

***RECUERDA PONER A GRABAR LA
CLASE***





¿DUDAS DEL ON-BOARDING?

MIRALO AQUI



Clase 08. JAVASCRIPT

MODELO DE OBJETOS DEL DOCUMENTO (DOM)

GLOSARIO:

Clase 7

Abstracción: Resumen de un grupo complejo de instrucciones bajo una palabra clave (función) que sugiere cuál es el problema a resolver por la misma.

Función de orden superior: Es aquella que bien retorna una función, o recibe una función por parámetro. También es conocida como *función de alto orden* o *higher-order functions*.

Objeto Math: Contenedor de herramientas y serie de métodos propio de Javascript para realizar funciones matemáticas complejas.

Clase Date: Clase propia de Javascript que nos permite representar y manipular fechas.



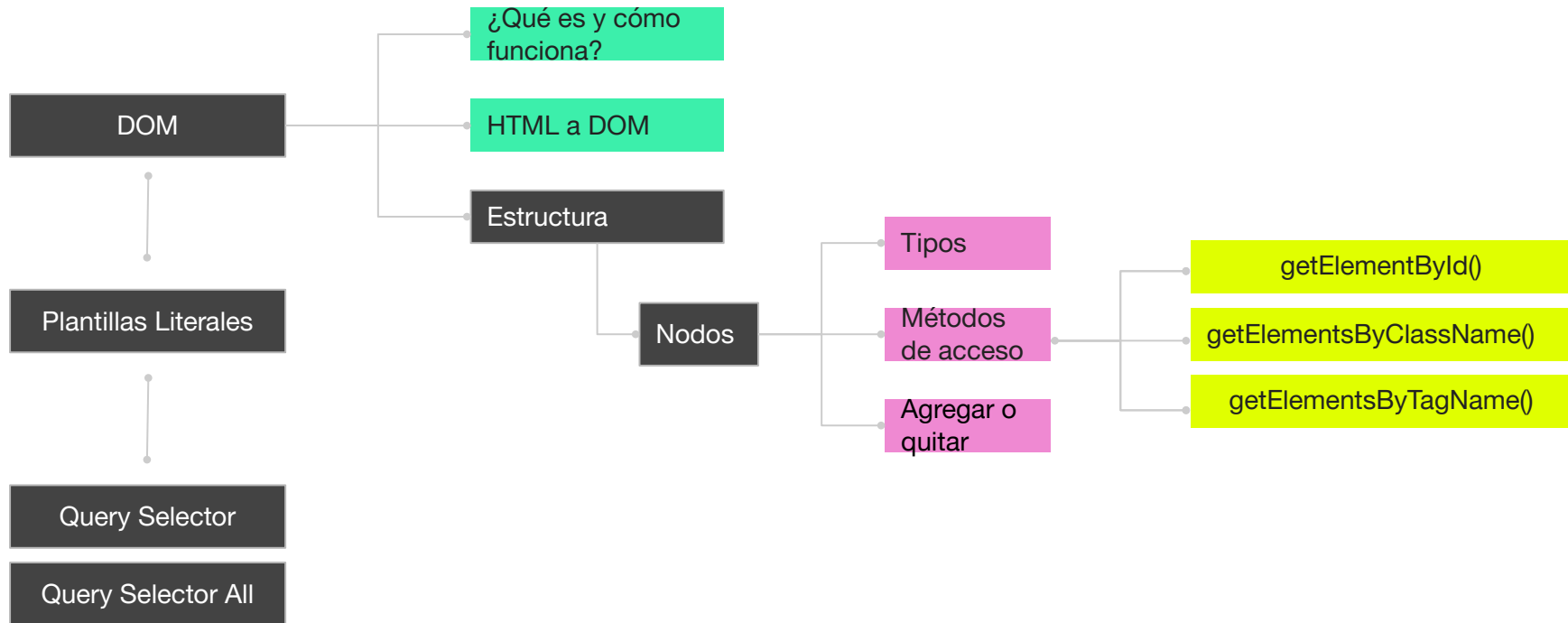
OBJETIVOS DE LA CLASE

- Comprender el **DOM**, su alcance y su importancia para operar sobre elementos HTML.
- Identificar la **estructura** del DOM y los **tipos de nodos**.
- Conocer los **métodos** de acceso y modificación de nodos.
- Aprender cómo **agregar** y **quitar** nodos.
- Entender la importancia de la **plantilla de texto** en ES6 de JavaScript.

