

# Introdução ao R

*Iara Passos*

*2019-11-04*



# Contents

<b>1</b>	<b>Introdução</b>	<b>5</b>
1.1	O que é a linguagem R? . . . . .	5
1.2	Instalando o R . . . . .	6
1.3	Instalando o RStudio . . . . .	6
<b>2</b>	<b>Aula 1</b>	<b>7</b>
2.1	Janela de comando do R . . . . .	7
2.2	O R como uma calculadora . . . . .	8
2.3	Objetos e classes de objetos . . . . .	9
2.4	Ambiente do RStudio . . . . .	14
2.5	Pacotes . . . . .	15
2.6	Como conseguir ajuda . . . . .	15
2.7	Vetores . . . . .	15
2.8	Listas de exercícios . . . . .	23
<b>3</b>	<b>Aula 2</b>	<b>25</b>
<b>4</b>	<b>Aula 3</b>	<b>27</b>
<b>5</b>	<b>Aula 4</b>	<b>29</b>
<b>6</b>	<b>Aula 5</b>	<b>31</b>



# Chapter 1

## Introdução

A ideia dessa apostila é que seja utilizada como complemento das aulas, junto com as listas de exercícios disponibilizadas.

### 1.1 O que é a linguagem R?

R é uma linguagem de programação e um ambiente para implementação de análises estatísticas e gráficos. Um Ambiente de Desenvolvimento Integrado<sup>1</sup> (do inglês, *Integrated Development Environment*, IDE) é um *software* que contém ferramentas para auxiliar o desenvolvimento de *softwares*, de modo a otimizar e facilitar esse processo. As principais funções de uma IDE são: edição, compilação, depuração e modelagem.

Foi criado em 1993, por Ross Ihaka e por Robert Gentleman do Departamento de Estatística da Universidade de Auckland na Nova Zelândia, baseado na linguagem de programação e ambiente S, desenvolvido por John Chambers na Bell Laboratories (hoje Lucent Technologies). A partir de 1997 passou a ser desenvolvido por um grupo de colaboradores do mundo todo<sup>2</sup>.

#### 1.1.1 Comunidade

Uma das vantagens da linguagem R é que, por ser Software Livre, tem uma comunidade de usuários muito grande, que contribui melhorando o código e criando documentação e tutoriais para outros usuários. Além do site do R-Project, o R-bloggers (em inglês) e o bRbloggers reúnem diversos sites/blogs sobre a linguagem R, com tutoriais e informações atualizadas sobre pacotes

---

<sup>1</sup>[https://pt.wikipedia.org/wiki/Ambiente\\_de\\_desenvolvimento\\_integrado](https://pt.wikipedia.org/wiki/Ambiente_de_desenvolvimento_integrado)

<sup>2</sup>Contributors: <https://www.r-project.org/contributors.html>

e códigos. O Stack Overflow é uma plataforma que possibilita os usuarios a fazerem perguntas relacionadas a qualquer linguagem de programação e obter respostas de outros usuarios. Procurar a sua dúvida lá pode ser bem útil para conseguir avançar no seu código. Para uma melhor aprendizagem, é essencial pesquisar documentação, tutoriais e, principalmente, o que tem sido discutido na comunidade de usuários do R ao redor do mundo. Para começar, é bom acompanhar os três sites citados.

## 1.2 Instalando o R

A linguagem R é disponibilizada como um Software Livre <sup>3</sup>, sem custos e seu código-fonte é aberto e por ser acessado por qualquer pessoa, sobre os termos da Free Software Foundation's GNU General Public License. Para que o seu computador seja capaz de interpretar a linguagem R é necessário que você faça download e instale a linguagem no seu computador, seguindo os passos:

- Acesse o link: <https://cran.r-project.org/mirrors.html> e escolha um *mirror* para realizar o *download*, dê preferência um no Brasil.
- Na página que abriu escolha o *download* para o seu sistema operacional (Windows, MacOS ou Linux).
- Após o download abra o arquivo e realize a instalação da forma que o seu sistema operacional exige.

## 1.3 Instalando o RStudio

Só a instalação do R é o suficiente para começar a usá-lo, mas há algumas IDEs que melhoram a interface para o usuário, deixando-a mais prática e fácil de utilizar os recursos e ferramentas disponíveis. Uma das mais utilizadas é o RStudio, que também é um software livre. O download<sup>4</sup>. Para realizar o download para qualquer sistema operacional acesse o link: <https://www.rstudio.com/products/rstudio/download/#download>. Escolha o sistema operacional correspondente ao seu computador, faça o download do instalador e siga a instalação.

