	ESCUELA SUPERIOR DE ECONOMÍA Y NEGOCIOS **INGENIERÍA DE SOFTWARE Y NEGOCIOS DIGITALES**
CICLO 03-2025	**GUIA DE LABORATORIO Nº 7** **Nombre de la practica:** Document Object Model (DOM) **Tiempo estimado:** 2 horas **Materia:** Desarrollo Web I

I. OBJETIVOS

Que al finalizar la practica el estudiante:

- Adquiera un dominio amplio del Modelo de Objetos de Documento (DOM).
- Tenga la capacidad de acceder, eliminar y modificar el contenido de una página web a través del acceso a los nodos que forman el documento.
- Empleen los métodos y propiedades del estándar DOM para tener un dominio completo sobre el documento HTML.

II. INTRODUCCIÓN TEORICA

Los modelos de objetos definen la interfaz para los diversos aspectos del navegador y del documento web que podrán controlarse haciendo uso de JavaScript. En JavaScript se emplean dos modelos de objetos fundamentales, que son:

1. El Modelo de Objetos de Navegador (BOM), y
2. El Modelo de Objetos de Documento (DOM)

El **BOM (Browser Object Document)** proporciona acceso a las diversas características del navegador. Por ejemplo, la ventana del navegador, las características de la pantalla, el historial del navegador, etc. Entre tanto, el **DOM (Document Object Model)**, proporciona acceso al contenido de la ventana del navegador; es decir, el documento web que se muestra (aunque en algunos casos puede ser el conjunto de marcos que forman la página) incluidos los diversos elementos HTML que estén presentes en él, como enlaces, imágenes, tablas, etc.

Uno de los problemas principales es que la división entre el BOM y el DOM es poco clara. Además, las implementaciones de JavaScript de los diferentes navegadores varían significativamente.

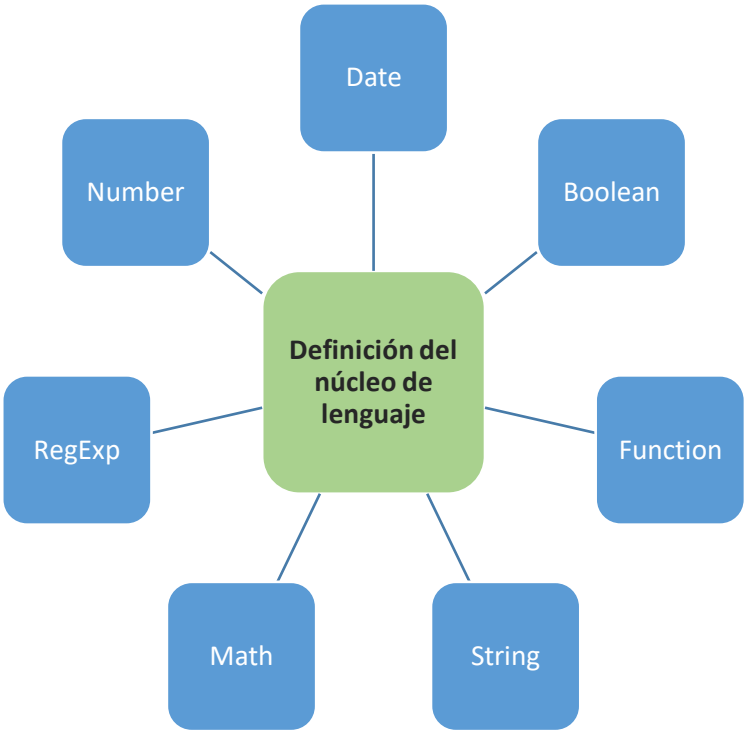
Descripción general del modelo de objetos

El modelo de objetos posee cuatro componentes principales, que son:

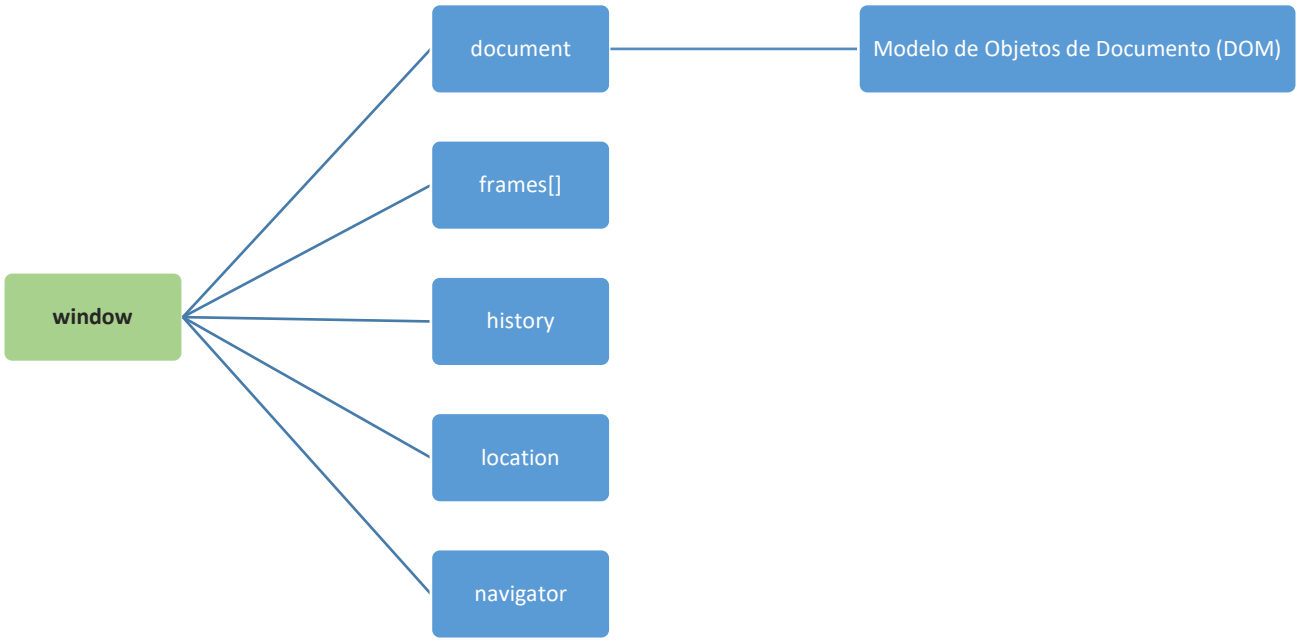
1. Los elementos fundamentales del lenguaje JavaScript (tipos de datos, operadores, instrucciones, etc.)
2. Los objetos fundamentales relacionados esencialmente con los tipos de datos (Date, String, Math, etc.)
3. Los objetos de navegador (Window, Navigator, Location, etc.)
4. Los objetos de documento (Document, Form, Image, etc.).

