COMO CREAR OBJETOS

FORMA 1: Con {}

```
const casa = {
    ventanas: 4,
    puertas: 3,
    banyos: 2,
    cocina: 1
}
Console.log(casa)
```

FORMA 2: PROTOTIPO OBJECT JS

```
const casa2 = new Object;
casa2.ventanas = 4;
casa2.puertas = 3;
casa2.banyos = 2;
casa2.cocina = 1;

console.log(casa2);
```

Forma 1 y 2 tienen como limitación que no las puedo usar para crear nuevos objetos iguales a ellos. Es decir, no sirven como molde

FORMA 3: FUNCIÓN CONSTRUCTORA

Función que sirve de modelo para crear objetos de tipo casa.

```
//Plantilla para crear objetos
function hogar(ventanas, puertas, banyos, cocina){
    this.ventanas = ventanas;
    this.puertas = puertas;
    this.banyos = banyos;
    this.cocina = cocina
}
// se crea un objeto a partir de la plantilla
const apartamento = new hogar(4,5,1,1);
const casa3 = new hogar(6,7,2,1);
// imprimir los objetos
console.log(apartamento);
console.log(casa3);
```

FORMA 3: CLASES E6

```
class planos {
    constructor(ventanas, puertas, banyos, cocina){
        this.ventanas = ventanas;
        this.puertas = puertas;
        this.banyos = banyos;
        this.cocina = cocina
    }
}
// crear el objeto
const nuevaCasa = new planos(4,2,3,1);
// imprimir el objeto
console.log(nuevaCasa);
```

ARRAYS

Como iniciar un array:

```
let lista = new Array();
let lista2 = [];
```

Como usar **Pop** (eliminar último elemento del array) Método Rápido:

```
let fruits = ["Apple", "Orange", "Pear"];
alert(fruits.pop()); // quita "Pear" y lo muestra en un alert
alert(fruits); // Apple, Orange
```

Como usar **Push** (añadir un elemento al final de la lista) método Rápido:

```
let fruits = ["Apple", "Orange"];
fruits.push("Pear");
alert( fruits ); // Apple, Orange, Pear
```

Como usar **Shift** (quita el Primer elemento de la lista) Método Lento:

```
let fruits = ["Apple", "Orange", "Pear"];
alert( fruits.shift() ); // quita Apple y lo muestra en un alert
alert( fruits ); // Orange, Pear
```

Como usar **Unshift** (añade un elemento en la Primera posición de la lista) Método Lento:

```
let fruits = ["Orange", "Pear"];
fruits.unshift('Apple');
alert( fruits ); // Apple, Orange, Pear
```

Como Recorrer una lista con un bucle for …of:

FOR … OF → solo devuelve el valor.

```
let fruits = ["Apple", "Orange", "Plum"];

// itera sobre los elementos del array
for (let fruit of fruits) {
  alert( fruit );
} // Apple, Orange, Plum
```

NO olvidar el LET

La forma más rápida de limpiar un array:

```
arr.length = 0;
```

METODOS ESPECIALES PARA ARRAYS

PARA AGREGAR REMOVER ELEMENTOS

splice(posicionInicialDeLaModificacion, NumeroDeElementosAEliminar , ...ElementosAAgregarDespuesDePosicionInicialDeLaModificacion) – opera sobre el array original.

Posición empieza en 1.

```
let array = [1, 2, 3, 4, 5];
let replaced = array.splice(1, 3, 'a', 'b', 'c');

console.log(array);  // Output: [1, 'a', 'b', 'c', 5]
console.log(replaced);  // Output: [2, 3, 4] Elementos remplazados
```

slice(start, end) – crea un nuevo array y copia elementos desde la posición start hasta end (no incluido) en el nuevo array.

