PRIMER CAPITULO

(var= define variable global) (let=define variable en un bloque) (const=define variable fija)

Valores BOOLEANOS= TRUE , FALSE.

Tipos de datos:

Undefined= existe la variable pero no tiene un valor asignado.

Null=variable esta vacía, su valor es estar vacía (nula).

Nan= no se puede realizar la operación. No estamos usando números.

Prompt= función que nos devuelve el valor que escribimos. Y el valor se pude guardar en una variable.

Operadores:

Aritméticos= toman valores numéricos (ya sea literales o variables) como sus operando y retornan un valor numérico único. numero1 + numero2, numero1 / numero 2, etc.

De asignación=asigna un valor al operando de la izquierda basado en el valor del operando de la derecha. numero += 5, remplaza a numero= numero + 5.

Concatenación: Unir strings o cadenas de texto. Para concatenar numero sin que se sumen forzamos una cadena de texto con comillas " ".

Concat ( )= es una forma de concatenar pero siempre debe haber un string " ".

${}= usar comillas fuertes (alt + 96) para concatenar . ej = `soy + ${frase1} + y tengo calor`.

Escape de comillas simples y escape de comillas dobles. Usar comillas simples por fuera y dobles por dentro, o viceversa.

Operadores de comparación: comparan dos expresiones y devuelven un valor BOOLEAN que representa la relación de sus valores. Funcionan con textos. Por ejemplo: (==)igualdad.(¡=)inigualdad.(===)idénticos en cuanto a datos y valor.(¡==)estrictamente distinto. (<)menor.(>)mayor.(<=)menor o igual.(>=)mayor o igual.

Operadores lógicos: Nos devuelven un resultado a partir de que se cumpla (o no) una condición, su resultado es BOOLEANO, y sus operandos son valores lógicos o asimilables a ellos. (&&)si las dos condiciones se cumplen es true, sino false.(||) si un valor es verdadero, todo es verdadero, si todo es falso es falso. (!) cambia el valor al opuesto, si es true lo cambia a false.

Camel case: La primera letra de todas va con minúscula y después cada palabra nueva que comienza va con mayúscula. Por ejemplo: decimeAlgo. parseInt.

Condicionales: Es una sentencia que nos permite validar algo. Hace que un bloque { } se ejecute solo si una condición se cumple. IF(primero va una condición y luego si es verdadera se ejecuta algo), ELSE IF(va con una condición igual que if,se pueden poner los que queramos),ELSE(si no se cumple ninguna condición, que se ejecute el ELSE).

CAPITULO 2

ARRAYS: Para crear arreglos se usan llaves [ ]. Puede tener muchos tipos de datos y de varios tipos y se separan con comas dentro de la misma. Si son textos van comillas. Se muestran por separado de distintas maneras.

ARRAYS ASOCIATIVOS: Otra forma de trabajar con arrays. Es un array que tiene un valor asociado. Ejemplo: **document.write(pc[“nombre”]);.** Primero puedo crear una array, luego creo variables llamando los datos de la misma, y luego puedo concatenar esos datos.
 sirve para continuar escribiendo abajo.

Bucles e interacción: Se repiten continuamente

While: sigue preguntando si la condición se sigue cumpliendo y se ejecuta hasta que la condición sea falsa. Let numero =0;

```
While (numero <10) {

numero++;

document.write (numero)
```

} Resultado: 12345678910

Do while: A diferencia del while en este primero se ejecuta y después pregunta condición.

. Let numero =0;

```
Do {

document.write (numero)

numero++;
```

}

While (numero <10)

Resultado:0123456789

Break: Esto hace que el bucle finalice.

. Let numero =0;

```
While (numero <10) {
 numero++;
  document.write (numero) ;
if (numero ==10) {
break; (cuando numero sea 10, el break corta el bucle)
}
```
Resultado: 12345678910

For: Es igual que el while pero es un bucle derteminado. (1)declaración(2)condición(3)aumento o decremento.

```
For (let i = 0; i<10;i++) {
Document.write (i)
} resultado: 0123456789
```

Continue: No finaliza el bucle, termina la condición que le ponemos.

```
For (let i = 0; i<10;i++) {
If (i=5) {
Continue;
}
Document.write (i)
} Resultado: 01234 “ “678910
```

For in: Es una estructura **for** en donde el **in** nos devuelve la posición de los elementos de un array. Si luego en el document.write llamamos a la variable y luego entre [ ] llamamos el in que agregamos, hacemos que en vez de números, diga los elementos.

```
Let animales=[“gato”, “perro”, ”pájaro”];
For (animal in animales) {
Document.write(animal);
}
```
Resultado: 0 1 2

For of: Se diferencia con for in porque IN nos muestra la posición de los elementos, en cambio OF nos muestra directamente el valor que hay en esas posiciones, osea “gato”,”perro”,”pájaro”.

Label: Asocia un bucle a un nombre, para poder terminarlo cuando queramos.

Funciones: se pueden CREAR solas o añadiéndolas a una variable. EL SCOPE DE LAS FUNCIONES ES **REGIONAL,** OSEA QUE LAS VARIABLES SIRVEN DENTRO DE LA MISMA NOMAS. **PONER SIEMPRE LET.**

Return: Sirve para que la función nos devuelva un valor. Y finaliza la función.

```
function suma(num1,num2) {
let res=num1+num2;
return res
}
let resultado= suma(20,80);
document.write(resultado)
```

RESULTADO 100

Parámetros: Permite usar la misma función con numero distintos o datos. Se declaran los valores a usar y dentro de arma la operación. Los parámetros son variables que se declaran y crean en una función, especiales para esa función.

PARAMETROS

```
Function suma(num1,num2) {
let res = num1 + num2;
document.write(res);
document.write("br")
}
```

suma(22,25) RESULTADO 47

Funciones flecha: Se crean de una forma distinta. Y si usamos un solo parámetro no es necesario poner paréntesis. Y si solo tenemos una expresión o una línea de código dentro de las llaves, tampoco es necesario llaves.

```
const saludar = (nombre)=>{
let frase = `¡hola $(nombre! Como estas?`;
document.write(frase)
}
saludar("pedro")
```

CAPITULO 3

Programación orientada a objetos

Es un paradigma de la programación que lo que hace es permitirnos programar como lo haríamos en la vida real.

Clase: Es una plantilla que creamos, para que con esta podamos crear objetos, una "fabrica de objetos". Crean objetos.

Objeto: Son creados por las clases.

Atributo: Son las propiedades, las características, que tiene el objeto.

Método: Son las cosas que puede hacer nuestro objeto.

Constructor: Es una particularidad que tienen las clases, que es una función obligatoria. Cuando creamos una clase tenemos que crear un constructor que nos va a construir las propiedades del objeto.

Instanciación: Cuando finalizamos todo se puede decir que la clase esta instanciada.

