



PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL
FORMATO MATERIAL DE APOYO – ACTIVIDADES

**MATERIAL DE APOYO CODIFICACIÓN CLASES Y
MANEJO DE MODULOS EN JAVASCRIPT**

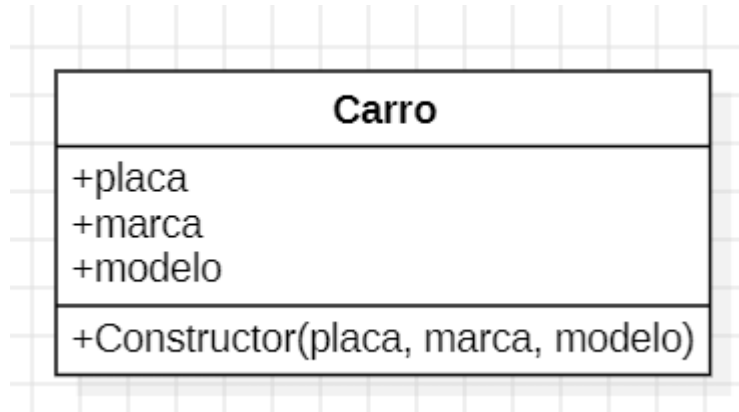
Programación Orientada a Objetos (POO)

¿Qué es una clase?

una **clase** sólo es una forma de organizar código de forma entendible con el objetivo de simplificar el funcionamiento de nuestro programa. Además, hay que tener en cuenta que las clases son «conceptos abstractos» de los que se pueden crear objetos de programación, cada uno con sus características concretas. La **clase** es el **concepto abstracto** de un objeto, mientras que el **objeto** es el elemento final que se basa en la clase

Ejercicios

1. Codificar clase Carro con 3 atributos públicos y el constructor.





Código Javascript

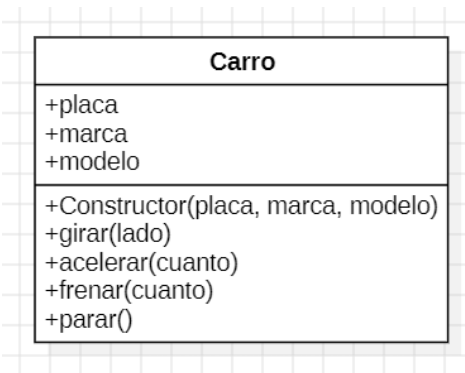
```
Clases > JS carro.js > Carro
1 class Carro{
2
3     /**
4     *
5     * @param {string} placa
6     * @param {string} marca
7     * @param {int} modelo
8     */
9     constructor(placa, marca, modelo){
10         this.placa=placa
11         this.marca=marca
12         this.modelo=modelo
13         this.velocidad=0
14     }
15 }
16
```

Código Javascript para crear un objeto de tipo Carro

```
let miCarro = new Carro("xxx123","Renault","2015")
//imprimir consola el tipo del objeto miCarro
console.log(typeof(miCarro))
//modificar el atribuo modelo del carro
miCarro.modelo=2016
//imprimir los atributos del objeto miCarro
console.log(`Placa: ${miCarro.placa}`)
console.log(`Marca: ${miCarro.marca}`)
console.log(`Modelo: ${miCarro.modelo}`)
```



Ahora modificamos la clase agregándole unas operaciones o métodos.