Posicion empieza en 0:

```
let array = [1, 2, 3, 4, 5];
let sliced = array.slice(0, 3);

console.log(sliced);  // Output: [1, 2, 3]

let array1 = [1, 2, 3, 4, 5];
let sliced1 = array.slice(2);

console.log(sliced);  // Output: [3, 4, 5]
```

concat(...items) – devuelve un nuevo array: copia todos los elementos del array actual y le agrega items. Si alguno de los items es un array, se toman sus elementos.

```
let array1 = [1, 2, 3];
let item1 = 4;
let item2 = [5, 6];
let combined = array1.concat(item1, item2);

console.log(combined);  // Output: [1, 2, 3, 4, 5, 6]
```

PARA BUSCAR ENTRE ELEMENTOS

indexOf/lastIndexOf(item, posAPartirDeLaCualBuscar) – se utilizan para buscar la primera y la última aparición de un elemento específico en un array, respectivamente. Ambos métodos devuelven el índice del elemento encontrado o -1 si el elemento no está presente en el array.

```
let array = ['a', 'b', 'c', 'b', 'a'];
let index = array.indexOf('b');

console.log(index);  // Output: 1
```

```
let array = ['a', 'b', 'c', 'b', 'a'];
let lastIndex = array.lastIndexOf('b');

console.log(lastIndex);  // Output: 3
```

```
let array = ['a', 'b', 'c', 'b', 'a'];
let index = array.indexOf('b', 2);

console.log(index);  // Output: 3
```

includes(value) – devuelve true si el array tiene value, si no false. Búsqueda estricta que sea del mismo tipo de datos y el mismo valor.

```
let array = [1, 2, 3, 4, 5];
let result = array.includes(3);

console.log(result);  // Output: true
```

```
let array = [1, 2, 3, 4, 5];
let result = array.includes(6);

console.log(result);  // Output: false
```

find/filter(func) – filtra elementos a través de la función, devuelve el primer/todos los valores que devuelven true.

```
array.find(element => {
    // Código para verificar la condición
});

array.filter(element => {
    // Código para verificar la condición
});
```

Find sirve para encontrar el primer elemento que cumple con la condición:

```
let array = [1, 2, 3, 4, 5];
let result = array.find(element => element > 3);

console.log(result);  // Output: 4
```

Filter devuelve los elementos que cumple una condición:

```
let array = [1, 2, 3, 4, 5];
let result = array.filter(element => element % 2 === 0);

console.log(result);  // Output: [2, 4]
```

```
let products = [
    { name: 'iPhone', category: 'Electronics', price: 1000, stock: 10 },
    { name: 'T-Shirt', category: 'Clothing', price: 25, stock: 20 },
    { name: 'Headphones', category: 'Electronics', price: 150, stock: 5 },
    { name: 'Jeans', category: 'Clothing', price: 50, stock: 15 },
    { name: 'TV', category: 'Electronics', price: 800, stock: 3 }
  ];

  let filteredProducts = products.filter(product => {
    return product.category === 'Electronics' &&
           product.price < 1000 &&
           product.stock > 0 &&
           product.name.includes('o');
  });

  console.log(filteredProducts);
  [
    { name: 'Headphones', category: 'Electronics', price: 150, stock: 5 }
  ]
```

findIndex es similar a find, pero devuelve el índice del primer elemento que cumpla la condicion y -1 si no se encuentra.

```
let array = [
    { id: 1, name: 'John' },
    { id: 2, name: 'Jane' },
    { id: 3, name: 'Mike' }
  ];
  let index = array.findIndex(element => element.id === 2);

  console.log(index);  // Output: 1
```

some() es un método disponible en los arrays en JavaScript. Su objetivo es verificar si al menos un elemento del array cumple con una determinada condición. Retorna true si se cumple la condición para al menos un elemento, y false en caso contrario.

```
const books = [
    { title: 'The Great Gatsby', author: 'F. Scott Fitzgerald', year: 1925 },
    { title: 'To Kill a Mockingbird', author: 'Harper Lee', year: 1960 },
    { title: '1984', author: 'George Orwell', year: 1949 },
    { title: 'Pride and Prejudice', author: 'Jane Austen', year: 1813 }
];

const searchTerms = ['great', 'pride', 'brave'];

const result = books.some(book => {
    const titleWords = book.title.toLowerCase().split(' ');
    return searchTerms.some(term => titleWords.includes(term));
});

console.log(result);  // true
```