En la siguiente figura se muestran estos componentes:

Objetos fundamentales y de tipos de datos

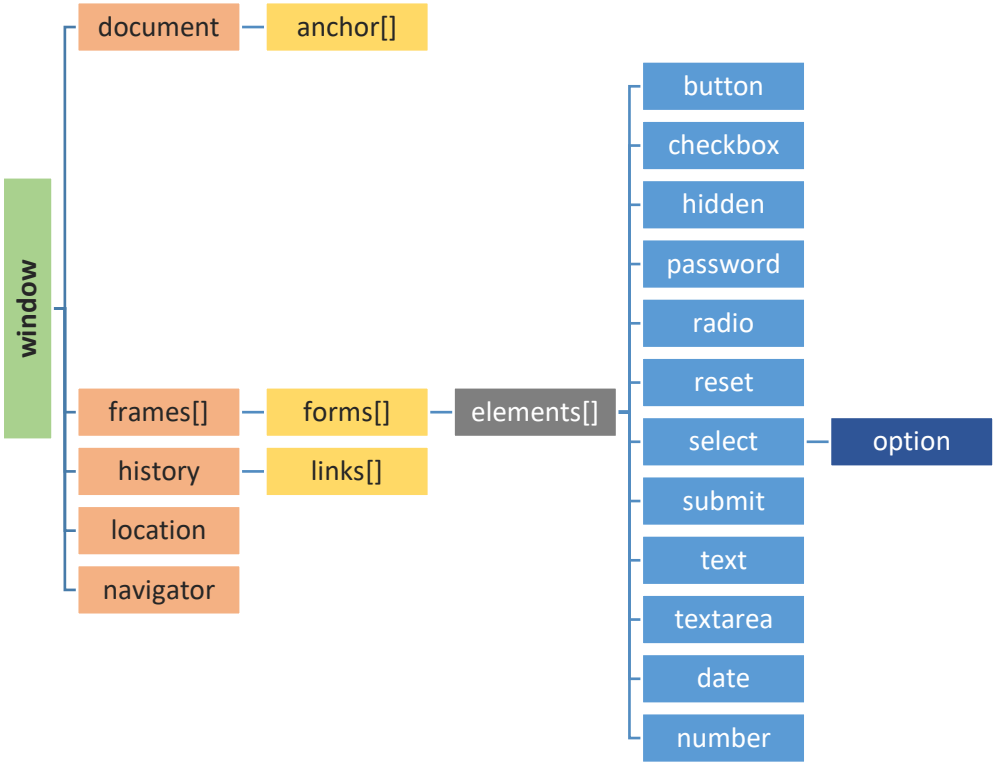


Modelos de objetos del navegador y del documento



Modelo de objetos tradicional

En un principio el modelo de objetos de JavaScript era bastante limitado y se concentraba en dar acceso a las características básicas del navegador y del documento. El modelo de objetos tradicional de JavaScript se muestra a continuación:



Todos estos objetos mostrados se relacionan con JavaScript mediante una sintaxis muy particular, en la que una propiedad de un objeto puede ser también objeto para otro conjunto de propiedades. Así por ejemplo, la propiedad document del objeto Window es objeto para el arreglo (o colección) forms[]. Del mismo modo la colección forms[] tiene los distintos elementos de formulario como propiedades.

Veamos algunos ejemplos que ya conocemos:

Para acceder al método alert() del objeto Window, escribimos la siguiente instrucción:

```
window.alert (mensaje);
```

Del mismo modo para un diálogo de confirmación o de ingreso de datos usamos los métodos confirm() y prompt() respectivamente.

Para mandar a desplegar un mensaje directamente en el documento web, escribimos una instrucción como la siguiente:

```
window.document.write ("<strong>Vamos a aprender a usar el DOM</strong>");
```

La descripción general de cada uno de los objetos mostrados en el modelo de objetos tradicional de JavaScript se muestra en la siguiente tabla:

Objeto	Descripción
window	Este objeto se refiere a la ventana del navegador actual
document	Este objeto contiene todos los elementos HTML y todos los fragmentos de texto que conforman el documento web cargado en la ventana del navegador. En el modelo tradicional de JavaScript hace referencia a lo que se encierra entre las etiquetas <body>… </body> del documento web actual.
frames[]	Es una colección con los marcos del documento web actual, si acaso el documento web está formado por marcos. Sólo bajo esta circunstancia esta propiedad contendrá datos. Cada marco de esta colección referencia a otro objeto window, que a su vez, puede contener otros marcos.
history	Es un objeto que tiene el historial de la ventana actual. Específicamente el conjunto de las diversas direcciones URL visitadas por el usuario recientemente.
location	Este objeto contiene la posición actual del documento que se ha visto en forma de dirección URL y las piezas que lo constituyen, como protocolos, nombre de host y rutas.
navigator	Este objeto muestra características básicas del navegador, como su tipo y versión.

Cuando se están haciendo scripts para controlar dinámicamente lo que se debe mostrar en la página web tenemos varias opciones: por posición, por nombre o por ID. Este último método se explicará en la sección del Modelo de Objetos de Documento (DOM).

Acceso a los elementos del documento por posición

El modelo de objetos de JavaScript tradicional facilita el acceso a un conjunto limitado de elementos HTML, tales como: anclas, enlaces, formularios y controles (elementos) de formulario.

```
1  <html>
2  <head>
3      <title>Formulario sencillo</title>
4  </head>
5  <body>
6      <form>
7          <input type="text">
8      </form>
9      <br><br>
10     <form>
11         <input type="text"><br>
12         <input type="text">
13     </form>
14 </body>
15 </html>
```

Cuando se carga una página web dentro de la ventana de un navegador se instancian objetos de JavaScript para todos los elementos HTML que son codificables en secuencias de comandos scripts. Ya se dijo que este conjunto de elementos codificables es limitados para el modelo de objetos tradicional.

Al observar el siguiente ejemplo de documentos web, notaremos la forma en que podemos acceder a los elementos del documento web a través de su posición dentro de la página web generada:

El ejemplo anterior contiene dos formularios y tres cuadros de texto. Para poder acceder a la primera etiqueta <form> usando acceso por posición de acuerdo al modelo tradicional, debería escribirse una sentencia como la siguiente:

```
window.document.forms[0];
```

Para acceder al segundo, habría que digitar:

```
window.document.forms[1];
```

Del mismo modo si queremos tener acceso al segundo cuadro de texto del segundo formulario. Debe digitar algo como esto:

```
window.document.forms[1].elements[1].text;
```

El método anterior es funcional en tanto no se modifique la posición de los elementos dentro del documento web. Esto es el principal inconveniente, porque en muchas ocasiones se requerirá cambiar de posición los elementos de formulario, o incluso, eliminarlos o agregar nuevos. Esto generaría un caos en el código JavaScript que se tuviese digitado en ese momento, porque sería necesario cambiarlo todo. Por ello se ideó una mejor forma de tener acceso a los elementos del documento, que es la que se explica a continuación.

Métodos tradicionales

1. **Acceso a los elementos por nombre**
   Es muy conveniente la colocación de nombres para todos los elementos HTML que se utilizan en una página. Esto facilita que los lenguajes de script tengan acceso a ellos más adelante. Se puede utilizar el atributo id, que es permitido casi en todos los elementos HTML conocidos. El objetivo de este atributo es vincular un identificador único al elemento.
   Para acceder al formulario definido por:

```
<form name="myform" id="myform">
    <input type="text" name="username" id="username">
</form>
```

   Mediante su nombre, debería digitar lo siguiente:

```
window.document.myform;
```

   Recuerde que como el objeto window se puede asumir, también sería válido digitar:

```
document.myform;
```

2. **Acceso a los elementos usando arreglos asociativos.**
   Es una estructura que le permite asociar datos con nombres. JavaScript proporciona *arrays* asociativos como consecuencia del hecho de que las siguientes dos instrucciones son equivalentes object.property, object["property"].

