

Un poco sobre funciones. . .

Jorge Loría

September 7, 2017

Recapitulación clase pasada

Vimos que existen 5 tipos básicos de estructuras:

- ▶ Vectores

Recapitulación clase pasada

Vimos que existen 5 tipos básicos de estructuras:

- ▶ Vectores
- ▶ Listas

Resumen de la clase pasada

Vimos que existen 5 tipos básicos de estructuras:

- ▶ Vectores
- ▶ Listas
- ▶ Matrices

Recapitulación clase pasada

Vimos que existen 5 tipos básicos de estructuras:

- ▶ Vectores
- ▶ Listas
- ▶ Matrices
- ▶ DataFrames

Recapitulación clase pasada

Vimos que existen 5 tipos básicos de estructuras:

- ▶ Vectores
- ▶ Listas
- ▶ Matrices
- ▶ DataFrames
- ▶ Arrays

Principio

Todo lo que existe es un objeto, y todo lo que sucede es una llamada a una función - John Chambers
(parafraseado)

Principio

Todo lo que existe es un objeto, y todo lo que sucede es una llamada a una función - John Chambers
(parafraseado)

```
f1 <- function(x) x^2
```


Principio

Todo lo que existe es un objeto, y todo lo que sucede es una llamada a una función - John Chambers
(parafraseado)

```
f1 <- function(x) x^2
```

```
f1(10)
```

```
## [1] 100
```

Partes de las funciones en R

- ▶ `body()`

Partes de las funciones en R

► `body()`

```
body(f1)
```

```
## x^2
```

Partes de las funciones en R

- ▶ `body()`

```
body(f1)
```

```
## x^2
```

- ▶ `formals()`

Partes de las funciones en R

- ▶ `body()`

```
body(f1)
```

```
## x^2
```

- ▶ `formals()`

```
formals(f1)
```

```
## $x
```

La parte menos conocida

La parte menos conocida

- ▶ `environment()`

La parte menos conocida

► `environment()`

```
environment(f1)
```

```
## <environment: R_GlobalEnv>
```


Variables dentro de las funciones

Se pueden declarar *objetos* dentro de las funciones:

```
g1 <- function(x){  
  a <- 20  
  a + 10 + 2*x  
}
```

Al evaluar `g1(7)` se obtiene:

Variables dentro de las funciones

Se pueden declarar *objetos* dentro de las funciones:

```
g1 <- function(x){  
  a <- 20  
  a + 10 + 2*x  
}
```

Al evaluar `g1(7)` se obtiene: 44

Variables dentro de las funciones

Se pueden declarar *objetos* dentro de las funciones:

```
g1 <- function(x){  
  a <- 20  
  a + 10 + 2*x  
}
```

Al evaluar `g1(7)` se obtiene: 44

¿Cuál es el body de `g1`? ¿Y los formals?

Variables dentro de las funciones

Se pueden declarar *objetos* dentro de las funciones:

```
g1 <- function(x){  
  a <- 20  
  a + 10 + 2*x  
}
```

Al evaluar `g1(7)` se obtiene: 44

¿Cuál es el body de `g1`? ¿Y los formals? ¿Qué pasa si se llama `g1` sin ponerle parámetros?

Variables externas

Podemos tomar variables del ambiente exterior para llamar funciones:

```
a <- 15  
g2 <- function(b) a + b^2
```

¿Qué valor toma $g2(3)$?

Variables externas

Podemos tomar variables del ambiente exterior para llamar funciones:

```
a <- 15  
g2 <- function(b) a + b^2
```

¿Qué valor toma $g2(3)$? 24

Variables externas

Podemos tomar variables del ambiente exterior para llamar funciones:

```
a <- 15  
g2 <- function(b) a + b^2
```

¿Qué valor toma $g2(3)$? 24

Por lo que al evaluar una función, si no se tiene una variable en el ambiente actual, se busca en el ambiente que está justo superior a este. Y si no se encuentra se vuelve a subir y así sucesivamente. . .

Funciones dentro de funciones

Yo Dawg'



Figure 1: Meme obligatorio:

Funciones dentro de funciones

```
f2 <- function(x){  
  f3 <- function(y){  
    x+y  
  }  
  f3(5)  
}
```

¿Qué creen que pase en esta función? ¿Cual es el body de f2?
¿Qué pasa si se intenta llamar a f3? ¿Se puede?

