UNIVERSIDAD DE EL SALVADOR
FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE
DEPARTAMENTO DE MATEMÁTICA

Licenciatura en Estadística

Control Estadistico del Paquete R

"UNIDAD UNO"

Alumna:
Martha Yoana Medina Sánchez

Fecha de elaboración
Santa Ana - 27 de noviembre de 2015

UNIDAD 1: Práctica 05-Estructuras de control y definición de función en R.

R es un lenguaje de expresiones, en el sentido de que el único tipo de orden que posee es una función o expresión que devuelve un resultado. Incluso una asignación es una expresión, cuyo resultado es el valor asignado y que puede utilizarse en cualquier sitio en que pueda utilizarse una expresión.

Las órdenes pueden agruparse entre llaves, {expr_1;. . .; expr_m}, en cuyo caso el valor del grupo es el resultado de la última expresión del grupo que se haya evaluado. Puesto que un grupo es por sí mismo una expresión, puede incluirse entre paréntesis y ser utilizado como parte de una expresión mayor. Este proceso puede repetirse si se considera necesario.

Las estructuras de control en R son muy similares a las de cualquier lenguaje de programación.

1. ESTRUCTURA CONDICIONAL: LA ORDEN IF() Y IFELSE().

La construcción condicional if(), la cual es la más fácil de utilizar tiene alguna de las siguientes formas:

- if(condicion) expr
- if(condicion) expresion1 else expresion2

Donde **condiciones** una expresión que debe producir un valor lógico, y si éste es verdadero, TRUE ó T, se evalúa expresion1, si es falso, FALSO ó F, y se ha escrito la opción else, que es opcional, se ejecutará expresion2.

Si la expresion1 ó expresion2 son complejas, esto es, tienen más de un comando entonces deben encerrarse entre llaves {. . . }

A menudo suelen utilizarse los operadores && (AND) y —— (OR) en una condicion. En tanto que & y —se aplican a todos los elementos de un vector, && y —— se aplican a vectores de longitud uno y sólo evalúan el segundo argumento si es necesario, esto es, si el valor de la condicion completa no se deduce del primer argumento.

ifelse(prueba, si, no)

Donde:

- prueba: Es un vector lógico o condición lógica a ser evaluada.
- si: devuelve valores para los elementos ciertos de **prueba**.
- no: devuelve valores para los elementos falsos de **prueba**.

El uso de if() está limitado a expresiones que no sean vectores. Si estamos evaluando vectores o matrices entonces lo indicado es hacerlo con ifelse() que devuelve un valor con la misma forma que el argumento **prueba** el cual es llenado con elementos seleccionados bien sea del argumento **si** o del argumento **no** dependiendo de si el elemento de **prueba** es **TRUE** o **FALSE**, si los argumentos **si** o **no** son muy cortos, entonces sus elementos son reciclados.

Por ejemplo, ejecute las siguientes instrucciones

```
x <- c(6:-4);
x

##  [1]  6  5  4  3  2  1  0 -1 -2 -3 -4

sqrt(x) # Produce un mensaje de advertencia

## Warning in sqrt(x): Se han producido NaNs

##  [1] 2.449490 2.236068 2.000000 1.732051 1.414214 1.000000 0.000000
##  [8]      NaN      NaN      NaN      NaN

sqrt(ifelse(x >= 0, x, NA)) # No produce advertencia

##  [1] 2.449490 2.236068 2.000000 1.732051 1.414214 1.000000 0.000000
##  [8]       NA       NA       NA       NA

ifelse(x >= 0, sqrt(x), NA) # Produce un mensaje de advertencia

## Warning in sqrt(x): Se han producido NaNs

##  [1] 2.449490 2.236068 2.000000 1.732051 1.414214 1.000000 0.000000
##  [8]       NA       NA       NA       NA

# Comente las diferencias entre cada una de las instrucciones anteriores.
```

2. ESTRUCTURAS ITERATIVAS O DE REPETICIÓN: FOR(), WHILE() Y REPEAT()

La función for() es una construcción repetitiva que tiene la forma:

for(nombre in expr1) expr2

Donde **nombre** es la variable de control del número de iteraciones, expr1 es un vector (a menudo de la forma m:n), y expr2 es una expresión, a menudo agrupada, en cuyas sub-expresiones puede aparecer la variable de control, expr2 se evalúa repetidamente conforme nombre recorre los valores del vector expr1.

