

# **INTRODUCCIÓN A LA PROGRAMACIÓN CON JAVASCRIPT – PLANIFICACIÓN y CONTENIDOS**

El presente documento es una guía de temas, clases, contenidos, tiempos y orden de dictado de los conceptos teóricos y prácticos; con el objetivo de educar e introducir al alumno del presente módulo en el amplio mundo de la programación con JavaScript; intentando dar mayor prioridad a los temas que tiene mayor uso en el campo real.

Los módulos tienen una duración de 1 (una hora) reloj, teniendo dos módulos continuos por clase con un descanso entre módulo y módulo de 10 minutos.

Haciendo un resumen de clases, temas el siguiente esquema dará una idea conceptual de la forma en la que se darán los contenidos.

A continuación, describiremos clase por clase los temas y posteriormente cada tema tendrá su desarrollo completo y cada tema tendrá un link de descarga de la clase correspondiente; de esta forma tendremos un documento principal como columna vertebral de todos los temas a brindar en clases.

## **CLASE NRO. 01: CONCEPTOS PREVIOS + INTRODUCCIÓN A LA PROGRAMACIÓN**

- Introducción a la programación.
- Que es la programación.
- Lenguajes de programación.
- Tipos de lenguajes de programación
- Componentes básicos de un programa
  - . variables, tipos de datos
  - . estructuras de control
  - . estructuras condicionales
  - . estructuras repetitivas
- Depuración y manejo de errores
- Introducción al protocolo HTTP y HTTPs
- Esquema de Petición / Respuesta
- Diferencias entre conceptos
  - . Página Web
  - . Sitio Web
  - . Aplicación Web
- Arquitectura de una aplicación Web
- Patrón de diseño Modelo Vista Controlador

### CLASE NRO. 02: PATRON MODELO VISTA CONTROLADOR

- creación de primer proyecto con JavaScript
- creación del documento HTML
- incorporación de hoja de estilos
- incorporación del controlador, código JavaScript
- incorporamos las primeras sentencias
  - . alert()
  - . console.log()
  - . prompt()
  - . confirm()
  - . console.error()
  - . console.warn()
  - . realizar líneas de comentarios

### CLASE NRO. 03: TIPOS DE DATOS – OPERACIONES CON VARIABLES

- que es una variable?.
- formas de declarar una variable
- particularidades al momento de asignarle un nombre
- diferentes tipos de Variables
  - . tipo numérico
  - . tipo string
  - . tipo boolean
  - . valores nulos
  - . valores indefinidos
- Bloques de código {}
- JavaScript lenguaje débilmente tipado
- operaciones entre variables
  - . suma
  - . resta
  - . multiplicación
  - . división
  - . suma de cadenas
  - . interpolación de cadenas \${variable}

### CLASE NRO. 04: ESTRUCTURAS CONDICIONALES Y COMPARACIONES LOGICAS

- Creación del documento HTML
- Incorporación del código JavaScript como controlador de las interacciones del usuario
- Declaración de variables desde el teclado a través de prompt
- Inicialización de variables
- Realizar operaciones numéricas simples entre las variables numéricas
- Concatenación y/o interpolación de valores strings.
- Mostrar valores por consola.
- Visualizar el problema de dividir por cero

## DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT

### MÓDULO 02 - JAVASCRIPT

- Visualizar el problema de realizar una operación con una variable no inicializada
- Visualizar en el archivo controlador.js cuando escribo o trato de realizar o mostrar por consola una variable que no existe.
- Respetar la indentación de código.
- Introducción a las estructuras condicionales
- Las operaciones de comparación lógicas básicas >, <, =, >=, <=
  - . A > B => se lee A es mayor a B
  - . A < B => se lee A es menor a B
  - . A = B => se lee A es igual a B
  - . A >= B => se lee A es mayor o igual que B
  - . A <= B => se lee A es menor o igual que B
- Introducción a las estructuras condicionales (lado verdadero, lado falso)

#### **CLASE NRO. 05: ERRORES TÍPICOS Y TRY/CATCH**

- No inicializar variables
- Utilizar el "==" en comparaciones en lugar de utilizar el "==="
- Utilizar una variable no inicializada dentro de un condicional.
- Olvidarse de cerrar paréntesis (), corchetes [], llaves {} en lugares donde son obligatorios.
- Ser desprolijo en la estructura del programa, no respetar la IDENTACIÓN.
- Olvidarse de colocar comillas dobles o comillas simples, en algún mensaje que sea por consola por ejemplo
- Introducción a la sentencia try/catch

#### **CLASE NRO. 06: ESTRUCTURAS REPETITIVAS**

- Estructuras repetitivas
  - . Ciclo for
  - . Ciclo while
  - . Ciclo do while
- Declaración e inicialización de variables
- Estructuras condicionales if/else dentro de ciclos
- Estructuras condicionales anidadas
- Contadores
- Acumuladores
- Cálculo de porcentajes sobre el total
- Salir de estructuras repetitivas con break
- Saltar una iteración en estructuras repetitivas con continue

### CLASE NRO. 07: FUNCIONES

- Que es una función => sinónimos (subprocesos, subrutinas, procedimientos, métodos).
- Definición ó declaración de una función
  - . Palabra function
  - . Parámetros ó argumentos
  - . Cuerpo de la función
  - . Retorno de la función
- Funciones que devuelven valor
- Invocación de Funciones.
- Mostrar como las funciones pueden participar dentro de expresiones matemáticas
- Funciones que no devuelven valor => procedimientos.
- Funciones y bibliotecas de funciones estándar
- Funciones como constantes => Expresión de función
- Funciones Flecha ó arrow functions
  - . Forma de declaración
  - . Omitir paréntesis, llaves y la palabra return cuando se recibe un solo parámetro de entrada. Ejemplo.
- Funciones que reciben como parámetro otras funciones.

### CLASE NRO. 08: MANEJO DEL DOM

- Introducción y explicación del DOM
- Estructura de árbol del DOM y sus ELEMENTOS
  - . Elementos
  - . Atributos
  - . Texto
  - . Comentarios
- Interacción con el DOM
  - Los selectores por ID y Nombre
  - . getElementById
  - . getElementsByClassName
  - . getElementsByTagName
  - . querySelector("selector")
  - . querySelectorAll("selector")
- Modificación de Elementos del DOM
  - innerHTML
  - textContent
- Eventos Principales en una página WEB
  - . window.onload
  - . click
  - . Incorporamos dos botones y capturamos el evento click en dos botones
  - . Incorporamos el Evento click al documento y observamos que se produce un efecto de propagación de a dentro hacia a fuera.
  - . stopPropagation()

#### CLASE NRO. 09: MANEJO DEL DOM

- Manejo del DOM
- integración HTML, CSS y JavaScript
- declaración de variables y constantes
- captura de elementos / objetos de la interfaz con getElementById
  - . Mostramos cuando una variable o constante es nula
  - . Mostraremos cuando una variable es undefined
- captura de valores de esos elementos
- declaración de funciones
- invocación de funciones
- captura de los eventos que surgen en la pantalla
- interpolación de string para mostrar resultados por consola

#### CLASE NRO. 10: MANEJO DEL DOM

- Manejo del DOM
- integración HTML, CSS y JavaScript
- declaración de variables y constantes
- captura de elementos / objetos de la interfaz con getElementById
  - . Mostramos cuando una variable o constante es nula
  - . Mostraremos cuando una variable es undefined
- captura de valores de esos elementos
- declaración de funciones
- invocación de funciones
- captura de los eventos que surgen en la pantalla

interpolación de string para mostrar resultados por consola

#### CLASE NRO. 11: ARRAYS / VECTORES

- Declarar un vector literal
- Mostrar el contenido de un vector
- Acceder a una posición del vector y modificarlo
- Agregar al final push()
- Eliminar el último valor y devolverlo pop()
- Eliminar el primer elemento del vector y devolverlo shift()
- Agregar al principio unshift()
- Eliminar elementos contiguos splice()
- Devolver la ubicación de un elemento indexOf() y lastIndexOf(). Uno retorna la primera ubicación y el otro devuelve la última posición.
- Devolver un elemento y la ubicación, pero con criterios más flexibles y definidos por el usuario como find y findIndex.
- Filtrar y seleccionar y devolver un conjunto de elementos del vector que cumplan con una condición específica. Filter
  - Forma 1: con una arrow function que no tiene la palabra los () para el parámetro, no tiene las llaves {} del cuerpo y no lleva la palabra return;

- Forma 2: con una arrow function completa que tiene los () y la palabra return y las llaves del cuerpo {}
- Forma 3: con una función anónima clásica definida dentro del parámetro de filter
- Forma 4: con una función declarada previamente y pasarle como parámetro la función dentro del filter.
- Recorrer tradicionalmente un vector con un ciclo for
- Recorre un vector con vector.forEach()
- Transformación de los elementos de un vector con vector.map() recorre los elementos, aplica la función pasada como parámetro y devuelve cada elemento transformado, a todos ellos los devuelve y los agrega en un vector de salida.
- Funciones reductoras con reduce(). Que permiten acumular valores, encontrar el mayor, el menor, etc.
- Función de ordenamiento sort() con las particularidades cuando son valores string o cuando son valores numéricos.
- Particularidad de copiar variables o constantes que contienen vectores. Es decir, se crean referencias a los elementos y no se realiza una copia del elemento. Dado este problema surge la necesidad de hacer una clonación de esos elementos y sale spreadOperator también se puede usar slice() pero es más completo y flexible spread. El problema de la copia por referencia ocurre con Vectores y con Objetos.
- spreadOperator ...

## CLASE NRO. 12: POO – PROGRAMACIÓN ORIENTADA A OBJETOS

- Introducción al Concepto de la POO
- Nueva estructura de datos
- Las clases, objetos, atributos, constructor, métodos get y set
- El constructor
- Métodos de una clase
  - públicos
  - estáticos
  - privados
- Los Objetos
- Abstracción
- Encapsulación
- Roles de un programador (Constructor de clases ó Consumidor de clases)
- Objetos literales
- JSON => JavaScript Object Notation
- **Destructuring**

### **CLASE NRO. 13: MÓDULOS - BIBLIOTECAS**

- que son los módulos?
- exportación / importación de elementos primitivos anónimos (variables / constantes)
- exportación / importación de vectores
- exportación / importación de funciones anónimas
- exportación / importación funciones tradicionales
- exportación / importación de una clase anónima
- exportación / importación de objetos
- exportación / importación de un botón
- exportación / importación de un fragmento
- exportaciones / importación por defecto
- exportaciones / importación de elementos con nombre
- importar todo un módulo completo

### **CLASE NRO. 14: MANIPULACIÓN DINÁMICA DEL DOM**

- que es la manipulación dinámica del DOM
- tres técnicas de manipulación dinámica del DOM
  - o con métodos propios del DOM como createElement
  - o con innerHTML
  - o con Template
- práctica intensiva
- generación de componentes
- módulos que almacenarán funciones que generan componentes
- ejercicios varios

### **CLASE NRO. 15: FUNCIONES. COMO ENTIDADES DE PRIMERA CLASE**

- Ser asignadas a una variable ó constante.
- Ser pasadas como argumentos a funciones.
- Ser devueltas como resultado de funciones.
- Ser almacenadas en estructuras de datos.
- Particularidades que tenemos con las funciones
  - pueden declararse de forma clásica
  - pueden guardarse dentro de variables o constantes
  - pueden declararse como arrow functions y guardarlas en constantes. Expresiones
  - En algunos casos, las arrow functions pueden no llevar (), {} y la palabra return
  - Funciones pueden ser pasadas como argumentos o parámetros de otra función
  - Funciones que retornan otras funciones
  - El Resultado de una función puede ser pasado a otra función
  - Pueden establecerse valores por defecto en los parámetros de entrada
  - Pueden ser almacenadas en cualquier estructura de datos
  - Funciones con Rest Parameters
  - Funciones con Spread Operator

## CLASE NRO. 16: ASINCRONISMO

- Que es el asincronismo.
  - Diferencias entre procesos síncronos y asíncronos
  - Event Loop
  - Manejar el asincronismo para conseguir un proceso síncrono
- Ejemplo 01: de dos procesos aislados y asíncronos

### Funciones callBack

Ejemplo 02: utilizando una función callback

Ejemplo 03: la función llamadora recibe como parámetro la próxima función a llamar

Ejemplo 04: la función callBack al momento de la invocación se declara de forma anónima como una arrow function

Ejemplo 05: en este quinto ejemplo la función llamadora pasa resultados a la callBack

### Promesas

Ejemplo 07: visualización de como instanciar el objeto Promise

Ejemplo 08: crear una función que retorne una promesa

Ejemplo 09: crear una función que retorne una promesa, pero la promesa se pasa como parámetro a la función

Ejemplo 10: En este ejemplo la función lanzadora anterior, se utiliza para enganchar dos promesas que se desean ejecutar de forma síncrona

### Async / await

## CLASE NRO. 17: APIs + FETCH + JSON + TRY/CATCH

## CLASE NRO. 18: ESTADO DE LA APLICACIÓN



**CLASE NRO. 19: PERSISTENCIA DE DATOS EN EL NAVEGADOR**

- Cookies
- Web Storage
  - Local Storage
  - Session Storage

## **MODULO 02 - JAVASCRIPT**

### **Documentación Recomendada:**

La documentación oficial y más completa de JavaScript es proporcionada por MDN Web Docs (anteriormente conocido como Mozilla Developer Network). MDN Web Docs es ampliamente reconocido como la mejor fuente de documentación para desarrolladores web, incluyendo información detallada sobre JavaScript, HTML, CSS, y otras tecnologías web.

Esta documentación cubre todos los aspectos de JavaScript, desde conceptos básicos hasta características avanzadas. Incluye tutoriales, referencias de API, guías, y ejemplos de código.

<https://developer.mozilla.org/es/docs/Web/JavaScript>

**Guía de JavaScript** - Cobertura desde los conceptos básicos hasta características avanzadas:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>

**Referencia de JavaScript** - Detalles sobre las construcciones del lenguaje, objetos predefinidos, métodos, y funciones:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>

**Tutoriales y recursos de aprendizaje:** Ejemplos prácticos y tutoriales para principiantes y desarrolladores avanzados.

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language\\_overview](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_overview)

### **Otras fuentes recomendadas:**

Recursos de lectura:

<https://web.dev/learn/javascript>

Página para practicar en línea:

<https://codepen.io/>

JavaScript en 30 días

<https://github.com/Asabeneh/30-Days-Of-JavaScript/>

## **CLASE NRO: 01 – CONCEPTOS PREVIOS + INTRODUCCIÓN A LA PROGRAMACIÓN**

**Tipo de Clase:** Teórica

**Duración:** 40 minutos

**Objetivo de la clase:**

El alumno al finalizar la clase debe entender claramente el protocolo HTTP y HTTPs a nivel conceptual, el ciclo de vida de la página web (desde que el cliente solicita un recurso/página web y el servidor responde la solicitud y luego, el navegador renderiza la página. Entendiendo que el código JavaScript comienza a interactuar con la página cuando esta está totalmente renderizada. Que es el evento window.onload

A todo esto, el alumno debe dominar conceptualmente la diferencia clara existente entre una página web, un sitio web, una aplicación web.

### **Introducción a la Programación:**

#### **1) ¿Qué es la Programación?**

La programación es el proceso de analizar, diseñar, construir, testear, probar y poner en funcionamiento un programa siendo este (el programa) un conjunto de instrucciones escritas es un lenguaje que la computadora puede entender, interpretar y ejecutar paso a paso siguiendo una lógica coherente con el objetivo de resolver un problema determinado. Este programa tiene un principio y un final. En nuestro caso el principio será programa estará escrito en este bloque {}

#### **Importancia de la Programación**

La programación es fundamental en la era digital. Permite automatizar tareas, desarrollar aplicaciones, analizar datos, y crear sistemas complejos que facilitan la vida diaria.

#### **2) Lenguajes de Programación**

Es un conjunto de instrucciones escritas en un lenguaje de programación que están ordenadas de forma lógica, que se ejecutan una a la vez y que en conjunto resuelven un problema determinado.

##### **Tipos de Lenguajes de Programación**

**Lenguajes de Alto Nivel:** Como JavaScript, Python, y Java, que son más fáciles de entender y escribir.

**Lenguajes de Bajo Nivel:** Como el lenguaje ensamblador, que están más cerca del lenguaje máquina y son más difíciles de entender para los humanos.

### **3) Componentes Básicos de un Programa**

Variables y Tipos de Datos  
Estructuras de Control  
Condicionales  
Bucles  
Funciones  
Estructuras de Datos (Vectores)

### **4) Programación orientada a objetos**

### **5) Depuración y manejo de errores**

**Depuración:** proceso de encontrar y corregir errores en el código

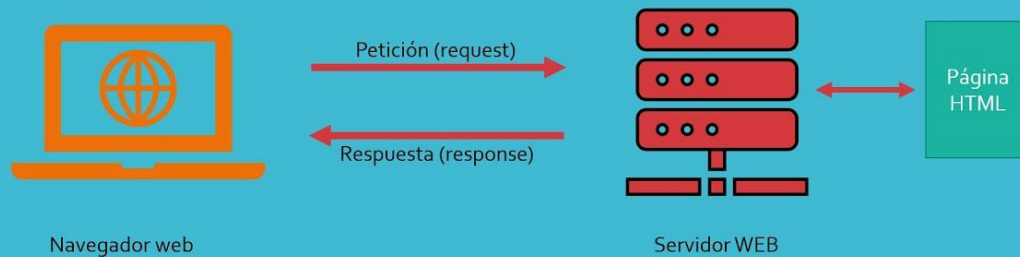
**Manejo de errores:** Permite gestionar errores de forma controlada sin detener la ejecución del programa.

## Introducción al protocolo HTTP y HTTPs.

- ✓ Inicio de la Solicitud: El ciclo de vida comienza cuando el usuario ingresa una URL en su navegador o hace clic en un enlace.
- ✓ Resolución DNS: El navegador traduce el nombre de dominio en la URL a una dirección IP utilizando el Sistema de Nombres de Dominio (DNS).
- ✓ Establecimiento de la Conexión: Se establece una conexión TCP (o TLS si se utiliza HTTPS) entre el cliente (navegador) y el servidor.
- ✓ Envío de la Solicitud: El navegador envía una solicitud HTTP al servidor web. Esta solicitud incluye la URL, los encabezados y, en algunos casos, datos adicionales como parámetros de consulta o contenido del formulario.
- ✓ Procesamiento de la Solicitud en el Servidor: El servidor web recibe la solicitud y la procesa. Esto implica la identificación del recurso solicitado, la ejecución de scripts si es necesario (por ejemplo, PHP, Python, etc.), y la recuperación de datos de la base de datos si es aplicable.
- ✓ Generación de la Respuesta: El servidor web genera una respuesta HTTP que incluye el código de estado (por ejemplo, 200 OK, 404 Not Found), los encabezados de respuesta y, opcionalmente, el contenido del recurso solicitado.
- ✓ Envío de la Respuesta: El servidor envía la respuesta al navegador a través de la conexión TCP/TLS establecida.
- ✓ Renderizado en el Navegador: El navegador recibe la respuesta y comienza a renderizar el contenido. Esto implica interpretar y renderizar el HTML, aplicar estilos CSS, ejecutar scripts JavaScript, y cargar recursos externos como imágenes, archivos CSS y JavaScript adicionales.
- ✓ Interacción del Usuario: Una vez que la página se ha renderizado completamente en el navegador, el usuario puede interactuar con ella haciendo clic en enlaces, enviando formularios, desplazándose por la página, etc.

