

A graphic on the left side of the slide features four stacked, 3D rectangular blocks in purple, orange, yellow, and blue. The orange block is the largest and has a white arrow pointing to the right. The text 'Agencia de Aprendizaje a lo largo de la vida' is written across the blocks in white.

Agencia de
Aprendizaje
a lo largo
de la vida

FULL STACK PYTHON

Clase 18

Javascript 6

DOM y Eventos

The logo for JavaScript (JS) is centered below the title. It features the letters 'JS' in a bold, black, sans-serif font, set against a solid yellow square background.

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 17

Arrays, Storage y JSON

- Arrays.
- Funciones para operar arrays.
- Trabajar con array de objetos.
- Web Storage.
- JSON. Formato y ejemplos de uso.

Clase 18

DOM y Eventos

- Manipulación del DOM.
- Definición, alcance y su importancia..
- Eventos en JS.
- Eventos. ¿Qué son, para qué sirven y cuáles son los más comunes?
- Escuchar un evento sobre el DOM.

Clase 19

Introducción a Vue

- Introducción a Vue.js. ¿Qué es? Instalación. CDN.
- Renderizado.
- Directivas condicionales, estructurales y de atributo.
- Componentes.

¿Qué es el DOM?

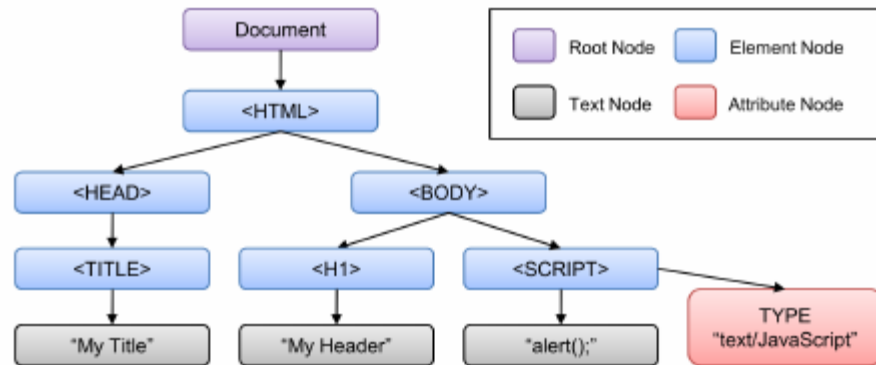
El **DOM** (**Document Object Model**) es una interfaz de programación para los documentos HTML. Proporciona una representación estructurada del documento y define de qué manera los programas pueden acceder y modificar su estructura, estilo y contenido. El DOM representa al documento como un grupo de **nodos** y **objetos** con sus propiedades y métodos. Esencialmente, conecta las páginas web a scripts o lenguajes de programación, como **JavaScript**.

Todo esto permite al desarrollador modificar esta estructura de forma dinámica, añadiendo o modificando elementos, cambiando sus atributos, etc. Estas tareas pueden automatizarse y responder a eventos como pulsar un botón, mover el ratón, hacer clic en un elemento, etc.

DOM | Manipulando la estructura

El **objeto document** contiene los atributos y métodos, de la estructura que representa al documento. JavaScript posee una API que permite su manipulación.





El DOM posee elementos (**element**) y nodos (**node**). Un elemento representa una etiqueta HTML y un nodo es una clase (no un objeto) en la que se basan muchos otros objetos del DOM. [+info](#)



DOM | Objeto document

JavaScript accede al DOM usando el **objeto document** mediante sus **atributos** y **métodos**.

Todos los elementos (**element**) HTML tendrán un tipo de dato específico. Algunos de ellos son:

Tipo de dato	Tipo específico	Etiqueta	Descripción
 HTMLElement	HTMLDivElement	<div>	Capa divisoria invisible (en bloque).
 HTMLElement	HTMLSpanElement		Capa divisoria invisible (en línea).
 HTMLElement	HTMLImageElement		Imagen.
 HTMLElement	HTMLAudioElement	<audio>	Contenedor de audio.

