

Funciones

Oscar Perpiñán Lamigueiro \
<http://oscarperpinan.github.io>

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Outline

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Conceptos Básicos

Lexical scope

Debug

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

Profiling

Miscelánea

Componentes de una función

Funciones

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

Conceptos Básicos

Lexical scope

Debug

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
}
```

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Mi primera función

Funciones

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

Conceptos Básicos

Lexical scope

Debug

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
```

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Argumentos: valores por defecto

- 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
```

Funciones sin argumentos

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

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

```
hello()
```

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

Argumentos sin nombre: ...

```
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
```

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Argumentos ausentes: missing

Funciones

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

Conceptos Básicos

Lexical scope

Debug

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)

```
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
```

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Mensajes para el usuario

stop para la ejecución y emite un mensaje de error

```
stop('Algo no ha ido bien.')
```

```
Error: Algo no ha ido bien.
```

warning no interfiere en la ejecución pero añade un mensaje a la cola de advertencias

```
warning('Quizás algo no es como debiera...')
```

```
Warning message:  
Quizás algo no es como debiera...
```

message emite un mensaje (**no usar cat o print**)

```
message('Todo en orden por estos lares.')
```

```
Todo en orden por estos lares.
```

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Conceptos Básicos

Lexical scope

Debug

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

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Lexical scope

Funciones

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

- 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"           "anidada"     "b"           "constructor" "fib"
[6] "foo"         "hello"       "lista"       "ll"          "myFoo"
[11] "myFun"       "noise"       "power"       "pwrSum"      "suma10"
[16] "sumNoise"    "sumProd"     "sumSq"       "tmp"         "x"
[21] "zz"
```

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Lexical scope: funciones anidadas

Funciones

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

Conceptos Básicos

Lexical scope

Debug

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: 0xacfef0>  
[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

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)

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

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

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

```
function(x)  
{  
  m*x + n  
}  
<environment: 0x1caaf38>
```

```
## 10*5 + 3  
myFoo(5)
```

```
[1] 53
```

Funciones que devuelven funciones

```
class(myFoo)
```

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

```
environment(myFoo)
```

```
<environment: 0x1caaf38>
```

```
ls()
```

```
[1] "a"           "anidada"      "b"            "constructor"  "fib"
[6] "foo"         "hello"        "lista"        "ll"           "myFoo"
[11] "myFun"       "noise"        "power"        "pwrSum"       "suma10"
[16] "sumNoise"    "sumProd"      "sumSq"        "tmp"          "x"
[21] "zz"
```

```
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

Profiling

Miscelánea

Conceptos Básicos

Lexical scope

Debug

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] 9.629438
```

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

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

```
traceback()
```

```
3: x^2 at #2  
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

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)
```

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

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

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

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)
```

Más recursos

- ▶ Debugging en RStudio
 - ▶ [Artículo](#)
 - ▶ [Vídeo](#)
- ▶ *Debugging* explicado por H. Wickham

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

¿Cuánto tarda mi función? `system.time`

Funciones

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

Conceptos Básicos

Lexical scope

Debug

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  
0.160   0.008   0.167
```

¿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)
```

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

¿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
"rnorm"      0.12   75.0      0.12   75.0
"array"      0.02   12.5      0.02   12.5
"unlist"     0.02   12.5      0.02   12.5

$by.total
      total.time total.pct self.time self.pct
"sapply"      0.16   100.0      0.00     0.0
"sumNoise"    0.16   100.0      0.00     0.0
"rnorm"      0.12   75.0      0.12   75.0
"FUN"        0.12   75.0      0.00     0.0
"lapply"     0.12   75.0      0.00     0.0
"simplify2array" 0.04   25.0      0.00     0.0
"array"      0.02   12.5      0.02   12.5
"unlist"     0.02   12.5      0.02   12.5
"as.vector"  0.02   12.5      0.00     0.0

$sample.interval
[1] 0.02
```


Conceptos Básicos

Lexical scope

Debug

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] 155.2168
```

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

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

```
[1] 155.2168
```

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,]	0.6957676	-0.47175880	-0.56271345	-0.8144094	-1.791272
[2,]	0.4840925	0.22255636	0.31664643	0.6632627	3.208656
[3,]	0.3368159	-0.10499292	-0.17818121	-0.5401674	-5.747578
[4,]	0.2343456	0.04953133	0.10026496	0.4399174	10.295476
[5,]	0.1630501	-0.02336684	-0.05642044	-0.3582729	-18.442002

- Este mismo ejemplo puede resolverse con sapply

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

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	0.6957676	0.4840925	0.3368159	0.23434559	0.16305007
[2,]	-0.4717588	0.2225564	-0.1049929	0.04953133	-0.02336684
[3,]	-0.5627135	0.3166464	-0.1781812	0.10026496	-0.05642044
[4,]	-0.8144094	0.6632627	-0.5401674	0.43991745	-0.35827292
[5,]	-1.7912723	3.2086565	-5.7475775	10.29547648	-18.44200195

Reduce

Funciones

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

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

```
## (((1+2)+3)+4)+5  
Reduce('+', 1:5)  
## equivalente a  
## sum(1:10)
```

[1] 15

Reduce

```
## (((1/2)/3)/4)/5  
Reduce('/', 1:5)
```

```
[1] 0.008333333
```

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

```
[1] 4.212948
```

```
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

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

Conceptos Básicos

Lexical scope

Debug

Profiling

Miscelánea

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

Profiling

Miscelánea