# Petición / Respuesta

## (Request / Response)



**Protocolo HTTP (Video de 13 minutos) donde explica funcionamiento de HTTP**

**VIDEO:**

[Protocolo HTTP](#)

**Diferencias Fundamentales de los siguientes CONCEPTOS**

- Página WEB
- Sitio WEB
- Aplicación WEB

**Página Web:**

Una página web es un único documento HTML que puede contener texto, imágenes, videos, enlaces, y otros elementos multimedia. Es una sola unidad de contenido que se puede visualizar en un navegador web. Cada página web tiene una URL única.

Ejemplo: La página de inicio de Wikipedia es una página web (<https://es.wikipedia.org/wiki/Wikipedia:Portada>).

**Sitio Web:**

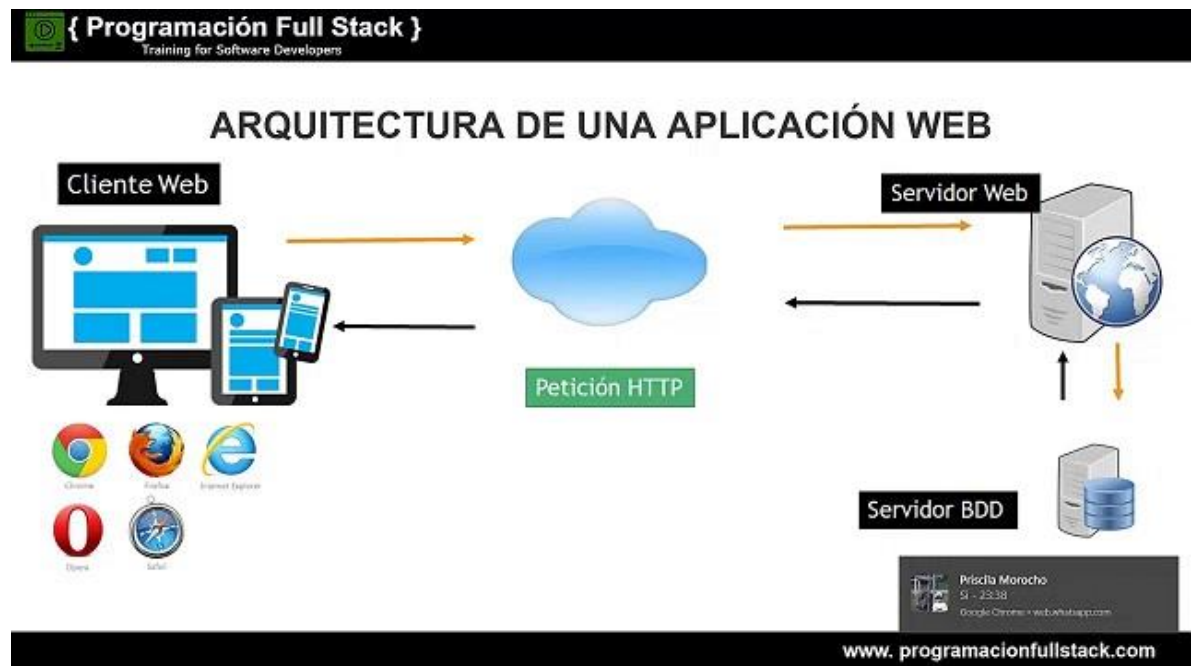
Un sitio web es un conjunto de páginas web relacionadas y conectadas entre sí mediante enlaces. Estas páginas comparten un nombre de dominio común y generalmente están organizadas en torno a un tema o propósito específico.

Ejemplo: Wikipedia (<https://www.wikipedia.org/>) es un sitio web que contiene numerosas páginas web sobre diferentes temas.

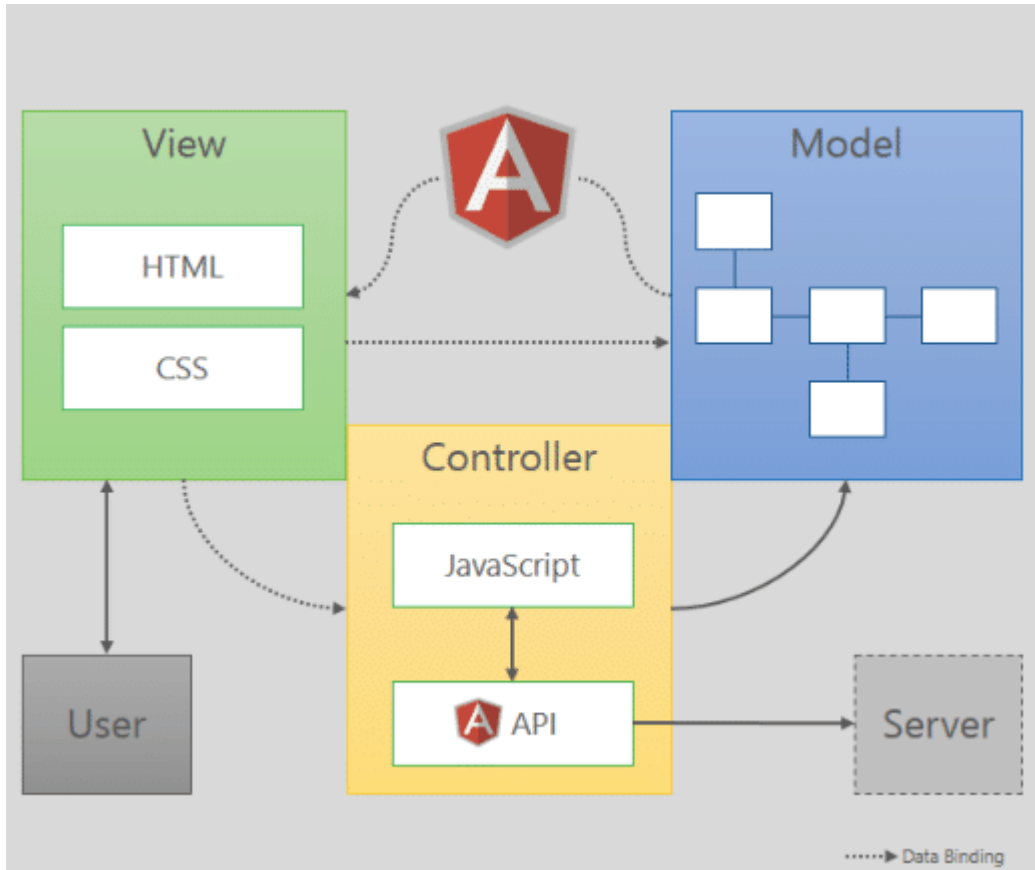
### Aplicación Web:

Una aplicación web es un tipo de sitio web avanzado que ofrece funcionalidad interactiva, similar a una aplicación de escritorio o móvil. Las aplicaciones web suelen incluir características como la autenticación de usuarios, interacciones en tiempo real, acceso a bases de datos, y manejo de datos complejos. Utilizan tecnologías como HTML, CSS, JavaScript, y a menudo frameworks y bibliotecas adicionales (por ejemplo, React, Angular, Vue.js en el frontend y Node.js, Django, Ruby on Rails en el backend).

## Arquitectura de una aplicación WEB



## Patrón de Diseño MVC – MODELO VISTA CONTROLADOR



### Modelo:

La lógica y los datos de la aplicación.

### Vista:

El HTML y el CSS forman la Vista. El HTML estructura la página su maquetado y diseño responsivo

### Controlador:

El JavaScript Maneja las interacciones del usuario, que cosas realiza, selecciona, elige el usuario.



## **CLASE NRO: 02 – PATRON MODELO VISTA CONTROLADOR**

**Tipo de Clase:** Práctica

**Duración:** 40 minutos

**Objetivo de la clase:**

El alumno al finalizar la clase, debe comprender la creación de una página web (documento HTML), la incorporación muy simple y sencilla de estilos a esa página CSS, y principalmente poder incorporar código JavaScript a la misma comenzando a incorporar las primeras sentencias del lenguaje y entendiendo como se escriben, como se hacen líneas de comentarios, como se realizan comentarios en bloque, como es la línea de ejecución de las sentencias, etc.

**Ejercicio propuesto Nro. 01:**

Proyecto: Construimos una carpeta y la abrimos con Visual Studio CODE, donde construiremos una página web e incrustaremos nuestro primer código JavaScript cuando la página se haya renderizado.

- ✓ Creamos documento HTML con un simple H1 de Titulo => llamaremos al archivo vista.html
- ✓ Incorporamos una simple hoja de estilos para modificar el h1
  - Cambiar de tamaño
  - Centralizar
  - Cambiar color de fondo
  - Cambiar color de la letra
- ✓ Creamos un archivo para incorporar el código JavaScript y lo llamaremos => controlador.js
- ✓ Al documento HTML le incorporamos la etiqueta <script> que nos permitirá vincular la vista con su controlador.
  - Aquí mostrar el esquema de vista ⇔ controlador
- ✓ Nuestras primeras instrucciones con JavaScript
  - Establecemos llaves {} como un bloque de código
  - Mensajes por la consola => console.log();
  - Alertas por pantalla => alert();
  - Confirmación de operaciones => confirm();
  - Ingresar valores => prompt();
  - Mensajes de error por consola => console.error()
  - Mensajes de Warning “advertencias” por consola. Console.warn(“”)
  - Comentarios de una sola línea //

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

- Bloque de comentarios `/* */`

### La vista (HTML)

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <title>Practica Nro. 001</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="stylesheet" href="estilos.css">
  </head>
  <body>
    <h1>Práctica Nro I - con Javascript</h1>
    <script src="controlador.js"></script>
  </body>
</html>
```

### El Código:

```
/* las llaves representan un bloque de código */
{

  /* mostramos datos por consola */

  console.log("esta es mi primera linea de codigo con javascript");

  /* otorgamos una advertencia por el navegador */

  alert("generamos un mensaje de alerta");

  /* generamos un mensaje donde solicitamos la confirmación de algo (si o
no) */

  confirm("confirma la operación ?");

  /*
      true => si la respuesta es verdadera
      false => si la respuesta es falsa
  */

  /* este mensaje saldrá en el navegador solicitando que ingrese un valor
*/

  prompt("Por favor ingrese su nombre");
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
/* mostrar un mensaje de error */

console.error("se produjo un error, ingreso de forma incorrecta un
valor");

/* mostramos una advertencia */

console.warn("Esto es una advertencia.");

// comentarios de una sola linea
/* bloque de comentarios */

}
```

**LINK PROYECTO:**

<https://drive.google.com/file/d/1RDHCKIF8xsshW0BW63ODXFiPIUMqdDeg/view?usp=sharing>

## **CLASE NRO. 03 – PRÁCTICA - TIPOS DE DATOS – OPERACIONES CON VARIABLES**

**Tipo de Clase:** Teórica

**Duración:** 40 minutos

**Objetivo de la clase:**

El alumno al finalizar la clase debe comprender que es una variable, para que se utilizan las variables, como es la sintaxis y recomendaciones para la creación de una variable, los tipos de variables existentes en JavaScript, los bloques de código entre {}, el ámbito de vida de la variable, y por último entender y poder saber el tipo de una variable.

- ✓ Que es una variable ¿?: Una variable en JavaScript es un contenedor o referencia que se utiliza para almacenar datos y asociarlos con un nombre simbólico. Este nombre puede luego ser utilizado para acceder y manipular el valor almacenado. Las variables pueden contener diferentes tipos de datos, incluyendo números, cadenas de texto, booleanos, objetos, funciones, y más. A continuación se presentan algunos puntos clave sobre las variables en JavaScript:
- ✓ Formas de declarar una variable
  - `let Numero1 = 0;`
  - `const NUMERO_FIJO = 220;`
- ✓ Particularidades al momento de asignarle un nombre a una variable
  - No pueden comenzar con símbolos excepto el signo \$
  - No pueden comenzar con números
  - Dos variables no pueden tener el mismo nombre
  - Las variables son case-sensitive
  - No pueden tener espacios en blanco en el medio del nombre de la variable
  - Declarar e inicializar una variable
  - Declarar y asignarle un valor posterior a la variable
  - Operaciones aritméticas simples entre variables (suma, resta, multiplicación, división).
  - Declaración de variables dentro de un bloque de códigos
    - En el mismo bloque de códigos no puede haber dos variables con el mismo nombre
    - Dos variables se pueden llamar igual, pero en bloques diferentes, y cada una de ellas puede tener nombres distintos.

## DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT

### MÓDULO 02 - JAVASCRIPT

#### ✓ Diferentes Tipos de Variables

- Variables numéricas
- Variables del tipo String
- Variables del tipo Boolean
- Valores Nulos
- Valor Indefinido

#### ✓ Bloques de Código => { }

- Que es un bloque de código.
- Pueden existir variables con el mismo nombre, pero en bloques diferentes
- JavaScript un lenguaje débilmente tipado, significa que las variables pueden cambiar de tipo de datos a lo largo del tiempo.
- Typeoff una instrucción que me permite determinar el tipo de datos de una variable.

#### El Código:

```
{

    /*****
    /** PRIMERA PARTE - DECLARACION Y OPERACIONES BÁSICAS */
    *****/

    /* Particularidades para nombrar variables

        - Un identificador (nombre de la variable) debe comenzar con una
        letra (a-z, A-Z), un signo de dólar ($) o un guion bajo (_).
        - Los caracteres siguientes pueden ser letras, números (0-9),
        guiones bajos o signos de dólar.
        - No puede comenzar con un número.
        - No se pueden crear dos variables con el mismo nombre en un
        mismo bloque

        - No se pueden usar palabras reservadas de JavaScript como
        nombres de variables.

        Algunas palabras reservadas incluyen
        let,
        const,
        var,
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
if,  
else,  
while,  
for,  
function,  
return,  
class,  
try,  
catch, entre otras.
```

- NO DEJAR ESPACIOS EN BLANCO entre el NOMBRE DE LA VARIABLE

```
let Total de Ingresos = 0;
```

```
*/
```

```
/* declaramos una variable y asignamos valor significa INICIALIZAR  
VARIABLES */
```

```
let a = 20;
```

```
let b = 30;
```

```
/* podemos crear una tercera variable y realizar la suma de esas  
variables
```

```
algunas operaciones básicas = EXPRESIONES
```

```
suma
```

```
resta
```

```
multiplicación
```

```
división
```

```
*/
```

```
let suma = a + b;
```

```
console.log("El resultado de la suma es: ",suma);
```

```
let resta = a - b;
```

```
console.log("El resultado de la resta es: ",resta);
```

```
let multiplicacion = a * b;
```

```
console.log("el resultado de la multiplicación es ",multiplicacion);
```

```
let division = a / b;
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
console.log("el resultado de la división es : ",division);

/* también se puede mostrar directamente el resultado de una operación
*/

console.log(a * 2 + 3 * b - 1);

/***** SEGUNDA PARTE - DIFERENTES TIPOS DE VARIABLES *****/

/* TIPO NUMERICO */

let z = 200;

/* TIPO STRING */

let nombreJugador = "JUAN ROMAN RIQUELME";

let nombreClub = "CLUB ATLETICO BOCA JUNIORS";

// esto se llama interpolación de cadenas //
let fraseCombinada = `El Jugador ${nombreClub} fue jugador de
${nombreClub}`;

/* TIPO BOOLEAN */

let AceptaElPago = true; // permite dos valores, true significa
verdadero, false que significa falso

/* VALORES NULOS */

let unValorNulo = null

/* VALORES INDEFINIDOS */

let unValorIndefinido = undefined

*/

}
```



DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
/* ***** */
/* TERCERA PARTE - BLOQUES DE CODIGO */
/* ***** */

/* esto es bloque 2 */
{
    let nombreProgramador = "DANIEL";
}

/* esto es bloque 3 */
{
    let nombreProgramador = "JORGE";
}

/* esto es bloque 4 */
{
    let a = 20;
    //let a = 30;
}

/* ***** */
/* CUARTA PARTE - JAVASCRIPT LENGUAJE DÉBILMENTE TIPADO */
/* ***** */

{

    let Numero1 = 100;
    console.log(Numero1);

    Numero1 = "JOSE DE SAN MARTIN";
    console.log(Numero1);

}

/* ***** */
/* QUINTA PARTE - case - Sensitive */
/* ***** */
{
    let nombre = "Ana";
    let Nombre = "Carlos";

    console.log(nombre); // "Ana"
    console.log(Nombre); // "Carlos"
}
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
/* *****  
/* SEXTA PARTE - Determinar el TIPO de una variable */  
/* *****  
  
{  
  let variableEjemplo = 99.99;  
  
  console.log ("el tipo de la variable es " + typeof variableEjemplo);  
  
  variableEjemplo = true;  
  
  console.log ("el tipo de la variable es " + typeof variableEjemplo);  
  
  variableEjemplo = "ahora me convertí en String";  
  
  console.log ("el tipo de la variable es " + typeof variableEjemplo);  
  
  variableEjemplo = undefined;  
  
  console.log ("el tipo de la variable es " + typeof variableEjemplo);  
  
  variableEjemplo=null;  
  
  console.log ("el tipo de la variable es " + typeof  
variableEjemplo);  
  
}
```

LINK PROYECTO:

<https://drive.google.com/file/d/1bMyMIholw-ILTUIMJOgfMg7ZuTEJeRfi/view?usp=sharing>

## **CLASE NRO. 04 – ESTRUCTURAS CONDICIONALES Y COMPARACIONES LOGICAS**

**Tipo de Clase:** Teórico Práctica

**Duración:** 40 minutos

**Objetivo de la clase:** Al finalizar la clase, los alumnos deberán comprender y dominar los conceptos de

- Creación del documento HTML
- Incorporación del código JavaScript como controlador de las interacciones del usuario
- Declaración de variables desde el teclado a través de prompt
- Inicialización de variables
- Realizar operaciones numéricas simples entre las variables numéricas
- Concatenación y/o interpolación de valores strings.
- Mostrar valores por consola.
- Visualizar el problema de dividir por cero
- Visualizar el problema de realizar una operación con una variable no inicializada
- Visualizar en el archivo controlador.js cuando escribo o trato de realizar o mostrar por consola una variable que no existe.
- Respetar la indentación de código.
- Introducción a las estructuras condicionales
- Las operaciones de comparación lógicas básicas >, <, =, >=, <=
  - $A > B \Rightarrow$  se lee A es mayor a B
  - $A < B \Rightarrow$  se lee A es menor a B
  - $A = B \Rightarrow$  se lee A es igual a B
  - $A \geq B \Rightarrow$  se lee A es mayor o igual que B
  - $A \leq B \Rightarrow$  se lee A es menor o igual que B
- Introducción a las estructuras condicionales (lado verdadero, lado falso)

### **Ejercicio propuesto Nro. 01:**

Realizar un programa para una Empresa de Transportes “La Catamarqueña” que realiza envíos de mercaderías a todo el País y además de cobrar el traslado, cobra además un seguro de protección de los bienes que equivale al 5% del valor Declarado. Realizar un programa en JavaScript que permita determinar el costo del seguro en función del valor declarado de la mercadería.

