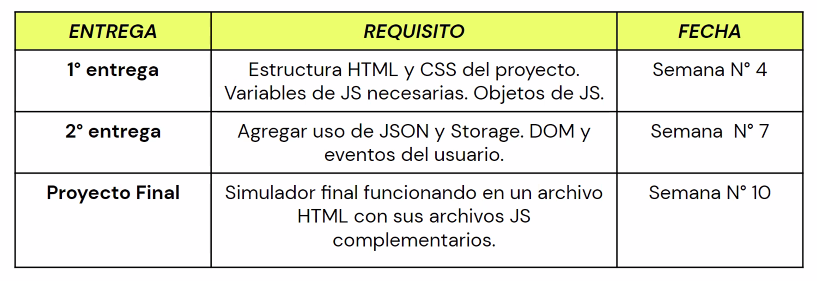
**JAVASCRIPT**

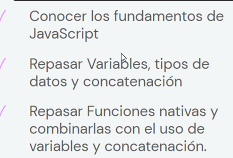


* Para la segunda entrega vamos a necesitar que haya un HTML un poco más estructurado porque necesitamos trabajar con eventos, es decir, ver lo que pasa cuando el usuario hace clic por ej en un botón

IDEAS

- Quiz App- Aplicación de notas- E-commerce- Aplicación de pedidos de comida- Aplicación de películas- Aplicación de chat- Clon de Twitter- Aplicación de encuestas- Clon de Whatsapp- Clon de Netflix- Aplicación de búsqueda de trabajo- Clon de Pinterest- Aplicación de Stock- TO-DO list- Aplicación de reserva de hoteles- Conversor de divisas

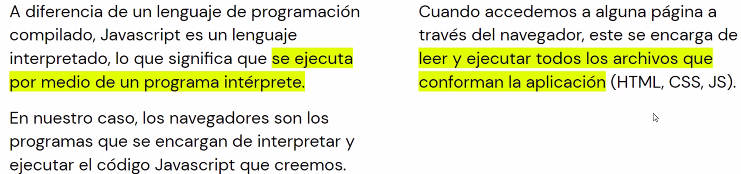
**CONCEPTOS**

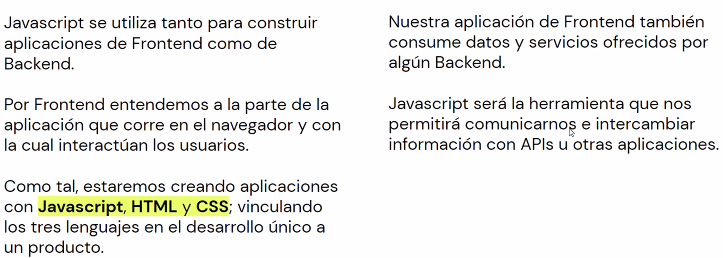


Fundamentos de JavaSript

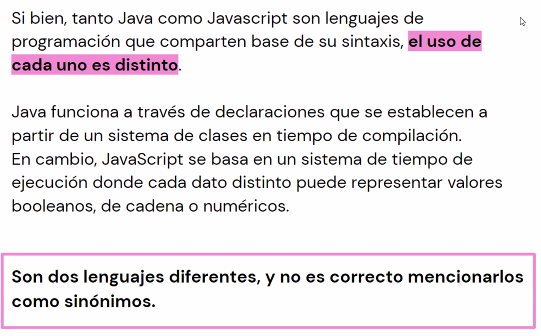
* Aporta dinamismo a sitios y aplicaciones web: pedir información por ej
* Se complementa con HTML y CSS
* Busca **interactuar** con otras aplicaciones o servicios de backend

**Es un lenguaje interpretado**: necesita de un programa para que lo interprete, no es un lenguaje compilado. El navegador es quien va a leer y ejecutar JS.





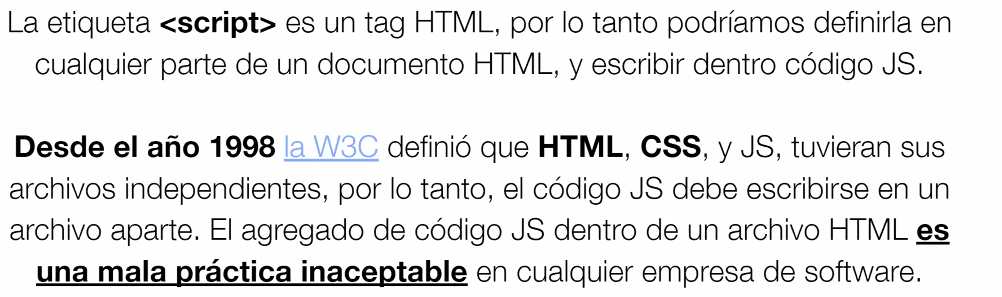
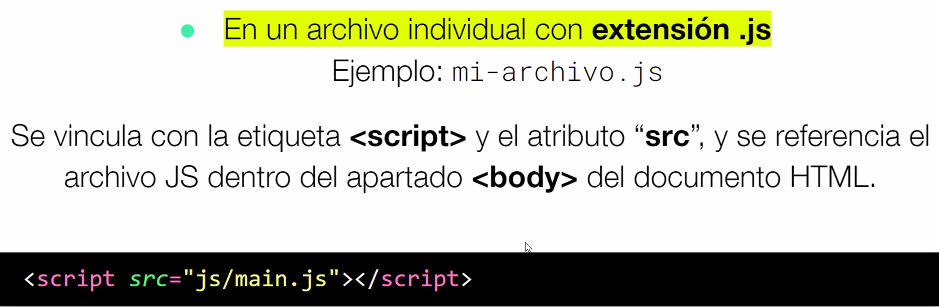
**Diferencia con Java**



