

INF 1514

Introdução à Análise de Dados

Material 5



Boas práticas de programação

- Um código precisa ser **simples** (fácil de entender), **eficiente** (faz o que é proposto) e **objetivo** (não dar “voltas”, não apresentar duplicidades, etc).
- Construir um programa não é somente escrever código e vê-lo funcionar. O código produzido deve ser **manutenível** uma vez que outras pessoas poderão alterá-lo.
- Se a linguagem (R, no caso) já possui suas **próprias bibliotecas**, com funções prontas, testadas e estressadas por diversas pessoas, é bem mais arriscado você criar o seu algoritmo do zero do que utilizar as bibliotecas.
- **Teste é fundamental!** Quando não se produz teste automatizado, a quantidade de testes manuais é maior e muitas vezes o custo desses testes também é maior.

Comente seu código

```
# Plotar o gráfico de y dado um intervalo de x.
# Parâmetro: inicial, valor inicial do eixo x.
# Parâmetro: final, valor final do eixo x.
# Parâmetro: intervalo, intervalo utilizado para calcular os valores de x.
# Retorna: média dos valores calculados para y.
plotaGrafico <- function (inicial, final, intervalo) {
  soma <- 0.0 # Acumula a soma dos valores de y.
  x <- seq (inicial, final, intervalo) # Vetor com os valores de x.
  y <- rep (0, length(x)) # Vetor com os valores de y.
  cont <- 0 # Contador auxiliar para calcular os valores de y.
  while (cont < length(x)) { # Percorre o vetor x e calcula o valor de y.
    cont <- cont + 1
    y[cont] <- calculaY(x[cont]) # calculaY retorna o valor de y dado um valor de x.
    soma <- soma + y[cont]
  }
  plot (x, y) # Plota o gráfico de x por y.
  return(soma/cont) # Retorna a média dos valores de y.
}
```

Comente seu código

- Os comentários devem ser **claros** e **objetivos**, pois se não o forem, podem prejudicar o entendimento ao invés de ajudar.
- **Comente antes de iniciar**, ou, no máximo, assim que terminar seu código. Além de você saber o que ele faz, você também se lembrará do motivo de estar escrevendo determinado código.
- Deixar para comentar funções, algoritmos, etc. **“para depois” não é uma boa prática**, pois, provavelmente, não o fará e, se o fizer, não ficará tão bom.
- As suas **funções** devem ser **comentadas antes mesmo de escrevê-las**, se possível for.
- É importante lembrar que um **comentário bem feito não irá esconder um código ruim** e que **um código bem feito reduz a necessidade de comentários**.

Valide os argumentos das suas funções

- Por exemplo, se você criar uma **função que recebe uma variável do tipo data** e com base nela a função calcula quando será a próxima data de vencimento de um parcelamento, a função deve se certificar de que a variável data **não é nula e está no formato de data**.
- Uma **função que recebe o valor de um depósito em dinheiro** e faz o lançamento na conta de um cliente deve **verificar se o valor fornecido como parâmetro é maior do que “0”**.

Nomes significativos

- Seja **coerente** e **intuitivo** com a nomenclatura utilizada ao longo de todo o código, evitando nomear variáveis como “x” e “y”, ao invés de “nome” e “data”, por exemplo.
- Funções, variáveis e etc. devem possuir **nomes que tenham significado** em relação ao seu objetivo.
- Nomes grandes podem ser utilizados, contanto que fique **compreensível**.
- **Evite utilizar nomes impronunciáveis** para funções e variáveis: **calcImpRen**, melhor seria **calculaImpostoRenda**.

Indentação

- **Indentar significa alinhar o código**, inserir espaços em determinados trechos, etc. Isso **facilita a compreensão** do que foi feito e ajuda também no processo de **depurar o código**.
- Sempre que iniciar uma estrutura condicional, de repetição, retornos, funções, etc, coloque as **delimitações do corpo entre chaves**, mesmo quando o corpo só apresentar um comando/expressão.
- **Muitos níveis de indentação devem ser evitados**, assim como **muitas condicionais múltiplas, laços de múltiplos níveis**, etc. Para resolver esses casos, **devem ser criadas funções** que tornarão o código mais legível.
- Deixar **espaços** entre os **sinas de operação**.
- **Estabeleça** um critério de espaços nos **recuos** (3, 4, e 5 são bons valores).

