Esta clase va a ser

• grabada

Certificados oficialmente por

PedidosYa

CODERHOUSE

Clase 09. JAVASCRIPT

Modelo de objetos del documento (DOM)

Certificados oficialmente por

CODERHOUSE

CLASE N°8

Glosario

Operar: 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.

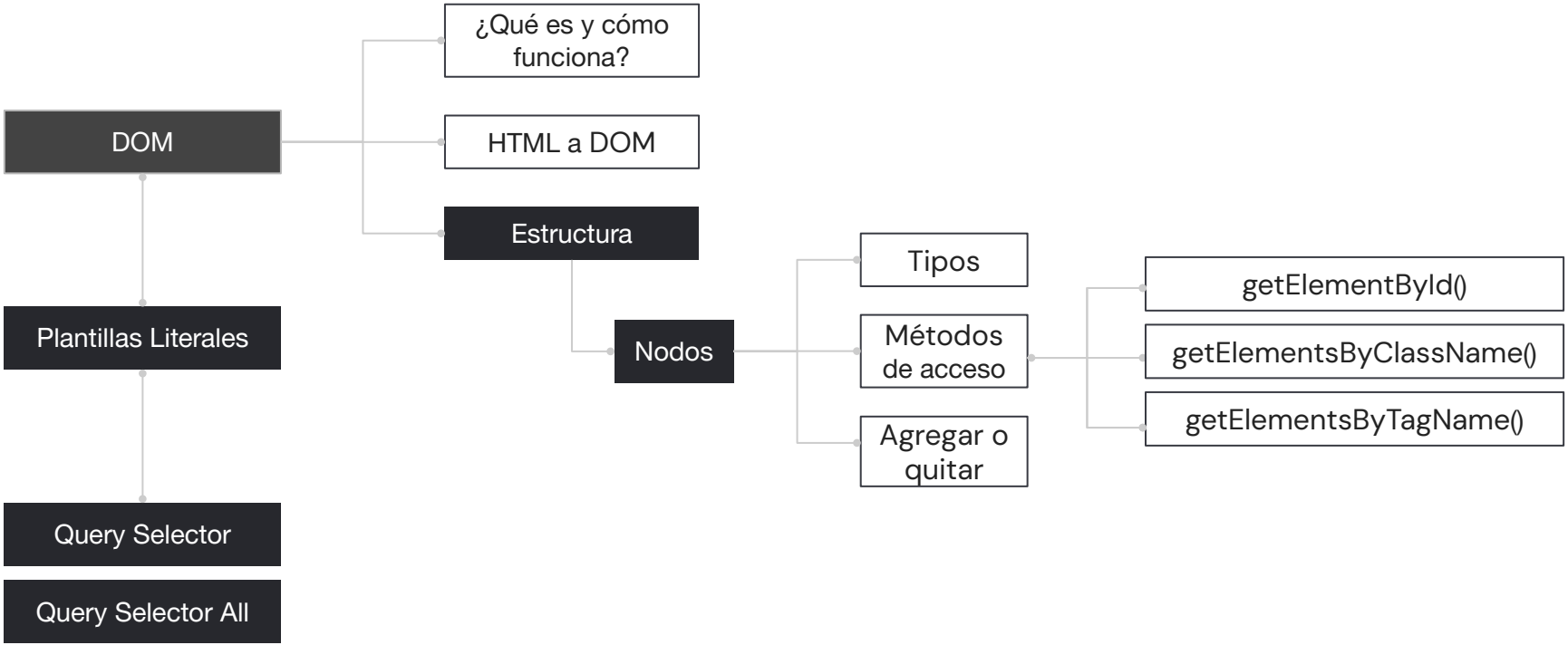
Objeto Math: Contenedor de herramientas y serie de métodos propio de Javascript para realizar funciones matemáticas complejas.

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.