MAPA DE CONCEPTOS

MAPA DE CONCEPTOS CLASE 8

¡Para
recordar!



MÓDULOS DE TRABAJO

MÓDULO 3 FRONTEND

CLASE 8 - DOM

CLASE 9 - EVENTOS

- Desafío entregable

CLASE 10 - STORAGE & JSON

CLASE 11 - Workshop I

- 2da pre-entrega

MÓDULO 4 OPTIMIZACIÓN DE PROYECTO

CLASE 12 - OPERADORES AVANZADOS

CLASE 12 - LIBRERÍAS

- Desafío entregable

MÓDULO 5 ASINCRONÍA Y PETICIONES

CLASE 14 - AJAX & FETCH

CLASE 15 - PROMISES & ASYNC

- Desafío entregable

CLASE 16 - FRAMEWORKS + Node JS

- Entrega TP final



HERRAMIENTAS DE LA CLASE

Les compartimos algunos recursos para acompañar la clase

- Guión de clase N° 8 [aquí](#).
- Quizz de clase N° 8 [aquí](#)
- Booklet de Javascript [aquí](#)
- FAQs de Javascript [aquí](#)

¡VAMOS A LA CLASE!



MODELO DE OBJETOS DEL DOCUMENTO (DOM)

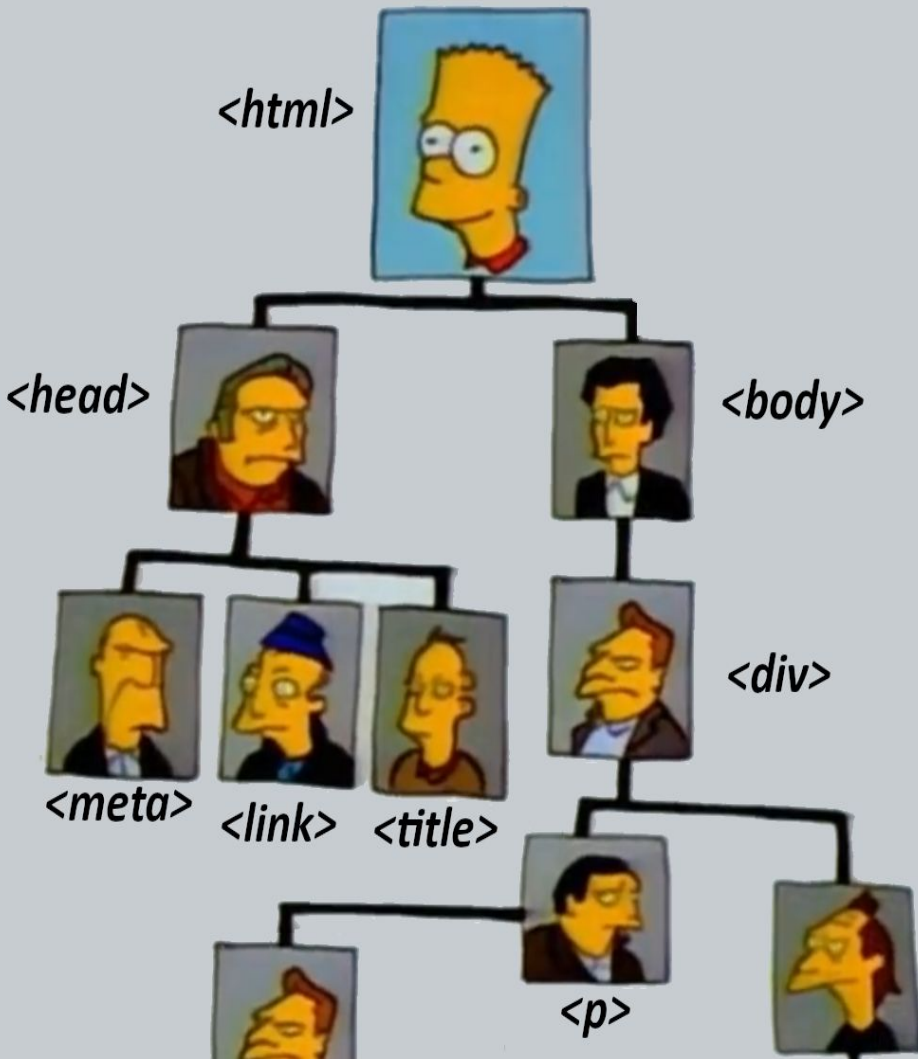
***¿QUÉ ES EL MODELO DE OBJETOS DEL
DOCUMENTO (DOM) Y CÓMO FUNCIONA?***

DOM ***(Document Object Model)***

El Modelo de Objetos del Documento (DOM) es una estructura de objetos generada por el navegador, la cual representa la página HTML actual.

Con JavaScript la empleamos **para acceder y modificar de forma dinámica elementos de la interfaz.**

Es decir que, por ejemplo, desde JavaScript podemos modificar el texto contenido de una etiqueta `<h1>`.



¿CÓMO FUNCIONA?

La estructura de un documento HTML son las **etiquetas**.

