Erros comuns e exercícios

2025-10-12

Erros Comuns e Exercícios

Erros comuns

Estruturas de Controle (IF-ELSE, SWITCH)

1. Não Usar a Versão Vetorial: O erro mais clássico em R é tentar aplicar a estrutura if-else padrão (não vetorizada) a um vetor com mais de um elemento. O R testará apenas o primeiro elemento do vetor, emitirá um warning, e o resultado será inesperado para os demais elementos.
2. Lógica Incompleta no IF-ELSE IF: Em cadeias de IF-ELSE IF, esquecer que o R só avalia a próxima condição se a anterior for FALSA. Isso pode levar a erros lógicos, especialmente em intervalos numéricos, se não forem definidos corretamente (ex: usar > 10 e depois > 5 sem considerar a ordem).
3. Confundir o SWITCH do R: Em R, a função switch() avalia uma expressão e a compara com valores literais, retornando o valor da expressão correspondente. Em outras linguagens, é um bloco de controle de fluxo.

Estruturas de Repetição (FOR, WHILE, REPEAT)

1. Loops Infinitos no WHILE/REPEAT: Esquecer de incluir uma instrução dentro do bloco do loop que modifique a variável de controle, garantindo que a condição de parada será atingida.
2. Ineficiência por Falta de Vetorização: Usar loops FOR para realizar operações que o R pode fazer de forma muito mais rápida usando funções vetorizadas (como sum(), mean(), ou a família apply). Vetorização é quase sempre mais rápida em R.
3. Não Pré-alocar Espaço (Em Loops FOR): Quando se constrói um vetor ou lista dentro de um FOR, não inicializar o objeto com o tamanho final (e.g., rend <- numeric(n_meses) no seu exemplo). O R tem que realocar o objeto na memória a cada iteração, tornando o código muito mais lento para grandes quantidades de dados.

Funções

1. Falta de Tratamento de Exceções: Não incluir verificações de validade para os argumentos de entrada (e.g., o peso ou altura não podem ser negativos na função imc). O uso de stop() para entradas inválidas é fundamental para que sua função seja robusta.
2. Efeitos Colaterais (Side Effects): Modificar variáveis que estão fora do escopo da função (variáveis globais) de forma desnecessária. Isso torna o código difícil de rastrear e depurar.
3. Nomes Não Descritivos: Usar nomes de funções e argumentos que não comunicam claramente o propósito (e.g., f() em vez de calcular_media()).

Exercícios

Estruturas de Controle e Lógica Condicional (IF-ELSE e SWITCH)

1. Calculadora de Descontos Simples (IF-ELSE):

- Crie uma variável valor_compra.
- Use uma estrutura IF-ELSE IF aninhada para aplicar os seguintes descontos:
  - Se valor_compra for maior ou igual a *R$ 300,00*, aplique 15% de desconto.
  - Se valor_compra estiver entre *R$ 100,00* e *R$ 299,99*, aplique 5% de desconto.
  - Caso contrário (abaixo de R$ 100,00), não aplique desconto (0%).
- Imprima o valor final a ser pago.

Solução 1, para apenas um valor

```
valor_compra_i <- sample(10:1000, 1)

if ((valor_compra_i > 100) & (valor_compra_i < 299)) {
  valor_compra_f <- valor_compra_i * 0.95
  compra <- paste("Seu desconto foi de 5%, o valor original era: ", valor_compra_i, "seu novo valor é: "
} else if (valor_compra_i >= 300) {
  valor_compra_f <- valor_compra_i * 0.85
  compra <- paste("Seu desconto foi de 15%, o valor original era: ", valor_compra_i, "seu novo valor é:
} else {
  compra <- paste("Você não teve desconto. Seu valor é: ", valor_compra_i)
}

print(compra)
```

```
## [1] "Você não teve desconto. Seu valor é:  76"
```

Solução 2, para uma série de valores

```
valor_compra <- sample(10:1000, 6)

valor_final <- ifelse (((valor_compra > 100) & (valor_compra < 299)), valor_compra * 0.95, ifelse((valo

resultado <- data.frame(Valor_Original = valor_compra, Valor_Final = valor_final)
print(resultado)
```

```
##   Valor_Original Valor_Final
## 1            127      120.65
## 2            867      736.95
## 3            593      504.05
## 4             51       51.00
## 5             88       88.00
## 6            504      428.40
```

