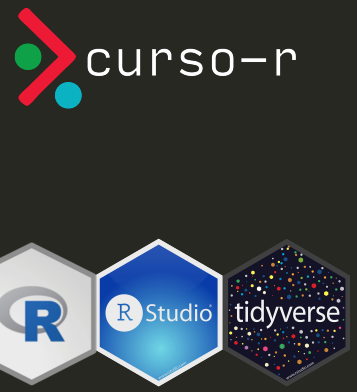
Introdução à programação com R

Introdução ao R



Introdução ao R

R como calculadora

O papel do **Console** no R é executar os nossos comandos. Ele avalia o código que passamos para ele e devolve a saída correspondente (se tudo der certo) ou uma mensagem de erro (se o seu código tiver algum problema).

> Para rodar um código, escreva o código no script e, com o cursor em cima da linha que você quer rodar, use o atalho "CTRL+ENTER". Você não precisa selecionar o código, a não ser que queria rodar várias linhas de uma única vez.

Vamos começar com um exemplo de operação comum:

```
1 + 1
```

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

Nesse caso, o nosso comando foi o código `1 + 1` e a saída foi o valor `2`.

Tente agora jogar no console a expressão: `2 * 2 - (4 + 4) / 2`.

Repare que as operações e suas precedências são mantidas como na matemática, ou seja, divisão e multiplicação são calculadas antes da adição e subtração. E parênteses nunca são demais!

Pronto! Você já é capaz de pedir ao R para fazer qualquer uma das quatro operações aritméticas básicas.

> **Quando compilamos?** Quem vem de linguagens como o C ou Java espera que seja necessário compilar o código em texto para o código das máquinas (geralmente um código binário). No R, isso não é necessário. O R é uma linguagem de programação dinâmica que interpreta o seu código enquanto você o executa.

Comandos incompletos

Se você digitar um comando incompleto, como `5 +`, e apertar `Enter`, o R mostrará um `+`, o que não tem nada a ver com a adição da matemática. Isso significa que o R está esperando que você enviar **mais** algum código para completar o seu comando. Termine o seu comando ou aperte `Esc` para recomeçar.

```
> 5 -
+
+ 5
[1] 0
```

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.

```
> 5 % 2
Error: unexpected input in "5 % 2"
> 5 ^ 2
[1] 25
```

Objetos

O R te permite salvar valores dentro de um **objeto**. Um objeto é um nome que guarda um valor. Para criar um objeto, utilizamos o operador `<-`.

No exemplo abaixo, salvamos o valor `1` em `a`. Sempre que avaliarmos o objeto `a`, o R vai devolver o valor 1.

```
# Salvando `1` em `a`
a <- 1

# Avaliando o objeto `a`
a
```

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

Nomeando objetos

Existem algumas regras para dar nomes aos objetos. A mais importante é: o nome deve começar com uma letra. O nome pode conter números, mas não pode começar com números. Você pode usar pontos `.` e underlines `_` para separar palavras.

```
# Permitido

x <- 1
x1 <- 2
objeto <- 3
meu_objeto <- 4
meu.objeto <- 5

# Não permitido

1x <- 1
_objeto <- 2
meu-objeto <- 3
```

O R **diferencia letras maiúsculas e minúsculas**, isto é, `b` é considerado um objeto diferente de `B`. Rode o exemplo abaixo e observe que dois objetos diferentes são criados no **Environment**.

```
b <- 2
B <- 3

b
```

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

```
B
```

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

Classes

A classe de um objeto é muito importante dentro do R. É a partir dela que as funções e operadores conseguem saber exatamente o que fazer com um objeto.

Por exemplo, podemos somar dois números, mas não conseguimos somar duas letras (texto):

```
1 + 1
```

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

```
"a" + "b"
```

```
## Error in "a" + "b": argumento não-numérico para operador binário
```

O operador `+` verifica que `"a"` e `"b"` não são números (ou que a classe deles não é numérica) e devolve uma mensagem de erro informando isso.

Texto

Observe que para criar texto no R, colocamos os caracteres entre aspas. As aspas servem para diferenciar *nomes* (objetos, funções, pacotes) de *textos* (letras e palavras). Os textos são muito comuns em variáveis categóricas.

```
a <- 10

# O objeto `a`, sem aspas
a
```

```
## [1] 10
```

```
# A letra (texto) `a`, com aspas
"a"
```

```
## [1] "a"
```

A classe de um objeto

Para saber a classe de um objeto, basta rodarmos `class(nome-do-objeto)`.

```
x <- 1
class(x)
```

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

```
y <- "a"
class(y)
```

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

```
class(mtcars)
```

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

Objetos atômicos

As classes mais básicas dentro do R são:

- *numeric*
- *character*
- *logical*

Um objeto de qualquer uma dessas classes é chamado de **objeto atômico**.

Esse nome se deve ao fato de essas classes não se misturarem, isto é, para um objeto ter a classe `numeric`, por exemplo, todos os seus valores precisam ser numéricos.

Mas como atribuir mais de um valor a um mesmo objeto? Para isso, precisamos criar **vetores**.

Vetores

Vetores são estruturas muito importantes dentro R. Em especial, pensando em análise de dados, precisamos estudá-los pois cada coluna de um *data frame* será representada como um vetor.