---

<sup>3</sup>Licença R Project: <https://www.r-project.org/COPYING>

<sup>4</sup>Há também uma versão que pode ser utilizada pelo navegador do seu computador e acessada via servidor

# Chapter 2

## Aula 1

### 2.1 Janela de comando do R

Abra o R no seu computador. Essa janela é a nossa comunicação com o R. É nesse local que passamos as informações para que sejam interpretadas pela linguagem. A aparência desse ambiente é de um *prompt* de comando, em que comandos são digitados pelo usuário e ao apertar a tecla **enter** esses comandos são enviados para o R e processados. Ao finalizar o processamento do comando, se o comando solicitar algum retorno, o R pode: a) retornar a saída do comando ou b) retornar uma mensagem de erro (se há algum erro na sintaxe do comando).

Isso é comum em todas as linguagens de programação. O que realmente acontece é que enviamos os comandos em uma linguagem de alto nível (mais próxima do usuário) e o programa traduz esses comandos para uma linguagem de baixo nível na qual a máquina consegue interpretar. O R, por ser uma linguagem interpretativa, faz esse processo diretamente sem que seja necessária a utilização de compiladores mediando essa tradução. Isso facilita a comunicação com o usuário

Digite qualquer número na linha de comando e aperte **enter**. Como a seguir:

```
1456
```

```
## [1] 1456
```

O R processou essa informação e retornou um resultado. Como apenas inserimos um número, sem nenhum comando ou cálculo, ele retornou apenas o próprio número.

O que é importante entendermos aqui é que essa tela de comando é dinâmica, de modo que cada linha é uma entrada e corresponde

a uma comunicação com o computador. Dessa forma, não podemos voltar e alterar a linha anterior pois aquele comando já foi enviado e processado.

Se você está em uma linha nova você pode utilizar as setas para cima e para baixo do seu teclado para navegar entre os comandos já utilizados. E, caso queira, apertar **enter** para enviar o comando novamente.

## 2.2 O R como uma calculadora

O R, como outras linguagens de programação, pode funcionar como uma calculadora interativa. Assim, podemos fazer contas no console. Por exemplo, para fazer uma soma:

```
9 + 2
```

```
## [1] 11
```

Agora ele nos retornou o resultado da soma que solicitamos. Teste outras contas básicas utilizando os seguintes símbolos:

- “+” adição
- “-” subtração
- “\*” multiplicação
- “/” divisão
- “^” potência

Se você digitar um comando incompleto, como `5 +`, e apertar **enter**, o R mostrará um `+`, o que não tem nada a ver com somar alguma coisa. Isso significa que o R está esperando que você complete o seu comando. Termine o seu comando ou aperte **esc** para recomeçar.

O R ainda tem outros símbolos relacionados a contas matemáticas. A seguinte sequência de caracteres `%%` irá retornar o resto da divisão entre dois números. Por exemplo:

```
5 %% 3
```

```
## [1] 2
```

Como a divisão de 5 por 3 não é exata, ao solicitarmos o resto da divisão com símbolo `%%` o R retorna 2. Teste uma divisão exata. O que ele retorna?

Por fim, a sequência `%%` retorna a parte inteira da divisão de X por Y. Tente novamente na divisão de 5 por 3. O que aconteceu? Qual a diferença?

**Exercise 2.1.** Faça cálculos mais complexos, com várias operações combinadas.

**Importante:** Tente entender como o R faz essas operações. **Atenção nos parênteses!!**



**ERROS** Se você digitar um comando que o R não reconhece, ele retornará uma mensagem de erro. **NÃO ENTRE EM PÂNICO!** Ele só está avisando que não conseguiu interpretar o comando. Você pode digitar outro comando normalmente em seguida ou corrigir o anterior.

Aprender esse tipo de funcionalidade no R, apesar de bem básico, é bastante útil para começarmos a aprender as primeiras interações com a linguagem. Porém, obviamente, não é para isso que queremos aprender alguma linguagem de programação, mas sim para automatizar processos, evitar repetições desnecessárias e realizar análises mais avançadas do que operações de soma e multiplicação.

## 2.3 Objetos e classes de objetos

Já enviamos informações de números para o R e fizemos operações básicas com ele. Mas e se agora nós enviarmos uma letra ou palavra para ele? Tente fazer isso.

```
a
```

```
## Error in eval(expr, envir, enclos): object 'a' not found
```

Ele retorna o erro. Preste bem atenção no que está escrito no erro. No caso, ele está nos informando que o objeto `a` não foi encontrado. Para entendermos isso, precisamos entender o que é um **objeto**.

Um **objeto**<sup>1</sup>, em termos da ciência da computação, é um valor armazenado na memória do computador. Ele pode assumir a forma de variáveis, funções<sup>2</sup> ou estruturas de dados<sup>3</sup>. O objeto que importa para nós, por ora, são aqueles que representam variáveis. Nas próximas etapas do curso, trabalharemos com funções mas não iremos considerá-las como objetos.

Basicamente, no R, funções são ações realizadas em objetos. Porém, como tudo o que existe no R é um objeto, funções são objetos também.

