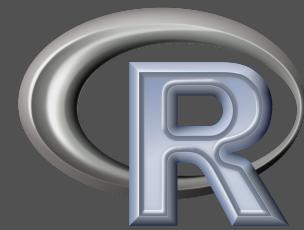




# ME115 - Linguagem R

Parte 03

1º semestre de 2023



# Funções

# Funções no R

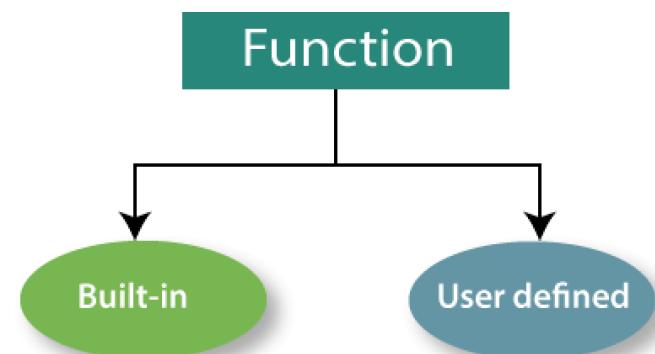
Esse é um dos conceitos fundamentais para quem programa em R: **funções**.

Praticamente tudo no R é escrito em termos de funções.

Funções são blocos de códigos organizados e que podem ser reutilizados para executar uma ação/tarefa, de acordo com seus *inputs* (argumentos).

Funções no R são “objetos de primeira classe”, e podem ser tratadas como qualquer outro objeto no R.

Existem dois tipos principais de funções:



Fonte da imagem: <https://www.javatpoint.com/r-functions>

# Funções no R

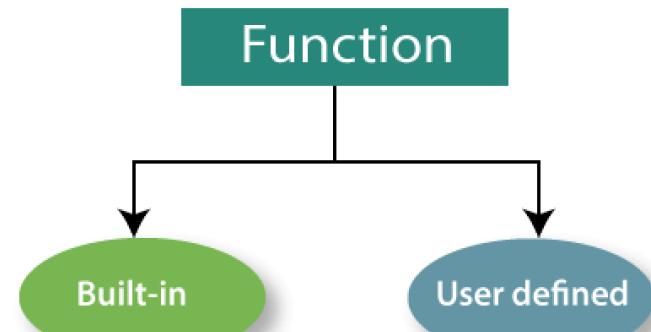
1. **Funções pré-definidas:** funções já existentes, armazenadas internamente no R ou em pacotes, que podem ser usadas pelo usuário para facilitar seu trabalho.

**Exemplos:** `mean()`, `sum()`, `sqrt()`, `sample()`.

```
class(mean)  
#> [1] "function"
```

2. **Funções definidas pelo usuário:** funções criadas pelo usuário para executar determinada tarefa, atendendo as exigências de cada usuário.

Nessa aula iremos estudar tanto funções pré-definidas quanto definir nossas próprias funções.



# Funções Pré-Definidas

Veja um exemplo:

```
sample(0:1, size = 10, replace = TRUE)
```

A função `sample()` retira uma amostra a partir dos elementos de um vetor `x`, com ou sem reposição. Veja: `?sample` (figura ao lado).

O que está dentro dos parênteses são os argumentos da função.

`x` e `size`: argumentos obrigatórios que deve ser especificados pelo usuário para que a função seja executada.

`replace` e `prob`: argumentos opcionais. Eles já apresentam um valor *default*, caso o usuário não os especifique.

sample {base} R Documentation

## Random Samples and Permutations

### Description

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.

### Usage

```
sample(x, size, replace = FALSE, prob = NULL)
sample.int(n, size = n, replace = FALSE, prob = NULL,
          useHash = (!replace && is.null(prob)) && size <= n)
```

### Arguments

<code>x</code>	either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'
<code>n</code>	a positive number, the number of items to choose from. See 'Details.'
<code>size</code>	a non-negative integer giving the number of items to choose.
<code>replace</code>	should sampling be with replacement?
<code>prob</code>	a vector of probability weights for obtaining the elements of the vector being sampled.

