UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Diplomatura en programación web full stack con React JS

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 2: JavaScript

Unidad 2: Integración JS y HTML

Presentación:

En la unidad anterior vimos la sintaxis de JavaScript y cómo ejecutar un programa utilizando la consola del navegador. Sin embargo, como dijimos en la introducción, JavaScript se puede utilizar para modificar el HTML y en esta unidad vamos a aprender a hacer justamente esto!

Objetivos:

Que los participantes:

- Comprendan los fundamentos del DOM.
- Sean capaces de manipular el DOM desde JavaScript.

Bloques temáticos:

1. Integración JS y HTML
2. Introducción y manejo del DOM
3. Selectores
4. Propiedades más utilizadas
5. Eventos más utilizados
6. Ejemplos
7. Trabajo Práctico

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

** El MEC es el modelo de E-learning colaborativo de nuestro Centro.*

Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Integración JS y HTML

Algo adelantamos en el ejemplo de la unidad anterior respecto a la forma de integrar un script js en HTML. Sin embargo vamos a verlo aquí con un poco más de detalle y vamos a sumar otra alternativa más.

Al igual que CSS es posible incluir el código JS es el mismo archivo HTML (aunque no es lo más aconsejable) y también incluir un archivo conteniendo el código js enlanzándolo desde la página html.

Algo de suma importancia es que tanto el código js (en caso de incluirlo en el html) como el enlace al archivo js deben estar dentro del body pero como última línea (del body). Esto es vital para que la ejecución del código JS se efectúe correctamente y en la próxima sección vamos a saber por qué esto es así. Por ahora vamos a adelantar que está relacionado con la forma de ejecución del código por parte del navegador.

Ejemplo 1: incluímos js en el mismo html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h1>Mi primer pagina con Js incluido</h1>
    <script>
        var vector = [22,25, 60, 98, 11, 78, 4, 33, 85, 10];
        var mayor = vector[0];
        var posicionMayor = 0;
        var menor = vector[0];
        var posicionMenor = 0;

        for(let i = 0; i < 10; i++) {
            if(mayor < vector[i]) {
                mayor = vector[i];
                posicionMayor = i;
            }
            if(menor > vector[i]) {
                menor = vector[i];
                posicionMenor = i;
            }
        }
        console.log("El mayor es " + mayor + " y se encuentra en la posicion " + posicionMayor);
        console.log("El menor es " + menor + " y se encuentra en ela posicion " + posicionMenor)
    </script>
</body>
</html>
```

Ejemplo 2: incluimos el enlace a un archivo js

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <script src="mayorMenorVector.js"></script>
</body>
</html>
```

2. Introducción y manejo del DOM

DOM = Document Object Model

Una página HTML normal es una sucesión de caracteres, esto lo hace muy difícil de manipular.

Los navegadores web transforman automáticamente las páginas HTML en una estructura más eficiente de manipular: DOM

Para que? Para poder modificar con JavaScript el HTML!!!!

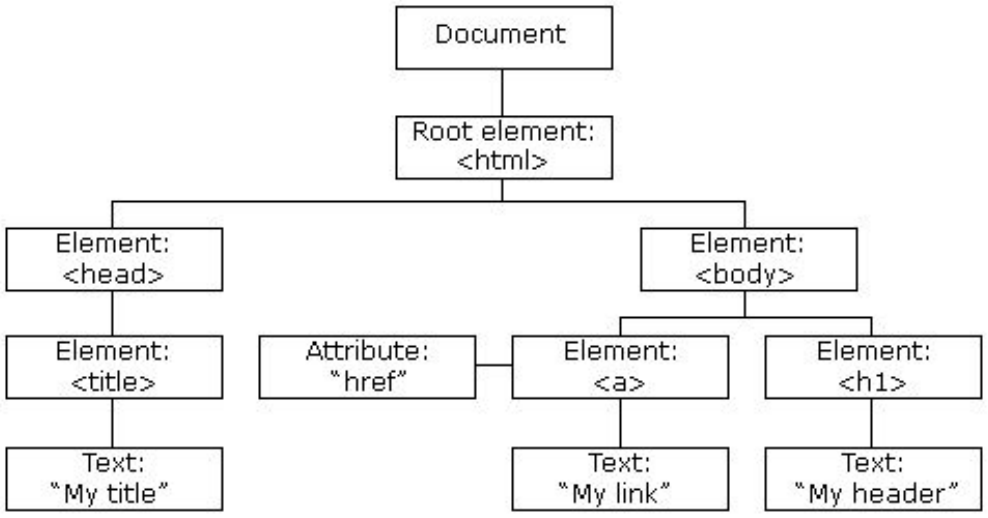
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
D/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Página sencilla</title>
</head>