   En la mayoría de los casos los arreglos (llamados también colecciones) presentes en el objeto **Document** son asociativos. Esto significa que podemos referirnos a ellos haciendo uso de cadenas de texto como índice del arreglo. Esta cadena tiene que ser el nombre del elemento definido al momento de crear el

documento web. Se suele utilizar los atributos ***id*** y ***name*** para establecer estos nombres. Se utilizan dos por razones de compatibilidad. Si bien es cierto, en los navegadores más actuales se usa el atributo id, para nombrar al control, navegadores más antiguos no reconocen ese atributo y por eso es que se suele usar el atributo ***name*** junto con el atributo ***id***.

Si se tiene el siguiente ejemplo:

```
<form name="myform">
    <input type="text" name="mytext" value="">
</form>
```

Para tener acceso al valor introducido en el campo de texto debemos escribir una sentencia como la siguiente:

```
document.forms["myform"].elements["mytext"].value;
```

3. **Acceso a los elementos usando getElementById(id).**

   .getElementById(id) busca un elemento HTML por medio del id, el parámetro que recibe es un id. En principio, un documento HTML bien construido no debería tener más de un elemento con el mismo id, por lo tanto, este método devolverá siempre un solo elemento. Se posee la siguiente estructura HTML:

```
<body>
    <div id="page"></div>
</body>
```

   Para acceder al con id=”page”, se realiza la siguiente instrucción en JavaScript:

```
const page = document.getElementById("page");   // <div id="page"></div>
```

4. **Acceso a los elementos usando getElementsByName(name).**

   El método .getElementsByName(name) busca en elemento HTML por medio del atributo name, veamos el siguiente ejemplo:

```
<body>
    <p name="nickname"></p>
</body>
```

   Ahora para acceder al elemento por medio del atributo name, realizamos la siguiente línea de código:

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

5. **Acceso a los elementos usando getElementsByClassName(class).**

   Por otro lado, el método .getElementsByClassName(class) permite buscar los elementos con la clase especificada en class. Es importante darse cuenta de la matriz que el método tiene, ya que se escribe getElements en plural, y esto es porque al devolver clases (al contrario que los id) se pueden repetir, y por lo tanto, devolvernos varios elementos, no sólo uno.

   Se muestra la siguiente estructura, notese que se posee cuatro con la clase “item”.

```
<body>
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
</body>
```

Para poseer la colección de los cuatro , realizamos la siguiente línea de código:

```
const items = document.getElementsByClassName("item"); // [div, div, div, div]

console.log(items[0]); // Primer item encontrado: <div class="item"></div>
console.log(items.length); // 4
```

6. **Acceso a los elementos usando getElementsByTagName(tag).**
   El método. getElementsByTagName(tag) busca en elemento HTML por medio del atributo name, veamos el siguiente ejemplo:

```
<body>
    <div>Bienvenido al curso de LIC</div>
</body>
```

   Ahora para acceder al elemento por medio de la etiqueta HTML, realizamos la siguiente línea de código:

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

Métodos modernos

1. **Acceso a los elementos por medio de .querySelector()**
   .querySelector(selector) devuelve el primer elemento que encuentra que encaja con el selector CSS suministrado en selector. Al igual que su «equivalente» .getElementById(), devuelve un solo elemento y en caso de no coincidir con ninguno, devuelve null:

```
<body>
    <div id="page"></div>
    <div class="info"></div>
</body>
```

   Para acceder desde JavaScript realizamos la siguiente línea de código:

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

   Lo interesante de este método, es que al permitir suministrarle un selector CSS básico o incluso un selector CSS avanzado, se vuelve un sistema mucho más potente.

   El primer ejemplo es equivalente a utilizar un .getElementById(), sólo que en la versión de .querySelector() indicamos por parámetro un SELECTOR, y en el primero le pasamos un simple STRING. Observa que estamos indicando un # porque se trata de un **id**.

   En el segundo ejemplo, estamos recuperando el primer elemento con clase info que se encuentre dentro de otro elemento con clase main. Eso podría realizarse con los métodos tradicionales, pero sería menos directo ya que tendríamos que realizar varias llamadas, con .querySelector() se hace directamente con sólo una.

2. **Acceso a los elementos por medio de .querySelectorAll()**
   Por otro lado, el método .querySelectorAll() realiza una búsqueda de elementos como lo hace el anterior, sólo que como podremos intuir por ese **All()**, devuelve un ARRAY con todos los elementos que coinciden con el SELECTOR CSS:

```html
<body>
    <div id="page"></div>
    <div class="info"></div>
    <p name="nickname"></p>
</body>
```

Accediendo desde JavaScript a cada elemento:

```javascript
// 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");
```

En este caso, recuerda que .querySelectorAll() siempre nos devolverá un de elementos. Depende de los elementos que encuentre mediante el SELECTOR, nos devolverá un ARRAY de **0** elementos o de **1, 2** o más elementos.

Modelo de Objetos de Documento

El **DOM** (**Document Object Model**: **Modelo de Objetos de Documento**) es una interfaz de programación de aplicaciones (API) propuesta por la W3C que crea una correspondencia entre un documento HTML o XML y la jerarquía de objetos del documento presentada al programador. Es mediante el uso de esta interfaz que la totalidad de los elementos que componen una página web (etiquetas, atributos, estilos y contenido) están a disposición de un lenguaje de programación o de scripts, como el caso de JavaScript.

De acuerdo con este estándar los documentos HTML son concebidos como una estructura en la que se realiza una correspondencia entre las etiquetas dispuestas en el documento web con un árbol de nodos. La manipulación de estos nodos es el método recomendado por la W3C para que los navegadores compatibles con los estándares soporten las páginas web que funcionan más como aplicaciones que como típicas páginas estáticas. Entre los usos más comunes del DOM están la exploración de la estructura del documento, el acceso a las propiedades y los métodos comunes, la modificación del contenido de una página web, la eliminación de elementos, la manipulación de reglas de estilo y la creación dinámica de elementos HTML.

Niveles del DOM

El consorcio W3C propone tres niveles del DOM:

- **DOM Nivel 0**: Es el llamado Modelo de Objetos de JavaScript clásico o tradicional. Este nivel del DOM es equivalente a lo que han soportado el Netscape 3.0 e Internet Explorer 3.0.
- **DOM Nivel 1**: Define las interfaces principales del DOM, como Node, Element, Attr y Document, mediante las cuales es posible manipular todos los elementos de un documento, permitiendo leer y escribir partes de una página web en todo moemento. Este nivel del DOM proporciona capacidades similares a las de la colección document.all[].
- **DOM Nivel 2**: Incorpora elementos de página relacionados con XML y soporte para Hojas de Estilo en Cascada (CSS). Se centra además en la combinación del DOM Nivel 0 y el DOM Nivel 1. Incorpora, además, soporte para el modelo de eventos que fusiona los conceptos de Netscape 4 e Internet Explorer 4.

Estructura jerárquica de árbol

Lo que el estándar DOM de la W3C establece, es que todos los elementos presentes en una página web siguen una estructura jerárquica por debajo del objeto document, que se corresponde con el elemento HTML, que se convierte en el elemento padre de todos los elementos presentes en un documento web.

