

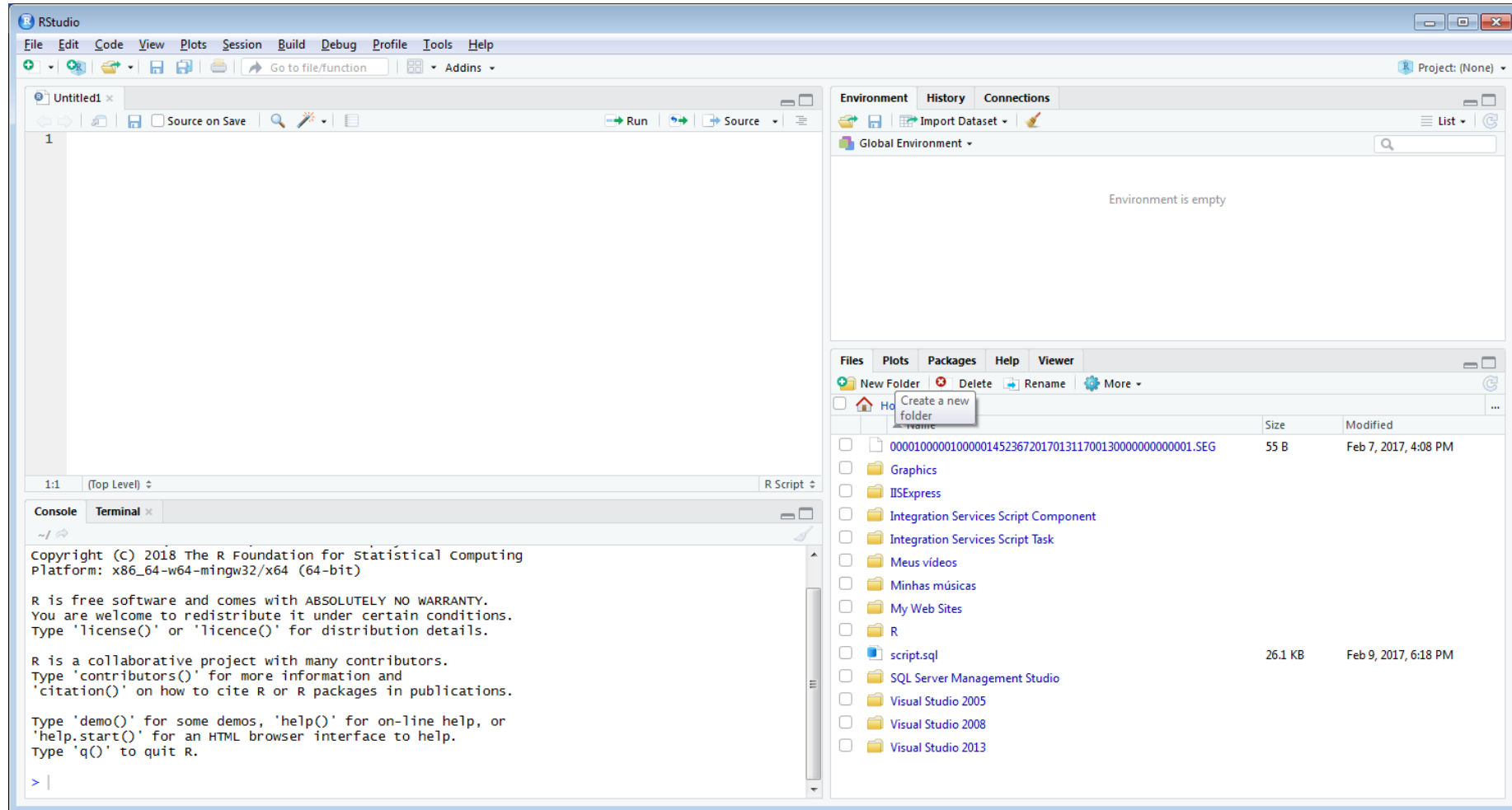
INF 1514

Introdução à Análise de Dados

Material 3



R Studio



Console

- O símbolo “>” no console indica que o R está esperando um comando.
- Funcionamento básico:
 - entre com uma expressão no console e digite <Enter>.
 - a expressão é avaliada e o resultado apresentado na tela
- Note que [1] sempre acompanha qualquer resultado.

```
> 1+2+3  
[1] 6  
> 1+2*3  
[1] 7  
> |
```

