Declarar una variable

Var number;

Let number;

Const number;

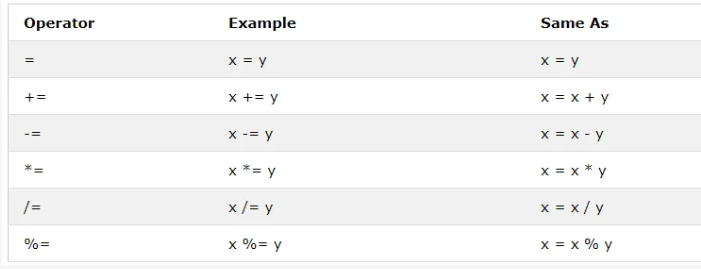
Definir una variable

Var number = 2;

Let number = 2;

Const number = 2;

OPERADORES LOGICOS



CAMEL CASE

La **primera letra** va con MAYUS

Let NOMBREUSUARIOPAGINA 

Let nombreusuariopagina 

Let nombre**U**suario**P**agina 

PARSEINT

Convierte un string o número decimal en un número entero

parseInt(0.5) // 0

parseInt(“2”) // 2

WHILE

Let numero = 0;

While (numero < 6) { // Hasta que la condición no sea false, se sigue ejecutando

Document.write(numero);

numero++

}

DO WHILE

Let número = 0;

// Se ejecuta mínimo una vez y si la condición es true se sigue ejecutando hasta que la condición sea false

Do {

Document.write(numero);

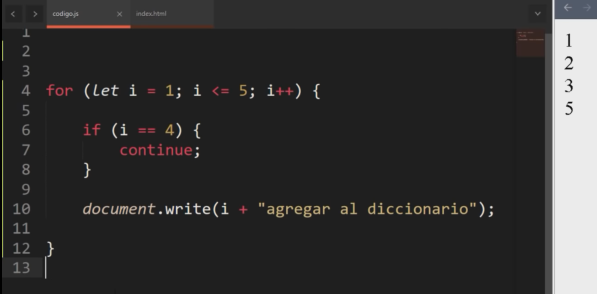
numero++

}

While (numero < 6)

CONTINUE

// El 4 no se ejecuta pero la iteración sigue, continue sirve para no ejecutar ese trozo de código



En este ejemplo, se salta el número 4 ya que i es igual a 4

Break

// El break sirve para cortar la iteración donde se le indique



En este ejemplo, se usa break para terminar la iteración en el número 12

FOR IN

// Muestra y Guarda el **indice** de los elementos… imprime 0, 1 , 2;

Let animales = [“gato”, ”perro”, ”vaca”];

for (animal in animales) {

document.write(animal)

}

FOR OF

// Muestra y Guarda el **valor** de los elementos… imprime “gato”, ”perro”, ”vaca”;

Let animales = [“gato”, ”perro”, ”vaca”];

for (animal of animales) {

document.write(animal)