This.(nombre del objeto que voy a crear) = Se usa para crear un objeto. Y SOLO SE USA EN EL CONSTRUCTOR, NO EN EL METODO.
Cuando se trabaja con objetos se usa "const" en vez de "let".
New: Instancia un objeto.
Accedemos a las propiedades con un . (punto).

```
class animal {

    constructor(especie,edad,color){
        this.especie = especie;
        this.edad = edad;
        this.color = color;
    }
}

const perro = new animal("perro",5,"rojo");


document.write(perro)
```

Manera de acceder a un parámetro.

```
document.write(perro.color)
```

Código anterior completo.

```
class animal {

    constructor(especie,edad,color){
        this.especie = especie;
```

```
        this.edad = edad;
        this.color = color;
        this.informacion = `Soy ${this.especie}, tengo
${this.edad} años, y soy de color ${this.color}`;
    }
}

const perro = new animal("perro",5,"rojo");
const gato = new animal("gato",7,"negro");
const pajaro = new animal("pajaro",2,"amarillo");


document.write(perro.informacion + "<br>")
document.write(gato.informacion + "<br>")
document.write(pajaro.informacion + "<br>")
```

LAS FUNCIONES FLECHAS NO PUEDEN SER USADAS PARA CREAR METODOS EN LAS CLASES.

Características de la programación orientada a objetos

Abstracción: Se refiere a intentar reducir lo que más podamos el objeto, hacer que sea lo menos complejo que podamos. Reducimos todos sus componentes básicos. Resumiendo un objeto a lo fundamental.

Modularidad: Es la capacidad de resolver un problema grande, separándolo por partes.

Encapsulamiento: Consiste en hacer que todos los datos estén privados, que el usuario no pueda acceder tan fácil, encapsular los datos para que la información no salga de ahí de forma tan sencilla.

Polimorfismo: Consiste en ver como un objeto se comporta de manera distinta ante el mismo método por sus propiedades.

```
class animal {
    constructor(especie,edad,color){
        this.especie = especie;
        this.edad = edad;
        this.color = color;
```

```
        this.info = `Soy ${this.especie}, tengo ${this.edad} años,
y soy de color ${this.color}`;
    }
    verIfo(){
        document.write (this.info + "<br>");
    }
    ladrar() {
        if (this.especie == "perro") {
            document.write("¡Guau!" + "<br>")
        } else {
            document.write("No puede ladrar ya que es un " +
this.especie + "<br>" )
        }
    }
}

const perro = new animal("perro",5,"rojo");
const gato = new animal("gato",7,"negro");
const pajaro = new animal("pajaro",2,"amarillo");


//document.write(perro.info + "<br>")
//document.write(gato.info + "<br>")
//document.write(pajaro.info + "<br>")

perro.ladrar();
gato.ladrar();
pajaro.ladrar();
```

Creamos un método donde se va a comportar distinto, según las propiedades que tengan.

Herencia: La herencia es crear una clase que va a tomar todo lo que pueda hacer otra clase y las propiedades de la misma, pero además agregarle cosas nuevas. Para esto se usa EXTENDS: `class perro extends animal {`

De esta manera creamos una clase perro que va a contener todo lo que tenga la clase animal, pero le agregamos algunas cosas específicas.

Ahora vamos a hacer que herede algunas propiedades de la clase anterior. De esta manera dentro de super, ponemos las propiedades que queremos que herede al igual que los métodos que incluyan esas propiedades. Y dentro de constructor añadimos alguna

propiedad nueva que queramos añadir.

```
class perro extends animal {

    constructor (especie,edad,color,raza)
    super(especie,edad,color)
}
```

No puedo tener un objeto con el mismo nombre que una clase.

Esta es la herencia que queda de una clase a otra, la clase perro al tener las mismas propiedades de la clase animal. Hereda el método verInfo.

```
class Animal {

    constructor(especie,edad,color){
        this.especie = especie;
        this.edad = edad;
        this.color = color;
        this.info = `Soy ${this.especie}, tengo ${this.edad} años,
y soy de color ${this.color}`;
    }
    verIfo(){
        document.write (this.info + "<br>");
    }
}
class Perro extends Animal {
    constructor (especie,edad,color,raza){
    super(especie,edad,color);
    this.raza = raza;
    }
}


const perroo = new Perro("perro",5,"rojo","dogo");
const gato = new Animal("gato",7,"negro");
const pajaro = new Animal("pajaro",2,"amarillo");


//document.write(perro.info + "<br>")
//document.write(gato.info + "<br>")
//document.write(pajaro.info + "<br>")
```

```
perroo.verIfo();
gato.verIfo();
pajaro.verIfo();
```

Métodos estáticos: Es un método que no se necesita que la clase se defina, para que pueda ser creado.
Esto puede ser un método estático, porque no estamos usando las propiedades, de esta manera no hace falta crear el objeto. Porque el constructor, construye el objeto cuando le damos todo lo que nos pide en las propiedades (especie, edad, color, raza)

```
class Perro extends Animal {

    constructor (especie,edad,color,raza){
    super(especie,edad,color);
    this.raza = raza;
    }
    static ladrar() {
        alert("waw");
    }
}
```

Métodos setters (set): son para modificarlos o definirlos. Modificamos una propiedad de la clase y de un método se convierte a propiedad.

```
class Perro extends Animal {

    constructor (especie,edad,color,raza){
    super(especie,edad,color);
    this.raza = raza;
    }
```

```
    set modificarRaza(newName) {
        this.raza = newName
    }
}
const perro = new Perro("perro",5,"rojo","dogo");
const gato = new Animal("gato",7,"negro");
const pajaro = new Animal("pajaro",2,"amarillo");

perro.modificarRaza = "pedro";
document.write(perro.raza);
```

Métodos getters (get): son para obtener un valor o información.

```
class Perro extends Animal {

    constructor (especie,edad,color,raza){
    super(especie,edad,color);
    this.raza = raza;
    }
    set setRaza(newName) {
        this.raza = newName
    }
    get getRaza (){
        return this.raza;
    }
}
const perro = new Perro("perro",5,"rojo","dogo");
const gato = new Animal("gato",7,"negro");
const pajaro = new Animal("pajaro",2,"amarillo");

perro.setRaza = "Pitbull";
document.write(perro.getRaza);
```

Raramente los get y los set funcionan como propiedad.

Los objetos se definen con CONST. Además los get y los set se utilizan para acceder a datos encapsulados en JS.

Problemas Cofla:

.Crear un sistema para mostrar las particularidades de los 3 celulares.

.Cada celular debe tener color, peso, resolución de pantalla, resolución de cámara y memoria ram.

.Cada celular debe prender, reiniciar, apagar, tomar fotos y grabar.

```
class celulares {

    constructor(color, peso, rPantalla, rCamara, memoriaRam){
    this.color = color;
    this.peso = peso;
    this.rPantalla = rPantalla;
    this.rCamara = rCamara;
    this.memoriaRam = memoriaRam;
    this.encender = false;
}
    presionarBotonEncendido() {
        if (encender == false) {
            alert("Encendiendo el celular");
            encender = true;
        } else {
            alert("El celular esta apagado");
        }
    }
    reiniciar() {
        if (encender == true) {
            alert("Reiniciando el celular");
        }else {
            alert("El celular esta apagado");
        }
    }
    tomarFoto() {
        alert(`Foto tomamda en una resolucion de ${this.rCamara}.`
);
    }
    grabarVideo() {
        alert(`Grabando un video en ${this.rCamara}. `);
    }
    verInfo() {
        return `
        <br>Color: ${this.color}<br>
        Peso: <br>${this.peso}<br>
        Tamaño: <br>${this.rPantalla}<br>
        Resolucion de camara: <br>${this.rCamara}<br>
        Memoria RAM: <br>${this.memoriaRam}<br>`
```

```
    }
}

class celularAltaGama extends celulares {
    constructor (color, peso, rPantalla, rCamara, memoriaRam,
rcExtra,) {
        super(color, peso, rPantalla, rCamara, memoriaRam);
        this.rcExtra = rcExtra;
    }
    grabarVideoLento () {
        alert("estas grabando en camara lenta");
    }
    reconocimentoFacial () {
        alert("Iniciado un reconocimiento facial");
    }
    infoAltaGama () {
        return this.verInfo() + `Resolucion de camara trasera:
${this.rcExtra}`;
    }
}
//const sansungA10 = new celulares ("blanco", "200gr", "7
pulgadas", "FullHd", "6gb");
//const iphoneX = new celulares ("blanco", "200gr", "7 pulgadas",
"FullHd", "6gb");
//const lgK10= new celulares ("negro", "240gr", "6.5 pulgadas",
"Hd", "4gb",);

//sansungA10.presionarBotonEncendido();
//sansungA10.reiniciar();
//sansungA10.tomarFoto();
//sansungA10.grabarVideo();

const celular1 = new celularAltaGama("rojo", "130gr", "5.2p",
"4k", "3GB","fullHD");
const celular2 = new celularAltaGama("negro", "142gr", "6p", "4k",
"4GB", "HD");

document.write(`
Celular 1:${celular1.infoAltaGama()}<br><br>
Celular 2:${celular2.infoAltaGama()}<br><br>
```

```
`);
```

