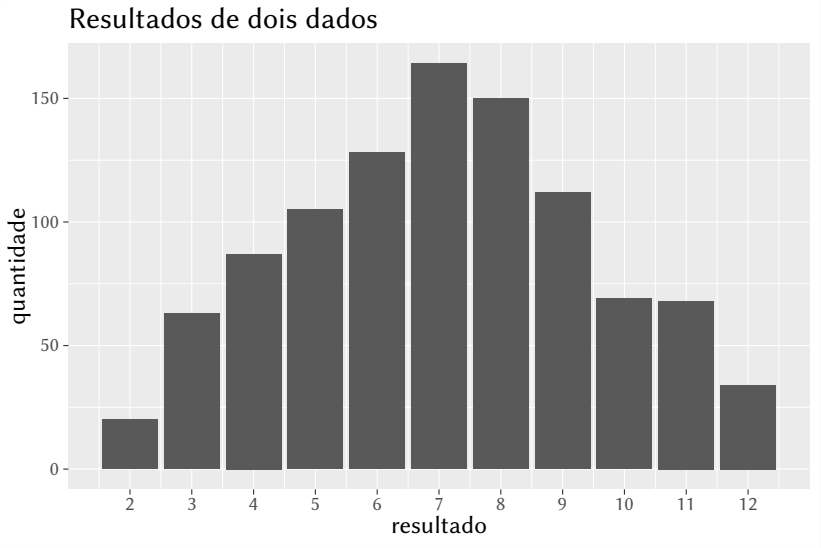
Veremos depois que essa solução não é muito boa!

Monte Carlo e dados – histograma

Usando o método de Monte Carlo, faça um histograma dos resultados esperados ao jogar dois dados

```
library(ggplot2)
library(scales)
main <- function() {
    n <- as.integer(readline("Quantas simulações? "))
    resultados <- joga_dados(n)
    p <- ggplot(NULL, aes(x=2:12, y=resultados)) +
         geom_col() +
         labs(x="resultado", y="quantidade", title="Resultados de dois dados") +
         scale_x_continuous(breaks=breaks_pretty(n=12))
    print(p)
}
main()
```

Monte Carlo e dados – histograma

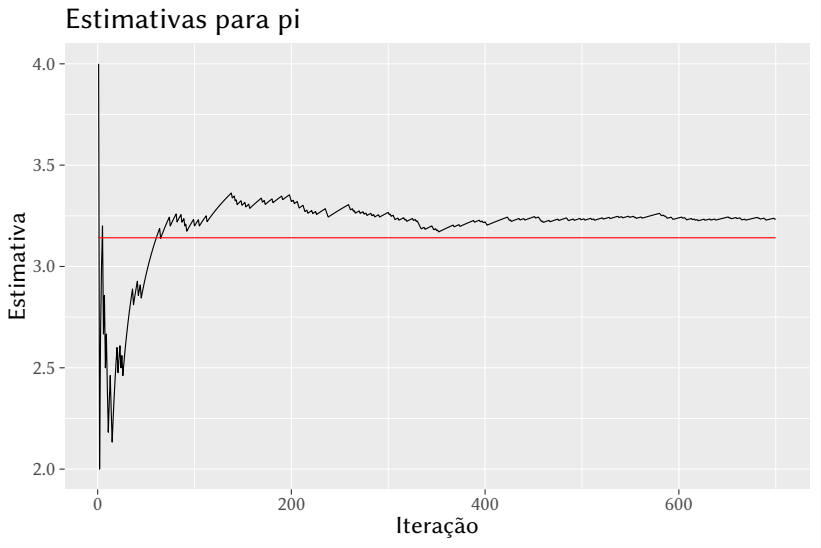


π e Monte Carlo, parte II

```
main <- function() {
    …

    p <- ggplot(NULL, aes(x=1:chutes, y=estimativas)) +
         geom_line() +
         labs(title="Estimativas para pi", x="Iteração", y="Estimativa") +
         geom_line(aes(x=1:chutes, y=pi), color="red")
   print(p)
}
main()
```

π e Monte Carlo, parte II



EP1 – função `main()`

Na função `main()`, você deve criar os vetores de nomes e de *scores* e pedir para o usuário informar o *score* de cada um dos times, além de identificar qual é o time favorito. A seguir, deve pedir para o usuário informar o número de simulações (campeonatos) que devem ser executadas e chamar a função `montecarlo(repetições, escolhido, scores)` com os parâmetros adequados. Ao término dessa função, `main()` deve imprimir a probabilidade estimada de vitória do time com maior *score*.