- Ejemplo:

```r
x <- c(2, 6, 4, 7, 5, 1)
suma<-0; for(i in 1:3) suma = suma+x[i]; suma

## [1] 12
```

Nota: En R, la función for() se utiliza mucho menos que en lenguajes tradicionales, ya que no aprovecha las estructuras de los objetos. El código que trabaja directamente con las estructuras completas suele ser más claro y más rápido.

Otras estructuras de repetición son:

- while (condición) expresión
- repeat expresión

La función break() se utiliza para terminar cualquier ciclo. Esta es la única forma (salvo que se produzca un error) de finalizar un ciclo repea. La función next() deja de ejecutar el resto de un ciclo y pasa a ejecutar el siguiente.

FUNCIONES ESCRITAS POR EL USUARIO

El lenguaje R permite al usuario definir objetos que sean funciones. Éstas se convierten en auténticas funciones de R, que se almacenan en una forma interna y se pueden utilizar en expresiones futuras.

Los argumentos pueden ser objetos (datos, fórmulas, expresiones, . . .), algunos de los cuales pueden ser definidos por defecto en la función; sin embargo, estos argumentos pueden ser modificados por el usuario con opciones. Una función en R puede carecer totalmente de argumentos, ya sea porque todos están definidos por defecto (y sus valores son modificados con opciones), o porque la función realmente no utiliza argumentos.

Una función se define por una asignación de la forma

$$\begin{array}{c} \text{nombreFunción} <- \text{function(arg1, arg2, . . .) expresión} \\ \text{return(valor)} \end{array}$$

Donde: arg1, arg2, . . . : son los argumentos de la funció u opciones del tipo opcion=expresión, una puede no tener argumentos.

Expresión: es una expresión en R, si ocupa más de una instrucción estas van encerradas entre llaves , y utiliza los argumentos para calcular su valor. El valor de la expresión es devuelto como el valor de la función por medio del nombre, o puede utilizar return() para retornar uno o más valores.

valor: es una expresión o una serie de expresiones separadas por comas.

- Ejemplo 1: Definir en R la función cuadrática y = f (x) = $(3x^2) - (5x) + 2$
  Como nombre de la función podemos usar cualquier palabra (que no sea una palabra reservada por R, como log o sum) que puede incluir letras y puntos.

  Llamémosle func.cuadratica y definámos la de la manera siguiente:

```
func.cuadratica <- function(x)
{
3*x^2-5*x+2
}
#Luego, si queremos calcular f(2) simplemente ejecutamos la instruccion:
y <- func.cuadratica(2);y

## [1] 4
```

NOTA: Toda función para usarla debe estar cargada en el área de trabajo (Workspace). Es decir, primero es necesario correr el código necesario el código de la función y asegurarse que no contenga errores de sintaxis.

- Ejemplo 2: Se quiere definir una función para calcular la media de un vector de datos

Una definición podría ser:

```
media <- function(x)
{

  n = length(x)
suma <- 0.0
for(i in 1:n) suma = suma + x[i]
media = suma/n

}
save(media, file= "media.RData")
rm(list=ls(all=TRUE))
load("media.RData")

x <- 1:5;
(media(x)) # Se usa doble parentesis para que muestre el resultado en pantalla

## [1] 3

y <- c(5, NA , 4, 9);
(media(y)) # El resultado no puede calcularse pues falta un dato

## [1] NA

z <- c(5, 1 , 4, 9);
(media(z))
```

```
## [1] 4.75

(media) # Nos muestra el codigo de la funcion

## function(x)
## {
##
##   n = length(x)
## suma <- 0.0
## for(i in 1:n) suma = suma + x[i]
## media = suma/n
##
## }
```

Note que al escribir (media), nos muestra el código de la función.