.Crear un sistema que ayude a elegir que app descargar.
.La información debe ser, cantidad de descargas, puntuación y peso.
.La app se debe poder, instalar, abrir, cerrar y desinstalar.

```
class app {

    constructor(descargas, puntuacion, peso) {
    this.descargas = descargas;
    this.puntuacion = puntuacion;
    this.peso = peso;
    this.iniciada = false;
    this.instalada = false;
    }
    instalar () {
        if (this.instalada == false) {
            this.instalada = true;
            alert("app instalada");
        }
    }
    abrir () {
        if (this.iniciada == false && this.instalada == true) {
            this.iniciada = true;
            alert("app encendida");
        }
    }
    cerrar () {
        if (this.iniciada == true && this.instalada == true) {
            this.iniciada = false;
            alert("app cerrada");
        }
    }
    desinstalar () {
        if (this.instalada == true) {
            this.instalada = false;
            alert("app desinstalada");
        }
```

```
    }
    info () {
        return document.write (`
        Descargas: ${this.descargas}<br>
        Valoracion: ${this.puntuacion}<br>
        Peso: ${this.peso}
        `);
    }

}

Aplicacion = new app ("16.000 descargas", "4.5 estrellas", "Pesa:
56Mb")

Aplicacion.instalar();
Aplicacion.abrir();
Aplicacion.cerrar();
Aplicacion.desinstalar();
Aplicacion.info();
```

Capítulo 4

Metodos de cadenas: El tipo de dato str es una clase incorporada cuyas instancias incluyen variados **métodos** —más de treinta— para analizar, transformar, separar y unir el contenido de las **cadenas** de caracteres.

Métodos para buscar cadenas y devolver un valor.

Métodos:

concat(): Junta dos o más cadenas y retorna una nueva.

```
let cadena = "cadena de prueba";


resultado = cadena.concat("hola");

document.write(resultado);

let cadena = "cadena de prueba";
let cadena2 = " cadena 2";

resultado = cadena.concat(cadena2);

document.write(resultado);
```

startswith(): si una cadena comienza con los caracteres de otra cadena, devuelve true, sino devuelve false. endwith(): funciona igual solo que tienen que terminar con los mismos caracteres. Estos métodos verifican que dos cadenas terminen o empiecen igual.

```
let cadena = "cadena de prueba";

let cadena2 = "cadena";

resultado = cadena.startsWith(cadena2);

document.write(resultado);
```

includes(): Si una cadena puede encontrarse dentro de otra cadena, devuelve true, sino devuelve false. Busca cadenas. Con que contenga cualquier elemento igual, es true.

```
let cadena = "cadena de prueba";

let cadena2 = "cadena";

resultado = cadena.includes(cadena2);

document.write(resultado);

let cadena = "cadena de prueba";

let cadena2 = "";

resultado = cadena.includes("prueba");

document.write(resultado);
```

indexOF(): Funciona igual que el anterior, pero nos tira la posición en la que esta la primer letra. El siguiente ejemplo nos da 10. Es la posición de la letra “p”.

```
let cadena = "cadena de prueba";

let cadena2 = "";

resultado = cadena.indexOf("prueba");

document.write(resultado);
```

O así nos tira la letra o elemento que buscamos según lo que ponemos entre corchetes. El cual el resultado es “c”.

```
let cadena = "cadena de prueba";

let cadena2 = "";

resultado = cadena.indexOf("prueba");

document.write(cadena[0]);
```

Ahora si lo que buscamos no parece, nos devuelve el dato “-1”, asique cuando nos da ese dato, es que no existe lo que estamos buscando.

lastindexOF(): Funciona igual que el anterior, nomas que en vez de devolvernos la posición del primer elemento, nos devuelve el ultimo. Recorre el array de atrás para adelante y devuelve el primero que encuentra.

```
let cadena = "cadena de prueba, prueba";

let cadena2 = "";

resultado = cadena.lastIndexOf("prueba");

document.write(resultado);
```

Métodos para rellenar cadenas.

padsStarts(): Rellena la cadena al principio con los caracteres deseados. Primero escribimos la cantidad de elementos que queremos que tenga, y luego entre comillas que es lo que se le va a agregar. El resultado de esto es “aaaabc”. Esta no es oficial pero se usa.

```
let cadena = "abc";

let cadena2 = "";

resultado = cadena.padStart(6,"a");

document.write(resultado);
```

padEnd(): Es igual que la anterior pero rellena al final. El resultado de esto es “abc1234”.

```
let cadena = "abc";

let cadena2 = "";
```

```
resultado = cadena.padEnd(7,"1234");

document.write(resultado);
```

repeat(): Devuelve la misma cadena pero repetida las veces que queramos. El resultado de esta es “abcabc”.

```
let cadena = "abc";

let cadena2 = "";

resultado = cadena.repeat(2);

document.write(resultado);
```

Métodos para convertir o transformar cadenas.

split(): Divide la cadena como le pidamos o según el elemento que usemos para dividir el array. Ya sea espacios “ “, o comas por ejemplo “,”. Y luego colocamos el numero del elemento que queremos que aparezca. El resultado de esto es “como”.

```
let cadena = "Hola como estas?";

//let cadena2 = "";

resultado = cadena.split(" ");

document.write(resultado[1]);

Otra cosa que se puede hacer es en vez de colocar un numero, colocamos elemento completo, separa lo que esta antes y lo que que esta después del que coloquemos. El resultado de esto es “Hola, estas?. Elimina el elemento que colocamos entre paréntesis.
let cadena = "Hola como estas?";
```

```
//let cadena2 = "";

resultado = cadena.split("como");

document.write(resultado);
```

substring(): Nos devuelve lo que seleccionamos. Crea un nuevo string y le decimos desde donde queremos que esta nueva array comience y termine. Con el que comienza esta incluido, con el que termina no. El resultado de esto es “AB”.

```
let cadena = "ABCDEFG";

//let cadena2 = "";

resultado = cadena.substring(0,2);

document.write(resultado);
```

Si nosotros ponemos “to” seguido de un método y sirve para convertir las cadenas.

