Capítulo

6 OPERAÇÕES DE REPETIÇÃO (LOOP)

As operações de repetição (**loop**) são a essência da computação em geral, pois foi justamente da necessidade de executar tarefas repetidas vezes (e de forma rápida) é que impulsionou o desenvolvimento e popularização dos computadores. Além disso, muitas dessas tarefas são maçantes e complexas, o que reforça a importância dessa forma de controle de fluxo.

O *loop* permite executar trechos do algoritmo quantas vezes for necessária, a quantidade de vezes pode ser defina a priori pelo programador, mas geralmente é determinada de acordo com o problema em questão. Cada repetição pode ser chamada de iteração[BOX3].

BOX 3: Iteração versus interação.
Iteração é diferente de interação! Iteração é um processo repetitivo, já interação diz respeito à comunicação em dois sentidos, isto é, que interage, como algum tipo de interface gráfica de software que interage com o usuário.

Geralmente, a quantidade de iterações é controlada com o auxílio de uma variável específica, que a cada iteração é incrementada, isto é, somar uma quantidade (geralmente unitária; igual a 1) ao valor atual dessa variável (`i <- i + 1`). Decrementos (`i <- i - 1`) também podem ser necessários, porém com menor frequência. Essa variável especial, que é

incrementada ou decrementada, também pode ser entendida como um contador (contador de iterações).

É possível executar processos repetitivos via programação sequencial, para isso, basta copiar (repetidamente) o trecho do algoritmo que requer repetição, porém essa estratégia pode ser impraticável conforme a quantidade de repetições. Além disso, com as cópias, o código aumenta de tamanho e complexidade, tornando o algoritmo difícil de ser entendido, corrigido, adaptado e mantido.

6.1 TIPOS DE LOOP

Existem dois tipos básicos de loop, os condicionais e os incondicionais. Os condicionais podem ser subdivididos em duas partes: em que a condição é avaliada inicialmente (**enquanto** - `while`) e em que é avaliada posteriormente (**repita** - `repeat`). Já os incondicionais, as instruções são repetidas até que toda uma sequência seja avaliada (**para toda sequência** - `for`).

Tipos principais de repetição (Loop)

while

```
Enquanto Condições Faça:
    Tarefas
```

```
i <- 1 # contador
N <- 5 # máximo de vezes
while(i <= N){
    cat("Bom dia R!\n")
    i <- i + 1 # incremento
}
```

repeat

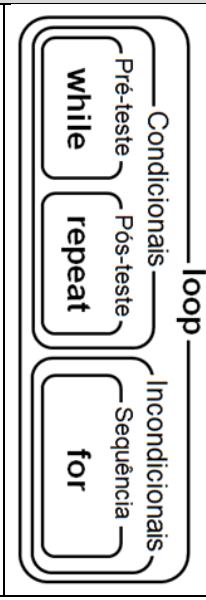
```
Repita:
    Tarefas
Até que Condições
```

```
i <- 1 # contador
N <- 5 # máximo de vezes
repeat{
    cat("Bom dia R!\n")
    if(i >= N) break
    i <- i + 1 # incremento
}
```

for

```
Para Toda Sequência Faça:
    Tarefas
```

```
N <- 5 # máximo de vezes
for(i in 1:N){
    cat("Bom dia R!\n")
}
```



A tabela acima apresenta as três formar de implementar um loop no R, utilizando as estruturas de controle: **`while`**, **`repeat`** e **`for`**. É importante dizer que é possível desenvolver qualquer algoritmo dominando apenas uma dessas formas, porém conforme o problema uma pode ser mais indicada (por ser mais lógico, natural, mais fácil de entender) em relação a outra.

No caso do R, o loop **`repeat`** requer a implementação explicita do teste condicional (utilizando a estrutura de controle condicional **`if`**), além disso é necessário a utilização da estrutura de controle auxiliar **`break`**, afim de interromper o processo repetitivo do loop quando determinada condição for atendida. Devido a este fato, no **`repeat`** do R, o teste condicional para

continuidade de repetição pode ser aplicado em qualquer momento do loop (início, meio ou fim). O **next** é uma estrutura de controle auxiliar, assim como **break**, porém ao invés de encerrar o loop, o **next** encerra apenas a iteração atual.