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

DOM transforma todos los documentos en un conjunto de elementos llamados ***nodos***.

Los nodos están interconectados y representan los contenidos de la página web y la relación entre ellos.

Por su aspecto, la unión de todos los nodos se llama "***árbol de nodos***"



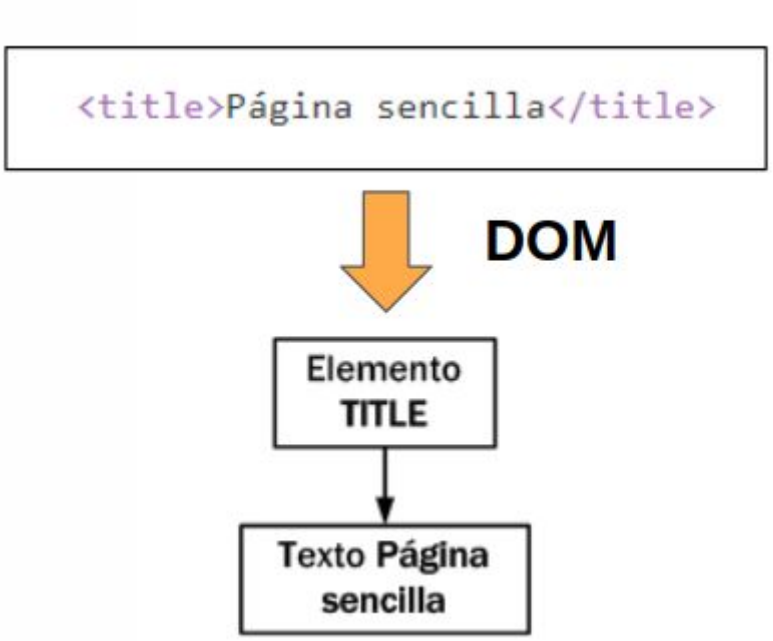
Ejemplo

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
D/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Página sencilla</title>
</head>

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

La **transformación** de cualquier **etiqueta** HTML genera 2 nodos:

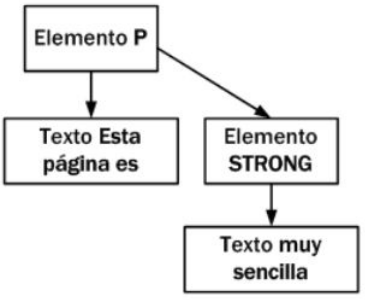
Primero **nodo** de tipo ***Elemento***
Segundo **nodo** de tipo ***Texto***



DOM - Document Object Model

Ejemplo etiquetas anidadas

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



Reglas de generación:

- Las etiquetas se transforman en 2 nodos: uno con la propia etiqueta y otro (hijo del anterior) con el contenido textual de la etiqueta
- Si una etiqueta se encuentra dentro de otra, idem anterior pero los nodos generados serán nodos hijos de su etiqueta padre

Tipos de nodos

La especificación completa de DOM define 12 tipos de nodos pero habitualmente se pueden manejar solamente 4 o 5 tipos de nodos:

- **Document,** nodo raíz del que derivan todos
- **Element**, cada una de las etiquetas. Único nodo que puede contener atributos y del que pueden derivar otros nodos.
- **Attr**, se define uno para cada par atributo=valor
- **Text**, contiene el texto encerrado por una etiqueta
- **Comment**, representa los comentarios incluidos en la página.

3. Selectores

Acceso directo a los nodos

Hay 2 métodos para acceder a un nodo específico: acceso a través de sus nodos padre y acceso directo.

