Introdução a Linguagem R

Funções em R

Delermando Branquinho Filho

Funções

As funções são criadas usando a diretiva `function ()` e são armazenadas como objetos R como qualquer outra coisa. Em particular, são objetos R de classe "função".

```
f <- function(<arguments>) {
        ## Do something interesting
}
```

Funções em R são "objetos de primeira classe", o que significa que eles podem ser tratados como qualquer outro objeto R. Importante.

- Funções podem ser passadas como argumentos para outras funções
- Funções podem ser aninhadas, de modo que você pode definir uma função dentro de outra função
- O valor de retorno de uma função é a última expressão no corpo da função a ser avaliada.

Argumentos da função

Funções têm *named argumentos* que potencialmente têm valores *default*. - Os argumentos *formal* são os argumentos incluídos na definição da função - A função `formals` retorna uma lista de todos os argumentos formais de uma função - Nem todas as chamadas de função em R fazem uso de todos os argumentos formais - Os argumentos de função podem ser *missing* ou podem ter valores padrão

---

Argument Matching

Os argumentos de funções R podem ser combinados posicionalmente ou por nome. Então as chamadas para `sd` são todas equivalentes

```
> mydata <- rnorm(100)
> sd(mydata)
> sd(x = mydata)
> sd(x = mydata, na.rm = FALSE)
> sd(na.rm = FALSE, x = mydata)
> sd(na.rm = FALSE, mydata)
```

Mesmo que seja legal, eu não recomendo mexer com o agumentos demasiado, uma vez que pode levar a alguma confusão.

Argument Matching

Você pode misturar correspondência posicional com correspondência pelo nome. Quando um argumento é correspondido por nome, ele é "retirado" da lista de argumentos e os argumentos não nomeados restantes são

correspondidos na ordem em que são listados na função definition.usion.

```
> args(lm)
function (formula, data, subset, weights, na.action,
          method = "qr", model = TRUE, x = FALSE,
          y = FALSE, qr = TRUE, singular.ok = TRUE,
          contrasts = NULL, offset, ...)
```

As duas chamadas a seguir são equivalentes.

```
lm(data = mydata, y ~ x, model = FALSE, 1:100)
lm(y ~ x, mydata, 1:100, model = FALSE)
```

- Na maioria das vezes, os argumentos nomeados são úteis na linha de comando quando você tem uma longa lista de argumentos e você deseja usar os padrões para tudo exceto para um argumento próximo ao final da lista
- Argumentos nomeados também ajudam se você pode se lembrar do nome do argumento e não da sua posição na lista de argumentos (plotar é um bom exemplo).

Argument Matching

Os argumentos de função também podem ser comparados *partially*, o que é útil para o trabalho interativo. A ordem das operações quando se dá um argumento é

1. Verifique a correspondência exata de um argumento com nome
2. Verifique se há uma correspondência parcial
3. Verifique se há uma correspondência de posição.

Definindo uma Função

```
f <- function(a, b = 1, c = 2, d = NULL) {

}
```

Além de não especificar um valor padrão, você também pode definir um valor de argumento para `NULL`.

Lazy Evaluation

Argumentos para funções são avaliados *lazily*, portanto, eles são avaliados apenas conforme necessário.

```
f <- function(a, b) {
        a^2
}
f(2)
```

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

Esta função nunca usa o argumento `b`, então chamar`f (2)`não produzirá um erro porque o 2 obtém posicionalmente correspondente`a`.

Lazy Evaluation

```
f <- function(a = NULL, b = NULL) {
        print(a)
        print(b)
}
f(45)
```

```
## [1] 45
## NULL
```

Observe que "45" foi impresso primeiro antes do erro ser disparado. Isso ocorre porque `b` não precisa ser avaliado depois de`print (a)`. Uma vez que a função tentou avaliar `print (b)` teve que lançar um erro.

O argumento ...

O argumento ... indica um número variável de argumentos que normalmente são passados para outras funções.

- ... é freqüentemente usado ao estender uma outra função e você não quer copiar toda a lista de argumentos da função original

```
myplot <- function(x, y, type = "l", ...) {
        plot(x, y, type = type, ...)
}
```

Funções genéricas usam ... para que argumentos extras possam ser passados para métodos (Mais sobre isso mais tarde).

```
> mean
function (x, ...)
UseMethod("mean")
```

O argumento ... também é necessário quando o número de argumentos passados para a função não pode ser conhecido antecipadamente.

```
> args(paste)
function (..., sep = " ", collapse = NULL)

> args(cat)
function (..., file = "", sep = " ", fill = FALSE,
    labels = NULL, append = FALSE)
```

---

argumentos que vêm após o argumento "..."

Uma captura com ... é que os argumentos que aparecem *after* ... na lista de argumentos devem ser nomeados explicitamente e não podem ser parcialmente correspondidos.

```
> args(paste)
function (..., sep = " ", collapse = NULL)

> paste("a", "b", sep = ":")
[1] "a:b"

> paste("a", "b", se = ":")
[1] "a b :"
```

.

The Scientist