Código de la clase

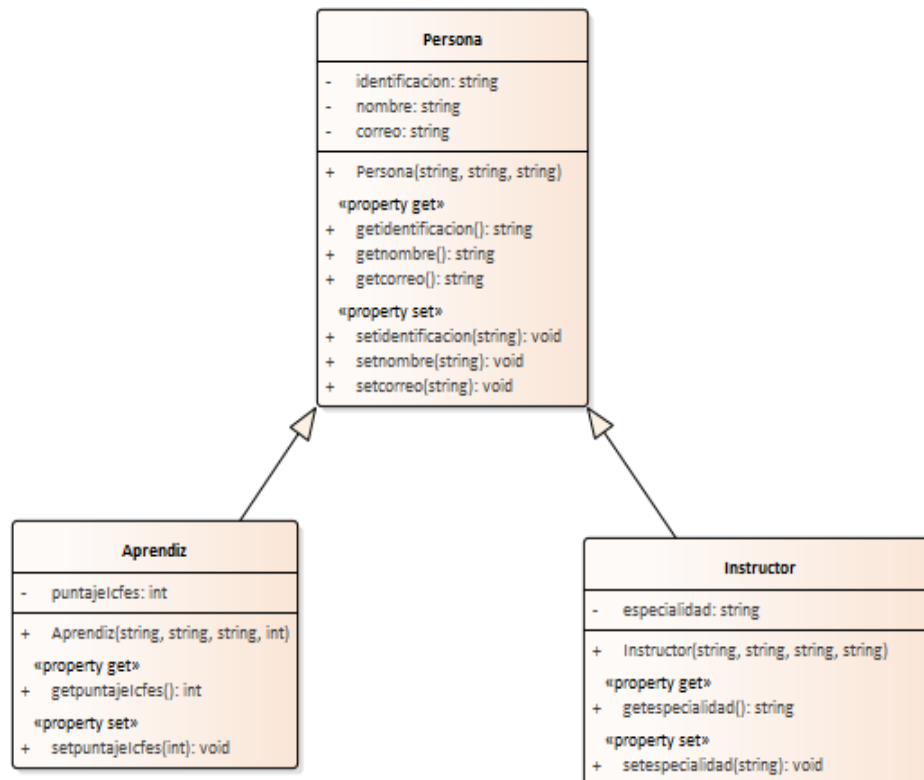
```
Clases > JS carro.js > ...
1  class Carro{
2      /**
3       *
4       * @param {string} placa
5       * @param {string} marca
6       * @param {int} modelo
7       */
8      constructor(placa, marca, modelo){
9          this.placa=placa
10         this.marca=marca
11         this.modelo=modelo
12         this.velocidad=0
13     }
14
15     Tabnine | Edit | Test | Explain | Document
16     acelerar(valor){
17         this.velocidad +=valor
18     }
19
20     Tabnine | Edit | Test | Explain | Document
21     frenar(valor){
22         this.velocidad -=valor
23     }
24
25     Tabnine | Edit | Test | Explain | Document
26     parar(){
27         this.velocidad=0
28     }
29
30     Tabnine | Edit | Test | Explain | Document
31     girar(lado){
32         return "El carro ha girado a la " + lado
33     }
34 }
```



Código javascript implementación

```
//crear el objeto miCarro de tipo Carro
let miCarro = new Carro("xxx123","Renault","2015")
//girar el carro a la derecha e imprimir lo que retorna
console.log(miCarro.girar("derecha"))
//acelerar el carro en 100km +
miCarro.acelerar(100)
//imprimir datos del carro y a que velocidad aceleró
console.log(`El carro con placa ${miCarro.placa} ha acelerado a ${miCarro.velocidad} km/hora`)
//frenar el carro a cierto kilometraje
miCarro.frenar(50)
//imprimir datos del carro y a que velocidad frenó
console.log(`El carro con placa ${miCarro.placa} ha frenado a ${miCarro.velocidad} km/hora`)
```

2. Codificar modelo de clases que representan **herencia de clases**



En el diagrama encontramos que los atributos son privados y se crean unos métodos públicos para poder acceder a ellos o para modificarlos



Código javascript Clase Persona: Clase Padre

```
Clases > JS hercia.js > Persona
1  class Persona{
2
3      /**
4      * Definir atributos cuando son privados
5      * anteponiendo caracter # es obligatorio
6      * definirlos
7      */
8      #nombre=null
9      #correo=null
10     /**
11     *
12     * @param {string} nombre
13     * @param {string} correo
14     */
15     constructor(nombre, correo){
16         this.#nombre=nombre
17         this.#correo=correo
18     }
19
20     Tabnine | Edit | Test | Explain | Document
21     getNombre(){
22         return this.#nombre
23     }
24
25     Tabnine | Edit | Test | Explain | Document
26     getCorreo(){
27         return this.#correo
28     }
29
30     Tabnine | Edit | Test | Explain | Document
31     setNombre(nombre){
32         this.#nombre=nombre
33     }
34
35     Tabnine | Edit | Test | Explain | Document
36     setCorreo(correo){
37         this.#correo=correo
38     }
39 }
```



Código de las clases Hijas: Para representar la herencia al definir la clase agregamos la palabra `extend` y después el nombre de la clase Padre.