# Funções Úteis



## Maths Functions

<code>log(x)</code>	Natural log.	<code>sum(x)</code>	Sum.
<code>exp(x)</code>	Exponential.	<code>mean(x)</code>	Mean.
<code>max(x)</code>	Largest element.	<code>median(x)</code>	Median.
<code>min(x)</code>	Smallest element.	<code>quantile(x)</code>	Percentage quantiles.
<code>round(x, n)</code>	Round to n decimal places.	<code>rank(x)</code>	Rank of elements.
<code>signif(x, n)</code>	Round to n significant figures.	<code>var(x)</code>	The variance.
<code>cor(x, y)</code>	Correlation.	<code>sd(x)</code>	The standard deviation.

Fonte: [Base R Cheat Sheet](#)

# Funções Pré-Definidas

Simule o lançamento de uma moeda 10 vezes, sendo (1) “cara” e (0) “coroa”.

```
coin <- sample(0:1, size=10, replace=TRUE)  
coin
```

```
## [1] 1 0 1 1 1 1 0 0 1
```

Experimente essa função trocando os argumentos.

Agora conte o número de caras que você observou:

```
sum(coin)
```

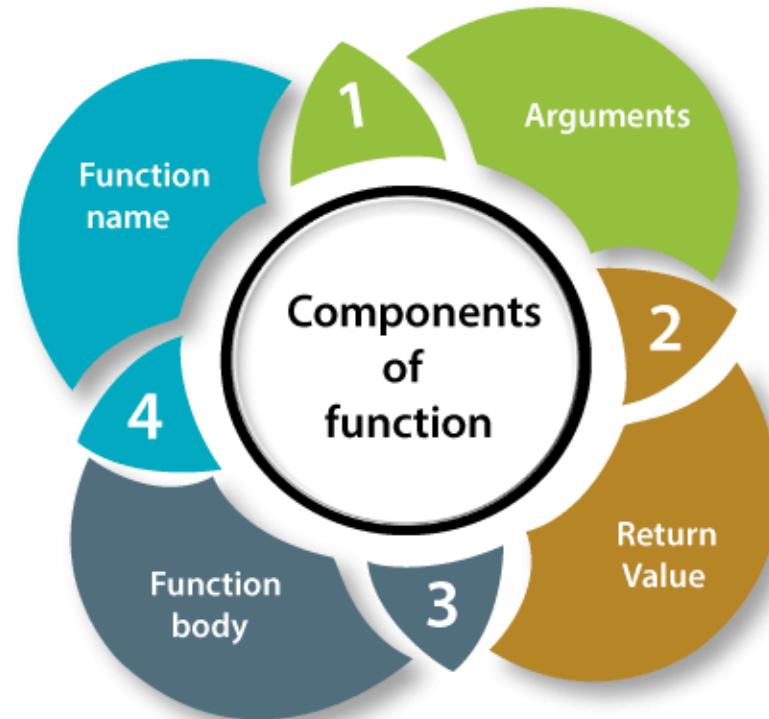
```
## [1] 7
```

Calcule a proporção de caras nos 10 lançamentos:

```
mean(coin)
```

```
## [1] 0.7
```

# Componentes de uma Função



Fonte da imagem: <https://www.javatpoint.com/r-functions>

# Estrutura de uma Função

```
nome_da_funcao <- function(argumentos) {  
  # lista de comandos a serem executados com os argumentos de entrada  
  return(resultado)  
}  
  
# uso da função  
nome_da_funcao(arg1, arg2, ...)
```

Argumentos são objetos cujos valores devem ser atribuídos pelo usuário. Já vimos que existem argumentos **obrigatórios** e **opcionais**.

**Exemplo:** a função `mean()`, `x` é um argumento obrigatório e se ele não for especificado, a função retorna um erro. Já os argumentos `trim` e `na.rm` são opcionais.

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

# Definir argumentos pela ordem ou pelo nome

Os argumentos de uma função podem ser definidos pela ordem em que foram inseridos na construção da função ou pelo nome. Veja um exemplo:

```
mean(x, trim = 0, na.rm = FALSE, ...)

v <- c(10, 5, 8, NA, 7)
mean(v)  ## a função assume que x = v

mean(v, TRUE)  ## o 2o argumento é `trim`, que não aceita valores lógicos

mean(v, na.rm = TRUE)  ## o 1o argumento vai para x e define-se explicitamente na.rm

mean(na.rm = TRUE, x = v)
```

Veja que no último comando, trocamos a ordem de entrada dos argumentos. Isso é possível desde que sejam chamados pelos nomes explicitamente.

