**Temas:** Introducción a R

1. (**Instalar R**) Descargue e instale R de `https://www.r-project.org/`.

2. (**Instalar un IDE**) Como IDE (ambiente integrado de desarrollo) descargue e instale RStudio de `https://www.rstudio.com/`.

3. (**Operaciones aritméticas**) En RStudio tiene una ventana de consola donde puede puede interactuar con R a través de su CLI (interfaz de línea de comandos). Escriba allí algunas operaciones aritméticas, por ejemplo:

```
> 3 + 4
[1] 7
```

   Note que la CLI espera comandos en la línea que inicia con >. La respuesta se muestra en una línea numerada. Intente ahora otras operaciones: -, *, /, %. R también tiene la operación ^.

4. (**Funciones en R**) R permite definir funciones y trae un número de funciones por defecto. Intente por ejemplo,

```
> sqrt(10)
[1] 3.162278
> exp(1)
[1] 2.718282
```

   Si tiene dudas sobre una función puede usar el comando "?" antes del nombre de la función. Intente por ejemplo,

```
?exp
```

   Este comando debe desplegar una ventana de ayuda con la documentación de esta función.

5. (**Reutilizar comandos**) Si ya ha escrito varios comandos en la consola este es un buen momento para utilizar las teclas arriba ↑ y abajo ↓. Éstas le permiten navegar por comandos utilizados anteriormente y usarlos como base para el nuevo comando que escribirá.

6. (**Opciones**) La función "options" permite modificar un número de parámetros, por ejemplo, el número de cifras decimales.

```
> pi
[1] 3.141593
> options(digits=16)
> pi
[1] 3.141592653589793
```

   Para explorar otros parámetros que permite modificar esta función use "?options".

7. (**Variables**) Nuevas variables se pueden definir asignándoles un valor. Por ejemplo,

```
> x<-5
```

Su valor puede explorarse luego

```
> x
[1] 5
```

Varias variables pueden definirse y ejecutar operaciones de acuerdo con su tipo, en esta tenemos enteros. Por ejemplo,

```
> y<-10
> x+y
[1] 15
```

Note que todas las variables creadas quedan listadas en la ventana "environment" de RStudio. Esta ventana muestra el "workspace" de R, es decir, todas las variables y objetos que se han definido hasta el momento. Para eliminar una variable del "workspace" podemos ejecutar el siguiente comando

```
> rm(x)
```

Tal que al explorar su valor obtenemos

```
> x
Error: object 'x' not found
```

8. (**Tipos de datos y variables**) Las variables en R pueden ser de diversos tipos. Por ejemplo, números reales:

```
> x <- pi
> x
[1] 3.141592653589793
```

Número complejos:

```
> x <- as.complex(1)
> x
[1] 1+0i
> y <- 2+1i
> y
[1] 2+1i
```

Cadenas de caracteres:

```
> z <- "hola"
> z
[1] "hola"
```

La función "typeof" permite saber a qué clase pertenece una variable:

```
> typeof(pi)
[1] "double"
> typeof(y)
[1] "complex"
> typeof(z)
[1] "character"
```

9. (**Vectores**) Para definir vectores se pueden concatenar valores usando la función "c", así

```
> x <- c(2,3,5,4,8,6)
> x
[1] 2 3 5 4 8 6
```

También podemos definir vectores de enteros que crecen en pasos de 1, así

```
> x <- 3:8
> x
[1] 3 4 5 6 7 8
```

O de manera más general podemos usar las función "seq", así

```
> y <- seq(from=2, by =0.2, length.out = 5)
> y
[1] 2.0 2.2 2.4 2.6 2.8
```

Note que la función recibe tres parámetros con nombre, y los valores de los parámetros se asignan con el operador "="; esto permite pasar los parámetros en cualquier orden. Si no se usan los nombres de los parámetros, la función interpreta los parámetros en el orden en que están definidos internamente según la documentación.

También es posible construir vectores vacíos de diversos tipos.

```
> vector("numeric", 5)
[1] 0 0 0 0 0
> z <- vector("numeric", 5)
> z
[1] 0 0 0 0 0
> z <- vector("complex", 5)
> z
[1] 0+0i 0+0i 0+0i 0+0i 0+0i
> z <- vector("logical", 5)
> z
[1] FALSE FALSE FALSE FALSE FALSE
> z <- vector("character", 5)
> z
[1] "" "" "" "" ""
```

Y podemos usar la función "length" para determinar la longitud de un vector

```
> length(z)
[1] 5
```

Los vectores se indexan empezando en 1:

```
> y[1]
[1] 2
> y[3]
[1] 2.4
> y[2:4]
[1] 2.2 2.4 2.6
> y[c(1,3)]
[1] 2.0 2.4
```

También puede usar el signo "−" para indicar elementos a exluir:

```
> y[-2]
[1] 2.0 2.4 2.6 2.8
```

10. (**Constantes**) R incorpora un número de constantes que pueden ser de utilidad, por ejemplo

```
> pi
[1] 3.141592653589793
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P
    " "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p
    " "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

11. (**Matrices**) Es posible crear matrices usando la función "array":

```
> matriz <- array(1:12,dim = c(3,4))
> matriz
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Note que con esta misma estructura es posible crear arreglos $n$-dimensionales. Además es posible definir nombres para los índices en cada dimensión,

```
> matriz <- array(1:12,dim = c(3,4), dimnames=list( c("uno", "dos",
    "tres"), c("A", "B", "C", "D")  ) )
> matriz
     A B C  D
uno  1 4 7 10
dos  2 5 8 11
tres 3 6 9 12
```

El tamaño del arreglo puede obtenerse con la función "dim":

```
> dim(matriz)
[1] 3 4
```

12. (**Scripts y Funciones**) Después de habernos familiarizado con los elementos básicos del lenguaje a través del CLI, creemos un nuevo script de R dando click en "File → New File → R Script". En la nueva ventana (archivo de text vacío de extensión .R) escriba el siguiente código

```
distanciaEuclideana <- function(x1=2, y1=2, x2=3, y2=3){
  a <- (x1-x2)
  b <- (y1-y2)
  sqrt( a^2 + b^2 )
}

x <- distanciaEuclideana(1,3, 4,5)
y <- distanciaEuclideana(1,2, 1,2)
```

Una vez termine de escribir el código, guarde el script en una ubicación conocida y haga click en "Source". Note el cambio en el valor de las variables $x$ y $y$ en el "workspace". Alternativamente, ejecute cada línea del script usando "CTRL+ENTER".

13. (**Condicionales**) Expanda su código anterior de la siguiente manera:

```
distanciaEuclideana <- function(x1=2, y1=2, x2=3, y2=3){
  a <- (x1-x2)
  b <- (y1-y2)
  sqrt( a^2 + b^2 )
}

evaluarNumero <- function(x=0){
  if(is.nan(x))
  {
    message("x is missing")
  } else if(is.infinite(x))
  {
    message("x is infinite")
  } else if(x > 0)
  {
    message("x is positive")
  } else if(x < 0)
  {
    message("x is negative")
  } else
  {
    message("x is zero")
  }
}

x <- distanciaEuclideana(1,3, 4,5)
y <- distanciaEuclideana(1,2, 1,2)
evaluarNumero(2)
```

14. (**Ciclos**) R cuenta con estructuras de control similares a otros lenguajes, incluyendo ciclos. Es posible definir ciclos de tipo "while", por ejemplo

```
n = 5
while(n > 0)
{
  message("contador: ",n)
  n = n-1
}
```

También es posible definir ciclos de tipo "for". Considere y ejecute los siguientes ejemplos:

```
for(i in 1:5){
  message("i^2: ",i^2)
}

for(i in c(3, 4, 6, 9)){
  message("i^2: ",i^2)
```

```
}

for(i in seq(from=0.1, by=0.2, length.out = 6)){
  message(i,"^2: ",i^2)
}
```

15. (**Más funciones**) Explore las siguientes funciones:

```
matrix
list
sample
runif
plot
barplot
```