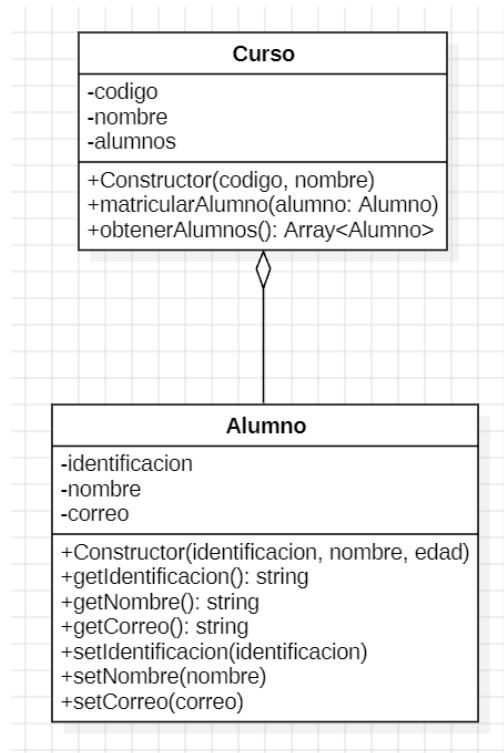
```
37 class Aprendiz extends Persona{
38     /**
39      * definir atributo privado icfes
40      */
41     #icfes=0
42
43     /**
44      *
45      * @param {int} icfes
46      * @param {string} nombre
47      * @param {string} correo
48      */
49     constructor(icfes, nombre, correo){
50         //con super llamamos al constructor de la clase padre
51         super(nombre, correo)
52         this.#icfes=icfes
53     }
54
55     Tabnine | Edit | Test | Explain | Document
56     getIcfes(){
57         return this.#icfes
58     }
59
60     Tabnine | Edit | Test | Explain | Document
61     setIcfes(icfes){
62         this.#icfes=icfes
63     }
64 }
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
```

Implementación: Creación de objetos de tipo Aprendiz e Instructor

```
90 //crear un objeto de tipo Aprendiz
91 let aprendiz = new Aprendiz(250, "Pedro", "peter@gmail.com")
92
93 print(`Nombre Aprendiz: ${aprendiz.getNombre()}`)
94 print(`Correo Aprendiz: ${aprendiz.getCorreo()}`)
95 print(`Puntaje Icfes Aprendiz: ${aprendiz.getIcfes()}`)
96
97 //crear un objeto de tipo Instructor
98 let instructor = new Instructor("Software", "Juan", "juan@sena.edu.co")
99
100 print(`Nombre Instructor: ${aprendiz.getNombre()}`)
101 print(`Correo Instructor: ${aprendiz.getCorreo()}`)
102 print(`Especialidad Instructor: ${aprendiz.getEspecialidad()}`)
103
```