toLowerCase(): Sirve para pasar una cadena a minúscula. El resultado seria “abcdefg”.

```
let cadena = "ABCDEFG";


resultado = cadena.toLowerCase();

document.write(resultado);
```

toUpperCase(): Sirve para pasar a mayúscula. El resultado seria “HOLA, COMO ESTAS?”

```
let cadena = "hola, como estas?";


resultado = cadena.toUpperCase();

document.write(resultado);
```

toString(): Sirve para convertir un dato a string. A un string no se le aplica. Convierte de cadena de texto a string, para poder acceder a elementos particulares. El resultado de esto es “P”, pero si lo dejamos en cadena el resultado seria “pedro”.

```
let cadena = "Pedro, Matias";


resultado = cadena.toString();

document.write(resultado[0]);
```

trim(): Elimina los espacios en blanco. Los espacios cuentan como caracteres, pero con trim se eliminan. La cadena original tiene 11 caracteres, pero ahora tiene 5. El resultado es 5.

```
let cadena = "    Pedro    ";

resultado = cadena.trim();

document.write(resultado.length);
```

trimEnd(): Elimina solo los espacios del final. El resultado de esto ahora es 8. Solo elimino los espacios finales.

```
let cadena = "    Pedro    ";


resultado = cadena.trimEnd();

document.write(resultado.length);
```

trimStart(): Hace exactamente lo mismo pero eliminando los espacios iníciales. Y también el resultado es 8, pero esta vez elimina los espacios iniciales.

```
let cadena = "    Pedro    ";


resultado = cadena.trimStart();
```

```
document.write(resultado.length);
```

length : Sirve para mostrar la cantidad de caracteres que tiene una cadena. En string cuantas letras o caracteres tiene y en los array cuantos elementos tiene.

Métodos Transformadores de arrays:

pop(): Elimina el último elemento del array y además nos muestra cual elimino. El resultado es "juan".

```
let nombres = ["Pedro","Maria", "Juan"];


resultado = nombres.pop();

document.write(resultado);
```

shift(): Hace lo mismo que pop pero elimina el primer elemento y lo muestra. El resultado de esto es "pedro".

```
let nombres = ["Pedro","Maria", "Juan"];


resultado = nombres.shift();

document.write(resultado + "<br>");
```

push(): Agrega un elemento al array al final de la lista. Pero nos devuelve la cantidad de elementos que tiene el array luego de agregar. Luego a lo ultimo cuando pedimos que nos muestre el array, nos aparece la modificación. "Pedro, María, Juan, Juancito". Se pueden agregar más de uno.

```
let nombres = ["Pedro","Maria", "Juan"];


document.write(nombres + "<br>");

let resultado = nombres.push("Juancito", "Pedro");

document.write(nombres);
```

Reverse(): Invierte el orden de los elementos de un arrays. El resultado de esto es "Juan, Maria, Pedro".

```
let nombres = ["Pedro","Maria", "Juan"];
```

```
document.write(nombres + "<br>");

let resultado = nombres.reverse();

document.write(nombres);
```

unshift(): Agrega uno o más elementos al inicio del arrays y devuelve la nueva longitud del arrays. Como parámetros ponemos directamente que elementos le agrego. El resultado de esto es "0,1,2,3,4,5".

```
let numeros = [3,4,5];


document.write(numeros + "<br>");

let resultado = numeros.unshift(0,1,2);

document.write(numeros);
```

sort(): Ordena los elementos de un array localmente y devuelve el arreglo ordenado. (orden lexicográfico). Ordena de manera numérica o alfabética según el dato. El resultado de esto es "1,2,3,4,5,6,7,8,9"

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


document.write(numeros + "<br>");

let resultado = numeros.sort();

document.write(numeros);
```

splice(): Cambia el contenido de un array eliminando elementos existentes y/o agregando nuevos elementos. Primero colocamos como parámetro en qué posición comenzamos. Y luego ponemos cuantos elementos eliminamos a partir de que comenzamos. El resultado seria “1,2,3”.

```
let numeros = [1,2,3,4,5,6];


document.write(numeros + "<br>");

let resultado = numeros.splice(3,3);

document.write(numeros)
```

Y si quiero agregar elementos luego de eliminar los otros. Coloco como parámetros los que quiero agregar después de una coma(,). El resultado de estos seria, “1,2,3,7,8,9”.

```
let numeros = [1,2,3,4,5,6];


document.write(numeros + "<br>");

let resultado = numeros.splice(3,3,7,8,9);

document.write(numeros);
```

La forma para referirse al final de un array es (-1). Per so queremos agregar algo al final de un array, lo mejor es usar push().

Y si no quiero eliminar ninguno, primero colocamos la posición en donde queremos agregarlos, y luego un cero(0) en donde pondríamos los elementos a eliminar. Ya luego lo que vamos a agregar. El resultado de esto es “1,2,3,4,5,6,7,8,9”;

```
let numeros = [1,2,3,4,5,6];


document.write(numeros + "<br>");

let resultado = numeros.splice(6,0,7,8,9);

document.write(numeros);
```

Métodos accesores:

Join(): Se usa para poner separadores, y los escribimos como parámetros del método. Según como queramos separar. Y lo pasa a cadena de texto. El resultado sería una lista con los elementos. En vez de uno al lado de otro, se pueden usar guiones o lo que queramos para separar, y sin necesidad de ser una lista.

Elemento: uno
Elemento:dos
Elemento:tres
Elemento:cuatro

```
let numeros = ["uno", "dos", "tres", "cuatro"];

document.write(numeros + "<br>");

let resultado = numeros.join("<br> Elemento:");

document.write("Elemento: " + resultado);
```

Para seleccionar todos los elementos de un array se puede, o poner -1 pero no nos muestra el último elemento. O dejamos el segundo espacio del parametro vacio. numeros.slice(0) seria.

slice(): Devuelve una parte del array dentro de un nuevo array, empezando de inicio a fin(fin no incluido). Y como parámetro ponemos la posición desde donde inicia y donde termina (sin incluir el termino en donde termina en el array). El resultado es “uno, dos, tres”.

```
let numeros = ["uno", "dos", "tres", "cuatro"];


document.write(numeros + "<br>");

let resultado = numeros.slice(0,3);

document.write(resultado);
```

include(): funciona igual con los string que con las cadenas.
indexOF():funciona igual con los string que con las cadenas.
lastIndexOF():funciona igual con los string que con las cadenas.
toString():funciona igual con los string que con las cadenas.

Métodos de arrays de repetición:

filter(): Funciona como un bucle, recibe como parámetro una función. Y en cada vuelta del bucle el parámetro que le pasemos va a ser igual a cada uno de los elementos que tiene el array y podemos hacer con ellos lo que queramos.

```
let numeros = ["uno", "dos", "tres", "cuatro"];


numeros.filter((numero) => {
    document.write(numero + "<br>")
})
```

El filter es más completo que el forEach por que tiene la cualidad de crear un nuevo array solamente con los elementos que cumplan la condición que nostros queramos. Esto solo nos muestra las palabras con más de 4 elementos.

```
let numeros = ["uno", "veintidos", "tres", "veinticuatro"];


document.write(numeros + "<br>");

let resultado = numeros.filter(numero => numero.length > 4);

document.write(resultado);
```

forEach(): Es muy similar a filter. Ejecuta la función indicada una vez por cada elemento del array.

```
let numeros = ["uno", "veintidos", "tres", "veinticuatro"];


document.write(numeros + "<br>");

numeros.forEach((numero) => {
    document.write(numero + "<br>");
});

document.write(resultado);
```

Continuación capitulo 4

Objeto Math: Básico

Métodos:

sqrt(): Devuelve la raíz cuadrada positiva de un numero. El resultado es “5”.