El documento siguiente:

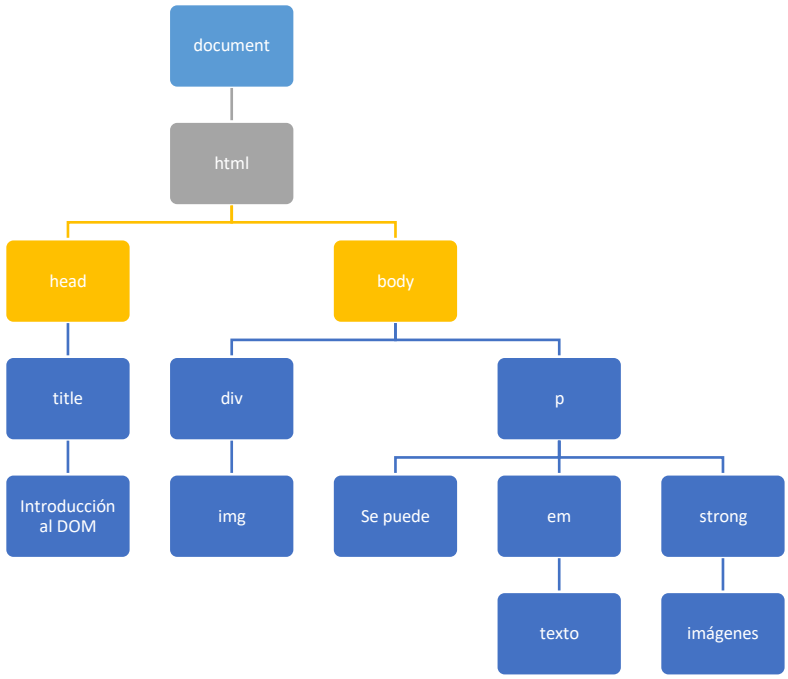
```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Introducción al DOM</title>
</head>

<body>
    <!-- Comentario -->
    <div>
        <img src="imagen.jpg">
    </div>
    <p>
        Se puede incluir <em>texto</em> e
        <strong>imágenes</strong>.
    </p>
</body>

</html>
```

Se puede representar mediante una estructura jerárquica como la siguiente:



Puede observarse que todos los nodos del árbol de documento son nodos de document. Del mismo modo, puede notarse que head y body son hijos del nodo html, que también es hijo de document. La sección title es hijo de head, y así, sucesivamente.

Tipos de nodos

Los diversos tipos de nodos pueden ser, de acuerdo a su naturaleza, de hasta 12 tipos diferentes. Sin embargo, los que están estrictamente relacionados con HTML, son 6, que se describen a continuación:

No. Tipo de nodo	Tipo	Descripción	Ejemplo
1	Elemento	Un elemento HTML o XML	Texto
2	Atributo	Un atributo para un elemento HTML o XML	cols="12"
3	Texto	Un fragmento de texto incluido en un elemento HTML o XML.	Esto es un fragmento de texto
8	Comentario	Comentario	<!-- Este es un comentario HTML -->
9	Documento	Documento	<html>
10	Tipo de documento	Tipo de documento	<!DOCTYPE html>

Los **nodos de elemento** son aquellos que están formados por una etiqueta HTML. Por ejemplo: <a> … , … , … , <img>, etc. Independientemente de si es un elemento abierto o cerrado (con etiqueta de cierre, además de apertura).

Los **nodos de texto** son los fragmentos de texto que se incluyen en una página web. Para que sean correctamente tratados deben de estar incluidos dentro de un nodo de elemento. Por ejemplo: Esto es un nodo de texto . El fragmento "Esto es un nodo de texto" constituye un nodo de texto.

Jerarquías entre nodos

- **Nodo padre**: es aquel nodo que tiene otros nodos bajo de él. Un nodo de texto no puede ser nodo padre, puesto que jerárquicamente, de un fragmento de texto no puede depender ningún elemento de la página. Los nodos de elemento sólo pueden ser nodos padre si se tratan de elementos con etiquetas de cierre. Un elemento vacío, por tanto, no puede ser nodo padre.
- **Nodo hijo**: son los nodos que están por debajo de otros nodos en un diagrama de árbol de documento. Un nodo determinado puede poseer varios nodos hijo.
- **Nodo hermano**: Son los nodos que están al mismo nivel de jerarquía que otro nodo determinado. Es decir, ambos nodos poseen el mismo nodo padre.

Dentro de un documento los nodos forman matrices, denominadas genéricamente **childNodes**. Así, el objeto document tiene una matriz childNodes con un único elemento (**html**). Éste, a su vez, posee una matriz childNodes con dos elementos (**head** y **body**). Sucesivamente, el nodo head, posee normalmente su propia matriz childNodes, donde generalmente el elemento **title** es uno de sus hijos.

Algo muy importante a tener en cuenta es que para hacer referencia a cualquier nodo, es imprescindible que éste haya sido cargado en la ventana del navegador. Por lo tanto, será importante que el código que hace referencia a nodos esté presente al final del elemento body. De lo contrario, tendrá que colocar el código dentro de funciones que deberán ser invocadas mediante el manejador de evento onload asociado al elemento body del documento.

Propiedades y métodos de los nodos

La siguiente tabla resume las principales propiedades y métodos de los nodos definidos por el estándar W3C:

#	Propiedad o método	Utilidad
1	className	Indica o establece el origen de clase CSS que afecta al nodo referido.
2	firstChild	Se refiere al primer nodo hijo de aquél con el que estemos trabajando.
3	lastChild	Se refiere al último nodo hijo de aquél con el que estemos trabajando.
4	nextSibling	Se refiere al nodo hermano siguiente de aquél con el que estemos trabajando.
5	nodeName	Se refiere al nombre que identifica al nodo actual.
6	nodeType	Se refiere al tipo de nodo (elemento, atributo, comentario, texto, etc.)
7	nodeValue	El texto que constituye un nodo. Aplicable a nodos de texto. Esta propiedad es null para un nodo de tipo elemento.
8	ownerDocument	Se refiere al documento propietario de aquél con el que estamos trabajando.
9	parentNode	Se refiere al nodo padre de aquél con el que estamos trabajando.
10	previousSibling	Se refiere al nodo hermano anterior de aquél con el que se está trabajando.
11	tagName	El nombre de la etiqueta de un nodo.
12	appendChild()	Añade un nodo hijo a aquél nodo con el que se está trabajando.
13	cloneNode()	Copia un nodo.
14	createElement()	Crea un nodo de tipo elemento para añadirlo como hijo, al nodo con el que se está trabajando.
15	createTextNode()	Crea un nodo de texto para añadirlo como hijo, al nodo con el que se está trabajando.
16	getAttribute()	Obtiene el valor de un atributo.
17	getElementById()	Se usa para referirse a un nodo mediante su valor de id.
18	getElementsByTagName()	Permite referirse a un nodo (o conjunto de nodos) por el nombre de la etiqueta.
19	hasChildNodes()	Determina si un nodo tiene hijos.
20	insertBefore()	Inserta un nodo hijo en la matriz childNodes de aquél con el que estamos trabajando.
21	removeAttribute()	Elimina un atributo de un nodo de un elemento.
22	removeChild()	Elimina el hijo indicado en la matriz childNodes del nodo con el que se está trabajando.
23	replaceChild()	Sustituye el hijo indicado de la matriz childNodes del nodo con el que se está trabajando por otro.
24	setAttribute()	Establece un atributo del nodo elemento con el que se está trabajando y su valor.
25	createComment(text)	Crea y devuelve un nodo de comentarios HTML <!-- text -->
26	isConnected	Inicia si el nodo HTML está insertando en el documento HTML.

