Clase 4 (Beta)

Alexis Zúñiga & Ernesto Barrios

Capítulo 10: RStudio

Hasta ahora la consola ha sido algo útil para empezar a familiarizarse con el lenguaje R, pero existen otras maneras más sencillas de visualizar los comandos, editarlos y recibir las respuestas.

RStudio es una interfaz o IDE (Entorno Integrado de Desarrollo) diseñado especificamente para R. RStudio nos va a permitir escribir archivos ejecutables en los que podemos incluir todo un programa y editarlo sin tener que ir linea por linea.

Aqui está la guía para instalarlo:

Descargar R studio

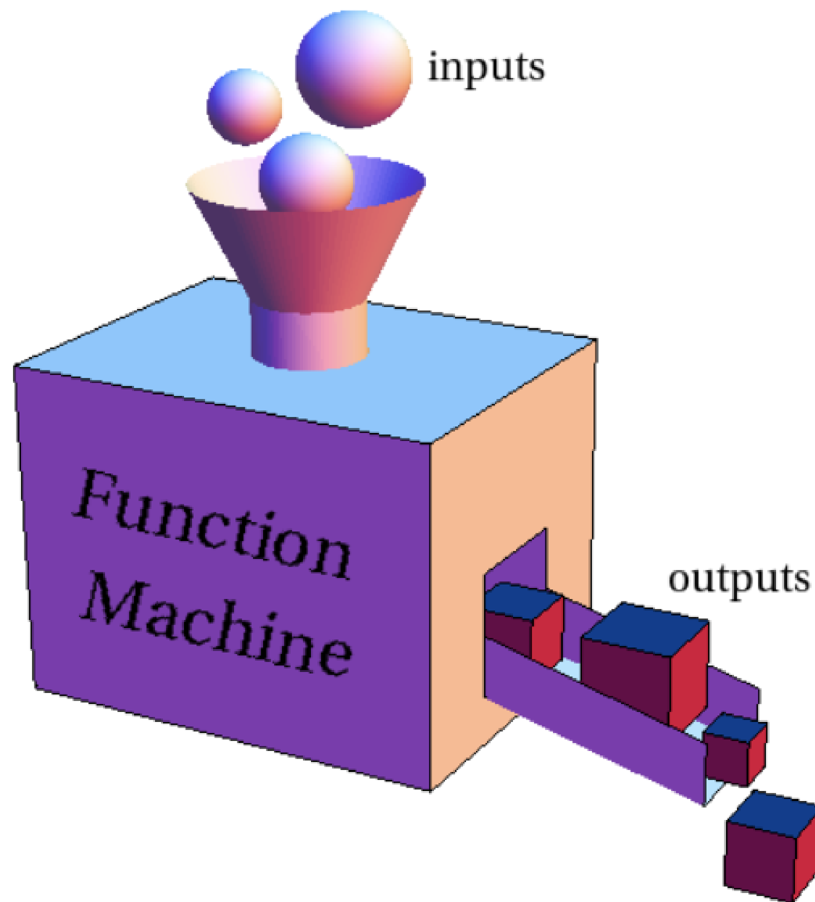
1. Abre tu navegador y dirigite al sitio al que pertenence el siguiente link: https://posit.co/
2. Hacer click en **DOWNLOAD RSTUDIO**
3. En la sección llamada **RStudio Desktop** hacer click donde diga **Download RStudio**
4. Hacer click en **Download Rstudio** para el sistema operativo que maneje.
5. En caso de no aparecer, buscar en la lista *All Installers* y guardar el archivo ejecutable.
6. Correr el archivo que acaba de descargar (.dmg para *macOS* o .exe para *Windows*) y seguir las instrucciones de instalación.

Capítulo 11: Funciones

Una de las ventajas de R es la gran cantidad de funciones que contiene para realizar distintos procedimientos.

¿Qué es una función?

Como se define en el manual de R (Hernández & Usuga, 2024), "una función es un conjunto de instrucciones que convierten las entradas (inputs) en resultados (outputs) deseados." En la siguiente figura se muestra una ilustración de lo que es una función.