El acceso a los nodos, su modificación y su eliminación sólo es posible cuando el árbol DOM ha sido construido completamente, es decir, después de que la página se cargue por completo.

Hay 3 funciones para acceder directamente a un nodo:

- getElementsByTagName(nombreEtiqueta)
- getElementsByClassName(nombreAtributo)
- getElementById(id)

Crear un nodo

Recordar que cada elemento genera 2 nodos.

4 pasos:

1. Creación de un nodo de tipo Element que represente al elemento.
2. Creación de un nodo de tipo Text que represente el contenido del elemento.
3. Añadir el nodo Text como nodo hijo del nodo Element.
4. Añadir el nodo Element a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

Implica utilizar 3 funciones DOM:

- **createElement(etiqueta)**: crea un nodo de tipo **Element** que representa al elemento cuya etiqueta se pasa como parámetro.
- **createTextNode(contenido)**: crea un nodo de tipo **Text** que almacena el contenido textual de los elementos.
- **nodoPadre.appendChild(nodoHijo)**: añade un nodo como hijo de otro nodo. Se debe utilizar al menos dos veces con los nodos habituales: en primer lugar se añade el nodo **Text** como hijo del nodo **Element** y a continuación se añade el nodo **Element** como hijo de algún nodo de la página.

Ejemplo para la creación de un párrafo

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");

// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola Mundo!");

// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);

// Añadir el nodo Element como hijo de la página
document.body.appendChild(parrafo);
```

Eliminar un nodo

Usar la función **removeChild(nodo)** que debe ser **invocada desde** el elemento **padre**.

Para **acceder al padre** de un nodo: **nodoHijo.parentNode**

```
var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);

<p id="provisional">...</p>
```

PROPIEDADES DE UN NODO

Para acceder al valor se indica el nombre del atributo HTML detrás del nombre del nodo.

Ejemplo:

```
var enlace = document.getElementById("enlace");
alert(enlace.href); // muestra http://www...com

<a id="enlace" href="http://www...com">Enlace</a>
```

4. Propiedades más utilizadas

Propiedades de un nodo

Para propiedades **CSS** se accede mediante el atributo *style*.

Ejemplo:

```
var imagen = document.getElementById("imagen");
alert(imagen.style.margin);

<img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

El resultado puede ser un poco diferente en los distintos navegadores

Para propiedades **CSS** con nombre compuesto, se accede eliminando los guiones (-).

- font-weight → fontWeight
- line-height → lineHeight
- border-top-style → borderTopStyle
- list-style-image → listStyleImage

Ejemplo:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.style.fontWeight);  // muestra "bold"

<p id="parrafo" style="font-weight: bold;">...</p>
```

Atributo HTML ***class*** se accede de forma diferente porque “class” es palabra reservada en JavaScript.

DOM utiliza **className** para referiste a ***class*** de HTML

Ejemplo:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.class); // muestra "undefined"
alert(parrafo.className); // muestra "normal"

<p id="parrafo" class="normal">...</p>
```

QuerySelector

Permite recorrer el documento de una forma más fácil que las opciones anteriores, pudiendo utilizar los selectores CSS.

document.querySelector(‘<selector CSS>’);

Retorna solo la primer ocurrencia

document.querySelector(‘.menu’);

document.querySelector(‘#id-principal .clase-secundaria’);

QuerySelectorAll

Retorna todas las ocurrencias

document.querySelectorAll(‘<selector CSS>’);

Class

Modificar el class desde JS

Cambio de las clases de un elemento (respuesta de getElementById, querySelector, etc)

elemento.className = ‘nueva-clase-css’;

Agregar class

Agregar una clase CSS a un elemento

elemento.classList.add(‘nueva-clase-css’);

Eliminar un class

Eliminar una clase CSS a un elemento

elemento.classList.remove(‘clase-css-a-borrar’);

5. Eventos más utilizados

Eventos

Cuando utilizamos JavaScript en el navegador disponemos de una serie de eventos a los cuales responder. Los eventos son aquellas interacciones que tiene el usuario con nuestra página web y a las cuales nosotros podemos responder. Por ejemplo, cuando el usuario hace click sobre un botón, se genera un evento al cual podemos responder con un mensaje o cualquier código que nosotros seleccionemos.