```
let numero = Math.sqrt(25);


document.write(numero);
```

cbrt(): Devuelve la raíz cubica de un numero. El resultado es “3”.

```
let numero = Math.cbrt(27);


document.write(numero);
```

max(): Este método trabaja solo con varios números. Y nos devuelve el numero más grande de todos. El resultado es “500”.

```
let numero = Math.max(4,79,85,2,500);


document.write(numero);
```

min(): Es la función inversa de max. Nos devuelve el número más chico. El resultado es “2”.

```
let numero = Math.min(4,79,85,2,500);


document.write(numero);
```

random(): Devuelve un numero pseudo-aleatorio entre 0 y 1.

```
let numero = Math.random();


document.write(numero);
```

round(): Devuelve el valor de un numero redondeando al número entero más cercano. E l resultado van a ser número enteros del 1 al 100, sin decimales. El cero no esta incluido.

```
let numero = Math.random() * 100;


numero = Math.round(numero);

document.write(numero);
```

floor(): Devuelve el mayor entero, menor que, o igual a un numero. Redondea para abajo. El cero esta incluido, el 100 no.

```
let numero = Math.random() * 100;


numero = Math.floor(numero);

document.write(numero);
```

fround(): Devuelve la representación flotante de precisión simple más cercana a un numero. Nos redondea un número con 15 decimales. Es de precisión simple (4 byts). ES MEDIO IRRELEVANTE.

```
let numero =
Math.fround(2.752749429724971297123947132496123972792197);


document.write(numero);
```

trunc(): Devuelve la parte entera del numero x, la eliminación de los digitos fraccionarios. Es como round, pero la diferencia es que raund redondea al número más cercano, pero trunc elimina directamente la fracción. El resultado de esto es “9”. Quita los decimales.

```
let numero = Math.trunc(9.465);


document.write(numero);
```

Propiedades: van sin paréntesis.

PI: Ratio de la circunferencia de un circulo respecto a su diámetro, aproximadamente 3.14.
El resultado de esto es “3.141592653589793”. Con representación flotante simple mas cercana a un numero.

```
let numero = Math.PI;


document.write(numero);
```

SQRT1_2: Raíz cuadrada de un medio. Con representación flotante simple más cercana a un numero. El resultado es “0.7071067811865476”.

```
let numero = Math.SQRT1_2;


document.write(numero);
```

SQRT2 : Raíz cuadrada de 2. . Con representación flotante simple más cercana a un numero.

```
let numero = Math.SQRT2;


document.write(numero);
```

Constantes y logaritmos:

(E): Constante de Euler, la base de los logaritmos naturales. El resultado es “2.718281828459045”.

```
let numero = Math.E;


document.write(numero);
```

LN2 : Logaritmo natural de 2, aproximadamente “0.6931471805599453”.

```
let numero = Math.LN2;


document.write(numero);
```

LN10: Logaritmo natural de 10, aproximadamente “2.302585092994046”

```
let numero = Math.LN10;


document.write(numero);
```

LOG2E : Logaritmo de E con base 2. Aproximadamente “1.4426950408889634”

```
let numero = Math.LOG2E;


document.write(numero);
```

LOG10E: Logaritmo de E con base 10. Aproximadamente “0.4342944819032518”

```
let numero = Math.LOG10E;


document.write(numero);
```

Problemas de cofla:

-Ahora necesita una calculadora que también calcule potencia, raíz cuadrada y cubica.

--crear una función que al pasarle como parámetro la materia, nos devuelva:
.profesor asignado.
.el nombre de todos los alumnos.

```
const obtenerInformacion = (materia) => {
    materias = {
        fisica: ["Perez", "pedro", "juan", "pepe"],
        programacion:["Maidana","pedro", "juan", "pepe", "cofla",
"maria"],
        logica:["Lopez","pedro", "pepe", "cofla", "maria"],
        quimica:["Perez","pedro", "juan", "pepe", "cofla",
"maria"],
    }
    if (materias[materia] !== undefined) {
        return [materias[materia], materia];
    }else {
        return false;
    }
```

```
}
let informacion = obtenerInformacion(prompt("ingrese la
meteria"));

if (informacion !== false) {
    let profesor = informacion[0][0];
    let alumnos = informacion[0];
    alumnos.shift();
    document.write(`El profesor de <b>${informacion[1]}</b> es:
${profesor} y los alumnos son = <b style= "color:red">
${alumnos}</b>`);
}
```

Función que dice en que clases esta cofla

```
const obtenerInformacion = (materia) => {
    materias = {
        fisica: ["Perez", "pedro", "juan", "pepe"],
        programacion:["Maidana","pedro", "juan", "pepe", "cofla",
"maria"],
        logica:["Lopez","pedro", "pepe", "cofla", "maria"],
        quimica:["Perez","pedro", "juan", "pepe", "cofla",
"maria"],
    }
    if (materias[materia] !== undefined) {
        return [materias[materia], materia, materias];
    }else {
        return materias;
    }
}
const mostrarInformacion = (materia) => {
    let informacion = obtenerInformacion(materia);

if (informacion !== false) {
    let profesor = informacion[0][0];
    let alumnos = informacion[0];
    alumnos.shift();
```

```
    document.write(`El profesor de <b>${informacion[1]}</b> es: <b
style="color:blue">${profesor}</b> y los alumnos son = <b style=
"color:red"> ${alumnos}</b><br><br>`);
}
}
const cantidadDeClases = (alumno) => {
    let informacion = obtenerInformacion();
    let cantidadTotal = 0;
    for (info in informacion) {
        if (informacion[info].includes(alumno)) {
            cantidadTotal++;
        }
        }
        return  `${alumno} esta en ${cantidadTotal} clases<br>`;
}

mostrarInformacion("fisica");
mostrarInformacion("logica");
mostrarInformacion("quimica");
mostrarInformacion("programacion");

document.write(cantidadDeClases("cofla"));
```

Nombrar las clases en las que esta y los profesores de cada una.

```
const cantidadDeClases = (alumno) => {

    let informacion = obtenerInformacion();
    let cantidadTotal = 0;
    let clasesPresentes = [];
    for (info in informacion) {
        if (informacion[info].includes(alumno)) {
            cantidadTotal++;
            clasesPresentes.push(" " + info);
        }
        }
        return  `${alumno} esta en ${cantidadTotal} clases:
${clasesPresentes}
        `;
}
```

Crear una función que le pregunte a cofla en que materia se quiere inscribir.
Si hay más de 20 alumnos, negarle la inscripción.
Si hay menos de 20 alumnos, inscribir a cofla y añadirlo a la lista de alumnos.