El objeto Document

El objeto Document posee varios (sub) objetos, a veces en forma de colecciones, además de métodos que tienen por propósito posibilitar la manipulación de los elementos presentes en la página web.

Entre las colecciones del objeto Document se pueden mencionar: forms[], links[], images[], anchors[]. Casi todos los objetos de JavaScript del DOM poseen además del objeto una colección correspondiente.

Los métodos más importantes del objeto Document son:

1. getElementById(id): Devuelve el elemento con el identificador id. Si no existe, devuelve NULL.
2. getElementsByTagName(tag). Devuelve un objeto NodeList de todos los elementos en el documento con esta etiqueta tag.

Estos métodos permiten el acceso directo a un elemento mediante su id o a una lista o colección de nodos con el mismo nombre.

Otros métodos del objeto Document son:

1. createElement(tag), que crea un elemento dentro de la estructura del documento. El argumento tag, debe ser el nombre de un elemento HTML válido.
2. createTextNode(text), que crea un nodo de texto con el texto text que luego debería ser agregado a un nodo de tipo elemento.

El sub objeto Element

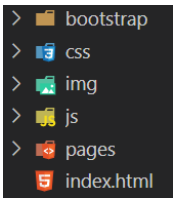
El tipo de nodo Element amplía el objeto Node con varios métodos y propiedades. Puede poseer diversos tipos de nodos hijos, como: Element, Text, Comment, etc.

Además posee una serie de métodos, como los que se describen a continuación:

1. getAttribute(att) que devuelve el valor del atributo att aplicado sobre el elemento o una cadena vacía si este atributo no tiene ningún valor.
2. setAttribute(att, val) que asigna el valor val al atributo att. Si el atributo no existiese, se crea uno nuevo con el nombre indicado.
3. removeAttribute(att) que elimina el atributo att sobre el elemento aplicado.

III. DESARROLLO DE LA PRÁCTICA

Cree una carpeta con el nombre Guia7_[su número de carnet], y luego proceda a descomprimir los recursos proporcionados. Deberá de tener la siguiente estructura en su proyecto.



Ejemplo 1: Recuperando elementos HTML con DOM.

El ejemplo consiste en encontrar elementos HTML y contarlos.

1. Localice el archivo index.js y comience a definir las siguientes referencia a botones:

```
// OBTENIENDO LA REFERENCIA DE LOS BOTONES
// POR MEDIO DEL .getElementById

const buttonSpan = document.getElementById("idBtnSpan");
const buttonP = document.getElementById("idBtnP");
const buttonDiv = document.getElementById("idBtnDiv");
const buttonButton = document.getElementById("idBtnButton");
const imprimir = document.getElementById("idImprimirResultado");

```

2. Defina la siguiente función para contar los elementos dentro de nuestro documento HTML.

```
// DEFINICION DE FUNCIONES
const contarElementos = function (elemento) {
  // OBTENIENDO EL NUMERO DE ETIQUETAS SPAN QUE SE HAN CREADO
  // EN EL DOCUMENTO HTML
  let arrayElement = document.getElementsByTagName(elemento);
  console.log(
    `Etiquetas buscadas <${elemento}></${elemento}> / Total encontradas : ${arrayElement.length}`
  );
  for (const i of arrayElement) {
    console.log(i);
  }

  alert("Revise la consola del navegador");
};
```

3. Ahora defina los eventos clic para cada botón.

```
//DEFINICION DE EVENTOS PARA LOS BOTONES
buttonSpan.onclick = () => {
  contarElementos("span");
};

buttonP.onclick = () => {
  contarElementos("p");
};

buttonDiv.onclick = () => {
  contarElementos("div");
};

buttonButton.onclick = () => {
  contarElementos("button");
};
```

4. Verifique el funcionamiento de su página index.html, tendría que obtener un resultado como el siguiente.

Ejemplo 2: Creando formulario con DOM.

El ejemplo permite crear un formulario desde cero, tendrá que seleccionar el tipo de elemento a crear y luego pulsar el botón.

1. Localice el archivo ejemplo1.js y comience a definir las siguientes referencias a elementos HTML:

```
// ACCEDIENDO A LA REFERENCIA DEL FORMULARIO QUE
// TENDRA LOS NUEVOS ELEMENTOS
const newForm = document.getElementById("idNewForm");

// ACCEDIENDO A LA REFERENCIA DE BOTONES
const buttonCrear = document.getElementById("idBtnCrear");
const buttonAddElemento = document.getElementById("idBtnAddElement");

// ACCEDIENDO AL VALOR DEL SELECT PARA DETERMINAR EL TIPO DE ELEMENTO A CREAR
const cmbElemento = document.getElementById("idCmbElemento");

// ACCEDIENDO A LOS CONTROLES DEL MODAL
const tituloElemento = document.getElementById("idTituloElemento");
const nombreElemento = document.getElementById("idNombreElemento");

// CREANDO MODAL CON BOOTSTRAP
const modal = new bootstrap.Modal(document.getElementById("idModal"), {});
```

2. Defina la siguiente función para verificar el tipo de elemento que se creara en el formulario.

```
// AGREGANDO FUNCIONES
const vericarTipoElemento = function () {
  let elemento = cmbElemento.value;
  //validando que se haya seleccionado un elemento
  if (elemento != "") {
    // Metodo perteneciente al modal de boostrap
    modal.show();
  } else {
    alert("Debe seleccionar el elemento que se creara");
  }
};
```

3. Defina la siguiente función para crear un elemento de tipo SELECT.

```
const newSelect = function () {
    // Creando elementos
    let addElemento = document.createElement("select");
    //creando atributos para el nueveo elemento
    addElemento.setAttribute("id", `id${nombreElemento.value}`);
    addElemento.setAttribute("class", "form-select");

    //creando option para el select
    for (let i = 1; i <= 10; i++) {
      let addOption = document.createElement("option");
      addOption.value = i;
      addOption.innerHTML = `Opcion ${i}`;
      addElemento.appendChild(addOption);
    }

    //creando label para el nuevo control
    let labelElemento = document.createElement("label");
    labelElemento.setAttribute("for", `id${nombreElemento.value}`);
    //creando texto para label
    labelElemento.textContent = tituloElemento.value;

    //Creando label de id
    let labelId = document.createElement("span");
    labelId.textContent = `ID de control : ${nombreElemento.value}`;

    // Creando plantilla de bootstrap para visualizar el nuevo elemento
    let divElemento = document.createElement("div");
    // Agregando atributos
    divElemento.setAttribute("class", "form-floating");

    //Creando el input que sera hijo del div
    divElemento.appendChild(addElemento);
    //Creando el label que sera hijo del div
    divElemento.appendChild(labelElemento);

    //Creando el SPAN que sera hijo del nuevo Formulario
    newForm.appendChild(labelId);

    //Creando el Div que sera hijo del nuevo Formulario
    newForm.appendChild(divElemento);
};
```