6.2 VETOR EM UM LOOP

Dentre os três tipos de loop, o **for** é um dos mais utilizados. A componente essencial do loop **for** é a sequência a ser utilizada, no R uma sequência é representada por um vetor[3]. Um vetor é uma estrutura de dados de apenas uma dimensão e composta de apenas um tipo de elemento, tal como números inteiros (**interger**), números reais (**numeric**), lógicos (**logical**), texto (**character**).

No R não existe conceito de escalar, assim: **a <- 123.4** é um vetor de apenas um elemento (vetor unitário) e **x <- c(10.4, 5.6, 3.1, 6.4, 21.7)** é um vetor de 5 elementos. A função **c()** serve para concatenar/combinar/juntar elementos. Após execução, retorna um vetor de comprimento/tamanho (**length**) correspondente à quantidade de elementos concatenados.

Existem diversas maneiras de gerar vetores sequenciais no R. Por exemplo, **1:30** gera um vetor **c(1, 2, ..., 29, 30)**. Pode-se também construir sequências decrescentes **30:1**. A

[3] Neste momento, uma descrição sucinta de vetor será feita. No capítulo sobre estrutura de dados, uma descrição mais completa será apresentada, também serão apresentadas outras estruturas.

função **seq** é a função mais geral do R para gerar sequências, e os principais argumentos dessa função são: **from = valor_inicial, to = valor_final, by = incremento e length = comprimento_da_sequência**. O comando **seq(from = 1, to = 30, by = 1)** gera o mesmo resultado do operador **":"** para **1:30**.

6.3 INDEXAÇÃO EM UM LOOP

Uma característica dos vetores é a possibilidade de acessar cada um de seus elementos via indexação, isto é, acessar o elemento de acordo com seu índice, que corresponde a posição do elemento no vetor. Por exemplo: No vetor **x <- c(10.4, 5.6, 3.1, 6.4, 21.7)**, para acessar o terceiro elemento, basta utilizar a instrução **x[3]** (nome do vetor - abre colchete - índice/posição do elemento - fecha colchete), e o resultado será o retorno do valor numérico **3.1**, já o quinto e último elemento, **x[5]**, que irá retornar o valor **21.7**. Ao tentar acessar um índice inexistente no vetor **x**, como um sexto elemento (**x[6]**), uma mensagem de erro será emitida e a execução do algoritmo interrompida.

6.4 EXEMPLOS DE LOOP

A seguir, exemplos de loop serão apresentados.

Exemplo de soma em um vetor

```
# Operador while
vetor <- c(1, 2, 34, 12, 24, 2)
soma <- 0
i <- 1
while(i <= length(vetor)){
    soma <- soma + vetor[i]
    i <- i + 1
}
print(soma)
```

```
# Operador repeat
vetor <- c(1, 2, 34, 12, 24, 2)
soma <- 0
i <- 1
repeat{
    soma <- soma + vetor[i]
    if(i >= length(vetor)) break
    i <- i + 1
}
print(soma)
```

```
# Operador for v1
vetor <- c(1,2,34,12,24,2)
soma <- 0
for(i in 1: length(vetor)){
    soma <- soma + vetor[i]
}
print(soma)
```

```
# Operador for v2
vetor <- c(1,2,34,12,24,2)
soma <- 0
for(i in vetor){
    soma <- soma + i
}
print(soma)
```

