

Práctico 11

Funciones y Loops en R

Funciones

Definición de funciones en R

El lenguaje R hace uso de múltiples funciones: las mismas se encuentran pre-instaladas por defecto, o se pueden descargar de librerías específicas programadas para el trabajo en diversas áreas. En el práctico de la semana pasada, por ejemplo, se trabajó con funciones de la librería *seqinr*, las cuales facilitan el trabajo con secuencias biológicas en este lenguaje.

El usuario de R puede a su vez definir sus propias funciones. Esto es de gran utilidad: si se define una función que realiza una función dada, no es necesario reescribir el código que lleva a cabo este conjunto de acciones cada vez que se quiere realizar las mismas sobre diferentes variables.

Una función queda definida en R por dos elementos principales: el conjunto de elementos sobre los que opera (sus **argumentos**), y el **código** en el cual se especifica el conjunto de acciones que realiza dicha función. Al igual que en el caso de loops, el código de una función queda delimitado a través del uso de llaves.

A continuación se muestra un código simple para ilustrar estos conceptos, donde se define una función sencilla. La misma toma un número y devuelve el resultado de multiplicar el mismo por cuatro.

```
# Se define la funcion multiplica_por_cuatro, la cual tiene como argumento un numero.
multiplica_por_cuatro = function(numero){
# se guarda el resultado de multiplicar a la variable numero por 4
  resultado <- numero * 4
# se devuelve al usuario el resultado
  return(resultado)
}
```

```
# se utiliza la funcion con diferentes numeros, y se guardan los resultados en
# variables
```

```
resultado_1 = multiplica_por_cuatro(numero = 1)
resultado_3 = multiplica_por_cuatro(numero = 3)
```

```
resultado_1
```

```
## [1] 4
```

```
resultado_3
```

```
## [1] 12
```

Algo a destacar es que los argumentos de una función son representados por variables (en este caso la variable *numero*, las cuales son empleadas para representar a estos argumentos en el código de la función.

En el caso anterior la función *multiplica_por_cuatro* siempre devuelve el resultado de multiplicar el argumento *numero* por cuatro. A continuación se muestra una función más flexible, donde el usuario puede pasar como argumento el número por el cual quiere multiplicar a otro número

```

# se define una nueva funcion, que toma como argumentos dos numeros y los multiplica.
multiplica = function(numero_1, numero_2){
# se guarda el resultado de multiplicar ambos numeros
resultado <- numero_1 * numero_2
# se devuelve al usuario el resultado
return(resultado)
}

# se utiliza la funcion con diferentes numeros, y se guardan los resultados en
# variables
resultado_1 = multiplica(numero_1 = 2, numero_2 = 5)
resultado_2 = multiplica(numero_1 = 3, numero_2 = 17)

resultado_1

```

```
## [1] 10
```

```
resultado_2
```

```
## [1] 51
```

Ejercicio 1

Defina funciones que realicen las siguientes tareas:

a) Tome como argumento a un nombre e imprima a pantalla un saludo

b) Dado un número, compute su raíz cuadrada.

Nota: En caso de que el argumento no sea numerico, devuelva al usuario un mensaje.

c) Dado un número, compute su raíz cuadrada. Si el número no es entero, redondee primero y luego devuelva la raíz

Nota: puede usar la función round().

d) Dados dos vectores, uno con nombres y otro con números de teléfono, se devuelva un data.frame con los mismos. El mismo deberá indexar a dichas personas con números (ver ejemplo a continuación). Considere, ademas, que se realice la operacion si poseen igual largo los vectores.

Ejercicio 2

a) Defina una función que, dado un conjunto de números, devuelva el mayor de los mismos.

b) Cree una función que, dada una palabra, de la posición de sus letras en un diccionario. La palabra “vaca” da como resultado, entonces, el vector [22, 1, 3, 1]

Nota: utilice el vector **letters** (que se encuentra por defecto en R) y la funcion s2c() de la libreria seqinr.

c) Defina una función que, dada una palabra, la devuelva en mayúscula. Llame a la función creada en b) para esto, y el vector LETTERS.

Loops

La estructura de un *for loop* en R es similar a la encontrada en otros lenguajes (como Bash): *i*) se define un conjunto de elementos sobre los cuales se van a realizar acciones, *ii*) se define una variable con la cual se hace referencia a estos elementos y *iii*) se escriben las acciones a realizarse en el loop, las cuales están delimitadas en un bloque.

A continuación se muestra un *for loop* sencillo. En el mismo se toma un conjunto de números (*i.e.* los números del 1 al 5), se les suma un número y se guarda el resultado en un vector previamente definido.

```
# Se define el vector en el que se depositaran los resultados
resultados <- c()

# Se declara que se realizara el for loop sobre los elementos del vector 1:5
#(números del 1 al 5).

# A su vez se define la variable i, que representara a estos números
# en el loop
for (i in 1:5) {
  # Al número i se le suma 3, y se deposita el resultado en el
  # i-esimo elemento del vector resultados
  resultados[i] <- i + 3
}
```

En este caso, al comenzar el loop la variable *i* tomará en primer lugar el valor de 1. El resultado de sumar a este número 3 (lo cual se realiza en el bloque de acciones a realizarse en el loop) es guardado en el *i*-ésimo elemento del vector *resultados* (es decir, su primer elemento).

Algo a destacar es el uso de las llaves (*i.e.* `{ }`) para delimitar el loop: estos cumplen un rol análogo al que cumplen las palabras **do** y **done** en el lenguaje Bash, ayudando a delimitar el bloque de acciones que se ejecutará durante el loop.

Otra forma de escribir el loop descrito arriba sería la siguiente:

```
# Se define el vector en el que se depositaran los resultados
resultados <- c()
# Se define el vector de números del 1 al 5
numeros <- 1:5
# Se declara que se realizara el for loop sobre los elementos
# del vector numeros.
# A su vez se define la variable i, que representara a estos números
# en el loop
for (i in 1:length(numeros)) {
  # Al número i se le suma 3, y se deposita el resultado en el
  # i-esimo elemento del vector resultados
  resultados[i] <- numeros[i] + 3
}
```

En esta versión no se hace referencia directa a los números, si no que se realizan operaciones sobre los mismos a partir de los índices que tienen en el vector *numeros* (definido de forma previa al loop). La variable *i* toma, entonces, los valores desde 1 a `length(numeros)` (el largo del vector *numeros*), recorriendo así los índices de los elementos que componen el vector. Estos índices son usados para llamar a los elementos del mismo, lo cual se ve en el uso de la expresión `numeros[i]`.

Ejercicio 1

- a) Elija alguna de las funciones que definió en la primer sección del práctico. Aplique la misma sobre un conjunto de valores.
- b) Modifique este loop para almacenar los resultados en otro vector.
- c) Realice la misma operación, pero valiendose de una función del tipo `*apply()`.

Ejercicio 2

En el archivo **pedidoMesaUno.lista** se encuentra una tabla que muestra el pedido que hizo un conjunto de personas en un bar montevideano.

Los precios de los productos se encuentran en la tabla alojada en el archivo **carta.lista**.

Utilizando un loop, obtenga:

- a) Un data frame que represente a la cuenta. En la misma se debe detallar lo que debe pagar cada cliente.
- b) Una variable en donde se aloje el total de la cuenta a pagar por la mesa.

Nota: en caso de leer los archivos mencionados con la función `read.table()`, introduzca como argumento **stringsAsFactors = FALSE**. De lo contrario se leerán las columnas de la tabla como factores.