4. Defina la siguiente función para crear un elemento de tipo RADIO o CHECKBOX.

```
73  const newRadioCheckbox = function (newElemento) {
74    // Creando elementos
75    let addElemento = document.createElement("input");
76    //creando atributos para el nueveo elemento
77    addElemento.setAttribute("id", `id${nombreElemento.value}`);
78    addElemento.setAttribute("type", newElemento);
79    addElemento.setAttribute("class", "form-check-input");
80
81    //creando label para el nuevo control
82    let labelElemento = document.createElement("label");
83    labelElemento.setAttribute("class", "form-check-label");
84    labelElemento.setAttribute("for", `id${nombreElemento.value}`);
85    //creando texto para label
86    labelElemento.textContent = tituloElemento.value;
87
88    //Creando label de id
89    let labelId = document.createElement("span");
90    labelId.textContent = `ID de control : ${nombreElemento.value}`;
91
92    // Creando plantilla de bootstrap para visualizar el nuevo elemento
93    let divElemento = document.createElement("div");
94    // Agregando atributos
95    divElemento.setAttribute("class", "form-check");
96
97    //Creando el input que sera hijo del div
98    divElemento.appendChild(addElemento);
99    //Creando el label que sera hijo del div
100   divElemento.appendChild(labelElemento);
101
102   //Creando el SPAN que sera hijo del nuevo Formulario
103   newForm.appendChild(labelId);
104
105   //Creando el Div que sera hijo del nuevo Formulario
106   newForm.appendChild(divElemento);
107 };
108
```

5. Defina la siguiente función para crear un elemento de tipo TEXTAREA, TEXT, NUMBER, DATE U OTRO.

```
const newInput = function (newElemento) {
  // Creando elementos de tipo = text, number, date y password
  let addElemento =
    newElemento == "textarea"
      ? document.createElement("textarea")
      : document.createElement("input");

  //creando atributos para el nueveo elemento
  addElemento.setAttribute("id", `id${nombreElemento.value}`);
  addElemento.setAttribute("type", newElemento);
  addElemento.setAttribute("class", "form-control");
  addElemento.setAttribute("placeholder", tituloElemento.value);

  //creando label para el nuevo control
  let labelElemento = document.createElement("label");
  labelElemento.setAttribute("for", `id${nombreElemento.value}`);

  //creando icono para el label
  let iconLabel = document.createElement("i");
  iconLabel.setAttribute("class", "bi bi-tag");

  //creando texto para label
  labelElemento.textContent = tituloElemento.value;

  //creando el elemento i como hijo del label, afterbegin le
  // indicamos que se creara entes de su primer hijo
  labelElemento.insertAdjacentElement("afterbegin", iconLabel);

  //Creando label de id
  let labelId = document.createElement("span");
  labelId.textContent = `ID de control : ${nombreElemento.value}`;

  // Creando plantilla de bootstrap para visualizar el nuevo elemento
  let divElemento = document.createElement("div");
  // Agregando atributos
  divElemento.setAttribute("class", "form-floating mb-3");

  //Creando el input que sera hijo del div
  divElemento.appendChild(addElemento);
  //Creando el label que sera hijo del div
  divElemento.appendChild(labelElemento);

  //Creando el SPAN que sera hijo del nuevo Formulario
  newForm.appendChild(labelId);

  //Creando el Div que sera hijo del nuevo Formulario
  newForm.appendChild(divElemento);
};
```

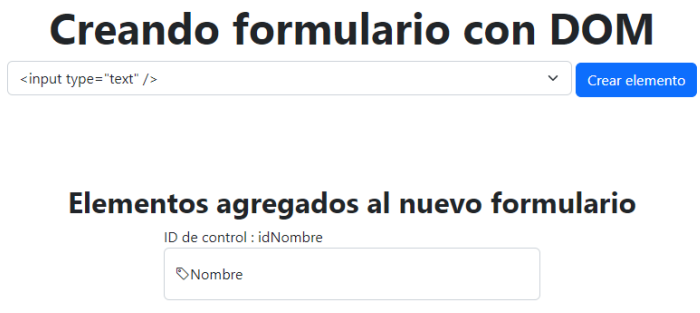
6. Agregue eventos a los botones

```
158  // AGREGANDO EVENTO CLIC A LOS BOTONES
159  buttonCrear.onclick = () => {
160    vericarTipoElemento();
161  };
162
163  buttonAddElemento.onclick = () => {
164    if (nombreElemento.value != "" && tituloElemento.value != "") {
165      let elemento = cmbElemento.value;
166
167      if (elemento == "select") {
168        newSelect();
169      } else if (elemento == "radio" || elemento == "checkbox") {
170        newRadioCheckbox(elemento);
171      } else {
172        newInput(elemento);
173      }
174    } else {
175      alert("Faltan campos por completar");
176    }
177  };
```

7. Agregue la siguiente función para limpiar el formulario creado por el MODAL de Bootstrap.

```
179  // Agregando evento para el modal de bootstrap
180  document.getElementById("idModal").addEventListener("shown.bs.modal", () => {
181    // Limpiando campos para los nuevos elementos
182    tituloElemento.value = "";
183    nombreElemento.value = "";
184    // inicializando puntero en el campo del titulo para el control
185    tituloElemento.focus();
186  });
```

8. Verifique el funcionamiento de su página ejemplo1.html, tendría que obtener un resultado como el siguiente.



Ejemplo 3: Recorriendo un formulario con DOM.

El ejemplo permite recorrer un formulario y mostrar la cantidad de elementos que posee como hijos.

1. Localice el archivo ejemplo2.js y comience a definir las siguientes referencia a elementos HTML:

```
// Obteniendo la referencia de los elementos
// por medio de arreglos asociativos
// aqui se esta utilizando el atributo name de cada elemento
const formulario = document.forms["frmRegistro"];
const button = document.forms["frmRegistro"].elements["btnRegistro"];

// CREANDO MODAL CON BOOTSTRAP
const modal = new bootstrap.Modal(document.getElementById("idModal"), {});

// OBTENIENDO LA REFERENCIA DEL CUERPO DEL MODAL
// PARA IMPRIMIR EL RESULTADO
const bodyModal = document.getElementById("idBodyModal");
```

2. Defina la siguiente función para recorrer los elementos del formulario.

```
// Recorrer el formulario
const recorrerFormulario = function () {
    let totText = 0;
    let totRadio = 0;
    let totCheck = 0;
    let totDate = 0;
    let totSelect = 0;
    let totFile = 0;
    let totPass = 0;
    let totEmail = 0;

    // Recorriendo elementos del formulario
    let elementos = formulario.elements;
    let totalElementos = elementos.length;

    for (let index = 0; index < totalElementos; index++) {
        // Accediendo a cada hijo del formulario
        let elemento = elementos[index];

        // verificando el tipo de control en el formulario
        let tipoElemento = elemento.type;
        // verificando el tipo de nodo
        let tipoNode = elemento.nodeName;