- Use os parênteses para calcular expressões, por exemplo:

```
> ((20 + 7)/3)^2  
[1] 81  
>
```

Console

- Para conseguir ajuda sobre um comando pode ser usada a função help:

`help(comando)` ou `?comando`

- Para instalar um pacote no seu ambiente você pode usar o comando `install.packages()` com o nome do pacote entre `""` dentro do parênteses:

`install.packages("nomepacote")`

- Para usar o pacote em seu programa use o comando `library()` com o nome do pacote como parâmetro (sem `""`):

`library(nomepacote)`

Algumas características do R

- R é **case-sensitive**: então “A” e “a” são símbolos diferentes e se referem a diferentes variáveis.
- Comandos diferentes são separados por ponto e vírgula “;”.
- Comentários começam com “#”.
- Como a maioria das linguagens de programação, R permite atribuir valores a variáveis.

```
> # Comentário
> A = 2; a = "uva";
> print(A);
[1] 2
> print(a);
[1] "uva"
> .C=3;
> print(.C+2);
[1] 5
>
```

Constantes armazenadas no R

```
> pi
[1] 3.141593
>
> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
   "r" "s" "t" "u" "v" "w" "x" "y" "z"
>
> LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
   "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
>
> month.abb
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "
Dec"
>
> month.name
[1] "January" "February" "March" "April" "May" "June"
   "July" "August" "September"
[10] "October" "November" "December"
```