3. Representación relación de agregación y composición



En la relación de agregación o composición en la clase donde llega el rombo es la clase contenedora y aquí se crea un arreglo o lista de la otra clase. En nuestro ejemplo la clase Curso es la clase contenedora, por lo que aquí debemos crear un atributo como arreglo o lista de alumnos.

```
Clases > JS cursoAlumno.js > Alumno > setCorreo
1 class Alumno{
2     /**
3      * definición de atributos privados
4      */
5     #identificacion=null
6     #nombre=null
7     #correo=null
8     /**
9      *
10     * @param {string} identificacion
11     * @param {string} nombre
12     * @param {string} correo
13     */
14     constructor(identificacion,nombre,correo){
15         this.#identificacion=identificacion
16         this.#nombre=nombre
17         this.#correo=correo
18     }
19     Tabnine | Edit | Test | Explain | Document
20     getIdentificacion(){
21         return this.#identificacion
22     }
23     Tabnine | Edit | Test | Explain | Document
24     getNombre(){
25         return this.#nombre
26     }
27     Tabnine | Edit | Test | Explain | Document
28     getCorreo(){
29         return this.#correo
30     }
31 }
```

```
30
31 Tabnine | Edit | Test | Explain | Document
32 setIdentificacion(identificacion){
33     this.#identificacion=identificacion
34 }
35 Tabnine | Edit | Test | Explain | Document
36 setNombre(nombre){
37     this.#nombre=nombre
38 }
39 Tabnine | Edit | Test | Explain | Document
40 setCorreo(correo){
41     this.#correo=correo
42 }
```



```
44 class Curso{
45     /**
46      * Definición de atributos privados
47      */
48     #codigo=null
49     #nombre=null
50     #alumnos=null
51
52     /**
53      *
54      * @param {int} codigo
55      * @param {string} nombre
56      */
57     constructor(codigo,nombre){
58         this.#codigo=codigo
59         this.#nombre=nombre
60         //inicializa la lista vacía de alumnos
61         this.#alumnos=[]
62     }
63
64     Tabnine | Edit | Test | Fix | Explain | Document
65     getCodigo(){
66         return this.#codigo
67     }
68
69     Tabnine | Edit | Test | Explain | Document
70     getNombre(){
71         return this.#nombre
72     }
73
74     Tabnine | Edit | Test | Explain | Document
75     matricularAlumno(alumno){
76         this.#alumnos.push(alumno)
77     }
78
79     Tabnine | Edit | Test | Explain | Document
80     getAlumnos(){
81         return this.#alumnos
82     }
83 }
```




Uso de Módulos en Javascript

¿Qué es un Módulo?

Un módulo es simplemente un archivo. Un script es un módulo. Tan sencillo como eso. Los módulos pueden cargarse entre sí y usar directivas especiales `export` e `import` para intercambiar funcionalidad, llamar a funciones de un módulo de otro:

- La palabra clave **export** etiqueta las variables y funciones que necesitan ser accesibles desde fuera del módulo actual.
- **import** permite importar funcionalidades desde otros módulos.

A partir de **ECMAScript** se introduce una característica nativa denominada **Módulos ES** (ESM), que permite la importación y exportación de fragmentos de datos entre diferentes ficheros Javascript, eliminando las desventajas que teníamos hasta ahora y permitiendo trabajar de forma más flexible en nuestro código Javascript.

Para trabajar con **módulos** tenemos a nuestra disposición las siguientes palabras clave:

- **export**: Pone los datos indicados (variables, funciones, clases...) a disposición de otros ficheros
- **import**: Incorpora datos (variables, funciones, clases...) desde otros ficheros .js al código actual
- **import()**: Permite importar módulos de forma más flexible, en tiempo real (imports dinámicos).

Ejemplo2

1. Aquí un archivo llamado **datos.js** que exporta 3 variables como se muestra a continuación

```
modulos > JS datos.js > ...  
1   export let empleado="Monik Galindo"  
2   export let cargo="Gerente de Meradeo"  
3   export let sueldo=4350000  
4  
5
```



Ahora como **importar** los datos **exportados** del archivo **datos.js** en otro **archivo js**

```
modulos > JS main.js
1  //importar elementos exportados por el archivo datos.js
2  import {empleado,cargo, sueldo} from './datos.js'
3
4  console.log("DATOS DEL EMPLEADO")
5  console.log(`Empleado: ${empleado}`)
6  console.log(`Cargo: ${cargo}`)
7  console.log(`Sueldo: ${sueldo}`)
8
```

2. El mismo ejemplo pero definiendo las variables y al final exportar.

```
modulos > JS datos.js > ...
1  let empleado="Monik Galindo"
2  let cargo="Gerente de Meradeo"
3  let sueldo=4350000
4
5
6  export {empleado,cargo,sueldo}
7
```

3. Uso del export default normalmente utilizando para exportar clases.

```
modulos > JS libro.js > ...
1  export default class Libro{
2
3      #paginas=0
4      constructor(titulo,autor,paginas){
5          this.titulo=titulo
6          this.autor=autor
7          this.#paginas=paginas
8      }
9
10     Tabnine | Edit | Test | Explain | Document
11     getPaginas(){
12         return this.#paginas
13     }
14 }
```