En el Modelo de Objetos del Documento (DOM), cada etiqueta HTML es un objeto, al que podemos llamar **nodo**.

Las etiquetas anidadas son llamadas “nodos hijos” de la etiqueta “nodo padre” que las contiene.

¿CÓMO FUNCIONA?

Todos estos objetos son accesibles empleando JavaScript mediante el objeto global **document**.

Por ejemplo, **document.body** es el nodo que representa la etiqueta `<body>`.

ESTRUCTURA DEL DOM Y NODOS



ESTRUCTURA DOM

Cada etiqueta HTML se transforma en un nodo de tipo "**Elemento**". La conversión se realiza de forma jerárquica.

De esta forma, del **nodo raíz** solamente pueden derivar los nodos **HEAD** y **BODY**.

Cada etiqueta HTML se transforma en un nodo que deriva del correspondiente a su "**etiqueta padre**".

La transformación de las etiquetas HTML habituales genera **dos nodos**

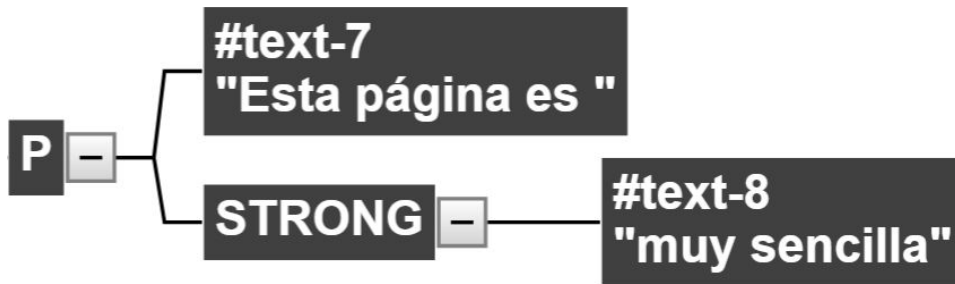
Nodo elemento:
correspondiente a la propia etiqueta HTML.

Nodo texto: contiene el texto encerrado por esa etiqueta HTML.

EJEMPLO

```
<p>Esta página es <strong>muy sencilla</strong></p>
```

La etiqueta <p> se transforma en los siguientes nodos del DOM:



EDITAR EL DOM DESDE EL NAVEGADOR

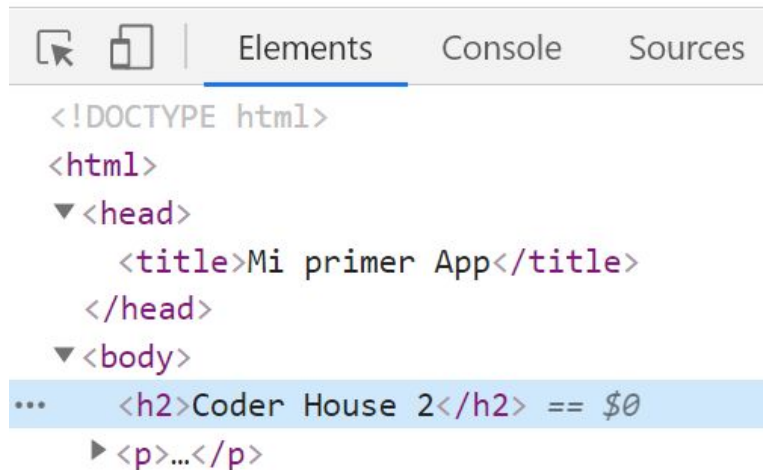
Los **navegadores modernos** brindan medios para editar el DOM de cualquier página en tiempo real.