DOM | Modificar elementos | Método tradicional

document permite identificar elementos por sus atributos, por ejemplo, su **id** o **class**. La forma tradicional de hacerlo es mediante el método **getElementById()** o algunos de los que vemos en la tabla. Devuelven o bien un elemento, un arreglo con todos los que son de esa clase, o **null** en caso de que no exista el elemento buscado.

Métodos de búsqueda	Descripción
ELEMENT <code>.getElementById(id)</code>	Busca el elemento HTML con el id id . Si no, devuelve NULL .
ARRAY <code>.getElementsByClassName(class)</code>	Busca elementos con la clase class . Si no, devuelve <code>[]</code> .
ARRAY <code>.getElementsByName(name)</code>	Busca elementos con atributo name name . Si no, devuelve <code>[]</code> .
ARRAY <code>.getElementsByTagName(tag)</code>	Busca elementos tag . Si no encuentra ninguno, devuelve <code>[]</code> .

DOM | Modificar elementos | Método tradicional

Los métodos que comienzan con **get** devuelven un valor. Los que comienzan con **set** modifican o establecen un valor.

El argumento del método **getElementById()** es el id del elemento, y retorna el objeto referenciado. El método **innerHTML()** escribe código HTML en un elemento. El argumento es una cadena de texto, y si usamos comillas invertidas ` para definir el string, se respetan los saltos de línea.

getElementsByClassName retorna un array de objetos, ya que pueden existir múltiples elementos de una clase CSS.

DOM | Modificar elementos | Método tradicional

El código del ejemplo guarda en el array **x** todos los elementos (**objetos**) de la página HTML que sean de la clase “**ejemplo**”.

Luego, al elemento **x[0]** se le modifica su contenido utilizando el método **innerHTMLHTML**. El segundo **<div>** permanece inalterado, porque es el elemento **[1]** del arreglo.

```
<!DOCTYPE html>
<html>
<body>
  <h1>DOM</h1>
  <p>Solo cambia el primer elemento:</p>

  <div class="ejemplo">Elemento 1</div>
  <div class="ejemplo">Elemento 2</div>
  <script>
    var x =
document.getElementsByClassName("ejemplo")
    x[0].innerHTML = "Hola Codo a Codo!"
  </script>
</body>
</html>
```

DOM | Modificar elementos | Método tradicional

El código del ejemplo espera a que se dispare el evento “**onclick**” del botón, que invoca a la función “**cambiarTexto**”.

La función guarda en el array **x** todos los elementos (**objetos**) de la página HTML que sean de la clase “**ejemplo**”, los recorre usando un bucle **for**, y usando su método `innerHTML` les cambia su contenido.

```
<!DOCTYPE html>
<html>
<body>
<h1>DOM</h1>
<p>Cambiar todos los elementos:</p>
<div class="ejemplo">Elemento 1</div>
<div class="ejemplo">Elemento 2</div>
<div class="ejemplo">Elemento 3</div>
<script>
  function cambiarTexto() {
    var x=document.getElementsByClassName("ejemplo")
    for (i = 0; i < x.length; i++) {
      x[i].innerHTML="Codo a Codo! " +(i+1)
    }
  }
</script>
<button onclick="cambiarTexto()">
  Cambiar todos los párrafos</button>
</body>
</html>
```

DOM | Modificar elementos | Método tradicional

De forma similar funcionan los métodos **getElementsByName(name)** y **getElementsByTagName(tag)**, que se encargan de buscar elementos HTML por su atributo **name** o por su **tag** (etiqueta) de elemento HTML, respectivamente. En el siguiente ejemplo las constantes **nicknames** y **divs** contendrán una lista de objetos:

```
// Obtiene todos los elementos con atributo name="nickname"
const nicknames = document.getElementsByName("nickname")

// Obtiene todos los elementos <div> de la página
const divs = document.getElementsByTagName("div")
```

