Funciones

Oscar Perpiñán Lamigueiro

19 de Febrero de 2013

Contenidos

Conceptos Básicos

Lexical scope

Debug y profiling

Sofisticaciones

Fuentes de información

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

Componentes de una función

- 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

```
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

- 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

```
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
```

Argumentos ausentes: `missing`

```
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`

```
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

```
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") : arguments must be numeric.
```

Contenidos

Conceptos Básicos

Lexical scope

Debug y profiling

Sofisticaciones

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

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

Lexical scope: funciones anidadas


```
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: 0xa645674>
[1] 113
```

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

```
[1] 113
```

Lexical scope: funciones anidadas

```
xn
```

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

```
yn
```

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

```
interna
```

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

Funciones que devuelven funciones

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

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

```
function(x){
        m*x + n
        }
<environment: 0xa63e3b8>
```

Funciones que devuelven funciones

```
class(myFoo)
```

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

```
environment(myFoo)
```

```
<environment: 0xa63e3b8>
```

```
ls()
```

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

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

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

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

```
[1] 10
```

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

```
[1] 3
```

Contenidos

Conceptos Básicos

Lexical scope

Debug y profiling

Sofisticaciones

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] 15.21856
```

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

```
Error en x^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])
```

Debugger

system.time


```
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
  0.244   0.020   0.265
```

Rprof

- Usaremos un fichero temporal

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

- Activamos la toma de información

```
Rprof(tmp)
```

- Ejecutamos el código a analizar

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

Rprof

- Paramos el análisis

```
Rprof()
```

- Extraemos el resumen

```
summaryRprof(tmp)
```

```
$by.self
        self.time self.pct total.time total.pct
"rnorm"      0.24       75       0.24        75
"array"      0.08       25       0.08        25

$by.total
                total.time total.pct self.time self.pct
"sapply"              0.32       100      0.00        0
"sumNoise"            0.32       100      0.00        0
"rnorm"               0.24        75      0.24       75
"FUN"                 0.24        75      0.00        0
"lapply"              0.24        75      0.00        0
"array"               0.08        25      0.08       25
"simplify2array"      0.08        25      0.00        0

$sample.interval
[1] 0.02

$sampling.time
[1] 0.32
```

Contenidos

Conceptos Básicos

Lexical scope

Debug y profiling

Sofisticaciones

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] 167.6913
```

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

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

```
[1] 167.6913
```

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,]  2.477327 -0.311525664 -1.342538  1.650633 0.365936506
[2,]  6.137148  0.097048239  1.802409  2.724589 0.133909527
[3,] 15.203720 -0.030233017 -2.419804  4.497296 0.049002384
[4,] 37.664581  0.009418361  3.248680  7.423385 0.017931761
[5,] 93.307473 -0.002934061 -4.361478 12.253284 0.006561886
```

- Este mismo ejemplo puede resolverse con `sapply`

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

```
           [,1]       [,2]        [,3]         [,4]         [,5]
[1,]  2.4773267 6.13714765 15.20371982 37.664581254 93.307473313
[2,] -0.3115257 0.09704824 -0.03023302  0.009418361 -0.002934061
[3,] -1.3425384 1.80240946 -2.41980398  3.248679858 -4.361477583
[4,]  1.6506329 2.72458904  4.49729637  7.423385446 12.253284406
[5,]  0.3659365 0.13390953  0.04900238  0.017931761  0.006561886
```

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

- 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
```