**Creación de funciones en R**

**Estructuras de programación**

R permite crear estructuras repetitivas (loops) y la ejecución condicional de sentencias.  
A este fin, los comandos pueden agruparse entre llaves, utilizando la siguiente sintaxis:

**{comando1 ; comando2; comando3 ; ....}**

**El bucle for**

Para crear un bucle repetitivo (un bucle for), la sintaxis es la siguiente:

**for (i in listadevalores)  { secuencia de comandos }**

Por ejemplo:

**> for(i in 1:10) { print(i)}**  
**[1] 1**  
**[1] 2**  
**[1] 3**  
**[1] 4**  
**[1] 5**  
**[1] 6**  
**[1] 7**  
**[1] 8**  
**[1] 9**  
**[1] 10**

Un ejemplo de dibujo:

**> x = seq(-10,10)**  
**> plot(x,x,xlim=c(0,10),ylim=c(0,10))**  
**> for(i in 1:10)**  
**+ abline(h=i,col=i)**  
**> for(i in 1:10)**  
**+ abline(v=i,col=i)**

No obstante, los bucles for son lentos en R (y en Splus), y deben ser evitados en la medida de lo posible.

**El bucle while**

La sintaxis es como sigue:

**while ( condicion logica)  { expresiones a ejecutar }**

Por ejemplo, si queremos calcular qué número es el mayor cuyo cuadrado  
no excede de 1000, podemos hacer:

**> cuadrado = 0**  
**> while(cuadrado<=1000)**  
**+ {**  
**+ n<-n+1**  
**+ cuadrado<-n^2**  
**+ }**  
**> cuadrado**  
**[1] 1024**  
**> n**  
**[1] 32**  
**> 32^2**  
**[1] 1024**

¿Qué ha sucedido? El cuadrado de 32 excede 1000. En realidad, cuando n valía 31, su cuadrado (961)  
no excedía 1000, y el while() permitió entrar en el bucle, lo que hizo n=32. El número correcto sería  
en este caso n-1 = 31.

**Ejecución condicional: if**

La sintaxis general es:

**if (condicion) comando1 else comando2**

Por ejemplo, vamos a crear dos listas; una para guardar los  
números pares de 1 a 10, y otra para los impares:

**> n = 10           # Se inicializa n**  
**> pares = c()      # Se crea un vector vacío**  
**> impares = c()    # Idem**  
**> for(i in 1:n){   # Se van a procesar los números de 1 a n**  
**+ if(i%%2==0) pares<-c(pares,i)    # Si al dividir por 2 sale 0**  
**+ else impares<-c(impares,i)}     # el numero es par, impar en otro caso**  
**> pares**  
**[1]  2  4  6  8 10**  
**> impares**  
**[1] 1 3 5 7 9**

**Creación de funciones en R**

La estructura general de una función en R es la siguiente:

**nombre = function(argumento1 , argumento2, .....)   comandos**

Por ejemplo, podemos definir una función que calcule la desviación típica:

**> desv = function(x){sqrt(var(x))}    # Definimos la función**  
**> x<-1:10                            # Generamos datos**  
**> desv(x)                            # Utilizamos la función**  
**[1] 3.027650**  
**> sd(x)           # La definida en R coincide con la nuestra**  
**[1] 3.027650**

Una vez definida una función, se la puede llamar y utilizar como a cualquiera otra función  
predefinida en el sistema. Por ejemplo, vamos a utilizar la función apply combinada con  
desv para calcular las desviaciones típicas de las columnas de una matriz:

**> x = matrix(rnorm(15),nrow=3)**  
**> x**  
**[,1]       [,2]       [,3]       [,4]       [,5]**  
**[1,] 0.1578703  1.6712974 -0.5419452 0.03345786 -0.6675674**  
**[2,] 0.3215741 -0.6352143 -1.0222260 0.39006069  0.3609624**  
**[3,] 0.4770036 -0.3508383 -0.5147970 1.36219826 -1.6669992**  
**> apply(x,2,desv)**  
**[1] 0.1595845 1.2576365 0.2854502 0.6877219 1.0140156**

**Alcance de las variables**

Las variables definidas dentro del cuerpo de una función son locales, y desaparecen al terminar  
la ejecución de la función. Por ejemplo:

**> y = 10           # Definimos la variable y**  
**> cuadrado = function(x){ y <- x^2 ;  return(y)} # Definimos otra y local**  
**> x = 2             # Asignamos valor a x**  
**> cuadrado(x)      # Calculamos el cuadrado de x : Se hace y=4 (localmente)**  
**[1] 4**  
**> y                # Sin embargo, y no ha cambiado. La y local desaparece**  
**[1] 10**

**Parámetros por defecto**

Una función puede tener varios argumentos, y podríamos querer omitir especificar  
algunos de ellos, asumiendo que la función tomará por defecto unos valores  
preespecificados.