**Ejercicio propuesto Nro. 02:**

Realizar un programa utilizando JavaScript que permita ingresar dos números a través de la sentencia prompt y determinar mediante estructuras condicionales (if/else).

- Cuál de los dos números ingresados es mayor y mostrar el caso que ambos sean iguales.

**Solución: al final del capítulo**

**Ejercicio propuesto Nro. 03:**

Realizar un programa que permita calcular el precio de venta de los artículos de un negocio (comercio) sabiendo que en líneas generales el margen de ganancia es igual al 45%. Es decir, si un comerciante compra por mayor "ventanas" y las adquiere a \$ 250.000,00 pesos cada una (precio de costo), el precio de venta será de \$ 362.500,00 dado que tiene el 45% aplicado. Siendo la ganancia neta de \$ 112.500,00.

**Análisis:**

Si el precio de compra es igual a \$ 250.000,00 pesos, el 45% de ganancia nos daría un total de \$ 112.500 pesos que se calculan de la siguiente forma.

precioCompra = \$ 250.000,00

porcentajeGanancia = (\$ 250.000,00 \* 45) / 100 => me da un total de 112.500,00

precioFinalVenta = precioCompra + porcentajeGanancia => 250.000,00 + 112.500,00

**Solución:**

```
{
  /* Declaramos e Inicializamos las variables */
  let valorCompra = 250000;
  valorCompra = Number(prompt("Ingrese el valor de Compra"));

  /* Realizamos una operación de cálculo de variables */
  let porcentajeGanancia = (valorCompra * 45) /100;

  /* Calculamos el valor de venta */
```

```
let valorVenta = valorCompra + porcentajeGanancia;

/* Mostramos los resultados de la operación */
console.log("El Valor de Compra: ",valorCompra);
console.log("El porcentaje de Ganancia: ",porcentajeGanancia);
console.log("El valor de Venta :",valorVenta);
}
```

#### **Ejercicio propuesto Nro. 04: (práctica de estructuras condicionales)**

Sobre la base del ejercicio anterior, realizaremos a pedido del propietario del comercio un cambio muy importante, donde nos indica las siguientes reglas de negocio.

El porcentaje de ganancia estará en función del precio de compra, y nos indica las siguientes reglas que quiere que se apliquen.

- Los precios de aquellos productos donde el precio sea menor a 250.000 mil pesos, el porcentaje de ganancia será de 45% por ciento.
- En precios que sean iguales o mayores a 250.000 mil pesos el porcentaje de ganancia será el 65%.

```
{
  /* Declaramos e Inicializamos las variables */
  let valorCompra = 0;
  valorCompra = Number(prompt("Ingrese el valor de Compra"));

  /* Realizamos una operación de cálculo de variables */
  let porcentajeGanancia = 0;

  /* Realizamos una operación de cálculo de variables condicional */
  if(valorCompra < 250000)
  {
    console.log("el valor de costo es inferior a 250.000,00 pesos");
    porcentajeGanancia = (valorCompra * 45) /100;
  }
  else
  {
    console.log("el valor de costo es mayor o igual a 250.000,00 pesos");
    porcentajeGanancia = (valorCompra * 65) /100;
  }

  /* Calculamos el valor de venta */
}
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
let valorVenta = valorCompra + porcentajeGanancia;

/* Mostramos los resultados de la operación */
console.log("El Valor de Compra: ",valorCompra);
console.log("El porcentaje de Ganancia: ",porcentajeGanancia);
console.log("El valor de Venta :",valorVenta);
}
```

**LINK PROYECTO:**

<https://drive.google.com/file/d/17MjMS8Ne0Zqe6e0Jk5TJhHLcXISUJFW/view?usp=sharing>

## **CLASE NRO. 05 – ERRORES TIPICOS PARTE + TRY CATCH**

**Tipo de Clase:** Teórico Práctica

**Duración:** 40 minutos

**Objetivo:**

Al finalizar la clase, los alumnos deberán comprender con el nivel y conocimientos que tienen hasta el momento con las clases anteriores, cuáles son los errores comunes que cometen los programadores en sus inicios y comienzos con JavaScript.

- No inicializar variables
- Utilizar el “==” en comparaciones en lugar de utilizar el “===”
- Utilizar una variable no inicializada dentro de un condicional.
- Olvidarse de cerrar paréntesis (), corchetes [], llaves {} en lugares donde son obligatorios.
- Ser impropio en la estructura del programa, no respetar la indentación.
- Olvidarse de colocar comillas dobles o comillas simples, en algún mensaje que sea por consola por ejemplo
- Introducción a la sentencia try/catch

Console.log(“mensaje que le falta una comilla);

Truncar una estructura condicional (if/else) con un ; dentro de la comparación

```
let valor1 = 100;
let valor2 = 300;
if (valor1 > valor2);
{
    console.log("el valor 1 es mayor que el valor 2");
}
```

### **Ejercicio propuesto Nro. 05: (práctica de estructuras condicionales)**

Realizar un programa con JavaScript que permita declarar e introducir dos valores y determine cuál de los dos números es más grande y en caso contrario determine que ambos son iguales.

```
{
    /* declaramos e inicializamos los dos números*/

    let Numero1 = 0;
    let Numero2 = 0;

    /* solicitamos que ingrese el número 1 por teclado */
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
Numero1 = Number(prompt("Ingrese el número 1"));

/* solicitamos que ingrese el número 2 por teclado */
Numero2 = Number(prompt("Ingrese el número 2"));

if(Numero1 > Numero2) // Si el Numero1 es mayor al Numero2
{
    // si la condición se cumple se ejecuta este código
    console.log("El número mayor es el Numero1 ",Numero1," El menor es
",Numero2);
}
else
{
    if(Numero2 > Numero1) // si el Numero 2 es mayor al Numero 1
    {
        console.log("El número mayor es el Numero2 ",Numero2, "El menor
es ",Numero1);
    }
    else // por defecto, por descarte son iguales
    {
        console.log("Los números son iguales");
    }
}
}
```

Errores típicos más comunes al iniciar con JavaScript:

- Declarar variables y no asignarle valores y que las mismas participen de algún cálculo, forma, o alguna sentencia condicional (if/else). Valores undefined.
- Equivocarse en el nombre de variables al ser Case Sensitive (distingue mayúsculas de minúsculas)
- No cerrar paréntesis (), [] corchetes, {} llaves en instrucciones que requieren apertura y cierre.
- No cerrar comillas cuando estoy haciendo operaciones con valores String o concatenación de Strings.
- Realizar operaciones matemáticas, cálculos de porcentaje con valores que se capturaron desde el prompt ó desde la interfaz HTML sin convertir a números y sin determinar que realmente sean números.
- Errores de ámbito.
- **Errores de valores nulos.**



DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

**LINK PROYECTO:**

[https://drive.google.com/file/d/101JK-e7LI7yi85N\\_MLd0I-P-jlKCrKpV/view?usp=sharing](https://drive.google.com/file/d/101JK-e7LI7yi85N_MLd0I-P-jlKCrKpV/view?usp=sharing)

## **CLASE NRO. 06 – ESTRUCTURAS REPETITIVAS**

**Tipo de Clase:** Teórico Práctica

**Duración:** 120 minutos

**Objetivo:**

Al finalizar este capítulo, el alumno debe dominar y tener los conceptos de las tres estructuras repetitivas básicas, estructurales y fundamentales en programación y especialmente en JavaScript. Estas instrucciones son ciclo for, ciclo while y do while. Debe entender y comprender en que momentos se utilizan (ejemplos básicos donde vemos estos temas y logramos comprender la implementación de estas herramientas), las formas de salir de los bucles iterativos como así también iremos incorporando los temas previos vistos con ejercicios de lógica como ser:

- ✓ Declaración e inicialización de variables
- ✓ Estructuras condicionales if/else
- ✓ Estructuras condicionales anidadas
- ✓ Contadores
- ✓ Acumuladores
- ✓ Cálculo de porcentajes sobre el total
- ✓ Salir de estructuras repetitivas con break
- ✓ Saltar una iteración en estructuras repetitivas con continue
- ✓ Estructuras repetitivas
  - Ciclo for
  - Ciclo while
  - Ciclo do while

En JavaScript, las estructuras repetitivas permiten ejecutar un bloque de código varias veces según una condición. Las estructuras repetitivas básicas que veremos son: for, while y do...while.

### **Ciclo for**

El ciclo for se utiliza cuando se conoce de antemano cuántas veces se debe ejecutar un bloque de código. Su sintaxis básica es:

```
for (inicialización; condición; actualización)
{
    // código a ejecutar
}
```

**inicialización:** Se ejecuta una vez al comienzo del bucle. Aquí generalmente se inicializa una variable de control.

**condición:** Se evalúa antes de cada iteración. Si es true, se ejecuta el bloque de código; si es false, el bucle termina.

**actualización:** Se ejecuta al final de cada iteración. Aquí generalmente se actualiza la variable de control.

**Ejemplo:**

```
for (let i = 0; i < 5; i++)  
{  
    console.log(i);  
}
```

### Ciclo while

El ciclo while se utiliza cuando no se sabe cuántas veces se debe ejecutar el bloque de código y se depende de una condición que puede cambiar durante la ejecución. Su sintaxis básica es:

```
while (condición)  
{  
    // código a ejecutar  
}
```

**condición:** Se evalúa antes de cada iteración. Si es true, se ejecuta el bloque de código; si es false, el bucle termina.

Ejemplo:

```
let i = 0;

while (i < 5)

{

    console.log(i);

    i++;

}
```

### Ciclo do...while

El ciclo do...while es similar al ciclo while, pero con la diferencia de que el bloque de código se ejecuta al menos una vez antes de evaluar la condición. Su sintaxis básica es:

```
do

{

    // código a ejecutar

} while (condición);
```

**condición:** Se evalúa después de cada iteración. Si es true, el bloque de código se ejecuta nuevamente; si es false, el bucle termina.

Ejemplo:

```
let i = 0;

do {

    console.log(i);

    i++;

} while (i < 5);
```

### Comparación de las Estructuras Repetitivas

**for:** Se usa cuando se conoce el número de iteraciones. Es útil para iterar sobre rangos de números o colecciones conocidas.

**while:** Se usa cuando no se conoce el número de iteraciones y depende de una condición externa. Es más flexible, pero puede ser propenso a bucles infinitos si no se maneja correctamente la condición.

**do...while:** Se usa cuando se quiere asegurar que el bloque de código se ejecute al menos una vez. Es útil cuando la condición depende de operaciones que se realizan dentro del bucle.

### **Consideraciones**

**Bucle Infinito:** Un bucle que nunca termina se llama bucle infinito. Es crucial asegurarse de que la condición de salida eventualmente se cumplirá para evitar que el programa se bloquee.

### **Ejemplo de bucle infinito (evitar):**

```
while (true) {  
    console.log("Esto se ejecutará infinitamente");  
}
```

**Interrupción de Bucles:** La declaración break se puede usar para salir de un bucle prematuramente.

Ejemplo con break:

```
for (let i = 0; i < 10; i++) {  
    if (i === 5) {  
        break;  
    }  
    console.log(i); // Imprimirá del 0 al 4  
}
```

**Continuar en Bucles:** La declaración continue se usa para saltar a la siguiente iteración del bucle.

Ejemplo con continue:

```
for (let i = 0; i < 5; i++) {  
    if (i === 3) {  
        continue;  
    }  
    console.log(i); // Imprimirá 0, 1, 2, 4 (se saltará el 3)  
}
```

### Conclusión

Las estructuras repetitivas for, while y do...while son fundamentales en JavaScript para ejecutar bloques de código repetidamente bajo diferentes condiciones. Comprender cuándo y cómo usar cada tipo de bucle es crucial para escribir código eficiente y efectivo.

### Ejercicio propuesto Nro. 06: (estructuras repetitivas)

Realizar un proceso que permita cargar (tipo caja de supermercado) los productos comprados por los clientes de un pequeño comercio, donde el cajero irá introduciendo el precio de los productos y al final de ello visualizar el Total. Realizar este proceso con una estructura repetitiva para 8 productos.

**Solución:**

```
/* EJERCICIO NRO. 06  
Realizar un proceso que permita cargar (tipo caja de supermercado)  
los productos comprados por los clientes de un pequeño comercio,  
donde el cajero irá introduciendo el precio de los productos  
y al final de ello visualizar el Total. Realizar este proceso  
con una estructura repetitiva para 8 productos.  
*/  
  
{  
  
    /*  
    total de factura es una variable  
    donde iré acumulando los precios de los  
    productos que voy registrando, debe estar  
    fuera del ciclo repetitivo porque debe  
    ser global al cuerpo del ciclo.  
    */  
}
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
    */  
    let totalFactura = 0;  
  
    for(let i = 1;i <= 8;i=i+1)  
    {  
        /* declaro una variable y la inicializo en 0,  
        en esta variable iré solicitando que me ingrese  
        el precio de los productos  
        Esta variable se declarará e inicializará en cada  
        paso del ciclo. en cada iteración.  
        */  
  
        let precioProducto = 0;  
  
        /* solicito al usuario ingresar el precio del producto y lo guardo  
        justamente en una variable que se llama precioProducto */  
  
        precioProducto = Number(prompt(`Ingrese el Precio del Producto  
        ${i}`));  
  
        /* antes de comenzar la siguiente iteración  
        la guardo y acumulo en la variable totalFactura */  
  
        totalFactura = totalFactura + precioProducto;  
  
        /* visualizo el producto registrado */  
        console.log(`Producto Registrado:= ${i} valor ${precioProducto}`);  
    }  
  
    /* Muestro el total de los Productos */  
    console.log(`El Total de la Factura:= ${totalFactura}`);  
}
```

### Ejercicio propuesto Nro. 07: (estructura repetitiva while)

Realizar un programa en JavaScript que permita ingresar las notas de los trabajos finales de los alumnos de la diplomatura en “Desarrollo Web Full Stack con JavaScript” para ello se establecen las siguientes condiciones.

- No está establecido la cantidad de trabajos finales que se evaluarán
- Este será el cuadro con el que se analizará y asignará la clasificación de los mismos.
  - o Si la nota  $\geq 0$  y  $\leq 4$  serán trabajos desaprobados
  - o Si la nota  $> 4$  y  $\leq 7$  serán trabajos aprobados
  - o Si la nota  $> 7$  y  $< 10$  serán trabajos muy buenos
  - o Si la nota  $= 10$  serán trabajos excelentes
- Contemplar que el operador podría ingresar notas incorrectas, es decir podría poner una nota menor a cero o mayor a 10 con lo que sería claramente un error. Contemplar la cantidad de veces que se equivoca.
- Siempre preguntar si desea continuar cargando notas ¿?.

#### Solución:

```
let continua = "SI";
let contadorNotas = 0;
let contadorErrores = 0;
let contadorDesaprobados = 0;
let contadorAprobados = 0;
let contadorMuyBuenos = 0;
let contadorExcelentes = 0;

while (continua === "SI")
{
    let notaAlumno = Number (prompt("Ingrese la nota del alumno"));

    contadorNotas = contadorNotas + 1;

    if((notaAlumno < 0) || (notaAlumno > 10))
    {
        contadorErrores = contadorErrores + 1;
    }
    else
    {
        if((notaAlumno >= 0) && (notaAlumno <=4))
        {
            contadorDesaprobados = contadorDesaprobados + 1;
        }
    }
}
```



DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
}
if((notaAlumno >4) && (notaAlumno <=7))
{
    contadorAprobados = contadorAprobados + 1;
}
if((notaAlumno >7) && (notaAlumno <10))
{
    contadorMuyBuenos = contadorMuyBuenos + 1;
}
if(notaAlumno ===10)
{
    contadorExcelentes = contadorExcelentes + 1;
}
}
continua = prompt("continua con otro alumno ? (SI/NO)");
}

console.log(`La Cantidad de Desaprobados fueron
${contadorDesaprobados}`);
console.log(`La Cantidad de Aprobados fueron ${contadorAprobados}`);
console.log(`La Cantidad de Muy Buenos fueron ${contadorMuyBuenos}`);
console.log(`La Cantidad de Excelentes fueron ${contadorExcelentes}`);
```

**Ejercicio propuesto Nro. 08: (estructuras repetitivas) – ciclo for**

Realizar un programa en JavaScript que permita ingresar la edad de todos los estudiantes de la diplomatura de la “Diplomatura Universitaria en Desarrollo Web Full Stack con JavaScript” y obtener a partir de esos datos los siguientes datos.

Cantidades:

Cantidad de alumnos mayores a 50 años.

Cantidad de alumnos cuya edad sea menor o igual a 50 años.

Porcentajes:

Porcentaje de alumnos mayores a 50 años sobre el Total.

Porcentaje de alumnos menores o iguales a 50 años sobre el Total

**Solución:**

```
/*
Ejercicio propuesto Nro. 08: (estructuras repetitivas)
Realizar un programa en JavaScript que permita ingresar la
edad de todos los estudiantes de la diplomatura de la
“Diplomatura Universitaria en Desarrollo Web Full Stack con JavaScript”
y obtener a partir de esos datos los siguientes datos.

Cantidades:
    Cantidad de alumnos mayores a 50 años.
    Cantidad de alumnos cuya edad sea menor o igual a 50 años.
    Cantidad total de alumnos.
Porcentajes:
    Porcentaje de alumnos mayores a 50 años sobre el Total.
    Porcentaje de alumnos menores o iguales a 50 años sobre el Total

Considere como error aquellos que están por debajo de los 18 años
y por arriba de los 100.