* No se usan para lo mismo
* Java es un lenguaje de compilación y JavaScript de interpretación
* JavaScript es débilmente tipado: que los variables no tienen que ser declaradas con el tipo de dato. Si declaro una variable “nombre” no tengo que indicarle el tipo de dato que va a contener
* JS se ejecuta del lado del cliente, usuario, navegador y J se ejecuta del lado del servidor, en el lado del backend

**SINTAXIS Y VARIABLES**

Tenemos 2 maneras de escribir JS en nuestro código

1. 
2. 

Nosotros usamos la segunda forma. La etiqueta script la ponemos al final del código HTML, antes de cerrar la etiqueta body. Esto es para que primero se ejecute todo el HTML y luego ejecute el archivo JS.

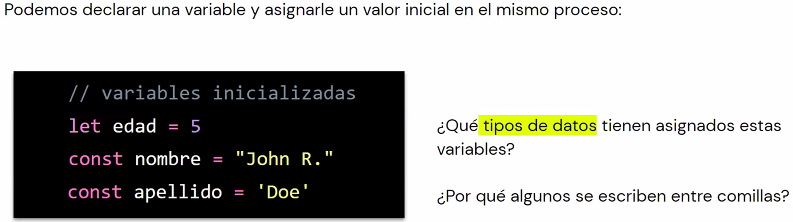
**¿Cómo se escribe?**

* Acostumbrarse a cerrar cada línea con un ;
* Es una buena práctica

**VARIABLES Y VALORES**

**Variables**: es un espacio reservado en la memoria que puede cambiar de contenido a lo largo de la ejecución de un programa. Para guardar un tipo de dato. En las variables almacenamos diversos tipos de datos que utilizamos en la aplicación.

**Inicializar variables:**



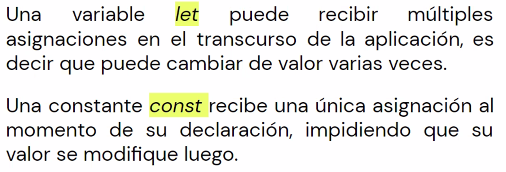
Para declarar una variable uso let. ¿Qué va a guardar la variable? Se lo indicamos con = y luego “”

Pueden declararse variables sin ningún valor, pero ya se creó esa variable y se reservó.

Extensión error lens + live server

* Uso comillas para declarar valores string
* No uso comillas para declarar valores numéricos

**Diferencia entre let y const**

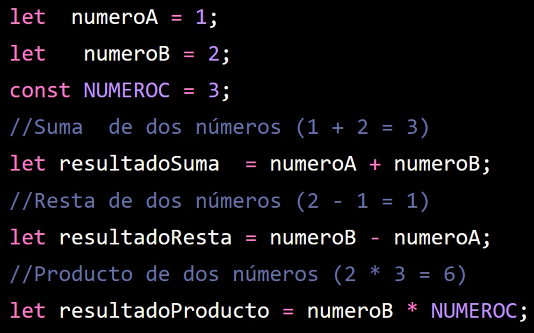


Const: es algo que no cambia a lo largo del código, su valor no se modifica.

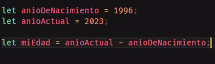
* Cuando declaro una variable con const, tengo que asignarle un valor SI O SI.

Ej: ¿por qué hay veces que determino variables sin valor? Porque por ej cuando una persona compra un curso en coderhouse todavía no le asigné valor al curso, pero cuando lo compra la variable se pasa a llamar *curso = “javascript”* por ejemplo

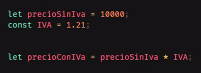
**Operaciones básicas**



Otro ejemplo:



Otro ejemplo: quiero calcularle siempre al precio de mi producto el IVA. Declaro la variable del precio con un valor y la constante del IVA porque es un valor que siempre queda asi.