PARA ITERAR SOBRE ELEMENTOS

forEach(func) – Simplemente recorre la lista elemnto por elemento y puedes indicar que hacer sobre cada elemento

```
let array = [1, 2, 3, 4, 5];
array.forEach((element, index) => {
  array[index] = element * 2;
});

console.log(array);  // Output: [2, 4, 6, 8, 10]
```

PARA TRANSFORMAR EL ARRAY

map(func) – crea un nuevo array a partir de los resultados de llamar a la func para cada elemento.

```
let array = [1, 2, 3, 4, 5];
let doubledArray = array.map(element => element * 2);

console.log(doubledArray);  // Output: [2, 4, 6, 8, 10]
```

```
let array = ['apple', 'banana', 'orange'];
let uppercaseArray = array.map(element => element.toUpperCase());

console.log(uppercaseArray);  // Output: ['APPLE', 'BANANA', 'ORANGE']
```

sort(func) – ordena el array y lo devuelve. Trabaja sobre el array original

En este caso a-b indica que los números negativos van antes, es decir ordena de menor a mayor

```
let array = [
    { name: 'John', age: 25 },
    { name: 'Jane', age: 30 },
    { name: 'Mike', age: 20 }
  ];

  array.sort((a, b) => a.age - b.age);

  console.log(array);
  /*
  { name: 'Mike', age: 20 },
  { name: 'John', age: 25 },
  { name: 'Jane', age: 30 }
  */
```

```
let array = ['banana', 'apple', 'orange'];
array.sort();

console.log(array);  // Output: ['apple', 'banana', 'orange']
```

reverse() – ordena el array de forma inversa y lo devuelve.

```
let array = [1, 2, 3, 4, 5];
array.reverse();

console.log(array);  // Output: [5, 4, 3, 2, 1]
```

```
let array = ['apple', 'banana', 'orange'];
array.reverse();

console.log(array);  // Output: ['orange', 'banana', 'apple']
```

split/join – convierte una cadena en un array y viceversa.

Split: convierte un string en un array con cada elemento de la cadena como elemento del string

```
let texto = "Hola, cómo estás?";
let palabras = texto.split(" ");

console.log(palabras);  // Output: ["Hola,", "cómo", "estás?"]
```

```
let texto = "Hola";
let caracteres = texto.split("");

console.log(caracteres);  // Output: ["H", "o", "l", "a"]
```

Join: convierte los elementos de un array en una cadena

```
let palabras = ["Hola,", "cómo", "estás?"];
let texto = palabras.join(" ");

console.log(texto);  // Output: "Hola, cómo estás?"
```

```
let numeros = [1, 2, 3, 4, 5];
let texto = numeros.join();

console.log(texto);  // Output: "1,2,3,4,5"
```

reduce/reduceRight(func, initial) – se utiliza para reducir los elementos de un array a un único valor, aplicando una función acumuladora a cada elemento. El método reduceRight() funciona de manera similar, pero comienza desde el último elemento del array hacia el primero.

En este ejemplo, se utiliza reduce() para sumar todos los elementos del array array. La función acumuladora (acumulador, elemento) => acumulador + elemento suma el elemento actual al acumulador en cada iteración, comenzando con un valor inicial de 0.

```
let array = [1, 2, 3, 4, 5];
let suma = array.reduce((acumulador, elemento) => acumulador + elemento, 0);

console.log(suma);  // Output: 15
```

En este ejemplo, se utiliza reduce() para concatenar los elementos del array array en una cadena de texto. La función acumuladora (acumulador, elemento) => acumulador + ' ' + elemento agrega un espacio entre cada elemento concatenado.

```
let array = ['Hola', 'cómo', 'estás'];
let texto = array.reduce((acumulador, elemento) => acumulador + ' ' + elemento);

console.log(texto);  // Output: "Hola cómo estás"
```