Indentação

```
plotaGrafico <- function (inicial, final, intervalo) {  
  soma <- 0.0  
  x <- seq (inicial, final, intervalo)  
  y <- rep (0, length (x))  
  cont <- 0  
  
  while (cont < length (x))  
  {  
    cont <- cont + 1  
    y[cont] <- calculaY (x[cont])  
    soma <- soma + y[cont]  
  }  
  plot (x, y)  
  
  return (soma/cont)  
}
```

Apresenta boa prática!

```
plotaGrafico <- function (inicial, final, intervalo) {  
  soma<-0.0  
  x<-seq (inicial, final, intervalo)  
  y<-rep (0, length(x))  
  cont<-0  
  while(cont<length(x)) {  
    cont<-cont + 1  
    y[cont]<-calculaY(x[cont])  
    soma<-soma+y[cont]  
  }  
  plot(x,y)  
  return(soma/cont)  
}
```

NÃO apresenta boa prática!

Cenários negativos devem ser tratados

- O “**e se**” **não pode ser ignorado**, é preciso garantir que o código vai ter um tratamento para cada situação.
- **Devem ser avaliados os diferentes cenários** em que a execução do código pode retornar um erro.
- Perguntas que **devem sempre ser avaliadas**:
 - A variável está no **formato correto**?
 - Existe algum **limitador de valor** para a variável?
 - As **casas decimais** podem afetar o processamento?
 - E se houver **nulos**?

Outras boas práticas

- Declare cada **variável em uma linha** para facilitar a escrita de comentários.
- Declare as **variáveis sempre no início** das funções e do script principal.
- Coloque **uma linha em branco** entre a declaração de variáveis e os comandos seguintes.
- Escreva **parênteses**, ainda que redundantes, para facilitar o entendimento de expressões aritméticas.
- Quebre um **comando longo** em comandos menores e mais simples.
- Escreva **um comando por linha**.
- **Inicialize as variáveis ao declará-las**, sempre que possível.

Estruturas de controle

- A Linguagem R, como qualquer linguagem de programação, possui instruções destinadas a **alterar o curso sequencial de execução** dos programas.
- **Como já visto**, para executar parte do código somente se determinada condição for **verdadeira**, utilizamos a instrução ***if***, incluindo a condição a ser testada entre os parênteses. Se houve algum código alternativo a ser executado no caso da condição ser **falsa**, podemos informar isso para o R com a instrução ***else***.
- Agora, como fazer para:
 - contar de 1 a 50.
 - executar uma ou mais instruções 100 vezes.
 - executar uma ou mais instruções repetidas vezes até que um critério de parada seja alcançado.

Execução condicional – *if / else if / else*

```
calcularImposto = function (salarioPessoa) {  
  imposto <- 0.0  
  
  if (salarioPessoa <= 1903.98)  
  {  
    imposto <- 0  
  }  
  else if(salarioPessoa <= 3751.05)  
  {  
    imposto <- salarioPessoa * (15/100) - 130  
  }  
  else  
  {  
    imposto <- salarioPessoa * (27.5/100) - 330  
  }  
  
  return(imposto)  
}
```

Instrução iterativa

- O R tem várias instruções iterativas que nos permitem repetir blocos de instruções.

```
....  
while (condição booleana)  
    <bloco de instruções>  
...
```

```
....  
for (<var> in <conjunto>)  
    <bloco de instruções>  
...
```

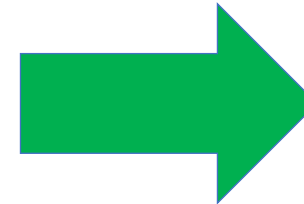
```
....  
repeat  
    <bloco de instruções>  
...
```

Instrução iterativa – *while*

- A sua semântica pode ser descrita por: **enquanto a condição booleana for verdadeira, repita o bloco de instruções.**

```
....  
while (condição booleana)  
  <bloco de instruções>  
...
```

```
x <- 0  
while (x <= 5) {  
  x <- x + 1  
  print (x)  
}
```



[1]	1
[1]	2
[1]	3
[1]	4
[1]	5
[1]	6

- As instruções no bloco dentro do **while** podem nunca ser executadas, bastando para isso que a condição seja **falsa da primeira vez** em que é avaliada.
- Um erro relativamente frequente consiste em escrever instruções iterativas que levam a **loops (ciclos) que nunca terminam.**

Exercício 1

- Escreva uma função chamada **somaValor** que recebe um número inteiro ***n*** como parâmetro e retorna a soma dos números pares até ***n*** utilizando a instrução **while**.