```

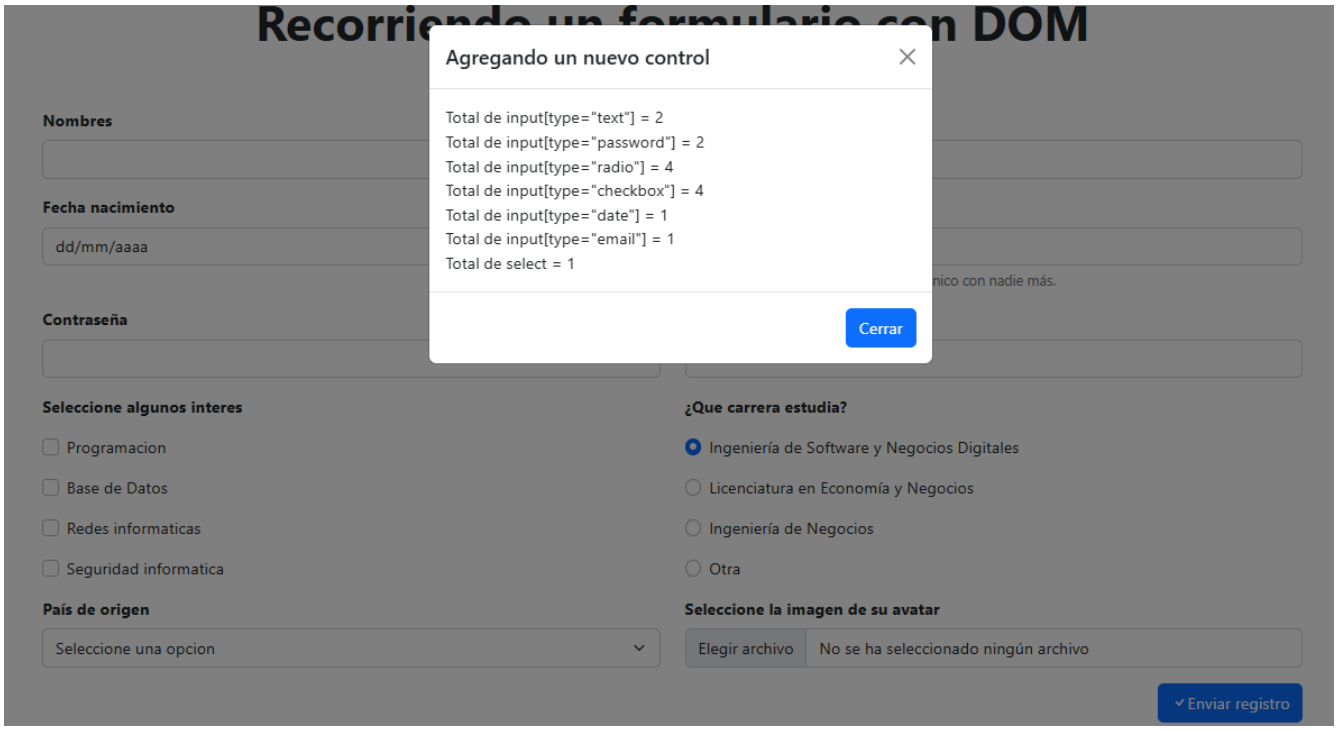
```
    // Contabilizando el total de INPUT TYPE = TEXT
    if (tipoElemento == "text" && tipoNode == "INPUT") {
      console.log(elemento);
      totText++;
    }
    // Contabilizando el total de INPUT TYPE = PASSWORD
    else if (tipoElemento == "password" && tipoNode == "INPUT") {
      console.log(elemento);
      totPass++;
    }
    // Contabilizando el total de INPUT TYPE = EMAIL
    else if (tipoElemento == "email" && tipoNode == "INPUT") {
      console.log(elemento);
      totEmail++;
    }
    // Contabilizando el total de INPUT TYPE = RADIO
    else if (tipoElemento == "radio" && tipoNode == "INPUT") {
      console.log(elemento);
      totRadio++;
    }
    // Contabilizando el total de INPUT TYPE = CHECKBOX
    else if (tipoElemento == "checkbox" && tipoNode == "INPUT") {
      console.log(elemento);
      totCheck++;
    }
    // Contabilizando el total de INPUT TYPE = FILE
    else if (tipoElemento == "file" && tipoNode == "INPUT") {
      console.log(elemento);
      totFile++;
    }
    // Contabilizando el total de INPUT TYPE = CHECKBOX
    else if (tipoElemento == "date" && tipoNode == "INPUT") {
      console.log(elemento);
      totDate++;
    }
    // Contabilizando el total de INPUT TYPE = EMAIL
    else if (tipoNode == "SELECT") {
      console.log(elemento);
      totSelect++;
    }
  }
```

```
80    let resultado = `
81      Total de input[type="text"] = ${totText}<br>
82      Total de input[type="password"] = ${totPass}<br>
83      Total de input[type="radio"] = ${totRadio}<br>
84      Total de input[type="checkbox"] = ${totCheck}<br>
85      Total de input[type="date"] = ${totDate}<br>
86      Total de input[type="email"] = ${totEmail}<br>
87      Total de select = ${totSelect}<br>
88    `;
89
90    bodyModal.innerHTML = resultado;
91    //Funcion que permite mostrar el modal de Bootstrap
92    //Esta funcion es definida por Bootstrap
93    modal.show();
94  };
```

3. Agregue eventos a los botones

```
96  // agregando eventos al boton
97  button.onclick = () => {
98    recorrerFormulario();
99  };
```

4. Verifique el funcionamiento de su página ejemplo2.html, tendría que obtener un resultado como el siguiente.



Ejemplo 4: Manipulación de estilos con DOM

El ejemplo permite manipular el color de fondo de la página web, títulos y párrafos, adicionalmente demuestra que se puede alterar el tamaño de la letra.

1. Localice el archivo ejemplo3.js y comience a definir las siguientes funciones:

```
// Inicializando primer color de fondo en el input color
const primerColorFondo = function (event) {
  document.body.style.backgroundColor = event.target.value;
};

const cambiarColorFondo = function (color) {
  document.body.style.backgroundColor = color;
};

//Funciones para modificar el color de los titulos
const primerColorTitulos = function (event) {
  let colorSeleccionado = event.target.value;
  const titulos = document.querySelectorAll("h1");
  for (let index = 0; index < titulos.length; index++) {
    titulos[index].style.color = colorSeleccionado;
  }
};

const cambiarColorTitulos = function (colorSeleccionado) {
  const titulos = document.querySelectorAll("h1");
  for (let index = 0; index < titulos.length; index++) {
    titulos[index].style.color = colorSeleccionado;
  }
};

//Funciones para modificar el color de los parrafos
const primerColorParrafos = function (event) {
  let colorSeleccionado = event.target.value;
  const parrafos = document.querySelectorAll("p");
  for (let index = 0; index < parrafos.length; index++) {
    parrafos[index].style.color = colorSeleccionado;
  }
};

const cambiarColorParrafos = function (colorSeleccionado) {
  const parrafos = document.querySelectorAll("p");
  for (let index = 0; index < parrafos.length; index++) {
    parrafos[index].style.color = colorSeleccionado;
  }
};
```

2. Defina las siguientes funciones para manipular el tamaño de la fuente mostrada en el documento HTML.

```
let contadorAumentar = 1;
const aumentarLetra = function () {
  contadorAumentar += 0.005;
  document.body.style.fontSize = `${contadorAumentar}em`;
  const parrafos = document.querySelectorAll("p");
  for (let index = 0; index < parrafos.length; index++) {
    parrafos[index].style.fontSize = `${contadorAumentar}em`;
  }
  const titulos = document.querySelectorAll("h1");
  for (let index = 0; index < titulos.length; index++) {
    titulos[index].style.fontSize = `${contadorAumentar}em`;
  }
};

let contadorDisminuir = 1;
const disminuirLetra = function () {
  contadorDisminuir -= 0.005;
  document.body.style.fontSize = `${contadorDisminuir}em`;
  const parrafos = document.querySelectorAll("p");
  for (let index = 0; index < parrafos.length; index++) {
    parrafos[index].style.fontSize = `${contadorDisminuir}em`;
  }
  const titulos = document.querySelectorAll("h1");
  for (let index = 0; index < titulos.length; index++) {
    titulos[index].style.fontSize = `${contadorDisminuir}em`;
  }
};
```