En este ejemplo, se utiliza reduce() para encontrar el número más grande en el array array. La función acumuladora Math.max(acumulador, elemento) compara el acumulador con el elemento actual y devuelve el número más grande.

```
let array = [10, 5, 20, 8, 15];
let maximo = array.reduce((acumulador, elemento) => Math.max(acumulador, elemento));

console.log(maximo);  // Output: 20
```

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

let contador = numeros.reduce((acumulador, numero) => {
  if (numero in acumulador) {
    acumulador[numero] += 1;
  } else {
    acumulador[numero] = 1;
  }
  return acumulador;
}, {});

console.log(contador);
// Resultado: {1: 4, 2: 1, 3: 1, 4: 1, 5: 1}
```

TIEMPO

Formato de texto para una fecha: Fecha = '2023-06-15'

Date.parse(fecha): transforma la cadena de fecha en **segundos** para ver si la fecha es válida.

Date.now(): da la fecha actual en **segundos**.

let fechaLocal = **new Date(fecha a transformar).toLocaleString():** fecha en formato localizado

let fechaFormateada = **new Date(fecha a formatear).toISOString();** // --> AAAA-MM-DDTHH:mm:ss.sssZ

DOM

Añadir al html el archivo .js, cuando contienen imports se llaman mediante modulos:

```
<script type="module" src="ruta del js"></script>
```

Si no se usan imports basta con:

```
<script src="rudaDel.js"></script>
```

Como importar librerías propias:

```
import * as DarNombreALaLibreria from 'rutaDelArchivo.js';
```

El objeto **document** es el punto de entrada a la página. Con él podemos cambiar o crear cualquier cosa en la página.

For .. of para iterar sobre una colección de elementos:

```
// Supongamos que tienes una lista de elementos <li> dentro de un <ul> con id "miLista"
const lista = document.getElementById('miLista');
const elementos = lista.getElementsByTagName('li');

// Iterar sobre los elementos usando for...of
for (const elemento of elementos) {
  // Realizar alguna acción con cada elemento
  elemento.style.color = 'red';
}
```

Métodos para buscar nodos en DOM

Si un elemento tiene el atributo id, podemos obtener el elemento usando el método **document.getElementById(id)**, sin importar dónde se encuentre.

```
// Obtener una referencia al elemento <div> con id "miDiv"
const divElement = document.getElementById('miDiv');

// Realizar alguna acción con el elemento
divElement.style.backgroundColor = 'blue';
divElement.textContent = 'Nuevo contenido';
```

Sin duda el método más versátil, **elem.querySelectorAll(css)** devuelve todos los elementos dentro de elem que coinciden con el selector CSS dado.

```
// Obtener una lista de elementos <li> dentro del <ul>
const listItems = document.querySelectorAll('ul li');

// Iterar sobre los elementos y realizar alguna acción
listItems.forEach((item) => {
  item.style.color = 'red';
});
```

```
// Obtener todos los elementos descendientes de la clase "container" que sean <li> o <p>
const elements = document.querySelectorAll('.container li, .container p');

// Iterar sobre los elementos y mostrar su contenido
elements.forEach((element) => {
  console.log(element.textContent);
});
```

La llamada a **elem.querySelector(css)** devuelve el primer elemento para el selector CSS dado.

```
// Obtener el primer elemento <li> dentro del elemento con id "container"
const element = document.querySelector('#container li');

// Mostrar el contenido del elemento
console.log(element.textContent);
```

El **elem.matches(css)** no busca nada, sólo comprueba si el elem coincide con el selector CSS dado. Devuelve true o false.

