



UNIVERSIDADE ESTADUAL DA PARAÍBA  
CENTRO DE CIÊNCIAS E TECNOLOGIA  
CIÊNCIA DA COMPUTAÇÃO

LUCAS DE LUCENA SIQUEIRA  
DANIEL XAVIER BRITO DE ARAUJO

Linguagem de Programação R

CAMPINA GRANDE  
2022

<b>1. Introdução</b>	<b>3</b>
<b>2. Paradigmas da Linguagem R</b>	<b>3</b>
<b>2.1. Paradigma Orientado a Objetos</b>	<b>3</b>
<b>2.2. Paradigma Funcional</b>	<b>3</b>
<b>3. Estruturas Lógicas</b>	<b>4</b>
<b>3.1. Entrada de Dados</b>	<b>4</b>
<b>3.1.1. Entrada Via Teclado</b>	<b>4</b>
<b>3.1.2. Entrada Via Arquivos CSV</b>	<b>6</b>
<b>3.1.3. Entrada Via Arquivos XLS</b>	<b>6</b>
<b>3.2. Saída de Dados</b>	<b>7</b>
<b>3.3. Estrutura de Condição</b>	<b>8</b>
<b>3.4. Estrutura de Repetição</b>	<b>9</b>
<b>3.4.1. For</b>	<b>9</b>
<b>3.4.2. While</b>	<b>10</b>
<b>3.5. Definição de Funções</b>	<b>11</b>
<b>3.6. Funções Nativas</b>	<b>12</b>
<b>3.6.1 Operações Aritméticas</b>	<b>12</b>
<b>3.6.2 Estatística Descritiva</b>	<b>12</b>
<b>3.6.3 Geração de números, sequências e dados aleatórios</b>	<b>13</b>
<b>3.6.4 Tratamento de vetores ou tabela de dados</b>	<b>13</b>
<b>Referências Bibliográficas</b>	<b>14</b>

## 1. Introdução

De acordo com o blog “Didática Tech”, a linguagem R foi desenvolvida por 2 estatísticos da Universidade Auckland, localizada na Nova Zelândia, tendo sido construída a partir de uma real necessidade presente nos laboratórios do departamento de estatística, referente à necessidade de um melhor ambiente para desenvolvimento computacional. A partir disso então, dois estatísticos “Ross” e “Robert” desenvolveram a linguagem “R”, com seu nome fazendo referência aos seus criadores.

Um dos desenvolvedores conta que a construção da linguagem não foi uma tarefa muito complicada, já que o acesso a diversos livros era facilitado, a exemplo do livro “The Structure and Interpretation of Computer Programs” de Hal Abelson e Gerald Sussman. Além de consultas bibliográficas, houve muita interação com códigos fontes de interpretadores da linguagem *Scheme*.

O conhecimento da linguagem S, sendo ela uma linguagem voltada para a área da estatística também, levou os criadores a tomar por referência algumas interfaces e características dela ao desenvolver a linguagem R, tornando ambas bem semelhantes, apesar de suas diferenças características.

Por fim, a linguagem teve como objetivo inicial auxiliar na manipulação, análise e visualização de dados, tendo por característica ser multi-paradigma, abrangendo paradigmas como o funcional e orientada a objetos, sendo ela fracamente tipada e possuindo código aberto, dando assim, a disponibilidade de colaboração de outros desenvolvedores voluntários.

## 2. Paradigmas da Linguagem R

Por ser constatado que a linguagem abrange os paradigmas de orientação a objetos e funcional, diferentes metodologias de desenvolvimento podem ser adotadas.

### 2.1. Paradigma Orientado a Objetos

O paradigma orientado a objetos é um modelo de programação que se concentra em objetos e suas interações em vez de em outras estruturas de programação, como funções ou sequências de instruções. Ele possui algumas características como métodos, atributos, classes, herança, encapsulamento e polimorfismo.

### 2.2. Paradigma Funcional

O paradigma de programação funcional está presente em linguagens que adotem o desenvolvimento de programas escritos com uma sequência de funções que tem por responsabilidade fazer todas as transformações necessárias nos dados de entrada, gerando por sua vez, uma saída respectiva esperada, sendo por sua vez, ideal para

ambientes que necessitem de soluções na área da matemática, inteligência artificial e de aplicação lógicas, como afirma o professor Gustavo, da PUCRS.

