

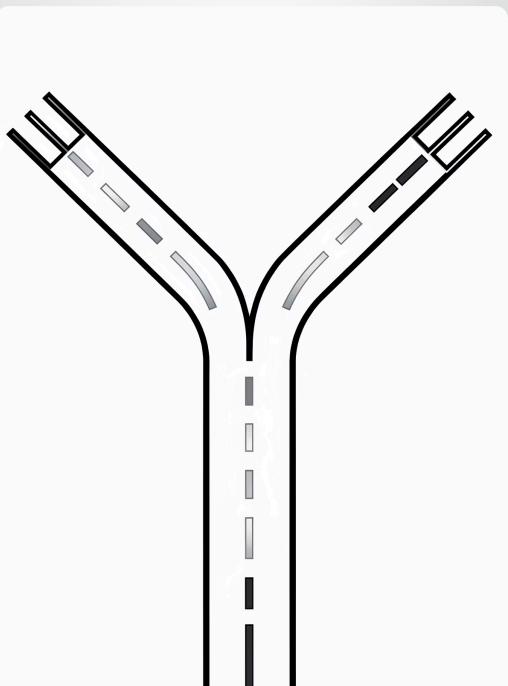
# Estrutura de Repetição em R

Aprenda a usar estruturas de repetição para automatizar tarefas e processar dados de forma eficiente em R.



Eduardo Ogasawara  
[eduardo.ogasawara@cefet-rj.br](mailto:eduardo.ogasawara@cefet-rj.br)  
<https://eic.cefet-rj.br/~eogasawara>

# Controle de Fluxo: Tomada de Decisão



## O que o controle de fluxo permite?

- Decidir se um bloco de código será executado
- Repetir instruções múltiplas vezes
- Adaptar o comportamento baseado em condições

### Condições Lógicas

Estruturas de controle dependem de **condições lógicas** que avaliam para verdadeiro ou falso

### Exemplos de Uso

Executar algo apenas se uma condição for satisfeita, ou repetir um cálculo até atingir um critério

# Trabalhando com Dados Simples

## Variáveis Escalares

Começamos com valores individuais para representar informações de uma única pessoa:

```
weight <- 60  
height <- 1.75  
subject <- "A"  
healthy <- TRUE  
bmi <- weight/height^2  
bmi
```

Resultado: **19.59184**

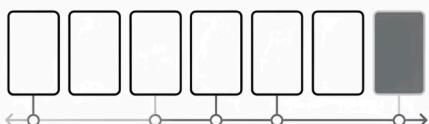
Este exemplo calcula o IMC (Índice de Massa Corporal) para um único indivíduo usando dados escalares.

## EXPANSÃO

# Evoluindo para Múltiplos Valores

Quando precisamos trabalhar com dados de várias pessoas, usamos vetores para armazenar múltiplos valores:

```
weight <- c(60, 72, 57, 90, 95, 72)  
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)  
subject <- c("A", "B", "C", "D", "E", "F")
```



Agora temos informações de **6 indivíduos diferentes**. Para calcular o IMC de cada um, precisamos de estruturas de repetição!

# Controle de Fluxo: Estruturas Condicionais (if, else)

Estruturas condicionais permitem que o programa **tome decisões** baseadas em condições lógicas. Uma decisão avalia para TRUE ou FALSE.

## Como funciona

Se a condição for verdadeira, um bloco de código é executado. Caso contrário, outro bloco pode ser executado.

```
if (condicao) {  
    # código executado se TRUE  
} else {  
    # código executado se FALSE  
}
```



### Avaliação

Condição é testada

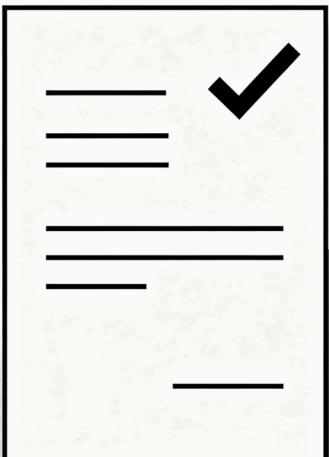
### Decisão

Um caminho é escolhido

### Execução

Apenas um bloco executa

## Exemplo de Decisão com if



O `if` avalia **uma única condição**. Apenas **um caminho** é seguido durante a execução. O fluxo do programa depende do valor atual da variável.

```
nota <- 7  
  
if (nota >= 6) {  
    resultado <- "aprovado"  
} else {  
    resultado <- "reprovado"  
}
```

**Condição: nota  $\geq 6$**

Se verdadeiro  $\rightarrow$  "aprovado"

**Caso contrário**

Se falso  $\rightarrow$  "reprovado"

## Condições em Vetores: if vs ifelse

Em R, muitas variáveis são **vetores** contendo múltiplos valores. Para trabalhar com eles, precisamos entender a diferença entre duas abordagens.

