

# Funciones

Oscar Perpiñán Lamigueiro

<http://oscarperpinan.github.io>

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

# Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

# Fuentes de información

- ▶ R introduction
- ▶ R Language Definition
- ▶ Software for Data Analysis

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

# Componentes de una función

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

- ▶ Una función se define con `function`

```
name <- function(arg_1, arg_2, ...) expression
```

- ▶ Está compuesta por:
  - ▶ Nombre de la función (`name`)
  - ▶ Argumentos (`arg_1, arg_2, ...`)
  - ▶ Cuerpo (`expression`): emplea los argumentos para generar un resultado

# Mi primera función

## ► Definición

```
myFun <- function(x, y){  
  x + y  
}
```

## ► Argumentos

```
formals(myFun)
```

```
$x
```

```
$y
```

## ► Cuerpo

```
body(myFun)
```

```
{  
  x + y  
}
```

# Mi primera función

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

```
myFun(1, 2)
```

```
[1] 3
```

```
myFun(1:10, 21:30)
```

```
[1] 22 24 26 28 30 32 34 36 38 40
```

```
myFun(1:10, 3)
```

```
[1] 4 5 6 7 8 9 10 11 12 13
```

# Argumentos: nombre y orden

- Una función identifica sus argumentos por su nombre y por su orden (sin nombre)

```
power <- function(x, exp){  
  x^exp  
}
```

```
power(x=1:10, exp=2)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

```
power(1:10, exp=2)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

```
power(exp=2, x=1:10)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

# Argumentos: valores por defecto

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

- Se puede asignar un valor por defecto a los argumentos

```
power <- function(x, exp=2){  
  x ^ exp  
}
```

```
power(1:10)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

```
power(1:10, 2)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea



# Funciones sin argumentos

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

```
hello <- function(){  
  print('Hello world!')  
}
```

```
hello()
```

```
[1] "Hello world!"
```

# Argumentos sin nombre: ...

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

```
pwrSum <- function(x, power, ...){  
  sum(x ^ power, ...)  
}
```

```
x <- 1:10  
pwrSum(x, 2)
```

```
[1] 385
```

```
x <- c(1:5, NA, 6:9, NA, 10)  
pwrSum(x, 2)
```

```
[1] NA
```

```
pwrSum(x, 2, na.rm=TRUE)
```

```
[1] 385
```

# Argumentos ausentes: missing

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

```
suma10 <- function(x, y){  
  if (missing(y)) y <- 10  
  x + y  
}
```

```
suma10(1:10)
```

```
[1] 11 12 13 14 15 16 17 18 19 20
```

# Control de errores: stopifnot

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

```
foo <- function(x, y){  
  stopifnot(is.numeric(x) & is.numeric(y))  
  x + y  
}
```

```
foo(1:10, 21:30)
```

```
[1] 22 24 26 28 30 32 34 36 38 40
```

```
foo(1:10, 'a')
```

```
Error: is.numeric(x) & is.numeric(y) is not TRUE
```

# Control de errores: stop

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

```
foo <- function(x, y){  
  if (!(is.numeric(x) & is.numeric(y))){  
stop('arguments must be numeric.')  
  } else { x + y }  
}
```

```
foo(2, 3)
```

```
[1] 5
```

```
foo(2, 'a')
```

```
Error en foo(2, "a") (from #3) : arguments must be numeric.
```

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

# Clases de variables

- ▶ Las variables que se emplean en el cuerpo de una función pueden dividirse en:
  - ▶ Parámetros formales (argumentos): x, y
  - ▶ Variables locales (definiciones internas): z, w, m
  - ▶ Variables libres: a, b

```
myFun <- function(x, y){  
  z <- x^2  
  w <- y^3  
  m <- a*z + b*w  
  m  
}
```

```
a <- 10  
b <- 20  
myFun(2, 3)
```

[1] 580

# Lexical scope

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

- Las variables libres deben estar disponibles en el entorno (environment) en el que la función ha sido creada.