*/

{

    /* declaro las variables principales */

    let cantidadErrores = 0; // para cantidad de errores
    let cantidadMayores50 = 0; // para mayores a 50 años
    let cantidadMenores50 = 0; // para menores a 50 años
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
let cantidadTotal = 0; // cantidad total

for(let i = 1;i <= 5;i = i + 1)
{
    let edadAlumno = Number(prompt(`Por favor Ingrese la Edad del alumno
nro. ${i}`));

    console.log(`Edad leida del alumno ${i} - ${edadAlumno}`);

    if((edadAlumno < 18) || (edadAlumno >= 100))
    {
        console.log("la edad registrada corresponde a un error");
        cantidadErrores = cantidadErrores + 1;
    }
    else
    {
        if((edadAlumno >= 18) && (edadAlumno <= 50))
        {
            console.log("edad registrada entre 18 y 50 años");
            cantidadMenores50 = cantidadMenores50 + 1;
        }
        else
        {
            console.log("edad registrada > 50 y < 100 años");
            cantidadMayores50 = cantidadMayores50 + 1;
        }
    }
}

/* Muestro los Totales */
console.log(`1 - Cantidad de Errores son : ${cantidadErrores}`);
console.log(`2 - Cantidad de Mayores a 50 años ${cantidadMayores50}`);
console.log(`3 - Cantidad de Menores o iguales a 50 años
${cantidadMenores50}`);

/* obtengo la cantidad total */
cantidadTotal = cantidadErrores + cantidadMayores50 + cantidadMenores50;

/* declaro variables y las inicializo, alli
voy a guardar los porcentajes sobre el total */

let porcentajeErrores = 0;
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
let porcentajeMayores50 = 0;
let porcentajeMenores50 = 0;

/* si hubo al menos 1 error, obtengo el porcentaje */
if(cantidadErrores > 0)
{
    porcentajeErrores = (cantidadErrores/cantidadTotal)*100;
}

/* si hubo al menos 1 mayor a 50 años obtengo el porcentaje */
if(cantidadMayores50 > 0)
{
    porcentajeMayores50 = (cantidadMayores50/cantidadTotal)*100;
}

/* si hubo al menos 1 (una persona) entre 18 y 50 años obtengo el promedio */
if(cantidadMenores50 > 0)
{
    porcentajeMenores50 = (cantidadMenores50/cantidadTotal)*100;
}

console.log(`El porcentaje de errores es de ${porcentajeErrores} %`);
console.log(`El porcentaje de mayores a 50 años es de
${porcentajeMayores50} %`);
console.log(`El porcentaje de menores o iguales a 50 años es de
${porcentajeMenores50} %`);
}
```

**Ejercicio propuesto Nro. 09: (estructuras repetitivas)**

Una Frigorífico posee una cinta transportadora y clasificadora de huevos para consumo humano. La cinta no tan solo los transporta sino también los clasifica según su peso. es decir, al final de la cinta existe una balanza electrónica de alta precisión que evalúa su peso y los clasifica.

- a) XL, super grandes: peso  $\geq$  73 gramos.
- b) L, grandes: peso  $\geq$  63 gramos y  $<$  73 gramos.
- c) M, medianos: peso  $\geq$  53 gramos y  $<$  63 gramos.

Determinar lo siguiente:

- 1) Cantidad total de Huevos (Todas las categorías)
- 2) Cantidad total de Huevos XL
- 3) Cantidad total de Huevos L
- 4) Cantidad total de Huevos M
- 5) Cantidad total de Huevos descartados
- 6) Determinar el % de Huevos XL sobre el Total
- 7) Determinar el % de Huevos L sobre el Total
- 8) Determinar el % de Huevos M sobre el Total

**Solución:**

```
{  
  
  let cantidadXL = 0;  
  let cantidadL = 0;  
  let cantidadM = 0;  
  let cantidadDescarte = 0;  
  let cantidadTotal = 0;  
  
  let porcentajeXL = 0;  
  let porcentajeL = 0;  
  let porcentajeM = 0;  
  let porcentajeDescarte = 0;  
  
  for(let i = 1; i <= 5; i++)  
  {  
    /* Dentro del Ciclo debería ir registrando el peso de cada huevo */  
  
    /* declaro e inicializo la variable donde
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
leeré el peso de cada huevo */

let pesoDeHuevo = 0;

/* leo desde el teclado el Peso de cada
huevo y lo convierto a número */
pesoDeHuevo = Number(prompt(`Ingrese el peso del huevo ${i}`));

/* muestro el peso del huevo registrado*/
console.log(`Huevo Registrado Nro. ${i} con peso ${pesoDeHuevo}`);

/*-----
comenzamos a analizar en que categoría
cae en función de su peso
-----*/

/* Si el peso del huevo es mayor o igual que 73 gramos es XL */
if(pesoDeHuevo >= 73)
{
    //cuento la cantidad de XL en su contador //
    cantidadXL = cantidadXL + 1;
    console.log(`Cantidad de XL := ${cantidadXL}`);
}
else
{
    /* si el peso es mayor o igual a 63 gramos y menor
    o a 73 es L*/
    if((pesoDeHuevo >= 63) && (pesoDeHuevo < 73))
    {
        cantidadL = cantidadL + 1;
        console.log(`cantidad de L := ${cantidadL}`)
    }
    else
    {
        /* Si el peso es mayor a 53 y menor a 63 es M */
        if((pesoDeHuevo >= 53) && (pesoDeHuevo < 63))
        {
            cantidadM = cantidadM + 1;
            console.log(`cantidad de M:= ${cantidadM}`);
        }
        else
        { // los que no ingresan a ninguna categoría son
descartes //
```

```
        cantidadDescarte = cantidadDescarte + 1;
        console.log(`cantidad de descartes:
${cantidadDescarte}`);
    }
}

/* obtengo la cantidad total de huevos
sumando las cantidades parciales
de cada categoria */

cantidadTotal = cantidadXL + cantidadL + cantidadM + cantidadDescarte;

/* muestro las cantidades de cada huevo */
console.log(`total de huevos XL: ${cantidadXL}`);
console.log(`total de huevos L:  ${cantidadL}`);
console.log(`total de huevos M:  ${cantidadM}`);
console.log(`cantidad de descartes: ${cantidadDescarte}`);

/* obtengo el porcentaje de huevos XL sobre el total */
if(cantidadXL > 0)
{
    porcentajeXL = (cantidadXL/cantidadTotal)*100;
}

/* obtengo el porcentaje de huevos L sobre el total */
if(cantidadL > 0)
{
    porcentajeL = (cantidadL/cantidadTotal)*100;
}

/* obtengo el porcentaje de huevos M sobre el total */
if(cantidadM > 0)
{
    porcentajeM = (cantidadM/cantidadTotal)*100;
}

/* obtengo el porcentaje de huevos M sobre el total */
if(cantidadDescarte > 0)
{
    porcentajeDescarte = (cantidadDescarte/cantidadTotal)*100;
}
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
/* visualizo los porcentajes */  
console.log(`-----`);  
console.log(`el porcentaje de XL ${porcentajeXL} %`);  
console.log(`el porcentaje de L ${porcentajeL} %`);  
console.log(`el porcentaje de M ${porcentajeM} %`);  
console.log(`el porcentaje de Descartes ${porcentajeDescarte} %`);  
  
}
```

**LINK PROYECTO:**

<https://drive.google.com/file/d/1fXQiMRMpxccf6gYhCrmZnP8YpurixOVD/view?usp=sharing>



## **CLASE NRO. 07 – FUNCIONES**

**Tipo de Clase:** Teórico Práctica

**Duración:** 120 minutos

**Objetivo de la clase:**

En el presente módulo de 120 minutos estimados. Introduciremos al alumno en el concepto de funciones, subprocesos, subrutinas, procedimientos, métodos con JavaScript, al finalizar la clase deberá comprender, dominar, entender y poner en práctica los siguientes conceptos que a continuación se detallan.

- Que es una función => sinónimos (subprocesos, subrutinas, procedimientos, métodos).
- Definición ó declaración de una función
  - o Palabra function
  - o Parámetros ó argumentos
  - o Cuerpo de la función
  - o Retorno de la función
- Funciones que devuelven valor
- Invocación de Funciones.
- Mostrar como las funciones pueden participar dentro de expresiones matemáticas
- Funciones que no devuelven valor => procedimientos.
  - o Hacer un proceso que imprime
  - o Hacer un proceso que muestra mensajes por consola
  - o Hacer un proceso que cambia de color objetos en la pantalla
  - o Hacer un proceso que oculta objetos de la pantalla
- Funciones y bibliotecas de funciones estándar
  - o Funciones de conversión de tipos de datos
    - parseInt()
    - parseFloat()
    - String()
    - Number()
  - o Funciones de manipulación de cadenas
    - Substring()
    - toUpperCase(), toLowerCase(): Convierten una cadena a mayúsculas o minúsculas.
    - indexOf(), lastIndexOf(): Devuelven la posición de una sub cadena dentro de una cadena.
    - split(): Divide una cadena en un array de sub cadenas.
    - concat(): Une dos o más cadenas.

### Funciones matemáticas

- `Math.random()`: Devuelve un número pseudoaleatorio entre 0 y 1.
- `Math.round()`, `Math.floor()`, `Math.ceil()`: Redondean números.
- `Math.min()`, `Math.max()`: Devuelven el valor mínimo o máximo de una serie de números.
- `Math.sqrt()`, `Math.pow()`: Calculan la raíz cuadrada y la potencia de un número, respectivamente.
- Funciones como constantes => Expresión de función
- Funciones Flecha ó arrow functions
  - Forma de declaración
  - Omitir paréntesis, llaves y la palabra `return` cuando se recibe un solo parámetro de entrada. Ejemplo.  
`const devolverParrafo = texto => `<p>${texto}</p>`;`
- Funciones que reciben como parámetro otras funciones.

### Las Funciones:

Las funciones son fragmentos de código que tienen una funcionalidad específica y tienen el propósito de aislar un conjunto de instrucciones que en conjunto resuelven algún problema. La palabra función tiene la intención de “encapsular” lógica para resolver algún problema y cumplir un objetivo muy puntual, específico.

Las funciones son bloques de código que se pueden definir una vez y ejecutar en múltiples ocasiones a lo largo de un programa. En JavaScript, las funciones son ciudadanos de primera clase, lo que significa que se pueden asignar a variables, pasar como argumentos a otras funciones y devolver desde otras funciones. Las funciones son fundamentales para estructurar el código, permitir la reutilización y modularización, y son esenciales para la programación funcional y la programación orientada a objetos en JavaScript.

Una Función se define básicamente con la palabra `function()`

```
/* declaración de una función - Forma clásica */  
  
function Cuadrado(numero)  
{  
    return numero * numero;  
}
```

Las funciones tienen un nombre, los parámetros de entrada que va entre los paréntesis, el cuerpo de la función que va entre `{ }` “llaves” y muy importante la palabra `return`.

**Ejercicio propuesto Nro. 10:**

Realizar una función que reciba como parámetro de entrada un número y devuelva el cuadrado del mismo.

Nota: visualizar las diferentes formas de invocación y la participación de la función dentro de una expresión matemática.

**Nota: expresar las funciones de forma tradicional y como arrow functions**

```
/* declaración de una función - Forma clásica */

/*
  1 - la palabra reservada function
  2 - el nombre de la función
  3 - los parametros de entrada entre () parentesis
  4 - {} el cuerpo de la función
  5 - el retorno de una función
*/

function Cuadrado(numero)
{
    return numero * numero;
}

// ejemplo de invocación de la función
let Resultado = Cuadrado(5);
```

**Ejercicio Propuesto Nro. 11:**

Realizar una función que reciba como parámetro de entrada un número y devuelva la raíz cuadrada del mismo.

Nota: visualizar las diferentes formas de invocación y la participación de la función dentro de una expresión matemática.

**Nota: expresar las funciones de forma tradicional y como arrow functions**

**Ejercicio propuesto Nro. 12:**

Realizar una función que reciba como parámetro de entrada el importe de una factura, el tipo de artículo que se está facturando y devuelva el importe de la misma con el IVA incluido sabiendo que:

Tipo 1: 21%: Es la alícuota general aplicable a la mayoría de los bienes y servicios.

Tipo 2: 27%: Se aplica a ciertos servicios públicos como energía eléctrica y gas natural

Tipo 3: 10.5%: Se aplica a bienes y servicios específicos

incluyendo:

Venta de ciertos alimentos básicos (frutas, verduras, carnes, etc.).

Prestación de servicios médicos y paramédicos.

Obras de construcción de viviendas sociales.

Venta de ciertos medicamentos.

Productos tecnológicos.

Tipo 4: 5%: Aplica a ciertos productos agrícolas y ganaderos, como frutas, hortalizas y carnes en algunos casos específicos.

Tipo 5: 0% Exentos

importe de una factura y devuelva su valor con el IVA incluido.

**Nota: expresar las funciones de forma tradicional y como arrow functions**

**Solución:**

```
/* Función que recibe dos parámetros.
uno es el tipo de articulo a facturar y otro
es el importe neto, la función analiza el tipo
de articulo y aplica el porcentaje dependiendo
lo que le corresponda */

const fnDevolverIVA = (tipoArticulo,importeNeto)=>
{
    if(tipoArticulo === 1)// alicuota general
    {
        return (importeNeto + (importeNeto * 21)/100);
    }
    if(tipoArticulo === 2) // servicios públicos
    {
        return (importeNeto + (importeNeto * 27)/100);
    }
    if(tipoArticulo === 3) // productos tecnológicos
    {
        return (importeNeto + (importeNeto * 10.5)/100);
    }
    if(tipoArticulo === 4) // productos agrícolas
    {
        return (importeNeto + (importeNeto * 5)/100);
    }
    if(tipoArticulo === 5) // Exentos
    {
        return importeNeto;
    }
}

/* EJEMPLO DE INVOCACIÓN DE LA FUNCIÓN */
{

    let tipo = 1;

    let neto = 20000;

    let Resultado = fnDevolverIVA(tipo,neto);

    console.log(`El Resultado es : ${Resultado}`);
}
```

```
}
```

### **Ejercicio propuesto Nro. 13:**

Realizar una función que pueda calcular y determinar la dosis de insulina recomendada para un paciente diabético. Basada en tres datos importantes para el cálculo.

- 1) Nivel de glucosa en sangre
- 2) Peso Corporal (en kilogramos)
- 3) Tipo de diabetes
  - a. Tipo 1
  - b. Tipo 2

Para Tipo 1: El cálculo es el 50% del Peso corporal del paciente + el 50% del nivel de glucosa en sangre, este último termino solamente si la glucosa es mayor a 180.

Para Tipo 2: El cálculo es el 20% del Peso corporal del paciente + el 50% del nivel de glucosa en sangre, este último termino solamente si la glucosa es mayor a 180.

La función debe retornar la dosis de insulina recomendada y recibir como parámetros de entrada (argumentos) nivel de glucosa, peso corporal y tipo de diabetes.

**Nota: expresar las funciones de forma tradicional y como arrow functions**

### **Ejercicio propuesto Nro. 14:**

Realizar una función que pueda obtener y calcular el IMC – índice de masa corporal sabiendo que la fórmula es la siguiente  $IMC = \text{peso (kg)} / \text{altura (metros)}^2$  al cuadrado

$$IMC = \frac{\text{peso (kg)}}{\text{altura (m)}^2}$$

Clasificación del IMC

La Organización Mundial de la Salud (OMS) clasifica el IMC en las siguientes categorías:

Bajo peso: IMC menor de 18.5

Peso normal: IMC entre 18.5 y 24.9

Sobrepeso: IMC entre 25 y 29.9

Obesidad grado I: IMC entre 30 y 34.9

Obesidad grado II: IMC entre 35 y 39.9

Obesidad grado III (Obesidad mórbida): IMC de 40 o más

La función debe recibir como parámetros la altura (en metros) y el peso (en kilogramos) y calcular el IMC, devolverlo y además mostrar los carteles de (bajo peso, peso normal, sobre peso, etc) según el cálculo.

**Nota: expresar las funciones de forma tradicional y como arrow function.**

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

**LINK PROYECTO:**

<https://drive.google.com/file/d/12WpknHNI RivJpjmImjBjS5WaVLXV2V3B/view?usp=sharing>

## **CLASE NRO. 08 – MANEJO DEL DOM**

**Tipo de Clase:** Teórico Práctica

**Duración:** 120 minutos

**Objetivo de la clase:**

Al finalizar la clase, el alumno deberá comprender que es el DOM, cuál es la estructura de árbol del DOM, los elementos, atributos, como se hace para interactuar con los elementos del DOM, como se hace desde JavaScript para seleccionar elementos del DOM, poder interactuar con esos elementos por medio de variables a través de los selectores más simples como getElementById(), cuales son los eventos más importantes de dominar y controlar al principio como lo son window.onload y el evento click en los elementos HTML. Entender y comprender la diferencia del renderizado de un documento HTML (estático) y la manipulación del DOM donde actualizaremos parte del documento por lo que se producirá un renderizado (en tiempo de ejecución).

- ✓ **Introducción y explicación del DOM**
- ✓ **Estructura de árbol del DOM y sus ELEMENTOS**
  - Elementos
  - Atributos
  - Texto
  - Comentarios
- ✓ **Interacción con el DOM**
  - Los selectores por ID
  - Los selectores por Nombre
  - getElementById
  - getElementByClassName
  - getElementByTagName
  - querySelector("selector")
  - querySelectorAll("selector")
- ✓ **Modificación de Elementos del DOM**
  - innerHTML
  - textContent
- ✓ **Crear y Agregar Elementos al DOM (Esto a Nivel de Comentario nada mas)**
  - createElement()



✓ Eventos Principales en una página WEB

- **window.onload**
- **click**
- **Incorporamos dos botones y capturamos el evento click en dos botones**
- **Incorporamos el Evento click al documento y observamos que se produce un efecto de propagación de a dentro hacia a fuera.**

## DOCUMENT OBJECT MODEL (DOM)

El Document Object Model (DOM) es una interfaz de programación para documentos HTML y XML. Proporciona una representación estructurada del documento, permitiendo a los desarrolladores acceder y manipular su contenido, estructura y estilo. El DOM representa el documento como una jerarquía de nodos, donde cada nodo corresponde a una parte del documento, como un elemento, atributo o texto.

### Estructura del DOM

El DOM organiza el contenido del documento en una estructura de árbol. Los nodos del árbol pueden ser de varios tipos, incluyendo elementos, atributos y texto. Aquí hay una descripción de los principales tipos de nodos:

- **Document:** El nodo raíz del documento. Todos los demás nodos son descendientes del nodo Document.
- **Element:** Representa un elemento HTML o XML.
- **Text:** Representa el contenido de texto dentro de un elemento.
- **Attribute:** Representa un atributo de un elemento.
- **Comment:** Representa un comentario en el documento.

### Acceso y Manipulación del DOM

El DOM proporciona varias interfaces y métodos para acceder y manipular el contenido del documento. Aquí se presentan algunos de los métodos y propiedades más comunes:

#### Acceso a Nodos / elementos

- `document.getElementById(id)`: Devuelve el elemento con el ID especificado.
- `document.getElementsByClassName(className)`: Devuelve una colección de todos los elementos con la clase especificada.
- `document.getElementsByTagName(tagName)`: Devuelve una colección de todos los elementos con el nombre de etiqueta especificado.
- `document.querySelector(selector)`: Devuelve el primer elemento que coincide con el selector CSS especificado.
- `document.querySelectorAll(selector)`: Devuelve una colección de todos los elementos que coinciden con el selector CSS especificado.

## Manipulación de Nodos

- `element.innerHTML`: Permite obtener o establecer el contenido HTML de un elemento.
- `element.textContent`: Permite obtener o establecer el contenido de texto de un elemento.
- `element.setAttribute(name, value)`: Establece el valor de un atributo en el elemento.
- `element.getAttribute(name)`: Obtiene el valor de un atributo del elemento.
- `element.removeAttribute(name)`: Elimina un atributo del elemento.
- `element.appendChild(child)`: Añade un nodo hijo al final de la lista de hijos de un elemento.
- `element.removeChild(child)`: Elimina un nodo hijo de un elemento.

## Eventos en el DOM

Los eventos son una parte fundamental del DOM, permitiendo a los desarrolladores interactuar con los usuarios. Algunos de los eventos más comunes incluyen:

- `click`: Ocurre cuando se hace clic en un elemento.
- `mouseover`: Ocurre cuando el puntero del mouse se coloca sobre un elemento.
- `mouseout`: Ocurre cuando el puntero del mouse se mueve fuera de un elemento.
- `keydown`: Ocurre cuando se presiona una tecla.
- `load`: Ocurre cuando una página o un recurso se ha cargado por completo.

## Manejadores de Eventos

Los manejadores de eventos son funciones que se ejecutan en respuesta a eventos. Existen dos formas de responder a los eventos, una forma es asignarle una función directamente al evento del botón y la otra es agregar una función al escuchador de eventos.

### Forma 1 – asignarle una función al evento:

```
const boton = document.getElementById('miBoton');

boton.onclick = function()
{
    alert("me están haciendo click");
}
```

### Forma 2 – agregar una función al escuchador de eventos

```
const boton = document.getElementById('miBoton');
boton.addEventListener('click', function() {
    alert('me están haciendo click !');
});
```

## Manipulación de Estilos

El DOM también permite la manipulación de estilos CSS de los elementos. Esto se puede hacer a través de la propiedad `style` de un elemento:

```
const div = document.getElementById('miDiv');  
div.style.color = 'red'; // Cambia el color del texto a rojo  
div.style.backgroundColor = 'yellow'; // Cambia el color de fondo a amarillo
```

Haciendo un resume final, observamos las formas de seleccionar los elementos del DOM para poder acceder a ellos y manipularlos.

## Métodos para Seleccionar Elementos en el DOM

Métodos que Devuelven un Único Elemento

- **`document.getElementById(id)`**

Devuelve: Un único elemento.

Descripción: Selecciona el primer elemento que coincide con el id especificado.

- **`document.querySelector(selector)`**

Devuelve: Un único elemento.

Descripción: Selecciona el primer elemento que coincide con el selector CSS especificado.

Métodos que Devuelven una Colección de Elementos

- **`document.getElementsByClassName(className)`**

Devuelve: Una colección (HTMLCollection) de elementos.

Descripción: Selecciona todos los elementos que tienen la clase especificada.

- **`document.getElementsByTagName(tagName)`**

Devuelve: Una colección (HTMLCollection) de elementos.

Descripción: Selecciona todos los elementos que tienen el nombre de etiqueta especificado.

- **`document.getElementsByName(name)`**

Devuelve: Una colección (NodeList) de elementos.

Descripción: Selecciona todos los elementos que tienen el atributo name especificado.

- **`document.querySelectorAll(selector)`**

Devuelve: Una colección (NodeList) de elementos.

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

Descripción: Selecciona todos los elementos que coinciden con el selector CSS especificado.

**LINK PROYECTO:**

<https://drive.google.com/file/d/1-1gxzUB3pmOI0nFQ9LHRNpzgYfmEPfld/view?usp=sharing>

## **CLASE NRO. 09 – MANEJO DEL DOM 2**

**Tipo de Clase:** Teórico Práctica

**Duración:** 40 minutos

**Objetivo de la clase:**

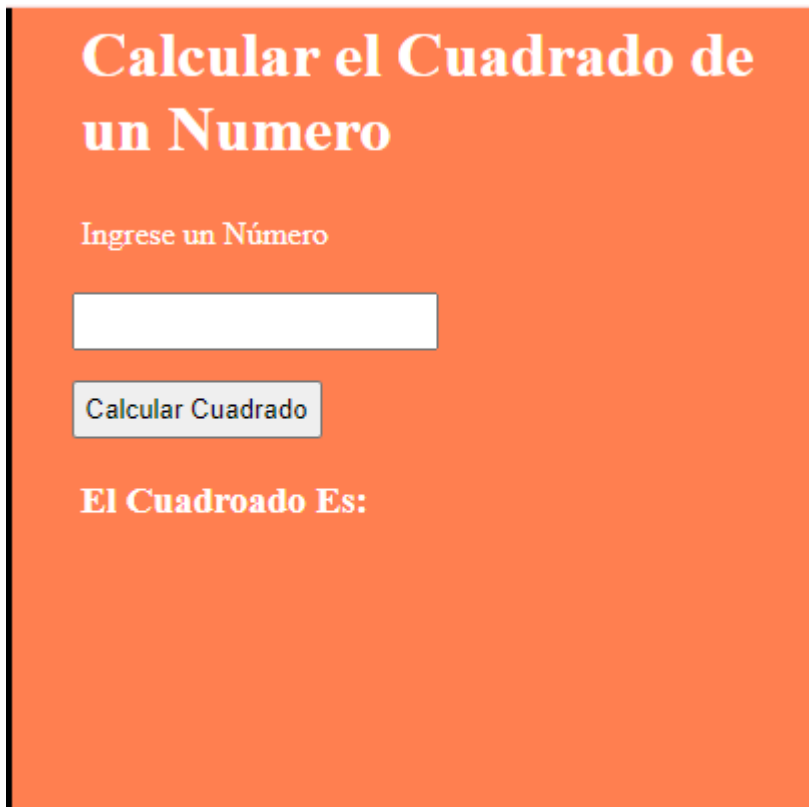
La presente clase tiene como objetivo fundamental, que el alumno vaya comprendiendo la forma de crear el documento HTML, incorporarle código a través de la etiqueta <script>, incorporar la hoja de estilos, en el código JavaScript capturar el evento onload, capturar los elementos del DOM con el que desea interactuar, capturar el evento click del botón donde realizaremos el cálculo, crear variables, realizar operaciones con variables, ir mostrando las etapas por consola, concatenar variables string y mostrarlas en el navegador.

Temas Adicionales:

Mostraremos cuando una variable es NULA = null

Mostraremos cuando una variable es INDEFINIDA = undefined

**Vista del Resultado Esperado:**



The image shows a web application interface with an orange background. At the top, the title "Calcular el Cuadrado de un Numero" is displayed in a large, bold, white serif font. Below the title, the text "Ingrese un Número" is written in a smaller, white serif font. Underneath this text is a white rectangular input field. Below the input field is a button with the text "Calcular Cuadrado" in a white serif font. At the bottom of the visible area, the text "El Cuadroado Es:" is written in a white serif font.

## HTML

```
<!DOCTYPE html>

<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="estilos.css">
</head>
<body>
  <div class="divPrincipal">
    <h1>Calcular el Cuadrado de un Numero</h1>
    <label>Ingrese un Número</label>
    <input type="number" id="idTxtNumeroIngresado">
    <input type="button" value="Calcular Cuadrado" id="btnCuadrado">
    <h3 id="idResultado">El Cuadroado Es:</h3>
  </div>
  <script src="controlador.js"></script>
</body>
</html>
```

## CSS

```
/* con esto le digo a todos
los elementos que tengan
margen 0, padding */

* /* selector para todos los atributos */
{
  margin: 0; /* sacamos margenes por defecto */
  padding: 0; /* le sacamos el padding por defecto */
}

body
{
  background-color: black; /* Color del body todo negro */
}

/* a todos los elementos que están
dentro del div */
.divPrincipal > *
{
  margin-left: 30px; /* margen izq. de 30 px*/
}
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
padding: 5px 5px; /* un padding de 5px arriba y abajo y izq y der */
display: block; /* se muestren en bloque */
margin-bottom:15px; /* separados hacia abajo por 15 px*/
}

/* al div principal
le damos */
.divPrincipal
{
    width: 400px; /* ancho */
    height: 400px; /* alto */
    background-color: coral; /*color de fondo */
    color: white; /* color de la letra */
    display: block; /* se muestren en bloque */
    margin-left:auto; /* margen derecho */
    margin-right:auto; /* margen izquierdo */
}
```

## JAVASCRIPT

```
4
/* onload es el evento que se dispara automáticamente
cuando la página esta cargada y lista para que el usuario
interactúe con ella */
window.onload = function()
{
    /* mensaje por consola */
    console.log("la aplicación está lista para interactuar");

    /* creamos variables que se vinculen
    a los elementos HTML del DOM que queremos
    capturar o con los que queremos interactuar */

    let idTxtNumeroIngresado =
document.getElementById("idTxtNumeroIngresado");
    let btnCuadrado = document.getElementById("btnCuadrado");
    let idResultado = document.getElementById("idResultado");

    /* mostramos por consola las variables
    creadas y comprobamos que hagan referencia
    a los elementos HTML que queremos capturar
    o interactuar */

    console.log(idTxtNumeroIngresado);
```

```
console.log(btnCuadrado);

console.log(idResultado);

/* vamos a programar el evento click
del boton */

btnCuadrado.onclick = function()
{
    console.log("estan haciendo click en el boton");

    let Numero = Number(idTxtNumeroIngresado.value);

    console.log(Numero);

    let Resultado = Numero * Numero;

    /* aqui mostrar la forma de concatenar
    dos strings */

    console.log("El Resultado es " + Resultado);

    idResultado.textContent = `El Resultado es: ${Resultado}`;

}
```

**LINK PROYECTO:**

[https://drive.google.com/file/d/1Z8lhmfT3vV3\\_MCwbABNgB6Ay6CZ5iB6x/view?usp=sharing](https://drive.google.com/file/d/1Z8lhmfT3vV3_MCwbABNgB6Ay6CZ5iB6x/view?usp=sharing)



## **CLASE NRO. 10 – MANEJO DEL DOM 3**

**Tipo de Clase:** Teórico Práctica

**Duración:** 40 minutos

**Objetivo de la clase:**

Apuntamos que el alumno consolide los conceptos que se vienen proporcionando desde el inicio del temario, crear documentos HTML, crear sus estilos, incorporarle funcionalidad JavaScript a través de su etiqueta correspondiente, capturar dentro de su script el momento de finalización de renderizado del documento “onload”, mostrar que la página esta renderizada, crear variables que apunten correctamente a los objetos visuales que estimamos capturar, manipular y controlar (“por algo el código lo denominamos controlador”), incorporar funcionalidad a los botones a través de su evento “click” y luego dentro del bloque de código de cada botón preparar toda la lógica necesaria para capturar los valores ingresados, convertirlos a número, realizar las operaciones establecidas y mostrar los resultados a través de la manipulación del DOM.

## HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Practica Nro. 005</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="stylesheet" href="estilos.css">
  </head>
  <body>
    <h1>Práctica Nro 05 - Calculadora Simple</h1>
    <div class="divPrincipal">
      <label for="idTxtValor1">Ingrese el Número 1</label>
      <input type="text" id="idTxtValor1">
      <label for="idTxtValor1">Ingrese el Número 2</label>
      <input type="text" id="idTxtValor2">
      <input type="button" value="Sumar" id="idBtnSumar">
      <input type="button" value="Restar" id="idBtnRestar">
      <h3 id="idResultadoSuma">El Resultado de la SUMA:</h3>
      <h3 id="idResultadoResta">El Resultado de la RESTA:</h3>
    </div>
    <script src="controlador.js"></script>
  </body>
</html>
```

## CSS

```
/* con esto le digo a todos
los elementos que tengan
margen 0, padding */

* /* selector para todos los atributos */
{
  margin: 0; /* sacamos margenes por defecto */
  padding: 0; /* le sacamos el padding por defecto */
}

body
{
  background-color: black; /* Color del body todo negro */
}
```

```
/* a todos los elementos que están
dentro del div */
.divPrincipal > *
{
    margin-left: 30px; /* margen izq. de 30 px*/
    padding: 5px 5px; /* un padding de 5px arriba y abajo y izq y der */
    display: block; /* se muestren en bloque */
    margin-bottom: 15px; /* separados hacia abajo por 15 px*/
}

/* al div principal
le damos */
.divPrincipal
{
    width: 400px; /* ancho */
    height: 400px; /* alto */
    background-color: coral; /*color de fondo */
    color: white; /* color de la letra */
    display: block; /* se muestren en bloque */
    margin-left: auto; /* margen derecho */
    margin-right: auto; /* margen izquierdo */
}
```

## JAVASCRIPT

```
/* este es el evento principal. digamos que determina cuando
la página esta completamente cargada */
window.onload = function()
{
    // mensaje por consola
    console.log("La pagina esta cargada");

    /* creo una variable y lo vinculo al elemento HTML cuyo
    id es el que le pasamos entre comillas */
    let idTxtValor1 = document.getElementById("idTxtValor1");

    /* creo una variable y lo vinculo al elemento HTML cuyo
    id es el que le pasamos entre comillas */
    let idTxtValor2 = document.getElementById("idTxtValor2");

    let idBtnSumar = document.getElementById("idBtnSumar");
```

```
let idBtnRestar = document.getElementById("idBtnRestar");

let idResultadoSuma = document.getElementById("idResultadoSuma");

let idResultadoResta = document.getElementById("idResultadoResta");

// VALORES NULOS //

/* aqui aprovecho de crear una variable
y apuntar a algo que no existe en el DOM */

let variableQueDevolveraNULL =
document.getElementById("cajaTextoINEXISTENTE");

console.log(idTxtValor1);
console.log(idTxtValor2);
console.log(variableQueDevolveraNULL);
console.log(idBtnSumar);
console.log(idBtnRestar);
console.log(idResultadoSuma);
console.log(idResultadoResta);

idBtnSumar.onclick = function()
{

    /* obtengo los valores que tienen
    los input en el documento HTML */

    /* debo convertir los valores introducidos a NUMBER */
    let numero1 = Number(idTxtValor1.value);
    let numero2 = Number(idTxtValor2.value);

    /* creo otra variable que se llame resultadoSuma
    donde realizaré las operación de suma entre
    las variables numero1 y numero2.
    */
    let resultadoSuma = numero1 + numero2;

    /* muestro por consola los
    valores por separado y
    también el del resultado de la Suma */

    console.log(numero1);
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
console.log(numero2);
console.log(resultadoSuma);

/* modifiko el DOM, específicamente la etiqueta
que contendrá la suma y donde mostraré
el resultado de la Suma */

idResultadoSuma.textContent = `La Suma es igual a ${resultadoSuma}`;

}

idBtnRestar.onclick = function()
{
    /* obtengo los valores que tienen
    los input en el documento HTML */

    let numero1 = Number(idTxtValor1.value);
    let numero2 = Number(idTxtValor2.value);

    /* realizo la operación de resta */
    let resultadoResta = numero1 - numero2;

    /* muestro los resultados */
    console.log(numero1);
    console.log(numero2);
    console.log(resultadoResta);

    /* modifiko el DOM, específicamente la etiqueta
    idResultadoResta para actualizarle el mensaje
    */

    idResultadoResta.textContent = `La Diferencia es igual a
    ${resultadoResta}`;

}

}
```

**LINK PROYECTO:**

<https://drive.google.com/file/d/1HGbwNLgHmDYhY-Ntaj1DDJp0YNymBpOt/view?usp=sharing>

## **CLASE NRO. 11 – ARRAYS / VECTORES.**

**Tipo de Clase:** Teórico Práctica

**Duración:** 120 minutos

**Objetivo de la clase:**

Al finalizar la clase, el alumno deberá comprender que la forma de declarar literalmente vectores, acceder a posiciones de un vector para cargar elementos, mostrar el contenido de una posición del vector, formas de recorrido de un vector, funciones predeterminadas que tienen los vectores. Estos son los puntos que se deberían observar en la clase

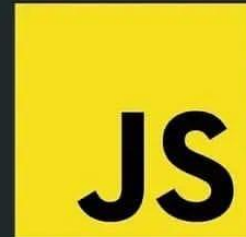
- Declarar un vector literal
- Mostrar el contenido de un vector
- Acceder a una posición del vector y modificarlo
- Agregar al final **push()**
- Eliminar el último valor y devolverlo **pop()**
- Eliminar el primer elemento del vector y devolverlo **shift()**
- Agregar al principio **unshift()**
- Eliminar elementos contiguos **splice()**
- Devolver la ubicación de un elemento **indexOf()** y **lastIndex()**. Uno retorna la primera ubicación y el otro devuelve la última posición.
- Devolver un elemento y la ubicación, pero con criterios más flexibles y definidos por el usuario como **find** y **findIndex**.
- Filtrar y seleccionar y devolver un conjunto de elementos del vector que cumplan con una condición específica. **filter**
  - o Forma 1: con una arrow function que no tiene la palabra los () para el parámetro, no tiene las llaves {} del cuerpo y no lleva la palabra return;
  - o Forma 2: con una arrow function completa que tiene los () y la palabra return y las llaves del cuerpo {}
  - o Forma 3: con una función anónima clásica definida dentro del parámetro de filter
  - o Forma 4: con una función declarada previamente y pasarle como parámetro la función dentro del filter.
- Recorrer tradicionalmente un vector con un ciclo **for**
- Recorre un vector con **vector.forEach()**
- Transformación de los elementos de un vector con **vector.map()** recorre los elementos, aplica la función pasada como parámetro y devuelve cada elemento transformado, a todos ellos los devuelve y los agrega en un vector de salida.
- Funciones reductoras con **reduce()**. Que permiten acumular valores, encontrar el mayor, el menor, etc.
- Función de ordenamiento **sort()** con las particularidades cuando son valores string o cuando son valores numéricos.

- Transformar un vector y darlo vuelta “al revés” con **reverse()**.
- Particularidad de copiar variables o constantes que contienen vectores. Es decir, se crean referencias a los elementos y no se realiza una copia del elemento. Dado este problema surge la necesidad de hacer una clonación de esos elementos y sale **spreadOperator** **también se puede usar slice() pero es más completo y flexible spread**. El problema de la copia por referencia ocurre con Vectores y con Objetos.
- spreadOperator ...
  - spread para copiar o clonar vectores
  - spread para unir en un vector dos o más vectores
  - spread para crear una copia ampliada de un vector
  - spread para clonar un objeto o ampliarle las propiedades (tipo fusión)
  - convertir lista de nodos ó elementos que se obtienen por `querySelectorAll` en un array
  - spread como una forma de pasar parámetros en una función, indicándole que los parámetros p1, p2, p3 están en el vector que tiene justamente 3 posiciones. Esto sucede en la invocación de las funciones.
  - Spread como forma de declarar una función que recibirá múltiples parámetros, de esta forma internamente la función los trata como si fueran un vector y en la invocación se puede hacer `invocacion_funcion(p1,p2,p3,p4,p5)`.
  - Convertir colecciones o listas en Arrays

-  
Definición: es una estructura de datos lineal que permite el almacenamiento de múltiples valores y la posibilidad de accederlos a todos mediante un solo identificador, en este caso el nombre del vector.