### if

**Controla o fluxo** do programa

Espera uma única condição lógica

Usado para decisões no código

### ifelse

**Processa vetores** elemento por elemento

Avalia condição para cada valor

Usado para transformar dados

**Comparação conceitual:** if → decisão única (controle do programa) | ifelse → escolha de valores elemento a elemento

## Exemplo de Condicional Vetorizada com ifelse

A função `ifelse` avalia uma condição para **cada elemento do vetor** e retorna um novo vetor com os valores escolhidos.

```
alturas <- c(1.65, 1.80, 1.55)
```

```
classe <- ifelse(  
  alturas < 1.70,  
  "baixa",  
  "alta"  
)
```

O vetor `classe` conterá: ["baixa", "alta", "baixa"]

## Repetição: Quando e Por Que Usar

### Operações Múltiplas

Quando a mesma operação precisa ser aplicada várias vezes



### Laços Explícitos

Repetição através de estruturas de controle como for e while

### Operações Vetorizadas

Forma mais eficiente em R - opera em conjuntos inteiros

- ❑ **Observação didática:** Em R, a repetição explícita existe, mas **nem sempre é a forma mais eficiente**. As operações vetorizadas são geralmente preferidas por serem mais rápidas e concisas.

## Relação entre Condição e Repetição

Estruturas condicionais e de repetição são **complementares** e trabalham juntas para criar programas poderosos e flexíveis.

**Condições**  
Decidem **se** um bloco deve ser executado



**Repetições**  
Executam blocos **múltiplas vezes**

**Controle**  
Decidem **quando** parar a repetição

**Ideia central:** Repetição executa várias vezes. Condição decide **se** e **quando** executar.

## O que é o for?

### Definição

A estrutura `for` é usada quando **sabemos previamente** quantas vezes o código deve ser executado.

### Aplicação

Ela percorre uma **sequência definida** como vetores, listas ou intervalos numéricos.

### Sintaxe básica

```
for (variavel in sequencia) {  
  # comandos a serem repetidos  
}
```

### Exemplo prático

```
for (i in 1:5) {  
  print(i)  
}
```

### Usos comuns

- Percorrer vetores e data frames
- Repetições com número fixo de iterações
- Aplicar cálculos elemento a elemento

→ O código imprime os números de 1 a 5.

### Vantagem

Simples e fácil de entender para iniciantes

### Limitação

Menos eficiente para grandes volumes de dados comparado a funções vetorizadas

## Calculando IMC com for

Vamos usar o `for` para calcular o IMC de todos os indivíduos no nosso vetor:

```
bmi <- 0  
for (i in 1:length(weight)) {  
  bmi[i] <- weight[i]/height[i]^2  
}  
bmi
```

- ❑ **Como funciona:** O loop percorre cada posição do vetor (de 1 até o comprimento do vetor de pesos), calcula o IMC para aquela posição e armazena no vetor `bmi`.

Resultado obtido:

```
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

## Inspecionando os Cálculos

Para entender o que acontece em cada iteração, podemos adicionar um `print()` dentro do loop:

```
bmi <- 0
for (i in 1:length(weight)) {
  bmi[i] <- weight[i]/height[i]^2
  print(bmi)
}
```

Isso nos mostra como o vetor `bmi` vai sendo preenchido gradualmente:

```
## [1] 19.59184
## [1] 19.59184 22.22222
## [1] 19.59184 22.22222 20.93664
## [1] 19.59184 22.22222 20.93664 24.93075
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

