```
# Taller: Rstudio Aplicado a Finanzas
# Colegio de Economistas de Loreto
# Profesor: Jose Rodney Menezes De la Cruz

#=========================================================================
# Sesión 2 "Introducción al uso de R project y Lenguaje de Programación"
#=========================================================================

#================================================
# I. Pedir ayuda
#================================================

help(lm)
help("[[")
help('if')
?lm
help.search("split")
??"split"

#================================================
# II. Los datos y sus tipos
#================================================

# 1. Los datos numericos

x <- 2 # Se asigna el valor 2 a x
print(x) # Se imprime el valor de x
class(x) # Muestra cuál es la clase de x

x <- 6/2 # Se asigna el valor de la operación dividir 6/2 a x
print(x)
class(x)

# 2. Los datos enteros

x <- 23L; print(x)
class(x)
x <- as.integer(6/2); print(x)
class(x)

# 3. Los datos complejos

x <- 21 + 2i
y <- 2i + 21 # El mismo valor que x
z <- -1 + 0i # Corresponde a -1
tt <- sqrt(z) # raíz cuadrada de -1
print(x); print(y); print(z); print(tt)
class(tt)

# 4. Los datos infinitos

x <- 1/0 # División por cero
x

y <- 1/Inf # Tambien dividir un número por Inf da cero:
y

# 5. Datos sin valor

x <- 0/0
x
```

```
#================================================
# III. Vectores
#================================================

# 1. El uso de la función c() para crear vectores

c(4,2,-8) # Creación de un vector sin asignarlo a una variable

## Distintas formas de asignar un vector a una variable
u <- c(4,2,-8) # Usando el operador <-
c(4, 2, -8) -> v # Usando el operador ->
assign("w", c(4, 2, -8)) # Usando la función assign
p = c(4, 2, -8) # Usando el operador =
print(u); print(v); print(w); print(p)

# 2. Creación de vectores a partir de secuencias y otros patrones

v <- vector("integer", 0)
v # Un vector de enteros sin elementos

w <- vector("numeric", 3)
w # Un vector de tres ceros

u <- vector("logical", 5)
u # Un vector de 5 FALSE

v <- 40:13
print(v); class(v)

v <- pi:6
print(v); class(v)

v <- seq(from = 5, to = 15, by = 2)
print(v) # secuencia desde 5 hasta 15 de 2 en 2
class(v)

v <- seq(from = 4, by = 2, length.out = 8)
print(v) # secuencia de 8 números iniciando desde 4 y de 2 en 2

v <- c(4, 8, -3)
w <- rep(v, times = 5)
print(w)

u <- c(3, 4, 5)
v <- c(5, 4, 3)
w <- c(u, v)
print(w) # La concatenación de u y v

# 3. Acceso a los elementos individuales de un vector

v <- c(8, 7, -3, 2, 182)
v[5] # El quinto elemento
print(v[1]); print(v[3])

v[4]+v[2] # La suma del cuarto y segundo elementos de v
v[1] <- v[2] - v[5]
v # Note que el resultado de la operación se ha guardado en v[1]

v[8] <- 213
v # v tiene ahora 8 elementos con espacios vacios: NA

frutas <- c(15, 100, 2, 30)
frutas
names(frutas) <- c("naranja", "pera", "manzana", "durazno")
```

```
frutas

frutas <- c(naranja = 15, pera = 100, manzana = 2, durazno = 30)
frutas

frutas["durazno"]
frutas["manzana"] <- 8
frutas
frutas[2]


#4. Operaciones sencillas con vectores

v <- 2 + 3 # Resulta en un vector de longitud 1
v

v <- c(2, 3) - c(5, 1) # Resulta en un vector de longitud 2
v

v <- c(2, 3, 4) * c(2, 1, 3) # Resulta en un vector de longitud 3
v

v <- c(2, 3, 4)^(3:1) # Eleva a potencias 3,2,1
v

u <- 2:33
v <- c(4, 5, 6)
w <- c(u, v)
w
length(w) #Loggitud de un vector

w <= 10 # Prueba elementos menores o iguales a 10

#Otros operadores logicos: <, >, >=, ==, y !=.

