Introducción a Instrucciones Básicas en R

¡Bienvenido/a a este recurso introductorio sobre algunas instrucciones fundamentales en R!

Una instrucción en R es una orden o comando que le indicas al programa para que realice una tarea específica. Puede ser desde realizar operaciones matemáticas simples hasta crear estructuras de datos complejas o generar gráficos. En este documento, nos enfocaremos en explorar y comprender algunas de estas instrucciones fundamentales en R que te permitirán iniciar y comprender mejor cómo funciona este lenguaje de programación.

Índice

1. Operaciones Aritméticas

En R, las operaciones aritméticas básicas se realizan utilizando símbolos específicos. Aquí tienes algunos ejemplos:

- **Suma**: El símbolo "+". Por ejemplo:

```
> 4+5
[1] 9
```

- **Resta**: El símbolo "-". Por ejemplo:

```
> 5-6
[1] -1
```

- **Multiplicación**: El símbolo "*". Por ejemplo:

```
> 44*100
[1] 4400
```

- **División**: El símbolo "/". Por ejemplo:

```
> 55/80
[1] 0.6875
```

- **Potencia**: El símbolo "**" o "^". Por ejemplo:

```
> 5**2
[1] 25
```

- pi → Representa el **número π**
- exp → Representa el **número e**
- 5e4 o 5e+4 / 5e-4→ Forma de representar la **notación científica** 5x10^4 / 5x10^-4. Por ejemplo:

```
> 5e2
[1] 500
> 5e+2
[1] 500
> 5e-2
[1] 0.05
```

```
> 6**19
[1] 6.093597e+14
```

2. Orden de Ejecución de Operaciones

Comprender el orden en el que R ejecuta las operaciones matemáticas del orden de operaciones es crucial para evitar errores y para asegurar que las expresiones matemáticas se evalúen correctamente en R. Este orden sigue las reglas convencionales:

1. **Potencias**: Las operaciones de potenciación tienen la prioridad más alta.
2. **Productos y Divisiones**: Multiplicaciones y divisiones se realizan antes que las sumas y restas.
3. **Sumas y Restas**: Si hay múltiples sumas o restas, se resuelven de izquierda a derecha.

- Cuando se dice que las operaciones se realizan de izquierda a derecha, significa que, en una secuencia de operaciones, como por ejemplo 4 + 2 * 3, se resolverían de la siguiente manera:

Primero se realizaría la multiplicación: 2 * 3 = 6.

Luego, la suma: 4 + 6 = 10.

```
> 4 + 2 * 3
[1] 10
```

Este orden se puede **modificar usando paréntesis** para agrupar operaciones y asegurar que se ejecuten en el orden deseado. Por ejemplo:

(3 + 5) * 2 se ejecutará primero la suma dentro del paréntesis y luego se multiplicará el resultado por 2.

- **Con Paréntesis**

```
> (3+5)*2
[1] 16
```

- **Sin Paréntesis**

```
> 3+5*2
[1] 13
```

3. Operadores Aritméticos y de Comparación en R

En R, los operadores aritméticos permiten realizar comparaciones entre valores. Estos operadores son fundamentales para establecer **condiciones** y realizar evaluaciones lógicas en el código.

Operadores de Comparación:

- **< (Menor):** Evalúa si un valor es menor que otro. Por ejemplo, 3 < 5 devuelve TRUE ya que 3 es menor que 5.

```
> 3 < 5
[1] TRUE
```

- **> (Mayor):** Compara si un valor es mayor que otro. Por ejemplo, 7 > 4 devuelve TRUE ya que 7 es mayor que 4.

```
> 7 > 4
[1] TRUE
```

- **<= (Menor o igual):** Verifica si un valor es menor o igual a otro. Por ejemplo, 3<=3 devuelve TRUE ya que 3 es igual a 3.

```
> 3<=3
[1] TRUE
```

- **>= (Mayor o igual):** Compara si un valor es mayor o igual a otro. Por ejemplo, 5>=9 devuelve FALSE ya que 5 no es mayor ni igual a 9.

```
> 5>=9
[1] FALSE
```

- **!= (Distinto de):** Comprueba si dos valores no son iguales. Por ejemplo, 4 != 3 devuelve TRUE ya que 4 es distinto de 3.

```
> 4!=3
[1] TRUE
```

- **== (Igualdad lógica):** Evalúa si dos valores son iguales. Es importante resaltar que este operador verifica la **igualdad lógica**, es decir, **si los valores son idénticos**. Por ejemplo, 4 == 4.0 devuelve TRUE aunque uno sea un entero y otro un número decimal, porque tienen el mismo valor.

```
> 4==4.0
[1] TRUE
```

4. Generación de Números Aleatorios

Una de las funcionalidades interesantes de R es su capacidad para generar números aleatorios. Aquí hay un par de ejemplos:

- rnorm(n) : Esta función genera "n" números aleatorios distribuidos normalmente alrededor de cero, con desviación estándar 1.

```
> rnorm(7)
[1] -2.1362226 -0.5859401  0.8901199  1.1341112 -1.2102310
[6] -0.4194429 -0.7124721
> rnorm(7)
[1]  0.9328231 -0.6191353 -1.4317970 -0.6226483  0.9710651
[6]  1.0048095 -0.1387574
> rnorm(7)
[1]  0.683812190  1.388479715 -0.373046871  0.008783302
[5]  1.201518416  0.797632071 -0.932465933
```

Ideal para generar números aleatorios que sigan una distribución normal. Útil en estadística, simulaciones o modelado que requieren datos con comportamiento normalizado alrededor de cero.

- runif(n) : Genera "n" números al azar entre 0 y 1. Es útil en pruebas estadísticas o simulaciones donde se requiera una distribución uniforme de valores. Por ejemplo:

```
> runif(7)
[1] 0.1300820 0.4001708 0.6773421 0.7441510 0.1124235
[6] 0.2079270 0.1618900
> runif(7)
[1] 0.22743704 0.06991437 0.14427766 0.86182794 0.99663926
[6] 0.77030614 0.77336515
> runif(7)
[1] 0.2490893 0.7140841 0.7609768 0.4427775 0.6584211
[6] 0.8341570 0.4765795
```

Además de ser útil para crear números aleatorios distribuidos uniformemente entre 0 y 1 (como se observa en el ejemplo), esta función también permite obtener N valores aleatorios en un **intervalo diferente**. Por ejemplo, si se desea generar 15 valores aleatorios en el intervalo de [0, 150], se puede utilizar la expresión: 150*runif (15)

```
> 150*runif(15)
 [1] 117.723678  24.869726  44.422489  89.488347  68.029810  21.288133
 [7]  57.918140   8.260955  59.635523  19.973658 123.959892  93.759879
[13] 114.896824  19.518723  18.181783
```

- Sample (a:b,M) : Función que genera números enteros aleatorios, M en total, dentro del rango o intervalo [a, b], con la opción de **permitir o no la repetición** de valores. Si se utiliza replace = TRUE, algunos valores pueden repetirse, como se ilustra en el siguiente ejemplo:

```
> sample(1:35,9,replace=TRUE)
[1] 31 35  9 31 34 31 16 32 14
```

Observamos que al usar la opción replace = TRUE, el número 31 se repite en múltiples ocasiones. Por el contrario, al emplear replace = FALSE, no se presentan valores repetidos, como se muestra en el siguiente ejemplo:

```
> sample(1:35,9,replace=FALSE)
[1] 18 24 30 11  8 14  4 32 35
```

- x = 1:n : Esta instrucción crea una secuencia de números enteros desde 1 hasta “n”.

```
> x=1:7
> x
[1] 1 2 3 4 5 6 7
> x=1:20
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
[20] 20
```

Útil al crear una secuencia de números enteros consecutivos desde 1 hasta "n". Se utiliza frecuentemente en bucles, iteraciones o al generar índices para acceder a datos en R.

Reemplaza “n” con el número deseado para generar la cantidad específica de números aleatorios que necesites.

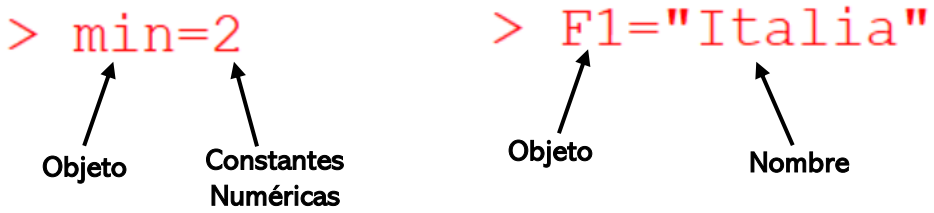
Estos son solo algunos ejemplos de cómo trabajar con números aleatorios en R. Estas funciones son esenciales para simular datos, realizar pruebas estadísticas y trabajar en análisis exploratorios.

5. Gestión de Objetos en la Memoria

En R, es fundamental tener el control y la gestión adecuada de los objetos que están almacenados en la memoria del entorno de trabajo.

En R, un objeto puede ser un nombre que representa un valor específico, como en el caso de **a = 4**, donde **a** sería el **objeto** y 4 sería el valor asignado a ese objeto. Los objetos en R pueden referirse a:

- Constantes: Valores fijos como números, caracteres o valores lógicos.
- Estructuras de Datos: Vectores, matrices, marcos de datos, listas, entre otros.
- Variables: Nombres que representan valores asignados.



Algunas instrucciones clave para manejar estos objetos incluyen:

- ls() : Muestra todos los objetos guardados actualmente en la memoria.
- ls.str() : Divide los objetos en función de si son caracteres o números.
- ls(pat="letra") : Filtra y muestra los objetos que contienen la letra especificada dentro de su nombre. Es importante destacar que las **letras o palabras** se suelen encerrar **entre comillas** para indicar a R que se trata de caracteres.
- ls(pat="^letra") : Muestra objetos que comienzan con la letra especificada. El uso de comillas para especificar patrones es crucial en este tipo de operaciones.
- rm(objeto) : Permite eliminar un objeto específico de la memoria de R. Es importante mencionar que, al usar esta función, se debe proporcionar el nombre del "**objeto**" que se desea eliminar.

En este ejemplo, comenzamos definiendo los objetos que vamos a utilizar. Luego, aplicamos secuencialmente cada una de las instrucciones previamente mencionadas.

Esta secuencia nos permite observar cómo estas instrucciones operan en relación con los objetos que hemos creado.

```
> min=2
> max=3
> F1="Italia"
> H2="Portugal"
>
1 > ls()
[1] "F1"  "H2"  "max" "min"
2 > ls.str()
F1 :  chr "Italia"
H2 :  chr "Portugal"
max :  num 3
min :  num 2
3 > ls(pat="F")
[1] "F1"
4 > ls(pat="^m")
[1] "max" "min"
5 > rm("H2")
> ls()
[1] "F1"  "max" "min"
```

Nota importante: Para **eliminar todos los objetos** del entorno actual en R, se puede utilizar la instrucción rm(list=ls(all=TRUE)). Por ejemplo:

En este ejemplo, observamos como los objetos fueron eliminados de la memoria

```
> min=2
> max=3
> F1="Italia"
> H2="Portugal"
>
> ls()
[1] "F1"   "H2"   "max" "min"
> rm(list=ls(all=TRUE))
> ls()
character(0)
```

Esta operación es drástica y borrará todos los objetos existentes. Se recomienda precaución al usar esta instrucción, ya que puede resultar en la pérdida de datos no deseados.

6. Funciones Matemáticas Avanzadas

6.1. Funciones Exponenciales y Logaritmos

En R, también puedes trabajar con logaritmos y funciones exponenciales:

- exp(x) : Representa la función exponencial de 'x'.

-Casos Específicos:

- log(x) : logaritmo neperiano.
- log10(x) : logaritmo en base 10.

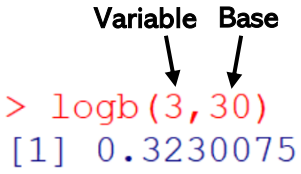
```
> log10(3)
[1] 0.4771213
```

- log2(x) : logaritmo en base 2.

```
> log2(3)
[1] 1.584963
```

-Casos Generales:

- logb(x,base) : Calcula el logaritmo en cualquier base. Donde puedes reemplazar 'x' por el número o variable y "base" por la base en la que deseas calcular el logaritmo.



Observamos que esta instrucción también sirve para base 10 y base 2

6.2. Funciones Trigonométricas

R permite trabajar con funciones trigonométricas básicas y sus inversas:

- sin(x) : Calcula el seno de 'x'.
- cos(x) : Calcula el coseno de 'x'.
- tan(x) : Calcula la tangente trigonométrica de 'x'.
- asin(x) : Calcula el arco seno de 'x'.
- acos(x) : Calcula el arco coseno de 'x'.
- atan(x) : Calcula el arco tangente de 'x'.

6.3. Otras Funciones Matemáticas

Además de las funciones trigonométricas y logarítmicas, hay otras funciones útiles:

- abs(x) : Devuelve el valor absoluto de 'x'.
- sqrt(x) : Calcula la raíz cuadrada de 'x'.
- factorial(x) : Calcula el factorial de 'x'.

7. Vectores y Matrices en R

7.1. Vectores en R

Los vectores son fundamentales en R y se manejan de la siguiente manera:

- Cama= c (número, …) o (“carácter”, …) : Define/crea un vector llamado “Cama”.

```
> Cama= c(8,6,5)
```

- Cama [n]: Accede al componente n del vector “Cama”. Por ejemplo: acceder al componente 2 del vector “Cama”.

```
> Cama[2]
[1] 6
```

- **Operaciones Componente a Componente entre vectores**: Usa +, -, * para sumar, restar y multiplicar componentes respectivamente. Por ejemplo: Definimos 2 vectores “Cama” y “Silla” y los sumamos.

```
> Cama=c(8,6,5)
> Silla=c(10,4,9)
>
> Cama+Silla
[1] 18 10 14
```

- %*%: Realiza el producto escalar de vectores. Por ejemplo: Usando los vectores previamente mencionados aplicamos el producto escalar.

```
> Cama=c(8,6,5)
> Silla=c(10,4,9)
>
> Cama%*%Silla
     [,1]
[1,]  149
```

7.2. Matrices en R

Las matrices en R son arreglos bidimensionales que se manejan de la siguiente manera:

-Creación de una Matriz:

A = matrix(c (número o "carácter", número, …), nrow = n, ncol = n)

Esta instrucción crea una matriz "A" con "n" filas y "n" columnas. Por ejemplo

```
> A = matrix(c(3,4,5,9),nrow=2,ncol=2)
> A
     [,1] [,2]
[1,]    3    5
[2,]    4    9
```

R primero completa las columnas y después las filas

-Acceso a Posiciones:

A [nrow, ncol]

Permite acceder a una posición específica dentro de la matriz "A". Por ejemplo: La posición columna (1) fila (2).

```
> A
     [,1] [,2]
[1,]    3    5
[2,]    4    9
> A [2,1]
[1] 4
```

-Generación de Matrices a partir de Vectores:

Utiliza cbind() (para colocar vectores en columnas) y rbind() (para colocar vectores en filas) para construir matrices. Por ejemplo: B = cbind (vector1, vector2, vector3)

```
> vector1= c(3,4,8,10)
> vector2= c(2,7,9,40)
> vector3= c(6,1,5,15)
>
> B=cbind(vector1, vector2, vector3)
> B
     vector1 vector2 vector3
[1,]       3       2       6
[2,]       4       7       1
[3,]       8       9       5
[4,]      10      40      15
```

-Operaciones Componente a Componente:

Utiliza + y - para realizar suma y resta componente a componente, respectivamente. Por ejemplo: A+C

```
> A = matrix(c(3,4,5,9),nrow=2,ncol=2)
> C = matrix(c(7,15,6,20),nrow=2,ncol=2)
>
> A+C
     [,1] [,2]
[1,]   10   11
[2,]   19   29
```

-Producto Escalar de Matrices:

A %*% C realiza el producto escalar de las matrices "A" y "C". Por ejemplo:

```
> A = matrix(c(3,4,5,9),nrow=2,ncol=2)
> C = matrix(c(7,15,6,20),nrow=2,ncol=2)
>
> A %*% C
     [,1] [,2]
[1,]   96  118
[2,]  163  204
```

-Otras Operaciones con Matrices

-Transpuesta de una Matriz:

t(A) :Obtiene la transpuesta de la matriz "A".

```
> A = matrix(c(3,4,5,9),nrow=2,ncol=2)
> t(A)
     [,1] [,2]
[1,]    3    4
[2,]    5    9
```

-Determinante de una Matriz:

det (A) : Calcula el determinante de la matriz "A".

```
> A = matrix(c(3,4,5,9),nrow=2,ncol=2)
> det(A)
[1] 7
```

-Solución de Sistemas de Ecuaciones:

solve (A,b) : Resuelve el sistema de ecuaciones lineales Ax=b, donde b puede ser una matriz a un vector, compatible con el número de filas y columnas de A. Por Ejemplo:

```
> A= matrix(c(3,4,5,9),nrow=2,ncol=2)
> b= c(2,3)
> solve (A, b)
[1] 0.4285714 0.1428571
```

solve (A) : Calcula la inversa de la matriz “A”.

```
> A = matrix(c(3,4,5,9),nrow=2,ncol=2)
> A
     [,1] [,2]
[1,]    3    5
[2,]    4    9
> solve(A)
           [,1]       [,2]
[1,]  1.2857143 -0.7142857
[2,] -0.5714286  0.4285714
```

8. Data.Frames en R

Los data.frames en R son estructuras de datos que permiten almacenar información en forma tabular, similar a una hoja de cálculo o una base de datos, donde cada columna puede ser de un **tipo de datos diferente,** ya sean datos numéricos o palabras. Algunos puntos clave son:

-Creación de un Data.Frame:

AA = data.frame (vector1, vector2, vector3)

Esta instrucción crea un data.frame llamado "AA" a partir de los vectores previamente proporcionados. Los **nombres** de los **vectores se convertirán en los nombres de las columnas** en el data.frame resultante.

```
> vector1= c(4,5,6)
> vector2= c("Alicia","Pedro","Sonia")
> vector3= c(1.3,1.4,4)
>
> AA = data.frame(vector1, vector2, vector3)
> AA
  vector1 vector2 vector3
1       4  Alicia     1.3
2       5   Pedro     1.4
3       6   Sonia     4.0
```

Los **data.frames** en R **permiten que cada columna contenga diferentes tipos de datos**, mientras que en las matrices todos los elementos deben ser del mismo tipo.

Esta versatilidad hace que los data.frames sean ideales para trabajar con conjuntos de datos heterogéneos, donde distintas columnas pueden contener números, texto, fechas u otros tipos de datos. Entre algunas situaciones en donde los data.frame son muy útiles tenemos:

- **Análisis de Datos:** Para almacenar datos de encuestas, donde una columna puede representar la edad (numérica), otra el género (categórico), y otra el puntaje obtenido en una prueba (numérico).
- **Regresión y Modelado:** Al trabajar con conjuntos de datos para regresión lineal o modelado estadístico, donde diferentes columnas pueden representar variables independientes o predictivas de diferentes tipos.