

Nota 3 - Funciones y Manipulación de Datos

Santiago Casanova y Ernesto Barrios

Funciones

En esta sección vamos a aprender a definir y usar funciones para facilitar la organización y el uso de código en el futuro.

Definición de funciones

Las funciones se definen como cualquier objeto en R. El nombre del objeto seguido del operador de asignación `<-` pero en este caso después viene la palabra clave `function()`.

Veámoslo en acción para entender mejor este concepto.

```
divide_dos_num <- function(num1, num2){ #El código a correr por la función se escribe entre llaves  
  num1/num2  
}
```

Nótese que, similar a cómo no es necesario utilizar `print()` para que la consola responda, la respuesta de la función no requiere ni `print()` ni `return`.

Si llamamos a nuestra función:

```
divide_dos_num(10,5)
```

```
## [1] 2
```

podemos verificar que funciona como esperamos.

Ejecución Controlada

if, if else, else e ifelse

Es posible que queramos controlar o limitar cuándo corre nuestra función y cuándo no. En nuestro ejemplo no queremos que el segundo número sea cero porque obtendremos NaN por respuesta. Utilizaremos la función `if()` para controlar la ejecución.

```
divide_dos_num <- function(num1, num2){  
  if(num2 == 0){  
    resp <- "No es posible dividir entre cero"  
  } else{  
    resp <- num1/num2  
  }  
  resp  
}  
  
divide_dos_num(10,5)
```

```
## [1] 2
```

```
divide_dos_num(10,0)
```

```
## [1] "No es posible dividir entre cero"
```

Si queremos evaluar más de dos condiciones utilizamos `if` en combinación con `else if` y `else` para manejar los casos que no cumplan ninguna de las dos. Es importante notar que el orden de las condiciones si se toma en cuenta. Se evaluará la primera que se cumpla

Veamos un ejemplo:

```
divide_dos_num <- function(num1, num2){  
  if(num2 == 0){  
    resp <- "No es posible dividir entre cero"  
  } else if(num1 == 0){  
    resp <- 0  
  } else{  
    resp <- num1/num2  
  }  
  resp  
}
```

```
divide_dos_num(10,5)
```

```
## [1] 2
```

```
divide_dos_num(0,0)
```

```
## [1] "No es posible dividir entre cero"
```

```
divide_dos_num(0,10)
```

```
## [1] 0
```

Aunque nuestra segunda llamada a la función cumplía ambas condiciones, solamente obtenemos el resultado de que se cumpla la primera condición.

Si queremos evaluar más de una condición en la misma expresión, podemos utilizar los operadores `&` y `|` que simbolizan AND y OR respectivamente.

Aplicamos esto a nuestro ejemplo anterior.

```
divide_dos_num <- function(num1, num2){  
  if(num2 == 0){  
    resp <- "No es posible dividir entre cero"  
  } else if(num1 == 0 | num2 == 1){ #operador OR |  
    resp <- 'Es trivial'  
  } else{  
    resp <- num1/num2  
  }  
  resp  
}
```

```
divide_dos_num(0,5)
```

```
## [1] "Es trivial"
```

```
divide_dos_num(0,0)
```

```
## [1] "No es posible dividir entre cero"
```

```
divide_dos_num(20,1)
```

```
## [1] "Es trivial"
```

Otra forma de obtener un valor a partir de una expresión lógica en R es con la función `ifelse`. Esta condensa las funcionalidades de `if` y `else` en una función llamable con 3 parámetros. El primer parámetro es la expresión a evaluar, el segundo el resultado en caso de que la expresión sea `TRUE` y el tercero el resultado en caso de que la expresión sea `FALSE`.

Esta función es especialmente útil para darle valor a una variable u obtener una respuesta rápida a partir de una condición.

```
pos_neg <- function(numero){  
  ifelse(numero > 0, 'positivo', 'negativo')  
  #expresion      #resultado T #resultado F  
}  
  
pos_neg(-2)
```

```
## [1] "negativo"
```

```
pos_neg(3)
```

```
## [1] "positivo"
```

Como se podrán haber dado cuenta, en este ejemplo no estamos evaluando todas las condiciones posibles. Si corremos

```
pos_neg(0)
```

```
## [1] "negativo"
```

nos regresa como resultado que cero es negativo ya que no cumple la condición `x>0`. Para estos casos podemos anidar nuestros `ifelse` para que manejen mas de una condición.

```
pos_neg_cero <- function(numero){  
  ifelse(numero == 0, 'cero', ifelse(numero > 0, 'positivo', 'negativo'))  
}
```

En este caso evaluamos primero si el numero es igual a cero. Si no lo es, regresamos al `ifelse` que habíamos planteado en la función anterior.

```
pos_neg_cero(-1)
```

```
## [1] "negativo"
```

```
pos_neg_cero(1231)
```

```
## [1] "positivo"
```

```
pos_neg_cero(0)
```

```
## [1] "cero"
```

for y while

En esta sección no analizaremos muy a fondo el funcionamiento o propósito de los ciclos `for` y `while` sino que veremos su sintaxis específica en R.

el `for` se escribe de la siguiente manera:

```
for(i in c(1,2,3)){# el rango debe ser un vector por el cual pueda correr la i
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

Recordemos las maneras de escribir rangos que analizamos en la nota anterior. Ya sea con la sintaxis inicio:fin, con el uso de una función generadora como seq() o bien proporcionando un vector explícito como el rango buscado.

```
#secuencia del 1 al 5
for(j in seq(5)){
  print(j)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
#de dos en dos del 3 al 10
```

```
for(k in seq(3,10, 2)){
  print(k)
}
```

```
## [1] 3
## [1] 5
## [1] 7
## [1] 9
```

```
#rangos específicos (no necesitan ser numéricos)
```

```
for(l in c('opcion1', 'opcion2', 'opcion3')){
  print(l)
}
```

```
## [1] "opcion1"
## [1] "opcion2"
## [1] "opcion3"
```

Por otro lado el while es muy sencillo e intuitivo. Se escribe la función while()y como parámetro se proporciona una expresión que tenga como salida TRUE o FALSE.

Nuevamente, el código a correr se escribe entre llaves ({}).

Mientras la condición tenga salida TRUE se corre el código dentro de las llaves del while.

```
i = 5
while(i>0){
  print(i)
  i = i-1
}
```

```
## [1] 5
## [1] 4
```

```
## [1] 3  
## [1] 2  
## [1] 1
```

Nótese que en muchos casos es necesario inicializar la condición antes de llamar a la función while.