O vetor com os nomes dos times pode ser inicializado no próprio código:

```
nomes <- c("PBinthians", "Arbustos", "Pão Saulo",
           "Minha várzea, minha vida", "Bola murcha",
           "Na trave", "Canelada FC", "Inimigos do gol")
```

EP1 – função `main()`

```
Bem-vindo ao simulador de campeonatos da APPSP!

Digite o score do time PBinthians: 7.6
Digite o score do time Arbustos: 7.9
Digite o score do time Pão Saulo: 7.5
Digite o score do time Minha várzea, minha vida: 8.4
Digite o score do time Bola murcha: 7.2
Digite o score do time Na trave: 9.3
Digite o score do time Canelada FC: 8.1
Digite o score do time Inimigos do gol: 7.7
Quantas simulações? 10000
A probabilidade estimada de o time Na trave vencer o campeonato é 15.29%
```

EP1 – função `main()`

```
main <- function() {
    nomes <- c("PBinthians", "Arbustos", "Pão Saulo", "Minha várzea, minha vida",
               "Bola murcha", "Na trave", "Canelada FC", "Inimigos do gol")
    scores <- numeric(length(nomes))
    cat("Bem-vindo ao simulador de campeonatos da APPSP!", "\n")
    cat("\n")
    melhor <- 1
    i <- 1
    while (i <= length(scores)) {
        scores[i] <- as.numeric(readline(glue("Digite o score do time {nomes[i]}: ")))
        if (scores[i] > scores[melhor]) {melhor <- i}
        i <- i + 1
    }
    n <- as.integer(readline("Quantas simulações? "))
    prob <- montecarlo(n, melhor, scores)
    cat(glue("A probabilidade estimada de o time {nomes[melhor]} ",
             "vencer o campeonato é {prob}%"), "\n")
}
```

EP1 – função `partida(a, b, scores)`

`partida(a, b, scores)` recebe os times `a` e `b` e o vetor `scores` para simular uma partida entre os times `a` e `b`, devolvendo o número do time vencedor. O time vencedor é definido aleatoriamente, com probabilidade de vitória para o time `a` $= \frac{score_a}{score_a + score_b}$.

Antes de realizar o cálculo, no entanto, é preciso decidir se o juiz é tendencioso ou não. A probabilidade de o juiz de uma partida qualquer ser tendencioso é 20%; quando isso acontece, a atuação do juiz equivale a somar um valor entre 1.0 e 2.0 ao *score* do time "pior" (mas sem ultrapassar o máximo de 10). Caso o *score* dos dois times seja igual, o juiz pode privilegiar qualquer um deles.

EP1 – função `partida(a, b, scores)`

```
partida <- function(a, b, scores) {
    scoreA <- scores[a]
    scoreB <- scores[b]
    honesto <- runif(1, 0, 1) < .8
    if (!honesto) {
        viés <- runif(1, 1, 2)
        if (scoreA < scoreB) {
            scoreA <- scoreA + viés
            if (scoreA > 10) { scoreA <- 10 }
        } else {
            scoreB <- scoreB + viés
            if (scoreB > 10) { scoreB <- 10 }
        }
    }
    if (runif(1, 0, 1) <= scoreA / (scoreA + scoreB)) {
        return(a)
    } else {
        return(b)
    }
}
```

EP1 – função `montecarlo(repetições, escolhido, scores)`

Esta função recebe o número de simulações a serem executadas, o time para o qual se deve calcular a probabilidade de vitória e o vetor com os *scores* de todos os times. A função deve executar `campeonato(scores)` quantas vezes forem necessárias e devolver a probabilidade estimada de vitória do time `escolhido` (em porcentagem).

EP1 – função `montecarlo(repetições, escolhido, scores)`

```
montecarlo <- function(repetições, escolhido, scores) {
    sucessos <- 0
    i <- 1
    while (i <= repetições) {
        vencedor <- campeonato(scores)
        if (vencedor == escolhido) {sucessos <- sucessos + 1}
        i <- i + 1
    }
    return(sucessos / repetições * 100)
}
```

EP1 – função `campeonato(scores)`

A função `campeonato(scores)` recebe o vetor `scores` e simula um campeonato, devolvendo o número do time vencedor. Para isso, ela deve definir quais são os pares de times a se enfrentarem em cada rodada (a escolha é aleatória), simular os jogos das rodadas e o jogo da final. Cada partida é simulada pela função `partida(a, b, scores)`.

EP1 – função campeonato(scores)

```
campeonato <- function(scores) {
    times <- sample(1:length(scores)) # todos os times, ordem aleatória
    semi <- integer(4)
    i <- 1
    while (i <= 4) {
        timeA <- times[2 * i - 1]
        timeB <- times[2 * i]
        semi[i] <- partida(timeA, timeB, scores)
        i <- i + 1
    }
    semi <- sample(semi) # embaralha
    final <- integer(2)
    i <- 1
    while (i <= 2) {
        timeA <- times[2 * i - 1]
        timeB <- times[2 * i]
        final[i] <- partida(timeA, timeB, scores)
        i <- i + 1
    }
    return (partida(final[1], final[2], scores))
}
```