EVENTOS

Eventos disponibles

Los eventos que tenemos disponibles para interactuar con el usuario son los siguientes:

- onClick: se produce cuando el usuario hace click sobre una etiqueta HTML (por ejemplo un botón)
- onChange: se produce cada vez que el usuario cambia el contenido de una etiqueta del tipo input (y abandona el campo de entrada)
- onFocus: se produce cada vez que el usuario ingresa a una etiqueta del tipo input
- onSubmit: se produce cuando el usuario envía un formulario
- onScroll: que se produce cada vez que el usuario se desplaza en la página (siempre y cuando exista un desplazamiento de la barra de scroll lateral)

OnClick

Ejemplo

```
<script>
      function presionoBoton() {
            alert('Presiono botón');
      }
</script>
<body>
...
    <button onClick="presionoBoton()">Botón</button>
...
</body>
```

Cuando el usuario hace click sobre Botón, se muestra un alerta con el mensaje “Presionó botón”.

OnChange

Ejemplo

```
<script>
	function cambioInput() {
		console.log("Cambio el valor");
	}
</script>
<body>
...
    <input type="text" onChange="cambioValor()">
...
</body>
```

OnFocus

Ejemplo

```
<script>
	function accedioAlElemento() {
		console.log("Accedió al elemento");
	}
</script>
<body>
...
    <input type="text" onFocus="accedioAlElemento()">
...
</body>
```

OnSubmit

Ejemplo

```
    <script>
        function envioFormulario() {
            console.log('Envio formulario');
        }
    </script>
<body>
    ...
    <form onsubmit="envioFormulario()">
        ...
    </form>
    ...
</body>
```

Parámetros a los eventos

Cuando se produce un evento, también tenemos la posibilidad de acceder al contexto del mismo (en donde se ejecutó el mismo). Por ejemplo:

- El input en el cual se produjo el evento, y al valor
- El formulario en el cual se produjo el submit
- El botón que fue presionado

Estas operaciones principalmente se realizan sobre el evento onChange, cuando queremos acceder al valor que el usuario introdujo en el campo de entrada (input). Veamos un ejemplo

OnChange

Ejemplo

```
<script>
      function cambioInput(e) {
         console.log("Cambio el valor del input y ahora es " + e.target.value);
      }
</script>
<body>
...
    <input type="text" onChange="cambioValor(event)">
...
</body>
```

Ahora cuando el usuario cambia el valor del campo de entrada, y abandona el mismo, se ejecuta el método **onChange**, el cual llama a la función **cambioInput**. La función, en el parámetro **e**, recibe la información del contexto en el cual se ejecutó el evento, siendo **e.target** el campo que originó el evento. Para acceder a la propiedad value (el valor que tiene el input) podemos utilizar **e.target.value**

OnSubmit y preventDefault()

Ejemplo

```
    <script>
        function envioFormulario(e) {
            console.log('Envio formulario');
            e.preventDefault();

        }
    </script>
</head>
<body>
    <form onsubmit="envioFormulario(event)">
        <input type="text">
        <button type="submit">Enviar</button>
    </form>
```

De manera predeterminada, luego de ejecutarse el código de la función **onSubmit** se envía el formulario al servidor. Pero en varias oportunidades deseamos que el formulario no se envíe o que se envíe solo si pasa determinadas validaciones. Para prevenir que el formulario sea enviado al servidor, se hace uso de **preventDefault()**. Este evento, no continúa el circuito normal de enviar la información al servidor, sino que evita ese paso.