}

ARROW FUNCTION

Const/Let NOMBREFUNCION = (parametros) => {código}

Const saludar = (nombre) => {

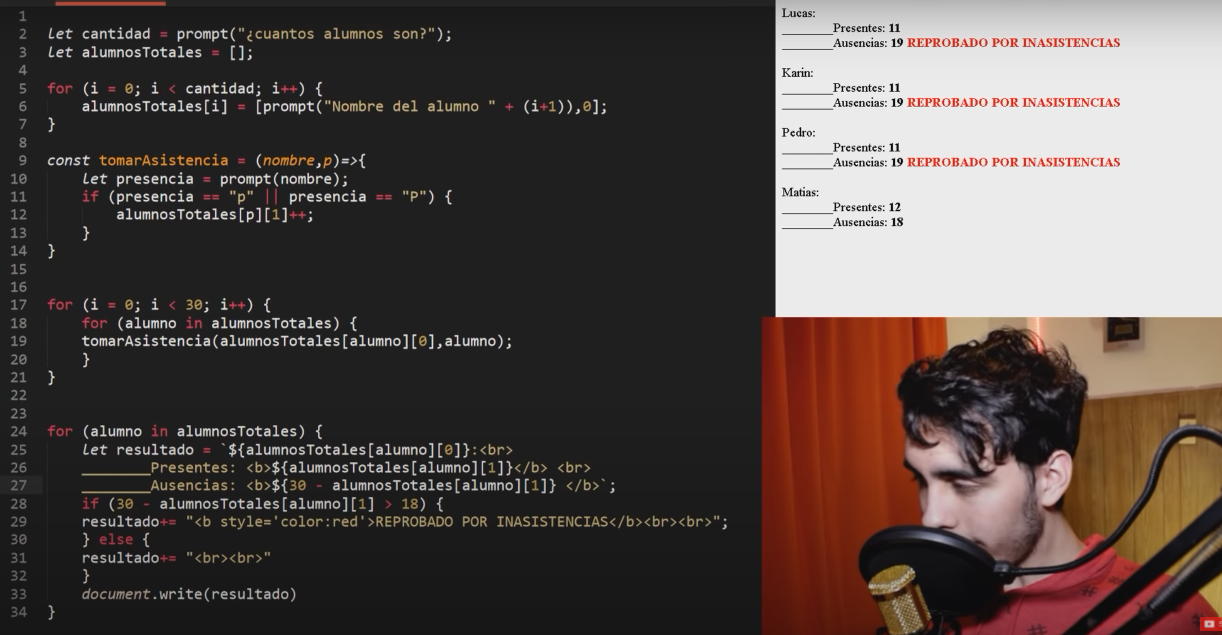
document.write(`Hola ${nombre}`)

};

Const saludar = nombre => document.write(`Hola ${nombre}`);

// De ser un solo param, se puede NO usar los ( ) y si hay una sola expresión/línea de código se puede NO usar las { }

/\*



\*/

HERENCIA

Creamos una clase/class llamada **animal** que tiene varias propiedades, luego creamos una class llamada **Perro** que **hereda** las **propiedades** del padre (En este caso animal) y le podemos agregar la propiedad **raza** (se pueden agregar cuantas propiedades como uno quiera usando **this.ALGO**)

\*Poner **mayus** en la **primera letra** de las **class**



SUPER, SETTER Y GETTER (class)

class Animal {

  constructor(especie, edad, color) {

    this.especie = especie;

    this.edad = edad;

    this.color = color;

  }

}

class Perro extends Animal {

  constructor(especie, edad, color, raza) {

    super(especie, edad, color); // super se usa para dar acceso al constructor y sus funciones

    this.raza = null; // Le agrego una propiedad raza que su valor es nulo

  }

  set setRaza(newName) { // Acá le digo que se puede modificar

    this.raza = newName;

  }

get getRaza() { // Acá le digo que quiero obtener esta propiedad

    return this.raza;

  }

}

const dog = new Perro("can", 7, "negro", "Doberman");

dog.setRaza = "Pedro"; // Acá modifico el valor de raza (raza pasa de ser nulo a ser Pedro)

document.write(dog.getRaza);

Esto al imprimirlo mostraría “Pedro” en vez de “Doberman”, ya que el **setter** se utiliza para **modificar** una propiedad de un objeto y **getter** para mostrarlo (no necesariamente deben usarse juntos)

\* setter modifica el valor de la propiedad de un objeto

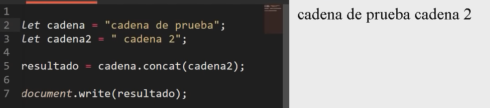
\* getter llama a la propiedad para obtenerla y mostrarla

CONCAT

Concatena (Une/Junta/Agrupa) el primer string con el segundo

**\* Solo concatena strings/cadenas de texto**

// “String” = “Cadena de Texto”



// Otro Ejemplo

numero1 = “53”;

numero2 = 2;

numero1.concat(numero2)