Ejemplo: en Chrome podemos hacerlo mediante la Herramienta para desarrolladores en la pestaña **“Elements”**.

Referencia: [Editar el DOM](#) (Chrome)

Coder House 2

Esta página es **muy sencilla**



```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primer App</title>
  </head>
  <body>
    ... <h2>Coder House 2</h2> == $0
    > <p>...</p>
```



EDITAR EL DOM DESDE EL NAVEGADOR

Si bien la estructura DOM está simplificada, es un medio muy útil para verificar y probar actualizaciones en la estructura.

Referencia: [Editar el DOM](#) (Chrome)

ACCESO AL DOM

ACCEDER A LOS NODOS

Existen distintos métodos para acceder a los elementos del DOM empleando en la clase [Document](#).

Los más comunes son:

`getElementById()`

`getElementsByClassName()`

`getElementsByTagName()`

GETELEMENTBYID()

El método **getElementById()** sirve para acceder a un elemento de la estructura HTML, utilizando su atributo ID como identificación.

```
//CODIGO HTML DE REFERENCIA
<div id = "app">
    <p id = "parrafo1" >Hola Mundo</p>
</div>

//CODIGO JS
let div      = document.getElementById("app");
let parrafo = document.getElementById("parrafo1");
console.log(div.innerHTML);
console.log(parrafo.innerHTML);
```

GETELEMENTSBYCLASSNAME()

El método `getElementsByClassName()` sirve para acceder a un conjunto de elementos de la estructura HTML, utilizando su atributo `class` como identificación. Se retornará un Array de elementos con todas las coincidencias:

```
//CODIGO HTML DE REFERENCIA
<ul>
  <li class="paises">AR</li>
  <li class="paises">CL</li>
  <li class="paises">UY</li>
</ul>

//CODIGO JS
let paises = document.getElementsByClassName("paises");
console.log(paises[0].innerHTML);
console.log(paises[1].innerHTML);
console.log(paises[2].innerHTML);
```


GETELEMENTSBYTAGNAME()

El método **getElementsByTagName()** sirve para acceder a un conjunto de elementos de la estructura HTML, **utilizando su nombre de etiqueta como identificación**. Esta opción es la menos específica de todas, ya que es muy probable que las etiquetas se repitan en el código HTML.

```
//CODIGO HTML DE REFERENCIA
<div>
  <div>CONTENEDOR 2</div>
  <div>CONTENEDOR 3</div>
</div>

//CODIGO JS
let contenedores = document.getElementsByTagName("div");
console.log(contenedores[0].innerHTML);
console.log(contenedores[1].innerHTML);
console.log(contenedores[2].innerHTML);
```

EJEMPLO APLICADO: RECORRE HTMLCollection CON FOR...OF

```
let paises          = document.getElementsByClassName("paises");  
let contenedores = document.getElementsByTagName("div");  
  
for (const pais of paises) {  
    console.log(pais.innerHTML);  
}  
  
for (const div of contenedores) {  
    console.log(div.innerHTML);  
}
```

MODIFICAR NODOS

INNER TEXT

La propiedad **innerText** de un nodo nos permite modificar su nodo de texto. Es decir, acceder y/o modificar el contenido textual de algún elemento del DOM.

```
//CODIGO HTML DE REFERENCIA
```

```
<h1 id="titulo">Hola Mundo!</h1>
```

```
//CODIGO
```

JS

```
let titulo = document.getElementById("titulo")
```

```
console.log( titulo.innerText ) // "Hola Mundo!"
```

```
// cambio el contenido del elemento
```

```
titulo.innerText = "Hola Coder!"
```

```
console.log( titulo.innerText ) // "Hola Coder!"
```

INNER HTML

innerHTML permite definir el **código html interno** del elemento seleccionado.