Então, se um objeto armazena valores, conseguiremos fazer processos mais avançados e evitar repetições nos nossos códigos daqui pra frente. O objeto mais simples que encontramos no R é aquele que armazena apenas uma informação na memória.

A primeira coisa que precisamos definir é um nome para esse objeto (ou variável) e para isso precisamos seguir algumas preceitos - obrigatórios ou aconselhados.

---

<sup>1</sup>Para mais informações sobre objetos: <https://pt.stackoverflow.com/questions/205482/em-programa%C3%A7%C3%A3o-o-que-%C3%A9-um-objeto>

<sup>2</sup>[https://pt.wikipedia.org/wiki/M%C3%A9todo\\_\(programa%C3%A7%C3%A3o\)](https://pt.wikipedia.org/wiki/M%C3%A9todo_(programa%C3%A7%C3%A3o))

<sup>3</sup>[https://pt.wikipedia.org/wiki/Estrutura\\_de\\_dados](https://pt.wikipedia.org/wiki/Estrutura_de_dados)

- **Nomes válidos** para variáveis (obrigatório)<sup>4</sup>

1. **TEM QUE COMEÇAR COM UMA LETRA OU UM PONTO (.)**  
– exemplo `nome`, `genero23`, `.name` são nomes válidos
2. **NÃO PODE COMEÇAR COM NÚMEROS**  
– exemplo `1x0meunome` são nomes inválidos
3. **SE O PONTO (.) FOR O PRIMEIRO DÍGITO, ELE NÃO PODE SER SEGUIDO DE UM NÚMERO**  
– exemplo: `.23bananas`, `.45ois` são nomes inválidos
4. **\*\*PODE CONTER UNDERSCORE \_, DESDE QUE NÃO SEJA O PRIMEIRO DÍGITO\*\***  
– exemplo: `meu_nome`, `hello_world` são nomes válidos
5. **NÃO PODE CONTER ESPAÇO EM BRANCO**  
– exemplo: `meu nome`, `idade jovens` são nomes inválidos
6. **NÃO PODE SER UMA DAS PALAVRAS RESERVADAS DA LINGUAGEM**  
– Palavras reservadas<sup>5</sup> são: `if` `else` `repeat` `while` `function` `for` `in` `next` `break` `TRUE` `FALSE` `NULL` `Inf` `NaN` `NA` `NA_integer_` `NA_real_` `NA_complex_` `NA_character_` são nomes inválidos

- **Boas práticas** para criação de variáveis

1. **CAIXA BAIXA** R é *case sensitive*, de modo que ele interpreta como variáveis diferentes `Meu_nome`, `MEU_NOME`, `meu_nome`, `MeU_NoMe` e qualquer outra variação possível. Portanto é recomendado que sejam criadas apenas variáveis em caixa baixa, para evitar confusão.
2. **VARIÁVEIS COM PONTOS** Ainda que seja permitido pela linguagem, é recomendável evitar variáveis como `meu.nome` pois diversos pacotes utilizam `.` no nome das funções. Para fazer com que seu código seja mais fácil de ser lido por você e por outras pessoas, é bom evitar esse tipo de sintaxe no nome das variáveis.
3. **VARIÁVEIS COM NOMES DE FUNÇÕES** Também para evitar confusões, evite utilizar nomes de funções do R como nomes de variáveis, como `mean`, `sum`, etc.

Iremos indicar outras **boas práticas** mais adiante no curso.

Bom, agora que já sabemos como podemos nomear nossos objetos, vamos criar nosso primeiro objeto. Para criar um objeto precisamos armazenar algo dentro dele. Para que o R entenda que queremos armazenar algo em uma variável, utilizamos o símbolo `<-`.

<sup>4</sup>[https://cran.r-project.org/doc/FAQ/R-FAQ.html#What-are-valid-names\\_003f](https://cran.r-project.org/doc/FAQ/R-FAQ.html#What-are-valid-names_003f)

<sup>5</sup><https://stat.ethz.ch/R-manual/R-devel/library/base/html/Reserved.html>

O R aceita `=` como atribuição de variável, mas não é recomendado, por poder ser confundido com operadores lógicos e argumentos de funções.

O R interpreta como o nome da variável o lado para o qual a seta está apontando e como valor atribuído o que está no lado oposto. Assim, atribuímos valores a variável `x` de qualquer um dos modos: `x <- 5` ou `5 -> x`. Porém, para evitar confusão, usaremos aqui sempre o primeiro formato

Sendo assim, criemos `x`:

```
x <- 5
```

Como só realizamos a atribuição da variável o R não retornou nada no *prompt* de comando. Para que ele retorne o que está armazenado dentro de uma variável precisamos pedir para que ele faça isso. Por sorte, a sintaxe do R possibilita que façamos isso de forma simples: apenas digitando o nome da variável e apertando **enter**.

```
x
```

```
## [1] 5
```

Pronto! Agora pedimos para o R nos mostrar o que é `x` ou, equivalentemente, o que está armazenado na variável `x`. Dessa forma, podemos fazer cálculos com `x`. Por exemplo:

```
x * 5
```

```
## [1] 25
```

E vejam só, podemos armazenar esse resultado em uma nova variável.

```
y <- x * 5
```

Agora temos duas variáveis: uma `x`, que tem armazenada o número 5, e `y`, que tem armazenada o resultado da operação `x * 5`.