```
let materias = {
        fisica: ["Perez", "pedro", "juan", "pepe"],
        programacion:["Maidana","pedro", "juan", "pepe", "cofla",
"maria"],
        logica:["Lopez","pedro", "pepe", "cofla", "maria"],
        quimica:["Perez","pedro", "juan", "pepe", "cofla",
"maria"],
    }


const inscribir = (alumno,materia) => {
    let personas = materias[materia];
    if (personas.length >= 10) {
    document.write (`Lo siento ${alumno}, no podes inscribirte en
${materia}, limite superado.<br>`);
    } else {
        personas.push(alumno);
        if (materia == "fisica") {
            materias = {
                fisica: personas,
                programacion:materias["programacion"],
                logica:materias["logica"],
                quimica:materias["quimica"],
            }
        } else if (materia == "programacion") {
            materias = {
                fisica: materias["fisica"],
                programacion:personas,
                logica:materias["logica"],
                quimica:materias["quimica"],
            }
    }else if (materia == "logica") {
        materias = {
```

```
            fisica: materias["fisica"],
            programacion:materias["programacion"],
            logica:personas,
            quimica:materias["quimica"],
        }
}else if (materia == "quimica") {
    materias = {
        fisica: materias["fisica"],
        programacion:materias["programacion"],
        logica:materias["logica"],
        quimica:personas,
    }
}
document.write (`Felicidades ${alumno}, te inscribiste
correctamente a ${materia}`);
    }
}
document.write (materias["fisica"] + "<br>");
inscribir("cofla","fisica");
document.write( "<br>" + materias["fisica"]);
```

Capitulo 5

La consola

La consola es uno de tantos objetos o Apis que podemos encontrar en el navegador o en el interprete de JavaScript que podemos usar a la hora de ver cuáles son los errores cuando escribimos código, etc. En el navegador apretamos F12 y vamos a consola.
Es como si escribiéramos código js pero secuencialmente o paso por paso.

Métodos de la consola.

FUNCIONES DE REGISTRO

console.assert (): aparece un mensaje de error en la consola si la afirmación es falsa. Si la afirmación es verdadera, no aparecerá nada. (NO ESTANDAR).

console.clear(): se usa para limpiar la consola, borra todo lo anterior a escribir esto.

console.error(“ ”): Muestra un mensaje de error en la consola con lo que nosotros coloquemos entre paréntesis. Se puede hacer desde la consola o desde el editor.

console.info (): emite un mensaje informativo con lo que nosotros coloquemos entre paréntesis. Se parece al console.log, pero info tira un mensaje informativo, en cambio log tira un mensaje de depuración. Tienen diferencias minimas.

console.table(): Es una función que toma un argumento obligatorio: data, que debe ser un array o un objeto, y un parámetro adicional: columns, y nos muestra una tabla en consola. Y NOS GENERA UNA TABLA EN CONSOLA.
console.warn(): Imprime un mensaje de advertencia en la consola web.
console.dir(): Despliega una lista interactiva de las propiedades del objeto JavaScript especificado.(NO ESTANDAR).

FUNCIONES DE CONTEO

console.count(): Registra el número de veces que se llama a count(). Esta función toma como argumento opcional una etiqueta.
console.countReset(): Resetea el contador console.count().

FUNCIONES DE AGRUPACION

console.group(): Crea un nuevo grupo dentro de la consola web. Dentro de los paréntesis podemos poner el nombre del grupo. Con este el grupo ya arranca abierto.
console.groupEnd(): Da por finalizados el o los grupos que creamos. Y todo lo que escribamos a partir de ese momento que afuera de los grupos.
console.groupColapsed(): empieza el grupo directamente cerrado. Es la única diferencia con group.

FUNCIONES DE TEMPORIZACION:

console.time(): Inicia un temporizador. Aunque no aparezca nada.
console.timeLog(): Nos dice cuanto tiempo paso hasta que escribimos este comando.
console.timeEnd(): Termina la cuenta del temporizador.

PROBLEMAS DE COFLA

- Solicitar los datos y decirle si aprobó o no.
-Mostrar todo esto con colores representativos en consola.
-Todo esto estructurado como tabla.

```
let materias = {

    fisica: [90,6,4,"fisica"],
    matematicas: [84,8,2,"matematicas"],
    logica: [92,8,4,"logica"],
    quimica: [96,8,4,"quimica"],
    calculo: [91,6,3,"calculo"],
    programacion: [79,7,4,"programacion"],
    biologia: [75,9,2,"biologia"],
    bbdd: [98,9,1,"bbdd"],
    algebra: [100,10,4,"algebra"]
}
```

```
const asistencia = () => {
    for (materia in materias) {
        let asistencias = materias[materia][0];
        let promedio = materias[materia][1];
        let trabajos = materias[materia][2]
        console.log(materias[materia][3]);
        if (asistencias >= 90){
            console.log("%cAsistencias en orden", "color:green");
        } else {
            console.log("%cMuchas inasistencias", "color:red");
        }
        if (promedio >= 7) {
            console.log("%cPromedio alcanzado", "color:green");
        } else {
            console.log("%cPromedio insuficiente", "color:red");
        }
        if (trabajos >= 4) {
            console.log("%cTrabajos practicos en orden",
"color:green");
        } else {
            console.log("%cFaltan trabajos practicos",
"color:red");
        }
    }
}
asistencia();
```

-Organizar las actividades diariamente.
-Utilizar la consola para organizar todo.
-El tiempo por tarea no debe superar las 4 horas.

```
let trabajo = "240 minutos de trabajo";
let estudio = "100 minutos de estudio";
let tp = "100 minutos trabajos practicos";
let homework = "30 minutos de cosas de la casa";
let descanso = "10 minutos de descanso"

for (var i = 0; i < 14; i++) {
```

```
    if (i == 0) {
        console.group("Semana 1");
    }
console.groupCollapsed ("DIA " +(i+1));
console.log (trabajo);
console.log (descanso);
console.log (estudio);
console.log (tp);
console.log (homework);
console.groupEnd();
if (i == 7) {
    console.groupEnd();
    console.groupCollapsed("semana 2")
}
}
console.groupEnd();
```

CAPITULO 6
EL ARBOL “DOM”

Nodo: Un nodo en el DOM, es cualquier etiqueta del cuerpo, como un párrafo, el mismo body o incluso las etiquetas de una lista. No siempre son etiquetas, una etiqueta crea un nodo, pero no todos los nodos son etiquetas.
TIPOS DE NODO
-Document: Es el nodo raíz, a partir del cual derivan el resto de nodos.
-Element: Nodos definidos por etiquetas html.
-Text: El texto dentro de un nodo element se considera un nuevo nodo hijo de tipo text.
-Attribute: Los atributos de las etiquetas definen nodos, (en JavaScript no los veremos cómo nodos, sino como información asociada al nodo de tipo element)
-Comentarios y otros: Los comentarios y otros elementos como las declaraciones doctype en cabecera de los documentos HTML generan nodos.

Document – **Metodos de selección de elementos.**
Son todos los métodos que nos van a permitir obtener los elementos o los grupos de elementos que queramos seleccionar.
-Document.getElementById(): Selecciona un elemento por ID.
-**getElementsByTagName():** Selecciona todos los elementos que coincidan con el nombre de la

etiqueta especificada.
-**querySelector():** Devuelve el primer elemento que coincida con el grupo especificado de selectores.
-**querySelectorAll():** Devuelve todos los elementos que coincidan con el grupo especificado de selectores.

Metodos para Definir, obtener y eliminar valores de atributos.
-variable.**setAttribute():** Modifica el valor de un atributo. El primer valor dentro de los paréntesis denomina el tipo de atributo al que queremos que cambie, y el segundo a la característica del atributo.

```
document.getElementById("parafo");
const rangoEtario = document.querySelector(".rangoEtario");

rangoEtario.setAttribute("type", "number");
```

-variable.**getAttribute():** obtiene el valor de un atributo, o lo que está entre comillas del atributo "type" en este caso. No da de resultado "range".