```html
<ul id="myList">
    <li class="item">Elemento 1</li>
    <li class="item">Elemento 2</li>
    <li class="item highlight">Elemento 3</li>
    <li class="item">Elemento 4</li>
</ul>
```

```javascript
const listItems = document.querySelectorAll('.item');

listItems.forEach(item => {
  if (item.matches('.highlight')) {
    console.log('El elemento coincide con el selector CSS ".highlight"');
  } else {
    console.log('El elemento no coincide con el selector CSS ".highlight"');
  }
});
```

En este ejemplo, seleccionamos todos los elementos con la clase "item" dentro de la lista con id "myList". Luego, utilizamos un bucle forEach para recorrer cada elemento de la lista.

Dentro del bucle, verificamos si cada elemento coincide con el selector CSS ".highlight" utilizando matches. Si el elemento tiene la clase "highlight", se imprimirá en la consola el mensaje "El elemento coincide con el selector CSS ".highlight"". De lo contrario, se imprimirá el mensaje "El elemento no coincide con el selector CSS ".highlight"".

elem.getElementsByTagName(tag) busca elementos con la etiqueta dada y devuelve una colección con ellos. El parámetro tag también puede ser un asterisco "*" para “cualquier etiqueta”.

```javascript
// se obtiene el elemento div con el id 'myDiv'
const myDiv = document.getElementById('myDiv');
// se obtienen los elementos p de dicho div
const paragraphs = myDiv.getElementsByTagName('p');
// se imprime el contenido de dicho p
for (let i = 0; i < paragraphs.length; i++) {
  console.log(paragraphs[i].textContent);
}
```

elem.getElementsByClassName(className) devuelve elementos con la clase dada.

```javascript
const myDiv = document.getElementById('myDiv');
const paragraphs = myDiv.getElementsByClassName('paragraph');

for (let i = 0; i < paragraphs.length; i++) {
  console.log(paragraphs[i].textContent);
}
```

document.getElementsByName(name) devuelve elementos con el atributo name dado, en todo el documento. Muy raramente usado.

```html
<div>
    <input type="radio" name="color" value="red"> Rojo
    <input type="radio" name="color" value="blue"> Azul
    <input type="radio" name="color" value="green"> Verde
</div>
```

```javascript
const colorInputs = document.getElementsByName('color');

colorInputs.forEach(input => {
  input.addEventListener('change', () => {
    console.log(`Seleccionaste el color: ${input.value}`);
  });
});
```

En este ejemplo, utilizamos getElementsByName para obtener una colección de elementos de tipo radio con el atributo name igual a "color".

Luego, utilizamos el método forEach para iterar sobre cada uno de los elementos y agregar un event listener para el evento change. Cuando se produce un cambio en la selección del color, se ejecuta la función callback y mostramos en la consola el valor del color seleccionado.

Modificar/ alterar Elementos de DOM

La propiedad **innerHTML** se utiliza para obtener o establecer el contenido HTML de un elemento. Puede contener etiquetas HTML, texto y elementos anidados.

Las etiquetas html las interpreta como tal.

```html
<div id="myDiv"></div>
```

```javascript
const divElement = document.getElementById('myDiv');
divElement.innerHTML = '<h1>Nuevo Título</h1><p>Nuevo párrafo.</p>';
```

Nuevo Título

Nuevo párrafo.

Poniendo += concatena lo anterior con lo nuevo

```javascript
divElement.innerHTML += '<h1>Nuevo Título 2</h1>'
```

Nuevo Título

Nuevo párrafo.

Nuevo Título 2

La propiedad **innerText** en JavaScript se utiliza para obtener o establecer el texto visible de un elemento, incluyendo su contenido y el de sus elementos descendientes, pero sin incluir elementos ocultos mediante CSS. Las etiquetas de html las interpreta como texto

```
// Ejemplo con innerText
const elemento1 = document.getElementById("miElemento");
console.log(elemento1.innerText);
// Salida: "Título\nEste es un párrafo con un span."

// Ejemplo con innerHTML
const elemento2 = document.getElementById("miElemento");
console.log(elemento2.innerHTML);
// Salida: "<h1>Título</h1><p>Este es un párrafo <span>con un span</span>.</p>"
```

Crear Elementos en DOM añadir ID/class y Guardar

El método **createElement()** se utiliza para crear un nuevo elemento HTML en el DOM. Recibe como argumento el nombre de la etiqueta del elemento que se desea crear y devuelve el nuevo elemento creado.

