Funciones

Jorge Meneses y Paulo Peña

¿Por qué crear funciones?

¡Las funciones nos sirven para ahorrar tiempo y espacio de codificación!

Cuándo crear funciones? Digamos que quieras realizar una operación compleja que debes repetir más de una vez (digamos 100 veces): crear un mismo gráfico para diferentes variables, o realizar una misma transformación para diferentes datos. Aquí es donde crear tu propia función resulta útil y económico, tanto en tiempo como espacio.

Partes de una función

Como este es un taller práctico, entenderemos las partes de una función a través de un ejemplo. Crearemos nuestra propia función de varianza.

La varianza es la desviación estándar de una muestra de datos elevada al cuadrado y la desviación estándar es la distancia promedio de cada elemento en una muestra con respecto a su media. La siguiente formula la describe.

$$S^2 = \frac{\sum_{i=1}^{n} (X_j - \overline{X})^2}{n-1}$$

El nombre y argumentos

Primero se le designa un **nombre.** Con este nombre podremos utilizarla luego en nuestrios guiones. Es importante que este nombre sea claro, no ambiguo y que otra función no tenga el mismo nombre.

Por su parte, los **argumentos** son las variables que nuestra función requerirá para operar. Estas van entre paréntesis. Una función puede tener más de un argumento.

```
varianza <- function(variable){

}
```

El cuerpo

Esta es la parte que va entre llaves. Son todas las operaciones que realizará nuestra función. En el caso de nuestra función sería la fórmula de desviación estándar elevada al cuadrado, es decir la suma de la distancia de cada punto hacia la media, elevada al cuadrado y todo este resultado dividido por la cantidad de datos menos uno (n-1).

```
varianza <- function(variable){
  sum((variable-mean(variable))^2) /
    (length(variable) - 1)
}
```

Podemos separar las operaciones por partes

```
varianza <- function(variable){
  a <- sum((variable-mean(variable))^2)
  b <- length(variable) - 1

  return (a / b)
}

# con el comando return, nos aseguramos que la función nos devuelva solo el resultado final
```

Comando if

Regularmente, las operaciones que realizamos son complejas y depende de una cadena de resultados. Los comando if e ifelse nos permiten plasmar esta cadane de condiciones.

Trabajemos con un ejemplo de clasificar notas de un estudiante. Crearemos un vector con tres datos que representan las notas de un estudiante en los controles de lectura de un curso.

```
# Creemos el vector de notas de un alumno

alumno_1 <- c(15, 10, 8)

calificacion <- function(notas){
  if(mean(notas) > 10){"Aprobado"} # la condición a evaluar
  else{"Desaprobado"} # el resultado si la condición no se cumple
}

# Ejecutamos la función

calificacion(alumno_1)
```

```
## [1] "Aprobado"
```

Comando ifelse

Imaginemos que queremos evaluar todos los elementos del vector en vez de solo el conjunto como hemos visto. Acá es donde el comando ifelse, logra este objetivo. Probemos con las notas de toda una clase.

```
# Creemos el vector de notas del salón

notas_salon <- c(10, 8, 15, 10, 8)

calificacion_salon <- function(notas){
  ifelse(notas > 10,
  "Aprobado",  # la condición a evaluar
  "Desaprobado") # el resultado si la condición no se cumple
}

# Ejecutamos la función

calificacion_salon(notas_salon)
```

```
## [1] "Desaprobado" "Desaprobado" "Aprobado"    "Desaprobado" "Desaprobado"
```