Podemos ir além e criar uma variável `z` que armazena a divisão de `x` por `y`. Qual valor estará armazenado em `z`?

Certo! Mas e se agora digitarmos `x <- 10` quanto vai valer `x`? O que aconteceu?

**ATENÇÃO!** Chegamos em uma parte muito importante do curso. Quando re-atribuímos uma variável já existente nós apagamos o valor que estava dentro dela e escrevemos um valor novo.

Ok! Já sabemos como criar, substituir e fazer contas com variáveis. Só que até agora só trabalhamos com números, mas variáveis podem - e devem - armazenar outros tipos de dados.

Neste caso precisamos de objetos diferentes, ou seja, de classes e formas de armazenamentos diferentes. O R apresenta **quatro** principais classes de objetos: `integer`, `numeric`, `character` e `logical`. Vamos ver cada uma delas:

- **integer** Um objeto de classe **integer** é uma variável numérica que armazena apenas números inteiros (sem pontos flutuantes - os decimais). A diferença para a classe **numeric** é que a classe **integer** armazenam números com menos casas e não consegue fazer cálculos com casas decimais. Porém, quando declaramos uma variável numérica o R automaticamente a armazena como tipo **numeric**. É possível forçar a alteração dessa variável para **integer** se a sua variável não for ser utilizada para fazer cálculos e for necessário ocupar menos espaço de armazenamento (por exemplo - ID number). Abreviação: **int**
- **numeric** Classe padrão para objetos numéricos. Armazena números inteiros e do tipo **double** (ou **float**), ou seja, com casas decimais. Necessários para realização de cálculos matemáticos. Abreviação: **num**
- **character** Classe do tipo string que armazena dados textuais. Mesmo que haja uma sequência de números os dados serão lidos como textos. Não é possível fazer operações matemáticas e aplicar funções de variáveis numéricas. Abreviação: **char**
- **logical** Classe do tipo lógica (booleana), permite apenas armazenamentos de dados lógicos (**TRUE** ou **FALSE**). Utilizada para fazer operações lógicas com os dados. Também pode ser utilizada como variável binária. Abreviação: **logi**

Ainda há uma classe de variável para números complexos, utilizada para análises matemáticas mais avançadas mas não utilizaremos nesse curso.

Agora que já sabemos as classes de objetos. Vamos criar objetos para cada classe. Para criar objetos **integer** basta criar uma variável com números, assim como a variável da classe **numeric**. Porém, automaticamente o R cria variáveis numéricas como da classe **numeric**. Então, para que forcemos o R a armazenar um número como **integer**, é necessário colocar no final do número a letra L. Assim, `inteiro <- 8L` será do tipo **integer** e não **numeric**. O L só entra para avisar o R que queremos que aquela variável seja de outra classe. Se pedirmos para ele nos retornar a variável armazenada ele irá retornar apenas a parte numérica da variável (sem o L).

```
inteiro <- 8L
inteiro
```

```
## [1] 8
```

Para criar variáveis de classe **character** também precisamos avisar para o R que aquela variável não é do tipo numérico. Sendo assim, não podemos criar a variável da seguinte forma:

```
string <- texto
```

```
## Error in eval(expr, envir, enclos): object 'texto' not found
```

Para que o R identifique que a variável é da classe **character** precisamos definir o valor entre aspas (" "):

```
string <- "texto"
```

Agora sim, o R interpreta o que está dentro dos parênteses como uma sequência de caracteres. Qualquer sequência de caracteres dentro das aspas será interpretada como um objeto da classe **character**, inclusive números, espaços e caracteres especiais.

Por fim, para criar uma variável do tipo lógica (que so aceitam **TRUE** e **FALSE**) é necessário atribuir valores **TRUE**, **T**, **FALSE** OU **F**. Atenção: não podem estar entre aspas, se não o R interpreta como **character**.

```
log <- FALSE
```

Uma outra forma de utilizar os objetos booleanos é comparando variáveis a partir de **operadores relacionais**. Podemos perguntar para o R se uma variável é **igual** a outra (**==**), **diferente** (**!=**), **maior** ou **menor** (**>** ou **<**) ou **maior/menor ou igual** (**<=** ou **>=**). Como é uma pergunta utilizando sinais de comparação a resposta vai sempre ser do tipo **logical**, ou seja, **TRUE** e **FALSE**.