A versão 2 do loop **for** é uma simplificação da versão 1, e na prática geralmente essa simplificação é adotada. Observe que o **for** itera sobre um vetor, não necessariamente este vetor necessita ser uma sequência numérica regular de numéricos inteiros em ordem crescente, na verdade pode ser um vetor composto por elementos de qualquer tipo e organizados de qualquer forma. Isto, obviamente se o tipo de loop utilizado for a versão 2. No caso de necessitar utilizar a versão 1, uma sequência (que representa os índices do vetor) será necessária. O exemplo abaixo do loop **for** ilustra uma situação em que a versão 1 é mais indicada.

```
# Operador for v2
vetor <- c(1, 2, 34, 12, 24, 2)
info <- c("n", "n", "c", "n", "n", "n")
soma <- 0
for(i in 1: length(vetor)){
  if(info[i] != "c"){
    soma <- soma + vetor[i]
  }
}
print(soma)
```

Exemplo de valor máximo de um vetor

```
# Operador while
vetor <- c(1, 2, 34, 12, 24, 2)
i <- 1
maxi <- vetor[i]
while(i <= length(vetor)){
    if(vetor[i] > maxi){
        maxi <- vetor[i]
    }
    i <- i + 1
}
print(maxi)
```

```
# Operador repeat
vetor <- c(1, 2, 34, 12, 24, 2)
i <- 1
maxi <- vetor[i]
repeat{
    if(vetor[i] > maxi){
        maxi <- vetor[i]
    }
    if(i >= length(vetor)) break
    i <- i + 1
}
print(maxi)
```

```
# Operador for v1
vetor <- c(1, 2, 34, 12, 24, 2)
maxi <- vetor[1]
for(i in 1: length(vetor)){
    if(vetor[i] > maxi){
        maxi <- vetor[i]
    }
}
print(maxi)
```

```
# Operador for v2
vetor <- c(1, 2, 34, 12, 24, 2)
maxi <- vetor[1]
for(i in vetor){
    if(i > maxi){
        maxi <- i
    }
}
print(maxi)
```

O exemplo abaixo do loop **for** ilustra uma situação em que a versão 1 é mais indicada.

```
# Operador for v2
vetor <- c(1, 2, 34, 12, 24, 2)
info <- c("c", "n", "c", "n", "n", "n")

j <- 1
while(info[j] != "n") j <- j + 1
# Obs: os dados devem ter pelo menos uma observação normal (n)

maxi <- vetor[j]
for(i in 1: length(vetor)){
    if(info[i] != "c"){
       if(vetor[i] > maxi){
           maxi <- vetor[i]
       }
    }
}
print(maxi)
```

Exemplo de valor mínimo de um vetor

```
# Operador while
vetor <- c(1, 2, 34, 12, 24, 2)
i <- 1
mini <- vetor[i]
while(i <= length(vetor)){
    if(vetor[i] < mini){
        mini <- vetor[i]
    }
    i <- i + 1
}
print(mini)
```

```
# Operador repeat
vetor <- c(1, 2, 34, 12, 24, 2)
i <- 1
mini <- vetor[i]
repeat{
    if(vetor[i] < mini){
        mini <- vetor[i]
    }
    if(i >= length(vetor)) break
    i <- i + 1
}
print(mini)
```

```
# Operador for v1
vetor <- c(1, 2, 34, 12, 24, 2)
mini <- vetor[1]
for(i in 1: length(vetor)){
    if(vetor[i] < mini){
        mini <- vetor[i]
    }
}
print(mini)
```

```
# Operador for v2
vetor <- c(1, 2, 34, 12, 24, 2)
mini <- vetor[1]
for(i in vetor){
    if(i < mini){
        mini <- i
    }
}
print(mini)
```

O exemplo abaixo do loop **for** ilustra uma situação em que a versão 1 é mais indicada.

```
# Operador for v2
vetor <- c(1, 2, 34, 12, 24, 2)
info <- c("c", "n", "c", "n", "n", "n")

j <- 1
while(info[j] != "n") j <- j + 1
# Obs: os dados devem ter pelo menos uma observação normal (n)

mini <- vetor[j]
for(i in 1: length(vetor)){
    if(info[i] != "c"){
       if(vetor[i] < mini){
           mini <- vetor[i]
       }
    }
}
print(mini)
```