DOM | Modificar elementos | Métodos modernos

En los últimos años **JS** ha añadido dos nuevos métodos de búsqueda de elementos que son simples de usar, sobre todo si conocemos los selectores CSS. Son los métodos **.querySelector()** y **.querySelectorAll()**:

Método de búsqueda	Descripción
ELEMENT <code>.querySelector(sel)</code>	Busca el primer elemento que coincide con el selector CSS <code>sel</code> . Si no, NULL .
ARRAY <code>.querySelectorAll(sel)</code>	Busca todos los elementos que coinciden con el selector CSS <code>sel</code> . Si no, <code>[]</code> .

Con estos métodos podemos reemplazar los “*métodos tradicionales*” e incluso realizar nuevas intervenciones en el DOM gracias a su flexibilidad.

DOM | Métodos modernos | querySelector()

.querySelector(selector) devuelve **el primer elemento** que encaja con el selector CSS suministrado en *selector*. Al igual que **.getElementById()**, en caso de no coincidir con ninguno devuelve null.

```
const page = document.querySelector("#page") // <div id="page"></div>  
const info = document.querySelector(".main .info") // <div class="info"></div>
```

En la primera línea incluimos en el argumento un **#** porque se trata de un **id**. En la segunda estamos recuperando el primer elemento con clase **info** que esté dentro de un elemento de la clase **main**. Eso podría realizarse con los métodos tradicionales, pero sería necesario un código más extenso y complejo. **querySelector()** simplifica el proceso.

DOM | Métodos modernos | querySelectorAll()

El método **.querySelectorAll(selector)** es similar a **.querySelector()**, pero en caso de que haya más de un elemento que se ajuste a lo indicado por **selector**, devuelve un array con todos los elementos que coinciden con él:

```
// Obtiene todos los elementos con clase "info"
const infos = document.querySelectorAll(".info")

// Obtiene todos los elementos con atributo name="nickname"
const nicknames = document.querySelectorAll('[name="nickname"]')

// Obtiene todos los elementos <div> de la página HTML
const divs = document.querySelectorAll("div")
```

.querySelectorAll() siempre nos devolverá un array con uno o más objetos, o vacío si no encuentra elementos de ese tipo en el documento.

DOM | Crear elementos HTML

Existen métodos para crear diferentes **elementos** HTML o **nodos**, que nos permiten agregar al documento estructuras dinámicas, mediante bucles o estructuras definidas:

Métodos	Descripción
Element .createElement(tag, options)	Crea y devuelve el elemento HTML definido por tag. Ejemplo
Node .createComment(text)	Crea y devuelve un nodo de comentarios HTML <!-- text -->.
Node .createTextNode(text)	Crea y devuelve un nodo HTML con el texto text. Ejemplo
Node .cloneNode(deep)	Clona el nodo HTML y devuelve una copia. deep es false por defecto. Ejemplo
Boolean .isConnected	Indica si el nodo HTML está insertado en el documento. Ejemplo

DOM | .createElement() y .appendChild()

.createElement() podemos crear un elemento HTML en memoria. Este elemento puede insertarse en el documento HTML con **.appendChild()**, en una posición determinada. El ejemplo crea un botón y lo coloca en el body:

```
<!DOCTYPE html>
<html>
<body>
<p>Creamos un elemento 'button':</p>
<script>
  const btn = document.createElement("button") //Creamos el boton y lo guardamos en btn
  btn.innerHTML = "Soy un botón!" // Le ponemos el texto
  document.body.appendChild(btn) // Lo agregamos al <body>
</script>
</body>
</html>
```

DOM | .createTextNode()