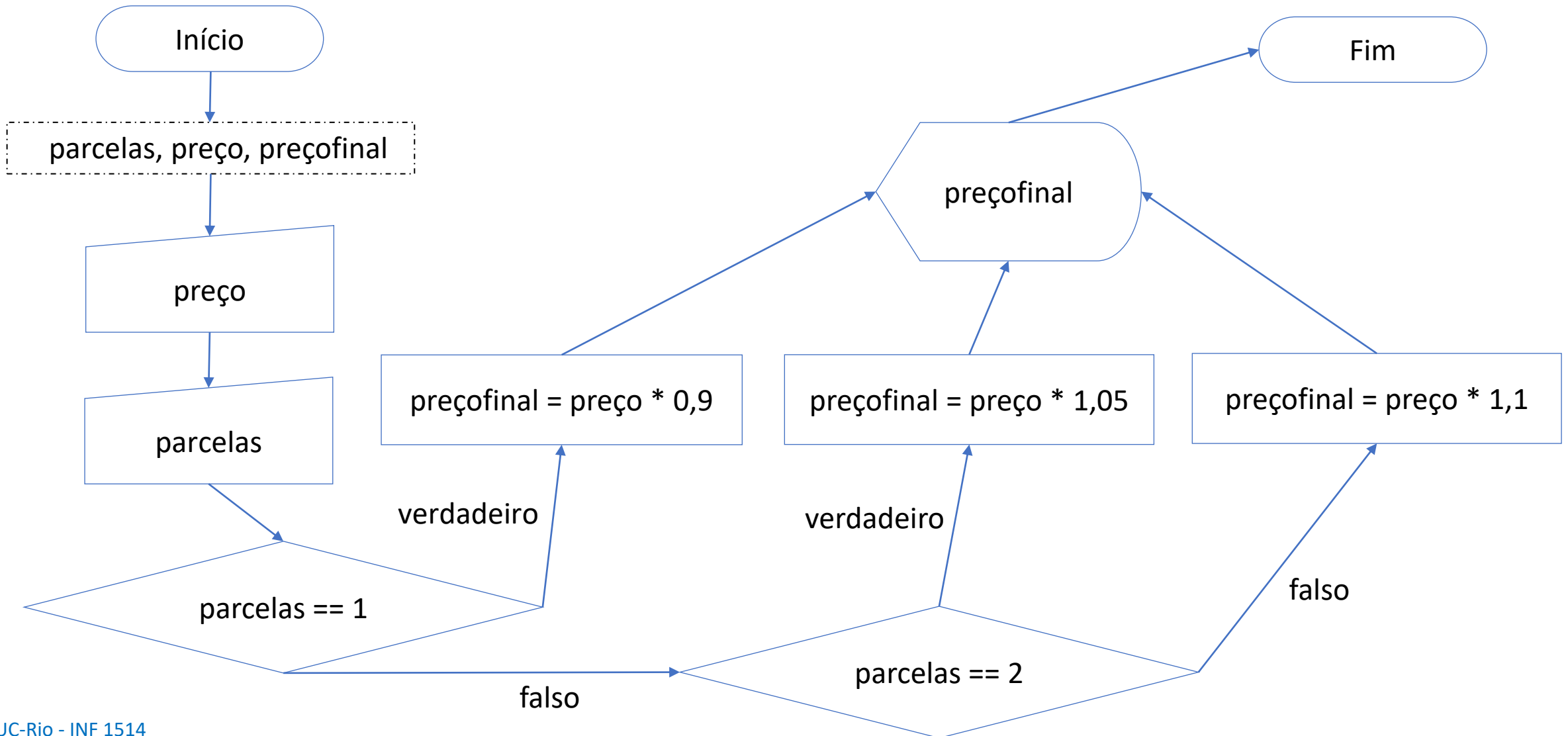
Tipos de dados no R

- character (“aula”)
- numeric (1.2)
- integer (3)
- logical (TRUE or FALSE)
- vector (tipos homogêneos)
- list (parecidos com vectors, mas heterogêneos)
- matrix
- missing values (NA)
- ...

Exemplo 1

- Sabe-se que em uma determinada loja o **preço final** que um cliente irá pagar 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%**.
- Defina um algoritmo para calcular o preço final pago pelo cliente dado o número de parcelas e o preço do produto.

Usando fluxograma



Pseudocódigo

```
variáveis
    parcelas, preço, preçofinal
início
    leia preço;
    leia parcelas;
    se (parcelas == 1) então
        preçofinal = preço * 0,9;
    senão
        se (parcelas == 2) então
            preçofinal = preço * 1,05;
        senão
            preçofinal = preço * 1,1;
        fim-se;
    fim-se;
    escreva preçofinal;
fim.
```

Pseudocódigo

```
variáveis
    parcelas, preço, preçofinal
início
    leia preço;
    leia parcelas;
    se (parcelas == 1) então
        preçofinal = preço * 0,9;
    senão
        se (parcelas == 2) então
            preçofinal = preço * 1,05;
        senão
            preçofinal = preço * 1,1;
        fim-se;
    fim-se;
    escreva preçofinal;
fim.
```

Programa em R

```
parcelas <- 0
preço <- 0.0
precofinal <- 0.0

preço <- scan()
parcelas <- scan()
if (parcelas == 1) {
    precofinal <- preço * 0.9
} else {
    if (parcelas == 2) {
        precofinal <- preço * 1.05
    } else {
        precofinal <- preço * 1.1
    }
}

print(precofinal)
```

Programa em R

```
# variáveis
parcelas <- 0
preco <- 0.0
precofinal <- 0.0
# inicio
preco <- scan()      # lendo o preço
parcelas <- scan()   # lendo as parcelas
if (parcelas == 1) {
  precofinal <- preco * 0.9
} else {
  if (parcelas == 2) {
    precofinal <- preco * 1.05
  } else {
    precofinal <- preco * 1.1
  }
}
print(precofinal)
# fim
```

Programa em R

```
# variáveis
parcelas <- 0
preco <- 0.0
precofinal <- 0.0
# inicio
preco <- scan()      # lendo o preço
parcelas <- scan()   # lendo as parcelas
if (parcelas == 1) {
  precofinal <- preco * 0.9
} else {
  if (parcelas == 2) {
    precofinal <- preco * 1.05
  } else {
    precofinal <- preco * 1.1
  }
}
print(precofinal)
# fim
```

Comentários são marcados por # .

Programa em R

```
# variáveis
parcelas <- 0
preco <- 0.0
precofinal <- 0.0
# inicio
preco <- scan()      # lendo o preço
parcelas <- scan()   # lendo as parcelas
if (parcelas == 1) {
  precofinal <- preco * 0.9
} else {
  if (parcelas == 2) {
    precofinal <- preco * 1.05
  } else {
    precofinal <- preco * 1.1
  }
}
print(precofinal)
# fim
```

Variáveis são declaradas por seu valor.

Para atribuir valores a variáveis basta usar o operador <-. O operador = também pode ser utilizado no lugar do operador <-.