3. Definamos una función principal para inicializar referencia y eventos a elementos HTML. Aquí podrá observar que existe un evento input y change para el tipo de elemento <input type=”color”>, ambos se complementan para capturar en tiempo real el color seleccionado.

```
const startDOM = () => {
    // Obteniendo la referencia del boton cambiar fondo
    const buttonFondo = document.getElementById("idFondo");
    buttonFondo.value = "#ffffff";
    buttonFondo.addEventListener("input", primerColorFondo, false);
    buttonFondo.addEventListener("change", cambiarColorFondo, false);
    buttonFondo.select();

    // Obteniendo la referencia del boton cambiar color de titulos
    const buttonTitulos = document.getElementById("idTitulos");
    buttonTitulos.value = "#000000";
    buttonTitulos.addEventListener("input", primerColorTitulos, false);
    buttonTitulos.addEventListener("change", cambiarColorTitulos, false);
    buttonTitulos.select();

    // Obteniendo la referencia del boton cambiar color de parrafos
    const buttonParrafos = document.getElementById("idParrafos");
    buttonParrafos.value = "#000000";
    buttonParrafos.addEventListener("input", primerColorParrafos, false);
    buttonParrafos.addEventListener("change", cambiarColorParrafos, false);
    buttonParrafos.select();

    //Obteniendo las referencias de los botonos
    const buttonAumentar = document.getElementById("idBtnAumentar");
    const buttonDisminuir = document.getElementById("idBtnDisminuir");

    buttonAumentar.addEventListener("click", aumentarLetra, false);
    buttonDisminuir.addEventListener("click", disminuirLetra, false);
};
```

4. Verifique el funcionamiento de su página ejemplo3.html, tendría que obtener un resultado como el siguiente.

Ejemplo 4: Creando un sitio Web

El ejemplo permite crear elementos y clonar elementos HTML.

1. Localice el archivo ejemplo4.js y comience a definir las siguientes referencias a elementos:

```
// Inicializando referencia de botones con metodos actuales
const buttonAgregarPagina = document.querySelector("#idAgregarPagina");
const buttonMenu = document.querySelector("#idAgregarMenu");
const buttonTitulo = document.querySelector("#idAgregarTitulo");
const buttonParrafo = document.querySelector("#idAgregarParrafo");

const pagina = document.querySelector("#idPagina");
```

2. Defina la siguiente función para crear el contenedor de la nueva página web.

```
buttonAgregarPagina.onclick = function () {
  const contenedorVerificando = document.querySelector("#idDivPage");

  if (!contenedorVerificando) {
    //Creando el contenedor de la pagina
    const contenedor = document.createElement("div");
    contenedor.setAttribute("id", "idDivPage");
    contenedor.setAttribute("class", "container");
    contenedor.setAttribute(
      "style",
      "border:solid 1px black; height:500px; overflow: scroll; overflow-x: hidden;"
    );

    pagina.appendChild(contenedor);
  } else {
    alert("Ya se agrego el contenedor de la pagina");
  }
};
```

3. Definamos la siguiente función para crear el menú de la página. Aquí se estará clonando el menú de la página principal.

```
buttonMenu.onclick = function () {
  //verificando que exista el contenedor de la pagina
  const contenedor = document.querySelector("#idDivPage");

  if (contenedor) {
    // Verificando que exista el menu
    const menuVerificar = document.querySelectorAll("#idDivPage > header");

    if (menuVerificar.length == 0) {
      // Clonando el menu principal de nuestra pagina
      // Para luego crearlo en la nueva pagina
      const menu = document.querySelector("header").cloneNode(true);
      contenedor.appendChild(menu);
    } else {
      alert("Ya ha sido agregado el menu");
    }
  } else {
    alert("Primero debe agregar un contendor de pagina");
  }
};
```

4. Definamos la siguiente función para crear títulos en nuestra página web.

```
buttonTitulo.onclick = function () {
  //verificando que exista el contenedor de la pagina
  const contenedor = document.querySelector("#idDivPage");

  // Verificando que exista el menu
  const menu = document.querySelectorAll("#idDivPage > header");

  if (contenedor) {
    if (menu.length > 0) {
      let titulo = prompt("Agregue el titulo de la pagina");

      if (titulo != "" && titulo != null) {
        const h1 = document.createElement("h1");
        // Agregando clases de Bootstrap
        h1.setAttribute("class", "display-5 text-center fw-bold py-4 my-4");
        h1.innerHTML = titulo;

        contenedor.appendChild(h1);
      } else {
        alert(
          "No se ha registrado ningun titulo, por favor ingrese informacion"
        );
      }
    } else {
      alert("Debe agregar un menu primero");
    }
  } else {
    alert("Primero debe agregar un contendor de pagina");
  }
};
```

5. Definamos la siguiente función para crear párrafos en nuestra página web.

```
buttonParrafo.onclick = function () {
  //verificando que exista el contenedor de la pagina
  const contenedor = document.querySelector("#idDivPage");

  // Verificando que exista el menu
  const menu = document.querySelectorAll("#idDivPage > header");

  if (contenedor) {
    if (menu.length > 0) {
      let texto = prompt("Agregue un parrafo a su pagina web");

      if (texto != "" && texto != null) {
        const parrafo = document.createElement("p");
        // Agregando clases de Bootstrap
        parrafo.setAttribute("class", "lead mb-4 py-4");
        parrafo.innerHTML = texto;
        // Creando parrafo como hijo del contenedor
        contenedor.appendChild(parrafo);
      } else {
        alert(
          "No se ha registrado ningun parrafo, por favor ingrese informacion"
        );
      }
    } else {
      alert("Debe agregar un menu primero");
    }
  } else {
    alert("Primero debe agregar un contendor de pagina");
  }
};
```

6. Verifique el funcionamiento de su página ejemplo4.html, tendría que obtener un resultado como el siguiente.

IV. EJERCICIOS COMPLEMENTARIOS

1. Modifique el ejemplo1.html para que realice las siguientes acciones:
    a. Valide que el ID de los controles nuevos no se repita, muestre un mensaje adecuado al usuario haciéndole saber que no es permitido controles con el mismo ID.
    b. Cree un botón que permite validar la información de los nuevos controles agregados al formulario. Esta validación solamente incluirá campos llenos y opciones seleccionadas (radio, checkbox y select).
    c. Adicione la creación de tipos de elementos <input type="color"> y <input type="email">

2. Modifique el ejemplo2.html para que pueda validar la información del formulario, considere las siguientes validaciones:
    a. Valide que los campos no estén vacíos.
    b. Valide que la fecha de nacimiento no supere la fecha actual.
    c. Utilice expresiones regulares para validar el campo correo electrónico.
    d. Valide que los campos contraseña y repetir contraseña, sean iguales.
    e. Verifique que debe estar seleccionada al menos una opción para "algunos intereses".
    f. Verifique que el usuario seleccione una carrera.
    g. Verifique que sea seleccionado un país de origen.

    Una vez que se valide la información, proceda a mostrarla en un modal de bootstrap, utilice funciones de DOM para crear una tabla acorde a la información capturada. No se permite utilizar innerHTML, utilice las funciones mostradas en la parte teórica de esta guía.