OPERADORES Y COMANDOS DE R.

OPERADORES BÁSICOS

Nombre	Cómo expresarlo en R
Escribe un mensaje	print()
Repite un valor en un vector un número determinado de veces	rep()
Concatena valores para crear un vector tanto numéricos como alfabéticos	cat()
Produce un gráfico	plot()
Borra directorios	rm()
Da como número la longitud de un vector	length()

OPERADORES ARITMÉTICOS

Nombre	Cómo expresarlo en R
Suma	+
Resta	-
Multiplicación	*
División	/
Potencia	^
División entera	%/%
Multiplicación matricial	%*%
Resto	%%
Aplica el valor absoluto a un número o vector	abs()

OPERADORES LÓGICOS

Nombre	Cómo expresarlo en R
Y	&
Y (vectores)	&&
O	|
O (vectores)	||
No	!

OPERADORES RELACIONALES

Nombre	Cómo expresarlo en R
Menor que	<
Mayor que	>
Distinto que	!=
Igual que	==
Menor o igual que	<=
Mayor o igual que	>=

SEGÚN TIPO DE DATOS

Nombre	Función que realiza	Cómo expresarlo en R
Integers	Solo lee valores que sean números enteros	as.integer()
Numerics	Admite cualquier número (entero o decimal)	as.numeric()
Strings	Admite valores no numéricos como palabras	as.character()

SOLVE

Este comando se usa para calcular la inversa de una matriz. Este comando es útil para ejercicios de interpolación

Si, por ejemplo estamos realizando un código para el polinomio interpolador, y queremos calcular los coeficientes:

a = solve(M)%*%b

%*% se usa para multiplicar matricialmente la inversa de M y b

b es un vector o matriz de términos independientes

UNIQUE

Este comando obtiene los valores únicos de su argumento (unique (x)), eliminando además los duplicados. Es decir, solo devuelve los elementos no repetidos. Se puede usar en vectores, listas… Si se quiere que el resultado tras aplicar unique sea un vector, se debe poner el comando c después de unique: unique(c(….)).

Por ejemplo:

```
edades_de_la_familia = c (23, 45, 21, 67, 67, 21)

vector = unique(edades_de_la_familia)

print(vector)

## [1] 23 45 21 67
```

BREAK

Se utiliza para interrumpir un bucle. Para ello, necesitamos que se cumpla una condición. Cuando esto ocurre, el bucle se detiene, aunque existan elementos a los cuales aún podría aplicarse.

Por ejemplo:

Interrumpimos un for cuando i es igual a 4, aunque aún queden 3 elementos en el objeto.

```
for(i in 1:7) {
  if(i == 4) {
    break
  }
  print(i)
}

## [1] 1

## [1] 2

## [1] 3

## [1] 4
```

NEXT

Sirve para saltar una iteración en un bucle cuando se cumple una condición.

Por ejemplo:

Cuando i toma el valor 4, el for salta la iteración y sigue al siguiente, es decir, al valor 5, y así sucesivamente.

```
for(i in 1:7) {
  if(i == 4) {
    next
  }
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 5
## [1] 6
## [1] 7
```

REPEAT

Este es un bucle que se llevará a cabo el número de veces que especifiquemos, usando un break para detenerse. Si no incluimos un break, el bucle se repetirá indefinidamente y sólo lo podremos detener manualmente.

La estructura de repeat es el siguiente:

```
repeat {
  operaciones
#Si queremos detener el programa usamos break
  break
}
```

Por ejemplo:

El repeat sumará 1 a la variable valor hasta que este sea igual a cinco, entonces se detendrá.

```
valor <-  0
repeat{
  valor <- valor + 1
  if(valor == 5) {
    break
  }
}
print(valor)
## [1]  5
```

LENGTH OUT

Este comando sirve para equiespaciar una muestra previamente proporcionada en un intervalo determinado.

Por ejemplo:

```
muestra1 = seq(3, 3.5, length.out=3)

# creará una secuencia de 3 valores equiespaciados en el intervalo proporcionado.

## [1] 3  3.25  3.5
```

SAPPLY

Este comando es extremadamente útil cuando queremos aplicar una función a un elemento de un objeto como un vector o una matriz, devolviendo resultados simplificados cuando sea posible.

Por ejemplo:

```
muestra1 = seq(3, 3.5, length.out=3)

muestra2 = seq(3.5, 4.5, length.out=3)

muestra_total = unique(c(muestra1, muestra))


apartado_b = function(x) {
  sumatorio= 0
  for (k in 1:5){
    sumatorio = sumatorio + (((-1)^k)/factorial (k)*x^k*sin(k*x))
  }
}

eval_datos = sapply(muestra_total, apartado_b)

print (eval_datos)
```

SEQ

Crea una lista de números con la capacidad de establecer rangos personalizados. Su estructura general puede ser: seq(from, to, by, length.out).

from: El valor del que se parte

to: El último valor.

by: El incremento de los valores.

length.out: La cantidad total de valores equiespaciados en la secuencia (ignora by).

Por ejemplo:

seq(from = 1, to = 21, by = 5)

[1] 1 6 11 16 21

SEQ_ALONG

Este comando genera una secuencia de valores enteros empezando en 1 hasta la longitud del argumento introducido entre paréntesis. No se pueden añadir ningún parámetro más, gran diferencia con el comando seq.

Además, es una buena alternativa para usar en un bucle for, por ejemplo. En vez de poner for (i in 1:vector) puedes sustituirlo por for (i in seq_along (vector)).

SEQ_LEN

Es muy similar a seq_along, ya que crea una secuencia de valores desde 1, pero genera la secuencia hasta el valor indicado entre paréntesis. La diferencia con seq_along es que este crea una secuencia basada en la longitud de un objeto, mientras que en seq_len introduces un valor específico entero. Seq_len te permite crear un vector vacío también: seq_len (0). Por ejemplo:

ejemplo <- seq_len(3)

[1] 1 2 3