El navegador lo interpreta como código HTML y no como contenido de texto, permitiendo desde un string crear una nueva estructura de etiquetas y contenido.

INNER HTML

Al pasar un string con formato de etiquetas html y contenido a través de la propiedad innerHTML, el navegador **genera nuevos nodos con su contenido** dentro del elemento seleccionado.

INNER HTML

```
//CODIGO HTML DE REFERENCIA
```

```
<div id="contenedor"></div>
```

```
//CODIGO
```

JS

```
let container = document.getElementById("contenedor")
```

```
// cambio el código HTML interno
```

```
container.innerHTML = "<h2>Hola mundo!</h2><p>Lorem ipsum</p>"
```

```
//Resultado en el DOM
```

```
<div id="contenedor">
```

```
    <h2>Hola mundo!</h2>
```

```
    <p>Lorem ipsum</p>
```

```
</div>
```

CLASS NAME

A través de la propiedad

className de algún

nodo seleccionado

podemos acceder al

atributo **class** del mismo y

definir cuáles van a ser sus

clases:

```
//CODIGO HTML DE REFERENCIA
<div id="contenedor" ></div>

//CODIGO JS
let container = document.getElementById("contenedor")
// cambio el código HTML interno
container.innerHTML = "<h2>Hola mundo!</h2>"
// cambio el atributo class
container.className = "container"
//Resultado en el DOM
<div id="contenedor" class="container">
  <h2>Hola mundo!</h2>
</div>
```




BREAK

¡5/10 MINUTOS Y VOLVEMOS!

¡VOLVEMOS!



AGREGAR O QUITAR NODOS

CREACIÓN DE ELEMENTOS

Para crear elementos se utiliza la función `document.createElement()`, y se debe indicar el nombre de etiqueta HTML que representará ese elemento.

Luego debe agregarse como hijo el nodo creado con `append()`, al body o a otro nodo del documento actual.

```
// Crear nodo de tipo Elemento, etiqueta p
let parrafo = document.createElement("p");
// Insertar HTML interno
parrafo.innerHTML = "<h2>¡Hola Coder!</h2>";
// Añadir el nodo Element como hijo de body
document.body.append(parrafo);
```

ELIMINAR ELEMENTOS

Se pueden eliminar nodos existentes y nuevos. El método `remove()` permite eliminar un nodo seleccionado del DOM:

```
let parrafo = document.getElementById("parrafo1");  
//Eliminando el propio elemento  
parrafo.remove();  
  
let paises = document.getElementsByClassName("paises");  
//Eliminando el primer elemento de clase paises  
paises[0].remove()
```

OBTENER DATOS DE INPUTS

Para obtener o modificar datos de un formulario HTML desde JS, podemos hacerlo mediante el DOM. Accediendo a la **propiedad value** de cada input seleccionado:

```
//CODIGO HTML DE REFERENCIA
```

```
<input id = "nombre" type="text">
```

```
<input id = "edad" type="number">
```

```
//CODIGO JS
```

```
document.getElementById("nombre").value = "HOMERO";
```

```
document.getElementById("edad").value = 39;
```

EJEMPLO APLICADO: CREANDO OPCIONES DESDE UN ARRAY

```
//Obtenemos el nodo donde vamos a agregar los nuevos elementos
let padre = document.getElementById("personas");
//Array con la información a agregar
let personas = ["HOMERO", "MARGE", "BART", "LISA", "MAGGIE"];
//Iteramos el array con for...of
for (const persona of personas) {
    //Creamos un nodo <li> y agregamos al padre en cada ciclo
    let li = document.createElement("li");
    li.innerHTML = persona
    padre.appendChild(li);
}
```

PLANTILLAS DE TEXTO

PLANTILLAS LITERALES

En versiones anteriores a ES6, solía emplearse la concatenación para incluir valores de las variables en una cadena de caracteres (string). Esta forma puede ser poco legible ante un gran número de referencias. **En JS ES6 que solventa esta situación son los *template strings*.**

```
let producto = { id: 1, nombre: "Arroz", precio: 125 };  
let concatenado = "ID : " + producto.id + " - Producto: " + producto.nombre + "$  
"+producto.precio;  
let plantilla = `ID: ${producto.id} - Producto ${producto.nombre} $  
${producto.precio}`;  
//El valor es idéntico pero la construcción de la plantilla es más sencilla  
console.log(concatenado);  
console.log(plantilla);
```