Observe el problema que se da en el cálculo de la media, debido a los datos omitidos o perdidos, qué propone usted para solucionar esto.

- Ejemplo 3: Se quiere definir una función para graficar la función seno de x

Una definición de esta función puede ser:

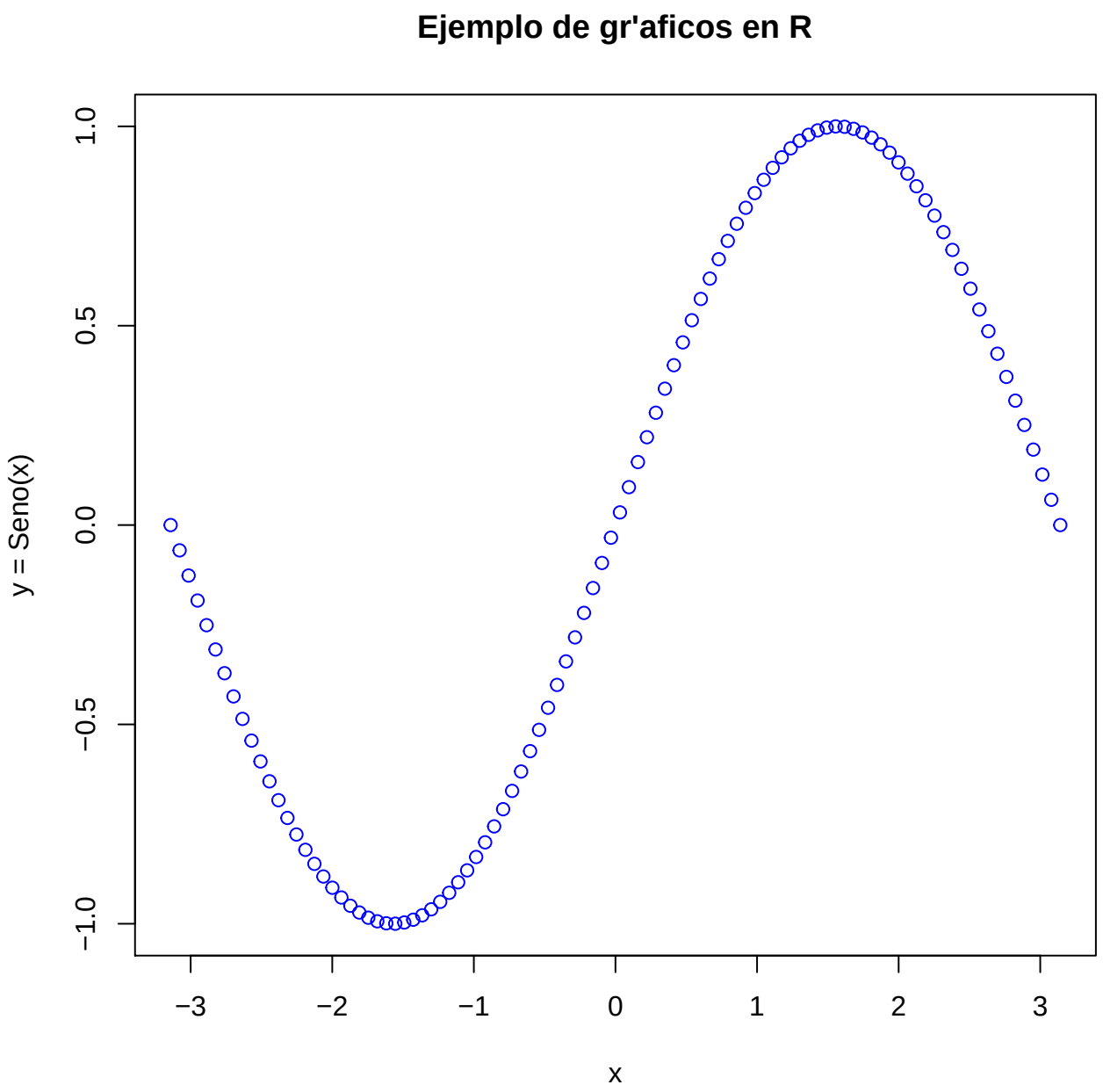
```
Seno <- function(x)
{

  y = sin(x)
plot(x, y, main="Ejemplo de gr\'aficos en R",
xlab="x", ylab="y = Seno(x)", col="blue", pch=1)

}
# Pruebe la funcion con el siguiente vector:

x<-seq(-pi, pi, len=100)

Seno(x)
```


