Crear funciones en R

La sintaxis básica de una función de R es la siguiente.

```
nombre_funcion <- function(arg1, arg2, ... ) {
        # Código
}
```

En el bloque de código anterior tenemos las siguientes partes:

- ✓ arg1, arg2, ... son los argumentos de entrada.
- ✓ # Código representa el código a ser ejecutado dentro de la función para obtener la salida deseada.

La salida de la función puede ser un número, una lista, un data.frame, un gráfico, un mensaje o cualquier objeto que queramos.

Creando una función en R

La siguiente función corresponde a una progresión aritmética:

$$a_2 = a_1 \cdot r \; ;$$
$$a_3 = a_2 \cdot r = a_1 \cdot r_2; \; \ldots$$

$$a_n = a_1 \cdot r^{n-1}$$

```
an <- function(a1, r, n){
    a1 * r ** (n - 1)
}
```

En el siguiente bloque podemos ver algunos ejemplos, mostrando su salida como comentarios.

```
an(a1 = 1, r = 2, n = 5)  # 16
an(a1 = 4, r = -2, n = 6) # -128
```

Con la función anterior puedes obtener varios valores de la progresión pasando un vector al argumento n.

```
an(a1 = 1, r = 2, n = 1:5)   # a_1, ..., a_5
an(a1 = 1, r = 2, n = 10:15) # a_10,..., a_15
values) # 31
```

Argumentos de entrada

Los argumentos son valores de entrada de las funciones. Como ejemplo, en la función que creamos antes tenemos tres argumentos de entrada llamados a1, r y n.

Hay varias consideraciones cuando se trata con este tipo de argumentos.

- ✓ Si mantenemos el orden de entrada, no necesitamos llamar a los nombres de los argumentos. Como ejemplo, las siguientes llamadas a la función son equivalentes.

```
an(1, 2, 5) # Devuelve 16

an(a1 = 1, r = 2, n = 5) # Devuelve 16
```

- ✓ Si ponemos el nombre de los argumentos, podemos usar cualquier orden.

```
an(r = 2, n = 5, a1 = 1) # Devuelve 16

an(n = 5, r = 2, a1 = 1) # Devuelve 16
```

- ✓ Podemos usar la función args para conocer los argumentos de entrada de cualquier función que queramos usar.

```
args(an)
```

- ✓ Si escribimos el nombre de la función, la consola devolverá el código de la función.

Argumentos adicionales en las funciones de R

En ocasiones resulta muy interesante tener argumentos predeterminados en la función. En tal caso se utilizarán los valores predeterminados a menos que se incluyan otros al ejecutar la función. Al escribir una función, como la de nuestro ejemplo,

```
nombre_función <- function(arg1, arg2, arg3 ) {
        # Código
}
```

si queremos que arg2 y arg3 sean los valores a y b por defecto, podemos asignarlos en los argumentos de nuestra función.

```
nombre_función <- function(arg1, arg2 = a, arg3 = b) {
        # Código
}
```

Ilustraremos esto con un ejemplo muy simple. Considera una función que dibuja la función coseno.

```
coseno <- function(w = 1, min = -2 * pi, max = 2 * pi) {
    x <- seq(-2 * pi, 2 * pi, length = 200)
    plot(x, cos(w * x), type = "l")
}
```

Argumentos adicionales en R

El argumento **...** (punto punto punto) permite pasar argumentos libremente que se usarán en una subfunción dentro de la función principal. Como ejemplo, en la función,

```
coseno <- function(w = 1, min = -2 * pi, max = 2 * pi, ...) {
    x <- seq(-2 * pi, 2 * pi, length = 200)
    plot(x, cos(w * x), type = "l", ...)
}

mi_funcion <- function (...) {
  # Los argumentos se capturan en una lista
  argumentos <- list(...)
```

```
  # Imprimimos cada argumento
  for(i in seq_along(argumentos)) {
    print(argumentos[[i]])
  }
}

# Llamamos a la función con diferentes números y tipos de argumentos
mi_funcion("Hola", "Mundo", 123)
```

La función return

Por defecto, las funciones de R devolverán el último objeto evaluado dentro de ella. También podemos hacer uso de la función return, que es especialmente importante cuando se quiere devolver un objeto u otro dependiendo de ciertas condiciones o cuando se quiere ejecutar algún código después del objeto que queremos devolver.

```
# Definimos una función que calcula el cuadrado de un número
cuadrado <- function(x) {
  # Calculamos el cuadrado del argumento
  resultado <- x^2

  # Devolvemos el resultado
  return(resultado)
}

# Llamamos a la función con el argumento 4
print(cuadrado(4))
```

Variables locales y globales en R

En R no es necesario declarar las variables utilizadas dentro de una función. La regla llamada "ámbito lexicográfico" se usa para decidir si un objeto es local a una función o global. Considera el siguiente ejemplo:

```
fun <- function() {
    print(x)
}

x <- 1

fun() # 1
```

La variable x no se define dentro de **fun**, por lo que R buscará x dentro del ámbito "circundante" e imprimirá su valor. Si x se usa como el nombre de un objeto dentro de la función, el valor de x en el entorno global (fuera de la función) no cambia.

```
x <- 1
fun2 <- function() {
    x <- 2
    print(x)
}
fun2() # 2

x #1
```

Para cambiar el valor global de una variable dentro de una función, puedes usar el operador de doble asignación **(<<-)**.

```
x <- 1
y <- 3
fun3 <- function() {
    x <- 2
    y <<- 5
    print(paste(x, y))
}

fun3() # 2 5
x # 1 (el valor no cambió)
y # 5 (el valor cambió)
```