2. Mapeamento de Cores (SWITCH)

- Crie uma variável idioma com os valores "pt" (português), "en" (inglês) ou "es" (espanhol).
- Use a estrutura SWITCH para traduzir o nome de uma cor. Por exemplo, se a entrada for "red", o SWITCH deve retornar:
  - "pt": "vermelho"
  - "en": "red"
  - "es": "rojo"
- Defina um caso de escape que retorne "Idioma não suportado" se a entrada for diferente dos três idiomas válidos.

```
cor <- "vermelho"

idioma <- switch (cor,
  "vermelho" = "Português. Em inglês é red e espanhol rojo",
  "rojo" = "Espanhol Em inglês é red e português vermelho",
  "red" = "Inglês. Em português é vermelho e espanhol rojo",
  "Idioma não suportado"
)

print(idioma)
```

```
## [1] "Português. Em inglês é red e espanhol rojo"
```

3. Classificação Vetorial de Notas (ifelse ou case_when):

- Crie um vetor pontuacoes com 10 notas aleatórias entre 0 e 100 (e.g., sample(0:100, 10)).
- Usando a função dplyr::case_when() (ou ifelse() aninhado se não quiser usar o dplyr), crie um novo vetor conceito com base nas seguintes regras (Aplicação Vetorial!):
  - Notas 90: "A"
  - Notas 80 e <90: "B"
  - Notas 70 e <80: "C"
  - Notas <70: "D"
- Imprima o vetor original de notas e o vetor de conceitos.

```
notas <- sample(0:100, 10)

casos <- dplyr::case_when(notas >= 90 ~ "A",
                          notas >= 80 ~ "B",
                          notas >= 70 ~ "C",
                          TRUE ~ "D")

resultado <- data.frame(notas, casos)
print(resultado)
```

```
##    notas casos
## 1      6     D
## 2     29     D
## 3      0     D
## 4     13     D
## 5     32     D
## 6     78     C
## 7     70     C
## 8      2     D
## 9     49     D
## 10     8     D
```

Estruturas de Repetição (Loops)

4. Simulador de Juros Compostos (FOR):

- Defina um capital_inicial (e.g., 1000), uma taxa_juros (e.g., 0.05) e o numero_meses (e.g., 6).
- Crie um vetor saldo com numero_meses posições e pré-aloque o capital_inicial na primeira posição.
- Use um loop FOR para calcular e armazenar o saldo final para cada mês (mês a mês).
  - Dica: Saldo Atual = Saldo Anterior * (1 + taxa_juros)

```r
capital_inicial <- sample(500:2000, 1)
opcoes_juros <- c(0.01, 0.02, 0.03, 0.04, 0.05) # sample só cria sequências de números inteiros
taxa_juros <- sample(opcoes_juros, 1)
numero_meses <- sample(5:24, 1)

saldo <- numeric(numero_meses)
saldo[1] <- capital_inicial


for (i in 2:numero_meses) {
  saldo[i] <- saldo[i - 1] * (1 + taxa_juros)
}

resultado <- data.frame(Mês = 1:numero_meses, Capital_Inicial = capital_inicial, Taxa = taxa_juros, Sal
print(resultado)
```

```
##   Mês Capital_Inicial Taxa    Saldo
## 1   1            1571 0.01 1571.000
## 2   2            1571 0.01 1586.710
## 3   3            1571 0.01 1602.577
## 4   4            1571 0.01 1618.603
## 5   5            1571 0.01 1634.789
## 6   6            1571 0.01 1651.137
## 7   7            1571 0.01 1667.648
## 8   8            1571 0.01 1684.325
## 9   9            1571 0.01 1701.168
```

5. Adivinhe o Número (WHILE/REPEAT com break):

- Crie uma variável numero_secreto (e.g., 42).
- Use a função sample(1:100, 1) para gerar um palpite aleatório a cada tentativa.
- Use um loop WHILE (ou REPEAT) para continuar gerando palpites até que o palpite seja igual ao numero_secreto.
- Dentro do loop, use um contador de tentativas.
- Quando o número for adivinhado, use break para sair do loop e imprima o número total de tentativas necessárias.

```
numero_secreto <- sample(1:100, 1)
palpite <- sample(1:100, 1)
tentativas <- 1

while (palpite != numero_secreto) {
  tentativas <- tentativas + 1
  palpite <- sample(1:100, 1)
}

print(paste("O número secreto era: ", numero_secreto))
```

```
## [1] "O número secreto era:  56"
```

```
print(paste("O número foi descoberto em ", tentativas, "tentativas."))
```

```
## [1] "O número foi descoberto em  1 tentativas."
```

