Creación de funciones

David García Callejas

Las funciones son la base de R, hasta el punto de que cualquier operación entre variables puede ser considerada como aplicar una función a uno o más objetos. Hay infinidad de situaciones en las que nos puede resultar útil crear funciones específicas para resolver operaciones. En particular, crear una función nos ayudará a modularizar el código, de manera que cuando queramos cambiar la función, sólo tendremos que hacerlo en su definición, y podremos aplicarla sin cambiar ninguna otra parte del código donde la ejecutamos. Tener código "encapsulado" en funciones también nos ayudará a hacer más legible nuestro código, y a que sea más fácilmente compartible.

Ya conocemos la sintaxis general para utilizar funciones en R. Veamos el esqueleto para crear una función desde cero.

```
mi.funcion <- function(argumento){
  # codigo
  return()
}
```

En este esqueleto, creamos una función que se llama `mi.funcion`, con un único argumento. Dentro de la definición de la función es donde irá nuestro código, y la última linea, `return`, contendrá el resultado que queremos que devuelva la función.

Como primer ejemplo, podemos crear una función que normalice un vector numérico respecto a su valor máximo

```
v1 <- sample(1:100,10,FALSE)
max(v1)

# este es el código que queremos incluir en nuestra función
v2 <- v1/max(v1)

normaliza_vector <- function(v = NULL){
  res <- v/max(v)
  return(res)
}

v3 <- normaliza_vector(v1)
identical(v2,v3)
```

Es recomendable que cada función que definamos esté codificada en un script diferente. El nombre del script debe ser igual al del nombre de la función (en el ejemplo anterior, tendríamos un archivo "normaliza_vector.R"). En el archivo sólo debería incluirse la definición de la función. Una vez hayamos creado el archivo "normaliza_vector.R", podemos usar nuestra función en otros scripts. Para ello la cargamos en memoria con la orden

```
source("normaliza_vector.R")
```

donde, dependiendo del directorio de trabajo, necesitaremos especificar la ruta a nuestro archivo "normaliza_vector.R".

Una función puede aceptar tantos argumentos como sea necesario. A su vez, podemos especificar si los argumentos tienen o no valores por defecto, en caso de que no los especifiquemos de manera explícita al llamar a la función. Si no especificamos un valor por defecto, llamar a la función dará error si no incluimos el argumento en cuestión. En el siguiente ejemplo, `arg1` y `arg3` tienen valores por defecto que podemos sobreescribir. `arg2`, sin embargo, no tiene valor por defecto.

```
prueba.argumentos <- function(arg1 = NULL, arg2, arg3 = 0){
  cat("arg1:",arg1,"\narg2:",arg2,"\narg3:",arg3)
}

prueba.argumentos(arg1 = 1,arg2 = 1,arg3 = 1)
prueba.argumentos(arg2 = 1)
# prueba.argumentos()
```

Una parte importante de la creación de funciones es la documentación del funcionamiento, resultados, y argumentos de las mismas. Fijaos que cualquier función que usamos en R tiene una documentación asociada, a la que podemos acceder a través de la orden `?nombre_función`. Idealmente, deberíamos aspirar a documentar nuestras funciones de manera igual de completa que las funciones que consultamos. Esto se puede hacer a través de un método general, con el paquete `roxygen2`. En Rstudio, debemos tener el cursor en la definición de una función nueva (es decir, en la parte `function(...)`). Una vez ahí, pinchando en el menú `Code/Insert Roxygen Skeleton`, nos creará una serie de líneas de este estilo:

```
#' Title
#'
#' @param v
#'
#' @return
#' @export
#'
#' @examples
normaliza_vector <- function(v = NULL){
  res <- v/max(v)
  return(res)
}
```

Estas líneas contienen la información básica que debemos documentar para nuestras funciones. Tienen una estructura parecida a los comentarios normales de R, pero añadiendo un apóstrofe a la almohadilla. La primera línea es el título de nuestra función. Debajo podemos incluir una descripción más pormenorizada. Los argumentos de la función vienen dados por el campo `@param`. El resultado de la función por el campo `@return`. El campo `@export` lo podemos dejar en blanco, nos indica que esa función debe estar disponible en memoria al cargarla. En `@examples` podemos incluir ejemplos de uso de nuestra función. Nuestra función documentada quedaría así:

```
#' Normaliza un vector numerico
#'
#' Normaliza un vector numerico en función del valor máximo del mismo.
#'
#' @param v vector numérico de longitud arbitraria
#'
#' @return vector numérico normalizado entre 0-1
#' @export
#'
#' @examples
#' v1 <- runif(10,0,100)
#' vnorm <- normaliza_vector(v1)
normaliza_vector <- function(v = NULL){
```

```
  res <- v/max(v)
  return(res)
}
```

Si nuestras funciones utilizan funciones de otros paquetes, lo ideal es llamarlas usando la notación paquete::función. Por ejemplo, podemos definir una función que nos devuelva el número de palabras con más de N letras en una cadena de caracteres.

```
#' Número de palabras de longitud mayor que un valor
#'
#' @param v vector de caracteres.
#' @param l numero; palabras con un número de caracteres mayor que l
#' serán incluidas en el conteo.
#'
#' @return vector numérico, número de palabras con más de l caracteres
#' en cada elemento.
#' @export
#'
#' @examples
#' v1 <- c("este es un vector enormemente variado",
#' "en cuanto a longitud de palabras")
#' v2 <- num_palabras(v1,6)
num_palabras <- function(v = NULL, l = 0){

  # creamos el objeto que devolverá la función
  res <- NULL

  if(!is.null(v)){
    #https://stackoverflow.com/questions/33226616/
    #how-to-remove-words-of-specific-length-in-a-string-in-r
    mi.regex <- paste("\\w{",l,",}",sep="")
    # selecciona palabras mayores que l
    palabras <- stringr::str_extract_all(v, mi.regex)
    # las cuenta
    res <- sapply(palabras,length)
  }
  return(res)
}
```