Vetores são apenas **conjuntos indexados de valores**. Para criá-los, basta colocar os valores separados por vírgulas dentro de um `c()`.

```
vetor1 <- c(1, 5, 3, -10)
vetor2 <- c("a", "b", "c")

vetor1
```

```
## [1]   1   5   3 -10
```

```
vetor2
```

```
## [1] "a" "b" "c"
```

Sequências

Uma maneira fácil de criar um vetor com uma sequência de números é utilizar o operador `:`.

```
# Vetor de 1 a 10
1:10
```

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

```
# Vetor de 10 a 1
10:1
```

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

```
# Vetor de -3 a 3
-3:3
```

```
## [1] -3 -2 -1  0  1  2  3
```

Subsetting

Quando dizemos que vetores são conjuntos *indexados*, isso quer dizer que cada valor dentro de um vetor tem uma **posição**. Essa posição é dada pela ordem em que os elementos foram colocados no momento em que o vetor foi criado. Isso nos permite acessar individualmente cada valor de um vetor.

Para isso, colocamos o índice do valor que queremos acessar dentro de colchetes `[]`.

```
vetor <- c("a", "b", "c", "d")

vetor[1]
```

```
## [1] "a"
```

```
vetor[4]
```

```
## [1] "d"
```

Você também pode colocar um conjunto de índices dentro dos colchetes, para pegar os valores contidos nessas posições:

```
vetor[c(2, 3)]
```

```
## [1] "b" "c"
```

```
vetor[c(1, 2, 4)]
```

```
## [1] "a" "b" "d"
```

Essas operações são conhecidas como *subsetting*, pois estamos pegando subconjuntos de valores de um vetor.

Classe de um vetor

Um vetor só pode guardar um tipo de objeto e ele terá sempre a mesma classe dos objetos que guarda. Para saber a classe de um vetor, rodamos `class(nome-do-vetor)`.

```
vetor1 <- c(1, 5, 3, -10)
vetor2 <- c("a", "b", "c")

class(vetor1)
```

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

```
class(vetor2)
```

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

Coerção

Se tentarmos misturar duas classes, o R vai apresentar o comportamento conhecido como **coerção**.

```
vetor <- c(1, 2, "a")

vetor
```

```
## [1] "1" "2" "a"
```

```
class(vetor)
```

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

Veja que todos os elementos do vetor se transformaram em texto.

Como um vetor só pode ter uma classe de objeto dentro dele, classes mais fracas serão sempre reprimidas pelas classes mais fortes. Como regra de bolso: caracteres serão sempre a classe mais forte. Então, sempre que você misturar números e texto em um vetor, os números virarão texto.

Operações com vetores

De forma bastante intuitiva, você pode fazer operações com vetores.

```
vetor <- c(0, 5, 20, -3)
vetor + 1
```

```
## [1]  1  6 21 -2
```

Ao rodarmos `vetor1 + 1`, o R subtrai `1` de cada um dos elementos do vetor. O mesmo acontece com qualquer outra operação aritmética.

Vetorização

Você também pode fazer operações que envolvem mais de um vetor:

```
vetor1 <- c(1, 2, 3)
vetor2 <- c(10, 20, 30)

vetor1  + vetor2
```

```
## [1] 11 22 33
```

Neste caso, o R irá alinhar os dois vetores e somar elemento a elemento. Esse tipo de comportamento é chamado de **vetorização**.

Recliclagem

Isso pode ficar um pouco confuso quando os dois vetores não possuem o mesmo tamanho.

```
vetor1 <- c(1, 2)
vetor2 <- c(10, 20, 30, 40)

vetor1 + vetor2
```

```
## [1] 11 22 31 42
```

Embora estejamos somando dois vetores de tamanho diferentes, o R não devolve um erro (o que parecia ser a resposta mais intuitiva). O R alinhou os dois vetores e, como eles não possuíam o mesmo tamanho, o primeiro foi repetido para ficar do mesmo tamanho do segundo. É como se o primeiro vetor fosse na verdade `c(1, 2, 1, 2)`.

Esse comportamento é chamado de **reciclagem**.

Embora contra-intuitiva, a reciclagem é muito útil no R graças a um caso particular muito importante.

Quando somamos `vetor + 1` no nosso primeiro exemplo, o que o R está fazendo por trás é transformando o `1` em `c(1, 1, 1, 1)` e realizando a soma vetorizada `c(0, 5, 20, -3) + c(1, 1, 1, 1)`. Isso porque o número `1` é um vetor de tamanho 1, isto é, `1` é igual a `c(1)`.

Usaremos esse comportamento no R o tempo todo. É muito importante a reciclagem para termos certeza de que o R está fazendo exatamente aquilo que gostaríamos que ele fizesse.

Um outro caso interessante de reciclagem é quando o comprimento dos vetores não são múltiplos um do outro.

```
vetor1 <- c(1, 2, 3)
vetor2 <- c(10, 20, 30, 40, 50)

vetor1 + vetor2
```

```
## Warning in vetor1 + vetor2: comprimento do objeto maior não é múltiplo do
## comprimento do objeto menor
```

```
## [1] 11 22 33 41 52
```

Neste caso, duas coisas aconteceram:

1. O R realizou a conta, repetindo cada valor do primeiro vetor até que os dois tenham o mesmo tamanho. No fundo, a operação realizada foi `c(1, 2, 3, 1, 2) + c(10, 20, 30, 40, 50)`.

2. Como essa operação é ainda menos intuitiva e raramente desejada, o R devolveu um aviso dizendo que o comprimento do primeiro vetor maior não é um múltiplo do comprimento do vetor menor.

Pertence

Um outro operador muito útil é o `%in%`. Com ele, podemos verificar se um valor está dentro de um conjunto de valores (vetor).

```
3 %in% c(1, 2, 3)
```

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

```
"a" %in% c("b", "c")
```

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