Programación Estadística: Programas

Jocelyn Simmonds (jsimmond@dcc.uchile.cl)

Departamento de Ciencias de la Computación

Agrupando expresiones

En R, se usan las llaves $\{\ldots\}$ para agrupar expresiones:

$$\{\texttt{expr\_1; expr\_2; ...expr\_n}\}$$

El "valor" de un grupo es el valor de la ultima expresión del grupo:

```
1  > x <- NULL
2  > y <- NULL
3  > mis_valores <- { x <- 1:10; y <- x^2 }
4  > x
5   [1]  1  2  3  4  5  6  7  8  9 10
6  > y
7   [1]   1   4   9  16  25  36  49  64  81 100
8  > mis_valores
9   [1]   1   4   9  16  25  36  49  64  81 100
```

Un grupo de expresiones es a la vez una expresión, pueden usar un grupo en cualquier parte donde pueden colocar una expresión.

Ejecutando comandos desde un script

En un archivo llamado `miprograma.R`:

```
1  mis_valores <- { x <- 1:10; y <- x^2 }
2
3  otros_valores <- { lista <- mis_valores + 2 }
```

En R:

```
1  > source("miprograma.R")
2  > x
3   [1]  1  2  3  4  5  6  7  8  9 10
4  > y
5   [1]   1   4   9  16  25  36  49  64  81 100
6  > mis_valores
7   [1]   1   4   9  16  25  36  49  64  81 100
8  > lista
9   [1]   3   6  11  18  27  38  51  66  83 102
10 > otros_valores
11  [1]   3   6  11  18  27  38  51  66  83 102
```

Condiciones

Ejecución condicional de expresiones

Podemos incluir condiciones usando `if (cond) expr`:

```
> x <- 1:10
> a
[1] 5
> if (a != 0) x <- x/a      # en este caso, se ejecuta la expr del if
> x
 [1] 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
>
>                           # en este caso, no se ejecuta la expr del if
> {x <- NULL; a <- 0; if (a != 0) x <- x/a }
> x
NULL
> a
[1] 0
```

Ojo: la condición debe producir un único valor lógico, o sea, la evaluación de la condición debe generar un vector lógico de largo 1.

Ejecución condicional de expresiones

Podemos incluir caminos alternativos usando `if ... else`:

```
> a
[1] 5
> if (a > 1) x <- 1 else x <- 0
> x
[1] 1
>
>  {x <- NULL; a <- 4; b <- 7; if (a - b > 0) x <- (b:a)^(a - b)
+ else x <- (a:b)^(b - a)}
> x
[1]  64 125 216 343
```

De nuevo, la evaluación de la condición debe generar un único valor lógico.

Condiciones compuestas

- Los operadores lógicos & y | se aplican sobre vectores, elemento por elemento (ver Clase 1).
- Caso especial: si una condición es acerca de vectores de largo 1, podemos usar los operadores && y ||.
  - `expr_1 && expr_2`: si `expr_1` es `FALSE`, no se evalúa `expr_2`, dado que la condición ya es falsa.
  - `expr_1 || expr_2`: si `expr_1` es `TRUE`, no se evalúa `expr_2`, dado que la condición ya es verdadera.

Ejemplos:

```
1  > # no se imprime el mensaje "12 es mayor que 5" porque "0 == 1" es falso
2  > {a <- 12; if (0 == 1 && a > 5) print(paste(a, "es mayor que 5"))}
3  >
4  > # se imprime el mensaje, porque "1 == 1" es siempre verdadero
5  > {a <- 3; if (1 == 1 || a > 5) print(paste(a, "es mayor a 5"))}
6  [1] "3 es mayor a 5"
```

Cadenas de `if ... else`

Pueden ejecutar cadenas de instrucciones `if ... else`:

```
> cliente <- "habitual"
> descuento <-
+ {if (cliente == "habitual"){
+     descuento <- 0.1
+ } else if (cliente == "semi-habitual"){
+     descuento <- 0.05
+ } else {
+     descuento <- 0.0
+ }}
>
> descuento
[1] 0.1
```

Ejecución condicional sobre vectores

Existe una versión del `if` para condiciones sobre vectores:
`ifelse(cond, vector_1, vector_2)`

```
> labels
 [1] "Libro 1"  "Libro 2"  "Libro 3"  "Libro 4"  "Libro 5"  "Libro 6"
 [7] "Libro 7"  "Libro 8"  "Libro 9"  "Libro 10"
> digits
 [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
> ifelse(as.integer(digits) %% 2 == 0, digits, labels)
 [1] "0"        "Libro 2"  "2"        "Libro 4"  "4"        "Libro 6"
 [7] "6"        "Libro 8"  "8"        "Libro 10"
>
```

Esta instrucción retorna un vector de largo `max(vector_1, vector_2)`, donde en la *i*-esima posición tiene por valor `vector_1[i]` si `cond[i]` es `TRUE`, sino tiene por valor `vector_2[i]`.