6. Soma de Ímpares (FOR com next):

- Crie um vetor numeros de 1 a 100.
- Crie uma variável soma_impares inicializada em 0.
- Use um loop FOR para iterar sobre o vetor.
- Dentro do loop, use uma estrutura IF para verificar se o número atual é par (numero %% 2 == 0).
- Se for par, use a instrução next para pular a adição.
- Se for ímpar, adicione o número à soma_impares.
- Imprima a soma_impares final.

```
numeros <- 1:100
soma_impares <- 0

for (i in numeros) {
  if (i %% 2 == 0) next
  soma_impares <- soma_impares + i
}

print(soma_impares)
```

```
## [1] 2500
```

Funções e Tratamento de Exceções

7. Função de Cálculo de Média Aritmética:

- Crie uma Função chamada calcular_media_vetor que aceite um argumento: vetor_dados.
- Tratamento de Exceção: Dentro da função, use stop() para verificar se o vetor_dados não é numérico (!is.numeric(vetor_dados)). Se não for, emita a mensagem de erro: "Erro: O argumento deve ser um vetor numérico.".
- A função deve retornar a média dos valores do vetor.
- Teste a função passando:
  - Um vetor numérico válido (e.g., c(10, 20, 30)).
  - Um vetor de caracteres (e.g., c("a", "b", "c")) para testar o stop().

```
calcular_media_vetor <- function(vetor_dados) {
  if (!is.numeric(vetor_dados)) stop("Erro: O argumento deve ser um vetor numérico.")
  media <- mean(vetor_dados)
  cat("Resultado do Cálculo \n")
  cat("\n")
  cat("Valores de Entrada:\n")
  print(vetor_dados)
  cat("\n")
  cat("Média: ", round(media, 2))
}

calcular_media_vetor(vetor_dados = sample(1:100, 10))
```

```
## Resultado do Cálculo
##
## Valores de Entrada:
##  [1] 36 13 99 21 48 60 81 88 67 45
##
## Média:  55.8
```

```
# calcular_media_vetor(vetor_dados = c("a", "b", "c", "d"))
```

8. Função de Classificação do Triângulo:

- Crie uma Função chamada classificar_triangulo que aceite três argumentos: lado_a, lado_b e lado_c.
- A função deve usar uma estrutura IF-ELSE IF para retornar:
  - "Equilátero": se todos os lados forem iguais.
  - "Isósceles": se apenas dois lados forem iguais.
  - "Escaleno": se todos os lados forem diferentes.

-Tratamento de Exceção: Antes da classificação, inclua uma regra simples para verificar se os lados formam um triângulo válido: A soma de dois lados deve ser maior que o terceiro. Se for inválido, use warning() com a mensagem "Aviso: Os lados não formam um triângulo válido." e retorne NA.

```r
classificar_triangulo <- function(lado_a, lado_b, lado_c) {
  if ( ((lado_a + lado_b) < lado_c) |
       ((lado_a + lado_c) < lado_b) |
       ((lado_c + lado_b) < lado_a)) {
    warning("Aviso: Os lados não formam um triângulo válido.")
    return(NA)
    }
  if ((lado_a == lado_b) & (lado_b == lado_c)) {
    print((paste("Equilátero"))) # todos iguais
  } else if (lado_a == lado_b | lado_a == lado_c | lado_b == lado_c) {
    print(paste("Isósceles")) # pelo menos dois iguais
  } else print(paste("Escaleno")) # todos diferentes
}

#---TESTES---
classificar_triangulo(lado_a = 3, lado_b = 4, lado_c = 5) # Escaleno
```

```
## [1] "Escaleno"
```

```r
classificar_triangulo(lado_a = 1, lado_b = 2, lado_c = 10) # NA
```

```
## Warning in classificar_triangulo(lado_a = 1, lado_b = 2, lado_c = 10): Aviso:
## Os lados não formam um triângulo válido.
```

```
## [1] NA
```

```r
classificar_triangulo(lado_a = 7, lado_b = 7, lado_c = 7) # Equilátero
```

```
## [1] "Equilátero"
```

```r
classificar_triangulo(lado_a = 7, lado_b = 5, lado_c = 7) # Isóceles
```

```
## [1] "Isósceles"
```

```r
classificar_triangulo(lado_a = sample(10:50, 1), lado_b = sample(10:50, 1), lado_c = sample(10:50, 1))
```

```
## [1] "Escaleno"
```