Estructura de una función:

```
fun1 <- function(p1,p2,...){
  cuerpo
  cuerpo
  cuerpo
  cuerpo
```

```r
  return(resultados)
}
```

Ejemplos

Ejemplo 1: Construir una función que reciba dos números y que entregue la suma de estos números.

```r
suma <- function(x,y){
  resultado <- x + y
  return(resultado)
}
```

Para acceder a la función deseada solamente se debe ejecutar:

```r
suma(x=4,y=6)
```

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

Para funciones simples es posible escribirlas de manera más sencillas como se muestra a continuación:

```r
suma2 <- function(x,y){
  return(x+y)
}

suma2(2,8)
```

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

Ejemplo 2: Construya una función que reciba dos números de la recta real y que entregue el punto médio de estos números. El resultado debe ser un mensaje por pantalla.

```r
medio <- function(a, b) {
  medio <- (a + b) / 2
  cat("El punto medio de los valores", a, "y", b,
      "ingresados es", medio)
}

medio(a=-3, b=3)  # Probando la función
```

```
## El punto medio de los valores -3 y 3 ingresados es 0
```

La función 'cat' es muy útil para presentar resultados por pantalla.

For & While

En R, los bucles `for` y `while` son estructuras de control que permiten repetir un bloque de código varias veces. A continuación, se explican con detalle y se muestran ejemplos para cada uno.

For

Este bucle se utiliza para iterar sobre una secuencia (como vectores listas, conjuntos de números, etc.) y ejecutar un bloque de código para cada elemento de la secuencia.

Ejemplo:

```r
#Sintáxis básicas
#Supongamos que queremos imprimir los números del 1 al 5:
for (i in 1:5) {
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

En este ejemplo, i toma los valores de 1 a 5 en cada iteración, y la función print imprime cada valor.

While

Este bucle se utiliza para repetir un bloque de código mientras una condición especificada sea verdadera. La condición se evalúa antes de cada iteración del bucle.

Ejemplo:

```r
#Sintáxis básica
#Supongamos que queremos imprimir los números del 1 al 5 usando un bucle while:
i <- 1
while (i <= 5) {
    print(i)
    i <- i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

En este ejemplo, se inicializa i a 1 antes de comenzar el bucle. Luego, en cada iteración, se imprime el valor de i y se incrementa en 1. El bucle continúa ejecutándose mientras i sea menor o igual a 5.

- **for es útil cuando se conoce de antemano el núemro de iteraciones, ya que se itera sobre una frecuencia definida**
- **while es útil cuando no se conoce el número de iteraciones y depende de una condición que pueda cambiar dentro del bucle.**

Ejemplos adicionales

```
vec <- c("a", "b", "c", "d")
for (letra in vec) {
    print(letra)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

En este ejemplo, letra toma cada valor del vector vec en cada iteración y se imprime.

```
x <- 1
while (x^2 < 100) {
    print(x)
    x <- x + 2
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
```

En este caso, el bucle imprime valores de x mientras el cuadrado de x sea menor que 100, incrementando x en 2 en cada iteración.

Capítulo 12: Números aleatorios

En R, generar números aleatorios es una tarea común que se puede realizar de diversas formas utilizando funciones predefinidas. Aquí se presentan algunas maneras de hacerlo y ejemplos útiles

1. Generar números aleatorios uniformes

Para generar números aleatorios con una distribución uniforme, se usa la función `runif()`.

```
#Generar 5 números aleatorios entre 0 y 1:

set.seed(123)  # Establecer una semilla para reproducibilidad
(random_uniform <- runif(5))
```

```
## [1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673
```

```
#Generar 5 números aleatorios entre 10 y 20:

(random_uniform_10_20 <- runif(5, min = 10, max = 20))
```

```
## [1] 10.45556 15.28105 18.92419 15.51435 14.56615
```

```
#Si quisieramos convertirlo a una función discreta usamos round()
(round(random_uniform_10_20 <- runif(5, min = 10, max = 20)))
```

```
## [1] 20 15 17 16 11
```

Nótese que aunque en ambos casos usamos los mismos argumentos en la función, obtuvimos resultados diferentes (aparte de que los segundos están redondeados). Si quisiéramos obtener los mismos números cada vez que corremos el experimento podemos recurrir a la función `set.seed()`.

En su uso más básico, la función `set.seed()` recibe un parámetro numérico llamado semilla que hace que las funciones sean repetibles. No importa cuándo o dónde se corra la función pseudo-aleatoria, si tienen la misma semilla antes, obtendremos el mismo resultado.

