INF 1514

Introdução à Análise de Dados

Material 4

PUC
RIO

Qual o papel de uma função?

- Existem **procedimentos** que são **repetidos** diversas vezes em um script.
  - Encontrar o maior/menor valor em uma lista;
  - Encontrar a média de cada coluna de uma matriz;
  - Fazer o resumo estatístico de uma variável.
- Nestes casos é possível automatizar o processo através de uma **função**.
- Com isso o código fica mais **enxuto**, **simples** e **legível**.
- Outro ponto **positivo** é que dada uma alteração na função **não é necessário** alterar o código em **diversos pontos**.

Algumas funções internas do R

Função	Descrição
sqrt()	Raiz quadrada.
abs()	Valor absoluto.
exp()	Exponencial.
log10()	Logaritmo na base 10.
log()	Logaritmo na base e.
sin() cos() tan()	Funções trigonométricas.
asin() acos() atan()	Funções trigonométricas inversas.
sort()	Ordena os elementos de um vetor.
read.csv()	Lê um arquivo no padrão csv.
max(x)	Retorna o maior valor encontrado em x (vetor, etc).
mean(x)	Retorna a média dos elementos de x (vetor, etc).
tolower()	Reescreve uma string em letras minúsculas.
hist()	Cria um histograma.

Algumas funções internas do R

- Usando o console do R encontre os seguintes valores:
  - Raiz quadrada de 625;

```
sqrt(625)
```

  - O valor absoluto de -68;

```
abs(-68)
```

  - O seno de 3π.

```
sin(pi/2)
```

  - A exponencial de 5.

```
exp(5)
```

Definindo funções

```
nome_da_função <- function (parâmetro_1, parâmetro_2, etc...)
{
  corpo_da_função
}
```

- O **nome da função** deve seguir as mesmas regras de nomeação de variáveis, como, por exemplo, não começar com números.
- **Funções** podem ou não receber **parâmetros**.
- O **corpo da função** normalmente possui:
  - Declaração de variáveis;
  - Lógica de processamento;
  - Valor de retorno.
- Um comando **não obrigatório**, mas que é bastante comum no final das funções, é o **return()**.
- A maioria das linguagens de programação possui diversas funções **pré-programadas** (internas) que podemos utilizar em nossos programas.

Organização básica de um script R

Funções

Corpo principal

```
nome_da_função_1 <- function (parâmetro_1, parâmetro_2, etc...)
{
  corpo_da_função
}
nome_da_função_2 <- function (parâmetro_1, parâmetro_2, etc...)
{
  corpo_da_função
}

...
# variáveis
# início


# fim
```

Exemplo 1

- Sabe-se que, em um determinado período, o salário mínimo aumentou de R$1.212,00 para R$1.302,00. Considerando que a inflação desse mesmo período foi de 5,8%, qual o aumento real do salário mínimo?

Se i_n é a taxa nominal de juros, i_r, a taxa real de juros e θ a taxa de inflação, todas referidas ao mesmo período de tempo, então:

$$(1 + i_n) = (1 + i_r)(1 + \theta)$$

$$i_r = \frac{(1 + i_n)}{(1 + \theta)} - 1$$

Onde $i_n = \frac{\text{valor do novo salário mínimo}}{valo\ do\ salário\ mínimo\ anterior}$ - 1 .

Exemplo 1 – Resolução

- Sabe-se que, em um determinado período, o salário mínimo aumentou de R$1.212,00 para R$1.302,00. Considerando que a inflação desse mesmo período foi de 5,8%, qual o aumento real do salário mínimo?

Se i_n é a taxa nominal de juros, i_r, a taxa real de juros e θ a taxa de inflação, todas referidas ao mesmo período de tempo, então:

$$(1 + i_n) = (1 + i_r)(1 + \theta)$$

$$i_r = \frac{(1 + i_n)}{(1 + \theta)} - 1$$

Onde $i_n = \frac{\text{valor do novo salário mínimo}}{valo\ do\ salário\ mínimo\ anterior}$ - 1 .

```
# variáveis
salario_anterior <- 0.0
novo_salario <- 0.0
taxa_inflacao <- 0.0
taxa_nominal <- 0.0
taxa_real <- 0.0

# início
salario_anterior <- 1212.0
novo_salario <- 1302.0
taxa_inflacao <- 0.058 # 5,8%
taxa_nominal <- novo_salario/salario_anterior - 1
taxa_real <- (1 + taxa_nominal)/(1 + taxa_inflacao) - 1
print(taxa_real)
# fim
```

Exemplo 2

- Elabore uma função chamada **calculaTaxaRealSalario** que calcula a taxa real de juros da variação do valor do salário mínimo dados como parâmetros: o valor do salário mínimo anterior, o valor do novo salário mínimo e a taxa de inflação no período. Teste a função elaborada usando os dados do Exemplo 1.

Exemplo 2 – Resolução

- Elabore uma função chamada **calculaTaxaRealSalario** que calcula a taxa real de juros da variação do valor do salário mínimo dados como parâmetros: o valor do salário mínimo anterior, o valor do novo salário mínimo e a taxa de inflação no período. Teste a função elaborada usando os dados do Exemplo 1.

```
calculaTaxaRealSalario <- function (salario_anterior, novo_salario, taxa_inflacao) {
    # variáveis
    taxa_nominal <- 0.0
    taxa_real <- 0.0

    # início
    taxa_nominal <- novo_salario/salario_anterior - 1
    taxa_real <- (1 + taxa_nominal)/(1 + taxa_inflacao) - 1
    return(taxa_real)
    # fim
}
```

Nome da função.

Parâmetros da função.

Cláusula **function** usada para criação da função.

Retorno da função.

Exemplo 2 – Resolução

Função

```
calculaTaxaRealSalario <- function (salario_anterior, novo_salario, taxa_inflacao) {
   # variáveis
   taxa_nominal <- 0.0
   taxa_real <- 0.0

   # início
   taxa_nominal <- novo_salario/salario_anterior - 1
   taxa_real <- (1 + taxa_nominal)/(1 + taxa_inflacao) - 1
   return(taxa_real)
   # fim
}
```

Corpo principal

```
# variáveis
salario_anterior <- 1212.0
novo_salario <- 1302.0
taxa_inflacao <- 0.058
taxa_real <- 0.0
# início
taxa_real <- calculaTaxaRealSalario (salario_anterior, novo_salario, taxa_inflacao)
print (taxa_real)
# fim
```

Usando os dados do Exemplo 1 para testar a função.

Exemplo 2 – Resolução (outra forma de resolver)

Função

```
calculaTaxaRealSalario <- function (salario_anterior, novo_salario, taxa_inflacao) {
   # variáveis
   taxa_nominal <- 0.0
   taxa_real <- 0.0

   # início
   taxa_nominal <- novo_salario/salario_anterior - 1
   taxa_real <- (1 + taxa_nominal)/(1 + taxa_inflacao) - 1
   return(taxa_real)
   # fim
}
```

Corpo principal

```
# variáveis
anterior <- 1212.0
novo <- 1302.0
inflacao <- 0.058
taxa <- 0.0
# corpo principal
taxa <- calculaTaxaRealSalario (anterior, novo, inflacao)
print (taxa)
# fim
```

Usando os dados do Exemplo 1 para testar a função.

Não é necessário que as variáveis possuam o mesmo nome dos parâmetros da função.

Como funciona a chamada de uma função?

1. Os valores passados na chamada são **copiados** para os **parâmetros da função** seguindo a **ordem definida (salario_anterior, novo_salario, taxa_inflacao)**.
2. A função **calcularTaxaRealSalario** executa a sua lógica.
3. A função **calcularTaxaRealSalario** retorna o valor da taxa **return(taxa_real)**.
4. O valor retornado é armazenado na variável **taxa**.

```
# variáveis
anterior <- 1212.0
novo <- 1302.0
inflacao <- 0.058
taxa <- 0.0
# início
taxa <- calculaTaxaRealSalario (anterior, novo, inflacao)
print (taxa)
# fim
```

Exercício 1

- Sabe-se que em uma determinada loja o **preço final** que um cliente paga por um **produto** é definido com base no **preço do produto** e no seguinte **conjunto de regras**:
  - Se o pagamento for **à vista (parcela única)**, o cliente terá **10% de desconto**.
  - À prazo em **duas vezes iguais** sendo uma parcela para **30** e outra para **60** dias, sofrerá **acréscimo de 5%**.
  - À prazo em **três vezes iguais** sendo **uma parcela** para **30**, outra para **60** e uma terceira para **90** dias sofrerá **acréscimo de 10%**.

- Elabore uma função chamada **calculaPrecoFinal** para calcular o preço final pago pelo cliente dado o número de parcelas e o preço do produto.
- Teste a sua função para diferentes valores de parcelas e preço do produto.

Exercício 1 – Resolução

```
calculaPrecoFinal <- function (parcelas, preco) {
   # variáveis
   precofinal <- 0.0

   # início
   if (parcelas == 1) {
     precofinal <- preco * 0.9
   } else {
     if (parcelas == 2) {
       precofinal <- preco * 1.05
     } else {
       precofinal <- preco * 1.1
     }
   }
   return(precofinal)
   # fim
}
```

Exercício 1 – Resolução

```
calculaPrecoFinal <- function (parcelas, preco) {
   # variáveis
   precofinal <- 0.0

   # início
   if (parcelas == 1) {
     precofinal <- preco * 0.9
   } else {
     if (parcelas == 2) {
       precofinal <- preco * 1.05
     } else {
       precofinal <- preco * 1.1
     }
   }
   return(precofinal)
   # fim
}

numero_parcelas <- 1
preco_produto <- 100
preco_final <- calculaPrecoFinal(numero_parcelas, preco_produto)
print(preco_final)
```

Parcelas	Preço (R$)	Preço final (R$)
1	100	90
2	100	105
3	120	132

Exercício 1 – Resolução (outra forma de resolver)

```
calculaPrecoFinal <- function (parcelas, preco) {
   # variáveis
   precofinal <- 0.0

   # inicio
   if (parcelas == 1) {
      precofinal <- preco * 0.9
   } else if (parcelas == 2) {
      precofinal <- preco * 1.05
   } else {
      precofinal <- preco * 1.1
   }
   return(precofinal)
   # fim
}
```

Parcelas	Preço (R$)	Preço final (R$)
1	100	90
2	100	105
3	120	132

As duas funções resolvem o mesmo problema, mas foram implementadas de forma diferente.

Exercício 1 – Resolução (outra forma de resolver)

```
calculaPrecoFinal <- function (parcelas, preco) {
   # variáveis
   precofinal <- 0.0

   # inicio
   if (parcelas == 1) {
      precofinal <- preco * 0.9
   } else if (parcelas == 2) {
      precofinal <- preco * 1.05
   } else {
      precofinal <- preco * 1.1
   }
   return(precofinal)
   # fim
}

numero_parcelas <- 1
preco_produto <- 100
preco_final <- calculaPrecoFinal(numero_parcelas, preco_produto)
print(preco_final)
```

Parcelas	Preço (R$)	Preço final (R$)
1	100	90
2	100	105
3	120	132

Exercício 2

- A tabela abaixo apresenta o critério de classificação de empresas utilizado por um analista. Escreva uma função que retorna a classificação da empresa em função da dívida e do faturamento anual médio passados como parâmetros.

Dívida (milhões)	Faturamento Anual Médio (milhões)	Classificação
Acima de 8.9		F
8.0 a 8.9		E
7.0 a 7.9	Menor ou igual a 2.0	E
7.0 a 7.9	Acima de 2.0	D
6.0 a 6.9	Menor ou igual a 1.0	D
6.0 a 6.9	Acima de 1.0	C
5.0 a 5.9		B
Menor que 5.0		A

Exercício 2 – Resolução

Dívida (milhões)	Faturamento Anual Médio (milhões)	Classificação
Acima de 8.9		F
8.0 a 8.9		E
7.0 a 7.9	Menor ou igual a 2.0	E
7.0 a 7.9	Acima de 2.0	D
6.0 a 6.9	Menor ou igual a 1.0	D
6.0 a 6.9	Acima de 1.0	C
5.0 a 5.9		B
Menor que 5.0		A

```
classificaEmpresa = function(divida, faturamento) {
  classificacao <- ""

  if (divida > 8.9){
    classificacao <- "F"
  }
  else if (divida >= 8.0){
    classificacao <- "E"
  }
  else if ((divida >= 7.0) & (faturamento <= 2.0)){
    classificacao <- "E"
  }
  else if (divida >= 7.0){
    classificacao <- "D"
  }
  else if ((divida >= 6.0) & (faturamento <= 1.0)){
    classificacao <- "D"
  }
  else if (divida >= 6.0){
    classificacao <- "C"
  }
  else if (divida >= 5.0){
    classificacao <- "B"
  }
  else {
    classificacao <- "A"
  }
  return(classificacao)
}
```

Uma função como parâmetro da outra

```
calculaPrecoFinal <- function (parcelas, preco) {
   # variáveis
   precofinal <- 0.0

   # início
   if (parcelas == 1) {
     precofinal <- preco * 0.9
   } else {
     if (parcelas == 2) {
       precofinal <- preco * 1.05
     } else {
       precofinal <- preco * 1.1
     }
   }
   return(precofinal)
   # fim
}

numero_parcelas <- 1
preco_produto <- 100
print(calculaPrecoFinal(numero_parcelas, preco_produto))
```

O retorno da função **calculaPrecoFinal** é parâmetro da função interna **print()**.

Como fica o escopo das variáveis?

- Uma **função não enxerga** (não tem acesso) as **variáveis de outra função**.
- A **passagem de dados** para, ou entre funções, é feita **através de parâmetros**.
- As funções **podem** ou **não retornar valores** e, no caso de retorno, estes ficam disponíveis para utilização de quem chamou a função somente após o término da execução da função.
- **Parâmetros** e **variáveis declaradas no corpo** da função **não existem fora** da função. Em termos computacionais, eles são criadas **a cada vez** que a função é chamada, **deixando de existir** quando a função **termina sua execução**.
- Variáveis definidas como **globais** no corpo principal do programa (script) podem ser acessadas **dentro e fora das funções**.

Boas práticas no uso de funções

- **Familiarize-se** com o conjunto de funções e procedimentos da linguagem.
- **Não use** como nome da função os **mesmos nomes de parâmetros** que a função usa e **nomes de variáveis passadas** à função.
- Sempre escolha nomes de funções e parâmetros que possuam **significado condizente** com o que se pretende resolver com a função.
- Usando uma analogia de um problema que seja a construção de uma casa, **não crie** uma **única função** que **cria toda a casa**. Mas sim, faça uma função que cria quarto, outra que cria banheiro, outra que cria sala, outra que cria jardim, outra que cria cozinha, etc. Por fim, faça uma função que invoca as funções anteriores, montando, assim, sua casa.

Exemplo 3

- Elabore uma função chamada **calculaTaxaRealSalarioCenario** que calcula a taxa real de juros da variação do valor do salário mínimo dados como parâmetros: o valor do salário mínimo anterior, o valor do novo salário mínimo, a taxa de inflação no período e o cenário (“A”, “B” e “C”). Considere que no cenário “A” a taxa de inflação usada no cálculo da taxa real de juros é o próprio valor passado como parâmetro para a função e que para os cenários “B” e “C” deve ser utilizada uma taxa de inflação ajustada em 10% para cima e 10% para baixo, respectivamente. Se o cenário passado para a função não for um cenário válido, a função deverá retornar o valor **NULL**. A função **calculaTaxaRealSalarioCenario** deverá utilizar a função **calculaTaxaRealSalario** criada no Exemplo 2. Teste a função elaborada usando diferentes valores para os parâmetros.

Exemplo 3 – Resolução

```
calculaTaxaRealSalarioCenario <- function (anterior, novo, inflacao, cenario) {
   # variáveis
   inflacao_ajustada <- 0.0
   taxa_real <- 0.0

   # início
   if (toupper(cenario) == "A") {
      inflacao_ajustada = inflacao
   } else if (toupper(cenario) == "B") {
      inflacao_ajustada = inflacao * 1.1
   } else if (toupper(cenario) == "C") {
      inflacao_ajustada = inflacao * 0.9
   } else {
      return(NULL)
   }
   taxa_real <- calculaTaxaRealSalario(anterior, novo, inflacao_ajustada)
   return(taxa_real)
   # fim
}
```

A função interna **toupper()** retorna o valor do parâmetro **cenario** em maiúsculo.

Exemplo 3 – Resolução

```
anterior <- 1212.0
novo <- 1302.0
inflacao <- 0.058
cenario <- "A"
taxa_real <- calculaTaxaRealSalarioCenario(anterior, novo, inflacao, cenario)
print(taxa_real)
```

Anterior (R$)	Novo (R$)	Inflação (%)	Cenário	Taxa real (%)
1212.0	1302.0	5,8	A	1.54
1212.0	1302.0	5,8	B	0.98
1212.0	1302.0	5,8	C	2.10
1200.0	1310.0	6,5	A	2.50
1200.0	1310.0	6,5	F	NULL

Execução condicional

- Como visto anteriormente, a linguagem R dispõe de execução condicional da seguinte forma:
  - **if** (condição) expres_1 **else** expres_2.
- Onde o **resultado da condição** deve ser um valor lógico **TRUE** ou **FALSE**, e, se este for verdadeiro, será executada a expres_1; caso contrário, será executada a expres_2. O uso do condicional **else** não é obrigatório.
- Existe uma **versão vetorizada** da construção condicional **if/else**, que é a função **ifelse**, cuja sintaxe no R é:
  - **ifelse** (condição, expres_1, expres_2).
- A expres_1 será executada, caso a condição seja verdadeira, e a expres_2, caso contrário.

```
x <- 8
y <- ifelse (x >= 5, x + 3, x - 4)
print(y)
```

Funções envolvendo ciclos

- Ciclos são **processos iterativos** nos quais sua função irá executar uma **sequência de comandos** até uma **condição previamente estabelecida**.
- É importante, nestes casos, que as iterações tenham uma **condição finita**.
- Comandos como **while(condição)** e **for(condição)** permitem a criação de ciclos no R.
- Para o exemplo do comando **for** abaixo, o termo **(auxiliar in 1:10)** quer dizer que variável **auxiliar** irá variar de 1 a 10.

```
for (auxiliar in 1:10) {
   print(auxiliar)
}
```

```
x <- 1:10
for (auxiliar in x) {
   print(auxiliar)
}
```

```
auxiliar <- 1
while (auxiliar <= 10) {
   print(auxiliar)
   auxiliar <- auxiliar + 1
}
```

- A execução dos scripts acima irão apresentar na tela os números de 1 a 10.

Exemplo 4

- Elabore uma função chamada **calcularY** para calcular $\boldsymbol{f(x) = x^2 - 3x + 2}$. A seguir, crie uma sequência $\boldsymbol{x}$ de valores entre -30 e 30, com intervalo de 1, e apresente um gráfico de $\boldsymbol{x}$ por $\boldsymbol{f(x)}$.

Exemplo 4 – Resolução

- Elabore uma função chamada **calculaY** para calcular $f(x) = x^2 - 3x + 2$. A seguir, crie uma sequência x de valores entre -30 e 30, com intervalo de 1, e apresente um gráfico de x por $f(x)$.

```
calculaY <- function (x) {
   y <- x^2 - 3*x + 2
   return (y)
}
```

A função interna **seq()** gera um vetor com elementos entre **-30** e **30** com passo **1**.

A função interna **rep()** gera um vetor do tamanho do vetor x preenchido com **0**.

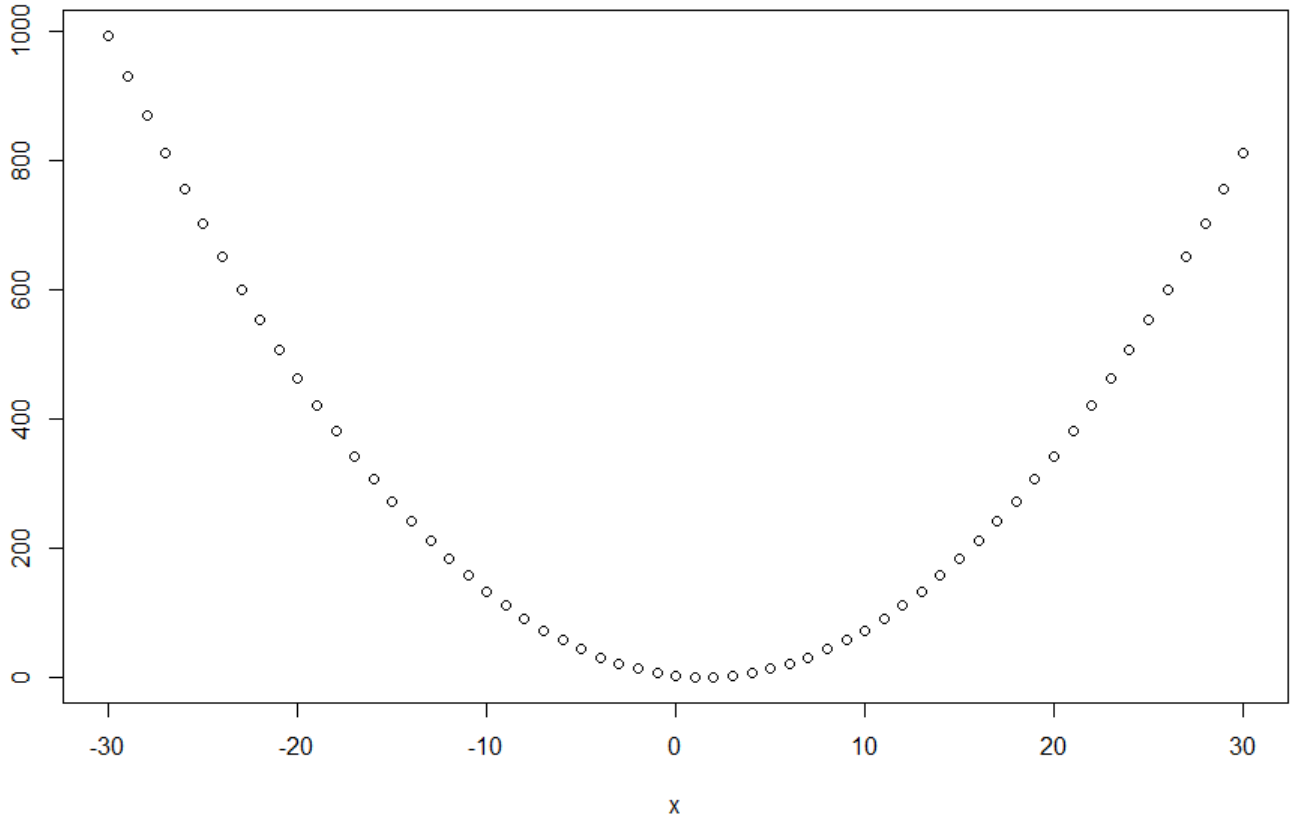
```
x <- seq (-30, 30, 1)
y <- rep (0, length(x))

cont <- 0
for (valorX in x) {
   cont <- cont + 1
   y[cont] <- calculaY(valorX)
}

plot (x, y)
```

A função interna **length()** retorna o número de elementos do vetor x.

cont	x	$y = f(x)$
1	-30	992
2	-29	930
3	-28	870
..	...	...
61	30	812

Exemplo 4 – Resolução

- Elabore uma função chamada **calculaY** para calcular $\boldsymbol{f(x) = x^2 - 3x + 2}$. A seguir, crie uma sequência $\boldsymbol{x}$ de valores entre -30 e 30, com intervalo de 1, e apresente um gráfico de $\boldsymbol{x}$ por $\boldsymbol{f(x)}$.



Exemplo 4 – Resolução (outra forma de resolver)

- Elabore uma função chamada **calculaY** para calcular $f(x) = x^2 - 3x + 2$. A seguir, crie uma sequência x de valores entre -30 e 30, com intervalo de 1, e apresente um gráfico de x por $f(x)$.

```
calculaY <- function (x) {
   y <- x^2 - 3*x + 2
   return (y)
}
```

A função interna **seq()** gera um vetor com elementos entre **-30** e **30** com passo **1**.