### 3. Estruturas Lógicas

Assim como em qualquer linguagem de programação, *R* também tem suas peculiaridades e particularidades quando se trata da sintaxe e de alguns mecanismos lógicos. Portanto, serão abordados alguns conceitos básicos da linguagem que são cruciais nas soluções de problemas, como a lógica de entrada e saída de dados, as estruturas de condição, estruturas de repetição e por fim, a utilização de funções nativas e definição de funções não nativas.

#### 3.1. Entrada de Dados

Como apresentado pelo professor Fernando Mayer, da UFPR, a entrada de dados em *R* pode ser estabelecida de diversas formas e em diversos formatos, dependendo do tamanho do conjunto de dados ou se há a preferência da importação de dados já existentes.

##### 3.1.1. Entrada Via Teclado

A utilização da função `scan()` irá ler os dados inseridos diretamente no console via teclado, podendo utilizar a tecla `Enter` para terminar a inserção de um dado e prosseguir para a inserção de um outro dado e por fim, podendo pressionar a tecla `Enter` duas vezes seguidas para encerrar a entrada de novos dados. A função `scan()` aceita por padrão a entrada de dados de valores numéricos, mas pode interpretar a entrada como outros valores a partir do preenchimento do parâmetro opcional `what`, que pode receber os seguintes valores: *logical*, *integer*, *numeric*, *complex*, *character*, *raw* e *list*.



```
1 data <- scan()
2 print(data)
3
```

**Imagem 1 - Script para entrada de dados**



```
project x +
1: 11
2: 22
3: 33
4: 44
5: 55
6: 66
7: 77
8:
Read 7 items
[1] 11 22 33 44 55 66 77

> |
```

**Imagem 2 - Console do script presente na Imagem 1**



```
+ > Parent environments
↑ > Functions
↓ > data = 11 22 33 44 55 66 77
```

**Imagem 3 - Console do script presente na Imagem 1**

Os dados também podem ser inseridos sequencialmente, porém separados por um espaço, como mostra o exemplo a seguir:



```
project x +
1: 11 22 33 44 55
6: 66 77
8:
Read 7 items
[1] 11 22 33 44 55 66 77

> |
```

**Imagem 4 - Console do script presente na Imagem 1**

Por fim, como já foi introduzido antes, é possível receber valores do tipo *character* por exemplo, passando o tipo de variável desejável como parâmetro na função `scan()` da seguinte forma:

```
project x +
1: 1 2 3 a b c
7:
Read 6 items
[1] "1" "2" "3" "a" "b" "c"
>
```

Imagem 5 - Parâmetro *what* da função *scan()*

### 3.1.2. Entrada Via Arquivos CSV

Há também a possibilidade de importar arquivos do tipo CSV (*comma separated values*) a partir do uso da função `read.csv2()` que recebe alguns parâmetros, dentre eles, o diretório do arquivo a ser lido, como mostra o exemplo abaixo:

```
1 data <- read.csv2("data/montgomery_6-26.csv")
2 head(data)
3
```

Imagem 6 - Utilização da função *read.csv2()*

```
project x +
Temperatura.Concentracao.Tempo.Pressao.Peso_Molecular.Viscosidade
1 100,4,20,60,2400,1400
2 120,4,20,60,2410,1500
3 100,8,20,60,2315,1520
4 120,8,20,60,2510,1630
5 100,4,30,60,2615,1380
6 120,4,30,60,2625,1525
>
```

Imagem 7 - Console do script presente na Imagem 6

**Nota:** Link de acesso ao arquivo csv utilizado: [montgomery\\_6-26.csv](#)

### 3.1.3. Entrada Via Arquivos XLS

A forma em que a entrada de arquivos de extensão xls é bem semelhante à de arquivos csv, já apresentada anteriormente, a diferença está na necessidade da instalação do pacote “*readxl*” a partir da função `library()`. Por fim, basta utilizar a função `read_excel()`, passando como parâmetros a localização do arquivo e a página do arquivo que deve ser lida, como mostra o exemplo abaixo:

```
1 # Carrega o pacote
2 library(readxl)
3
4 dados ← read_excel("data/crabs.xls", sheet = "Plan1")
5
6 print(dados)
7
```