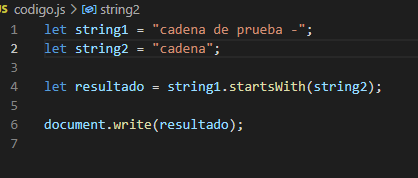
IMPRIME -> 532

startsWith

Sirve para decir si es **true** o **false** que arranca con X cosa (devuelve un valor **booleano** si arranca con “cadena”)

\* string2 arranca con la palabra **“cadena”** tal y como lo hace el string1

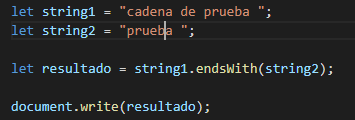
Si al string2 le agrego un espacio al inicio, startsWith() me va a devolver **false**



endsWith

Es lo mismo que startsWith() pero al revés, en vez de donde **Empieza**, es en donde **Termina**

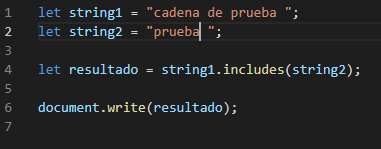
string1 finaliza con “prueba “, por lo tanto devuelve **TRUE,** ya questring2 termina con la misma frase



INCLUDES

Sirve para decir si se encuentra/incluye ese valor

\* En este caso “prueba “ si se encuentra en el string1 y en el string2, por lo tanto va a devolver **TRUE**, si yo pongo “pruebas” me va a devolver **FALSE**, porque “pruebas” no se encuentra en el string2

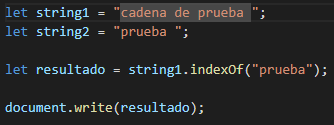


indexOf

Hace lo mismo que includes pero en vez de devolver un valor **booleano**, devuelve en que indice/posición se encuentra lo que se está buscando (¿Se encuentra esto? “SI”, bueno , ¿en qué posición?) (Tambien sirve para arrays)

\* En este caso se está buscando el string **“prueba”** en la variable string1, esto lo que va a devolver es el número 10, ya que el indice de la letra P (de Prueba) es 10

\* En el caso de no encontrar lo que buscamos, este método devuelve -10

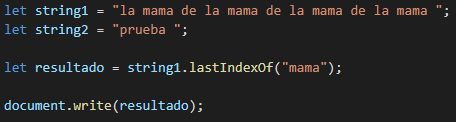


lastIndexOf

Hace lo mismo que indexOf pero este método devuelve donde se encuentra el ultimo string (tambien sirve para arrays)

\* En este caso se está buscando la última vez que aparece la palabra **“mama”** en el string1. El indice de la última vez que aparece **“mama”** es el número 36

\* En el caso de no encontrarse lo buscado, el método devuelve -1

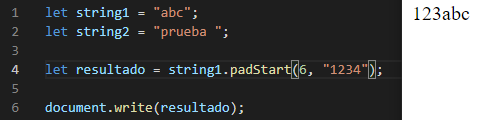


padStart

Hace que la cadena de texto comience luego del string que yo le pido

\* En este caso string1 vale **“abc”** y con el método padStart le estoy pidiendo que la cadena total sea de 6 dígitos, para eso le digo que rellene con **“1234”** hasta que se suficiente para rellenar con **“abc”**. Va a poner **“1234”** las veces que sean necesarias para que la cadena total tenga 6 dígitos

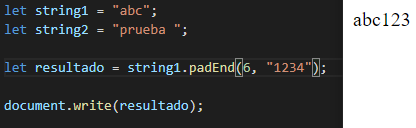
\* Si pusiera (10, “123”), esto me imprimiría – 1231231abc -



padEnd

Esto hace lo mismo que padStart pero comienza con **“abc”**, hace lo mismo pero al revés. Imprime lo que dice string1 y luego rellena con lo que le pido que rellene (“1234”) hasta que sean 6 dígitos