v <- c(4, 5, 6, 7, 8, 9, 10) * c(1, 2, 1, 2, 1, 2, 1)
v

v <- c(4, 5, 6, 7, 8, 9, 10) * c(1, 2)
v

options(warn = -1)
v <- c(4, 5, 6, 7, 8, 9, 10) * c(1, 2)
v

v <- c(2, -3, 4)
w <- 2 * (v^2) # Dos veces el cuadrado de v
w

v <- c(9, 8, 31)
sqrt(v) # Raiz cuadrada

#El sin de 30, 45 y 60 grados: Primero se hace la conversión a radianes:
angulos <- c(30, 45, 60) * (pi/180)
angulos # En radianes
senos <- sin(angulos)
senos
```

```
#================================================
# IV. Matrices
#================================================

# 1. Construccion de matrices

(m <- 11:30) # Un vector con 20 números
# Para convertirla en matriz simplemente se especifica el atributo dim
dim(m) <- c(4, 5) # 4 renglones y 5 columnas
m
class(m)

dim(m) <- c(5, 4) # ahora 5 renglones y 4 columnas
m

m[3, 2] #el elemento en el renglon 3 y columna 2
m[8] # acceso al mismo elemento, como vector, con un solo índice
m[3, ] # El renglón 3 de la matriz
m[, 2] # la columna 2 de la matriz
class(m[3, ])# La clase las columnas o renglones

(m <- matrix(11:30, nrow = 5, ncol = 4, byrow = TRUE))
rownames(m) <- c("uno", "dos", "tres", "cuatro", "cinco")
colnames(m) <- c("UNO", "DOS", "TRES", "CUATRO")
m

colnames(m) # Consulta de los nombres de las columnas
m[,"DOS"] # Una columna:
m["dos","UNO"]

m1<-rbind(c(1.5,3.2,-5.5),c(0,-1.1,60))
m1
class(m1[1, ])

m2<-cbind(c(1.5,3.2,-5.5),c(0,-1.1,60))
m2


#2. Operaciones con matrices

m<-matrix(1:15,nrow=5,ncol=3)
m
mm<-rbind(1:3,3:1,c(1,1,1),c(2,2,2),c(3,3,3))
mm
m*mm

A<-matrix(1:6,3,2)
A
B<-rbind(7:9,10:12)
B
A%*%B
t(A)


#================================================
# V. Factores y vectores de caracteres
#================================================

persona<-c("Hugo","Paco","Luis","Petra","Maria","Fulano",
           "Sutano","Perengano","Metano","Etano","Propano")
mes.nacimiento<-c("Dic","Feb","Oct","Mar","Feb","Nov",
                  "Abr","Dic","Feb","Oct","Dic")
persona
mes.nacimiento
```

```
print(persona[7]);print(mes.nacimiento[7])
print(persona[7]);print(mes.nacimiento[7])
paste(persona[7],"nacio en el mes de", mes.nacimiento[7])


#1. Los factores y su extructura

Fmes.nacimiento<-as.factor(mes.nacimiento)
Fmes.nacimiento
paste(persona[7],"nacio en el mes de", Fmes.nacimiento[7])
unclass(Fmes.nacimiento)

table(Fmes.nacimiento)

meses<-c("Ene","Feb","Mar","Abr","May","Jun","Jul","Ago",
         "Sep","Oct","Nov","Dic") # Se incluyen meses que no están el el vector
original
FFmes.nacimiento<-factor(mes.nacimiento,levels=meses)
FFmes.nacimiento
table(FFmes.nacimiento)


#2. Acceso a los elementos de un factor

Fmes.nacimiento[10] # Un elemento individual del factor:
levels(Fmes.nacimiento)[3] # Un elemento individual de los niveles:
levels(Fmes.nacimiento)[3]<-"febrero"
Fmes.nacimiento
as.integer(Fmes.nacimiento)

#================================================
# VI. Listas
#================================================

familia<-list("Maria","Juan",10,c("Hugo","Petra"),c(8,6))
familia

familia<-list(madre="Maria",padre="Juan",casados=10,
              hijos=c("Hugo","Petra"),edades=c(8,6))
familia


#1. Acceso a los elementos individuales de una lista

# Acceso de lectura
familia$madre
familia[["madre"]]

# Acceso de escritura
familia[["padre"]]<-"Juan Pedro"
familia$padre

familia$"madre"<-"Maria Candelaria"
mm<-"madre"
familia[[mm]]
familia[[paste("ma","dre",sep="") ]]
```

```
#================================================
# VII. Data frames
#================================================

m<-cbind(ord=1:3,edad=c(30L,26L,9L))
v<-c(1.80,1.72,1.05)

ff<-data.frame(familia=c("Padre","Madre","Hijo"),m,estatura=v)
ff

#================================================
# VIII. Funciones
#================================================


#1. Definicion

# Definición -- Versión 1
MiFunc.v1<-function(x,yyy,z=5,t) {
  w<-x+yyy+z
  w
}

# Definición -- Versión 2
MiFunc.v2<-function(x,yyy,z=5,t) {
  w<-x+yyy+z
  return(w)
  3.1416 # Este código nunca se ejecuta
}

# Definición -- Versión 3
MiFunc.v3<-function(x,yyy,z=5,t) {
  x+yyy+z
}


#2. Ejecuciones:

MiFunc.v1(1,2,3) # Ejecución 1
MiFunc.v2(1,2) # Ejecución 2
MiFunc.v3(1) # Ejecución 3
MiFunc.v3(z=3,yyy=2,x=1) # Ejecución 4
MiFunc.v2(1,y=2) # Ejecución 5


#3. Revision de argumentos

args(MiFunc.v2)
args(lm)


#4. El uso del argumento "..." para extender una función
NuevaMiFunc<-function(x,yyy=-1,...) {
  MiFunc.v3(x, yyy, ...)