Imagem 8 - Utilização da função `read_excel()`

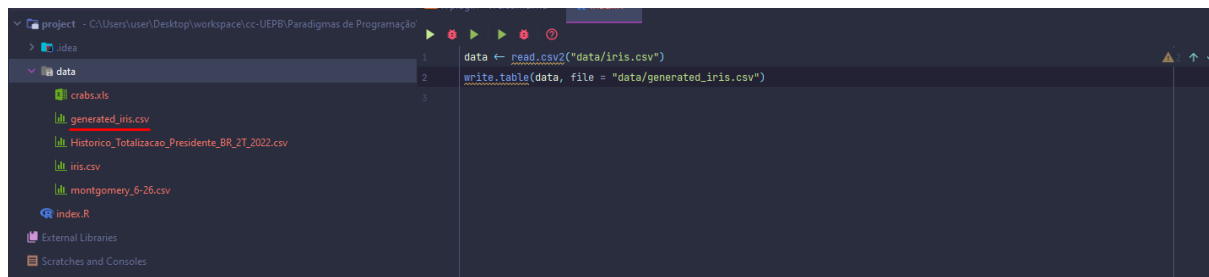
```
project x +
Warning: package 'readxl' was built under R version 4.1.3
# A tibble: 156 x 7
  especie sexo FL RW CL CW BD
  <chr> <chr> <chr> <chr> <chr> <chr> <chr>
1 azul M 8,1 6,7 16,1 19 7
2 azul M 8,8 7,7 18,1 20,8 7,4
3 azul M 9,2 7,8 19 22,4 7,7
4 azul M 9,6 7,9 20,1 23,1 8,2
5 azul M 10,8 9 23 26,5 9,8
6 azul M 11,6 9,1 24,5 28,4 10,4
7 azul M 11,8 10,5 25,2 29,3 10,3
8 azul M 12,3 11 26,8 31,5 11,4
9 azul M 12,6 10 27,7 31,7 11,4
10 azul M 12,8 10,9 27,4 31,5 11
# ... with 146 more rows
# i Use `print(n = ...)` to see more rows
>
```

Imagem 9 - Console do script presente na Imagem 8

Nota: Link de acesso ao arquivo csv utilizado: [crabs.xls](#)

### 3.2. Saída de Dados

Uma função que pode ser utilizada na saída de dados em *R* é bem semelhante à função `read.table()`, já apresentada anteriormente. Portanto, a função que pode ser utilizada é a `write.table()`, que pode gerar um arquivo em formato csv, dependendo assim, dos argumentos passados pelos parâmetros da função, como mostra o exemplo a seguir:



**Imagem 10 - Utilização da função *write.table()***

No exemplo acima, o primeiro parâmetro que foi passado faz referência ao conjunto de dados que o usuário deseja exportar, o segundo, se refere ao diretório de exportação desejado.

**Nota:** Link de acesso ao arquivo csv utilizado: [iris.csv](#)

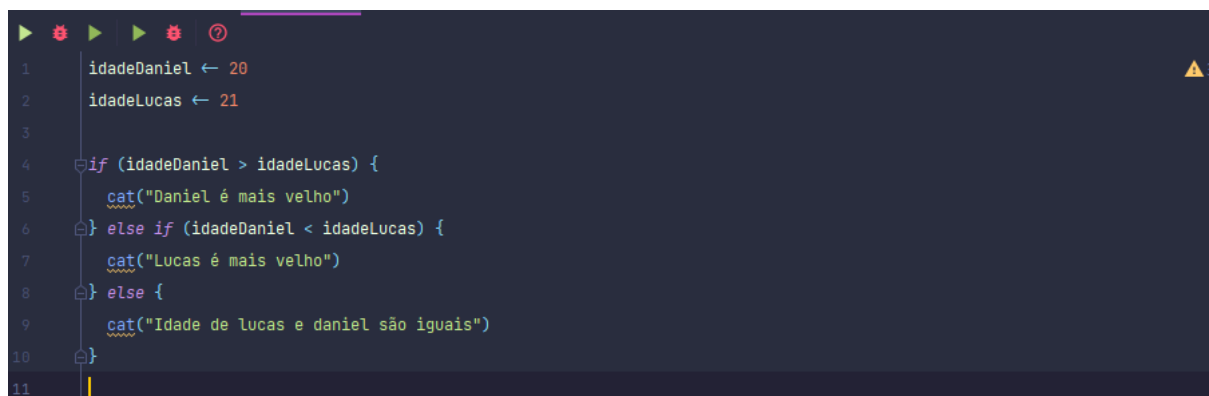
### 3.3. Estrutura de Condição

As estruturas de condição do *R* são similares às de outras linguagens como por exemplo o Java, com blocos de código identificados com chaves, como mostra o exemplo abaixo:



**Imagem 11 - Encadeamento condicional de *if* e *else***

Sendo possível também, realizar o encadeamento de vários blocos:



**Imagem 12 - Encadeamento condicional de *if*, *else if* e *else***



Há também uma estrutura condicional característica da linguagem R, sendo o `ifelse`, que se aplica a vetores de dados, atribuindo um valor para cada elemento do vetor, caso ele seja verdadeiro ou falso (`ifelse(vetor, valor_se_true, valor_se_false)`):



```
1 vetor <- c(TRUE, TRUE, FALSE)
2 ifelse(vetor, 1, -1)
```

Imagem 13 - Uso do `ifelse`



```
project x +
[1] 1 1 -1 NA
```

Imagem 14 - Console do script presente na Imagem 13

Nesse exemplo, é passado um vetor com os valores simples e a função `ifelse` verifica cada valor do vetor, atribuindo o valor 1 para caso o resultado da análise seja TRUE e -1 para caso seja FALSE.

### 3.4. Estrutura de Repetição

As estruturas de repetição são mecanismos lógicos que permitem com que o desenvolvedor escreva soluções menos repetitivas, deixando assim, o código mais legível, eficiente e enxuto. Em R, as estruturas de repetição são bem semelhantes a outras linguagens, como o Python e o Java.

#### 3.4.1. For

A estrutura `for` tem sua sintaxe similar ao `for` nominal do python ou a um `for each` do Java, recebendo uma definição de quantidade de repetições e um trecho de código que será executado em cada repetição, como mostra o exemplo abaixo:



```
1 for (i in 1:10){
2   cat(i, "\n")
3 }
4
```

Imagem 15 - Uso do `for`

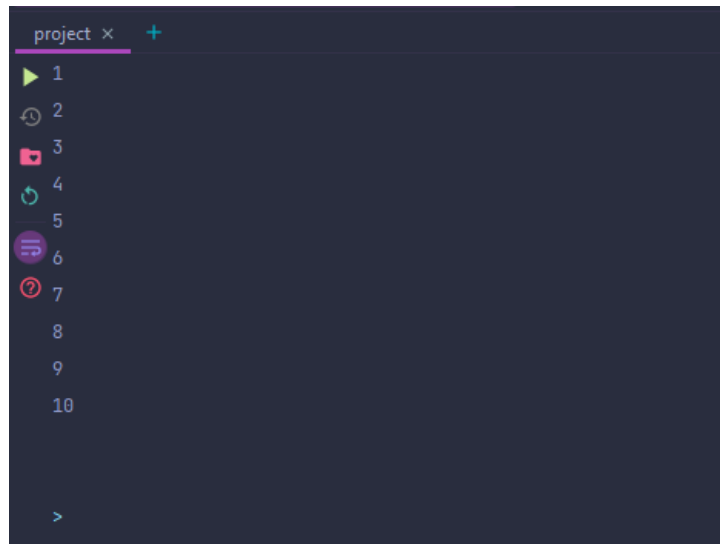


Imagem 16 - Console do script presente na Imagem 15

Segue um exemplo do aninhamento de um `if` dentro de um `for`, que irá exibir no terminar apenas os números pares de 1 a 10:

```
1  for (i in 1:10){
2    if (i % 2 == 0){
3      cat(i, "\n")
4    }
5  }
```