## Array Methods...

- |                  |                   |               |
|------------------|-------------------|---------------|
| 1. values()      | 16. concat()      | 31. isArray() |
| 2. length()      | 17. some()        | 32. filter()  |
| 3. reverse()     | 18. splice()      | 33. keys()    |
| 4. sort()        | 19. flat()        | 34. map()     |
| 5. at()          | 20. lastIndexOf() |               |
| 6. fill()        | 21. of()          |               |
| 7. from()        | 22. every()       |               |
| 8. join()        | 23. slice()       |               |
| 9. toString()    | 24. flatMap()     |               |
| 10. pop()        | 25. findIndex()   |               |
| 11. forEach()    | 26. find()        |               |
| 12. shift()      | 27. includes()    |               |
| 13. copyWithin() | 28. entries()     |               |
| 14. push()       | 29. reduceRight() |               |
| 15. unshift()    | 30. reduce()      |               |



### LINK PROYECTO:

<https://drive.google.com/file/d/1fWX-F2PzUfbw2dp0pGrPidR7lo2jldE8/view?usp=sharing>



## **CAPITULO NRO. 12 – POO – INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS.**

**Tipo de Clase:** Teórico Práctica

**Duración:** 120 minutos

**Objetivo de la clase:**

El objetivo del presente capítulo es introducir al alumno en el amplio mundo de la programación orientada a objetos, el alumno debe comprender al finalizar la clase los conceptos de clase, objetos, atributos de la clase, función constructora, la instanciación de objetos, del poder de encapsular lógica, con atributos, métodos que engloban una determinada complejidad y resuelven un problema determinado. Luego poder convertir esos objetos a formato JSON que significa JavaScript Object Notation. Debe comprender que a lo largo de su carrera profesional seguramente interactuará en el mundo de la POO en algunos de los roles que definimos ya sea como consumidor de clases o constructor de clases que provean soluciones específicas.

Los objetos que derivan de una clase son una estructura de datos y vienen a sumarse a las que ya venimos viendo que son

**Estructuras de datos:** hasta este momento el alumno viene manejando estructuras de datos como ser:

- Variables
- Constantes
- Vectores

Ahora podremos incorporar los objetos como una nueva estructura de datos, una estructura que permite contener atributos, estos atributos pueden ser estructuras que ya conocemos como variables, como vectores (pero dentro de la POO se los conoce como atributos). Los objetos provienen de una clase que sería en el mundo real como un “MOLDE”.

En el mundo real por ejemplo si deseo construir block para la construcción de una casa necesito una blockera que sería el “MOLDE = CLASE”. Y los block que se construyeron a partir de ese MOLDE se denominarían objetos.

**CLASE**



**OBJETOS A PARTIR DE LA CLASE**



**Definición técnica de Clase en POO:**

Una clase en Programación Orientada a Objetos (POO) es un modelo o una abstracción que representa un concepto o una entidad del mundo real. Esta clase encapsula datos (propiedades) y comportamientos (métodos) relacionados con ese concepto.

Definición alternativa (Profesor): una clase es como definir una estructura de datos nueva que permite incorporar y encapsular un conjunto de atributos, lógica, métodos (que son similares a las

funciones) y que en conjunto resuelven un problema determinado, encapsular significaría en este contexto englobar todo lo que sea necesario para resolver un problema.

Por ejemplo, se pueden construir clases que permiten a partir de series de números obtener todas las medidas estadísticas de dispersión como, por ejemplo, la media aritmética, la moda, la mediana, la desviación standard.

La idea de encapsular (atributos + métodos específicos) tiene la intención de englobar una problemática en particular y que en ella se encuentre toda la solución respecto a esa temática. Muchos programadores tienen experiencia y conocimiento respecto de temas particulares y que requieren mucha capacitación al respecto, como, por ejemplo.

#### Ejemplos de clases

- Ejemplos de clases o conjunto de clases nativas de JavaScript, que el lenguaje ya trae incorporadas y que uno utiliza normalmente y esto nos abstrae de cómo están implementados esos métodos.
  - String
  - Array
    - Push
    - Pop
    - forEach
    - map
    - filter
  - Promise
  - Fetch
- Existen Programadores ó empresas que construyen clases que encapsulan toda la lógica del Sistema Financiero y lógicamente venden y comercializan esas soluciones.
- Existen Programadores ó empresas que construyen clases que encapsulan toda la lógica de gráficos estadísticos.
- Existen Programadores ó empresas que construyen clases que encapsulan toda la lógica para manipular motores o componentes electro mecánicos. Esto para los programadores que se encargan de la robótica los abstrae completamente de cómo está implementado cada método que tiene ese conjunto de clases.

Es decir, nosotros como programadores podemos tener dos roles

**Consumidor de Objetos:** instanciados a partir clases que engloban comportamiento específico.

**Constructor de Clases:** que encapsulan complejidad y estarán provistos para que otros programadores puedan resolver problemas de esa temática sin necesidad de comprender a fondo los detalles de la implementación de las soluciones.

#### **Rol de Consumidor de Clases**

Los desarrolladores que asumen el rol de consumidores de clases utilizan las clases creadas por otros (terceros) para construir sus aplicaciones. Estos desarrolladores pueden no necesitar

entender completamente los detalles internos de cómo funcionan estas clases, siempre y cuando puedan interactuar con ellas de manera efectiva a través de sus interfaces públicas. Este enfoque promueve la reutilización de código, la abstracción, modularidad y estandarización.

### **Rol de Creador de Clases**

Por otro lado, los desarrolladores que asumen el rol de creadores de clases son aquellos que tienen un conocimiento específico en un área o dominio y construyen clases que encapsulan esa lógica. Estos desarrolladores tienen la responsabilidad de diseñar clases que sean fáciles de usar y que oculten la complejidad interna detrás de una interfaz clara y bien definida. Su objetivo es proporcionar a otros desarrolladores una abstracción limpia y efectiva para trabajar con conceptos complejos o especializados.

### **Importancia de Ambos Roles**

Ambos roles son igualmente importantes en el desarrollo de software orientado a objetos. Los consumidores de clases se benefician del trabajo de los creadores de clases al utilizar sus abstracciones para construir aplicaciones de manera eficiente y efectiva. Por otro lado, los creadores de clases juegan un papel fundamental al proporcionar soluciones especializadas y encapsular conocimiento específico del dominio, lo que facilita la construcción de sistemas complejos y la colaboración entre equipos multidisciplinarios.

### **Tipos de OBJETOS**

En JavaScript tenemos dos tipos de objetos, los objetos literales que no necesitan una clase para ser instanciados, solamente con definirlos el lenguaje los interpreta como tal y aquellos que necesitan clases para darle la forma, con sus atributos, constructor, etc.

#### **Objetos Literales:**

```
const objetoLiteral = {  
  nombre: 'Juan',  
  edad: 30  
};
```

#### **Objetos Instanciados a partir de clase:**

```
class Persona {  
  constructor(nombre, edad) {  
    this.nombre = nombre;  
    this.edad = edad;
```

```
}  
}
```

```
const objetoInstanciado = new Persona('Juan', 30);
```

Podríamos definir que existen dos tipos de clases y por ende sus objetos

- Clases estructurales
  - o Permiten ENCAPSULAR almacenar datos de cosas, hechos, entidades como por ejemplo datos de personas, datos de registración de eventos (compras, ventas, entradas y salidas de un depósito). Para este tipo de datos no es necesario construir una clase y luego instanciar la clase, simplemente podemos construir los objetos con sus atributos y valores.

Es por ello que los denominamos “objetos literales”.

```
const Persona1 = {Nombre:"MESSI LIONEL ANDRES",Domicilio:"MIAMI 824"};  
console.log(Persona1);  
  
/* (02) - Definimos otro objeto literal */  
  
const Persona2 = {Nombre:"RIQUELME JUAN ROMAN",Domicilio:"CIUDAD  
AUTONOMA DE BS.AS."};  
console.log(Persona2);  
  
/* (03) - Definimos un objeto literal con un método a dentro */  
  
const Factura1 =  
{FechaCompra:"21/03/2020",CantidadProductos:5,PrecioUnitario:2500.20,getTotal(){return this.CantidadProductos * this.PrecioUnitario}};  
console.log(Factura1);  
console.log(Factura1.getTotal()); // invocamos el método que está dentro  
del objeto  
  
/* (04) - Convertimos el OBJETO JAVASCRIPT A JSON => que quiere decir  
JAVASCRIPT OBJECT NOTATION */  
  
console.log(JSON.stringify(Persona1));  
console.log(JSON.stringify(Persona2));  
  
/* cuando convierte un objeto a JSON no transforma la función interna ó  
método */  
console.log(JSON.stringify(Factura1));
```

- Clases funcionales

- Permiten ENCAPSULAR almacenar atributos y lógica que tiene como objetivo realizar cálculos que son complejos o que requieren un conocimiento profundo del tema, es decir por ejemplo clases que se dedican a partir de números obtener medidas estadísticas de dispersión por dar un ejemplo.

A Continuación, realizaremos la construcción de una clase que tendrá como funcionalidad encapsular la lógica de sumar dos números. Es decir, a partir de esa clase se podrán obtener

Múltiples objetos que tendrán la lógica lista para encapsular:

```
class SumaDeDos
{
    // los atributos de la clase
    Numero1 = 0;
    Numero2 = 0;
    Resultado = 0; // un atributo de la clase

    /* metodo constructor de la clase permite
    inicializar los atributos de la clase */
    constructor(p1,p2)
    {
        this.Numero1 = p1;
        this.Numero2 = p2;
        this.Sumar();
    }

    /* Getters y Setters */

    // método que me permitirá retornar el Numero1 //
    getNumero1()
    {
        return this.Numero1;
    }

    // método que me permitirá retornar el Numero2 //
    getNumero2()
    {
        return this.Numero2;
    }

    // método que me permitirá modificar el Numero1 //
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
setNumero1(parametro)
{
    this.Numero1 = parametro; // actualizo el atributo y por ende
    // actualizo el resultado */
    this.Sumar(); // al cambiar el valor
}

// método que me permitirá modificar el Numero2 //
setNumero2(parametro)
{
    this.Numero2 = parametro; // actualizo el atributo y por ende tengo
    // que actualizar el resultado */
    this.Sumar();
}

Sumar()
{
    this.Resultado = this.Numero1 + this.Numero2;
}

getResultado()
{
    this.Sumar();
    return this.Resultado;
}
}
```

Forma de instanciar los objetos a partir de la clase SumaDeDos

```
// creamos un objeto a partir de la clase SumaDeDos //

let objeto1 = new SumaDeDos(10,20);
console.log(objeto1.getResultado());

let objeto2 = new SumaDeDos(22,34);
console.log(objeto2.getResultado());

objeto2.setNumero1(115);
console.log(objeto2.getResultado());
```

### Partes de una Clase

- Atributos de una clase: son lugares, estructuras de datos (“variables”) que pueden ser booleanos, numéricos, String u otros objetos que permiten almacenar datos que son importantes para el objetivo de la clase.
  - Públicos: pueden ser accedidos y modificados desde afuera de la clase
  - Privados: pueden ser modificados y accedidos solo por métodos internos de la clase
  - Estáticos: son atributos que permanecerán estáticos en todas las instancias, son útiles para mantener información única para todas las instancias. Es decir, son atributos que pertenecen a la clase y no a la instancia. Si bien cada instancia puede acceder a dichos atributos, los mismos son únicos y si se modifican en una instancia, esa modificación se reflejará en todas las instancias.
- El Constructor: es un método (“función”) que tiene como objetivo asignarles valores a los atributos de la clase. Siempre y cuando no sean estáticos.
- Métodos de la clase: es comportamiento, procesos, (“funciones”) que tienen como objetivo realizar cálculos, operaciones con los atributos o hacer cálculos específicos y devolverlos al programa principal que instanció el objeto.
  - Públicos: en JavaScript todos los métodos por defecto son públicos.
  - Privados: no soportados en JavaScript
  - Estáticos: métodos que tienen acceso a atributos estáticos ya sea para asignar valores o para retornar los valores.
  - Getters y Setters: conjunto de métodos (“funciones”) que se encargan de asignarle valor a cada atributo y también a devolver valores de los atributos.
    - Cada atributo tendrá dos métodos, 1 método get y 1 método set.

### Ejemplo de Clases y Bibliotecas de Clases de Terceros que ayudan a la tarea cotidiana del programador:

**Express.Router (Express.js):** Express.js es un marco web de Node.js que se utiliza para construir aplicaciones web y APIs. Express.Router es una clase que se utiliza para crear objetos de enrutador en una aplicación Express, lo que permite organizar las rutas de manera modular.

**React.Component (React):** React es una biblioteca de JavaScript para construir interfaces de usuario. React.Component es una clase base que se utiliza para crear componentes personalizados en aplicaciones React. Estos componentes encapsulan la lógica y la interfaz de usuario relacionada en una sola unidad.

**Mongoose.Model (Mongoose):** Mongoose es una biblioteca de modelado de datos para MongoDB en Node.js. Mongoose.Model es una clase que se utiliza para definir modelos de datos en una aplicación Node.js y proporciona métodos para interactuar con la base de datos MongoDB.



**Chart.js:** Chart.js es una biblioteca JavaScript simple pero poderosa para crear gráficos interactivos y responsivos en páginas web. Es fácil de usar y ofrece una amplia variedad de tipos de gráficos, como barras, líneas, radar, dispersión, etc.

**D3.js (Data-Driven Documents):** D3.js es una biblioteca de JavaScript para manipular documentos basados en datos. Si bien es más avanzada y flexible que otras bibliotecas, también puede ser más compleja de aprender. Se utiliza para crear visualizaciones de datos personalizadas y altamente interactivas.

**Highcharts:** Highcharts es una biblioteca de gráficos JavaScript que ofrece una amplia gama de gráficos y visualizaciones, incluidos histogramas, gráficos de dispersión, gráficos de líneas, gráficos de áreas y más. Es fácil de configurar y tiene una amplia documentación.

**Google Charts:** Google Charts es una biblioteca gratuita de visualización de datos desarrollada por Google. Ofrece una variedad de gráficos y visualizaciones, como gráficos de barras, líneas, áreas, histogramas, dispersión, entre otros. Es fácil de usar y se integra bien con otros servicios de Google.

**Plotly.js:** Plotly.js es una biblioteca de gráficos de código abierto que permite crear gráficos interactivos y personalizables en páginas web. Ofrece soporte para una variedad de tipos de gráficos, incluidos histogramas, gráficos de dispersión, gráficos de líneas, gráficos de barras y más.

### **Conceptos Avanzados de POO**

Herencia Extender las clases utilizando la palabra extends.

Herencia Método super();

Composición versus Herencia:

Definición de interfaces (no soportado nativamente 100% con JavaScript):

Conjunto de métodos que no tienen implementación, no tienen código, pero las clases que se adhieran a esa interfaz deberán implementar la solución a esos métodos.

Definición de clases abstractas:

No existe un concepto directo de "clase abstracta" como en otros lenguajes de programación orientada a objetos, como Java o C#. Sin embargo, es posible emular el comportamiento de una clase abstracta utilizando técnicas disponibles en JavaScript.

Una clase abstracta es una clase que no puede ser instanciada directamente y que generalmente contiene métodos abstractos, es decir, métodos que deben ser implementados por las clases que la heredan. En JavaScript.

### **LINK PROYECTO:**

<https://drive.google.com/file/d/16cymrqyYPWgokYLYiw3RZPxorsYjD9Xn/view?usp=sharing>

## **CLASE NRO. 13 – MÓDULOS – BIBLIOTECAS**

Los módulos son una funcionalidad de JavaScript que permite organizar el código en unidades separadas y reutilizables. Cada módulo puede contener variables, funciones, clases, u otros fragmentos de código que pueden ser importados y utilizados en otros módulos. Esto facilita la gestión y mantenimiento del código, especialmente en aplicaciones grandes.

### **Historia de los Módulos en JavaScript**

Antes de ES6 (ECMAScript 2015), JavaScript no tenía una forma nativa de gestionar módulos. Los desarrolladores utilizaban patrones como el Módulo de JavaScript (JavaScript Module Pattern) y herramientas de carga de módulos como CommonJS (utilizado principalmente en Node.js) y AMD (Asynchronous Module Definition) para estructurar su código.

Con la llegada de ES6, se introdujeron módulos nativos, proporcionando una sintaxis estándar para la exportación e importación de código.

### **¿Para Qué Sirven las Exportaciones?**

Las exportaciones permiten exponer partes del código de un módulo para que puedan ser utilizadas en otros módulos. Esto ayuda a dividir el código en partes más pequeñas y manejables, facilitando la reutilización y el mantenimiento.

### **¿Qué cosas se pueden exportar/importar?.**

Se pueden realizar exportaciones anónimas, de elementos primitivos, funciones, vectores, clases, objetos, fragmentos, etc.

Se pueden realizar exportaciones con nombre de los mismos elementos

Se pueden realizar exportaciones mixtas, es decir exportaciones con nombres, exportaciones por defecto.