Funciones

Creando una función

Pueden crear funciones usando la función `function`:
`nombre_funcion <- function(parametro_1, parametro_2, ...)`

```
1  potencia_vec <- function(x, n){
2     y <- x^n
3  }
```

Guarden sus funciones en un archivo .R y usen `source()` para cargarlas a la consola.

```
1  > valores <- potencia_vec(1:10, 3)
2  > valores
3   [1]    1    8   27   64  125  216  343  512  729 1000
```

¿Script o función? Parámetros . . .

Funciones reciben parámetros, o sea, podemos ejecutar un mismo "script" muchas veces, solo alterando los parámetros con que invocamos la función:

```
1  > valores <- potencia_vec(1:10, 3)
2  > valores
3   [1]    1    8   27   64  125  216  343  512  729 1000
4  > valores <- potencia_vec(5:13, 2)
5  > valores
6  [1]  25  36  49  64  81 100 121 144 169
```

¿Script o función? Asignaciones locales . . .

Toda asignación que ocurre dentro de una función es local, o sea, es temporal y se pierde al terminar la ejecución de la función:

```
> potencia_vec
function(x, n){
y <- x^n
}
> y <- NULL
> x <- NULL
> n <- NULL
> valores <- potencia_vec(1:10, 3)
> x
NULL
> n
NULL
> y
NULL
> valores
 [1]    1    8   27   64  125  216  343  512  729 1000
```

Valores por defecto

Pueden asignar valores por defecto cuando definen una función:

```
1  > potencia_vec
2  function(x, n=2){
3  y <- x^n
4  }
5  > potencia_vec(1:10)
6  > valores <- potencia_vec(1:10)
7  > valores
8   [1]   1   4   9  16  25  36  49  64  81 100
9  > valores <- potencia_vec(1:10, n=4)
10 > valores
11  [1]     1    16    81   256   625  1296  2401  4096  6561 10000
12 > valores <- potencia_vec(1:10)
13 > valores
14  [1]   1   4   9  16  25  36  49  64  81 100
```

Repetición

Repitiendo instrucciones: `for`

Podemos repetir instrucciones usando `for (var in expr_1) expr_2`:

Definición de función:

```
1  convertir <- function(x){
2     for (i in 1:length(x)){
3        if (x[i] < 0)
4           x[i] <- -1 * x[i]
5     }
6     x
7  }
```

Salida en consola:

```
1  > x <- c(-3, 4, 5, -2, -9, 5)
2  > convertir(x)
3  [1] 3 4 5 2 9 5
```

Es usual que `expr_1` sea un vector de índices.

Ojo: este es un ejemplo académico para mostrar como funciona el `for`. En general, es mas eficiente procesar vectores "en bloque" que uno a uno.

Repitiendo instrucciones: `repeat`

Podemos repetir instrucciones usando `repeat`.

Definición de función:

```
1  imprimir <- function(){
2      i <- 1
3      repeat {
4          if (i == 5) break
5          print(i)
6          i <- i + 1
7      }
8  }
```

Salida en consola:

```
1  > imprimir()
2  [1] 1
3  [1] 2
4  [1] 3
5  [1] 4
6  >
```

La única forma de finalizar un ciclo `repeat` es usando un `break`.

Repitiendo instrucciones: `while`

Podemos repetir instrucciones usando `while (cond) expr`:

Definición de función:

```
1  potencias2 <- function(){
2     i <- 0
3     while (2^i < 1000){
4        print(paste(i, 2^i))
5        i <- i + 1
6     }
7  }
```

Salida en consola:

```
1  > potencias2()
2  [1] "0 1"
3  [1] "1 2"
4  [1] "2 4"
5  [1] "3 8"
6  [1] "4 16"
7  [1] "5 32"
8  [1] "6 64"
9  [1] "7 128"
10 [1] "8 256"
11 [1] "9 512"
```

Pueden usar `break` para terminar el ciclo en forma temprana, y pueden usar `next` para avanzar a la siguiente iteración.