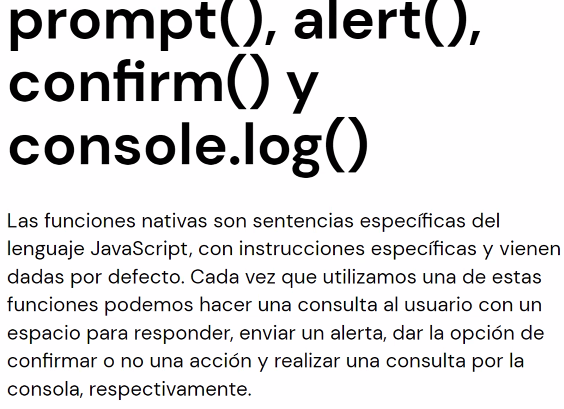


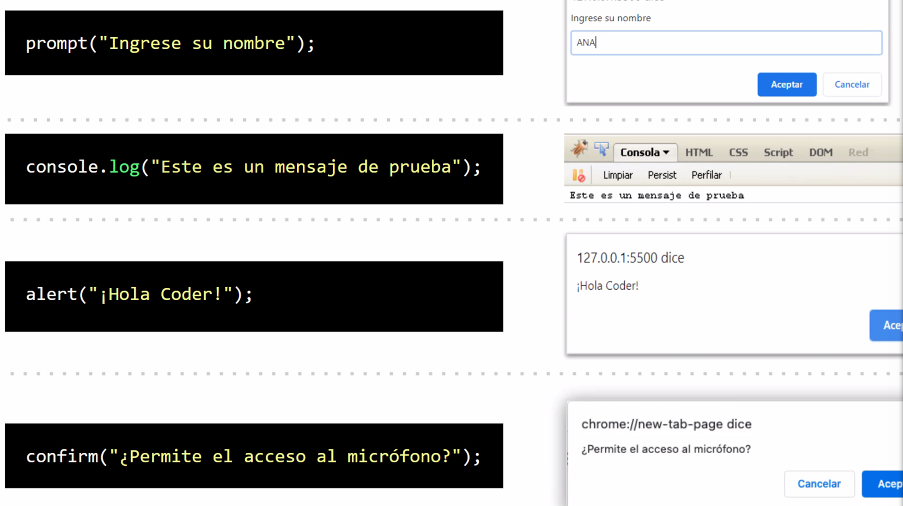
Puedo hacer concatenaciones de dos variables



**Funciones nativas**

Diferentes formas de recibir o mostrar información





* Promt: el usuario ingresa un dato
* Console.log: muestra la información por consola
* Alert: muestra la información por pop up
* Confirm: es un pop up para aceptar o cancelar

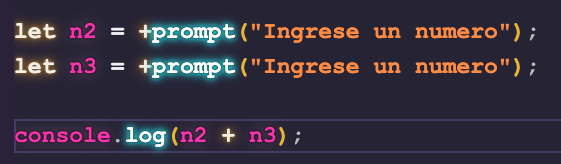
**MICRODESAFÍO**

**parseInt** delante del promt para que el valor que se ingrese lo transforme a tipo de valor entero. El promt solo es para recibir valor de string

**parseFloat:** parcear a tipo decimal

number: lo transforma a tipo de dato numérico

 Otra forma de escribir



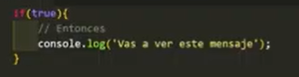
5/2

**CONTROL DE CICLOS**

**Condicionales**: estructuras sintácticas que nos permiten evaluar una cierta condición, algo, una expresión para determinar si es falsa o verdadera y elegir un curso de acción. Es un bloque de código que nos permite elegir un curso de acción.

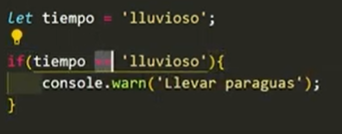
Al evaluar condiciones, lo que recibimos, el resultado, es una **expresión booleana**, es TRUE o FALSE.

**Estructura de una expresión condicional:**

* 
* 

Operadores de comparación:

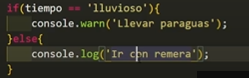
**Operador ==**



Si la condición no se cumple, no se va a ejecutar.