Importar datos que son **exportados** como **default**

```
modulos > JS main.js > ...
1  /**
2   * Cuando importamos un elementoe exportado
3   * como default no lo encerramos entre llaves
4   * como se muestra a continuación
5   */
6  import Libro from "../libro.js"
7
8  let libro=new Libro("Cien años de Soledad","Gabriel García",250)
9
10 console.log(`DATOS DEL LIBRO`)
11 console.log(`Título: ${libro.titulo}`)
12 console.log(`Autor: ${libro.autor}`)
13 console.log(`Número Páginas: ${libro.getPaginas()}`)
14
```

4. Uso de **Export default** y **export solo**

```
modulos > JS datos.js > ...
1  export let empleados=[]
2
3  export default function agregarEmpleado(nombre,cargo,sueldo){
4    let emp = {
5      "nombre":nombre,
6      "cargo":cargo,
7      "sueldo": sueldo
8    }
9    empleados.push(emp)
10 }
11
```



Importar los datos exportados en otro archivo.

```
modulos > JS main.js > ...
1  /**
2   * improtamos la función agregarEmpleado fuera
3   * de las llaves porque fue exportada como default
4   * y empelados dentro de llaves
5   */
6  import agregarEmpleado,{empleados} from "../datos.js"
7
8  //agregar empleados
9
10 agregarEmpleado("María","Gerente",5000000)
11 agregarEmpleado("Pedro","Secretario",1450000)
12 agregarEmpleado("Luna","Vendedora",1570000)
13
14 //listar los empleados
    Tabnine | Edit | Test | Explain | Document
15 empleados.forEach(empleado => {
16     console.log(`Nombre: ${empleado.nombre}`)
17     console.log(`Cargo: ${empleado.cargo}`)
18     console.log(`Sueldo: ${empleado.sueldo}`)
19     console.log(".".repeat(20))
20 });
21
```



5. Uso de módulos en una aplicación web. La aplicación web debe permitir gestionar los libros de una biblioteca donde le permita inicialmente agregar los libros y los pueda mostarr en una tabla en la misma interfaz

Interfaz propuesta

GESTIÓN LIBROS BIBLIOTECA

Título

Autor

Editorial

Idioma

Español ▾

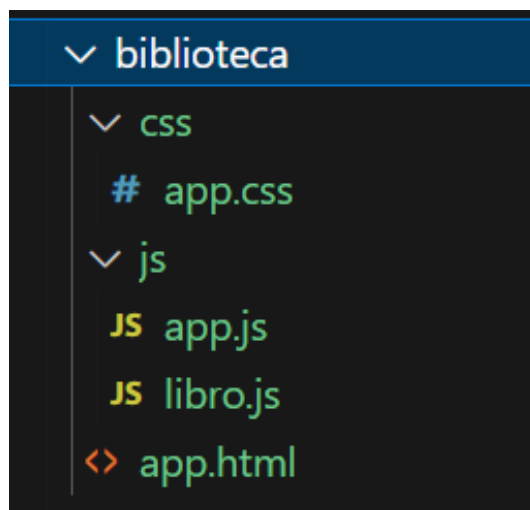
Agregar

Consultar

Eliminar

Título	Autor	Editorial	Idioma
--------	-------	-----------	--------

Estructura de Carpetas y archivos





- **libro.js:** El archivo contiene la clase libro
- **lpp.js:** archivo que se encarga de recibir los datos de la web, importar la clase Libro, crear los libros y guardarlos en un arreglo.
- **lpp.css:** Hoja de estilos

Código clase Libro

```
modulos > biblioteca > js > JS libro.js > Libro > constructor
1  export default class Libro{
2
3      constructor(titulo, autor, editorial, idioma){
4          this.titulo=titulo
5          this.autor=autor
6          this.editorial=editorial
7          this.idioma=idioma
8      }
9  }
```