```
const rangoEtario = document.querySelector(".rangoEtario");

let valorDelAtributo = rangoEtario.getAttribute("type");

document.write(valorDelAtributo);
```

-variable.**removeAttribute():** Remueve el valor de un atributo. Remueve el atributo type del input.

```
const rangoEtario = document.querySelector(".rangoEtario");

let valorDelAtributo = rangoEtario.removeAttribute("type");
```

Atributos globales: TODOS los elementos html tienen estos atributos

- class: Listas de clases del elemento separadas por espacios.

-id: define un identificador único.
-title: Contiene un texto con información relacionada al elemento al que pertenece.
-style: Contiene declaraciones de estilo css para ser aplicadas al elemento.

-contenteditable: Indica si el elemento puede ser modificable por el usuario. Si en donde dice true, colocamos false. decimos que el usuario no lo puede editar.

```
const titulo = document.querySelector(".titulo");

titulo.setAttribute("contentEditable" , "true")
```

-dir: Indica la direccionalidad del texto, cambia la dirección del texto para que se lea de derecha a izquierda o de izquierda a derecha. De esta manera de lee de derecha a izquierda.

```
const titulo = document.querySelector(".titulo");

titulo.setAttribute("dir" , "rtl")
```

-hidden: Indica si el elemento aun no es o ya no es relevante, funciona parecido a la propiedad display:hidden de igual nombre de css. Lo que tiene es que si está definida sin importar el valor que tenga, se muestra igual o funciona igual.

```
const titulo = document.querySelector(".titulo");

titulo.setAttribute("hidden", "true")
```

-tabindex: Indica si el elemento puede obtener un focus de input. Nos permite hacer un focus o seleccionar lo que queramos con la tecla tab. Y el orden va a ser según el numero que pongamos al lado del “tabindex”.

```
const titulo = document.querySelector(".titulo");

titulo.setAttribute("tabindex", "3")
```

-title: Contiene un texto con información relacionada al elemento que pertenece. Al pasar el mouse por arriba del elemento nos va a aparecer el texto que nosotros queramos.

```
const titulo = document.querySelector(".titulo");

titulo.setAttribute("title", "Hola peter");
```

ATRIBUTOS DE LOS INPUT

Accedemos a los atributos directamente desde javaScript.

-className: Nos muestra lo que está dentro del atributo “class:”.

```
const titulo = document.querySelector(".titulo");

titulo.setAttribute("title", "Hola peter");
```

-value: Nos muestra el valor del input, o lo que ponemos dentro del atributo “value:”.

```
const input = document.querySelector(".input-normal");

document.write(input.value);
```

-type: De esta forma se puede modificar type, cambiando el input desde js. En este caso pasa de text a color.

```
const input = document.querySelector(".input-normal");

document.write(input.type = "color");
```

-accept: Este lo usamos para decir que es lo que va a aceptar, se usa para los input type:file. Se pueden poner varios formatos de archivos.

```
const input = document.querySelector(".input-normal");


document.write(input.accept = "image/png");
```

-form: Nos permite ejecutar un input dentro de un form. Se usa con id. Si un botón submit esta fuera del formulario, no va a mandar los datos del mismo. Pero si a ese botón que esta fuera del formulario le ponemos el atributo “form” con el id del formulario, este va a mandar los datos.

```
 <form id="form">

    <input type="text" class="input-normal"><br><br>
    <script src="Capitulo 6.js"></script>
</form>
<input type="submit" form="form">
```

-minLength: Es la mínima cantidad de caracteres que tiene que tener un input. En este caso, lo que escribamos en el input type:text. No debe tener menos de 4 caracteres.

```
const input = document.querySelector(".input-normal");

input.minLenght = 4;
```

-placeholder: Sirve igual que el de html, para poner dentro de un input lo que querramos.

```
const input = document.querySelector(".input-normal");

input.placeholder = "hola mundo";
```

-required: Funciona también igual que el de html, sirve para poner si el dato requerido es obligatorio o no.

```
const input = document.querySelector(".input-normal");

input.required = "true";
```

ATRIBUTO STYLE

Se usa camelCase para escribir las propiedades de css en js.

-style: Nos permite modificar algo del css con el atributo. De esta manera cambiamos el color del elemento sobre el que aplica la clase. Se coloca los atributos con camelCase.

```
const titulo = document.querySelector(".titulo");

titulo.style.color = "red";
```

```
const titulo = document.querySelector(".titulo");

titulo.style.backgroundColor = "red";
```

Clases, classList y métodos de classList

classList: Es una particularidad de las clases de los objetos, se pueden hacer un monton de cosas con las clases de los objetos.

```
const titulo = document.querySelector(".titulo");

titulo.classList
```

ATRIBUTOS:

.add(): Añade una clase. Pasa de ser “titulo” a “título grande”. Y podemos realizar cambios en css de manera más precisa. “.grande” y solo cambiaria ese elemento.

```
const titulo = document.querySelector(".titulo");

titulo.classList.add("grande")
```

.remove(): Remueve una clase. Le saca la clase grande a titulo.

```
const titulo = document.querySelector(".titulo");

titulo.classList.remove("grande")
```

.item(): Devuelve la clase del índice especificado. Acá obtenemos la clase que está en el índice que indicamos entre paréntesis y lo escribimos en la pagina. Según el orden de las clases. NO MODIFICA LAS CLASES, SOLO NOS DA ESTA INFORMACIÓN.

```
const titulo = document.querySelector(".titulo");

let valor = titulo.classList.item(1);

document.write(valor);
```

.contains(): Verifica si ese elemento posee o no la clase especificada. Y nos devuelve un valor booleano en caso positivo o negativo. Si pose o no esa clase.

```
const titulo = document.querySelector(".titulo");

let valor = titulo.classList.contains("grande");

document.write(valor);
```

.toggle(): agrega una clase en caso de que el elemento no la tenga y si la tiene, la elimina. Y si al lado de “grande” colocamos una coma y luego “true”, forzamos a que en caso de que la tenga a la clase, no la borre, y si no la tiene que la agregue. (“grande”, true). Si la clase no existe, la agrega y muestra “true”. Y si existe la saca y muestra “false”.

```
const titulo = document.querySelector(".titulo");

let valor = titulo.classList.toggle("grande");
```

replace(): Remplaza una clase por otra. Primero entre paréntesis y comillas colocamos la clase que queremos cambiar, y luego colocamos la clase por la cual queremos cambiarla. Si la clase que queremos cambiar no existe, devuelve false, si la encuentra y la cambia nos devuelve false.

```
const titulo = document.querySelector(".titulo");


let valor = titulo.classList.replace("grande", "chico");
```

MODIFICACIÓN Y OBTENCIÓN DE ELEMENTOS EN HTML

.textContent: Devuelve el texto de cualquier nodo, sin incluir el html. Solo el contenido, sin modificaciones por etiquetas como <b> , style o similares.

```
const titulo = document.querySelector(".titulo");

let resultado = titulo.textContent;

document.write(resultado);
```

.innerText: Devuelve el texto visible de un node element. Ya no se usa.

.innerHTML: Muestra todo el contenido HTML.

```
const titulo = document.querySelector(".titulo");

let resultado = titulo.innerHTML;

alert(resultado);
```

EN UN ALERT SE LE MEJOR QUE EN UN DOCUMENT.WRITE

.outerHTML: No solo nos muestra el texto y el código sino que las etiquetas y todo lo que contienen. O el elemento HTML completo. Con todas las etiquetas.