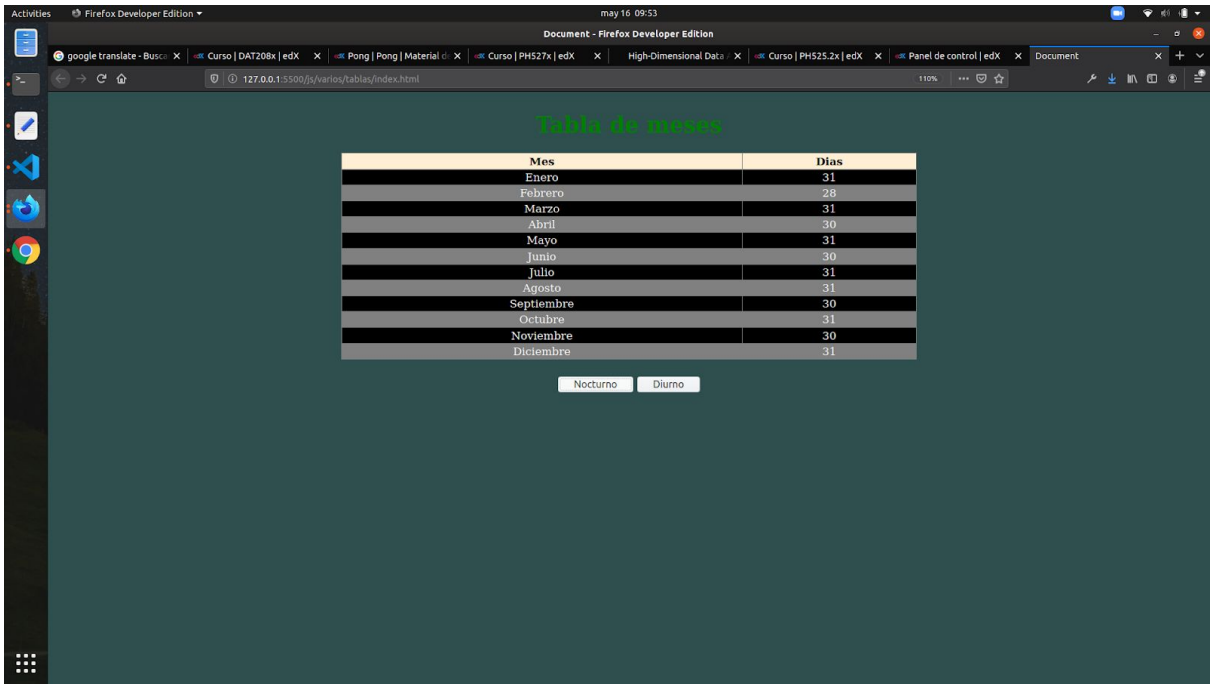
6. Ejemplos

Desarrollamos un programa que muestre una tabla con los meses del año y los días que tiene cada mes. Con 2 botones, vamos a cambiar el estilo de la tabla.

El efecto deseado es el siguiente:

- Al cargar la página se debe mostrar esta pantalla

- Al hacer clic en el botón “Nocturno” debe cambiar el estilo según la imagen que sigue



Al hacer clic en el botón “Diurno” se muestran los estilos como sigue

El html es el siguiente

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>
<body onload="dibujar()">
    <h1>Tabla de meses</h1>
    <table>
        <tr>
            <th>Mes</th>
            <th>Dias</th>
        </tr>
        <tr>
            <td>Enero</td>
            <td>31</td>
        </tr>
        <tr>
            <td>Febrero</td>
            <td>28</td>
        </tr>
        <tr>
            <td>Marzo</td>
            <td>31</td>
        </tr>
        <tr>
            <td>Abril</td>
            <td>30</td>
        </tr>
```

```
33          <tr>
34              <td>Mayo</td>
35              <td>31</td>
36          </tr>
37          <tr>
38              <td>Junio</td>
39              <td>30</td>
40          </tr>
41          <tr>
42              <td>Julio</td>
43              <td>31</td>
44          </tr>
45          <tr>
46              <td>Agosto</td>
47              <td>31</td>
48          </tr>
49          <tr>
50              <td>Septiembre</td>
51              <td>30</td>
52          </tr>
53          <tr>
54              <td>Octubre</td>
55              <td>31</td>
56          </tr>
57          <tr>
58              <td>Noviembre</td>
59              <td>30</td>
60          </tr>
61          <tr>
62              <td>Diciembre</td>
63              <td>31</td>
64          </tr>
```

```
61          <tr>
62              <td>Diciembre</td>
63              <td>31</td>
64          </tr>
65      </table>
66      <div class="botonera">
67          <button onclick="modoNocturno()">Nocturno</button>
68          <button onclick="modoDiurno()">Diurno</button>
69      </div>
70      <script src="script.js"></script>
71  </body>
72  </html>
```

El archivo css tiene