A função interna **rep()** gera um vetor do tamanho do vetor x preenchido com **0**.

```
x <- seq (-30, 30, 1)
y <- rep (0, length(x))

cont <- 0
while (cont < length(x)) {
   cont <- cont + 1
   y[cont] <- calculaY(x[cont])
}

plot (x, y)
```

A função interna **length()** retorna o número de elementos do vetor x.

cont	x	$y = f(x)$
1	-30	992
2	-29	930
3	-28	870
..	...	...
61	30	812

Exemplo 5 – Resolução

- Escreva uma função que recebe como parâmetros: um vetor $\boldsymbol{v}$ com $\boldsymbol{n}$ valores numéricos; um vetor **peso** com $\boldsymbol{n}$ valores numéricos; e uma **string** chamada **operacao** que identifica a operação a ser realizada sobre o vetor. Se a operação for **“A”**, a função deve retornar a **média aritmética** dos valores, se for **“P”**, a sua **média ponderada** usando o vetor **peso** para ponderação e se for **“H”**, a sua **média harmônica**.

Exemplo 5 – Resolução

```
realizaOperacao <- function (v, peso, operacao) {
   soma.ponderada <- 0
   soma.inverso <- 0
   resultado <- NULL

   if (toupper(operacao) == "A") {
      resultado <- sum(v) / length(v)
   } else if (toupper(operacao) == "P") {
      for (i in 1:length(v)) {
         soma.ponderada <- soma.ponderada + (v[i] * peso[i])
      }
      resultado <- soma.ponderada / sum(peso)
   } else if (toupper(operacao) == "H") {
      for (i in 1:length(v)) {
         soma.inverso <- soma.inverso + (1 / v[i])
      }
      resultado <- length(v) / soma.inverso
   }

   return(resultado)
}
```

A função interna **toupper()** retorna o valor do parâmetro **operacao** em maiúsculo.

A função interna **sum()** retorna a soma de todos os elementos do vetor v.

A função interna **length()** retorna o número de elementos do vetor v.

Cada elemento do vetor v é multiplicado pelo correspondente **peso** e o resultado é acumulado na variável **soma.ponderada**.

```
vetor <- c (1, 2, 3)
peso <- c (4, 4, 2)

realizaOperacao (vetor, peso, "A")
realizaOperacao (vetor, peso, "P")
realizaOperacao (vetor, peso, "H")
```

Exemplo 5 – Resolução

```
vetor <- c (1, 2, 3)
peso <- c (4, 4, 2)

print(realizaOperacao(vetor, peso, "A"))
print(realizaOperacao(vetor, peso, "P"))
print(realizaOperacao(vetor, peso, "H"))
```

Vetor	Peso	Operação	Resultado
(1, 2, 3)	(4, 4, 2)	A	2
(1, 2, 3)	(4, 4, 2)	P	1.8
(1, 2, 3)	(4, 4, 2)	H	1.636364

Exercício 3

- Usando a função **calculaY** do Exemplo 4, crie uma função chamada **plotaGrafico** que recebe como parâmetros os valores inicial, final e o intervalo de uma sequência $\boldsymbol{x}$ e apresente um gráfico de $\boldsymbol{x}$ por $\boldsymbol{f}(\boldsymbol{x})$.

Exercício 3 – Resolução

- Usando a função **calculaY** do Exemplo 4, crie uma função chamada **plotaGrafico** que recebe como parâmetros os valores inicial, final e o intervalo de uma sequência $\boldsymbol{x}$ e apresente um gráfico de $\boldsymbol{x}$ por $\boldsymbol{f(x)}$.

```
calculaY <- function (x) {
  y <- x^2 - 3*x + 2
  return (y)
}

plotaGrafico <- function (inicial, final, intervalo) {
  x <- seq (inicial, final, intervalo)
  y <- rep (0, length(x))

  cont <- 0
  while (cont < length(x)) {
    cont <- cont + 1
    y[cont] <- calculaY(x[cont])
  }

  plot (x, y)
}
```

```
inicial <- -30
final <- 30
intervalo <- 1
plotaGrafico (inicial, final, intervalo)
```

Exercício 4

- Altere o Exercício 3 de forma a permitir que a função **plotaGrafico** possa também retornar a média dos valores encontrados para $\boldsymbol{f(x)}$.

Exercício 4 – Resolução

- Altere o Exercício 3 de forma a permitir que a função **plotaGrafico** possa também retornar a média dos valores encontrados para $f(x)$.

```
plotaGrafico <- function (inicial, final, intervalo) {
  soma <- 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)
}
```

```
inicial <- -30
final <- 30
intervalo <- 1
media <- plotaGrafico (inicial, final, intervalo)
print(media)
```