Código archivo app.js

```
modulos > biblioteca > js > JS app.js > btnConsultar.addEventListener("click") callback
1  /**
2   * Importamos la clase Libro
3   */
4  import Libro from "../libro.js"
5
6  /**
7   * Definición de arreglo libros
8   */
9  let libros=[]
10 let libroBuscado=null //para guardar cuando se consulta
11
12 /**
13  * definición de objetos que van a representar
14  * los botones de Agregar y Eliminar de la interfaz
15  */
16 let btnAgregar = document.getElementById("btnAgregar")
17 let btnConsultar = document.getElementById("btnConsultar")
18 let btnEliminar = document.getElementById("btnEliminar")
19
```



```
42
43  /**
44   * Función que responde al evento click del
45   * botón agregar que permite agregar el libro
46   * Se crea un objeto de tipo libro con los datos
47   * de la interfaz y depues se agrega al arreglo
48   * libros. Por último llama a una funció para
49   * mostrarlo en la tabla.
50   */
    Tabnine | Edit | Test | Explain | Document
51  btnAgregar.addEventListener("click",function(){
52    //validar los campoo que no vengan vacíos
53    //crear un objeto de tipo libro
54    let titulo = document.getElementById("txtTitulo").value
55    let autor = document.getElementById("txtAutor").value
56    let editorial = document.getElementById("txtEditorial").value
57    let idioma = document.getElementById("cbIdioma").value
58    //se crea el objeto libro de tipo Libro
59    let libro = new Libro(titulo, autor, editorial,idioma)
60    //agregar el objeto creado al arreglo libros
61    libros.push(libro)
62    //agregar a la tabla el nuevo libro
63    console.log(libros)
64    mostrarLibrosTabla()
65  })
66
```

```
67  /**
68   * Función que crea de manera dinámica la tabla
69   * con los libros que hay en el arreglo libros
70   */
    Tabnine | Edit | Test | Explain | Document
71  function mostrarLibrosTabla(){
72    let datosLibros = document.getElementById("datosLibros")
73    datosLibros.innerHTML="" //limpia la tabla
74    libros.forEach(libro => {
75      let fila = document.createElement("tr")
76      let colTitulo = document.createElement("td")
77      colTitulo.textContent=libro.titulo
78      fila.appendChild(colTitulo)
79      let colAutor = document.createElement("td")
80      colAutor.textContent=libro.autor
81      fila.appendChild(colAutor)
82      let colEditorial = document.createElement("td")
83      colEditorial.textContent=libro.editorial
84      fila.appendChild(colEditorial)
85      let colIdioma = document.createElement("td")
86      colIdioma.textContent=libro.idioma
87      fila.appendChild(colIdioma)
88      datosLibros.appendChild(fila)
89    });
90  }
```



Función que consulta libro por título al dar clic botón consultar.

```
47  /**
48   * Función que responde al evento click del botón
49   * consultar. Consulta libro por el título
50   */
51  Tabnine | Edit | Test | Explain | Document
52  btnConsultar.addEventListener("click",function(){
53    let titulo = document.getElementById("txtTitulo").value
54    document.getElementById("frmLibro").reset()
55    document.getElementById("txtTitulo").value = titulo
56
57    if(titulo!=""){
58      libroBuscado = libros.find(l=>l.titulo===titulo)
59      if(libroBuscado){
60        document.getElementById("txtAutor").value=libroBuscado.autor
61        document.getElementById("txtEditorial").value=libroBuscado.editorial
62        document.getElementById("cbIdioma").value=libroBuscado.idioma
63      }else{
64        swal.fire("Consultar Libro","No existe libro con ese Título","warning")
65      }
66    }else{
67      swal.fire("Consultar Libro","Debe ingresar título para consultar","warning")
68    }
69  })
```

Función que responde al evento clic del botón eliminar

```
19
20  /**
21   * Función que responde al evento click del
22   * botón eliminar
23   */
24  Tabnine | Edit | Test | Explain | Document
25  btnEliminar.addEventListener("click",function(){
26    if(libroBuscado!=null){
27      Swal.fire({
28        text: `¿Está usted seguro de Eliminar el libro ${libroBuscado.titulo}?`,
29        icon: "warning",
30        showCancelButton: true,
31        confirmButtonColor: "#3085d6",
32        cancelButtonColor: "#d33",
33        confirmButtonText: "Yes, delete it!"
34      }).then((result) => {
35        if (result.isConfirmed) {
36          document.getElementById("frmLibro").reset()
37          //obtener la posición del libro a eliminar
38          let posicion = libros.indexOf(libroBuscado);
39          console.log(posicion)
40          //eliminar el libro de acuerdo a su posición
41          libros.splice(posicion,1)
42          mostrarLibrosTabla()
43          Swal.fire("Deleted!", "The book has been deleted.", "success");
44          libroBuscado=null
45        }
46      });
47    }else{
48      swal.fire("Eliminar Libro","Primero hay que consultar " +
49        "libro por título para poder eliminar","warning")
50    }
51  })
```