Imagem 17 - Uso do `for` aninhado com um `if`

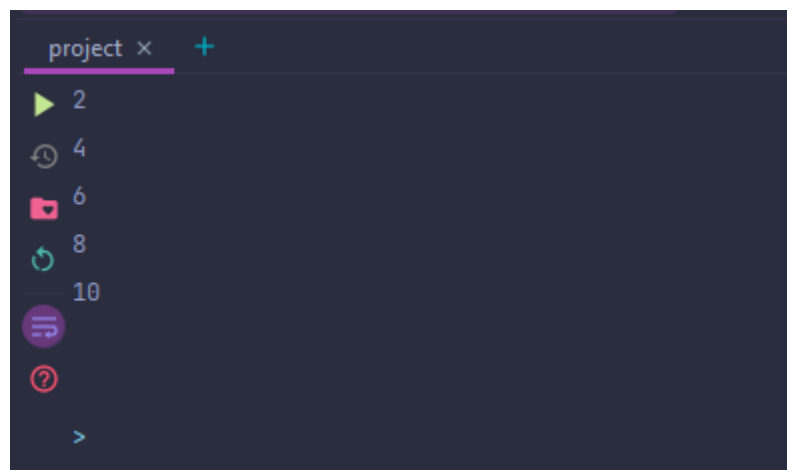


Imagem 18 - Console do script presente na Imagem 15

### 3.4.2. While

A sintaxe e o funcionamento do `while` é muito semelhante à de outras linguagens também, como seria em Java ou C por exemplo. Nela você também define uma trecho de código que será repetido até que a condição de parada seja satisfeita.

A screenshot of an R script editor showing a `while` loop. The code is as follows:

```
1 x <- 10
2
3 while(x > 0) {
4   cat("Decrementando \n")
5   x <- x - 1;
6 }
7
```

The editor has a dark theme with syntax highlighting. Line numbers 1 through 7 are visible on the left. There are icons at the top for running, saving, and other actions.

Imagem 19 - Uso do *while*

### 3.5. Definição de Funções

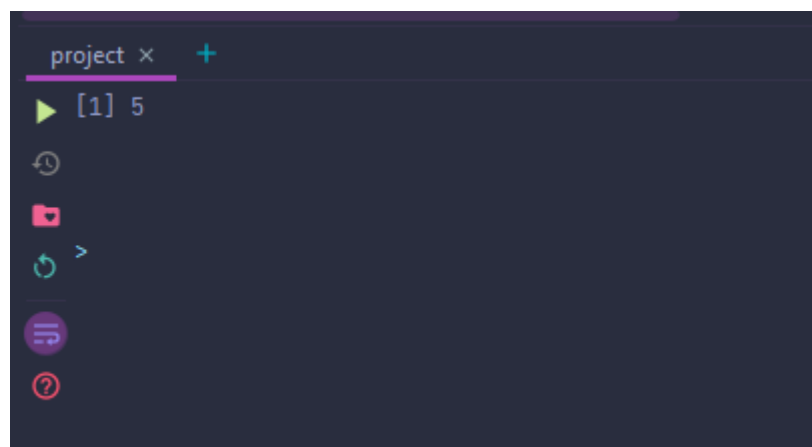
Há também a possibilidade de definição de funções com estrutura lógica definida pelo usuário, tornando o uso da linguagem versátil, assim como ocorre em outras linguagens de programação, como Javascript. Porém a sintaxe da definição de uma função em R é um pouco diferente, atribuindo o nome da função da mesma forma em que é atribuído um valor a uma variável, como mostra o exemplo abaixo, que está definindo uma função que retorna o menor número entre dois que são passados como parâmetro:

A screenshot of an R script editor showing a function definition. The code is as follows:

```
1 lower_number <- function(a, b) {
2   if (a < b) {
3     return (a) ^lower_number
4   } else {
5     return (b) ^lower_number
6   }
7 }
8 lower_number( a: 5, b: 22)
9
```

The editor has a dark theme with syntax highlighting. Line numbers 1 through 9 are visible on the left. There are icons at the top for running, saving, and other actions.

Imagem 20 - Definição de função

A screenshot of the R console window. The title bar says "project x" with a close button and a plus sign. The console shows the output of the function call from the previous image:

```
[1] 5
```

Below the output, there are several icons: a clock, a folder, a right arrow, a list icon, and a question mark icon.