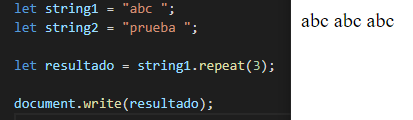
\* Si yo pongo (15, “1234”), esto me va a imprimir – abc123412341234 -



Repeat

Hace que se repita el string la cantidad de veces que yo le diga

\* string1.repeat(3), esto va a imprimir “abc” hasta 3 veces



Split

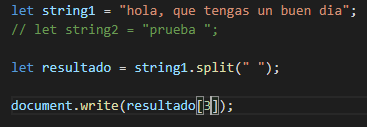
Esto lo que hace es dividir el string y devolver un array

\* **“hola, que tengas un buen dia”** – si pongo **Split(“ ”)**, lo que va a hacer esto es separar todo según donde estén los espacios

\* Al llamar resultado[2] me va a devolver **“tengas”**

\* Si llamo solo a resultado me va a devolver lo siguiente:



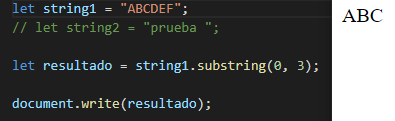
 Está dividido en un **array**

substring (se escribe todo en minúscula)

Este método lo que hace es mostrar lo que yo le pido

\* el string1 vale **“ABCDEF”**, al usar el método substring() tengo que pasarle 2 parametros los cuales son en que indice inicia y en que indice termina. Si elijo que inicie en el indice 0 y cierre en el indice 3, va a imprimir **“ABC”** ya que el indice 3 es la letra **“D”** y el indice donde se cierra no lo imprime

\* si pido que imprima substring(1,5), esto lo que me va a devolver es **“BCDE”**



toLowerCase

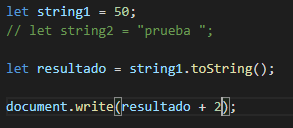
Agarra un string y lo pasa todo a minúscula

toUpperCase

Agarra un string y lo pasa todo a Mayúscula

toString

Agarra algo y lo convierte en un string



// Esto imprime 502 ya que es un string

trim

Elimina los espacios en blanco en un string

trimStart

Elimina los espacios en blanco del principio

trimEnd

Elimina los espacios en blanco del final

valueOf

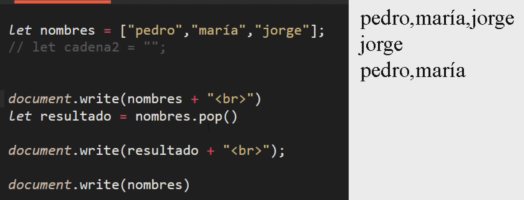
undefined

ARRAYS

pop

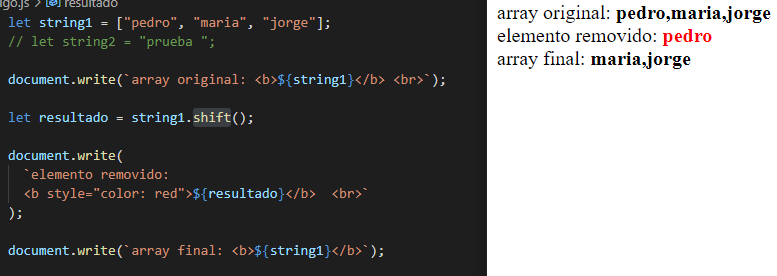
Saca solo el último elemento

\* nombres tiene 3 valores, nombres.pop() saca el ultimo el cual es **“Jorge”**



shift

Hace lo mismo que pop() pero elimina solo el primer elemento del array



push

Lo que hace es agregar un elemento **al final** de un array

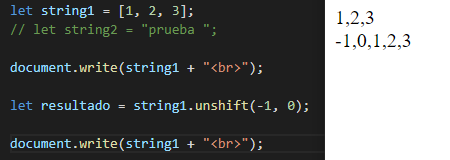
let string1 = ["pedro", "maria", "jorge"];

nombres.push(“juan”)