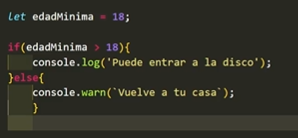


**ELSE**: si el tiempo es lluvioso, pero si no es lluvioso, ir con remera.

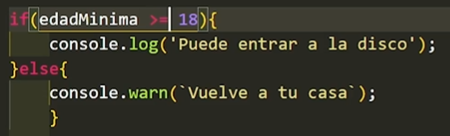


**Operador < e >**

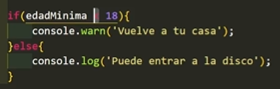
Si la edad mínima es mayor a 18, puede entrar al boliche, sino no podes salir



**Operador >= o**

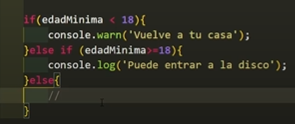


**Operador <=**



Agregar una segunda condición:

**ELSE IF:** sirve por si necesitamos agregar mas condiciones al código. El else if siempre lleva condición.



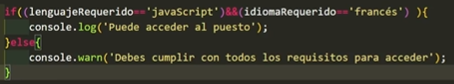
Condiciones simples: se debe cumplir con un requisito

Condiciones complejas: se debe cumplir con dos o más requisitos

**Operadores lógicos**

Situación: tenemos una empresa donde nos vamos a postular una vacante. Nos piden dos requisitos excluyentes. Nos piden un lenguaje de programación y tener un idioma.

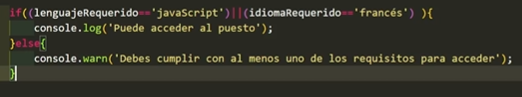




* **&&:** para expresar que tiene que se tienen que cumplir dos condiciones

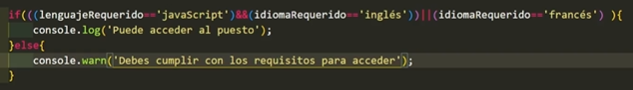
Situación: ahora la empresa, debido a la falta de candidatos, dice que de los dos requisitos el postulante al menos debe cumplir uno.

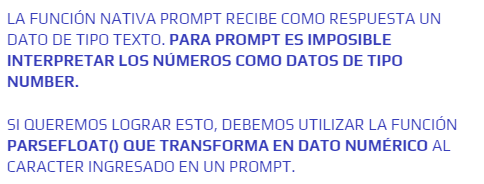




* **||:** operador or. Si alguna de las dos condiciones es real, podes postularte.

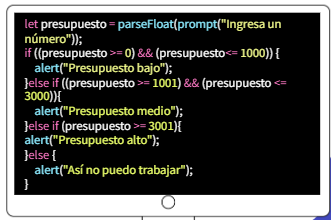
Situación: que sepa JS y francés o que por lo menos sepa inglés.





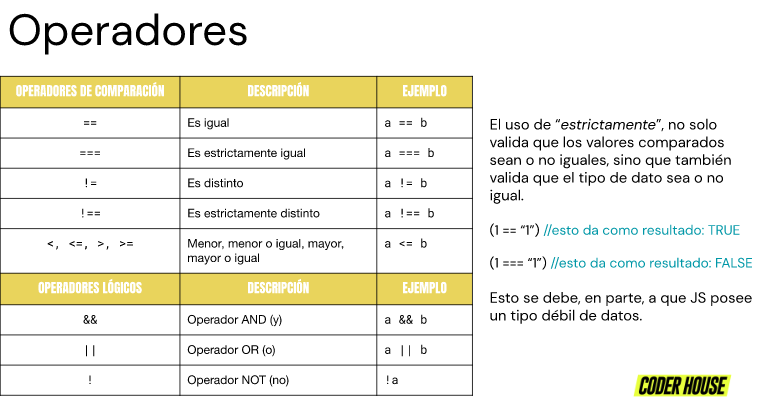
* TRANSFORMAR A MAYUSCULA: **toUpperCase**
* TRANSFORMAR A MINÚSCULA: **toLowerCase**

¿Cómo harías para corrobora si el presupuesto que te pasan para contratar un compañero es bajo medio o alto?

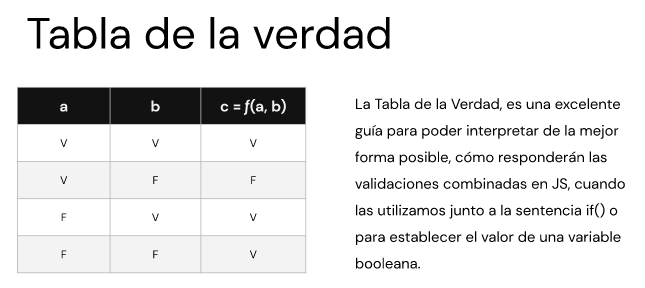


**RESUMEN OPERADORES**





**TABLA DE LA VERDAD**



**CICLOS**

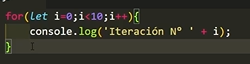
Los ciclos son un medio rápido y sencillo para realizar una misma instrucción muchas veces.

Ciclo **FOR**: ciclo denominado por conteo o determinado. Antes de que empiece ya sabemos cuantas veces se va a repetir el bloque código que esta adentro.

Estructura sintáctica:

* **For**(desde; hasta; actualización)
  + **Desde**: donde va a iniciar la iteración
  + **Hasta**: hasta cuantas veces se va repetir este bucle
  + **Actualización**: es la parte que hacer variar el valor de la variable para que el ciclo avance y el ciclo no se quede trabado

Ejemplo:



La iteración empiece en 0, hasta que la i sea menor a 10, ¿qué va a decir nuestro consolelog en nuestra última iteración?

 en total hacen diez vueltas del ciclo.

* Si tengo 3 manzanas, la primera manzana ocupa la posición 0



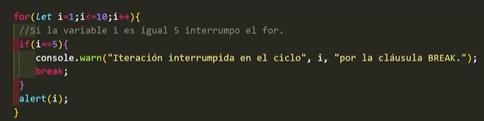
**i++:** usamos esta variable como un incremento, es decir, que en cada vuelta se sume una unidad al valor de la variable i.

**Sentencia break**: hace que bajo cierta condición el ciclo se interrumpa. Cuando necesitamos que el código se interrumpa antes de cumplir con todas las iteraciones.

Ej: cuando entramos a una página y nos loggeamos y nos dejan solamente 3 intentos para poder poner bien la contraseña. Si ponemos la contraseña bien la primera vez, ya está, no tenemos que volver a pedirle al usuario que ingrese la contraseña correcta.

Llevado a un ejemplo:

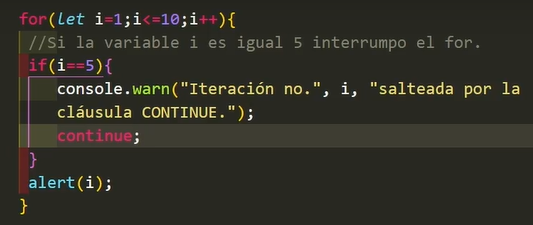
Cuando declaramos el for y queremos determinar una sentencia break, tenemos que agregar un condicional.

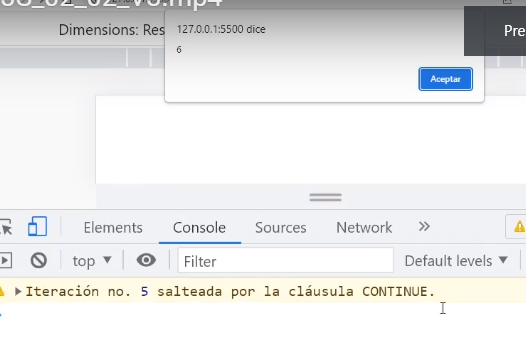


Si el valor de i es == interrumpo el ciclo for, sino, que me muestre el valor de i.

**Sentencia continue**: hace que el ciclo saltee una repetición. A veces queremos que cierta iteración del ciclo se saltee, pero el ciclo no termine, sino que siga ejecutándose. En vez de usar break que finaliza el ciclo for, vamos a usar la sentencia break.

Ej:



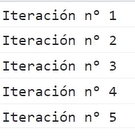


El uso de los ciclos nos permite tomar información y poder filtrarla, elegir y ejecutar diferentes tipos de bloques de código en diferentes condiciones.

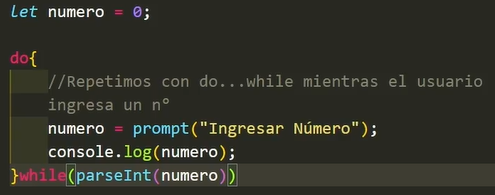
Ciclo **WHILE**: se recorre de forma lineal, pero le podemos agregar una interrupción en el medio.

Ejemplo



Mientras el valor de la variable i sea menor a 6, se tiene que imprimir en consola la frase definida más el valor que adopte la variable i. 

**Do While**: crea un bucle que ejecuta una instrucción específica hasta que la condición de prueba se evalúa como falsa.

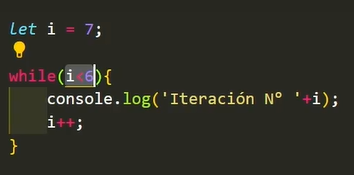


En este caso le estamos pidiendo que haga algo y que se ejecute siempre y cuando lo que el usuario ingrese sea un número, por eso usamos el ParseInt. Si el usuario ingresa un dato que no es un número, el parseInt va a devolver un FALSE y en ese caso vamos a terminar una la ejecución del bucle while.

Lo probamos y cuando pusimos una a en el promt se terminó el bucle, pero se mostró en consola la letra a.

**Esto sucede porque el bucle do while se asegura de al menos una vez se ejecute el bloque de código antes de evaluar la condición y siempre la ejecución del bloque de código sucede antes de evaluar la condición. A diferencia de lo que pasa con el bucle while donde primero evalúo y después ejecuto.**

*¿En qué momento podría pasar que el bucle do while se ejecute cero veces?*

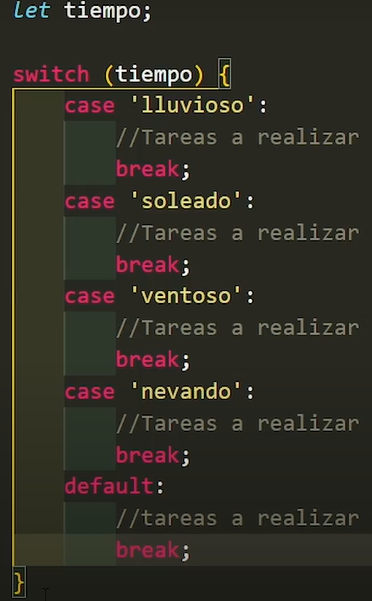


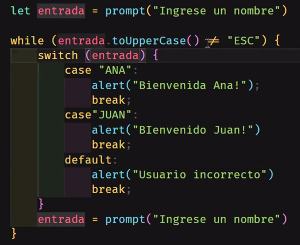
En este caso, se lee que i vale 7 por lo cual ya es mayor a 7, entonces no lo va a ejecutar porque en la condición le estamos pidiendo que lo ejecute siempre y cuando i sea menor a 6. Esto sucede porque la ejecución del while se hace antes de la ejecución del bloque de código.

**Ciclo Switch**: sirve para poder escribir diferentes bloques de código if anidados. Nos permite evaluar una misma variable que puede tener muchos valores diferentes.

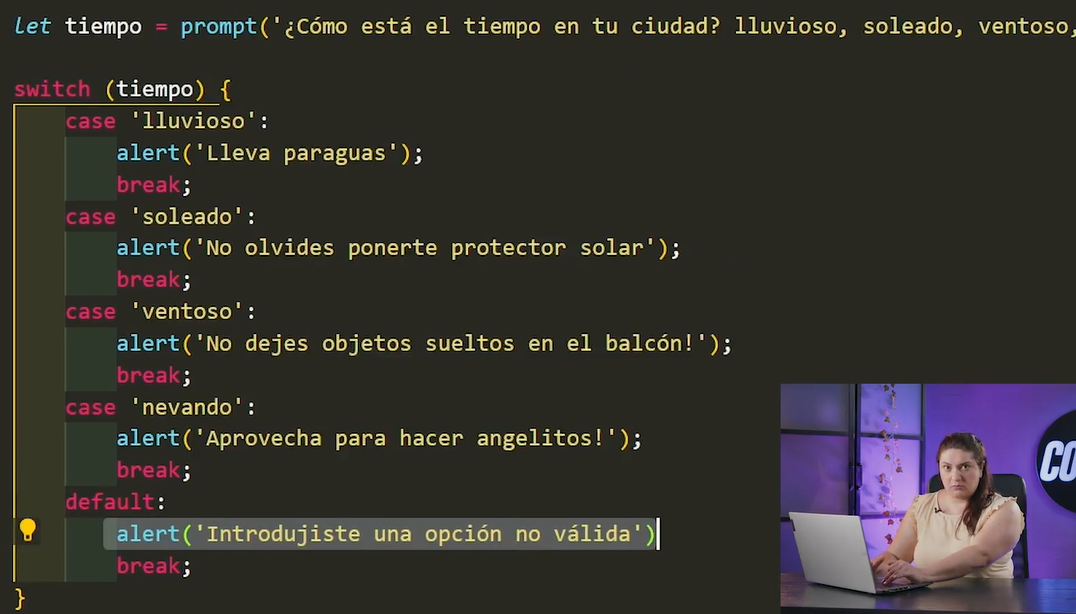
* Cases: cada una de las posibilidades que contempla la estructura de control switch.

Ej:



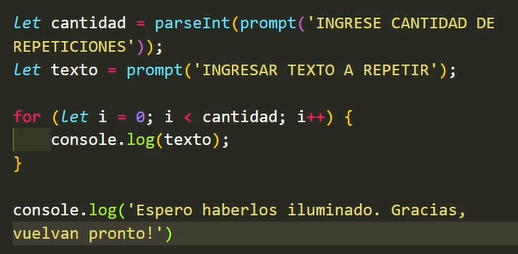


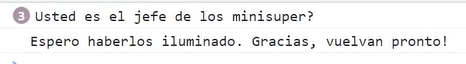
Ej: en este caso le estamos pidiendo al usuario que declare el valor de la variable mediante un prompt.



El valor de la variable debe coincidir exactamente con el nombre del caso indicado.

Ej de ciclo for:



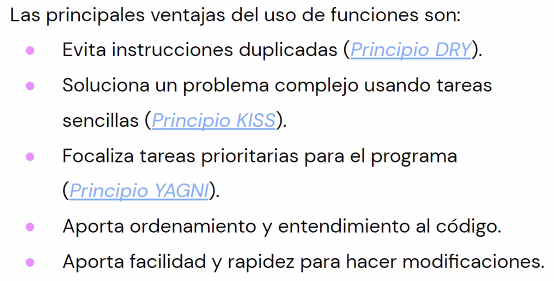


**FUNCIONES**

Las funciones son un conjunto de instrucciones para realizar un procedimiento. Sirven para resolver un problema o desarrollar llevar adelante cierta tarea. Su particularidad es que están **auto contenidas**.

**Auto contenidas**: estas funciones van a estar contenidas dentro de un bloque particular de código y nosotros vamos a poder llamarlas las veces que queramos y en el momento en el que las necesitemos.

Principales ventajes del uso de funciones:

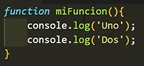


Ej: agregar un producto al carrito, inicio de sesión.

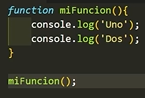
Las funciones son ciudadanos de primera clase: con las funciones podemos pasarlas como parámetro, podemos pasarlas como argumento, podemos declararlas como valores, podemos hacer lo que se nos ocurra.

**Declarar una función**: simplemente la estamos creando.

* **Funciones declaradas:**



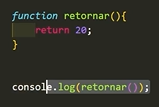
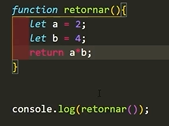
**Para ejecutarla, llamarla**: nombrar a la función poniendo los paréntesis al final



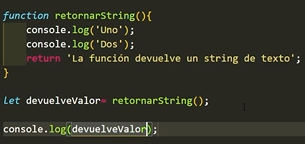
**Importante**: a las funciones tenemos que darles un nombre descriptivo de lo que la función esta haciendo. Toda función lleva adelante una acción y debe representarse con un verbo.

Ej: función sumar, función restar.

**Función retornar**: podemos hacer con la palabra reservada **return** que devuelva un valor en particular o podemos hacer que devuelva el valor de una operación.

**También podemos hacer que una función este contenida dentro de una variable.**



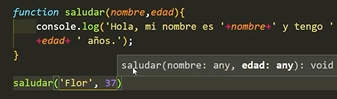
1. Declaro una función con lo que quiero que se ejecute
2. Dentro de la función declaro un return
3. Por fuera de la función, declaro una variable con un nombre que sea igual a la función
4. Por último, muestro por un console.log el valor de la variable que va a devolver el valor de la función.

Lo que le decimos a la variable es eu guardame está función adentro, cuando te llamen, cuando te quieran utilizar, se va a ejecutar la función y el valor de retorno que es esta cadena de string de texto es lo que se va a mostrar.

Lo que se guarda en la variable es el return de la función.

**PARÁMETROS**

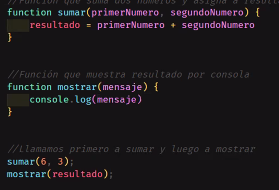
**También podemos hacer que una función reciba valores que se conocen como parámetros.**



Cuando creamos la función, le estamos diciendo qué es lo que necesitas para funcionar. Cuando llamemos la función, le vamos a decir este es el valor que quiero que uses en esta llamada.

Parámetros: datos que va a necesitar la función para poder ejecutar su tarea. Son valores que vamos a necesitar que la función utilice en sus operaciones.

Cuando la ejecuto, la llamo y entre los paréntesis pongo los valores que quiero que sean los parámetros de esa función.

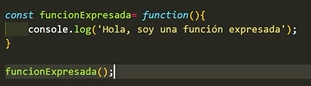


* **Funciones expresadas:**

No necesitan tener un nombre porque las vamos a guardar dentro de una variable y van a ser invocadas cuando llamemos a una variable.

Crear una constante a la que vamos a llamar funcionExpresadas y le asignamos la palabra clave function () pero no le ponemos nombre justamnte.

Cuando la quiero ejecutar, llamo al nombre de la variable.



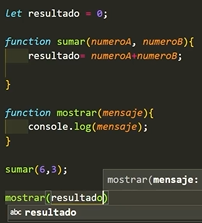
**Diferencia entre funciones declaradas y funciones expresadas**: si creamos una función declarada, una función con nombre, pero la ejecutamos antes dentro de las líneas de código, aparece primero la ejecución y luego la declaración, va a funcionar bien.

Pero ¿qué pasa si intento ejecutar una función expresada antes de guardarla en mi variable? Me salta un error de que no puede ejecutar algo que todavía no se ha inicializado, porque lo estamos guardando en un variable.

**Tres metodologías para trabajar con funciones**:

1. DRY: no te repitas a ti mismo. Si estoy escribiendo una tarea que se va a repetir 25 veces, genial, creo una función para esa tarea.
2. KISS: mantenlo simple. No hagamos un código muy complicado, desarmemos nuestros problemas grandes en problemas mas pequeños. Creemos funciones para cada uno de los problemas pequeños.
3. YAGNI: no lo necesitas, bórralo. No escribas código para algo que todavía no tenes que codificar. Primero nos enfocamos en las funcionalidades principales del problema.

Ej:



* Creo a la función, le digo vas a tener estos parámetros y cuando la llamo le digo qué valores tiene que tener cada parámetro.

**SCOPE EN JAVASCRIPT**

Es el contexto actual de ejecución. Es el lugar donde las variables pueden ser visibles y referenciadas.

Existen dos tipos de scope:

* **Scope local**: acceso por una parte del programa.
  + Scope de función: hay variables que están definidas dentro de una variable. Solo puede ser referenciada dentro de la función.
  + Scope de bloque: se limita a los bloques de código, por ej, dentro de una estructura if. Dentro de estos bloques las variables van a tener un scope local también y van a poder ser referenciadas dentro de este bloque.
* **Scope global**: acceso desde cualquier lugar del programa. Va a poder ser referenciada, accedida, modificada desde cualquier parte de nuestro código, cerca o lejos de ella.

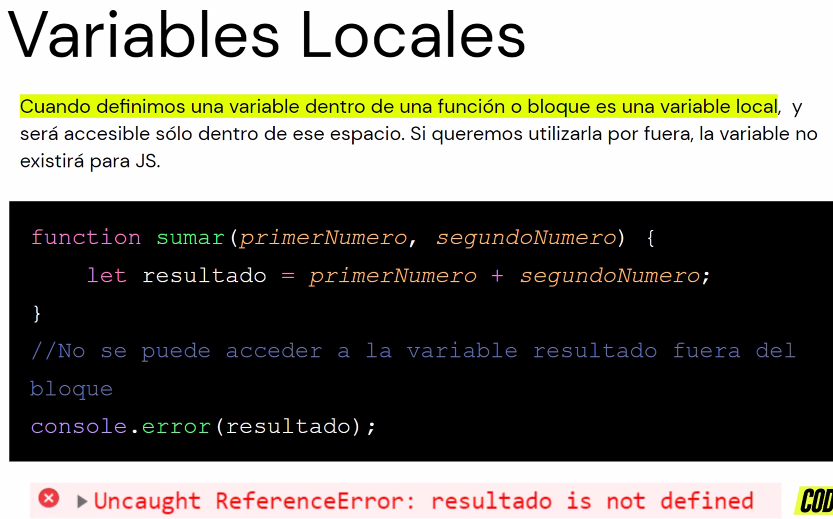
Cualquier variable declarada con la palabra reservada **var** dentro de una función va a tener scope de función. Independientemente de en qué parte de la función esté declarada.

**Dentro del scope de bloque, todas las variables tienen scope de bloque, salvo las variables que arrancan con la palabra reservada var.**

* Siempre es mejor trabajar con un scope lo mas reducido posible, para que sea mas difícil que se generen o resolver los bugs.

Jerarquía del scope: nos ahorra tiempo, uso de la memoria, ahorra errores.

Ej: queremos salir a pasear, pero si o si antes hay que ir a mandar el auto. **Agregamos otra función dentro de la función pasear.**

****

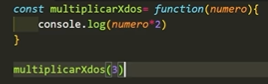
****

Yo puedo tomar una variable global y redefinirle el valor de la misma dentro de una función.

**FUNCIONES ANÓNIMAS Y FUNCIONES FLECHAS**

**Función anónima:** es una función que se define sin nombre y se utiliza para ser pasada como parámetro o asignada a una variable. Nos permite mantener un código ordenado y de que no ejecutemos la función antes de declararla. Generalmente, las funciones anónimas se asginan a variables declaradas como const.

1. Declaramos una variable
2. Le asignamos el valor de una función a la variable
3. Declaramos lo que va a hacer la función
4. Para ejecutarla, llamamos a la variable con el parámetro



* La función no necesita tener un nombre asignado porque cuando queramos ejecutarla vamos a llamar a la variable.

*multiplicarXdos(3)*

**Funciones flechas o arrow**: usamos el arrow operator =>. Las funciones flecha tiene el **return implicito**, no necesitamos escribir la palabra return para decir que nos devuelva ese valor.

1. Declaramos una variable
2. Primero la igualamos al parámetro
3. Usamos el arrow operator
4. Entre las llaves, escribimos lo que queremos que haga la función
5. Para ejecutarla, llamamos a la variable con el parámetro

Con esto que te paso entre paréntesis, hace esto.



Para simplificar el código aún más:

* Si tenemos un solo parámetro, no hace falta que lo pongamos entre paréntesis



* Si tenemos una sola tarea para hacer dentro de la función, también podemos prescindir de las llaves.



Otro ejemplo:





**isNaN**: is not a number.

**PRIMERA ENTREGA DEL TRABAJO**