```css
body {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
}

table {
    border-collapse: collapse;
    width: 50%;
}

table, th, td {
    border: 1px solid gray;
    text-align: center;

}

th {
    background-color: papayawhip;
}


.botonera {
    margin-top: 25px;
}
```

Finalmente el js

```
var table = document.getElementsByTagName("tr");
var titulo = document.getElementsByTagName("h1");

function dibujar(){
    i =0;

    titulo[0].style.color="green";

    var recorrido = setInterval(() => {
            if(i < table.length){
                table[i].style.display="table-row"
                i++;
            }
            else {
                clearInterval(recorrido);
            }
    }, 100);

}
```

```
21  function modoDiurno(){
22
23      document.body.style.backgroundColor =  "lightslategrey";
24
25      for(let i = 1; i < table.length; i++){
26          table[i].style.color = "black";
27
28          if (i%2 == 0) {
29              table[i].style.backgroundColor = "palevioletred";
30          }
31          else {
32              table[i].style.backgroundColor = "pink";
33          }
34      }
35  }
36
37
38  function modoNocturno(){
39
40      document.body.style.backgroundColor =  "darkslategrey";
41
42      for(let i = 1; i < table.length; i++){
43          table[i].style.color = "white";
44
45          if (i%2 == 0) {
46              table[i].style.backgroundColor = "grey";
47          }
48          else {
49              table[i].style.backgroundColor = "black";
50          }
51      }
52  }
```

7. Trabajo Práctico

Teniendo la siguiente estructura:

```
var productos = [
    {
        nombre: "harina",
        precio: 35
    },
    {
        nombre: "pan",
        precio: 25
    },
    {
        nombre: "papa",
        precio: 52
    },
    {
        nombre: "palta",
        precio: 55
    },
    {
        nombre: "fideos",
        precio: 85
    },
    {
        nombre: "aceite",
```

```
        precio: 350
    },
    {
        nombre: "sopa",
        precio: 86
    },
    {
        nombre: "mermelada",
        precio: 108
    },
    {
        nombre: "porotos",
        precio: 69
    },
    {
        nombre: "lentejas",
        precio: 85
    },
    {
        nombre: "mandarina",
        precio: 43
    },
    {
        nombre: "banana",
        precio: 79
    },
```

```
{
    nombre: "leche de almendras",
    precio: 145
},
{
    nombre: "papel higiénico",
    precio: 147
},
{
    nombre: "lavandina",
    precio: 55
},
{
    nombre: "alcohol en gel",
    precio: 123
},
{
    nombre: "shampoo",
    precio: 400
},
{
    nombre: "arroz",
    precio: 66
},
{
    nombre: "harina",
```

```
        precio: 35
    },
    {
        nombre: "salsa de tomate",
        precio: 35
    },
];
```

Como se puede apreciar, se trata de un vector con 20 posiciones. Cada posición tiene un objeto que posee nombre y precio. Para acceder a cada uno de ellos se utiliza la siguiente notación:

productos[<numero_de_posicion>].nombre para acceder al nombre

productos[<numero_de_posicion>].precio para acceder al precio.

Ejemplo:

```
for(let i = 0; i <20 ; i++) {
        console.log(productos[i].nombre);
        console.log(productos[i].precio);
}
```

Desarrollar un “carrito de compras” donde el usuario presione sobre cada producto y el mismo quede guardado en el carrito. Luego, al oprimir el botón “Comprar”, calcular el importe final y mostrar los productos comprados junto con el total a pagar. Solo puede comprar una unidad de cada producto.

Bibliografía utilizada y sugerida

MDN - JavaScript. (n.d.) Recuperado de: https://developer.mozilla.org/es/docs/Web/JavaScript

Udemy. (n.d.) Recuperado de: https://udemy.com

Wikipedia - JavaScript. (n.d.) Recuperado de https://es.wikipedia.org/wiki/JavaScript

w3schools.com - JavaScript. (n.d.) Recuperado de https://www.w3schools.com/js/

Lo que vimos:

- Integración entre JS y HTML.
- Manejo del DOM.

Lo que viene:

- Asincronismo.