```
set.seed(13)
round(runif(5,100,112))
```

```
## [1] 109 103 105 101 112
```

Si lo volvemos a correr:

```
set.seed(13)
round(runif(5,100,112))
```

```
## [1] 109 103 105 101 112
```

La consola nos regresa exactamente lo mismo.

2. Generar números aleatorios normales

Para generar números aleatorios con una distribución normal, se usa la función `rnorm()`.

```
# Generar 5 números aleatorios con media 0 y desviación estándar 1:

set.seed(123)
(random_normal <- rnorm(5))
```

```
## [1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
```

Si revisamos la media y desviación estándar de este experimento podemos confirmar nuestros argumentos iniciales (con cierto margen de error por el tamaño de la muestra).

```r
vector_normal_estandar <- rnorm(300)
# media cercana a cero
mean(vector_normal_estandar)
```

```
## [1] 0.01822809
```

```r
#desviación estándar cercana a 1
sd(vector_normal_estandar)
```

```
## [1] 0.947009
```

Podemos hacer lo mismo escogiendo la media y la desviación estándar

```r
# Generar 5 números aleatorios con media 10 y desviación estándar 2:
(random_normal_10_2 <- rnorm(5, mean = 10, sd = 2))
```

```
## [1] 10.662358  5.971579 10.423961 12.473350 14.075148
```

3. Generar número aleatorios

Para generar números enteros aleatorios, se puede usar la función `sample()`.

```r
# Generar 5 números enteros aleatorios entre 1 y 100:

set.seed(123)
(random_integers <- sample(1:100, 5))
```

```
## [1] 31 79 51 14 67
```

4. Generar números aleatorios binomiales

Para generar números aleatorios con una distribución binomial, se usa la función `rbinom()`.

```r
#Generar 5 números aleatorios con 10 ensayos y probabilidad de éxito 0.5:
set.seed(123)
(random_binomial <- rbinom(5, size = 10, prob = 0.5))
```

```
## [1] 4 6 5 7 7
```

Veamos una simulación más compleja. Un equipo de futból, con 11 jugadores, sabe que la probabilidad de que uno de ellos se lesione cada partido es de 5%. Podemos simular el número de lesiones en el calendario de 10 partidos con la función `rbinom()`

```r
# rbinom(número de partidos, número de jugadores, probabilidad de lesión)
set.seed(46)
rbinom(10, 11, 0.05)
```

```
##  [1] 0 0 1 0 0 1 1 1 1 2
```

En este caso vemos que se lesionó 1 en el tercer partido, 1 en el sexto, etc. Si queremos obtener el total de lesionados podemos utilizar operaciones de vectores.

```r
set.seed(46)
sum(rbinom(10, 11, 0.05))
```

```
## [1] 7
```

5. Generar números aleatorios exponenciales

Para generar números aleatorios con una distribución exponencial, se usa la función `rexp()`.

```
#Generar 5 números aleatorios con tasa (rate) 1:

set.seed(123)
(random_exponential <- rexp(5, rate = 1))
```

```
## [1] 0.84345726 0.57661027 1.32905487 0.03157736 0.05621098
```

- runif(n, min, max): Números uniformes entre min y max.
- rnorm(n, mean, sd): Números normales con media mean y desviación estándar sd.
- sample(x, size): Muestra de tamaño size de los elementos de x.
- rbinom(n, size, prob): Números binomiales con size ensayos y probabilidad de éxito prob.
- rexp(n, rate): Números exponenciales con tasa rate.