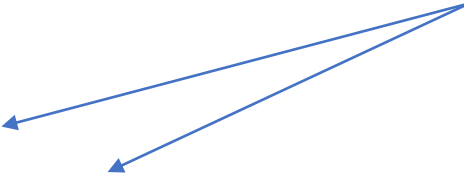
Variável é o nome que se dá a um espaço reservado na memória, onde será possível armazenar um valor de um determinado tipo.



Programa em R

```
# variáveis
parcelas <- 0
preco <- 0.0
precofinal <- 0.0
# inicio
preco <- scan()      # lendo o preço
parcelas <- scan()  # lendo as parcelas
if (parcelas == 1) {
  precofinal <- preco * 0.9
} else {
  if (parcelas == 2) {
    precofinal <- preco * 1.05
  } else {
    precofinal <- preco * 1.1
  }
}
print(precofinal)
# fim
```

scan() é uma função interna do R que lê valores do teclado e os atribui a uma variável.



Programa em R

```
# variáveis
parcelas <- 0
preco <- 0.0
precofinal <- 0.0
# inicio
preco <- scan()      # lendo o preço
parcelas <- scan()   # lendo as parcelas
if (parcelas == 1) {
  precofinal <- preco * 0.9
} else {
  if (parcelas == 2) {
    precofinal <- preco * 1.05
  } else {
    precofinal <- preco * 1.1
  }
}
print
# fim
```

print() é uma função interna do R que imprime o que lhe for passado como parâmetro.

Programa em R

```
# variáveis
parcelas <- 0
preco <- 0.0
precofinal <- 0.0
# inicio
preco <- scan() # lendo o preço
parcelas <- scan() # lendo as parcelas
if (parcelas == 1) {
  precofinal <- preco * 0.9
} else {
  if (parcelas == 2) {
    precofinal <- preco * 1.05
  } else {
    precofinal <- preco * 1.1
  }
}
print(precofinal)
# fim
```

Comandos de atribuição <- .

The diagram consists of three blue arrows originating from a blue rectangular box on the right. The first arrow points to the assignment operator '<-' in the line 'parcelas <- 0'. The second arrow points to the assignment operator '<-' in the line 'preco <- scan()'. The third arrow points to the assignment operator '<-' in the line 'precofinal <- preco * 1.05'.

Programa em R

```
# variáveis
parcelas <- 0
preco <- 0.0
precofinal <- 0.0
# inicio
preco <- scan()      # lendo o preço
parcelas <- scan()   # lendo as parcelas
if (parcelas == 1) {
  precofinal <- preco * 0.9
} else {
  if (parcelas == 2) {
    precofinal <- preco * 1.05
  } else {
    precofinal <- preco * 1.1
  }
}
print(precofinal)
# fim
```

Blocos de instrução são delimitados por { e }.

Operadores matemáticos

- A lista abaixo apresenta alguns dos principais operadores matemáticos do R.

Operador	Descrição
+	Operador de adição
-	Operador de subtração
*	Operador de multiplicação
/	Operador de divisão
:	Operador de sequência
^	Operador exponencial
%%	Operador de módulo

```
x <- (2 + 3) * 2      # x assume o valor 10.
y <- 2 + 3 * 2        # y assume o valor 8.
z = x - y             # z assume o valor 2.
x <- (4 / 5) * (-2)   # x assume o valor -1.6.
a <- 1:10             # a é um vetor de 1 a 10.
y <- z^3              # y assume o valor 8.
z <- 15 %% 4          # z assume o valor 3 (resto da
                     # divisão inteira de 15 por 4.
```

Se `(b %% 2)` retorna 0, então b é um número par.

Se `(b %% 2)` retorna 1, então b é um número ímpar.

Operadores relacionais e lógicos

- São operados binários para realização de testes entre duas variáveis (objetos). Estas operações retornam o valor TRUE (1) ou FALSE (0).

Operador	Descrição
<code>==</code>	Operador de igualdade
<code>></code>	Operador “maior que”
<code><</code>	Operador “menor que”
<code><=</code>	Operador “menor ou igual”
<code>!</code>	Operador “Não”
<code>&</code>	Operador lógico “E”
<code> </code>	Operador lógico “OU”

```
x <- 10
```

```
y <- 8
```

```
(x == y)           # retorna FALSE.
```

```
(x == (y + 2))     # retorna TRUE.
```

```
(x < y)            # retorna FALSE.
```

```
(x > y)            # retorna TRUE.
```

```
(x <= y)           # retorna FALSE.
```

```
(x != y)           # retorna TRUE.
```

Operadores relacionais e lógicos

- São operados binários para realização de testes entre duas variáveis (objetos). Estas operações retornam o valor TRUE (1) ou FALSE (0).