Exercício 1

- Escreva uma função chamada **somaValor** que recebe um número inteiro ***n*** como parâmetro e retorna a soma dos números pares até ***n*** utilizando a instrução **while**.

```
somaValor <- function(n)
{
  soma <- 0
  x <- 0

  while (x < n)
  {
    x <- x + 1
    if ((x %% 2) == 0)
    {
      print (x)
      soma <- soma + x
    }
  }

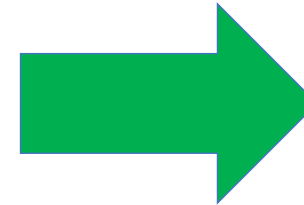
  return (soma)
}
```


Instrução iterativa – *repeat*

- A instrução **repeat** permite **executar** um **bloco de instruções uma ou mais vezes**.

```
....  
repeat  
  <bloco de instruções>  
...
```

```
x <- 0  
repeat  
{  
  x <- x + 1  
  print (x)  
  if (x > 5)  
  {  
    break  
  }  
}
```



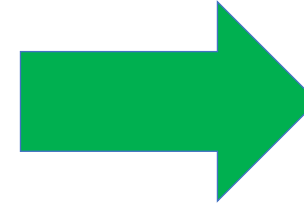
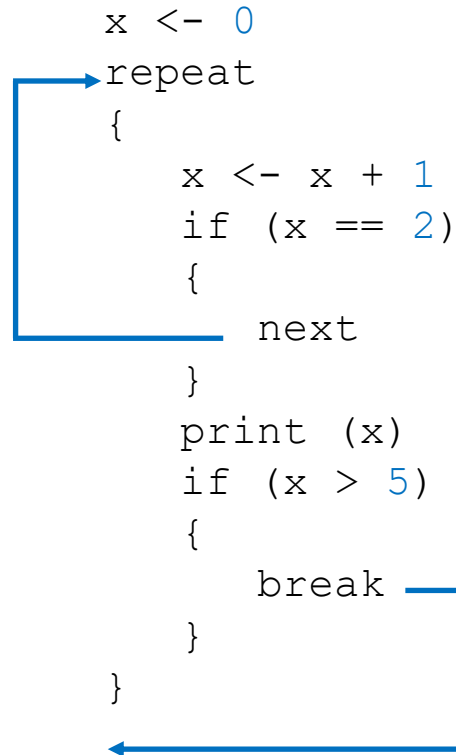
```
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] 6
```

- A instrução **break** dentro do loop faz o R **terminar a repetição** da execução do **bloco de instruções**.
- Um erro **relativamente frequente** consiste em escrever instruções iterativas que levam a **loops (ciclos) que nunca terminam**.

Instrução iterativa – *repeat*

```
....  
repeat  
  <bloco de instruções>  
...
```

```
x <- 0  
repeat  
{  
  x <- x + 1  
  if (x == 2)  
  {  
    next  
  }  
  print (x)  
  if (x > 5)  
  {  
    break  
  }  
}
```



```
[1] 1  
[1] 3  
[1] 4  
[1] 5  
[1] 6  
.
```

- A instrução **next** dentro do loop faz o R **saltar para o início** da execução do **bloco de instruções**.

Exercício 2

- Escreva uma função chamada **somaValor** que recebe um número inteiro ***n*** como parâmetro e retorna a soma dos números pares até ***n*** utilizando a instrução **repeat**.

Exercício 2

- Escreva uma função chamada **somaValor** que recebe um número inteiro ***n*** como parâmetro e retorna a soma dos números pares até ***n*** utilizando a instrução **repeat**.

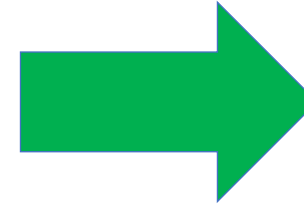
```
somaValor <- function (n)
{
  soma <- 0
  x <- 0
  repeat
  {
    x <- x + 1
    if ((x %% 2) == 0)
    {
      print (x)
      soma <- soma + x
    }
    if (x == n)
    {
      break
    }
  }
  return (soma)
}
```

Instrução iterativa – *for*

- O **for** permite controlar o **número de vezes** que um bloco de instruções é **executado** através de uma **variável de controle** que assume de **valores pré-definidos** em cada iteração do loop.

```
....  
for (<var> in <conjunto>)  
  <bloco de instruções>  
...
```

```
x <- 1:6  
for (i in x) {  
  print (i)  
}
```



```
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] 6
```

- Também no **for**, a instrução **break** dentro do loop faz o R **terminar a repetição** da execução do **bloco de instruções**.
- Um erro **relativamente frequente** consiste em escrever instruções iterativas que levam a **loops (ciclos) que nunca terminam**.

Exercício 3

- Escreva uma função chamada **somaValor** que recebe um número inteiro ***n*** como parâmetro e retorna a soma dos números pares até ***n*** utilizando a instrução **for**.

Exercício 3

- Escreva uma função chamada **somaValor** que recebe um número inteiro ***n*** como parâmetro e retorna a soma dos números pares até ***n*** utilizando a instrução **for**.

```
somaValor <- function (n)
{
  soma <- 0
  x <- 1:n

  for (i in x)
  {
    if ((x %% 2) == 0)
    {
      print (i)
      soma <- soma + i
    }
  }

  return (soma)
}
```