Assim, podemos perguntar para o R se **x** é maior que **y**:

```
x > y
```

```
## [1] FALSE
```

O **!** pode ser combinado com qualquer outro operador: **>!** (não é maior)

O R nos retorna a resposta **FALSE**, pois **x** não é maior que **y**. (**x** já havia sido definido anteriormente como 5 e **y** como 25).

**Exercise 2.2.** Utilize os outros operadores relacionais para comparar outras variáveis.

Há ainda os operadores lógicos, que baseiam-se na Teoria dos Conjuntos. São eles: **&&** (**AND/E**), **||** (**OR/OU**). Utilizando esses operadores podemos combinar perguntas para o R. A seguir, pergunto para o R se **x** é maior que **y** **E** se **y** é maior que 10. De modo que o R irá retornar **FALSE** se pelo menos uma das condições é falsa e irá retornar **TRUE** se, e somente se, as duas forem verdadeiras.

```
x > y || y > 10
```

```
## [1] TRUE
```

**Exercise 2.3.** Utilize o operador lógico “OU”. Em quais condições o R retorna **FALSE** e em quais ele retorna **TRUE**?

Por fim, se definirmos uma variável e queremos saber como o R a armazenou podemos utilizar a função **class()** para descobrir a classe do objeto.

Atenção! Essa é a primeira vez que utilizamos uma função no R. Funções tem uma sintaxe bem definida e argumentos obrigatórios e opcionais.

A função `class()` necessita apenas que seja inserido dentro dos parênteses o objeto que queremos saber a classe. Assim,

```
class(x)
```

```
## [1] "numeric"
```

Perguntamos para o R qual era a classe de `x`, no qual ele respondeu que `x` é da classe `numeric`. Faça isso com objetos de outras classes.

Ainda que possamos fazer várias coisas interessantes com o esse tipo mais simples de objeto, trabalhar com objetos mais complexos nos abre mais possibilidades para análises de dados. Porém, antes de avançarmos nos tipos de objetos, vamos ver algumas outras coisinhas que podem nos ajudar no aprendizado aqui para frente.

## 2.4 Ambiente do RStudio

Até agora trabalhamos apenas com o console do R. A vantagem dele é que ele é mais dinâmico e mais rápido, mas temos dificuldades de, por exemplo, ver quais variáveis já temos atribuídas e armazenadas na memória. De fato, se entrarmos com a função `ls()` o R irá retornar todas as variáveis já atribuídas e armazenadas. Porém, isso dificulta um pouco o trabalho da análise de dados pois temos que ficar solicitando a função toda vez que precisamos saber as variáveis armazenadas.

Para ajudar a comunicação com o usuário foram criadas IDEs (do inglês, *Integrated Development Environment*) para que facilitem algumas visualizações. Após a implementação e disponibilização dessas ferramentas o uso de várias linguagens de programação, inclusive o R, aumentou consideravelmente. Uma delas é o RStudio, software da empresa do mesmo nome que tem desempenhado um papel muito importante dentro da comunidade, criando documentação e ferramentas de aprendizagem para aumentar a utilização do R.

O ambiente do RStudio dispõe de 4 janelas principais. A janela `console` é a mesma que estávamos trabalhando antes. Ao clicarmos em `File > New File > R Script` criamos um arquivo de `script`. Esse arquivo funciona como um bloco de notas, o R só irá lê-lo se selecionarmos as linhas com os comandos e apertarmos `ctrl+enter`. é importante salvar os arquivos em seu computador com a extensão `.R`.

Na janela `Environment` são mostradas as variáveis que já estão atribuídas na memória. Por fim, a última janela (canto inferior direito) apresenta na aba `Files` os arquivos que estão na pasta raiz do projeto, na aba `Plots` os gráficos

quando solicitamos que o R imprima gráficos na tela, na aba **Packages** os pacotes instalados no computador e na aba **Help** podemos ver a documentação das funções e pacotes (esse precisa de conexão na internet).

## 2.5 Pacotes

Quando instalamos o R no computador, apesar de vir com várias funções básicas, essa instalação inicial não comporta todas as possibilidades de funcionalidade do R. Para tanto, existem os pacotes, que são bibliotecas de funções e até mesmo dados que possibilitam complementar ou otimizar tarefas. Há uma infinidade de pacotes criados pela comunidade de usuários (você pode criar um pacote se quiser) que são disponibilizados no site CRAN. Para usar um pacote você precisa primeiro instalá-lo e quando for utilizá-lo precisa “chamar” ele no código, mas isso veremos mais para frente. As IDEs possibilitam que a busca e a instalação desses pacotes também seja mais prática.

## 2.6 Como conseguir ajuda

Para buscar ajuda você pode ir na aba **Help** do RStudio e procurar o nome da função ou pacote na documentação disponível do R ou ir diretamente no console e digitar `?função` (onde função é o nome da função desejada).