```
// encontrar el contenedor
let contenedor = document.getElementById(idElemento);

// Crear un elemento <p>
const nuevoParrafo = document.createElement('p');

// Establecer atributos al elemento
nuevoParrafo.id = 'miParrafo';  // Establecer el ID del elemento
nuevoParrafo.className = 'parrafoDestacado';  // Establecer la clase del elemento

// Añadir texto al elemento
nuevoParrafo.innerText = '¡Hola, mundo!';

// Agregar el elemento al contenedor
contenedor.append(nuevoParrafo);
```

No olvidar el new al crear objetos!!!!!!

```
let gasto1 = new gesPres.CrearGasto(
```

Como recorrer las propiedades de un objeto:

```
const persona = {
    nombre: 'Juan',
    edad: 30,
    ocupacion: 'programador'
  };

  // Recorrer las propiedades del objeto
  for (const [clave, valor] of Object.entries(persona)) {
    console.log(clave, valor);
}
```

```
nombre Juan
edad 30
ocupacion programador
```

EVENTOS

Eventos del mouse:

- `click` – cuando el mouse hace click sobre un elemento (los dispositivos touch lo generan con un toque).
- `contextmenu` – cuando el mouse hace click derecho sobre un elemento.
- `mouseover` / `mouseout` – cuando el cursor del mouse ingresa/abandona un elemento.
- `mousedown` / `mouseup` – cuando el botón del mouse es presionado/soltado sobre un elemento.
- `mousemove` – cuando el mouse se mueve.

Solicitar Datos Prompt

```
prompt(mensaje, valorPredeterminado);
```

```
const nombre = prompt("Ingresa tu nombre", "John Doe");
```

Pasar de string a número (float):

```
const str = "3.14";
const num = parseFloat(str);
console.log(num); // Output: 3.14
```

OBJETO MANEJADOR DE EVENTOS (handleEvent)

```
// Definición de la función constructora
function ClickHandle() {
    this.handleEvent = function(event) {
      console.log("Se ha hecho clic en el elemento");
      console.log("Tipo de evento: " + event.type);
      console.log("Coordenadas del clic: X=" + event.clientX + ", Y=" + event.clientY);
    };
  }

  // Creación de un objeto handler
  var clickHandler = new ClickHandle();

  // Asignar el objeto handler al evento 'click' de un elemento
  var elemento = document.getElementById('miElemento');
  elemento.addEventListener('click', clickHandler);
```

```
let formulario = document.getElementById("anyadircoche");
formulario.addEventListener("submit", AnyadirCocheWeb);

function CrearLibro(event) {
    event.preventDefault();
    let form = document.getElementById("id-form");
    let autor = form.autor.value;
    let titulo = form.titulo.value;
    let libro = new base.Libro(autor, titulo);
    base.CrearLibro(libro);
}
let form = document.getElementById("id-form");
form.addEventListener("submit", CrearLibro);
```

FORMULARIOS:

Hacer una copia de un formulario en js:

```
let copia = document.getElementById("id-del-template").content.cloneNode(true);
```

Acceder al form del template:

```
var form = copia.querySelector("form");
```

VueJS:

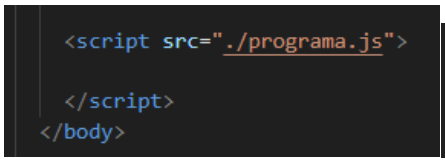
Página de vueJS: https://v2.vuejs.org/v2/guide/

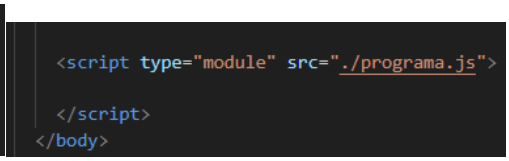
Script de desarrollo:

```
<!-- development version, includes helpful console warnings -->
<script
src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
```

Script de Produccion:

Los Scripts se cargan al final de body. Pero si lo haces con type module se puede poner donde sea ya que siempre se cargará primero.





Agregar la biblioteca Vue: <script src="https://unpkg.com/vue"></script>

Acceder a una propiedad: {{}}