Análisis y Tratamiento de Datos con R

Con ejemplos e ilustraciones

Primera Edición

Diego Paul Huaraca S.
MS-PLUS, Inc.

MS+
Soluciones
Matemáticas

Un aporte de Source Stat Lab Ecuador a la sociedad.

Índice general

1

Funciones

En los capítulos anteriores ya hemos hecho uso de una variedad de funciones que vienen con R. En este capítulo abordaremos el diseño y construcción de funciones propias realizadas por el usuario, la posibilidad de encapsular fragmentos de código que puedan ser ejecutados en múltiples ocasiones y con diferentes parámetros da una gran ventaja a R sobre el resto de programas estadísticos.

1.1. Creación

Las funciones dentro de R, son tratadas como cualquier otro objeto. El comando `function()` tiene como tarea crear funciones a partir de un número establecido de argumentos o valores iniciales necesarios para la ejecución. La declaración de una función consta de cuatro partes:

1. **Nombre**: Carácter asignado para el llamado a la función, mismo que debe cumplir con ciertas condiciones:

   - Iniciar con una letra,
   - No tener espacios en blanco, ni símbolos reservados.

Recomendados	No permitidos
MiFuncion	Mi-Funcion, Mi+Funcion, MiFunción, mi:funcion
Codigo_01	Código #1, 1 Código, Codigo.0.1, codigo$

   En el nombre de una función si se puede emplear el signo de punto (.), sin embargo no es muy aconsejable su uso debido que el mismo está reservado para la definición de métodos.

2. **Argumentos**: Son un serie de valores que recibe la función para operar y lograr obtener un resultado. Pueden existir funciones que carezcan de argumentos, siempre y cuando las acciones a realizarse no dependan de ningún valor enviado por el usuario.

   Vale la pena aclarar que el orden en que se pasen los valores de los argumentos corresponde con la ubicación de estos en la declaración de la función.

3. **Código**: Conjunto de instrucciones necesarias para alcanzar un resultado.

4. **Resultado**: Objeto devuelto tras la ejecución de la función.

```
NombreFuncion <- function (Argumento1,..., ArgumentoN)
{
 expresión 1
 .......     # Código
 expresión N
 return(val) # Resultado
 }
```

Procedemos a crear nuestra primera función:

```
fr <- function(x){
   res <- 2*x
   return(res)
}
```

La función `fr` recibe como argumento un objeto x (vector) y arroja como resultado el objeto x multiplicado por 2.

```
fr(2)

## [1] 4

fr(c(3,5))

## [1]  6 10
```

En las líneas anteriores observamos que las palabras `function` y `return` son *reservadas* (palabras propias del lenguaje R). Vale hacer énfasis en que toda función inicia con la especificación de su nombre y termina con el comando `return` que especifica el resultado que debe mostrarse como salida.

1.1.1. Componentes

Todas las funciones en R constan de tres partes fundamentales en la etapa de ejecución:

- **body**: Corresponde al código que se encuentra dentro de la función, es la encargada de realizar las operaciones para alcanzar el resultado deseado.

  ```
  body(fr)

  ## {
  ##     res <- 2 * x
  ##     return(res)
  ## }
  ```

- **formals**: Corresponde a la lista de argumentos que controlan la llamada a la función.

  ```
  formals(fr)

  ## $x
  ```

- **environment**: Localización de las variables de la función dentro de la jerarquía de ambientes.

```
environment(fr)

## <environment: R_GlobalEnv>
```

Si el ambiente no se visualiza significa que la función fue creada en el ambiente global.

Para evidenciar la inclusión de dos o más argumentos, crearemos una función que nos calcule la hipotenusa de un triángulo rectángulo, la cual constará de dos argumentos (catetos):

```
hipotenusa <- function(x,y){ # x, y son los catetos
   h <- sqrt(x^2 + y^2) # Se calcula la hipotenusa
   return(h) # Devuelve el resultado
}
```

La salida de la función corresponde a la hipotenusa del triángulo rectángulo, contrastamos los resultados:

```
hipotenusa(4,3)

## [1] 5

hipotenusa(12, 16)

## [1] 20
```

1.1.2. Estructuras de control

Son un conjunto de herramientas que permite manejar de forma mucho más elaborada el flujo de ejecución de nuestro código.

Estructura IF

La estructura de control condicional o selectiva `if` nos permite ejecutar un bloque de código siempre y cuando se cumpla con la condición impuesta por el usuario. Su sintaxis es la siguiente:

```
if(condición){
   ....
   código
   ....
}
```

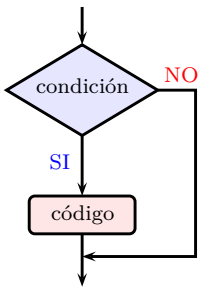

Figura 1.1: Estructura IF

Algunas consideraciones acerca del uso:

1. Olvidar los paréntesis al colocar la condición es un error sintáctico[1].
2. Confundir el operador de comparación == con el operador de asignación = puede producir errores inesperados.

[1]Se produce cuando se escribe código de una forma no admitida por las reglas del lenguaje. Los errores de sintaxis son detectados casi siempre por el compilador, que a su vez muestra un mensaje de error en pantalla.

3. Los operadores de comparación ==, =!, <= y >= han de escribirse sin espacios.

Cláusula ELSE

Una estructura de control condicional `if`, cuando incluye la cláusula `else`, permite ejecutar un bloque de código si se cumple la condición y otro bloque de código diferente si la condición no se cumple. Su sintaxis es la siguiente:

```
if(condición){
      código 1
   else
      código 2
}
```

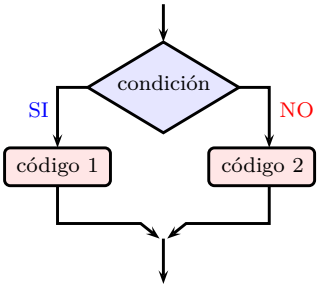

Figura 1.2: Cláusula ELSE

Los bloques de código especificados representan dos alternativas complementarias y excluyentes.

Estructura WHILE

La estructura de control de repetición `while`, repite un bloque de instrucciones mientras se cumple la condición. Su sintaxis es la siguiente:

```
while(condición){
   .....
   código
   .....
}
```

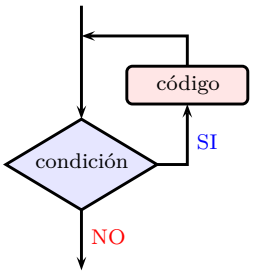

Figura 1.3: Estructura IF

1.2. Funciones primitivas

Existe un excepción para la regla que todas las funciones tienen 3 componentes; las funciones primitivas tales como: `any`, `sum`, `for`, etc. se encuentran construidas en `C`, `C++` o `Phyton`, por lo cual no cumple con lo antes mencionado.

```
# Funcion any
any

## function (..., na.rm = FALSE)  .Primitive("any")

body(any)

## NULL

formals(any)

## NULL

environment(any)

## NULL
```

En el caso que el usuario desee conocer todas las funciones primitivas del paquete `base`, lo puede hacer por medio de la siguiente línea de comando:

```r
ls("package:base")
```

Las funciones primitivas son creadas únicamente con el R Core Team, y en la actualidad se hace lo posible para no crear más funciones primitivas a excepción que no exista otra opción.

1.3. Funciones genéricas

Hemos observado que una gran mayoría de las funciones usadas hasta el momento se encuentran albergadas dentro de los paquetes base.

1.4. Lexical Scoping

1.4.1. Name Masking

1.4.2. Function vs Variables

1.4.3. Fresh Start

1.4.4. Dynamic Lookup