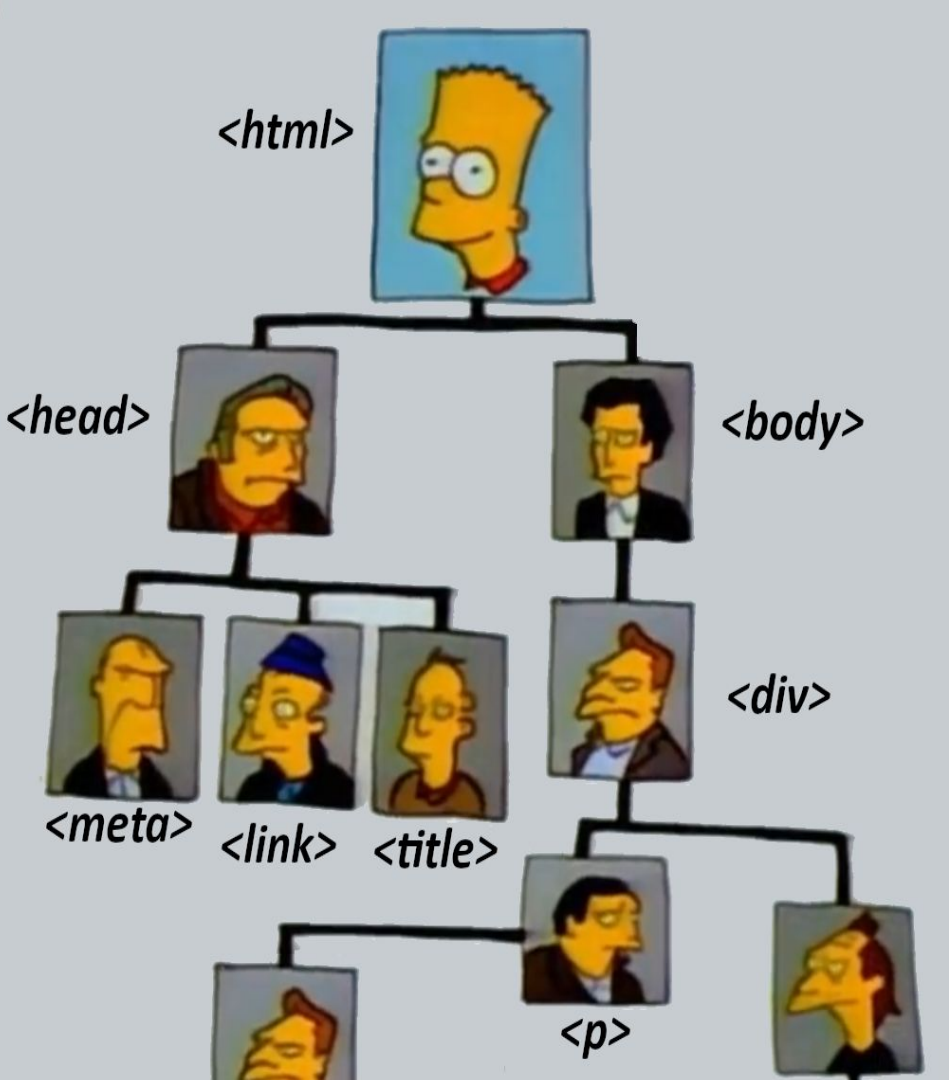
MAPA DE CONCEPTOS CLASE 7



¡Vamos a la clase!

CODERHOUSE

¿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>.

CODERHOUSE

¿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.

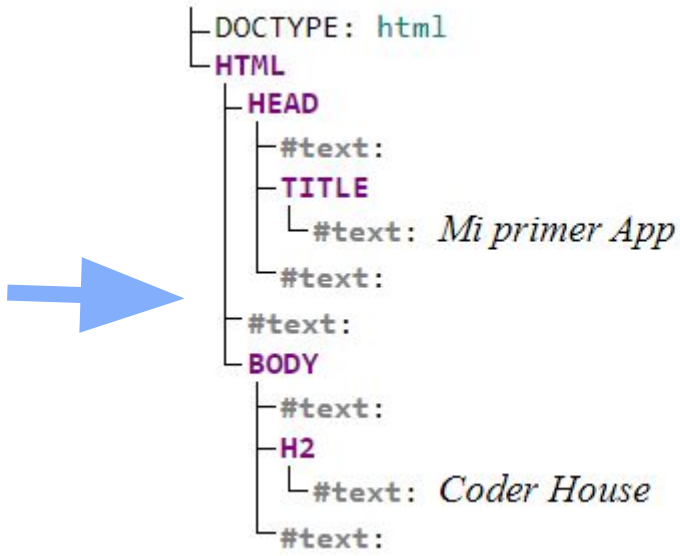
¿Cómo funciona?

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

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

HTML a DOM

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primer App</title>
  </head>
  <body>
    <h2>Coder House</h2>
  </body>
</html>
```



Herramienta: [Live DOM Viewer]

CODERHOUSE

Estructura del DOM y Nodos

CODERHOUSE

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**".

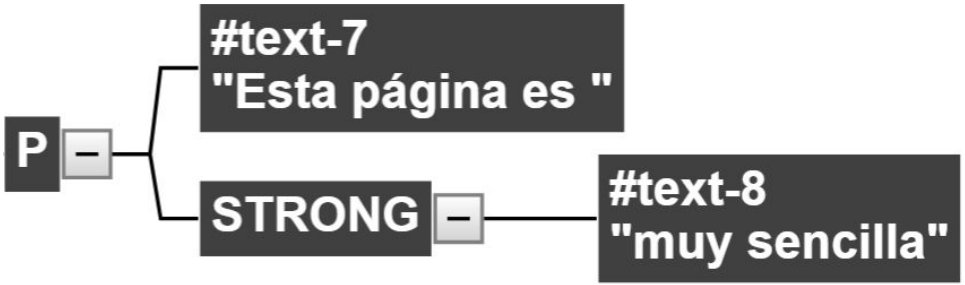
La transformación de las etiquetas HTML habituales genera **dos nodos**

- **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:



Herramienta: DOM node tree viewer

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)

CODERHOUSE

Editar el DOM desde el navegador

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

Referencia: Editar el DOM (Chrome)

CODERHOUSE

Ejemplo aplicado:
Acceso por objeto document

```
console.dir(document);
console.dir(document.head)
console.dir(document.body);
```

El acceso a body usando la referencia document.body requiere que el script se incluya al final del **<body>** en el HTML.