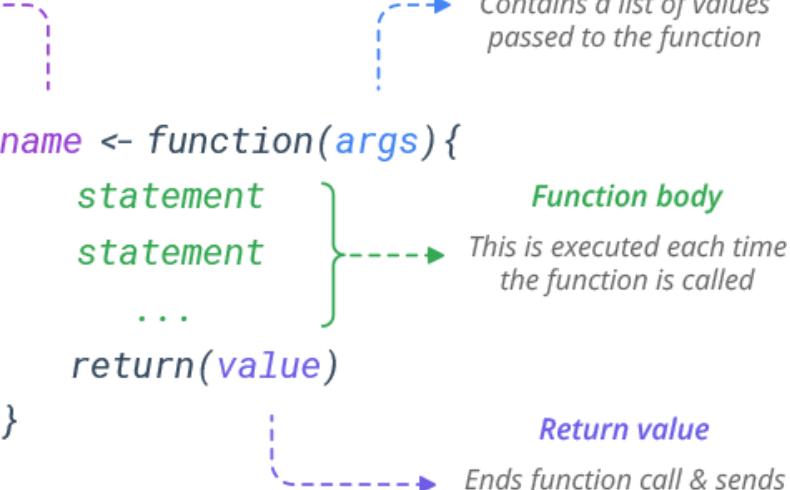
# Criando uma Função

```
Function name  
An identifier by which the function is called  
  
name <- function(args){  
    statement  
    statement  
    ...  
    return(value)  
}
```

Arguments  
Contains a list of values passed to the function

Function body  
This is executed each time the function is called

Return value  
Ends function call & sends data back to the program



**return:** especifica o que a função deve devolver ao usuário quando executá-la.

Seu uso não é obrigatório!  
Se o **return** não estiver presente no corpo da função, o último valor calculado será retornado.

**Nome da função:** ao definir sua própria função, escolha bem o nome. O nome da função deve seguir as mesmas regras de nomes de variáveis. A função será executada/chamada sempre pelo nome que atribuímos.

# Criando uma Função

Vamos escrever uma função que converte a temperatura de graus Fahrenheit para Celsius.

```
grausFtoC <- function(tempF){  
  # Essa função converte a temperatura em graus Fahrenheit para Celsius  
  
  tempC <- (tempF - 32) * 5/9  
  return(round(tempC, 1))  
}
```

Ao executar/rodar o código acima no R, a função `grausFtoC()` estará gravada no ambiente global e disponível para uso (verifique no seu RStudio).

Objetos definidos internamente na função, não são salvos no ambiente de trabalho. No caso, `tempC`, não aparecerá fora da função.

# Executando sua função

Agora que sua função está disponível para uso, você pode executá-la. Por exemplo, 100°F equivalem a quantos graus Celsius?

```
grausFtoC(100)
```

```
## [1] 37.8
```

A função que definimos também aceita vetores numéricos como argumento:

```
tempo <- c(Manhattan = 68, Washington = 72, Miami = 84)
grausFtoC(tempo)
```

```
## Manhattan Washington      Miami
##        20.0          22.2       28.9
```

Observe que a função retorna um vetor numérico com os mesmos nomes que o vetor de entrada, mas esse vetor não está armazenado no ambiente de trabalho.

# Executando sua função

Você pode armazenar a saída da função em uma variável:

The screenshot shows the RStudio interface. On the left, the script editor window titled "grausFtoC.R" contains the following R code:

```
1 grausFtoC <- function(tempF){  
2   # Essa função converte a temperatura  
3   # de graus Fahrenheit para Celsius  
4  
5   tempC <- (tempF - 32) * 5/9  
6   return(round(tempC, 1))  
7 }  
8
```

The "Console" tab in the bottom-left shows the execution of the script and some additional commands:

```
> temps <- c(Manhattan = 68, Washington = 72, Miami = 84)  
> grausFtoC(temps)  
Manhattan Washington      Miami  
       20.0          22.2        28.9  
> tempsC <- grausFtoC(temps) ←  
> tempsC  
Manhattan Washington      Miami  
       20.0          22.2        28.9  
>
```

The "Global Environment" pane on the right displays the current objects in memory:

Values	
temp	Named num [1:3] 68 72 84
tempC	Named num [1:3] 20 22.2 28.9

The "Functions" section shows the definition of the "grausFtoC" function.