\* si imprimimos el string1 lo que se vería es **"pedro", "maria", "jorge", "juan"**

unshift

Sirve para agregar cuantos elementos uno quiera al principio del array



reverse

Invierte el orden de los elementos de un array



sort

Ordena los elementos de un array por orden lexicográfico



splice

Sirve para sacar y agregar uno o más elementos de un array

\* let numeros = [1, "2", 3, "4", 5, "6", 7, "8", "9"];

numeros.splice(7, 0, ”x”)

splice(indice donde arranca, cantidad de elementos que quiero eliminar, luego lo que quiero agregar)

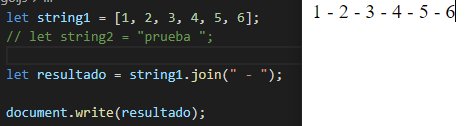
// El 1er parámetro se usa para decir en que indice arrancar (esto empuja al elemento del array haciendo que el indice de los elementos cambie), el 2do parámetro dice cuántos elementos se borran y a partir del 3er parámetro en adelante se pueden agregar cuantos elementos quiera uno

join

Este método sirve para unir todos los elementos de un array y transformarlos en un string (Esto devuelve una cadena de texto)

Además sirve para poner un separador, el cual nosotros podemos seleccionar

\* Se puede poner el “separador” que una quiera



slice

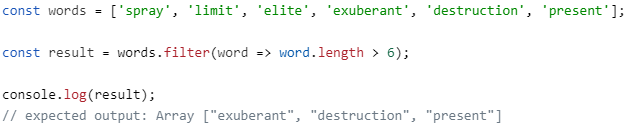
Este método lo que hace es mostrar lo que yo le pido de un array (Es como substring pero para arrays)

\* el string1 vale **[1,2,3,4,5,6]**, al usar el método slice() tengo que pasarle 2 parametros los cuales son en que indice inicia y en que indice termina. Si elijo que inicie en el indice 0 y cierre en el indice 3, va a imprimir **“123”** y el indice 3 que es el número **“4”** no lo imprime, ya que es el indice de cierre

filter

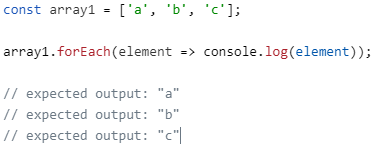
El método **filter()** crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

\* Cada palabra que tenga más de 6 letras se imprime



forEach

El método **forEach()** ejecuta la función indicada una vez por cada elemento del array.



Math.

Math.sqrt()

Lo que hace este método es mostrar la raíz cuadrada de un numero

Math.cbrt()

Lo que hace este método es mostrar la raíz cubica de un numero

Math.max()

Lo que hace este método es devolver el número más grande

Math.min()

Lo que hace este método es devolver el número más pequeño

Math.random

Lo que hace este método es devolver un numero aleatorio entre 0 y 1

\* Para hacer que el margen de aleatoriedad sea mayor ponemos algo como esto Ejemplo: Math.random()\*100 - Math.random()\*500 - Math.random()\*20000

Math.round

Lo que hace este método es redondear un numero

Let numero = Math.random() \* 100

número = Math.round(numero)

// Esto podría imprimir 68.4333 pero imprimiría solo 68, si pongo 5.9999 me imprimiría 6, ya que redondearía para arriba

Math.floor

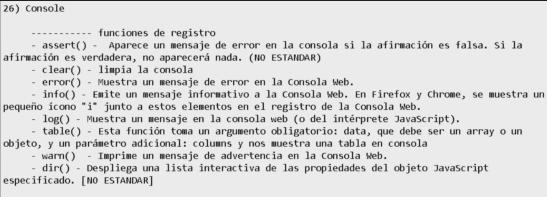
Lo que hace este método es devolver un número entero cuando un hay un numero decimal y lo devuelve redondeando para abajo