**Ejemplo 01: Exportación e importación de valores primitivos anónimos:**

**MÓDULO:** **ejemplo01\_versionsistema.js**

```
export default "2024";
```

**IMPORTACIÓN:**

**/\* Ejemplo 01 - importación de valor primitivo por nombre \*/**

```
import ejemplo01_versionsistema from "./ejemplo01_versionsistema.js";  
import versionsistema from "./ejemplo01_versionsistema.js";
```

**Ejemplo 02: Exportación e importación anónima de un vector**

**MÓDULO:** **ejemplo02\_provincias.js**

```
export default ["CATAMARCA", "LA RIOJA", "CORDOBA", "TUCUMAN"];
```

**IMPORTACIÓN:**

**/\* Ejemplo 02 - importación de valor primitivo como vector \*/**  

```
import ejemplo02_provincias from "./ejemplo02_provincias.js";  
import provincias from "./ejemplo02_provincias.js";
```

**Ejemplo 03: Exportación e importación de una función anónima arrow function**

**MÓDULO:** **ejemplo03\_cuadrado.js**

```
export default (numero)=>  
{  
    return numero * numero;  
}
```

**IMPORTACIÓN:**

**/\* Ejemplo 03 - importación de una función anónima POR DEFECTO \*/**  

```
import ejemplo03_cuadrado from "./ejemplo03_cuadrado.js";  
import cuadrado from "./ejemplo03_cuadrado.js"
```

**Ejemplo 04: Exportación de una función anónima tradicional**

**MÓDULO:** **ejemplo04\_funcioncubo.js**

```
export default function(numero)  
{  
    return (numero * numero * numero);  
}
```

**IMPORTACIÓN:**

```
/* Ejemplo 04 - imortación de una función CUBO */  
import ejemplo04_funcioncubo from "./ejemplo04_funcioncubo.js";  
import cubo from "./ejemplo04_funcioncubo.js";
```

**Ejemplo 05: Exportación de una clase anónima**

**MÓDULO: ejemplo05\_clasesumaanonima.js**

```
export default class  
{  
  constructor(p1,p2)  
  {  
    this.numero1 = p1;  
    this.numero2 = p2;  
  }  
  
  getSuma()  
  {  
    return (this.numero1 + this.numero2);  
  }  
}
```

**IMPORTACIÓN:**

```
/* Ejemplo 05 - importación de una clase anónima */  
import ejemplo05_clasesumaanonima from "./ejemplo05_clasesumaanonima.js";  
import Suma from "./ejemplo05_clasesumaanonima.js";
```

**Ejemplo 06: Exportación de un objeto anónimo literal en este ejemplo**

**MÓDULO: ejemplo06\_objetoanonimo.js**

```
export default  
{  
  fecha:"01/01/2020",  
  autor:"ING. DANIEL MALDONADO",  
  version:"K2B EVO II. revision 4"  
}
```

**IMPORTACIÓN:**

```
/* Ejemplo 06 - importación anónima de un objeto */
```

DIPLOMATURA EN DISEÑO WEB FULL STACK CON JAVASCRIPT  
MÓDULO 02 - JAVASCRIPT

```
import ejemplo06_objetoanonimo from "./ejemplo06_objetoanonimo.js";  
import version_sistema from "./ejemplo06_objetoanonimo.js";
```

### Ejemplo 07: Exportación de un Botón anónimo

**MÓDULO:** ejemplo07\_boton.js

```
export default function()
{
  let variableBoton = document.createElement("input");

  variableBoton.value = "soy un boton";
  variableBoton.type = "button";

  variableBoton.addEventListener("click",()=>
  {
    alert("soy un boton generado desde una biblioteca, me puedo replicar varias veces");
  })

  return variableBoton;
}
```

**IMPORTACIÓN:**

```
/*Ejemplo 07 - importación de un boton anonimo */

import boton1 from "./ejemplo07_boton.js";
```

### Ejemplo 08: Exportación de un fragmento anónimo

**MÓDULO:** ejemplo08\_crearunfragmento.js

```
export default function()
{
  let fragmento = document.createDocumentFragment();

  let boton1 = document.createElement("input");
  boton1.value = "Soy el Boton 1 del Fragment";
  boton1.type = "button";

  let boton2 = document.createElement("input");
  boton2.value = "Soy el Boton 2 del Fragment";
  boton2.type = "button";

  boton1.addEventListener("click",()=>
```

```
{
  alert("soy el boton1 del Fragment");
})

boton2.addEventListener("click",()=>{
  alert("soy el boton2 del Fragment");
})

fragmento.appendChild(boton1);
fragmento.appendChild(boton2);

return fragmento;
}
```

#### IMPORTACIÓN:

```
/*Ejemplo 08 - importación de un fragmento anónimo */
import fragmento from "./ejemplo08_crearunfragmento.js";
```

#### Ejemplo 09: Exportaciones nombradas

**MÓDULO:** `ejemplo09_exportaciones_nombradas.js`

```
/* exportamos una funcion clasica con nombre */
export function funcionClasica()
{
  console.log("soy una función clasica")
}
```

```
/* exportamos una función flecha con nombre */
export const funcionFlecha =()=>
{
  console.log("soy una funcion flecha");
}
```

```
/* exportamos una clase */
export class Superficie
{
  constructor(p1,p2)
  {
    this.largo = p1;
    this.ancho = p2;
  }
}
```

```
getSuperficie()
```



```
{
  return (this.largo * this.ancho);
}

}

/* exportamos un objeto */

export const Club = {nombre:"CLUB ATLETICO BOCA JUNIORS",copasintercontinentales:3};
```

#### **IMPORTACIÓN:**

```
import {Club,funcionClasica,funcionFlecha} from
"./ejemplo09_exportaciones_nombradas.js";

import {Superficie} from "./ejemplo09_exportaciones_nombradas.js";
```

#### **Ejemplo 10: Exportaciones por Defecto incluyendo exportaciones nombradas**

##### **MÓDULO: ejemplo10\_bibliotecamixta.js**

```
export const fnFlecha1 = ()=>
{
  console.log("soy una funcion flecha dentro de una modulo mixto");
}

export function fnClasica2()
{
  console.log("soy una funcion tradicional dentro de una modulo mixto");
}

/* exportación por defecto nombrada */
export default function fnFuncionPorDefecto()
{
  console.log("soy una funcion por defecto nombrada dentro un modulo mixto");
}
```

#### **IMPORTACIÓN:**

```
import {fnFlecha1,fnClasica2} from "./ejemplo10_bibliotecamixta.js"
import fnFuncionPorDefecto from "./ejemplo10_bibliotecamixta.js";
```

### **Ejemplo 11: IMPORTAR TODO UN MÓDULO COMPLETO**

#### **MÓDULO: ejemplo11\_bibliotecamixta2.js**

```
const funcionSaludar = function(nombre)
{
    console.log(`hola ${nombre} bienvenido`);
}

const funcionDespedir = function(nombre)
{
    console.log(`nos vemos pronto ${nombre}`);
}

const funcionRecordatorio = function(nombre)
{
    console.log(`Estimado ${nombre} recuerde que bla bal bal`);
}

export{funcionSaludar,funcionDespedir};

export default funcionRecordatorio;
```

#### **IMPORTACIÓN:**

```
import * as Funciones from "./ejemplo11_bibliotecamixta2.js";
```

#### **USO de LOS ELEMENTOS DEL MÓDULO:**

```
Funciones.funcionSaludar("DANIEL");

Funciones.funcionDespedir("DANIEL");

Funciones.default("DANIEL");
```

### **LINK DEL PROYECTO:**

<https://drive.google.com/file/d/1N-hZFmXPocn4K6rTumBoW37lZVyWMuBX/view?usp=sharing>

## **CLASE NRO. 14 – MANIPULACIÓN DINÁMICA DEL DOM**

El manejo dinámico del DOM se refiere a la capacidad de crear, modificar y eliminar elementos del DOM utilizando JavaScript en tiempo real, basándonos en acciones del usuario u otros eventos.

El manejo dinámico apunta principalmente a la creación de elementos/objetos HTML como etiquetas, botones, imágenes, contenedores (div) y así todo tipo de etiquetas que se van a crear mediante programación y se van a incrustar en el DOM (document object model). Es decir, son etiquetas HTML que no estaban escritas originalmente en el documento HTML estático y por lo tanto se van a crear posteriormente, dependiendo de las interacciones y acciones que realice el usuario. Todas estas etiquetas se generarán mediante programación, se ejecutarán mediante instrucciones que permitirán construir tanto la etiqueta como así también su comportamiento.

Existen básicamente tres formas ó técnicas de incorporar elementos al DOM de forma programática (cuando nos referimos a la forma programática nos referimos mediante programación con el lenguaje JavaScript).

- **Forma 1: Manipulación Programática del DOM (Programmatic DOM Manipulation):**

Incluye métodos como

```
document.createElement(),  
document.createTextNode(),  
document.createDocumentFragment()  
document.createComment(data)
```

- **Forma 2: Manipulación del DOM mediante HTML Interno (Inner HTML Manipulation):**

Utiliza propiedades como innerHTML para establecer o devolver el contenido HTML de un elemento. Características: Facilita la creación rápida de estructuras HTML complejas, pero puede ser menos seguro y más susceptible a vulnerabilidades XSS.

- **Forma 3: Plantillas del DOM (Template-Based DOM Manipulation):**

Utiliza elementos <template> para definir fragmentos de HTML que no se renderizan inmediatamente y pueden ser clonados e insertados en el DOM cuando sea necesario.

Características: Permite la reutilización de fragmentos de HTML, mejora el rendimiento y la organización del código.

**Forma 1: Manipulación Programática del DOM (Programmatic DOM Manipulation):**

- `document.createElement(tagName)`

Descripción: Crea un nuevo elemento HTML con el nombre de etiqueta especificado.  
`var nuevoDiv = document.createElement('div');`

- `document.createTextNode(data)`

Descripción: Crea un nuevo nodo de texto con el texto especificado.  
`var texto = document.createTextNode('Hola, mundo!');`

- `element.innerHTML`

Descripción: Establece o devuelve el contenido HTML de un elemento. Puede usarse para crear múltiples elementos o nodos de texto de una sola vez.

```
var contenedor = document.getElementById('contenedor');
contenedor.innerHTML = '<div class="nuevo">Nuevo contenido</div>';
```

- `element.insertAdjacentHTML(position, text)`

Descripción: Inserta el texto HTML especificado en una posición determinada en relación con el elemento.

Posiciones posibles:

'beforebegin': Antes del elemento mismo.

'afterbegin': Justo dentro del elemento, antes de su primer hijo.

'beforeend': Justo dentro del elemento, después de su último hijo.

'afterend': Después del elemento mismo.

```
var contenedor = document.getElementById('contenedor');
contenedor.insertAdjacentHTML('beforeend', '<p>Nuevo párrafo</p>');
```

- `document.createDocumentFragment()`

Descripción: Crea un nodo de documento vacío que puede contener nodos. Es útil para agregar múltiples elementos al DOM de una vez, mejorando el rendimiento.

```
let fragmento = document.createDocumentFragment();
for (let i = 0; i < 5; i++)
{
    let nuevoP = document.createElement('p');
    nuevoP.textContent = 'Párrafo ' + (i + 1);
    fragmento.appendChild(nuevoP);
}

document.getElementById('contenedor').appendChild(fragmento);
```

**FORMA1 :**

Ventajas:

Control Granular: Permite un control detallado sobre cada aspecto del elemento. Puedes establecer propiedades, atributos, estilos y eventos de manera individual.

Seguridad: Es menos susceptible a ataques XSS (Cross-Site Scripting) porque no se interpreta HTML como código.

Mejor Rendimiento: Especialmente cuando se manipulan muchos elementos o se realizan múltiples operaciones en el DOM, ya que evita el reflujo y el repintado innecesarios del navegador.

Facilidad de Uso con Eventos: Permite agregar fácilmente event listeners directamente al elemento.

Desventajas:

Verbosidad: Puede ser más largo y detallado al escribir comparado con

**FORMA 2:**

Ventajas:

Simplicidad: Es más conciso y fácil de escribir cuando se necesita agregar múltiples elementos con HTML complejo.

Rapidez en Implementación: Ideal para agregar rápidamente bloques de HTML sin mucho detalle.

Desventajas:

Menos Seguro: Es más susceptible a ataques XSS, especialmente si el contenido HTML incluye datos no confiables.

Rendimiento: Puede causar reflujo y repintado innecesarios en el navegador, ya que toda la estructura interna del elemento se recrea.

Difícil Manipulación Posterior: No es tan conveniente para agregar event listeners directamente a los elementos creados de esta manera.

## **CLASE NRO. 15 – FUNCIONES. ENTIDADES DE PRIMERA CLASE**

Antes de introducirnos en las técnicas que disponemos en JavaScript para manejar el SINCRONISMO / ASINCRONISMO y como estas están directamente vinculadas a funciones, es más las tres técnicas que veremos están directamente vinculadas al concepto de funciones a continuación veremos y repasaremos algunas particularidades que tienen las funciones, observaremos toda la potencia que se puede conseguir con las funciones.

Para ello es importante e imprescindible comprender los conceptos de entidades de primera clase y las particularidades de las mismas.

Las entidades de primera clase en JavaScript son aquellas que pueden:

- ✓ Ser asignadas a una variable ó constante.
- ✓ Ser pasadas como argumentos a funciones.
- ✓ Ser devueltas como resultado de funciones.
- ✓ Ser almacenadas en estructuras de datos.

Las funciones son entidades de primera clase porque se pueden hacer las siguientes operaciones o particularidades.

- **Particularidad 1:** Se pueden declarar de forma clásica

```
function Cuadrado (parametro)
{
    return parametro * parametro;
}
```

- **Particularidad 2:** Se pueden guardar dentro de constantes o variables

```
const Cuadrado = function(parametro)
{
    return (parametro * parametro);
}
```

- **Particularidad 3:** Se pueden declarar como arrow functions y guardarlas en constantes

```
const Cuadrado2 = (parametro)=>{
    return parametro * parametro;
}
```

- **Particularidad 4:** Se pueden declarar como arrow functions y si solo reciben un parámetro y es una sola operación que se realiza con el parámetro no es necesario que lleven () para el parámetro, no necesitan que lleven las {} para el cuerpo de la función y no es necesario que lleve la palabra return.

```
const Cuadrado3 = parametro => parametro * parametro;
```

- **Particularidad 5:** Funciones pueden ser pasadas como argumento ó parámetro de otra función por ende hay dos funciones. una podríamos decirle función llamadora y la otra le podríamos decir función interna.

```
// esta es la función llamadora que recibe como parámetro una función que se llamara interna //
```

```
const funcionLlamadora = (fnInterna)=>
{

    // aquí invocamos a la función interna //
    fnInterna();

    // esto sería el código de la función llamada //
    console.log("esto es código de la función Llamadora");
}
```

- **Particularidad 6:** Funciones que retornan otras funciones.

```
const funcionRetornaOtraFuncion = ()=>
{
  return (parametro)=>
  {
    return parametro * parametro;
  }
}
```

- **Particularidad 7:** El resultado de funciones puede ser pasado como parámetro de otras funciones.

```
function Cubo(numero)
{
  return (numero * numero * numero);
}
```

```
let ResultadoCuadrado = Cuadrado(5);
```

```
let Resultado2 = Cubo(ResultadoCuadrado); // aqui tendría la potencia 2 de 5;
```

```
console.log(Resultado2);//aqui tendría la potencia 5 de 5
```

- **Particularidad 8:** se pueden establecer valores por defecto en los parámetros de entrada de una función.

```
function Promedio(num1=0,num2=0,num3=0)
{
  return (num1 + num2 + num3)/3;
}
```



- **Particularidad 9:** pueden ser almacenadas y ser parte de cualquier estructura de datos.

```
class CalculosEstadisticos
{
  constructor(num1,num2)
  {
    this.Numero1 = num1;
    this.Numero2 = num2;
  }

  // es una función que está dentro de una clase = estructura.
  //pero aquí por convención
  // les decimos que se llaman métodos.
  devolverMediaAritmetica()
  {
    return (this.Numero1 + this.Numero2 )/2;
  }
}
```

- **Particularidad 10:** funciones pueden recibir como parámetro otras funciones, que pueden ser invocadas dentro de la función llamadora podríamos decir, podríamos definir función parámetro a la función que se pasa como parámetro y función llamadora o (callback) a la función que llama a la primera. Entonces quedaría algo así.

```
// esta es una función principal que llama a otras dos //
const fnLLamadoraOCallBack = function(f1,f2)
{
  f1();
  f2();
}

// en la invocación de la función principal le paso dos funciones
// que se acaban de crear y se envían en el parámetro

fnLLamadoraOCallBack(=>{
  console.log("esto es el cuerpo de la primera callback");
},
(
  =>{
    console.log("este es el cuerpo de la segunda callback");
  });
```

**Particularidades a modo de comentario:**

**Particularidad 11:** Funciones Anónimas Autoejecutables (IIFE): Estas funciones se ejecutan inmediatamente después de ser definidas.

```
(function() {  
    console.log('IIFE ejecutada');  
})();
```

**Particularidad 12:** Funciones Generadoras: Las funciones generadoras (function\*) permiten pausar y reanudar la ejecución del

```
function* generador() {  
    yield 1;  
    yield 2;  
    yield 3;  
}  
  
const gen = generador();  
console.log(gen.next().value); // 1  
console.log(gen.next().value); // 2  
console.log(gen.next().value); // 3
```

**Particularidad 13:** Funciones Asíncronas (async/await): Las funciones asíncronas permiten escribir código asíncronico de manera más sencilla.

```
async function fetchData()  
{  
    const response = await fetch('https://api.example.com/data');  
    const data = await response.json();  
    console.log(data);  
}  
  
fetchData();
```

**Particularidad 14:** Closures. Una función puede acceder a las variables de su contexto exterior incluso después de que ese contexto haya terminado.

```
function crearContador() {  
    let contador = 0;  
    return function() {  
        contador++;  
        return contador;  
    }  
}
```

```
}
```

```
const contador = crearContador();  
console.log(contador()); // 1  
console.log(contador()); // 2
```

**Particularidad 15:** Funciones con Rest Parameters: Permiten a una función aceptar un número indefinido de argumentos como un array.

```
function sumar(...numeros) {  
  return numeros.reduce((acc, num) => acc + num, 0);  
}
```

```
console.log(sumar(1, 2, 3)); // 6
```

**Particularidad 16:** Funciones con Spread Operator: Permiten expandir elementos de un iterable (como un array) en lugares donde se esperan cero o más argumentos.

```
function sumar(x, y, z) {  
  return x + y + z;  
}
```

```
const numeros = [1, 2, 3];  
console.log(sumar(...numeros)); // 6
```

### Importancia de las Entidades de Primera Clase:

Tener entidades de primera clase en un lenguaje de programación permite una mayor flexibilidad y modularidad. Se puede crear código más dinámico y reutilizable, y se pueden aplicar patrones de diseño más avanzados.

Por ejemplo, el hecho de que las funciones sean de primera clase en JavaScript permite la programación funcional, donde se pueden crear funciones de orden superior (funciones que toman otras funciones como argumentos o que devuelven funciones). Esto también facilita el uso de callbacks y promesas para manejar la asincronía, lo que es una parte fundamental del desarrollo moderno en JavaScript.

## **CAPITULO NRO. 15 – ASINCRONISMO EN JAVASCRIPT**

**Tipo de Clase:** Teórico Práctica

**Duración:** 120 minutos

**Objetivo de la clase:**

Al finalizar el presente capítulo, los alumnos deberán comprender que JavaScript es un lenguaje básicamente asíncrono, la forma de trabajar del Navegador es totalmente asíncrona. En determinadas ocasiones necesitamos tener un sincronismo en esta ejecución y terminación de los procesos, es por ello que en este capítulo explicaremos el funcionamiento de los procesos y como lograr un sincronismo, esto significa que si ejecuto un proceso1 la ejecución debe esperar a que termine el proceso1 para continuar con el proceso2 y este debe esperar para continuar con un tercer proceso por dar un ejemplo.

Existen tres formas o técnicas con las que se pueden conseguir que los procesos sean síncronos y para ello veremos las funciones CallBack, las Promesas y los procesos Async await.

**Introducción al concepto de Asincronismo:**

el asincronismo en JavaScript es un concepto fundamental que permite que el código se ejecute de manera no secuencial, permitiendo que ciertas operaciones se realicen mientras otras están en progreso. Esto es especialmente útil en el contexto de la programación web, donde es común realizar tareas que pueden tomar tiempo, como la obtención de datos de un servidor, sin bloquear la ejecución del resto del código.