# Funções que retornam vários valores

Uma função retorna um objeto (vetor, lista, matrizes, etc). Então, ela pode retornar vários valores se eles forem combinados em um vetor (ou uma lista) ou qualquer outro objeto que seja apropriado.

```
math <- function(x, y){  
  add <- x + y  
  sub <- x - y  
  mult <- x * y  
  div <- x / y  
  c(adicao = add, subtracao = sub, multiplicacao = mult, divisao = div)  
}  
  
math(24, 2)
```

##	adicao	subtracao	multiplicacao	divisao
##	26	22	48	12

# Funções com Argumentos Opcionais

Suponha que a função `mean()` não exista. Uma versão simplificada de uma função que calcula a média pode ser escrita como:

```
media <- function(x){  
  total <- sum(x)  
  n <- length(x)  
  total/n  # não estamos usando o return!  
}
```

Aplicando a função `media()` num vetor de 1 a 100:

```
x <- 1:100  
media(x)  
  
## [1] 50.5  
  
identical(mean(x), media(x))  # comparando a saída com a função mean()  
  
## [1] TRUE
```

# Funções com Argumentos Opcionais

```
medial <- function(x){  
  n <- count <- length(x)  
  soma <- 0  
  while (count != 0) {  
    soma <- soma + x[count]  
    count <- count - 1  
  }  
  soma/n  
}  
medial(x)
```

```
## [1] 50.5
```

```
media2 <- function(x){  
  n <- length(x)  
  soma <- 0  
  for(i in 1:n)  soma <- soma + x[i]  
  soma/n  
}  
media2(x)
```

```
## [1] 50.5
```

# Funções com Argumentos Opcionais

As funções podem ter vários argumentos, obrigatórios ou opcionais. Vamos então adaptar a função anterior para calcular a média aritmética ou geométrica, dependendo da escolha do usuário:

```
media <- function(x, aritmetica = TRUE){  
  n <- length(x)  
  m <- ifelse(aritmetica, sum(x)/n, prod(x)^(1/n))  
  return(m)  
}
```

Aplicando a função para calcular as médias aritmética ou geométrica:

```
media(x = 1:100)
```

```
## [1] 50.5
```

```
media(x = 1:100, aritmetica = FALSE)
```

```
## [1] 37.99269
```

# Funções com o argumento (...)

Existe um argumento especial no R que é o três pontinhos (...), que indica um número variável de argumentos que são usualmente passados para outras funções que aparecem dentro de uma função.

Veja um exemplo:

```
rep(x, ...)
```

A função `rep()` tem como argumento obrigatório `x`, que é um vetor, e os ... podem ser um ou mais dentre esses três argumentos: `times`, `length.out` e `each`.

```
rep(1:5, times = 2)
rep(1:5, times = c(5:1))
rep(1:5, each = 2)
rep(1:5, length.out = 12)
```