```
environment(myFun)
```

```
<environment: R_GlobalEnv>
```

```
ls()
```

```
[1] "a"           "addTask"    "b"          "createTask" "createToDo"
[6] "foo"         "hello"      "lmFertEdu"  "myFun"      "myList"
[11] "myListOps"   "myToDo"     "power"      "print"      "pwrSum"
[16] "suma10"      "task1"      "task2"      "valida"     "x"
[21] "xyplot"
```



# Lexical scope: funciones anidadas

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

```
anidada <- function(x, y){  
  xn <- 2  
  yn <- 3  
  interna <- function(x, y){  
sum(x^xn, y^yn)  
}  
  print(environment(interna))  
  interna(x, y)  
}
```

```
anidada(1:3, 2:4)
```

```
<environment: 0x9482cc4>  
[1] 113
```

```
sum((1:3)^2, (2:4)^3)
```

```
[1] 113
```

# Lexical scope: funciones anidadas

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

```
xn
```

```
Error: objeto 'xn' no encontrado
```

```
yn
```

```
Error: objeto 'yn' no encontrado
```

```
interna
```

```
Error: objeto 'interna' no encontrado
```

# Funciones que devuelven funciones

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

```
constructor <- function(m, n){  
  function(x){  
    m*x + n  
  }  
}
```

```
myFoo <- constructor(10, 3)  
myFoo
```

```
function(x){  
  m*x + n  
}  
<environment: 0x9471d84>
```

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

# Funciones que devuelven funciones

```
class(myFoo)
```

```
[1] "function"
```

```
environment(myFoo)
```

```
<environment: 0x9471d84>
```

```
ls()
```

```
[1] "a"           "addTask"      "anidada"      "b"            "constructor"
[6] "createTask"  "createToDo"   "foo"          "hello"        "lmFertEdu"
[11] "myFoo"       "myFun"        "myList"       "myListOops"   "myToDo"
[16] "power"       "print"        "pwrSum"       "suma10"       "task1"
[21] "task2"       "valida"       "x"            "xyplot"
```

```
ls(env=environment(myFoo))
```

```
[1] "m" "n"
```

```
get('m', env=environment(myFoo))
```

```
[1] 10
```

```
get('n', env=environment(myFoo))
```

```
[1] 3
```

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

# Post-mortem: traceback

```
sumSq <- function(x, ...){  
  sum(x ^ 2, ...)  
}  
  
sumProd <- function(x, y, ...){  
  xs <- sumSq(x, ...)  
  ys <- sumSq(y, ...)  
  xs * ys  
}
```

```
sumProd(rnorm(10), runif(10))
```

```
[1] 24.38999
```

```
sumProd(rnorm(10), letters[1:10])
```

```
Error en x^2 (from #2) : argumento no-numérico para operador binario
```

```
traceback()
```

```
2: sumSq(y, ...) at #3  
1: sumProd(rnorm(10), letters[1:10])
```

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

# Analizar antes de que ocurra: debug

- ▶ Activa la ejecución paso a paso de una función

```
debug(sumProd)
```

- ▶ Cada vez que se llame a la función, su cuerpo se ejecuta línea a línea y los resultados de cada paso pueden ser inspeccionados.
- ▶ Los comandos disponibles son:
  - ▶ n o intro: avanzar un paso.
  - ▶ c: continua hasta el final del contexto actual (por ejemplo, terminar un bucle).
  - ▶ where: entrega la lista de todas las llamadas activas.
  - ▶ Q: termina la inspección y vuelve al nivel superior.
- ▶ Para desactivar el análisis:

```
undebug(sumProd)
```

# Analizar antes de que ocurra: trace

- trace permite mayor control que debug

```
trace(sumProd, tracer=browser, exit=browser)
```

```
[1] "sumProd"
```

- La función queda modificada

```
sumProd
```

```
Object with tracing code, class "functionWithTrace"
```

```
Original definition:
```

```
function(x, y, ...){  
  xs <- sumSq(x, ...)  
  ys <- sumSq(y, ...)  
  xs * ys  
}
```

```
## (to see the tracing code, look at body(object))
```

```
body(sumProd)
```

```
{  
  on.exit(.doTrace(browser(), "on exit"))  
  {  
    .doTrace(browser(), "on entry")  
    {  
      xs <- sumSq(x, ...)  
      ys <- sumSq(y, ...)  
      xs * ys  
    }  
  }  
}
```

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea



# Analizar antes de que ocurra: trace

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

- ▶ Los comandos `n` y `c` cambian respecto a debug:
  - ▶ `c` o `intro`: avanzar un paso.
  - ▶ `n`: continua hasta el final del contexto actual (por ejemplo, terminar un bucle).
- ▶ Para desactivar

```
untrace(sumProd)
```

# ¿Cuánto tarda mi función? `system.time`

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

```
noise <- function(sd) rnorm(1000, mean=0, sd=sd)
```

```
sumNoise <- function(nComponents){  
  vals <- sapply(seq_len(nComponents), noise)  
  rowSums(vals)  
}
```