.createTextNode() es un método que crea **nodos de texto**. Esos elementos luego pueden ser asignados a un objeto. En el ejemplo se crea un nodo de texto y se lo asigna a un **<h1>**, que luego se coloca en el **<body>**:

```
<!DOCTYPE html>
<html>
<body>
<p>Creamos un h1 con texto:</p>
<script>
  const h1 = document.createElement("h1")           //Creamos el <h1>
  const textNode = document.createTextNode("¡Hola!") //Creamos el texto
  h1.appendChild(textNode)                          //Colocamos el texto como hijo del <h1>
  document.body.appendChild(h1)                     //Y ponemos el <h1> dentro del <body>
</script>
</body>
</html>
```

DOM | .cloneNode()

.cloneNode() toma un nodo, y devuelve una copia: [+info](#)

```
<!DOCTYPE html>
<html><body>
<button onclick="clonar()">Copiar</button>
<p>Presionando el botón se copia un elemento de una lista a otra.</p>
<ul id="lista1"><li>Café</li><li>Té</li></ul>
<ul id="lista2"><li>Agua</li><li>Leche</li></ul>
<p>Cambiando <b>deep</b> a false sólo se clonan elementos vacíos.</p>
<script>
function clonar() {
  const nodo = document.getElementById("lista2").lastChild //Leemos el nodo a clonar, lo
  const clon = nodo.cloneNode(true);                      //clonamos y guardamos en "clon"
  document.getElementById("lista1").appendChild(clon)      //Y lo agregamos en la lista2
}
</script>
</body></html>
```

DOM | Modificar atributos de un elemento

Los objetos que obtenemos a partir de métodos como **.createElement()** o **.getElementById()**, entre otros, poseen atributos que pueden ser modificados:

```
<!DOCTYPE html>
<html>
<body>
  <p id="p1">Este texto se va a borrar.</p>
  <script>
    const p = document.getElementById("p1")
    p.innerHTML = "Codo a Codo" // <p id="p1">Codo a Codo</p>
    p.className = "dato"       // <p id="p1" class="data">Codo a Codo</p>
    p.style.color = "red"      // <p id="p1" class="data" style="color:red">Codo a Codo</p>
  </script>
</body>
</html>
```

DOM | Reemplazar contenido de un elemento

.textContent e **.innerHTML** permiten recuperar o modificar el contenido de texto de un elemento, pero no son equivalentes:

Propiedades	Descripción
.textContent	Devuelve o asigna el texto del elemento. No atiende la sintaxis HTML. Ejemplo
.innerHTML	Devuelve o asigna el contenido HTML del elemento. Ejemplo

```
const div1 = document.querySelector("div") // <div></div>
div1.textContent = "Hola a todos"          // <div>Hola a todos</div>
div1.textContent                           // "Hola a todos"
const div2 = document.querySelector(".info") // <div class="info"></div>
div2.innerHTML = "<strong>Importante</strong>" // Interpreta el HTML
div2.innerHTML                             // "<strong>Importante</strong>"
div2.textContent                           // "Importante"
```

DOM | Insertar una imagen

Las propiedades y métodos vistos permiten, por ejemplo, **insertar** una imagen en el documento HTML:

```
const img = document.createElement("img")
img.src = "https://lenguajejs.com/assets/logo.svg"
img.alt = "Logo Javascript"
document.body.appendChild(img)
```

.appendChild() es el método que permite agregar un elemento al DOM. En este caso, se agrega en el **<body>**. [+info](#)

También es posible eliminar elementos, para ello debemos utilizar el método **.remove()** [+info](#)

DOM | API nativa de Javascript

Entre las herramientas que provee la API de JS se encuentran:

Capítulo del DOM	Descripción
Buscar etiquetas	Métodos como <code>.getElementById()</code> , <code>.querySelector()</code> o <code>.querySelectorAll()</code> , entre otras. +info
Crear etiquetas	Métodos para crear elementos en la página y trabajar con ellos de forma dinámica. +info
Insertar etiquetas	Métodos para añadir elementos al DOM, como <code>.appendChild()</code> , <code>.insertAdjacentHTML()</code> , entre otros. +info
Gestión de clases CSS	<code>.classList</code> permite manipular clases CSS desde JS, para añadir, modificar o, eliminar clases CSS de un elemento. +info
Navegar entre elementos	Métodos y propiedades para «navegar» a través de la jerarquía del DOM, por la estructura del documento y la posición de los elementos en la misma. +info

