



## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

# Introdução ao uso do software R

11 de maio de 2014



# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# Sumário

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# Histórico

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

- 1980 Linguagem S: desenvolvida por R. Becker, J. Chambers e A. Wilks (AT&T Bell Laboratories)
- 1980 Versão comercial: S-Plus (Insightful Corporation)
- 1996 Versão livre: R desenvolvido por R. Ihaka e R. Gentleman (Universidade de Auckland)



# Histórico

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

**1997** R Development Core Team

**Hoje** 20 desenvolvedores principais e muitos outros colaboradores em todo o mundo

- Estatísticos, matemáticos e programadores



# O que é o R?

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

- Programa estatístico para análise de dados e produção de gráficos
- Uma completa linguagem de programação:
  - Interpretada (contrário de compilada)
  - Orientada a objetos:

*“Tudo no R é um objeto. . .”*

- Livre distribuição (código-aberto)
- Mais de 2000 pacotes adicionais
- Disponível em <http://www.R-project.org>
- Versão atual: 3.1.0 (10/04/2014). Ciclo de lançamentos: 6 meses (versões menores), 1 ano (versões maiores).



# Vantagens

## Módulo I Básico

### Introdução

#### Configuração inicial

#### Visão geral

#### Funções e argumentos

#### Objetos Classes

#### Valores perdidos

#### Manipulação de dados

#### Indexação Seleção condicional

#### Finalizando o programa

- Software livre
- Funciona em praticamente todos os sistemas operacionais: Unix (Linux, FreeBSD, ...), Macintosh e Windows
- É o produto da cooperação entre estatísticos do mundo todo
- Linguagem lógica e intuitiva
- Flexibilidade nas análises estatísticas
- Gráficos de alta qualidade



# Desvantagens

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

- Sem interface gráfica (?)
- Não há visualização direta dos dados
- Curva de aprendizado longa
- Pode ser lento com grandes (GB, TB, ...) bases de dados
  - Necessidade de **vetorização**





# Sumário

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# Configurando o diretório de trabalho

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

- O diretório de trabalho é uma pasta onde o R será direcionado. Todos os arquivos que serão importados (base de dados, ...) ou exportados (base de dados, gráficos, ...) por ele ficarão nesta pasta.
- No sistema Windows, existem duas maneiras de configurar o diretório de trabalho (suponha que vamos usar a pasta `C:\cursoR`):



# Configurando o diretório de trabalho

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

- 1 Utilizando a função `setwd()` dentro do R:

```
> setwd("C:/cursoR")
```

Note que a barra é invertida!

- 2 Pelo menu do RStudio em `Session > Set Working Directory > Choose Directory ...`

Confira o diretório que está trabalhando com a função

```
getwd()
```



# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# O R como uma calculadora

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

O símbolo `>` indica que o R está pronto para receber um comando:

```
> 2 + 2
```

```
[1] 4
```

O símbolo `>` muda para `+` se o comando estiver incompleto:

```
> 2 *
```

```
+ 2
```

```
[1] 4
```

Espaços entre os números não fazem diferença:

```
> 2+      2
```

```
[1] 4
```



# O editor de scripts

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

- Para criar rotinas computacionais é necessário utilizar um editor de scripts.
- Clique em File > New file> R script. Salve com a extensão .R.
- Para enviar comandos diretamente para o console, selecione-os e aperte Ctrl + <Enter>.
- Para adicionar comentários ao script, utiliza-se o símbolo # antes do texto e/ou comandos. O que estiver depois do símbolo não será interpretado pelo R. Portanto:

```
2 + 2      # esta linha será executada  
# 2 + 2    esta linha não será executada
```



# Operadores aritméticos

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Operador	Significado
+	adição
-	subtração
*	multiplicação
/	divisão
^	potência
exp()	exponencial
sqrt()	raíz quadrada
factorial()	fatorial
log(); log2(); log10()	logaritmos



# Ordens de execução

## Módulo 1 Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos

### Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

As operações são realizadas sempre seguindo as prioridades:

- 1 De dentro para fora de parênteses ( )
- 2 Multiplicação e divisão
- 3 Adição e subtração

$5 * 2 - 10 + 7$

[1] 7

$5 * 2 - (10 + 7)$

[1] -7

$5 * (2 - 10 + 7)$

[1] -5

$5 * (2 - (10 + 7))$

[1] -75





# Exercícios

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Calcule a seguinte equação:  $32 + 16^2 - 25^3$
- 2 Divida o resultado por 345
- 3 Qual o resultado da expressão  $\frac{e^{-2}2^4 - 1}{4!}$ ?
- 4 E do logaritmo desta expressão?



# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# Funções e argumentos

Módulo I  
Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

As funções no R são definidas como:

```
nome(argumento1, argumento2, ...)
```

Exemplo: função `runif()` (para gerar valores aleatórios de uma distribuição uniforme):

```
runif(n, min = 0, max = 1)
```

```
runif(10, 1, 100)
```

```
[1] 60.987 18.436 73.635 79.109 36.117 11.424 14.115 75.931  
[9] 59.455 14.065
```



# Funções e argumentos

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados  
Indexação  
Seleção  
condicional

Finalizando o  
programa

Argumentos que já possuem um valor especificado (como `max` e `min`) podem ser omitidos:

```
runif(10)
```

Se os argumentos forem nomeados, a ordem deles dentro da função não tem mais importância:

```
runif(min = 1, max = 100, n = 10)
```

Argumentos nomeados e não nomeados podem ser utilizados, desde que os não nomeados estejam na posição correta:

```
runif(10, max = 100, min = 1)
```



# Outros tipos de argumentos

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Exemplo: função `sample()`:

```
sample(x, size, replace = FALSE, prob = NULL)
```

- `x` e `size` devem ser obrigatoriamente especificados
- `replace` é lógico: TRUE (T) ou FALSE (F)
- `prob` é um argumento vazio ou ausente (“opcional”)

Exemplo: função `plot()`:

```
plot(x, y, ...)
```

- “...” permite especificar argumentos de outras funções (por exemplo `par()`)



# Mecanismos de ajuda

## Módulo 1 Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Argumentos e detalhes do funcionamento das funções:

```
?runif
```

OU

```
help(runif)
```

A documentação contém os campos:

- **Description:** breve descrição
- **Usage:** função e todos seus argumentos
- **Arguments:** lista descrevendo cada argumento
- **Details:** descrição detalhada
- **Value:** o que a função retorna
- **References:** bibliografia relacionada
- **See Also:** funções relacionadas
- **Examples:** exemplos práticos



# Mecanismos de ajuda

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Procura por funções que contenham "palavra":

```
help.search("palavra")
```

Ajuda através do navegador (também contém manuais, ...):

```
help.start()
```

Busca por "palavra" nos arquivos da lista de discussão do R:

```
RSiteSearch("palavra")
```



# Criando uma função

## Módulo 1 Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

A ideia original do R é transformar usuários em programadores

Criar funções para realizar trabalhos específicos é um dos grandes poderes do R

Por exemplo, podemos criar a famosa função

```
ola.mundo <- function(){  
  writeLines("Olá mundo")  
}
```

E chama-la através de

```
ola.mundo()  
  
Olá mundo
```





# Criando uma função

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

A função acima não permite alterar o resultado de saída. Podemos fazer isso incluindo um **argumento**

```
ola.mundo <- function(texto){  
  writeLines(texto)  
}
```

E fazer por exemplo

```
ola.mundo("Funções são legais")
```

Funções são legais

(Veremos detalhes de funções mais adiante)



# Exercícios

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- Usando a função `runif()` gere 30 números aleatórios entre:
  - 0 e 1
  - -5 e 5
  - 10 e 500alternando a posição dos argumentos da função.
- Veja o help da função (?) "+"
- Crie uma função para fazer a soma de dois números: x e y



# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos**
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# Programação orientada a objetos

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

O que é um objeto?

- Um **símbolo** ou uma **variável** capaz de armazenar qualquer valor ou estrutura de dados

Por quê objetos?

- Uma maneira simples de acessar os dados armazenados na memória (o R não permite acesso direto à memória)

Programação:

- Objetos → Classes → Métodos



# Objetos

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos

### Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

*“Tudo no R é um objeto.”*

*“Todo objeto no R tem uma classe”*

- **Classe:** é a definição de um objeto. Descreve a forma do objeto e como ele será manipulado pelas diferentes funções
- **Método:** são **funções genéricas** que executam suas tarefas de acordo com cada classe. As funções genéricas mais importantes são:
  - `summary()`
  - `plot()`

Veja o resultado de

```
methods(summary)
methods(plot)
```



# Objetos

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

A variável `x` recebe o valor 2 (tornando-se um objeto dentro do R):

```
x <- 2
```

O símbolo `<-` é chamado de **operador de atribuição**. Ele serve para atribuir valores a objetos, e é formado pelos símbolos `<` e `-`, obrigatoriamente *sem espaços*.

Para ver o conteúdo do objeto:

```
x
```

```
[1] 2
```

Obs.: O símbolo `=` pode ser usado no lugar de `<-` mas não é recomendado.



# Objetos

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Quando você faz

```
x <- 2
```

está fazendo uma **declaração**, ou seja, declarando que a variável x irá agora se tornar um objeto que armazena o número 2. As declarações podem ser feitas uma em cada linha

```
x <- 2
```

```
y <- 4
```

ou separadas por ;

```
x <- 2; y <- 4
```



# Objetos

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Operações matemáticas em objetos:

```
x + x
```

```
[1] 4
```

Objetos podem armazenar diferentes estruturas de dados:

```
y <- runif(10)
```

```
y
```

```
[1] 0.55842 0.44183 0.97171 0.58974 0.42993 0.34257 0.79874  
[8] 0.21318 0.44120 0.22809
```

Note que cada objeto só pode armazenar uma estrutura (um número ou uma sequência de valores) de cada vez! (Aqui, o valor 4 que estava armazenado em y foi sobrescrito pelos valores acima.)





# Nomes de objetos

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- Podem ser formados por letras, números, “\_”, e “.”
- Não podem começar com número e/ou ponto
- Não podem conter espaços
- Evite usar acentos
- Evite usar nomes de funções como:

```
c q t C D F I T diff df data var pt
```

- O R é *case-sensitive*, portanto:

```
dados ≠ Dados ≠ DADOS
```



# Gerenciando a área de trabalho (*workspace*)

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Liste os objetos criados com a função `ls()`:

```
ls()
```

Para remover apenas um objeto:

```
rm(x)
```

Para remover outros objetos:

```
rm(x, y)
```

Para remover todos os objetos:

```
rm(list = ls())
```



# Exercícios

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Armazene o resultado da equação  $32 + 16^2 - 25^3$  no objeto `x`
- 2 Divida `x` por 345 e armazene em `y`
- 3 Crie um objeto (com o nome que você quiser) para armazenar 30 valores aleatórios de uma distribuição uniforme entre 10 e 50
- 4 Remova o objeto `y`
- 5 Remova os demais objetos de uma única vez



# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos**
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# Vetor

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

## Características:

- Coleção ordenada de valores
- Estrutura unidimensional

Usando a função `c()` para criar vetores:

```
num <- c(10, 5, 2, 4, 8, 9)  
num
```

```
[1] 10  5  2  4  8  9
```



# Vetor

## Sequências de números

Módulo I  
Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Usando a função `seq()`

```
seq(1, 10)
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

Ou `1:10` gera o mesmo resultado. Para a sequência variar em 2

```
seq(from = 1, to = 10, by = 2)
```

```
[1] 1 3 5 7 9
```

Para obter 15 valores entre 1 e 10

```
seq(from = 1, to = 10, length.out = 15)
```

```
[1] 1.0000 1.6429 2.2857 2.9286 3.5714 4.2143 4.8571  
[8] 5.5000 6.1429 6.7857 7.4286 8.0714 8.7143 9.3571  
[15] 10.0000
```



# Vetor

## Sequências de números

### Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Selecção  
condicional

Finalizando o  
programa

Usando a função `rep()`

```
rep(1, 10)
```

```
[1] 1 1 1 1 1 1 1 1 1 1
```

Para gerar um sequência várias vezes

```
rep(c(1, 2, 3), 5)
```

```
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

Para repetir um número da sequência várias vezes

```
rep(c(1, 2, 3), each = 5)
```

```
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```



# Vetor

## Operações matemáticas em vetores

### Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Selecção  
condicional

Finalizando o  
programa

Operações podem ser feitas entre um vetor e um número:

```
num * 2
```

```
[1] 20 10 4 8 16 18
```

E também entre vetores de mesmo comprimento ou com comprimentos múltiplos:

```
num * num
```

```
[1] 100 25 4 16 64 81
```

```
num + c(2, 4, 1)
```

```
[1] 12 9 3 6 12 10
```





# Vetor

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Selecção  
condicional

Finalizando o  
programa

## A Regra da Reciclagem

Original		Expandido		Resposta
num	c(2,4,1)	num	c(2,4,1)	num + c(2,4,1)
10	2	10	2	12
5	4	5	4	9
2	1	2	1	3
4		4	2	6
8		8	4	12
9		9	1	10

Agora tente:

```
num + c(2, 4, 1, 3)
```



# Vetor

## Atributos de objetos

### Módulo I Básico

Introdução  
Configuração  
inicial  
Visão geral  
Funções e  
argumentos  
Objetos  
Classes  
Valores  
perdidos  
Manipulação  
de dados  
Indexação  
Seleção  
condicional  
Finalizando o  
programa

Classe:

```
class(num)
```

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

Comprimento:

```
length(num)
```

```
[1] 6
```



# Vetor

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Vetores também podem ter outras classes:

- Vetor de caracteres:

```
caracter <- c("brava", "joaquina", "armação")  
caracter  
[1] "brava"      "joaquina" "armação"
```

- Vetor lógico:

```
logico <- caracter == "armação"  
logico  
[1] FALSE FALSE TRUE
```

ou

```
logico <- num > 4  
logico  
[1] TRUE TRUE FALSE FALSE TRUE TRUE
```



No exemplo anterior, a condição `num > 4` é uma **expressão condicional**, e o símbolo `>` um **operador lógico**. Os operadores lógicos utilizados no R são:

Operador	Significado
<code>&lt;</code>	menor
<code>&lt;=</code>	menor igual
<code>&gt;</code>	maior
<code>&gt;=</code>	maior igual
<code>==</code>	igual
<code>!=</code>	diferente
<code>&amp;</code>	e
<code> </code>	ou



# Fator

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Selecção  
condicional

Finalizando o  
programa

## Características:

- Coleção de categorias ou **níveis** (*levels*)
- Estrutura unidimensional

Utilizando as funções `factor()` e `c()`:

```
fator <- factor(c("alta","baixa","baixa","media",  
                  "alta","media","baixa","media","media"))
```

```
fator
```

```
[1] alta  baixa baixa media alta  media baixa media media  
Levels: alta baixa media
```

```
class(fator)
```

```
[1] "factor"
```



# Fator

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Caso haja uma hierarquia, os níveis dos fatores podem ser ordenados:

```
fator <- factor(c("alta", "baixa", "baixa", "media",  
                  "alta", "media", "baixa", "media", "media"),  
               levels = c("alta", "media", "baixa"))
```

```
fator
```

```
[1] alta  baixa baixa media alta  media baixa media media  
Levels: alta media baixa
```



# Matriz

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

## Características:

- Podem conter apenas um tipo de informação (números, caracteres)
- Estrutura bidimensional

Utilizando a função `matrix()`:

```
matriz <- matrix(1:12, nrow = 3, ncol = 4)
matriz
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	4	7	10
[2,]	2	5	8	11
[3,]	3	6	9	12

```
class(matriz)
```

```
[1] "matrix"
```



# Matriz

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Alterando a ordem de preenchimento da matriz (por linhas):

```
matriz <- matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)  
matriz
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	2	3	4
[2,]	5	6	7	8
[3,]	9	10	11	12

Para verificar a dimensão da matriz:

```
dim(matriz)
```

```
[1] 3 4
```





# Matriz

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

## Adicionando colunas com `cbind()`

```
cbind(matriz, rep(99, 3))
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	2	3	4	99
[2,]	5	6	7	8	99
[3,]	9	10	11	12	99

## Adicionando linhas com `rbind()`

```
rbind(matriz, rep(99, 4))
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	2	3	4
[2,]	5	6	7	8
[3,]	9	10	11	12
[4,]	99	99	99	99



# Matriz

## Operações matemáticas em matrizes

### Módulo I Básico

#### Introdução

#### Configuração inicial

#### Visão geral

#### Funções e argumentos

#### Objetos

#### Classes

#### Valores perdidos

#### Manipulação de dados

#### Indexação Seleção condicional

#### Finalizando o programa

## Matriz multiplicada por um escalar

```
matriz * 2
```

	[,1]	[,2]	[,3]	[,4]
[1,]	2	4	6	8
[2,]	10	12	14	16
[3,]	18	20	22	24

## Multiplicação de matrizes (observe as dimensões!)

```
matriz2 <- matrix(1, nrow = 4, ncol = 3)  
matriz %*% matriz2
```

	[,1]	[,2]	[,3]
[1,]	10	10	10
[2,]	26	26	26
[3,]	42	42	42



# Matriz

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Selecção  
condicional

Finalizando o  
programa

Associando nomes às linhas e colunas:

```
rownames(matriz) <- c("A","B","C")  
colnames(matriz) <- c("T1","T2","T3","T4")  
matriz
```

	T1	T2	T3	T4
A	1	2	3	4
B	5	6	7	8
C	9	10	11	12



# Lista

Módulo 1  
Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

## Características:

- Pode combinar uma coleção de objetos
- Estrutura “unidimensional”: apenas o número de elementos é contado

Utilizando a função `list()`:

```
lista <- list(a = 1:10, b = c("T1", "T2", "T3", "T4"))
```

```
lista
```

```
$a
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
$b
```

```
[1] "T1" "T2" "T3" "T4"
```

```
class(lista)
```

```
[1] "list"
```



# Lista

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Selecção  
condicional

Finalizando o  
programa

Formando uma lista com objetos criados anteriormente:

```
lista <- list(fator = fator, matriz = matriz)
lista
```

```
$fator
```

```
[1] alta  baixa baixa media alta  media baixa media media
Levels: alta media baixa
```

```
$matriz
```

	T1	T2	T3	T4
A	1	2	3	4
B	5	6	7	8
C	9	10	11	12

```
length(lista)
```

```
[1] 2
```



# Data frame

Módulo I  
Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Selecção  
condicional

Finalizando o  
programa

## Características:

- Uma lista de vetores e/ou fatores, de mesmo comprimento
- Pode conter diferentes tipos de dados (numérico, fator, ...)
- Estrutura bidimensional

Utilizando a função `data.frame()`:

```
dataFrame <- data.frame(ano = 2000:2004,  
                        captura = c(32, 54, 25, 48, 29))
```

```
dataFrame
```

	ano	captura
1	2000	32
2	2001	54
3	2002	25
4	2003	48
5	2004	29

```
class(dataFrame)
```

```
[1] "data.frame"
```



# Data frame

## Módulo 1 Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

Data frames podem ser formados com objetos criados anteriormente, desde que tenham o mesmo comprimento!

```
dataFrame <- data.frame(numerico = c(num, NA, NA, NA),  
                        fator = fator)
```

dataFrame

	numerico	fator
1	10	alta
2	5	baixa
3	2	baixa
4	4	media
5	8	alta
6	9	media
7	NA	baixa
8	NA	media
9	NA	media



# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa





# Valores perdidos e especiais

## Módulo 1 Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

Valores perdidos devem ser definidos como NA (*not available*):

```
perd <- c(3, 5, NA, 2)
```

```
perd
```

```
[1] 3 5 NA 2
```

```
class(perd)
```

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

Podemos testar a presença de NAs com a função `is.na()`:

```
is.na(perd)
```

```
[1] FALSE FALSE TRUE FALSE
```

Ou:

```
any(is.na(perd))
```

```
[1] TRUE
```



# Valores perdidos e especiais

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação Seleção condicional

### Finalizando o programa

Outros valores especiais são:

- NaN (*not a number*) - exemplo:  $0/0$
- $-\text{Inf}$  e  $\text{Inf}$  - exemplo:  $1/0$

A função `is.na()` também testa a presença de NaNs:

```
perd <- c(-1,0,1)/0  
perd
```

```
[1] -Inf  NaN  Inf
```

```
is.na(perd)
```

```
[1] FALSE  TRUE FALSE
```

A função `is.infinite()` testa se há valores infinitos

```
is.infinite(perd)
```

```
[1]  TRUE FALSE  TRUE
```



# Exercícios

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Crie um objeto com os valores 54, 0, 17, 94, 12.5, 2, 0.9, 15.
- 2 Some o objeto acima com os valores 5, 6.
- 3 Construa um objeto que indique que você coletou 15 machos (M), 12 fêmeas (F) e 8 juvenis (J) (repetindo as letras o número de vezes especificado).
- 4 Mostre na tela, em forma de verdadeiro ou falso, onde estão as fêmeas (F) nesse objeto.
- 5 Crie um objeto para armazenar a seguinte matriz

$$\begin{bmatrix} 2 & 8 & 4 \\ 0 & 4 & 1 \\ 9 & 7 & 5 \end{bmatrix}$$

- 6 Você coletou 42 plantas na Joaquina, 34 no Campeche, 59 na Armação, e 18 na Praia Mole. Crie um data frame para armazenar estas informações (número de plantas coletadas e local).



# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# Indexação de vetores

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

Crie um vetor para exemplo:

```
cont <- c(8, 4, NA, 9, 6, 1, 7, 9)  
cont
```

```
[1] 8 4 NA 9 6 1 7 9
```

Para acessar o valor que está na posição 4, faça:

```
cont[4]
```

```
[1] 9
```

Os colchetes [ ] são utilizados para extração (seleção de um intervalo de dados) ou substituição de elementos. O valor dentro dos colchetes é chamado de **índice**.



# Indexação de vetores

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

Para acessar os valores nas posições 1, 4 e 8 é necessário o uso da função `c()`:

```
cont[c(1, 4, 8)]
```

```
[1] 8 9 9
```

Ou:

```
ind <- c(1, 4, 8)
```

```
cont[ind]
```

```
[1] 8 9 9
```

Para selecionar todos os valores, *excluindo* aqueles das posições 1, 4 e 8:

```
cont[-c(1, 4, 8)]
```

```
[1] 4 NA 6 1 7
```



# Indexação de vetores

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação

### Seleção condicional

### Finalizando o programa

Também é possível selecionar uma sequência de elementos:

```
cont[1:5]
```

```
[1] 8 4 NA 9 6
```

Para selecionar todos os elementos, menos os NAs:

```
cont[!is.na(cont)]
```

```
[1] 8 4 9 6 1 7 9
```

Para substituir os NAs por algum valor (e.g. 0):

```
cont[is.na(cont)] <- 0  
cont
```

```
[1] 8 4 0 9 6 1 7 9
```





# Indexação de matrizes

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

Crie uma matriz para exemplo:

```
mat <- matrix(1:9, nrow=3)  
mat
```

	[,1]	[,2]	[,3]
[1,]	1	4	7
[2,]	2	5	8
[3,]	3	6	9

Acesse o valor que está na linha 2 da coluna 3:

```
mat[2,3]  
  
[1] 8
```



# Indexação de matrizes

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação

### Seleção condicional

### Finalizando o programa

Para acessar todas as linhas da coluna 1:

```
mat[,1]
```

```
[1] 1 2 3
```

Para acessar todas as colunas da linha 1:

```
mat[1,]
```

```
[1] 1 4 7
```

Para acessar as linhas 1 e 3 das colunas 2 e 3:

```
mat[c(1,3), c(2,3)]
```

```
      [,1] [,2]  
[1,]     4     7  
[2,]     6     9
```



# Indexação de listas

## Módulo 1 Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

Crie uma lista para exemplo:

```
lis <- list(vetor1 = c(3, 8, 7, 4), vetor2 = 5:0)  
lis
```

```
$vetor1  
[1] 3 8 7 4
```

```
$vetor2  
[1] 5 4 3 2 1 0
```

Para acessar o segundo componente da lista:

```
lis[[2]]  
  
[1] 5 4 3 2 1 0
```



# Indexação de listas

## Módulo 1 Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

Para acessar o terceiro valor do primeiro componente:

```
lis[[1]][3]
```

```
[1] 7
```

Os componentes das listas também podem ser acessados com \$:

```
lis$vetor2
```

```
[1] 5 4 3 2 1 0
```

O símbolo \$ é utilizado para acessar componentes **nomeados** de listas ou data frames.



# Indexação de data frames

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

Crie um data frame para exemplo:

```
dframe <- data.frame(col1 = 4:1, col2 = c(2,NA,5,8))  
dframe
```

	col1	col2
1	4	2
2	3	NA
3	2	5
4	1	8

Para acessar o segundo elemento da primeira coluna:

```
dframe[2,1]  
  
[1] 3
```



# Indexação de data frames

Módulo I  
Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos

Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

Acesse todas as linhas da coluna 2:

```
dframe[,2]
```

```
[1] 2 NA 5 8
```

Ou:

```
dframe[, "col2"]
```

```
[1] 2 NA 5 8
```

Todas as colunas da linha 1:

```
dframe[1,]
```

```
  col1 col2  
1    4    2
```

Ou:

```
dframe["1",]
```



# Indexação de data frames

## Módulo 1 Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

As colunas de um data frame podem ser acessadas com \$:

```
dframe$col1
```

```
[1] 4 3 2 1
```

Para acessar o terceiro elemento da coluna 2:

```
dframe$col2[3]
```

```
[1] 5
```

Para acessar os elementos nas posições 2 e 4 da coluna 2:

```
dframe$col2[c(2,4)]
```

```
[1] NA 8
```



# A função with()

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

Para evitar fazer muitas indexações de um mesmo data frame, por exemplo, podemos utilizar a função with()

```
with(dframe, col1)
```

```
[1] 4 3 2 1
```

é o mesmo que

```
dframe$col1
```

```
[1] 4 3 2 1
```





# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa



# Seleção condicional em vetores

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação

### Seleção condicional

### Finalizando o programa

A **seleção condicional** serve para extrair dados que satisfaçam algum critério, usando **expressões condicionais** e **operadores lógicos**.

Crie o seguinte vetor:

```
dados <- c(5, 15, 42, 28, 79, 4, 7, 14)
```

Selecione apenas os valores maiores do que 15:

```
dados[dados > 15]
```

```
[1] 42 28 79
```

Selecione os valores maiores que 15 **E** menores ou iguais a 35:

```
dados[dados > 15 & dados <= 35]
```

```
[1] 28
```



# Seleção condicional em vetores

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos

### Classes

### Valores perdidos

### Manipulação de dados

### Indexação

### Seleção condicional

### Finalizando o programa

Para entender como funciona a seleção condicional, observe apenas o resultado da condição dentro do colchetes:

```
dados > 15 & dados <= 35
```

```
[1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE
```

Os valores selecionados serão aqueles em que a condição for TRUE, nesse caso apenas o quarto elemento do vetor dados.



# Seleção condicional em data frames

Módulo I  
Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

Crie um data frame:

```
dados <- data.frame(ano = c(2001,2002,2003,2004,2005),  
                    captura = c(26,18,28,26,NA),  
                    porto = c("SP","RS","SC","SC","RN"))
```

Extraia deste objeto apenas a linha correspondente ao ano 2004:

```
dados[dados$ano == 2004,]
```

	ano	captura	porto
4	2004	26	SC

Mostre as linhas apenas do porto "SC":

```
dados[dados$porto == "SC",]
```

	ano	captura	porto
3	2003	28	SC
4	2004	26	SC



# Seleção condicional em data frames

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados

### Indexação

### Seleção condicional

### Finalizando o programa

Observe as linhas onde a captura seja maior que 20, selecionando apenas a coluna captura:

```
dados[dados$captura > 20, "captura"]
```

```
[1] 26 28 26 NA
```

Também exclua as linhas com NAs (agora com todas as colunas):

```
dados[dados$captura > 20 & !is.na(dados$captura),]
```

	ano	captura	porto
1	2001	26	SP
3	2003	28	SC
4	2004	26	SC



# Seleção condicional em data frames

## Módulo I Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos

### Classes

### Valores perdidos

### Manipulação de dados

### Indexação

### Seleção condicional

### Finalizando o programa

A condição pode ser feita com diferentes colunas:

```
dados[dados$captura > 25 & dados$porto == "SP",]
```

	ano	captura	porto
1	2001	26	SP

A função `subset()` serve para os mesmos propósitos:

```
subset(dados, captura > 25 & porto == "SP")
```

	ano	captura	porto
1	2001	26	SP



# Exercícios

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação

Seleção  
condicional

Finalizando o  
programa

- 1 Com o vetor criado no exercício (1) da sessão anterior, mostre quais são os valores nas posições 2, 5, e 7.
- 2 Com esse mesmo vetor, mostre todos os valores menos o zero.
- 3 Com o data frame criado no exercício (6) da sessão anterior, mostre qual a praia onde foram coletadas menos de 30 plantas (usando seleção condicional!).
- 4 Crie uma nova coluna (região) neste data frame indicando que Joaquina e Praia Mole estão localizadas no leste da ilha (leste), e Campeche e Armação estão no sul (sul).
- 5 Você está interessado em saber em qual das duas praias do sul, o número de plantas coletadas foi maior do que 40. Usando a seleção condicional, mostre essa informação na tela.



# Sumário

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

- 1 Introdução
- 2 Configuração inicial
- 3 Visão geral
- 4 Funções e argumentos
- 5 Objetos
  - Classes de objetos
- 6 Valores perdidos e especiais
- 7 Manipulação de dados
  - Indexação
  - Seleção condicional
- 8 Finalizando o programa





# Finalizando o programa

## Módulo I Básico

Introdução

Configuração  
inicial

Visão geral

Funções e  
argumentos

Objetos  
Classes

Valores  
perdidos

Manipulação  
de dados

Indexação  
Seleção  
condicional

Finalizando o  
programa

O passo mais importante é salvar seu **script**! No RStudio:

- File > Save As...
- Na janela que abrir, digite o nome do arquivo (por exemplo script\_aula) e salve
- Automaticamente o script será salvo com a extensão .R (nesse caso script\_aula.R) no diretório de trabalho que você configurou no início



# Finalizando o programa

## Módulo 1 Básico

### Introdução

### Configuração inicial

### Visão geral

### Funções e argumentos

### Objetos Classes

### Valores perdidos

### Manipulação de dados Indexação Seleção condicional

### Finalizando o programa

Alternativamente, você pode também salvar toda sua área de trabalho, clicando em **Workspace > Save As Default Workspace**. Este processo irá gerar dois arquivos:

- **.Rdata**: contém todos os objetos criados durante uma sessão. Não é necessário (e nem recomendado) dar um nome antes do ponto. Dessa forma, a próxima vez que o programa for iniciado neste diretório, a área de trabalho será carregada automaticamente.
- **.Rhistory**: um arquivo texto que contém todos os comandos que foram digitados no console.

A qualquer momento durante uma sessão você pode usar o comando

```
save.image()
```

para salvar a área de trabalho. **Note que o mais importante é salvar o *script* que contém todos os comandos para gerar novamente os objetos.**