Ejemplo de gr'aficos en R
y = Seno(x)
x
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3

EJERCICIOS

- Ejercicio 1: Escriba una función para encontrar el factorial de un número mayor que cero

```
fac<-function(f){ prod<-1 # Inicializar el producto en 1
if (f==0){ # Cuando el valor ingresado es cero
  prod<-1 # El factorial es 1
  return(prod)
}
  else{
    if(f<0) # Cuando el valor ingresado es negativo
      print("No existe el factorial de un n?mero negativo")
    else {
      int<-c(1:f) # Cuando el valor es positivo

      for(i in int)
        prod<-prod * i
      return(prod)
    }
  }

  }
fac(4)

## [1] 24
```

- Ejercicio 2: Escriba una función para encontrar la varianza o la cuasi-varianza de un vector de datos.

```
vx<-function(k) { suma <- 0.0
  z<-length(k)
  for(i in 1:z){
    suma = suma + k[i]
    media = suma/z # Obtener la media aritmetica del vector
    for(i in 1:z){
      vx<-k[i]-media
      vx<-(vx)/z
    }
  }
  return(vx)
}
k <- c(2,3,4)
(vx(k))

## [1] 0.3333333
```

- Ejercicio 3: Escriba una función para encontrar la media geomótrica de un vector de datos

```
# Obtener la ra\'iz n-esima de cualquier valor
raiz=function(m,n){ # Este es la funcion que llamamos en los dos c\'odigos siguientes
  raiz=n^(1/m)
  return(raiz)
}
raiz(3,27)

## [1] 3

# Caso cuando el vector esta ordenado, es decir desde 1 hasta el valor deseado
MG<-function(m)
  {prod<-1 # Este c?digo se parece al del factorial de un numero positivo
  int<-c(1:m)

  for(i in int)
    prod<-prod * i
  raiz=raiz(m,prod)
  # Obtener la ra\'iz n-esima (corresponde a la cantidad de valores en el vector)
  return(raiz)
  }
MG(5)

## [1] 2.605171

# Caso cuando el vector es de calquier forma que el usuario desee
MG2<-function(g) {product<-1 # Inicializar el producto
  p<- length(g) # Guardar la longitud del vector
  for(i in 1:p)
    product<-product * g[i] # Realizar el producto de cada valor del vector
  MG2<-product # Guardar el producto en MG2
  raiz<-raiz(length(g),MG2)
  # Obtener la raiz n-esima (corresponde a la cantidad de valores en el vector)
  return(raiz)
}
g<-c(2,3,4,5) # Pasar el vector de datos
(MG2(g))

## [1] 3.309751
```

- Ejercicio 4: Escriba una función para encontrar la media armónica de un vector de datos

```
MA <- function(x)
  {suma <- 0.0 # Inicializar suma
  n <- length(x) # Guardar la longitud del vector
  for(i in 1:n)
    suma <- suma + (1/x[i])
  # Realizar la suma, pero de los reciprocos (1/x[i]) de los valores del vector
  denom <- suma/n # Encontrar la media de los reciprocos
  MA<-1/denom

}
x <- c(2,3)
# Pasar el vector de datos
(MA(x))

## [1] 2.4
```