\* Math.floor(5.999999) // Esto imprimiría 5 en vez de 6

Math.PI

Muestra cuánto vale **PI**

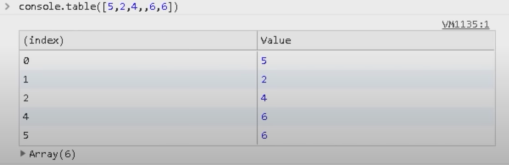
CONSOLE



Los mensajes de Consola se escriben dentro de los “**( )**”, ya sean los de Error, Información, etc…

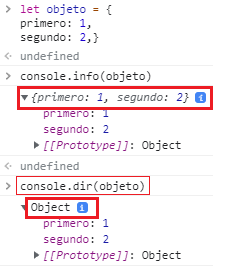
console.table()

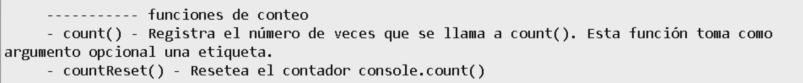
la data que toma como argumento debe ser un array o un objeto



console.dir()

Te dice de que tipo es la data que pongas entre los “**( )**”

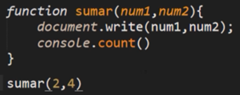




console.count()

Puede servir para saber cuántas veces se ejecuta algo

Y al usar countReset() hago que comience a contar de nuevo (haciendo que el contador se resetee)



Puedo llamar 100 veces a la función sumar() que la consola me va a decir que se ejecutó 100 veces

console.group()

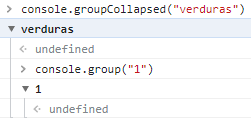
Muestra el grupo abierto



console.groupCollapsed()

Muestra el grupo Cerrado (es lo mismo que group() pero sin mostrar los detalles)



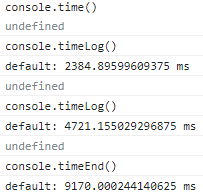


console.groupEnd()

Cierra el grupo e inhabilita el poder seguir escribiendo cosas dentro de este (todo lo que se escriba después de cerrarlo con groupEnd() se imprime en otro grupo)

time()

Inicia un temporizador



timeLog()

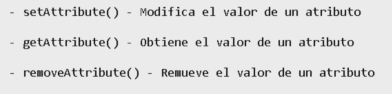
Registra la cantidad de tiempo que paso desde que se inició el temporizador hasta que ejecutamos este método (timeLog)

timeEnd()

Finaliza el temporizador

\* Para usar timeLog() de nuevo después de haber usado timeEnd() hay que usar time() para iniciar el contador otra vez.

DOCUMENT



setAttribute()

const rangoEtario = document.queryselector(“.rangoEtario”) // es un input con type “range”

rangoEtario.setAttribute(“type”,”text”) // elijo que atributo cambiar y le digo a cual hacerlo

// tambien sirve para darle valor a un atributo que no haya

\* Y esto pasa de esto

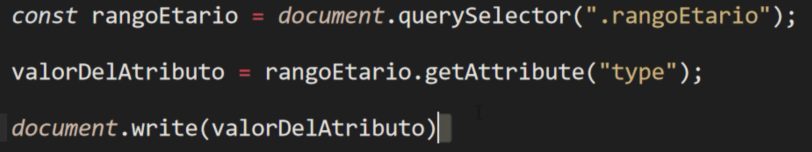


\* A esto



getAttribute()

sirve para obtener el valor de un atributo



\* Esto me imprimiría el valor del atributo seleccionado



removeAttribute()

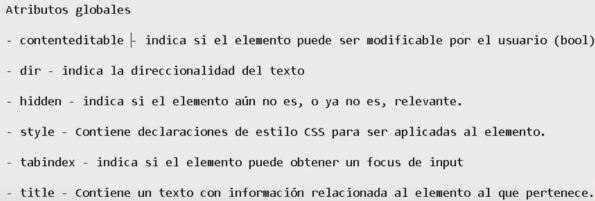
sirve para remover el atributo que yo le pida