```
const titulo = document.querySelector(".titulo");

let resultado = titulo.outerHTML;
```

```
alert(resultado);
```

CREACIÓN DE ELEMENTOS

- createElement(): Se usa para crear elementos HTML. Entre paréntesis colocamos la etiqueta del elemento que queremos crear, siempre en MAYÚSCULA.

-createTextNode(): Crea un nodo texto.

-appendChild(): Los métodos de los child son los que se aplican solo a los child. Los child son los elementos que están dentro de otro elemento padre. Inserta un nuevo nodo dentro de la estructura DOM de un documento, y es la segunda parte del proceso central uno-dos, crear-y-añadir para construir páginas web a base de programación.

Aca podemos ver que primero creamos un div con clase “contenedor”, luego creamos el elemento “li”. Luego creamos un nodo texto y lo insertamos con appendChild dentro de “li”.

```
const contenedor = document.querySelector(".contenedor");

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

const textDelItem = document.createTextNode("este es un item de la lista");

item.appendChild(textDelItem);

console.log(item);
```

Y si todo esto lo queremos insertar dentro del div con clase contenedor hacemos asi. Y el texto nos aparecerá en la pantalla.

```
const contenedor = document.querySelector(".contenedor");

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

const textDelItem = document.createTextNode("este es un item de la
lista");

item.appendChild(textDelItem);

contenedor.appendChild(item)

console.log(item);
```

Si queremos añadir muchos elementos, esta sería la manera.

-createDocumentFragment(): Crea un nuevo **DocumentFragment** vacio, dentro del cual un nodo del DOM puede ser adicionado para construir un nuevo arbol DOM fuera de pantalla.

```
const contenedor = document.querySelector(".contenedor");


const fragmento = document.createDocumentFragment();

for (i = 0; i < 20; i++) {
    const item = document.createElement("LI");
    item.innerHTML = "Este es un item de la lista";
    fragmento.appendChild(item);
}

contenedor.appendChild(fragmento)

console.log(contenedor);
```

OBTENCION Y MODIFICACION DE CHILDS (HIJOS)

Propiedades:

-firstChild: Sirve para acceder al primer nodo hijo.

```
const contenedor = document.querySelector(".contenedor");


const primerHijo = contenedor.firstChild;
```

```
console.log(primerHijo);
```

-lastChild: Sirve para acceder al último nodo hijo.

```
const contenedor = document.querySelector(".contenedor");


const primerHijo = contenedor.lastChild;

console.log(primerHijo);
```

-firstElementChild: Nos permite obtener el primer elemento hijo, y de esta forma vamos a poder hacerlo con espacios, y nos va a decir exactamente lo que contiene esa etiqueta. MANERA CORRECTA DE HACERLO.

```
const contenedor = document.querySelector(".contenedor");


const primerHijo = contenedor.firstElementChild;

console.log(primerHijo);
```

-lastElementChild: Hace exactamente lo mismo que firstElementChild solo que del último. Muestra el elemento.

```
const contenedor = document.querySelector(".contenedor");

const primerHijo = contenedor.lastElementChild;

console.log(primerHijo);
```

-childNodes: Nos devuelve TODOS los nodos hijos en forma de lista. No es un array, es una Node list que se puede recorrer como un array, pero no lo es.

```
const contenedor = document.querySelector(".contenedor");

const primerHijo = contenedor.childNodes;

console.log(primerHijo);
```

-children: No son todos los nodos, sino que son los elementos hijos. Solo nos devuelve las etiquetas HTML, pero no se puede recorrer como un array.

```
const contenedor = document.querySelector(".contenedor");

const primerHijo = contenedor.children;

console.log(primerHijo);
```

Métodos de child

-replaceChild(): Remplaza el contenido de un elemento que nosotros indiquemos, por otro contenido que querramos.

```
const contenedor = document.querySelector(".contenedor");


const h2_nuevo = document.createElement("h2");
h2_nuevo.innerHTML = "titulo";

h2_antiguo = document.querySelector(".h2");

contenedor.replaceChild(h2_nuevo,h2_antiguo);
```

-removeChild(): Remueve el elemento que no queremos. Podemos eliminar un hijo de un elemento padre.

```
const contenedor = document.querySelector(".contenedor");


h2_antiguo = document.querySelector(".h2");

contenedor.removeChild(h2_antiguo);
```

-hasChildNodes(): Este se usa para verificar si tiene un elemento hijo o no. En este caso nos va a decir que si tiene hijos el contenedor.

```
const contenedor = document.querySelector(".contenedor");
```

```
h2_antiguo = document.querySelector(".h2");

let respuesta = contenedor.hasChildNodes();

if (respuesta == true) {
    document.write("El elemento si tiene hijos");
} else {
    document.write("El elemento no tiene hijos");
}
```

Propiedades de Parents (padres)

-parentElement: Se usa para seleccionar el padre elemento de un elemento. Siempre va a buscar una etiqueta HTML padre. Acá nos va a decir que el contenedor es el elemento padre, del h2.

```
const contenedor = document.querySelector(".contenedor");


h2_antiguo = document.querySelector(".h2");

console.log(h2_antiguo.parentElement);
```

-parentNode: Se usa para seleccionar el padre de un elemento. Va a buscar un nodo padre. Son casi iguales con el parentElement. Excepto en casos muy específicos.

Propiedades de sibling (hermanos) o los que es tan en la misma linea o rango.

-nextSibling: Sirve para mostrar el nodo hermano del elemento que queramos. Aca nos va a decir que el nodo hermano es un nodo text.

```
const contenedor = document.querySelector(".contenedor");


h2_antiguo = document.querySelector(".h2");

console.log(h2_antiguo.nextSibling);
```

-previousSibling: Sirve para mostrarnos el nodo anterior al elemento que queramos.

```
const contenedor = document.querySelector(".contenedor");


h2_antiguo = document.querySelector(".h2");

console.log(h2_antiguo.previousSibling);
```

-nextElementSibling: Este es el más indicado porque nos muestra el ELEMENTO hermano que le sigue. En este caso sería el h4.

```
const contenedor = document.querySelector(".contenedor");


h2_antiguo = document.querySelector(".h2");

console.log(h2_antiguo.nextElementSibling);
```

-previousElementSibling: Y este al igual que el anterior nos muestra el ELEMENTO hermano ANTERIOR. En este caso nos va a decir que “null”, porque el h2 no tiene un elemento anterior.

```
const contenedor = document.querySelector(".contenedor");


h2_antiguo = document.querySelector(".h2");

console.log(h2_antiguo.previousElementSibling);
```

CODIGO HTML USADO ANTERIORMENTE

```
<body>

    <div class="contenedor">
        <h2 class="h2">Un h2 comun</h2>
        <h4>Un h4 comun</h4>
        <p>Un simple parrafo</p>
    </div>
```

```
    <script src="Capitulo 6.js"></script>
</body>
```

NODOS-EXTRAS

```
<body>

    <div class="div">
        DIV 1
        <div class="div">
            DIV 2
            <div class="div-3"></div>
        </div>
    </div>
    <script src="Capitulo 6.js"></script>
</body>
```

-closest(): Sirve para seleccionar el elemento ascendente CONTENEDOR más cercano. En este caso nos dice que el elemento ascendente más cercano es DIV 2. Ahora si coloco el div dos en el mismo rango o línea que el div 3, me va a decir que el elemento ascendente PADRE más cercano es DIV 1.

```
const div = document.querySelector(".div-3");

console.log(div.closest(".div"))
```