Diferencias entre procesos síncronos y asíncronos

**Síncrono:** Las tareas se ejecutan secuencialmente. Cada tarea debe completarse antes de que la siguiente pueda comenzar.

**Asíncrono:** Las tareas pueden comenzar y completar en diferentes momentos, sin esperar a que otras tareas terminen.

**Event Loop:** Es el mecanismo que maneja las operaciones asíncronas en JavaScript. Monitorea la cola de tareas y ejecuta las que están listas cuando el stack de ejecución está vacío.

**Eventos y el Event Loop:**

JavaScript utiliza un único hilo de ejecución, pero puede manejar múltiples operaciones de manera no bloqueante gracias al event loop. El event loop es un mecanismo que permite que el JavaScript maneje operaciones asíncronas al sacar elementos de la cola de tareas y ejecutarlos cuando la pila de llamadas está vacía.

Que cosas son síncronas en JavaScript

- estructuras condicionales
- estructuras repetitivas
- Operaciones de comparación

Que cosas son asíncronas en JavaScript

- Los Eventos, por ejemplo "click", "load", etc. Y todo lo que se ejecute dentro de esos eventos es totalmente asíncrono.

Los eventos en JavaScript, especialmente en el contexto del navegador web, son inherentemente asíncronos. Cuando ocurre un evento, como hacer clic en un botón o cargar una página, el navegador coloca ese evento en la cola de eventos y continúa ejecutando el código. El código relacionado con ese evento se ejecutará más tarde, cuando el navegador decida manejar ese evento de la cola.

Por lo tanto, los eventos son asíncronos en el sentido de que el código relacionado con ellos se ejecuta en un momento posterior, después de que haya ocurrido el evento y el navegador decida manejarlo. Esta naturaleza asíncrona de los eventos es fundamental para construir aplicaciones web interactivas y receptivas.

Podríamos decir que JavaScript, especialmente en el entorno del navegador, tiende a ser asíncrono por defecto debido a su naturaleza orientada a eventos. Esto significa que las operaciones en JavaScript no bloquean el hilo principal de ejecución, lo que permite que otras tareas se ejecuten mientras se esperan ciertas operaciones, como las operaciones de red o de temporización.

Para manejar esta asincronía y controlar el flujo de ejecución de manera más eficiente, JavaScript ofrece diferentes mecanismos:

**Callbacks:** Los callbacks son funciones que se pasan como argumentos a otras funciones y se ejecutan cuando se completa una tarea, como una operación de red o temporización.

**Promesas:** Las promesas son objetos que representan el resultado eventual (éxito o fracaso) de una operación asíncrona. Permiten un manejo más flexible y legible de las operaciones asíncronas y evitan el "callback hell".

**Async/await:** La combinación de las palabras clave `async` y `await` permite escribir código asíncrono de manera más similar a la sincrónica. Las funciones `async` devuelven promesas y el operador `await` pausa la ejecución de una función `async` hasta que una promesa sea resuelta.

Estos mecanismos proporcionan formas de gestionar la asincronía y, cuando es necesario, forzar cierto grado de sincronismo en el flujo de ejecución de JavaScript. Permiten escribir código más claro, legible y mantenible al manejar operaciones asíncronas de manera más estructurada y fácil de seguir.

El asincronismo en JavaScript es una característica clave que permite que las operaciones no bloqueen la ejecución del programa, haciendo posible la realización de múltiples tareas al mismo

tiempo. Aquí hay un resumen de los conceptos más importantes relacionados con el asincronismo en JavaScript.

## Técnicas y Herramientas para el Asincronismo

### 1 – LAS FUNCIONES CALLBACK

Las funciones callback son una característica fundamental en JavaScript que permite manejar la asincronía y la ejecución de código dependiente de eventos o procesos asíncronos

#### ¿Qué es una función callback?

Una función callback es simplemente una función que se pasa como argumento a otra función y se invoca después de que cierto proceso o evento ha ocurrido. Es una forma de asegurarse de que cierto código se ejecute únicamente cuando una tarea asíncrona haya sido completada o cuando ocurra algún evento.

#### Características principales:

**Argumento de función:** En JavaScript, las funciones son de primera clase, lo que significa que pueden ser tratadas como cualquier otro tipo de dato, incluyendo ser pasadas como argumentos a otras funciones.

**Asincronía:** Las funciones callback son esenciales para manejar operaciones asíncronas como las que involucran peticiones HTTP, temporizadores (setTimeout, setInterval), manejo de eventos (click, load, submit, etc.), entre otros.

#### Uso común:

**Eventos de usuario:** Por ejemplo, en una página web, una función callback puede ser utilizada para manejar la respuesta después de que el usuario haga clic en un botón.

**Peticiones HTTP:** Al realizar una solicitud a un servidor, se puede proporcionar una función callback que procese la respuesta una vez que se reciba.

Ejemplos:

**Ejemplo 01 – Dos procesos Asíncronos Independientes:** En este ejemplo visualizaremos la problemática de lanzar dos procesos ASÍNCRONOS de forma simultánea para observar como terminan ambos en momentos diferentes no siguiendo la línea de ejecución

**Ejemplo 02 – Funciones CallBack:** En este segundo ejemplo, mostraremos como sería una solución simple al problema del asincronismo y para ello recurrimos a una función callback que es llamar una función dentro de la otra. Cuando el primer proceso haya terminado.

**Ejemplo 03 – Funciones CallBack II:** En el tercer ejemplo, adaptaremos la función llamadora para recibir como parámetro cualquier función que se desea ejecutar posteriormente. es decir, la función

que inicia el proceso recibe como parámetro la siguiente función a ejecutar, es más flexible que el ejemplo anterior.

**Ejemplo 04 – Funciones CallBack III:** En este cuarto ejemplo, la función llamadora recibe como parámetro la próxima función a ejecutar, pero el cambio más importante es mostrar que la función callback al momento de ser invocada, puede definirse como una arrow function, como una función anónima.

**Ejemplo 05 – Funciones CallBack IV:** En este quinto ejemplo, la función principal que es la que inicia el proceso, no tan solo llama a la función CALLBACK sino también le pasa resultados.

## **2 – PROMESAS = Promis**

Las promesas son objetos en JavaScript que representan la eventual finalización o el fracaso de una operación asíncrona. Proporcionan una forma más estructurada y flexible de trabajar con código asíncrono en comparación con las funciones callback tradicionales.

### **Características principales:**

**Estado:** Una promesa puede estar en uno de tres estados: pendiente, cumplida (resuelta) o rechazada.

**Encadenamiento:** Permite encadenar operaciones asíncronas de manera más legible y mantenible utilizando los métodos `.then()` y `.catch()`.

**Manejo de errores:** Proporciona un manejo más robusto de errores a través del método `.catch()` para capturar cualquier excepción que ocurra durante la ejecución de la promesa.

### **Uso común:**

**Peticiones HTTP:** Al realizar solicitudes a servidores, las promesas son utilizadas para manejar la respuesta de manera asíncrona.

**Carga de archivos:** Cuando se carga un archivo de manera asíncrona, una promesa puede manejar la finalización o el error de esa operación.

**Animaciones y temporizadores:** Pueden utilizarse para manejar animaciones y esperas asíncronas en interfaces de usuario.

### **Beneficios:**

**Claridad:** Facilitan la escritura y lectura de código asíncrono, reduciendo la anidación de callbacks y mejorando la estructura del código.

**Manejo de errores:** Ofrecen un método centralizado para manejar errores, lo que mejora la robustez del código y facilita la depuración.

**Consideraciones:**

Compatibilidad: Las promesas son compatibles con la mayoría de los navegadores modernos, pero es recomendable verificar la compatibilidad en entornos específicos si se requiere soporte para versiones más antiguas.

**Encadenamiento adecuado:** Para evitar el callback hell, es importante utilizar el encadenamiento de promesas de manera efectiva y utilizar `async/await` para estructuras más complejas.

A continuación, daremos varios ejemplos

**Ejemplo 07 – Promesas – Ejemplo I:** En este simple ejemplo mostraremos como utilizar e instanciar el objeto promesa a partir de la clase `PROMESA`, como guardarlo en una constante y posteriormente capturar el caso de éxito (“`resolve`”) y el caso de fallo (“`reject`”). El objetivo de este ejemplo es observar cómo se instancia la promesa y como se capturan sus casos de éxito o fracaso.

**Ejemplo 08 – Promesas – Ejemplo II:** En este ejemplo, en lugar de instanciar un objeto Promesa y guardarlo en una constante, lo que realizaremos es construir una función que retornará una promesa ya instanciada, esta es la forma más utilizada de cómo implementar las promesas, y tiene como objetivo organizar mejor una cadena de promesas enganchadas unas de otras.

**Ejemplo 09 – Promesas – Ejemplo III:** En este ejemplo, la función que retorna la promesa instanciada, tiene la flexibilidad de recibir como parámetro la función a ejecutar dentro de la promesa.

**Ejemplo 10 – Promesas – Ejemplo IV:** En este ejemplo, la función que retorna la promesa instanciada, se utiliza para enganchar dos procesos asíncronos que se desean ejecutar de forma síncrona. Por el lado del `.then()` de la primera promesa, se debe colocar siempre, la palabra **RETURN** e invocar nuevamente la promesa siguiente.

### 3 – Async & Await

La palabra clave `async`:

se usa para declarar una función asíncrona. Una función marcada con `async` siempre devolverá una promesa.

`await`

La palabra clave `await` se usa dentro de las funciones asíncronas para esperar a que una promesa se resuelva. Cuando `await` se encuentra una promesa, la ejecución de la función se pausa hasta que la promesa se resuelve, y luego continúa con el valor resuelto.

Beneficios de `async` y `await`

Legibilidad mejorada: El código se lee de manera más secuencial y lógica, similar al código síncrono tradicional.



Manejo simplificado de errores: Se utiliza try...catch para manejar errores de manera más natural que con .then() y .catch(). Más fácil de depurar: El flujo de control es más predecible y fácil de rastrear, facilitando la depuración de problemas.

#### Consideraciones y Buenas Prácticas

Compatibilidad: async y await están ampliamente soportados en navegadores modernos y entornos Node.js, pero es importante verificar la compatibilidad si se necesita soporte para navegadores antiguos.

Uso con otras funciones asíncronas: async y await pueden usarse junto con Promise.all para manejar múltiples promesas concurrentemente.

#### **Ejemplo 11 – Promesas – Async / Await**

**CLASE NRO. 17: APIs + FETCH + JSON + TRY/CATCH**

**CLASE NRO. 18: ESTADO DE LA APLICACIÓN**

## **CLASE NRO. 19: PERSISTENCIA DE DATOS EN EL NAVEGADOR**

**Tipo de Clase:** Teórico Práctica

**Duración:** 120 minutos

**Objetivo de la clase:**

El objetivo de la presente clase es presentarle al estudiante las diferentes formas, alternativas, ventajas y desventajas de cada técnica existente para persistir datos temporalmente en el navegador, como una herramienta importante para mantener datos de sesión y datos de la aplicación. A continuación, presentaremos las tres formas existentes más utilizadas como lo son las cookies, localStorage y sessionStorage.

### **Introducción a la Persistencia de Datos en el Navegador**

La persistencia de datos en el navegador es una característica esencial para el desarrollo de aplicaciones web modernas. Permite almacenar información en el lado del cliente, lo que facilita la gestión de datos entre sesiones de usuario, la mejora de la experiencia de usuario y la reducción de la carga en el servidor. En el navegador, existen tres principales métodos de persistencia de datos:

**Cookies**

**localStorage**

**sessionStorage**

### **COOKIES**

Las cookies son pequeños fragmentos de datos que los navegadores web almacenan localmente en nombre de los sitios web. Estos datos son utilizados principalmente para recordar información específica sobre el usuario y mantener el estado de la sesión a lo largo de múltiples solicitudes HTTP. A continuación, se detallan los puntos clave sobre las cookies:

**Propósito y Funcionalidad:**

**Persistencia de Datos:** Las cookies permiten a los sitios web almacenar datos en el navegador del usuario, como preferencias de configuración, información de inicio de sesión y detalles de la sesión actual.

**Seguimiento de Sesiones:** Son fundamentales para mantener el estado de la sesión entre múltiples solicitudes HTTP, lo que permite a los sitios web recordar la actividad del usuario entre páginas.

**Personalización del Contenido:** Pueden utilizarse para adaptar la experiencia del usuario según su historial de navegación y preferencias.

### Estructura de una Cookie:

**Nombre y Valor:** Cada cookie tiene un nombre que actúa como identificador y un valor asociado que contiene los datos almacenados.

**Atributos Opcionales:** Pueden incluir atributos como la fecha de expiración (**expires** o **max-age**), el dominio (**domain**), la ruta (**path**), la seguridad (**secure** y **httpOnly**) y restricciones de intercambio (**sameSite**).

### Creación y Gestión:

**Creación:** Se crean desde el servidor web y se envían al navegador del cliente mediante cabeceras HTTP. También se pueden establecer desde el lado del cliente utilizando JavaScript.

**Acceso y Modificación:** Pueden ser accedidas y modificadas tanto desde el lado del cliente (JavaScript en el navegador) como desde el servidor web, dependiendo de la necesidad y configuración de seguridad.

**Envío Automático:** Las cookies se envían automáticamente al servidor web en cada solicitud HTTP que cumpla con los criterios definidos por la cookie (dominio y ruta).

### Seguridad y Privacidad:

**Atributos de Seguridad:** Los atributos **secure** y **httpOnly** ayudan a proteger las cookies de accesos no autorizados y ataques de scripts maliciosos.

**SameSite:** Este atributo controla si las cookies deben enviarse en solicitudes de origen cruzado, mejorando la seguridad al prevenir la exposición a sitios no confiables.

### Expiración y Eliminación:

**Expiración:** Se puede configurar una fecha de expiración para que la cookie sea válida solo hasta ese momento (**expires** o **max-age**).

**Eliminación:** Las cookies pueden ser eliminadas por el navegador automáticamente al expirar, o pueden ser eliminadas manualmente por el usuario a través de la configuración del navegador.

### Uso en Práctica:

**Autenticación y Sesiones:** Ampliamente utilizado para autenticación de usuarios y mantener el estado de sesión activo.

**Seguimiento de Usuarios:** Se utilizan para análisis y seguimiento de usuarios en sitios web, aunque ahora regulado por leyes de privacidad como GDPR y CCPA.

En resumen, las cookies son herramientas esenciales para la personalización y la gestión del estado de sesión en aplicaciones web modernas. Sin embargo, es crucial utilizarlas de manera responsable

y en cumplimiento con las regulaciones de privacidad para proteger los datos del usuario y garantizar una experiencia web segura y confiable.

## localStorage

### Características:

**Persistencia:** Los datos almacenados en `localStorage` persisten incluso después de cerrar el navegador o reiniciar el dispositivo.

**Almacenamiento por dominio:** Los datos son específicos del dominio y protocolo (http o https). Esto significa que los datos almacenados por un dominio no están accesibles para otros dominios.

**Capacidad de almacenamiento:** `localStorage` puede almacenar hasta 5-10 MB de datos por dominio, aunque esto puede variar entre navegadores.

**Formato de datos:** Los datos se almacenan en pares clave-valor como cadenas de texto (strings). Si se necesita almacenar objetos, estos deben ser convertidos a JSON usando `JSON.stringify()` y `JSON.parse()`.

### // Almacenar un dato

```
localStorage.setItem('key', 'value');
```

### // Recuperar un dato

```
let value = localStorage.getItem('key');
```

### // Eliminar un dato

```
localStorage.removeItem('key');
```

### // Limpiar todos los datos

```
localStorage.clear();
```

### Aislamiento de datos:

`localStorage` está aislado por dominio y protocolo. Esto significa que los datos almacenados en <https://example.com> no son accesibles desde <http://example.com> o <https://sub.example.com>.

## sessionStorage

### Características:

**Persistencia:** Los datos en `sessionStorage` solo persisten durante la sesión del navegador. Se borran cuando la pestaña o ventana del navegador se cierra.

**Almacenamiento por dominio:** Similar a `localStorage`, los datos son específicos del dominio y protocolo.

**Capacidad de almacenamiento:** Generalmente, `sessionStorage` tiene la misma capacidad de almacenamiento que `localStorage`, pero puede variar entre navegadores.

**Formato de datos:** Los datos se almacenan en pares clave-valor como cadenas de texto.

### // Almacenar un dato

```
sessionStorage.setItem('key', 'value');
```

### // Recuperar un dato

```
let value = sessionStorage.getItem('key');
```

### // Eliminar un dato

```
sessionStorage.removeItem('key');
```

### // Limpiar todos los datos

```
sessionStorage.clear();
```

### // Almacenar un valor temporal

```
sessionStorage.setItem('temp', '12345');
```

### // Recuperar un valor temporal

```
let tempValue = sessionStorage.getItem('temp');
```

```
console.log(tempValue); // 12345
```

## Comparación con Cookies

### Cookies:

**Persistencia:** Las cookies pueden tener una fecha de expiración específica o ser temporales (se eliminan cuando se cierra el navegador).

**Almacenamiento por dominio:** Las cookies son específicas del dominio y pueden ser accesibles en subdominios si se especifica.

**Capacidad de almacenamiento:** Las cookies tienen un límite de tamaño más pequeño, típicamente 4KB por cookie.

**Formato de datos:** Las cookies almacenan datos en pares clave-valor.

**Envío al servidor:** Las cookies se envían automáticamente al servidor en cada solicitud HTTP, lo que no ocurre con `localStorage` o `sessionStorage`.

**MUY IMPORTANTE, MUY IMPORTANTE, MUY IMPORTANTE ¡!!.**

### localStorage

**Persistencia y Alcance:** Los datos almacenados en `localStorage` son compartidos entre todas las pestañas y ventanas del mismo navegador que cargan el mismo origen (mismo protocolo, host y puerto).

**Independencia de Pestañas:** No son independientes entre pestañas. Si guardas, actualizas o eliminas un dato en `localStorage` en una pestaña, estos cambios se reflejarán inmediatamente en todas las otras pestañas abiertas con la misma aplicación web.

### sessionStorage

**Persistencia y Alcance:** Los datos almacenados en `sessionStorage` son específicos de la pestaña o ventana donde fueron creados. Persisten solo durante la sesión de la página y se eliminan al cerrar la pestaña o ventana.

**Independencia de Pestañas:** Son independientes entre pestañas. Cada pestaña tiene su propio `sessionStorage` separado. Si guardas, actualizas o eliminas un dato en `sessionStorage` en una pestaña, estos cambios no afectarán a las otras pestañas abiertas con la misma aplicación.

### **Cookies**

**Independencia de Pestañas:** Las cookies son compartidas entre todas las pestañas y ventanas del mismo navegador que están abiertas en el mismo dominio.

Esto significa que cualquier modificación a las cookies en una pestaña afecta a todas las demás pestañas y ventanas abiertas bajo el mismo dominio.