PLANTILLAS LITERALES E innerHTML

Las plantillas son un medio para incluir variables en la estructura HTML de nodos nuevos o existentes , modificando el innerHTML.

```
let producto    = { id: 1,  nombre: "Arroz", precio: 125 };
let contenedor = document.createElement("div");
//Definimos el innerHTML del elemento con una plantilla de texto
contenedor.innerHTML = `

### ID: ${producto.id}</h3> <p> Producto: ${producto.nombre}</p> <b> $ ${producto.precio}</b>`; //Agregamos el contenedor creado al body document.body.appendChild(contenedor);


```

EJEMPLO APLICADO: CREANDO ELEMENTOS DESDE OBJETOS

```
const productos = [{ id: 1, nombre: "Arroz", precio: 125 },
                    { id: 2, nombre: "Fideo", precio: 70 },
                    { id: 3, nombre: "Pan" , precio: 50},
                    { id: 4, nombre: "Flan" , precio: 100}];

for (const producto of productos) {
  let contenedor = document.createElement("div");
  //Definimos el innerHTML del elemento con una plantilla de texto
  contenedor.innerHTML = `<h3> ID: ${producto.id}</h3>
                          <p> Producto: ${producto.nombre}</p>
                          <b> $ ${producto.precio}</b>`;
  document.body.appendChild(contenedor);
}
```

QUERY SELECTOR

```
<div id="contenedor">  
  <p class="texto"></p>  
</div>
```

```
// puedo seleccionar la etiqueta <p> siguiendo la  
sintaxis de CSS para selectores:
```

```
let parrafo = document.querySelector("#contenedor p")
```

```
// seleccionar sólo el contenedor por id con #
```

```
let contenedor =  
document.querySelector("#contenedor")
```

```
// o por clase:
```

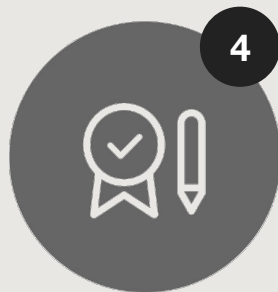
```
parrafo = document.querySelector(".texto")
```

El método `querySelector()` nos permite seleccionar nodos con la misma sintaxis que utilizamos en los selectores de CSS.



QUERY SELECTOR ALL

Query Selector me retorna *el primer elemento* que coincida con el parámetro de búsqueda, o sea un sólo elemento. Si quiero obtener una colección de elementos puedo utilizar el método `querySelectorAll()` siguiendo el mismo comportamiento.



INTERACTUAR CON HTML

Ahora podés sumar a tu proyecto lo trabajado sobre DOM, para interactuar entre los elementos HTML y JS.

INTERACTUAR CON HTML

Formato: Página HTML y código fuente en JavaScript. Debe identificar el apellido del estudiante en el nombre de archivo comprimido por “claseApellido”.

Sugerencia: Generalmente, identificamos a un único elemento del DOM con el atributo id y a un conjunto asociado por class.

Desafío
Complementario



>> Consigna: Traslada al proyecto integrador el concepto de objetos, visto en la clase de hoy. En función del tipo de simulador que hayas elegido, deberás:

- Crear elementos manipulando el DOM a partir de la información de tus objetos.
- Modificar etiquetas existentes en función del resultado de operaciones.

>>Aspectos a incluir en el entregable:

Archivo HTML y Archivo JS, referenciado en el HTML por etiqueta `<script src="js/miarchivo.js"></script>`, que incluya la definición de un algoritmo en JavaScript que opere sobre el DOM, modificando, agregando o eliminado elementos.

>>Ejemplo:

Podemos crear elementos HTML en función del listado de nuestros objetos identificados en la clase 6.