```
<body>
   <h2>Coder House</h2>
   <script src="js/main.js"></script>
</body>
```

Tipos de Nodos

La especificación completa de DOM define 12 tipos de nodos, los más usados son:

Document	Element	Attr	Text	Comment
Nodo raíz del que derivan todos los demás nodos del árbol.	Representa cada una de las etiquetas XHTML. Puede contener atributos y derivar otros nodos de él.	Se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo=valor.	Nodo que contiene el texto encerrado por una etiqueta HTML.	Representa los comentarios incluidos en la página HTML.

Ejemplo en vivo

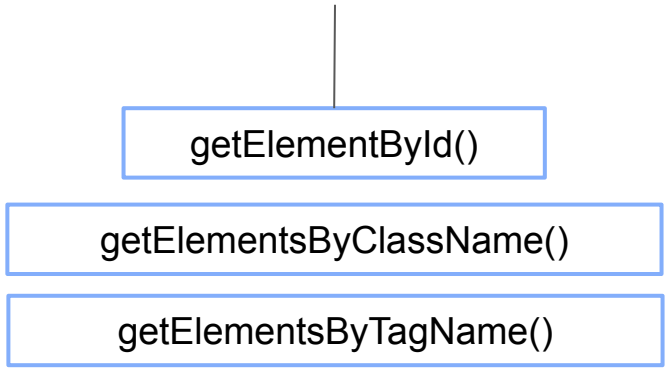
¡VAMOS AL CÓDIGO!

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()

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);
```

CODERHOUSE

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

Innet 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>
```

```JS
//CODIGO
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!"
```

Innet 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.

Innet 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.

Innet 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 row”
//Resultado en el DOM
<div id=”contenedor” class=“container row”>
    <h2>Hola mundo!</h2>
</div>
```

CODERHOUSE

Ejemplo en vivo

¡VAMOS A PRACTICAR LO VISTO!

Break

¡10 minutos y volvemos!

¡Volvemos!

CODERHOUSE

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");
//Elminando 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 màs sencilla
console.log(concatenado);
console.log(plantilla);
```

CODERHOUSE

Plantillas Literales e innerHTML

La 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 = `<h3> 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

Lo interesante del querySelector es que también aplica a pseudo-clases de CSS, brindando un nivel más avanzado de precisión:

```
let radioChecked = document.querySelector(".radio:checked")
```

Suponiendo que tengo elementos html radio button y quiero seleccionar sólo aquel que esté en checked, ésto lo puedo lograr muy fácil con querySelector y la pseudo-clase :checked 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.

CODERHOUSE

Ejemplo en vivo

¡VAMOS A PRACTICAR LO VISTO!

Segunda entrega de tu Proyecto final

Deberás entregar **la estructura del proyecto, las variables de JS necesarias y los objetos de JS**, correspondientes a la segunda entrega de tu proyecto final.

CODERHOUSE

Estructura, variables y objetos

Objetivos generales

- ✓ Codificar la funcionalidad inicial del simulador.
- ✓ Identificar el flujo de trabajo del script en términos de captura de entradas ingresadas por el usuario, procesamiento esencial del simulador y notificación de resultados en forma de salida.

Formato

- ✓ Página HTML y código fuente en JavaScript. Debe identificar el apellido del alumno/a en el nombre de archivo comprimido por “PreEntrega2+Apellido”.

Estructura, variables y objetos

Objetivos específicos

- ✓ Capturar entradas mediante prompt().
- ✓ Declarar variables y objetos necesarios para simular el proceso seleccionado.
- ✓ Crear funciones y/o métodos para realizar operaciones (suma, resta, concatenación, división, porcentaje, etc).
- ✓ Efectuar una salida, que es el resultado de los datos procesados, la cual puede hacerse por alert() o console.log().

Sugerencias

- ✓ Si bien, por el momento solo podemos hacer entradas con prompt() y salidas con alert() o console.log(), es suficiente para empezar a pensar el proceso a simular en términos de entradas, variables, estructuras, funciones, métodos y salidas. Verificar Rúbrica

Estructura, variables y objetos

Se debe entregar

- ✓ Estructura HTML del proyecto.
- ✓ Variables de JS necesarias.
- ✓ Funciones esenciales del proceso a simular.
- ✓ Objetos de JS.
- ✓ Arrays.
- ✓ Métodos de búsqueda y filtrado sobre el Array.

Para tener en cuenta

- ✓ La estructura hace referencia a el html y css, correspondientes al armado de la página general, pero que el JS que se evalúa, aún no está interactuando con ella.

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.

CODERHOUSE

¿Preguntas?

MATERIAL AMPLIADO

Recursos

Ejemplos interactivos: DOM

- ✓ Árbol del Modelo de Objetos del Documento (DOM)
- ✓ Recorriendo el DOM
- ✓ Propiedades de los nodos

Documentación

- ✓ Documentación DOM

Disponible en nuestro repositorio.

Resumen
de la clase hoy

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

Muchas gracias.

Opina y valora esta clase

#DemocratizandoLaEducación