Faça buscas na internet e em sites especializados (como o Stackoverflow) para encontrar documentação referente as funções e funcionalidades do R.

## 2.7 Vetores

Bom, depois dessa pequena pausa vamos continuar conhecendo outros tipos de objetos no R. Até agora já trabalhamos com objetos bem simples que armazenam apenas uma informação por vez mas, na maioria das vezes, precisamos conseguir armazenar em uma mesma variável muitas informações ao mesmo tempo. Um tipo de objeto um pouco mais complexo é o do tipo **vetor**. A sintaxe para a criação de um vetor é `c()`, onde os itens internos do vetor são separados por vírgula. Assim, se criamos um vetor `c(2, 3, 4, 5)`, criamos um vetor de tamanho 4 que armazena os itens 2, 3, 4 e 5. O vetor seria o equivalente a uma linha de uma tabela.

A partir de agora passaremos a tratar de tamanhos de objetos. O vetor nada mais é que uma matriz de tamanho 1 x c, ou seja, 1 linha e c colunas.

A principal característica do vetor é que ele só armazena objetos do mesmo tipo, ou seja, só podemos ter vetores só de `integer` ou só de `numeric` ou só de `character` ou só de `logical`. Dessa forma, o vetor irá herdar a classe dos objetos que ele contém. Sendo assim, para termos vetores da classe `numeric` precisamos criar um vetor com objetos da classe `numeric` dentro dele.

**Exercise 2.4.** Crie um vetor de classe `numeric`, `logical` e `character` e atribua cada um deles a uma variável diferente.

**Exercise 2.5.** Pergunte ao R a classe de cada um dos vetores.

Bom, como o vetor herda as características da classe então podemos fazer operações matemáticas com os vetores numéricos.

**Exercise 2.6.** Multiplique o seu vetor `numeric` por um número.

O que aconteceu? Ele mudou o seu vetor original?

**Exercise 2.7.** Agora crie mais um vetor `numeric` (do mesmo tamanho) e multiplique os dois vetores.

O que o R fez?

E se tivéssemos vetores de tamanho diferente?

**Exercise 2.8.** Multiplique dois vetores `numeric` de tamanhos diferentes.

O que aconteceu?

O R, diferente de outras linguagens, quando solicitado que faça operações com vetores de tamanhos diferentes ele faz uma reciclagem: alinha os dois vetores e, caso não possuam o mesmo tamanho, vai repetindo o vetor menor até completar o vetor maior. Outras linguagens não permitiriam a operação e retornariam um erro.

A letra `c` na sintaxe da criação de um vetor vem da palavra `combine` pois o vetor nada mais é do que a combinação de vários objetos na sequência.



### 2.7.1 Nomeação de vetores

Podemos vincular nomes aos vetores com a função `names()`, da seguinte forma:

```
sacola1 <- c(10, 5, 8, 7)
names(sacola1) <- c("Laranja", "Pera", "Uva", "Maça")
```

Agora quando pedimos para ver o vetor nomeado ele aparece da seguinte forma:

```
sacola1

## Laranja   Pera    Uva    Maça
##      10      5      8      7
```

No caso, como temos apenas uma sacola de feira, o jeito que fizemos faz sentido: atribuímos um vetor e depois usamos a função `names` para nomeá-lo. Mas e se tivermos mais de um vetor de sacolas de feira, com as mesmas características dentro?

```
sacola2 <- c(5, 9, 7, 6)
sacola3 <- c(8, 7, 5, 4)
sacola4 <- c(9, 12, 3, 9)
sacola5 <- c(5, 3, 10, 12)
```

Agora faz sentido otimizarmos a nossa função. Para isso criamos um vetor chamado `nomes` e associamos ele a cada uma dos vetores:

```
nomes <- c("Laranja", "Pera", "Uva", "Maça")
names(sacola2) <- nomes
names(sacola3) <- nomes
names(sacola4) <- nomes
names(sacola5) <- nomes
```

Pronto! Conseguimos nomear todos os vetores de forma mais rápida.

### 2.7.2 Operações com vetores

Agora queremos saber qual foi o total de cada fruta comprada. Como já sabemos como fazer operações matemáticas com vetores precisamos apenas somar os vetores correspondentes das nossas sacolas de feira e adicionar em um vetor de soma.

```
soma_feira <- sacola1 + sacola2 + sacola3 + sacola4 + sacola5
soma_feira
```

```
## Laranja   Pera    Uva    Maça
##      37      36      33      38
```

Agora o nosso vetor `soma_feira` armazena a soma de cada fruta em todas as sacolas de feira. Assim, podemos responder: quantas laranjas foram compradas

no total? Qual foi a fruta mais comprada?

O vetor total não é nomeado. Podemos nomeá-lo. Usando a função `names`.