DOM | Eventos en JS

Los **eventos** son acciones que realiza el usuario a las que podemos “atender” desde JavaScript e indicar qué función o bloque de código se debe ejecutar como respuesta. Estos eventos permiten interactuar con el usuario, por ejemplo cuando hace clic en un botón. Existen tres formas de definir eventos en nuestro código:

Estrategia	Ejemplo
A través de un atributo HTML, asociando al mismo una función.	<code><tag onclick="..."></code>
A través de una propiedad de JavaScript, a la que asociamos la función.	<code>tag.onclick = ...</code>
A través del método addEventListener() , que permite crear un “atendedor” de eventos.	<code>tag.addEventListener("click", ...)</code>

DOM | Eventos en JS desde atributos HTML

Probablemente sea la forma más sencilla de atender un evento. Definimos un evento a través de un **atributo** HTML, por ejemplo **onClick**. En el ejemplo, al hacer click sobre el botón se ejecuta la función flecha **enviarMensaje**. Esta función genera un mensaje *"Hola!"* mediante la función de javascript **alert**.

```
<button onClick="enviarMensaje()">Haz clic!</button>
<script>
  const enviarMensaje = () => alert("Hola!")
</script>
```

DOM | Eventos en JS desde propiedades JS

Otra forma de utilizar eventos es utilizar las **propiedades** de Javascript. Por cada evento, existe una propiedad disponible en el elemento en cuestión:

```
<button>Haz clic!</button>
<script>
  const button = document.querySelector("button")
  button.onclick = () => alert("Hola!")
</script>
```

DOM | Eventos con .addEventListener()

.addEventListener() es la forma más elaborada de utilizar eventos: [+info](#)

```
<!DOCTYPE html>
<html><head>
  <script>
    function modifyText() { // Función que modifica el contenido de #t2
      var t2 = document.getElementById("t2")
      t2.firstChild.nodeValue = "Tocado!"
    }
    function load() { // Función que establece el EventListener()
      var el = document.getElementById("t2")
      el.addEventListener("click", modifyText, false)
    }
    // Al cargar el documento, agregamos el EventListener()
    document.addEventListener("DOMContentLoaded", load, false)
  </script>
</head>
<body>
  <p id="t2">¡Haz click aquí!</p>
</body></html>
```

Material extra

Artículos de interés

Material de lectura:

- Eventos en JS: <https://developer.mozilla.org/es/docs/Web/Events>
- Lista de los tipos de eventos más habituales en Javascript:
<https://desarrolloweb.com/articulos/1236.php>
- addEventListener: <https://developer.mozilla.org/es/docs/Web/API/EventTarget/addEventListener>

Videos:

- 10 Fundamentos modernos que debes conocer en JS:
<https://www.youtube.com/watch?v=Z4TuS0HEJP8&list=PLPI81lqbj-4I2ZOzryjPKxfhK3BzTlaJ7>
- DOM fundamentos: <https://www.youtube.com/watch?v=bYdUoqi6JXE>
- DOM delegación de eventos: <https://www.youtube.com/watch?v=OspjzGQa86g&t=8017s>
- Formularios: <https://www.youtube.com/watch?v=L5Yin6K4ARs>
- Callback, Promesas y Async Await: <https://www.youtube.com/watch?v=V0tiKDHk7t0>
- Fetch: <https://www.youtube.com/watch?v=cBuTxGdGjM8>

Actividades prácticas:

- Agregar la validación de los campos obligatorios del formulario creado en el TPO. Opcional: el formulario podrá enviar un email utilizando algún servicio externo destinado para ello.

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- Realizar los Ejercicios obligatorios.

Todo en el Aula Virtual.

Muchas gracias por tu atención.

Nos vemos pronto