```
system.time(sumNoise(1000))
```

```
   user  system elapsed  
0.384   0.016   0.402
```

# ¿Cuánto tarda cada parte de mi función?:

## Rprof

- ▶ Usaremos un fichero temporal

```
tmp <- tempfile()
```

- ▶ Activamos la toma de información

```
Rprof(tmp)
```

- ▶ Ejecutamos el código a analizar

```
zz <- sumNoise(1000)
```

# ¿Cuánto tarda cada parte de mi función?:

## Rprof

- Paramos el análisis

Rprof()

- Extraemos el resumen

summaryRprof(tmp)

```
$by.self
      self.time self.pct total.time total.pct
".External"    0.26   81.25      0.26   81.25
"array"         0.02    6.25      0.02    6.25
"rowSums"       0.02    6.25      0.02    6.25
"unlist"        0.02    6.25      0.02    6.25

$by.total
      total.time total.pct self.time self.pct
"sumNoise"      0.32   100.00      0.00    0.00
"sapply"        0.30   93.75      0.00    0.00
".External"     0.26   81.25      0.26   81.25
"FUN"           0.26   81.25      0.00    0.00
"lapply"        0.26   81.25      0.00    0.00
"rnorm"         0.26   81.25      0.00    0.00
"simplify2array" 0.04   12.50      0.00    0.00
"array"         0.02    6.25      0.02    6.25
"rowSums"       0.02    6.25      0.02    6.25
"unlist"        0.02    6.25      0.02    6.25
"as.vector"     0.02    6.25      0.00    0.00
```

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea

## do.call

- Ejemplo: sumar los componentes de una lista

```
lista <- list(a=rnorm(100), b=runif(100), c=rexp(100))  
with(lista, sum(a + b + c))
```

```
[1] 159.441
```

- En lugar de nombrar los componentes, creamos una llamada a una función con do.call

```
do.call(sum, lista)
```

```
[1] 159.441
```

# do.call

- Se emplea frecuentemente con el resultado de `lapply`

```
x <- rnorm(5)
ll <- lapply(1:5, function(i)x^i)
do.call(rbind, ll)
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1.288730e-01	0.57662772	-0.61057272	-1.983144	-1.227716
[2,]	1.660825e-02	0.33249953	0.37279905	3.932859	1.507287
[3,]	2.140355e-03	0.19172845	-0.22762093	-7.799424	-1.850521
[4,]	2.758339e-04	0.11055594	0.13897913	15.467378	2.271914
[5,]	3.554755e-05	0.06374962	-0.08485687	-30.674033	-2.789265

- Este mismo ejemplo puede resolverse con `sapply`

```
sapply(1:5, function(i)x^i)
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	0.1288730	0.01660825	0.002140355	2.758339e-04	3.554755e-05
[2,]	0.5766277	0.33249953	0.191728447	1.105559e-01	6.374962e-02
[3,]	-0.6105727	0.37279905	-0.227620933	1.389791e-01	-8.485687e-02
[4,]	-1.9831437	3.93285876	-7.799423899	1.546738e+01	-3.067403e+01
[5,]	-1.2277161	1.50728694	-1.850520516	2.271914e+00	-2.789265e+00

# Reduce

- Combina sucesivamente los elementos de un objeto aplicando una función binaria

```
Reduce('+', 1:10)  
## equivalente a  
## sum(1:10)
```

[1] 55



# Reduce

```
Reduce('/', 1:10)
```

```
[1] 2.755732e-07
```

```
Reduce(paste, LETTERS[1:5])
```

```
[1] "A B C D E"
```

```
foo <- function(u, v)u + 1 /v  
Reduce(foo, c(3, 7, 15, 1, 292), right=TRUE)  
## equivalente a  
## foo(3, foo(7, foo(15, foo(1, 292))))
```

```
[1] 3.141593
```

# Funciones recursivas

Funciones

Oscar Perpiñán  
Lamigueiro  
[http://  
oscarperpinan.  
github.io](http://oscarperpinan.github.io)

## ► Serie de Fibonnaci

```
fib <- function(n){  
  if (n>2) {  
    c(fib(n-1),  
      sum(tail(fib(n-1),2)))  
  } else if (n>=0) rep(1,n)  
}
```

```
fib(10)
```

```
[1] 1 1 2 3 5 8 13 21 34 55
```

Conceptos Básicos

Lexical scope

Debug y profiling

Miscelánea