Operador	Descrição
<code>==</code>	Operador de igualdade
<code>></code>	Operador “maior que”
<code><</code>	Operador “menor que”
<code><=</code>	Operador “menor ou igual”
<code>!</code>	Operador “Não”
<code>&</code>	Operador lógico “E”
<code> </code>	Operador lógico “OU”

```
x <- 2
```

```
y <- 3
```

```
z <- 4
```

```
!(x == y)
```

```
# retorna TRUE.
```

```
!(x < y)
```

```
# retorna FALSE.
```

```
!TRUE
```

```
# retorna FALSE.
```

```
!FALSE
```

```
# retorna TRUE.
```

Operadores relacionais e lógicos

- São operados binários para realização de testes entre duas variáveis (objetos). Estas operações retornam o valor TRUE (1) ou FALSE (0).

Operador	Descrição
<code>==</code>	Operador de igualdade
<code>></code>	Operador “maior que”
<code><</code>	Operador “menor que”
<code><=</code>	Operador “menor ou igual”
<code>!</code>	Operador “Não”
<code>&</code>	Operador lógico “E”
<code> </code>	Operador lógico “OU”

```
x <- 2
```

```
y <- 3
```

```
z <- 4
```

```
(x < y) & (y < z) # retorna TRUE.
```

```
(x < y) & (x > 5) # retorna FALSE.
```

```
TRUE & TRUE      # retorna TRUE.
```

```
TRUE & FALSE     # retorna FALSE.
```

```
FALSE & TRUE     # retorna FALSE.
```

```
FALSE & FALSE    # retorna FALSE.
```

Operadores relacionais e lógicos

- São operados binários para realização de testes entre duas variáveis (objetos). Estas operações retornam o valor TRUE (1) ou FALSE (0).

Operador	Descrição
<code>==</code>	Operador de igualdade
<code>></code>	Operador “maior que”
<code><</code>	Operador “menor que”
<code><=</code>	Operador “menor ou igual”
<code>!</code>	Operador “Não”
<code>&</code>	Operador lógico “E”
<code> </code>	Operador lógico “OU”

```
x <- 2
```

```
y <- 3
```

```
z <- 4
```

```
(x < y) | (y < z) # retorna TRUE.
```

```
(x > y) | (x > 1) # retorna TRUE.
```

```
TRUE | TRUE      # retorna TRUE.
```

```
TRUE | FALSE     # retorna TRUE.
```

```
FALSE | TRUE     # retorna TRUE.
```

```
FALSE | FALSE    # retorna FALSE.
```

Exercício 1

- Tomando como base o script R a seguir, preencha a tabela abaixo.

```
x <- (2 - 3) * 2
```

```
y <- 1 + 3 ^ 2
```

```
z <- y - x
```

Expressão	Valor
x	
y	
z	
abs(x)	
sqrt(y)	
(x <= y)	
(x > y) & (x < z)	
!(x >= z)	
(x > y) (x < z)	
(x < y) & (x < z)	
((x > y) & (y > z)) ((x + y) > z)	

Exercício 1 – Resolução

- Tomando como base o script R a seguir, preencha a tabela abaixo.

```
x <- (2 - 3) * 2
```

```
y <- 1 + 3 ^ 2
```

```
z <- y - x
```

Expressão	Valor
x	-2
y	10
z	12
abs(x)	2
sqrt(y)	3.16
(x <= y)	TRUE
(x > y) & (x < z)	FALSE
!(x >= z)	TRUE
(x > y) (x < z)	TRUE
(x < y) & (x < z)	TRUE
((x > y) & (y > z)) ((x + y) > z)	FALSE

Exercício 2

- Construa um programa (script) em R para obter o resultado da divisão de dois números quaisquer fornecidos pelo usuário.

Exercício 2 – Resolução

- Construa um programa (script) em R para obter o resultado da divisão de dois números quaisquer fornecidos pelo usuário.

```
dividendo <- 0.0
divisor <- 0.0
resultado <- 0.0
dividendo <- scan()
divisor <- scan()
if (divisor != 0) {
  resultado <- dividendo / divisor
  print(resultado)
} else {
  print('Divisão não efetuada.')
}
```