**Exercise 2.9.** Nomeie o vetor total de sacolas de feira.

Mas e se quisermos saber a soma de itens de cada sacola? Nesse caso, fazer operações com vetores não nos ajuda. Em outras linguagens de programação teríamos que programar laços de repetição e laços condicionais para realizar essa operação e, se quiséssemos utilizar em outras situações (reaproveitá-la) teríamos que programar uma função. O R por já vir com várias funções pré-programadas facilita o nosso trabalho. Sendo assim, precisamos apenas utilizar a função `sum()`<sup>6</sup> (olhe a documentação da função).

**Exercise 2.10.** Faça a soma de cada um dos vetores de sacolas utilizando a função `sum()`. Armazene cada resultado em uma nova variável.

**Exercise 2.11.** Descubra o total de itens comprados na feira utilizando a função `sum()`. Armazene cada resultado em uma nova variável.

### 2.7.3 Selecionar elementos dentro de um vetor

Algo útil para fazermos é conseguirmos selecionar elementos dentro dos vetores. Para isso precisamos entender o conceito de **endereçamento**. Por exemplo, o vetor de tamanho quatro tem quatro espaços dentro dele, dentro de cada um desses espaços está armazenado um objeto (um número, um texto ou um objeto de tipo lógico), ou seja, esse vetor tem 4 “endereços” dentro dele. Isso nos facilita pois nem sempre sabemos o que tem dentro do vetor e para descobrirmos não precisamos pedir para o R nos retornar o vetor inteiro (às vezes o vetor é muito grande e esse procedimento se torna inviável).

Para indicarmos que queremos ver algo dentro de um **endereço** utilizamos a sintaxe de `[]`. Para indicarmos um endereço de um vetor utilizamos a sintaxe `vetor[x]`, onde `x` indica um número de 1 ate `n` (a maior casa do vetor).

---

<sup>6</sup>Sum function: <https://www.rdocumentation.org/packages/base/versions/3.6.1/topics/sum>

**Exercise 2.12.** Selecione o elemento 4 do vetor da sacola 3. Atribua esse valor a uma variável.

Podemos também selecionar mais de um valor do vetor. A sintaxe é indicar dentro dos `[]` a seleção a ser selecionada. Veja que isso indica um novo vetor então a sintaxe também deve seguir a sintaxe de um vetor: `vetor[c(x, y, z)]`. Se os números indicam um intervalo podemos utilizar a sintaxe `vetor[x:y]`.

**Exercise 2.13.** Selecione do vetor `sacola3` os valores 1 e 3 e do vetor `sacola2` os valores de 2 a 4.

Para vetores nomeados é possível selecionar a partir dos nomes. A lógica é a mesma, mas dentro dos `[]` adicionamos o nome entre " " (por ser uma variável de classe textual). Para selecionarmos mais de um nome também fazemos do mesmo jeito.

**Exercise 2.14.** Selecione a quantidade de laranjas do vetor da sacola 4. Selecione a quantidade de peras e maçãs do vetor da sacola 5.

#### 2.7.4 Comparações de vetores e entre vetores

Assim como fizemos comparações entre os objetos criados, podemos fazer comparações entre os vetores. Compare o vetor `sacola1` com o vetor `sacola2`. Quando utilizamos os operadores relacionais entre dois vetores o R retorna a comparação de cada um dos itens, ou seja, a comparação do item 1 do primeiro vetor com o item 1 do segundo vetor e assim por diante. Retornando `TRUE` ou `FALSE` em cada um deles. Em vetores nomeados com os mesmos nomes esse procedimento é ainda mais fácil pois o R nos retorna o resultado das comparações com os nomes dos vetores.

**Exercise 2.15.** Qual dos itens do vetor `sacola2` são maiores que os itens do vetor `sacola4`?

Também podemos fazer essas comparações com a soma de cada sacola.

**Exercise 2.16.** A soma dos itens da `sacola3` é menor que a soma dos itens da `sacola5`?

Podemos utilizar os mesmos sinais de comparação para saber quais os valores de um vetor são maiores que um número, por exemplo. Assim se perguntarmos ao R `vetor > 5` ele irá nos responder elemento por elemento se é ou não maior que cinco (respondendo com operadores lógicos).

**Exercise 2.17.** Veja quais casas do vetor da sacola 5 tem valores maiores ou iguais a 10.

Porém, seria mais útil se tivéssemos como retorno os números que correspondem a busca que desejamos. Para isso, precisamos inserir `vetor[vetor > x]`. Aqui o R irá retornar os números que são maiores que x. No caso do vetor nomeado a resposta virá acompanhada dos nomes. Lembre-se que dessa forma não temos como saber quais casas tem os valores retornados, o R retorna os elementos que são TRUE no seu vetor mas não indica qual era a posição original. Esse é o nosso primeiro contato um tipo de filtragem dos dados.