Imagem 21 - Console do script presente na Imagem 20

### 3.6. Funções Nativas

Assim como em toda linguagem de programação, em R também existem diversas funções nativas, algumas irão coincidir com a de outras linguagens, por ter um funcionamento similar ou igual, porém, existem funções mais voltadas para a área de estética, que podem gerar numeros, fazer alguns cálculos mais específicos ou analisar alguns dados, tudo isso de forma nativa.

#### 3.6.1 Operações Aritméticas

Função	Descrição
max(vetor)	Obter maior valor
min(vetor)	Obter menor valor
sum(vetor)	Obter a soma
factorial(6)	Obter fatorial
log(vetor)	Obter logaritmo
sqrt(vetor)	Obter raiz quadrada
abs(vetor)	Obter valor absoluto
rowSums(A)	Soma das linhas
colSums(A)	Soma das colunas
round(vetor, digits = 1)	Arredondar número para 1
sort(vetor)	Ordenar o vetor
order(vetor)	Obter posição

#### 3.6.2 Estatística Descritiva

Função	Descrição
mean(vetor)	Calcular a média
median(vetor)	Calcular a mediana
var(vetor)	Calcular a variância
sd(vetor)	Calcular o desvio padrão
cor(vetor1, vetor2)	Correlação entre 2 vetores
cor(vetor1, vetor2, method = "spearman")	Correlação utilizando o método de Spearman
cov(ex1, ex3)	Covariação entre 2 vetores

### 3.6.3 Geração de números, sequências e dados aleatórios

Função	Descrição
seq()	Obter sequência regular
paste()	Obter sequência regular de caracteres
rep()	Função de repetir valores ou sequências
runif()	Gerar distribuição uniforme
rnorm()	Gerar distribuição normal
rpois()	Gerar distribuição de Poisson
sample()	Amostrar valores

### 3.6.4 Tratamento de vetores ou tabela de dados

Função	Descrição
sapply()	Aplica funções em data.frames/listas
apply()	Aplica funções em matrizes
tapply()	Aplica funções em vetores
table()	Produz tabelas de contingência

## Referências Bibliográficas

- [01] LINGUAGEM R: o que é, para que usar e por que aprender?. [S. l.], 2 set. 2022. Disponível em: <https://blog.betrybe.com/linguagem-de-programacao/linguagem-r-tudo-sobre/>. Acesso em: 30 out. 2022.
- [02] A LINGUAGEM R. [S. l.], 2022. Disponível em: <https://didatica.tech/a-linguagem-r/>. Acesso em: 30 out. 2022.
- [03] R (linguagem de programação). [S. l.], 30 out. 2022. Disponível em: [https://pt.wikipedia.org/wiki/R\\_\(linguagem\\_de\\_programação\)](https://pt.wikipedia.org/wiki/R_(linguagem_de_programação)). Acesso em: 30 out. 2022.
- [04] PARADIGMA Funcional. [S. l.], 23 ago. 2018. Disponível em: <https://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula15.pdf>. Acesso em: 8 nov. 2022.
- [05] ENTRADA e saída de dados no R Introdução. [S. l.], ca. 2022. Disponível em: <http://leg.ufpr.br/~fernandomayer/aulas/ce083/entrada-dados.html#>. Acesso em: 9 nov. 2022.
- [06] INDEXAÇÃO e seleção condicional. [S. l.], ca. 2022. Disponível em: [http://leg.ufpr.br/~fernandomayer/aulas/ce083/indexacao\\_e\\_selecao.html#indexação](http://leg.ufpr.br/~fernandomayer/aulas/ce083/indexacao_e_selecao.html#indexação). Acesso em: 9 nov. 2022.
- [07] PROGRAMAÇÃO no R: if(), if() else e ifelse(). [S. l.], ca. 2016. Disponível em: <https://analysereal.com/2016/03/02/programacao-no-r-if-if-else-e-ifelse-2/>. Acesso em: 9 nov. 2022.
- [08] INTRODUÇÃO ao R. [S. l.], 27 maio 2020. Disponível em: [https://vanderleidebastiani.github.io/tutoriais/Introducao\\_ao\\_R.html#aplicar\\_funções\\_em\\_vetores\\_ou\\_tabela\\_de\\_dados](https://vanderleidebastiani.github.io/tutoriais/Introducao_ao_R.html#aplicar_funções_em_vetores_ou_tabela_de_dados). Acesso em: 9 nov. 2022.