Código del archivo app.html: Aquí debemos incorporar el archivo **app.js** como de tipo módulo. Aquí también estamos incorporando la librería de Bootstrap para los estilos y la librería de sweetalert para ventanas modales con estilo

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <!-- Latest compiled and minified CSS -->
7      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
8      <!-- Latest compiled JavaScript -->
9      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
10     <!-- sweetalert -->
11     <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
12     <script type="module" src="./js/app.js"></script>
13     <title>Biblioteca</title>
14 </head>
15 <body>
16     <div class="container">
17         <div class="w-50" style="margin: 0 auto">
18             <h3 class="fw-bold text-center">GESTIÓN LIBROS BIBLIOTECA</h3>
19             <form id="frmLibro">
20                 <div class="mt-2">
21                     <label for="txtTitulo" class="fw-bold">Titulo</label>
22                     <input type="text" name="txtTitulo" id="txtTitulo"
23                         class="form-control" placeholder="Titulo del Libro">
24                 </div>
25                 <div class="mt-2">
26                     <label for="txtAutor" class="fw-bold">Autor</label>
27                     <input type="text" name="txtAutor" id="txtAutor"
28                         class="form-control" placeholder="Autor del Libro">
29                 </div>
30                 <div class="mt-2">
31                     <label for="txtEditorial" class="fw-bold">Editorial</label>
32                     <input type="text" name="txtEditorial" id="txtEditorial"
33                         class="form-control" placeholder="Editorial del Libro">
34                 </div>
35                 <div class="mt-2">
36                     <label for="cbIdioma" class="fw-bold">Idioma</label>
37                     <select name="cbIdioma" id="cbIdioma" class="form-select">
38                         <option value="Español">Español</option>
39                         <option value="Inglés">Inglés</option>
40                         <option value="Francés">Francés</option>
41                         <option value="Portugues">Portugues</option>
42                     </select>
43                 </div>
44                 <div class="mt-2">
45                     <button id="btnAgregar" type="button" class="btn btn-dark">Agregar</button>
46                     <button id="btnEliminar" type="button" class="btn btn-danger">Eliminar</button>
47                 </div>
48             </form>
49         </div>
50     </div>
```



```
51     <div class="mt-3">
52         <table class="table table-bordered">
53             <thead class="table-dark">
54                 <tr class="text-center">
55                     <th>Título</th>
56                     <th>Autor</th>
57                     <th>Editorial</th>
58                     <th>Idioma</th>
59                 </tr>
60             </thead>
61             <tbody id="datosLibros">
62
63             </tbody>
64         </table>
65     </div>
66 </div>
67 </div>
68
69 </body>
70 </html>
```

Prueba Funcionalidades

→ 127.0.0.1:5500/modulos/biblioteca/app.html

GESTIÓN LIBROS BIBLIOTECA

Título

Autor

Editorial

Idioma

Agregar **Consultar** **Eliminar**


Título	Autor	Editorial	Idioma
Cien años de soledad	Gabriel García Márquez	Sudamericana	Español
La Vorágine	José Eustacio Rivera	Cromos	Español

→ 127.0.0.1:5500/modulos/biblioteca/app.html

GESTIÓN LIBROS BIBLIOTECA

Título

Autor



Consultar Libro

No existe libro con ese Título

OK

Título	Autor	Editorial	Idioma
n años d			Español
Vorágine	José Eustacio Rivera	Cromos	Español



Referencias:

1. Tutorial de Javascript Moderno <https://es.javascript.info/>
2. Tutorial bootstrap w3schools: <https://www.w3schools.com/bootstrap5/>
3. Sitio oficial librería Bootstrap: <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
4. Curso internet: <https://lenguajejs.com/javascript/>
5. Sito oficial librería sweetalert <https://sweetalert2.github.io/>

CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	César Marino Cuéllar Chacón	Instructor	CTPI-CAUCA	02-06-2025