

# Complementos

## Sesión 6

Natalie Julian - [www.nataliejulian.com](http://www.nataliejulian.com)

Estadística UC y Data Scientist en Zippedi Inc.



A la hora de crear funciones, existen distintas sintaxis ampliamente utilizadas en todos los lenguajes de programación, conocer dichas sintaxis es de mucha utilidad para desarrollar de manera óptima distintos procesos o procedimientos de interés.

# Entendiendo una función

Toda función en R se compone de tres partes:

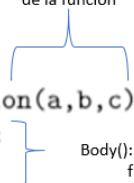
- `body()`: El código dentro de la función
- `formals()`: La lista de argumentos que utiliza la función
- `environment()`: Ubicación de la función

# Ejemplo

Formals(): Argumentos de la función

```
new.function <- function(a,b,c) {  
  result <- a * b + c  
  print(result)  
}
```

Body(): Cuerpo de la función



```
body(new.function)  
{  
  result <- a * b + c  
  print(result)  
}
```

```
formals(new.function)  
$a  
$b  
$c
```

```
environment(new.function)  
<environment: R_GlobalEnv>
```

# Proceso iterativo finito: for()

Muchas veces necesitamos replicar una o varias acciones. Una opción es utilizar `for()` el cual debe contener un índice que debe especificarse en un rango determinado. Al entrar en el proceso, se genera un loop, se debe utilizar estratégicamente el índice definido, pues este es el que cambia en cada iteración.

*Observación:* Se sugiere utilizar `for()` sólo cuando sea estrictamente necesario pues puede resultar muy costoso en término de operaciones y procesamiento. Las funciones que pueden ayudarnos a no usar `for` son `apply()`, `sapply()`, entre otras.

## Ejemplo

```
x<-c(25.3, 27.9, 20.6, 21.4, 67.9)
y<-c(48.1, 33.6, 52.1, 56.7, 24.3)

matriz<-cbind(x,y)

#Quiero obtener los promedios por fila

promediosfila<-function(matriz){
  vector<-rep(0, nrow(matriz)) #Vector que ser llenado por posicion

  for(i in 1:nrow(matriz)){ #i sera el indice que va de 1:numero de filas

    #La posicion i del vector se llenara con el promedio por fila
    vector[i]<-mean(matriz[i,])
  }
  return(vector) #Se retorna el vector
}

promediosfila(matriz)
[1] 36.70 30.75 36.35 39.05 46.10

#Sin for:
apply(matriz, MARGIN=1, FUN=mean)
[1] 36.70 30.75 36.35 39.05 46.10
```

# Proceso sujeto a condición: while()

También es posible ejecutar un código dependiendo de si se cumple determinada condición, `while()` permite realizar esto. En este ejemplo, se muestra otra manera de plantear el proceso anterior, pero definiendo un contador y replicando el proceso hasta que el contador no supere determinado valor.

## Ejemplo

```
promediosfilawhile<-function(matriz){  
  vector<-rep(0, nrow(matriz))  
  
  i<-1 #contador  
  while(i<=nrow(matriz)){  
    vector[i]<-mean(matriz[i,])  
    i<-i+1 #Se debe aumetar el contador  
  }  
  return(vector)  
}  
  
promediosfilawhile(matriz)  
[1] 36.70 30.75 36.35 39.05 46.10
```

# Proceso ramificado por casos: if() - else

A veces es necesario definir por casos la función. Podría ser útil por ejemplo para las funciones por tramo, o cuando se requiere un distinto procedimiento dependiendo del caso.

En este ejemplo, crearemos una función que recodifica un vector en ciertos valores:

## Ejemplo

```
y<-c(0, 1, 2, 0, 1, 2, 1)
```

```
codifica<-function(y){  
  for(i in 1:length(y)){ #Se recorre todo el vector  
    if(y[i]==0){  
      y[i]<-"Ninguno"  
    }  
    else{  
      if(y[i]==1){  
        y[i]<-"Primera categoria"  
      }  
      else{  
        y[i]<-"Segunda categoria"  
      }  
    }  
  }  
  return(y)  
}
```

```
codifica(y)  
[1] "Ninguno"          "Primera categoria" "Segunda categoria"  
[4] "Ninguno"          "Primera categoria" "Segunda categoria"  
[7] "Primera categoria"
```

#Sin if-else:

```
ifelse(y==0, "Ninguno", ifelse(y==1, "Primera categoria", "Segunda categoria"))  
[1] "Ninguno"          "Primera categoria" "Segunda categoria"  
[4] "Ninguno"          "Primera categoria" "Segunda categoria"  
[7] "Primera categoria"
```

- La secuencia de Fibonacci es una sucesión definida por recurrencia. Esto significa que para calcular un término de la sucesión se necesitan los términos que le preceden.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

Cree una función que retorne el n-ésimo valor de la serie de Fibonacci.



# PRÁCTICA 1

Los gráficos es sin duda una parte muy importante a la hora de realizar análisis de datos. Con la función `plot()` podemos crear gráficos sencillos en R. Existen muchas librerías para crear gráficos, pero utilizaremos las funciones base de R.

# Funciones para realizar gráficos

- `plot()` Realiza gráfico de puntos del eje x e y
- `barplot()` Realiza gráfico de barras
- `boxplot()` Realiza un gráfico de caja y bigotes
- `hist()` Realiza un histograma
- `pie()` Realiza gráfico de torta

# Funciones para realizar gráficos

- `plot()` Realiza gráfico de puntos del eje x e y para variables numéricas
- `barplot()` Realiza gráfico de barras para variables numéricas discretas
- `boxplot()` Realiza un gráfico de caja y bigotes para variables numéricas
- `hist()` Realiza un histograma para variables numéricas continuas
- `pie()` Realiza gráfico de torta para variables categóricas o numéricas discretas

Usaremos estas funciones en las próximas sesiones!

# RESPUESTAS PRÁCTICA 1

# Con while

```
fib2 <- function(n){  
  valores <- c()  
  valores[[1]] <- 0  
  valores[[2]] <- 1  
  valores[[3]]<-1  
  
  i<-3   #Se define un contador apropiado  
  
  while (i <=n){           #Proceso condicional  
  
    next_val <- sum(tail(valores,2) )  
    valores <- c(valores, next_val)  
    i<-i+1  
  
  }  
  
  return(valores[n])  
}
```

```
fib2(4)  
[1] 2
```

```
fib2(10)  
[1] 34
```

# Opción novedosa pero más compleja

```
fib <- function(n) {  
  if (n == 1) { #Proceso Ramificado  
    return(0)  
  }  
  else if (n == 2) {  
    return(1)  
  }  
  else if (n > 2) {  
    return(fib(n - 1) + fib(n - 2)) #Proceso Autorecursivo  
  }  
}  
  
fib(4)  
[1] 2  
  
fib(10)  
[1] 34
```

# Think twice, code once

Hacer una función o un código, es simple, pero hacerlo óptimo y eficiente, es lo complejo. Siempre que realices un código, pregúntate, ¿es posible mejorarlo? ¿qué tanto se demora? ¿si tenemos muchos datos, se cae? Si tienes dudas en alguna pregunta, vuelve a pensarlo de nuevo!