}
NuevaMiFunc(1)
NuevaMiFunc(x=1)
NuevaMiFunc(1,z=10)

#5. Asociacion de simbolos con valores

f<-function(x) {
  2*x
}
```

```
t(8)

#6. Contenido de los ambientes de las funciones

Construye.multiplicador<-function(n) {
  fff<-function(x) {
    n*x
  }
  fff # Regresa como resultado la función creada
}

duplica<-Construye.multiplicador(2)
triplica<-Construye.multiplicador(3)

duplica(5)
triplica(5)

hipotenusa<-function(x,y) {
  sqrt(x^2+y^2)
}
class(hipotenusa)

hipotenusa<-function(x,y) {
  return(sqrt(x^2+y^2))
}

hipotenusa(3,4)
hipotenusa(y=4,x=3)

hipotenusa<-function(x=3,y=4) {
  return(sqrt( x^2+y^2) )
}
hipotenusa()

ff<-function(r) {
  return(PI*r^2)
}
PI<-3.1416
ff(3)

#================================================
# IX. El operador []
#================================================


#1. Seleccion de una secuencia de elementos

#P.ej. para un vector de 20 números aleatorios, generados
# con la función rnorm(), que genera números aleatorios con
# una distribución normal:
(v<-rnorm(20))
# Si queremos seleccionar de esos, sólo los números en las posiciones de la 5 a la
15:
(subv<-v[5:15])
class(v)
class(subv)

Fmes.nacimiento<-factor(c("Dic","Feb","Oct","Mar","Feb",
                          "Nov","Abr","Dic","Feb","Oct","Dic"),
                        levels=c("Ene","Feb","Mar","Abr","May","Jun",
                                 "Jul","Ago","Sep","Oct","Nov","Dic"))
```

```
sub.Fmes.nacimiento<-Fmes.nacimiento[2:5]
sub1.v<-v[c(2,3,5:8)]
sub1.Fmes.nacimiento<-Fmes.nacimiento[c(2,3,5:8)]


#2. Selección de elementos de acuerdo con una condición

v<0
v[v<0]# Los negativos
v[!(v<0)]# positivos con negacion lógica, con el operador !