Funciones dentro de funciones

```
f2 <- function(x){  
  f3 <- function(y){  
    x+y  
  }  
  f3(5)  
}
```

¿Qué creen que pase en esta función? ¿Cual es el body de f2?
¿Qué pasa si se intenta llamar a f3? ¿Se puede?

Entonces hay mucha flexibilidad con lo que se hace! Además,
porque se pueden definir funciones dentro de otras funciones

Funciones dentro de funciones

```
f2 <- function(x){  
  f3 <- function(y){  
    x+y  
  }  
  f3(5)  
}
```

¿Qué creen que pase en esta función? ¿Cual es el body de f2?
¿Qué pasa si se intenta llamar a f3? ¿Se puede?

Entonces hay mucha flexibilidad con lo que se hace! Además,
porque se pueden definir funciones dentro de otras funciones

```
f2(2)
```

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

```
# f3(1)
```

Ambiente de *f3*

```
f2 <- function(x){  
  f3 <- function(y){  
    x+y  
  }  
  print(environment(f3))  
  f3(5)  
}  
  
f2(2)
```

```
## <environment: 0x0000000013cbef90>
```

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

Las funciones también son un objeto

Por lo que puede ser parámetros de otras funciones, o incluso el objeto que se obtiene de una función, por ejemplo:

```
fun1 <- function(x){  
  y <- 2  
  function() x - y^2  
}
```

```
fun2 <- fun1(20)
```

¿Cuánto vale ahora fun2()?

Las funciones también son un objeto

Por lo que puede ser parámetros de otras funciones, o incluso el objeto que se obtiene de una función, por ejemplo:

```
fun1 <- function(x){  
  y <- 2  
  function() x - y^2  
}
```

```
fun2 <- fun1(20)
```

¿Cuánto vale ahora fun2()? 16

Las funciones también son un objeto

Por lo que puede ser parámetros de otras funciones, o incluso el objeto que se obtiene de una función, por ejemplo:

```
fun1 <- function(x){  
  y <- 2  
  function() x - y^2  
}
```

```
fun2 <- fun1(20)
```

¿Cuánto vale ahora `fun2()`? 16 El ambiente de `fun2` ya no es el global:

Las funciones también son un objeto

Por lo que puede ser parámetros de otras funciones, o incluso el objeto que se obtiene de una función, por ejemplo:

```
fun1 <- function(x){  
  y <- 2  
  function() x - y^2  
}
```

```
fun2 <- fun1(20)
```

¿Cuánto vale ahora fun2()? 16 El ambiente de fun2 ya no es el global:

```
environment(fun2)
```

```
## <environment: 0x00000000126316c0>
```

¿Qué cree que pase con el valor de fun2 después del siguiente chunk?

```
y <- 3  
fun2()
```

¿Qué cree que pase con el valor de fun2 después del siguiente chunk?

```
y <- 3  
fun2()
```

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

¿Qué cree que pase con el valor de `fun2` después del siguiente chunk?

```
y <- 3  
fun2()
```

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

¿Qué se obtiene como `body` de `fun2`? `formals`? Y si se llama sin ponerle los paréntesis?

Funciones como parámetro:

```
x <- list(1:3,10:15,21:23)  
sapply(x,sum)
```

```
## [1] 6 75 66
```

Qué hace sapply?

Funciones como parámetro:

```
x <- list(1:3,10:15,21:23)
sapply(x,sum)
```

```
## [1] 6 75 66
```

Qué hace sapply?

```
sapply(x,mean)
```

```
## [1] 2.0 12.5 22.0
```

Recibe 2 objetos, una lista y una función que le aplica a cada entrada de la lista.

Funciones como parámetro:

```
x <- list(1:3,10:15,21:23)
sapply(x,sum)
```

```
## [1] 6 75 66
```

Qué hace sapply?

```
sapply(x,mean)
```

```
## [1] 2.0 12.5 22.0
```

Recibe 2 objetos, una lista y una función que le aplica a cada entrada de la lista. Existe toda una familia de funciones de apply, que incluye: lapply, mapply, vapply, tapply, entre otras...