**Exercise 2.18.** Armazene em uma variável os valores do vetor `sacola1` que são menores do que 10.

### 2.7.5 Adição e exclusão de valores em um vetor

Podemos adicionar e excluir valores de um vetor existente. Para adicionarmos valores em um vetor podemos fazer de três formas distintas:

- **Por endereçamento direto** - dessa forma precisamos indicar dentro dos colchetes o endereço da última casa mais um:
  - `vetor[x+1] <- 5`
- **Por endereçamento indireto** - dessa forma não precisamos saber o tamanho do vetor, indicamos a partir da função `length()` o tamanho do vetor + 1
  - `vetor[length(vetor) + 1] <- 9`
- **Por recursividade** - substituímos o vetor original por um novo vetor que tem o vetor original seguido de uma casa antes e/ou depois
  - `vetor <- c(vetor, 10)`
  - `vetor <- c(10, vetor)`

Das duas primeiras formas, se indicamos um número diferente de 1 no endereço o R, diferente de outras linguagens, coloca NA nas casas entre a última casa do vetor e a casa nova atribuída. Veja o exemplo a seguir.

```
vetor1 <- c(1, 2, 4)
vetor1[5] <- 5
vetor1
```

```
## [1] 1 2 4 NA 5
```

Como o vetor tinha inicialmente 3 casas e indicamos um valor para uma 5ª casa, o R criou a 5ª casa com o valor indicado, mas precisava para tanto criar uma 4ª casa também. Essa ele incluiu um valor NA pois o valor dessa casa não foi indicado.

**Exercise 2.19.** Adicione os valores de soma, calculados anteriormente, nos respectivos vetores de sacolas de feira.

Por fim, para excluir elementos de um vetor, indicamos por recursividade o vetor dentro de [] os valores que queremos excluir com um traço na frente. Se queremos excluir um intervalo indicamos [-c(x:y)] e se queremos excluir números específicos indicamos [-c(x, y, z)].

```
vetor1 <- vetor1[-c(4)]
vetor1
```

```
## [1] 1 2 4 5
```

No caso da exclusão não é necessário que seja por recursividade. É possível atribuir um novo vetor para o procedimento.

### 2.7.6 Funções com vetores

Na última parte dessa aula iremos aprender algumas funções úteis para utilizar em vetores. Ainda que possamos criar vetores de sequência utilizando apenas a sintaxe x:y, por exemplo as funções rep() e seq() nos auxiliam a criar sequências mais personalizadas.

```
#Criando sequências sem as funções rep() ou seq()
seq1 <- 1:8
seq1
```

```
## [1] 1 2 3 4 5 6 7 8
```

```
seq2 <- 2.5:10
seq2
```

```
## [1] 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5
```

A função `seq()` possibilita personalizar outros argumentos para a criação da sequência. O principal argumento é o `by` em que podemos definir o incremento da sequência. Por exemplo, se quero uma sequência de 0 a 10 de 2 em 2 utilizo: `seq(0, 10, by = 2)`. Veja outros argumentos na documentação da função `seq()`.

**Exercise 2.20.** Crie um vetor com uma sequência de numeros iniciando em -20 e indo ate 50 de 5 em 5. Atribua a uma variável.

A função `rep()` possibilita a criação de sequências de numeros repetidos. Por exemplo, se utilizarmos `rep(5,3)` iremos ter um vetor com o número 5 repetido 3 vezes.

```
rep(5,3)
```

```
## [1] 5 5 5
```

Para ter um intervalo podemos utilizar `rep(x:y, n)`, ou seja, um intervalo que vai de x a y repetido n vezes.

```
rep(1:5, 3)
```

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

Por fim, podemos criar um intervalo repetindo cada número do intervalo n vezes.

```
rep(1:5, each = 3)
```

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

Outras funções úteis para aprendermos agora são:

- `mean(x)` - cálculo da média do vetor x
- `var(x)` - cálculo da variância do vetor x
- `max(x)` - o valor máximo encontrado no vetor x
- `min(x)` - o valor mínimo encontrado no vetor x
- `sd(x)` - o desvio padrão do vetor x
- `range(x)` - a amplitude do vetor x
- `length(x)` - o tamanho do vetor x
- `rev(x)` - inverter o vetor x **Atenção: isso não altera o vetor original**

Se colocarmos duas ou mais classes diferentes dentro de um mesmo vetor, o R vai forçar que todos os elementos passem a pertencer

à mesma classe. Ordem de preferência: `character` > `complex` >  
`numeric` > `integer` > `logical`

## 2.8 Listas de exercícios

Faça as listas de exercícios referente a essa aula no repositório do curso.





## Chapter 3

### Aula 2



## Chapter 4

### Aula 3



## Chapter 5

## Aula 4



## Chapter 6

## Aula 5