INTRO

La consola es un sistema REPL: Read Evaluate Print Loop.

Un sistema REPL espera que el usuario le dé una instrucción y presiones ENTER. El intérprete entonces lee (READ) el comando, lo evalúa/corre/ejecuta (EVALUATES), lo imprime por pantalla (PRINT) y luego vuelve al comienzo a esperar por otra instrucción (LOOP).

Veamos como funciona el sistema REPL de la consola con nuestro primer tipo de datos **Los números**.

`2 + 3`

`3 * 5 + 4`

`3 * (5 + 4)`

`-300 * 2.57`

`25 % 6` (devuelve el resto de la división de 2 números)

`1e4 * 2`

¿Qué aprendimos? Podemos usar la consola como una calculadora simple si usamos valores con tipo de datos numéricos junto con operadores matemáticos como +, -, *, /, %.

Como vimos los números son un tipo de dato primitivo de JavaScript. Además de los números hay cuatro más que aprenderemos. Strings, Booleanos, null y undefined.

Los strings son básicamente texto. Y lo importante es que SIEMPRE están dentro de comillas.

Podemos concatenar string utilizando el operador de suma (+).Si queremos representar texto sin ponerlo dentro de comillas nos dará un error, ya que la consola pensara que le queremos dar una instrucción que no tiene definida.

`"Hola como estas"`

`"Hola"+ "como" + "estas"`

`"Hola "+ "como "+ "estas"`

`Hola como estas`

`"5"+"5"`

`"5"+5`

`"5+5"`

`"Tengo "+ 22+ " años"`

¿Podemos entender la diferencia de sumar 5+5 y "5"+"5"? Cuando sumamos números javascript los interpreta como tal y los suma, pero cuando utilizamos strings numéricos, javascript los lee como texto, y no los interpreta como números.

Cuando tenemos el string "5+5" pasa lo mismo. Javascript no interpreta el signo + como un operador de suma sino como solo un carácter del texto.

Variables

Pensemos a una variable como un contendor, que adentro de ese contenedor se encuentra un valor.

```
Var MiNombre = “Francisco”
```

Null and Undefined

Null y undefined, los dos son valores que podemos asignar a nuestras variables, y ambos significan nada, pero con distintos sentidos.

Por ejemplos si declaramos una variable, pero no le asignamos un valor, por defecto va estar en undefined, como dice el nombre sería una variable sin definir.

```
var age
```

```
var name
```

La diferencia con null es un tema conceptual, en undefined, estamos diciendo que la variable aún no fue definida, ósea que no tiene un valor por ahora. Con null, la variable esta explícitamente vacía.

Booleanos y Comparadores

> El Tipo de datos Booleano tiene dos posibles valores: verdadero o falso. Podemos escribirlos literalmente. Escribí lo siguiente en la consola:
>
> ```
> true
> ```
>
> ```
> false
> ```
>
> ... Pero comúnmente nos lo encontramos como producto del resultado de una expresión. Escribe las siguientes expresiones, compuestas de valores y de operadores, y determina qué valores **booleanos** retornará cada una:
>
> ```
> 5 > 3
> 5 < 3
> 5 >= 5
> 10 <= 100
> 5 === 5
> 5 !== 5
> ```

`5 === "5"`
`5 == "5"`

Un dato de tipo string y otro de tipo number nunca serán realmente equivalentes, sin importar cuanto se asemejan al mirarlos. "5" es un texto con el número cinco, y 5 es el número cinco, no son lo mismo.

Ojo con esto:

Comparar usando doble igual (==): importa en contenido y no el tipo de dato. Por ejemplo, si comparamos 5 == "5" el resultado es **true** a pesar de que uno sea número y el otro string. Podríamos decir que es una comparación "blanda". Lo que ocurre en realidad es que JavaScript hace lo que se llama *coerción de datos*, es decir que primero transforma el número en string (haciendo que los dos elementos sean strings) y después los compara.

Comparar usando triple igual (===): importa el contenido **y** el tipo de dato. Comparar 5 === "5" da **false** porque JavaScript no hace la *coerción de datos*, osea que no trasforma el número a string, y sabemos que el número cinco (5) es diferente al string cinco ("5"). Podríamos decir que es una comparación "dura" porque compara tanto el contenido como el tipo de dato. **Intenta siempre usar el triple igual, la coerción de datos, puede traer varios problemas a la larga.**

Asumiendo que `var x = 5`; `var y = 9`

Operador	Nombre	Ejemplo	Resultado
&&	Y lógico	x < 10 && x!==5	false
||	Ó logico	x>9 || x===5	true
!	Negador	!(x===y)	true

Para que && devuelva true, TODAS las expresiones deben ser verdaderas

Para que || devuelva true, alcanza con que solo una sea verdadera.

! niega el valor booleano, ósea que si algo es true, devuelve false, si algo es false, devuelve true.

Condicionales

Las condicionales nos permiten tomar decisiones en nuestro código. Un ejemplo práctico sería cuando un usuario se loguea, si el usuario y la contraseña que puso el usuario en el formulario matchean con el de la base de datos, lo redirige a la homepage, pero si no matchean, envía un mensaje de error, por lo tanto, hay dos caminos posibles, en un mismo código.

Para entender cómo funcionan las condicionales resolvamos el siguiente problema:

Queremos crear un programa que decide si una persona puede entrar un bar, y queremos crear tres caminos posibles.

1. Si el usuario es menor de 18 años no lo dejaremos ingresar.
2. Si el usuario esta entre los 18 y 21, puede ingresar, pero se le debe advertir que no puede tomar alcohol.
3. En el último caso, ósea q es mayor a 21 años, puede ingresar tranquilamente

Para crear esta lógica lo primero q vamos a ver es la sentencia if, esta ejecuta una serie de líneas de código (bloque de código) en función de una condición booleana especificada que se evalúa como verdadera o falsa.

La sentencia **if** toma dentro del paréntesis una condición que evaluará, si esta condición es verdadera, ejecutará su bloque de código entre las llaves, sino no hará nada.

```
if(edad < 18){
  console.log("No puedes ingresar al bar!")
}else if(edad < 21 ){
  console.log("Puedes ingresar, pero no se te permite consumir alcohol")
}else{
  console.log("Adelante, eres bienvenido!")
}
```

While

El while loop se parece a la declaración del **if.** Los dos ejecutan un bloque de código asociado dependiendo del resultado de evaluar su expresión condicional. La diferencia es que el while loop re-evalúa repetidas veces su expresión condicional y continúa ejecutando su bloque de código mientras que ésta resulte ser verdadera

```
while(unaCondicion){
  //corre algun codigo
}
```

Entonces mientras la condición del while loop siga siendo verdadera, este seguirá ejecutando su código, aquí hay un ejemplo de cómo podemos imprimir los números del 1 al 5

Helpers

IF

```
If(condicion) {
        //codigo si condicion es verdadero
} else{
        //codigo si condicion es falso
}
```

While

```
While(condicion){
        //codigo que será ejectutado hasta que condicions sea flase
}
```

Prompt

Función que permite ingresar datos con el teclado

propmt()

Console.log

Permite imprimir en pantalla un valor que va entre paréntesis console.log()