v[v>=0]# positivos mediante el operador >=

Fmes.nacimiento=="Mar" # igualdad
Fmes.nacimiento!="Mar" # desigualdad

as.integer(Fmes.nacimiento)# El factor convertido a enteros:
as.integer(Fmes.nacimiento)<=4

#================================================
# X. Matrices y data frames
#================================================

mt<-matrix(11:30,nrow=4,ncol=5)
df.mt<-as.data.frame(mt)# Se convierte la matriz a un data frame

rownames(df.mt)<-c("uno","dos","tres","cuatro")
colnames(df.mt)<-c("UNO","DOS","TRES","CUATRO","CINCO")
df.mt

mt[5] # La matriz con índice 5
df.mt[5] # El data frame con índice 5
mt[3, ]# El tercer renglón
df.mt[3, ]# El tercer renglón
mt[,3]# La tercer columna:
df.mt[,3]

class(mt[3, ])
class(mt[,3])
class(df.mt[,3])
class(df.mt[3, ]) # sólo en este caso se cumple

mt[3, ,drop=FALSE] # El tercer renglón
class(mt[3, ,drop=FALSE])

df.mt[3, ,drop=FALSE]# El tercer renglón
class(df.mt[3, ,drop=FALSE])

mt[,3,drop=FALSE]# La tercer columna
class(mt[,3,drop=FALSE])

df.mt[,3,drop=FALSE] # La tercer columna
class(df.mt[,3,drop=FALSE])

mt[,4:2]# Selección de las columnas 4,3,2, en ese orden
df.mt[,4:2]# Selección de las columnas 4,3,2, en ese orden

mt[1:3,4:2]
df.mt[1:3,4:2]
```

```
#1. uso para indices logicos y condiciones

mt[2, ]
df.mt[2, ]
mt[,2]==16
df.mt[,2]==16

# Se usan paréntesis, (), para enfatizar la condición, aunque
# se podría prescindir de ellos:
mt[(mt[,2]==16), ]
df.mt[(df.mt[,2]==16), ]

# En el caso de la matriz, si se quiere obtener como salida
# una matriz (de un solo renglón), se hace así:
mt[(mt[,2]==16), ,drop=FALSE]

mt[4,2]<-16L
df.mt[4,2]<-16L
mt # (El data frame es semejante)
mt[(mt[,2]==16), ]
df.mt[(df.mt[,2]==16), ]

# La prueba lógica hace uso del operador módulo o residuo: %%
mt[2, ]%%8!=0 # (Para el data frame es semejante)
# Ahora usemos la expresión como índice:
mt[, (mt[2, ]%%8!=0)]

df.mt[, (df.mt[2, ]%%8!=0)]

#=================================================
# XI. Los operadores [[]] y $
#=================================================

rownames(mt)<-c("uno","dos","tres","cuatro")
colnames(mt)<-c("UNO","DOS","TRES","CUATRO","CINCO")
mt
mt[,"TRES"]
df.mt[,"TRES"]

mt[[2,3]]
mt[["dos","TRES"]]
class(mt[["dos","TRES"]])

df.mt[[2,3]]
class(df.mt[[2,3]])
df.mt[["dos","TRES"]]

df.mt[["TRES"]]
df.mt$TRES
df.mt$"TRES"
df.mt$T
df.mt[["TR",exact= F]]
```

```
#================================================
# XII. La construcciones IF-ELSE
#================================================

#1. if

aa<-15
if(aa>14) # if sin else
  print("SI MAYOR")

if(aa>14)print("SI MAYOR")

if(aa>14) { # Instrucción compuesta
  print("PRIMER RENGLON")
  print("SI MAYOR")
}


#2. if else

if(10>aa) { # 1er. bloque
  print("RANGO MENOR")
}else if(10<=aa&&aa<=20) { # 2o. bloque
  print("primer renglon");print("RANGO MEDIO")
}else{ # 3er. bloque
  print("RANGO MAYOR")
}

#================================================
# XIII. Los Ciclos
#================================================

#1. Repeticiones por un número determinado de veces

letras<-c("c","l","i","M","T","A")
for(i in 1:6) {
  print(letras[i])
}

for(i in seq_along(letras)) {
  print(letras[i])
}

for(letra in letras) {
  print(letra)
}

i<-1
while(i<=6) {
  print(letras[i])
  i<-i+1
}


#2. bucles infinitos o repeticiones infinitas

i<-1
repeat{
  print(letras[i])
  i<-i+1
  if(i>6)
    break
}
```

```
#3. Interrupciones del ???ujo normal de los ciclos

#Ejem 1:
# se usará un generador de números aleatorios,
# la siguiente función asegura su repetibilidad:
set.seed(140) # el argumento puede ser cualquier número
aprox<-0.003 # Valor determinante para la salida del ciclo
Y_ini<-2.7 # Supuesto valor inicial de Y
for(iter in 1:1000) { # aseguro no más de 1000 iteraciones
  # Procedimiento para calcular la siguiente Y, que
  # en este caso simularemos mediante generador aleatorio:
  Y<-Y_ini+0.008*rnorm(1)
  # La condición de salida:
  if(abs(Y-Y_ini)<=aprox)
    break # Uso del break para salir del ciclo
  # Preparamos para la siguiente iteración
  Y_ini<-Y
}
# Veamos que ha resultado:
paste0("Y_ini: ", Y_ini,", Y: ", Y,", Num.iter: ", iter)