**Dica importante:** Usar `print()` dentro de loops é uma técnica valiosa para depuração e entendimento do código.

# Depurando no RStudio

## Criando a função

Defina sua função no editor

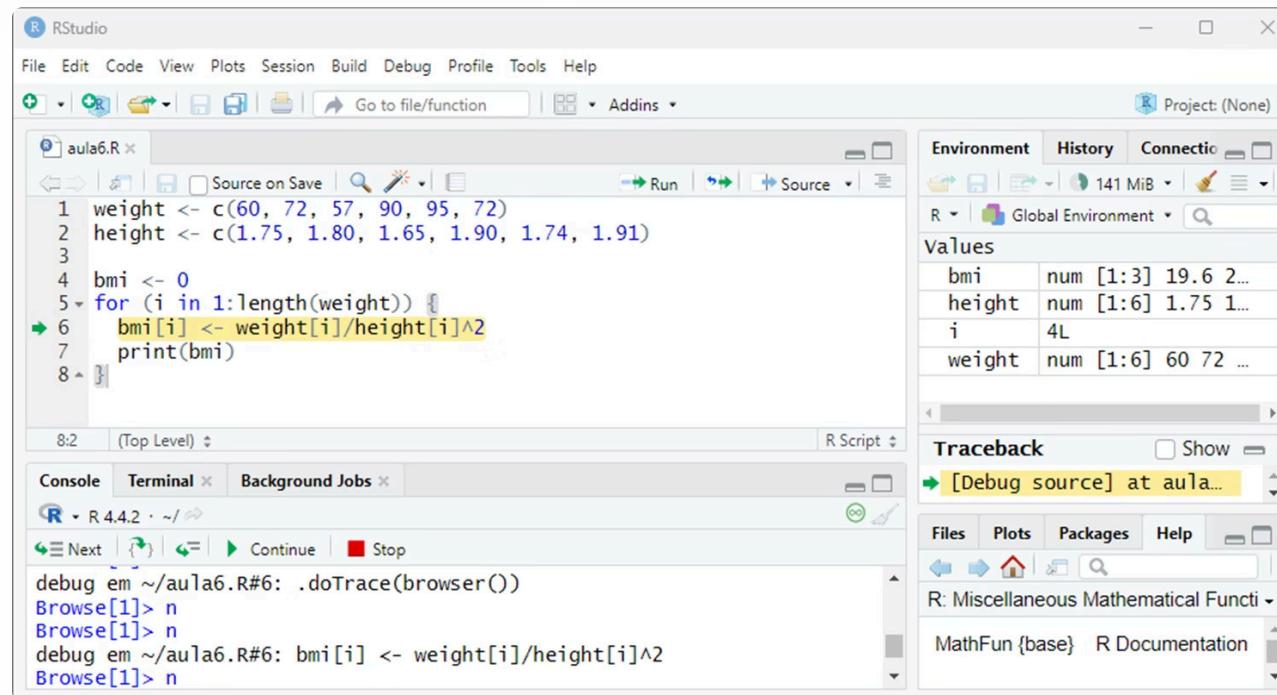
## Limpar variáveis

Clique na vassoura para limpar o ambiente

## Executando

Execute passo a passo para depurar

O RStudio oferece ferramentas visuais poderosas para depuração de código, permitindo inspecionar variáveis e acompanhar a execução linha por linha.



## Removendo Variáveis do Ambiente

Para remover variáveis e liberar memória, usamos a função rm():

```
rm(bmi)  
exists("bmi")
```

Resultado:

```
## [1] FALSE
```

A função exists() confirma que a variável bmi foi removida do ambiente de trabalho.

- Boa prática:** Manter seu ambiente limpo ajuda a evitar conflitos entre variáveis e facilita a organização do código.

## O que é o while?

### Definição

A estrutura `while` executa o código **enquanto uma condição for verdadeira**.

### Característica

O número de repetições **não é conhecido previamente** — depende da condição.

### Sintaxe básica

```
while (condicao) {  
    # comandos a serem repetidos  
}
```

### Exemplo prático

```
i <- 1  
while (i <= 5) {  
    print(i)  
    i <- i + 1  
}
```

### Usos comuns

- Processos com condições dinâmicas
- Simulações computacionais
- Algoritmos iterativos (convergência, otimização)

→ Imprime números de 1 a 5, controlando manualmente a variável.

 **Atenção:** Sempre garanta que a condição será alterada durante a execução, para evitar **loops infinitos** que travam o programa!