# Funções com o argumento (...)

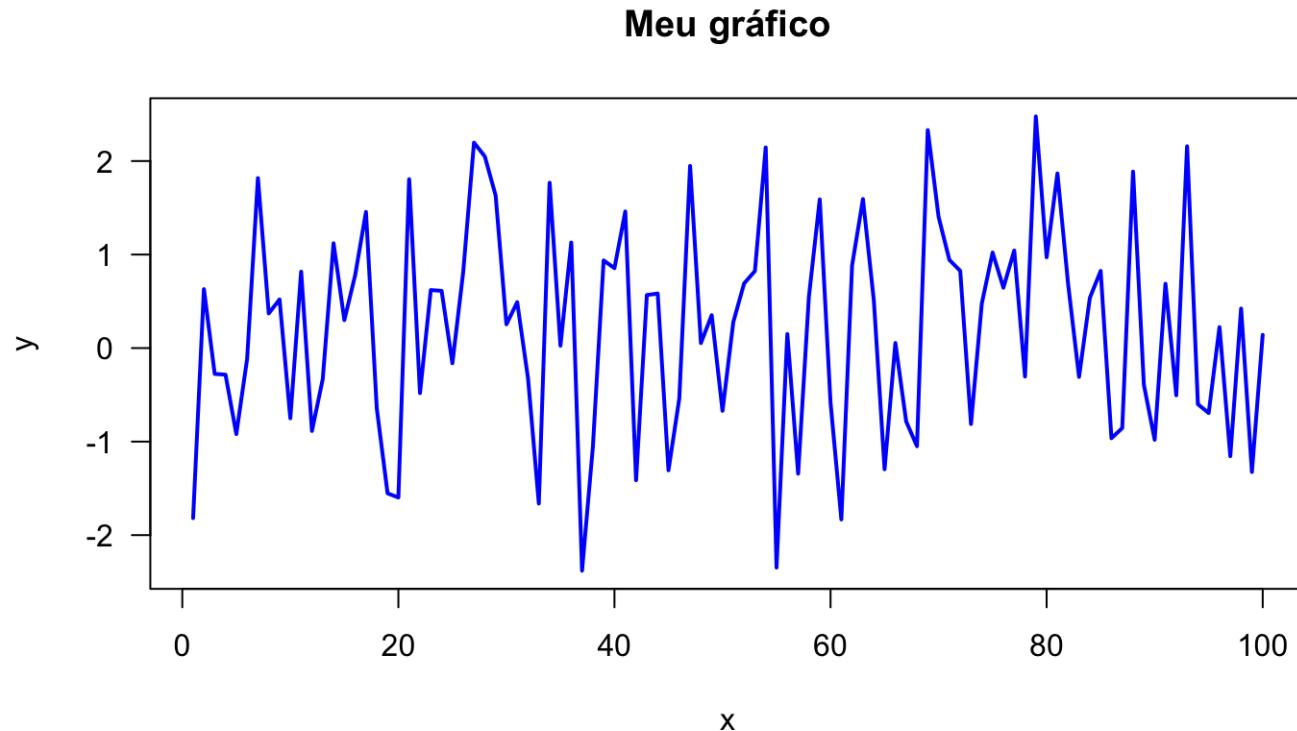
Você quer criar a sua própria função para fazer um gráfico de linhas, mas você irá usar a função pré-definida `plot()` dentro da sua função, com toda a sua lista de argumentos:

```
myplot <- function(x, y, type = "l", ...){  
  plot(x, y, type = type, ...)  ## `...` será passado para a função plot()  
}
```

Você pode querer mudar coisas como a cor (`color`), largura da linha (`lwd`), orientação do eixo (`las`), etc.

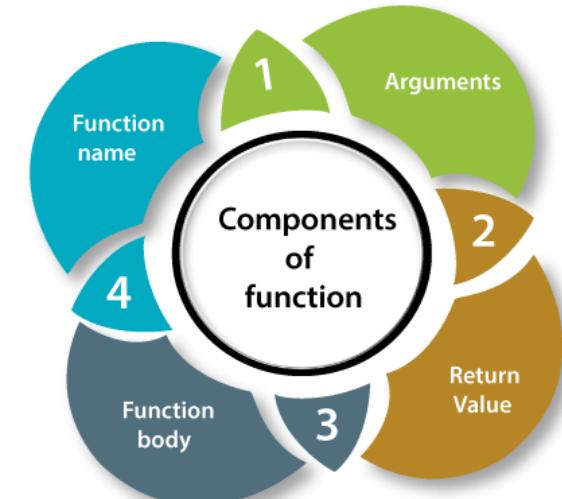
```
x <- 1:100  
y <- rnorm(100)  
  
myplot(x, y)  
myplot(x, y, col = "blue", las = 1, lwd = 2, main = "Meu gráfico")
```

# Funções com o argumento (...)



# Funções - Algumas Considerações

- Funções podem ser passadas como argumentos para outras funções. Isso é muito útil para as funções da família `apply` que iremos estudar em breve.
- Funções dentro de funções: você pode usar ou até definir funções dentro de funções. Isso é uma particularidade do R.
- A complexidade de uma função depende dos comandos que compõem o corpo da função.
- A melhor maneira de aprender sobre funções e suas potencialidades é praticando!!!



Fonte da imagem: <https://www.javatpoint.com/r-functions>

# Referências

Algumas referências utilizadas para a construção desse material:

[R Programming – Roger Peng](#)

[Introdução à Ciência de Dados - Rafael A. Irizarry](#)

[Curso de R online para iniciantes - Didática Tech - Aulas 4 e 17](#)

[Data Flair - R Tutorial Series](#)



Slides produzidos pela profa. Tatiana Benaglia.