#Ejem 2:
# Primero se crea la función:
fibbonacci<-function(n) {
  if(n%in%c(0,1))
    return(1)
  F0<-1; F1<-1; i<-2
  repeat{
    s<-F0+F1 # Suma de los fib anteriores
    if(i==n) # Ya es el que se busca
      return(s) # Sale hasta afuera de la función
    # recorremos los últimos dos próximos números
    F0<-F1
    F1<-s
    i<-i+1 # incrementamos el índice
  }
}
# El octavo número de fibbonacci se genera
# llamando a la función así:
fibbonacci(8)

#Ejem 3:
for(i in 1:7) {
  if(3<=i&&i<=5)
    next
  print(i)
}

#================================================
# XIII. Las funciones sapply() y lapply() y
#================================================

(misDatos<-data.frame(uno=runif(5,10.5,40.3),dos=runif(5),
                      tres=runif(5,155,890)))

sapply(misDatos, haz_promedio,simplify=TRUE) # aquí se podría haber usado
#la función 'mean' en vez de 'haz_promedio'

lapply(misDatos, mean)

r<-numeric() # vector numerico vacío para resultados
for(elt in misDatos) {
  r<-c(r,haz_promedio(elt)) # note el uso de c()
```

```
}
names(r)<-names(misDatos)

#1. Operaciones marginales en matrices y la función apply()

# Se hace una matriz arbitraria de 3 renglones y 5 columnas,
# con rbind, que la construye por renglones:
(mm<-rbind(5:9,runif(5,10,20),c(2,-2,1,6,-8)))

colSums(mm)
rowMeans(mm)

# para los renglones
apply(mm,1, sd) # el 1 es el margen: renglones
# para las columnas
apply(mm,2, sd) # el 2 es el margen: columna

# una función que toma un vector arbitrario y multiplica cada
# elemento, por su posición en el vector:
ff<-function(v) v*(1:length(v))
# probemos la función:
ff(c(2,4,6,8))
# Ahora la aplicaremos a todas las columnas de la matriz
apply(mm,2, ff)

#=================================================
# XIII. Loops
#=================================================

rm(list = ls(all = TRUE))


#=================================================
# XIV. For loops
#=================================================

# Ejemplo 1
primes_list <- list(2, 3, 5, 7, 11, 13)

# Primera linea: Sequence
# Segunda linea: Body

for (i in 1:length(primes_list)) {
  print(primes_list[[i]])
}

# Ejemplo 2
df <- data.frame (
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)

for (i in 1:ncol(df)) {
  print(median(df[[i]]))
}

# Ejemplo 3
n <- 10
for (i in 1:n) {
  print(i)
}
```

```
for (i in 1:n) {
  print(i + i^2)
}

# Ejemplo 4
res <- rep(NA, 10)
res
for (i in 1:n) {
  res[i] <- i + i^2
}
res


# Ejemplo 5
# General
for (variable in vector){
  commands
}

for (i in 1:5) {
  print(i^2)
}

# Ejemplo 6
x <- c(-3, 6, 2, 5, 9)
for(i in x){
  print(i^2)
}

for(i in x){
  print(c(i, i^2))
}

# Ejemplo 7
Storage <- numeric(5)
Storage
for (i in 1:5){
  Storage[i]<-i^2
}
Storage
mean(Storage)

# Ejemplo 8
x
Storage2 <- numeric(5)
for (i in 1:5){
  Storage2[i]<-(x[i])^2
}
Storage2

# Ejemplo 9
for(DegC in c(-3,6,2,5,9)){
  DegF<-DegC*(9/5)+32
  print(c(DegC,DegF))
}

# Ejemplo 10
# Este no funciona
Temp <- c(-4, 5, 10, -6, -40, 30)
if(Temp>0) {
  print("warm")
}else{
  print("not so warm")
}
```

```r
# Este si funciona
for(Temp in c(-4, 5, 10, -6, -40, 30)){
  if(Temp>0) {
    print("warm")
  }else{
    print("not so warm")
  }
}


#=================================================
# XIV. While loops
#=================================================

# General
while (logical condition){
  commands
}

# Ejemplo 1
x<-1
while(x<10){
  print(x)
  x <- x+1
}

# Ejemplo 2
x<-1
while(x<10){
  x <- x+1
  print(x)
}

# Ejemplo 3
x<-1
while(x<10){
  x <- x+1
}
print(x)

# Ejemplo 4
Storage <- c()
Storage

x<-1
while(x<10){
  Storage <- c(Storage,x)
  x<-x+1
}
Storage

# Ejemplo 5
i<-1
while(i<=2){
  j<-1
  while(j<=2){
    print(c(i,j))
    j<-j+1
  }
  i<-i+1
}
```

```
# Ejemplo 6 (Be Careful!)
x<-1
while(x<2){
  print(x)
}



#=========================================================================================#

  ############################ Fin sesion 2 ###################################}

#=========================================================================================#
```