### FOR

Repetições com número **definido** de iterações

### WHILE

Repetições baseadas em condições dinâmicas

## Calculando IMC com while

Podemos usar `while` para obter o mesmo resultado do `for`, controlando manualmente o índice:

```
i <- 1  
bmi <- 0  
while (i <= length(weight)) {  
  bmi[i] <- weight[i]/height[i]^2  
  i <- i + 1  
}
```

01

### Inicialização

Criamos a variável de controle `i <- 1`

03

### Cálculo

Calculamos e armazenamos `bmi[i]`

Note que, diferentemente do `for`, precisamos **gerenciar manualmente** a variável de controle `i`.

02

### Condição

Verificamos se `i <= length(weight)`

04

### Incremento

Atualizamos `i <- i + 1` para próxima iteração

## Encapsulando Cálculo do IMC

Podemos criar uma função que usa `while` para calcular o IMC:

```
compute_bmi <- function(weight, height) {  
  i <- 1  
  bmi <- 0  
  while (i <= length(weight)) {  
    bmi[i] <- weight[i]/height[i]^2  
    i <- i + 1  
  }  
  return(bmi)  
}
```

```
bmi <- compute_bmi(weight, height)  
bmi
```

Resultado:

```
## [1] 19.59184 22.22222 20.93664  
##   24.93075 31.37799 19.73630
```

Encapsular código em funções torna seu programa mais **organizado, reutilizável e fácil de manter**.

## Implementando a Função do Jeito Certo

Na verdade, não precisamos de loops para esta tarefa! O R é uma linguagem **vetorizada**, permitindo operações diretas em vetores:

```
compute_bmi <- function(weight, height) {  
  resposta <- weight/height^2  
  return(resposta)  
}  
  
bmi <- compute_bmi(weight, height)  
bmi
```

Resultado:

```
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

### Mais eficiente

Operações vetorizadas são muito mais rápidas

### Mais legível

Código mais limpo e conciso

### Mais idiomático

Aproveita os recursos nativos do R

- Regra de ouro:** Use loops quando necessário, mas prefira operações vetorizadas sempre que possível!

## Uso da Função: Escalares e Vetores

### Com valores escalares

```
compute_bmi(80, 1.79)
```

Resultado:

```
## [1] 24.96801
```

A função calcula o IMC para um único indivíduo.

### Com vetores

```
compute_bmi(weight, height)
```

Resultado:

```
## [1] 19.59184 22.22222 20.93664  
##    24.93075 31.37799 19.73630
```

A mesma função processa múltiplos valores automaticamente!

### Flexibilidade

Uma única função serve para diferentes casos de uso

### Vetorização

O R aplica automaticamente a operação a todos os elementos

### Simplicidade

Sem necessidade de loops explícitos para operações simples

Esta é a **elegância do R**: funções bem escritas funcionam perfeitamente tanto com valores individuais quanto com vetores inteiros!

# Referências



Hands-On  
Programming  
with R

WRITE YOUR OWN FUNCTIONS AND SIMULATIONS

## Hands-on Programming

Aprenda R criando suas  
próprias funções e simulações

[https://rstudio-  
education.github.io/hopr/index.html](https://rstudio-education.github.io/hopr/index.html)

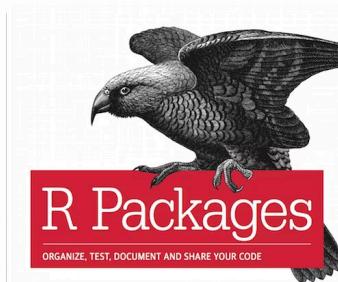


R Graphics  
Cookbook

## R Graphics Cookbook

Domine visualizações de dados  
em R

<https://r-graphics.org>



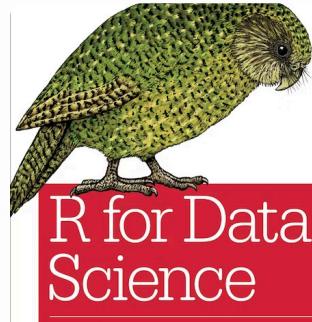
R Packages

ORGANIZE, TEST, DOCUMENT AND SHARE YOUR CODE

## R Packages

Desenvolva seus próprios  
pacotes R

<https://r-pkgs.org/index.html>



R for Data  
Science

VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA

## R for Data Science

Guia completo para ciência de  
dados

<https://r4ds.had.co.nz>