\* Esto debería remover el atributo “type”

 // no tiene el atributo “type”

ATRIBUTOS GLOBALES



Todos estos métodos se usan dentro del setAttribute()

\* .setAttribute(“contentEditable”, ”true”) - sus valores son TRUE o FALSE

\* Sin contentEditable



\* Con contentEditable



. setAttribute(“tabIndex”,”X número”) // para verlo más fácil ponerlo directo en HTML



\* Al apretar la tecla TAB, el “focus” se va a ir moviendo dependiendo del orden que yo haya elegido

\* Arranca por el tabIndex=”1”, sigue por el 2 y termina en el 3



. setAttribute(“title”,”jajaja xd”)

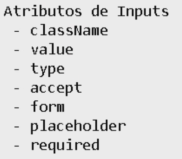
El atributo title lo que hace es que cuando se coloca el mouse sobre un elemento, a este le sale un cartel que contiene un texto

\* Atributo title normal

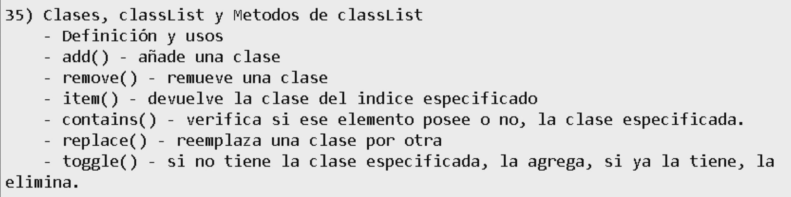


\* Atributo title modificado con setAttribute()



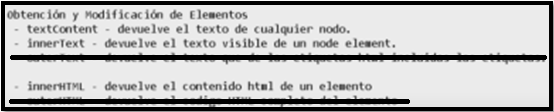


Ejemplo: Element.placeholder = “jkosdf”



<p class=”Hola” title=”Saludo”> </p>

Element.classlist.add(“Buenos Días”)



\* textContent e innerText hacen casi lo mismo



\* createElement(“DIV”) // poner el elemento en **MAYUS**

**\***  appendChild() lo que hace es insertar el elemento hijo en el padre;

EJemplo

const item = document.createElement("LI");

item.appendChild(itemText);

<li>

<p></p>

</li>



\* Para que firstChild y lastChild muestren el 1er y ultimo hijo se debe desactivar **“Prettier - Code Formatter”** del **VScode**

\* firstElementChild y lastElementChild hacen lo mismo que firstChild y lastChild pero muestran directamente el ELEMENTO sin que tener que sacar espacios en el código HTML



\* <div class=”contenedor”>

CODIGO EJEMPLO

<h2 class=”h2”> Un h2 comun </h2>

</div>

let contenedor = document.querySelector(“.contenedor”) // selecciona el div contenedor

let new\_h2 = document.createElement(“H2”) // crea un H2

new\_h2.innerHTML = “titulo”

let old\_h2 = document.querySelector(“.h2”) // esto toma el antiguo H2

contenedor.replaceChild(new\_h2, old\_h2) // esto reemplaza el antiguo H2 por el nuevo (se escribe 1ro el que reemplaza y luego el que va a ser reemplazado)

\* hasChildNodes() sirve para verificar si el elemento tiene hijos o no



\* Sirven para decir quién es el padre Y la diferencia es que parentNode busca un NODO padre y parentElement busca un padre que sea etiqueta HTML/ELEMENTO



\* se escribe de esta manera



Y sirve para mostrar cual es el hermano que está ANTES o DESPUES del elemento que yo le diga (si uso los métodos que contienen “Element” me va a mostrar los elementos HTML que están antes o después, de lo contrario podría mostrarme esto: #text)