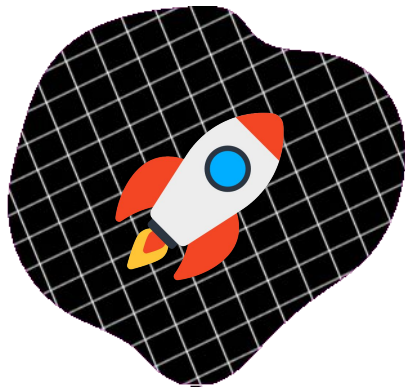
Establecer un mensaje de bienvenida aleatorio usando un array de mensajes.

Capturar una o más entradas por `prompt()` y mostrarlas en el HTML, modificando el DOM

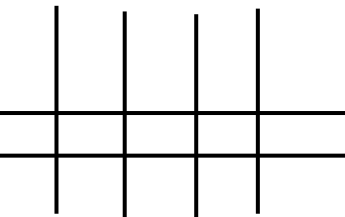
PREPARÁNDONOS PARA LA PRE-ENTREGA N° 2

*En la clase 11 tendremos el cierre del tercer módulo y se entregarán las consignas de la **segunda pre-entrega** del curso.
La misma incluirá temas vistos en las clases 8, 9, 10.*





- La pre-entrega se compone de temas vistos hasta el momento, más otros que verán durante el **módulo completo** 💪.
- Te recomendamos ir avanzando con los "Hands On" y "Desafíos Complementarios" ✨
- Recuerden que recién la consigna del desafío se entrega ¡en la clase N° 11! 🙌 **Y tendrán hasta 7 días para resolver el desafío y subirlo.**

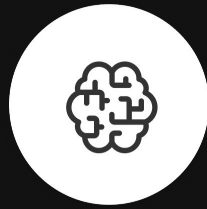


PRE-ENTREGA N° 2

Compuesta por...



- **Modificación del DOM** - Desafío complementario 
- **Detección de eventos de usuario.** Desafío entregable 
- **Implementación con uso de JSON y Storage.** 



¡PARA PENSAR!

¿Te gustaría comprobar tus conocimientos de la clase?

Te compartimos a través del chat de zoom
el enlace a un breve quiz de tarea.

Para el profesor:

- *Acceder a la carpeta "Quizzes" de la camada*
 - *Ingresar al formulario de la clase*
 - *Pulsar el botón "Invitar"*
 - *Copiar el enlace*
- *Compartir el enlace a los alumnos a través del chat*

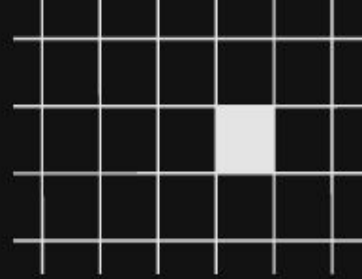
¿PREGUNTAS?



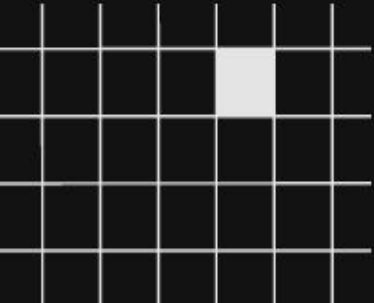
¡DESCUENTO EXCLUSIVO!



[Quiero saber más](#)



¡Completa tu carrera y potencia tu desarrollo profesional!
*Ingresando el cupón **CONTINUATUCARRERA** tendrás un*
descuento para inscribirte en el próximo nivel. Puedes
acceder directamente desde la plataforma, entrando en la
sección ["Cursos y Carreras"](#).





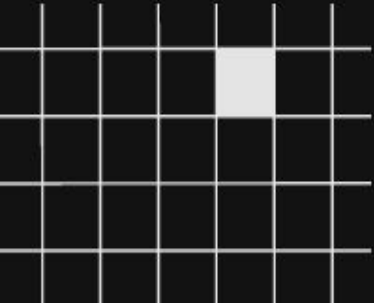
RECURSOS:

- Ejemplos interactivos: DOM |
Árbol del Modelo de Objetos del Documento (DOM)
Recorriendo el DOM
Propiedades de los nodos
- Documentación |
Documentación DOM



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- DOM: definición y uso.
 - Árbol de nodos.
 - Acceder a los elementos, crear y eliminar nodos.
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE