Começando a programar

Maxwel Coura Oliveira

7/8/2020

Vamos comerçar a programar!

- Operadores: aritmética, atribuição, extração, lógica
- Funções: nomes, argumentos, saída
- Tipos de dados: classes, vetores, quadros de dados (*data frame*)

Operadores

Operadores	O que faz	Simbolo
Aritmético	Matemática em números	+ - * / ^
Atribuição	Cria objetos (esquerda) com	<-
Extração	Retire ou substitua parte de um objeto	[ ] $
Lógica	Compara valores, retorna TRUE/FALSE	>< == ! %in% & |

Operadores

```
2 + 2 #adição
```

```
## [1] 4
```

```
8 - 5 #subtração
```

```
## [1] 3
```

```
4 * 4 #multiplicação
```

```
## [1] 16
```

```
6 / 2 #divisão
```

```
## [1] 3
```

Atribuição

- Salva valor em objeto
  - objeto <- valor
  - peso_kg <- 55
- Combina com operação aritimetica:
  - peso_lb <- 2.2 * peso_kg

Funções e argumentos

Uma sequência de instruções que executam uma tarefa. Uma função pode ter muitos comandos, mas no R existem as chamadas 'built-in', que já estão imbutidas no programa.

- Ter nomes
- Aceita argumentos (Input - entrada)
- Retorna um valor (Output - saída)

Input - entrada	Output - saída
sqrt(9)	3
round(3.14159)	3
round(x=3.14159), digits=2	3

Outros exemplos: `mean()`, `min()`, `max()`

Ajuda

Caso tenham dúvidas

```
?round #abre uma página com a descrição para round na interface "Files'
```

```
args(round) #mostra os argumentos
```

```
## function (x, digits = 0)
## NULL
```

- **Google "R +"nome da função"**
- Outros websites/blogs
  - Stack overfolow (Q&A)
  - R bloggers (tutorials)
  - GitHub

Tipos de dados

- RR adivinha que tipo de dados são armazenados em um objeto
- Tipos básicos:
  - Numérico (*numeric*)
  - nominal (*character*)
  - Lógico (*logical*)
- Pode ser fácil diferenciá-los

Determinando o tipo de dado

- Usamos a função `class()` ou `typeof()`

Examplos:

```
x <- 32
class(x)
```

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

```
x <- "car"
class(x)
```

```
## [1] "character"
```

```
x <- TRUE
class(x)
```

```
## [1] "logical"
```

Estrutura de dados

Vetores

- Tipo mais comum de estrutura de dados
- Uma série de tipos de dados (ex. numérico)
- Função concatenar: c()
  Input: valores separados por vírgulas
  Output: um objeto vetor

```
# Exemplo: uma lista com produtividade da soja
prod_soja <- c(3000, 2890, 3100, 2990)
# Exemplo: uma lista com nomes de animais
cars <- c("cavalo", "touro", "cachorro", "gato")
```

Inspencionando vetores

- Vetores tem caracteristicas:
  - **Comprimento**: número de valores
  - **Classe**: tipo de valores

```
length(prod_soja)
```

```
## [1] 4
```

```
class(prod_soja)
```

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

```
str(prod_soja)
```

```
##  num [1:4] 3000 2890 3100 2990
```

Adicionando valores a um vetor

- Use um vetor existente como argumento para c()
- Coloque na ordem em que deseja que eles apareçam na saída vecto

```
# Adicione um valor no final do vetor
prod_soja <- c(prod_soja, 3315)
```

```
# Adicione um valor no inicio do vetor
prod_soja <- c(3050, prod_soja)
```

Usando funções `built-in` em vetores:

Qual a média de produtividade de soja?

```
prod_soja <- c(3050, 3050, 3000, 2890, 3100, 2990, 3315) #produtividade de soja

(3050 + 3050 + 3000 + 2890 + 3100 + 2990 + 3315)/7 #calculando a média (muito trabalhoso desse jeito)
```

```
## [1] 3056.429
```

```
mean(prod_soja) #Usando funcões embutidas no programa
```

```
## [1] 3056.429
```

Qual o valor minimo de produtividade de soja?

```r
min(prod_soja)
```

```
## [1] 2890
```

Qual o valor máximo de produtividade de soja?

```r
max(prod_soja)
```

```
## [1] 3315
```

Fatores

- Representam dados categóricos
  - Armazenado como números inteiros com rótulos de texto
  - *Data frames* convertem colunas de caracteres em fatores
- *factor()* - cria um fator
- Crie um vetor de nomes de carros

```r
carros <- c("monza", "chevete", "monza", "chevete")
class(carros)
```

```
## [1] "character"
```

- Change vector to a factor

```r
carros <- factor(carros)
class(carros)
```

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

Níveis (levels)

- Texto único de um objeto de fator
- *levels()* - mostra os níveis
- *nlevels()* - mostra o número de níveis

Função	**Output**
levels(carros)	“monza”, “chevete”
nlevels(carros)	2

Subconjunto de vetores (*subset*)

- *Subset* por posição
- *Syntax*: colchetes `[ ]`
- Combine com `c()`

```r
culturas <- c("milho", "soja", "sorgo", "feijão", "café")
```

```r
culturas[2] #Mostra o segundo valor
```

```
## [1] "soja"
```

```r
culturas[c(3,2)]  #Mostra o terceiro e segundo valor
```

```
## [1] "sorgo" "soja"
```

```r
culturas[c(1,2,3,2,1)] #Mostra valores repetidos
```

```
## [1] "milho" "soja"  "sorgo" "soja"  "milho"
```

```r
culturas[c(1:3)] #Mostra sequencia de valores
```

```
## [1] "milho" "soja"  "sorgo"
```

Expressões lógicas

- Faz comparações
- Avalia cada elemento em um vetor em relação a um valor
- Output: TRUE ou FALSE
  - Para cada valor do vetor

Logical expressions

Operador lógico	Significado
>	Greater than
<	Less than
==	Igual a
!=	Não igual a
&	e
|	ou
!	não
%in%	Contido em

Examplo: expressões lógicas

- Crie uma variável peso

```r
biomassa_g <- c(22.4, 33.7, 37.1, 51.3, 59.9, 45.2)
```

- Avalia cada elemento do vetor:

```
biomassa_h <- biomassa_g > 45
biomassa_h
```

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

Coerção de classe

- O que acontece se você misturar tipos?
- O R converte para um tipo que funciona para todos os elementos
- Use `class()` para ver o que R escolheu

Tipo	as character	as numeric	as logical
logical	"TRUE"	1	TRUE
numeric	"35"	35	NA
character	"Paulista"	NA	NA

Subconjunto (*subset*) de vetores

- Mantém o TRUE, elimina o FALSE
- **Input**: uma expressão lógica
- **Output**: um vetor com elementos que correspondem à expressão lógica
- Subconjunto usando vetor TRUE/FALSE

```
biomassa_g[biomassa_h]
```

```
## [1] 51.3 59.9 45.2
```

- O mesmo que:

```
biomassa_g[biomassa_g>50]
```

```
## [1] 51.3 59.9
```

Subconjunto condicional

- Combine vários condicionais
- | = ou (um ou outro)
- & = e (os dois)
- Biomassa abaixo de 30 ou acima de 50:

```r
biomassa_g[biomassa_g<30 | biomassa_g>50]
```

```
## [1] 22.4 51.3 59.9
```

- Biomassa acima de 30 e abaixo de 50:

```r
biomassa_g[biomassa_g>30 & biomassa_g<50]
```

```
## [1] 33.7 37.1 45.2
```

Subconjunto condicional: caracteres (==)

- Operador ==
- Compara cada valor em um vetor com uma sequência de caracteres
- Combinar com | para múltiplos

Make a character vector

```r
plantas <- c("amargoso", "buva", "leiteira", "guanxuma")
```

Plantas que são buva

```r
plantas[plantas=="buva"]
```

```
## [1] "buva"
```

Plantas que são buva ou leiteira

```r
plantas[plantas=="buva" | plantas=="leiteira"]
```

```
## [1] "buva"     "leiteira"
```

Subconjunto condicional: caracteres (%in%)

- Operador **%in%**
- Seleciona elementos do primeiro vetor que estão no segundo vetor
- **Input**: vetor
- **Output**: uma lista de TRUE/FALSE

Quais plantas estão no vetor da mão direita?

```r
plantas %in% c("amargoso", "soja", "trigo", "buva", "tiririca")
```

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

Retornando os nomes das plantas"

```r
plantas[plantas %in% c("amargoso", "soja", "trigo", "buva", "tiririca")]
```

```
## [1] "amargoso" "buva"
```

Dados ausentes

- NA (not available) - harder to overlook missing data
- Argumento: na.rm = TRUE

```r
na.rm = TRUE #Ignores missing data
```

```r
altura <- c(2.5, 4.4, 4.1, NA, 6.5, 7.5) #um vetor
```

- Mean of a missing value?

```r
mean(altura)
```

```
## [1] NA
```

- Remove the missing data

```r
mean(altura, na.rm = TRUE)
```

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

Removendo dados ausentes

- is.na() - Retorna TRUE se o valor é NA
- complete.cases() - Retorna FALSE se o valor é NA
- na.omit() - Retorna objeto com os valores ausentes removidos

Remove NAs 3 ways:

```r
altura[!is.na(altura)]
```

```
## [1] 2.5 4.4 4.1 6.5 7.5
```

```r
altura[complete.cases(altura)]
```

```
## [1] 2.5 4.4 4.1 6.5 7.5
```

```r
na.omit(altura)
```

```
## [1] 2.5 4.4 4.1 6.5 7.5
## attr(,"na.action")
## [1] 4
## attr(,"class")
## [1] "omit"
```