Como ejemplo, vamos a redefinir la función desviación típica, de modo que tengamos  
la posibilidad de calcular la desviación típica corregida y sin corregir:

**> desv = function(x,n=length(x)-1){ sum((x-mean(x))^2)/n}  # Definición de**  
**# la función**  
**> x<-1:10                              # Generación de un conjunto de datos**  
**> desv(x)               # Desviación típica corregida (al no especificar el**  
**# segundo parámetro, se divide por n-1**  
**[1] 9.166667**  
**> desv(x,10)            # Desviación típica sin corregir**  
**[1] 8.25**

**Funciones con un número variable de argumentos**

En R es posible definir funciones con un número variable de argumentos. Para ello, la sintaxis es:

**f = function(x, ...) { cuerpo de la función }**

**f = function(...,x) { cuerpo de la función }**

En el primer caso, la función podría llamarse sin hacer referencia explícita a x (por ejemplo  f(2) ).  
En el segundo caso deberíamos especificar   f(x=2), dado que el sistema, al encontrar primero los  
argumentos variables, no podría saber si nos estamos refiriendo a x o a uno de los argumentos  
variables.

Vamos a poner un ejemplo en dos fases. En primer lugar, para entender como funciona al tema,  
definiremos una función que simplemente devuelve sus argumentos:

**> f = function(...){ L <- list(...) ; return(L)}**  
**> f(1,2,3)**  
**[[1]]**  
**[1] 1**  
**[[2]]**  
**[1] 2**  
**[[3]]**  
**[1] 3**

**> f(c(1,2),c(3,4,5))**  
**[[1]]**  
**[1] 1 2**  
**[[2]]**  
**[1] 3 4 5**

Así pues, es variable el número de argumentos, tanto como el número de elementos de cada uno.

Vamos a aprovechar esta facilidad para definir una función que devuelva algunas medidas resumen  
de las distribuciones que se le pasen como argumento. La entrada a la función será una serie de  
conjuntos de datos, y la salida la media, varianza, mínimo y máximo de cada uno de los conjuntos.

**f = function(...)**  
**{**  
**datos = list(...)**  
**medias = lapply(datos,mean)    # lapply aplica una función sobre una lista**  
**varianzas = lapply(datos,var)**  
**maximos = lapply(datos,max)**  
**minimos = lapply(datos,min)**

**for(i in 1:length(datos))**  
**{**  
**cat("Distribución ",i,": \n")   # La función cat es para visualizar cosas**  
**cat("media: ",medias[[i]],"varianza: ",varianzas[[i]],"maximo: ",maximos[[i]],"minimo: ",minimos[[i]],"\n")**  
**cat("------------------------------------------------\n")**  
**}**

**}**

Veamos un ejemplo sencillo:

**>  f(c(1,2),c(1,3,5,7),c(-1,2,-5,6,9))**

**Distribución  1 :**  
**media:  1.5 varianza:  0.5 maximo:  2 minimo:  1**  
**------------------------------------------------**  
**Distribución  2 :**  
**media:  4 varianza:  6.666667 maximo:  7 minimo:  1**  
**------------------------------------------------**  
**Distribución  3 :**  
**media:  2.2 varianza:  30.7 maximo:  9 minimo:  -5**  
**------------------------------------------------**

O también:

**> x = rnorm(100)**  
**> y = runif(50)**  
**> f(x,y)**

**Distribución  1 :**  
**media:  0.1616148 varianza:  0.87319 maximo:  2.201592 minimo:  -2.143932**  
**------------------------------------------------**  
**Distribución  2 :**  
**media:  0.4985783 varianza:  0.08253697 maximo:  0.9881924 minimo:  0.01329678.**

**BIBLIOGRAFIA:** [**http://ocw.uc3m.es/estadistica/aprendizaje-del-software-estadistico-r-un-entorno-para-simulacion-y-computacion-estadistica/algunas-estructuras-de-programacion-creacion-de-funciones-en-r**](http://ocw.uc3m.es/estadistica/aprendizaje-del-software-estadistico-r-un-entorno-para-simulacion-y-computacion-estadistica/algunas-estructuras-de-programacion-creacion-de-funciones-en-r)