WEB I

Organización de archivos

- En la **raíz:** HTML
- Carpeta **CSS:** Hojas de estilo y elementos que use el css
- Carpeta **Images:** imágenes
- Carpeta **JS:**Los códigos JS

Rutas a archivos (paths)

Las rutas (se usan en propiedades href y src entre otros) **deben ser siempre relativas**.
Las rutas **absolutas** empiezan con "/". No es recomendable, ya que solo van a funcionar bien en el servidor.

HTML5

El nuevo estándar fue pensado para que la estructura de una web sea más semántica. Se agregaron nuevos elementos 'estructurales'.
● Header <header> … </header>
● Footer <footer> … </footer>
● Nav <nav> ... </nav>
● Section <section> … </section>
● Article <article> … </article>
● Aside <aside> … </aside>

HTML5 elimina bastante restricciones y limitaciones. Es más ligero al ser más sencillo y simple el código
● Cargan más rápido en el navegador
Introduce:
● Contenido Semántico <header>, <footer>, <article>, <section>
● Elementos multimedia: Video, Audio
● Elementos gráficos: <svg>, <canvas>

CSS (cascading style sheets)

Es un lenguaje de presentación creado para dar estilo al contenido
Selector Determina el estilo que se le va a aplicar a un elemento.
Propiedades El estilo que se aplica puede ser una o múltiples propiedades.
Valor Define el valor que toma la propiedad. Dependiendo de la propiedad el valor puede ser decimal / hexadecimal, incluir unidades o puede ser un texto.

CSS3 es el último estándar para CSS.

- Expande la funcionalidad para bordes, efectos de texto, imágenes de fondo, colores y degradados, transparencias, fuentes de texto.
- También incorpora la posibilidad de animar.

Herencia

La herencia en CSS es el mecanismo mediante el cual determinadas propiedades de un elemento padre se transmiten a sus hijos.
● No todas las propiedades CSS son heredadas, porque algunas de ellas no tendría sentido que lo fueran.

Por ejemplo, los relacionados al tamaño de los elementos.
Todos los elementos de un documento HTML heredan todas las propiedades heredables de su padre excepto el elemento raíz (html), que no tiene padre.
Sirve para escribir menos código.

Cascada

CSS significa cascading style sheets (hojas de estilo en cascada)
La cascada es el mecanismo que controla el resultado final cuando se aplican varias declaraciones CSS contrapuestas al mismo elemento.

Hay tres conceptos principales que controlan el orden en el que se aplican las declaraciones de CSS:
○ Importancia.
○ Especificidad.
○ Orden en el código fuente.
La palabra reservada **!important**, sobrescribe toda cascada.

Más específico es el selector, entonces ese es el estilo que se aplica.
● Lo que tiene clases es más específico que lo que no.
● En un selector anidado, si tienen la misma cantidad de clases, gana el que tiene más elementos para cumplir.

Si dos declaraciones afectan al mismo elemento, tienen la misma importancia y la misma especificidad, la selección final es el orden en las fuentes. **La declaración que se ve después** en las hojas de estilo "ganará" a las anteriores.

HTML	**CSS**

Este es un párrafo	p {
	color: red;
	}
	p {
	color: blue;
	}

Semántica WEB

En el contexto de la Web, la semántica, es la práctica de darle al contenido de una página: **Sentido y Estructura**. No solo se usa por cuestiones de estilo.

Box Model

Dice que cada elemento en una página se representa mediante una caja rectangular (contenedor). **Siempre es utilizado.** Fundamental para construir y diagramar sitios.Cada caja consta de 4 partes:

- CONTENT: ancho y alto del elemento
- PADDING: espaciado o margen INTERIOR transparente dentro de un elemento.
- BORDER: Se utiliza para bordear con una línea alrededor del elemento.
- MARGIN: Usado para generar margen EXTERIOR transparente fuera de un elemento.

Para calcular el alto total y ancho total del element sumamos todas estas partes (left, right, bottom y top)

Layouts

Define la estructura básica de la interfaz de usuario en una aplicación.
¿Cómo se empieza? Hacer un diagrama del layout en papel, lo más completo posible y con sus medidas (wireframe). Una vez que se tiene una idea clara del diseño que se desea lograr, comenzar a escribir código para ajustarlo al diseño.

Diseño responsive

Mobile first

@media only screen and (min-width: 680px) {...}

ECMAScript 6 (ES6)

ECMAScript (ES) es el estándar que define a JavaScript.
ES6 Aprobado en Junio de 2015. Fué el cambio más grande en Javascript en años. Cambios de sintaxis y estructura para un código más simple y compacto. Ya está soportado casi 100% por navegadores modernos.

VARIABLE SIMPLE (Guarda un solo dato)

Es un nombre que le damos a un valor que puede cambiar (o no) con el tiempo (durante la ejecución del programa). El nombre no es el contenido, es como llamamos a ese valor, pero sin saber el valor exacto mientras escribimos

ARREGLO

Almacenan elementos en una colección de datos ordenados. Son una variable que tiene muchas posiciones.

OBJETO

Es una combinación de:

- **Campos o atributos**: almacenan datos. Estos datos pueden ser de tipo primitivo y/o otro tipo de objeto
- **Rutinas o métodos**: lleva a cabo una determinada acción o tarea con los atributos.

JAVASCRIPT

JavaScript es un lenguaje de programación interpretado. Le otorga comportamiento a la pagina.

- Principal uso, del lado del cliente. Existe un intérprete javascript en cada máquina (navegadores)
- Por su popularidad se ha extendido a otras aplicaciones y entornos fuera de la web. (por ej. mobile apps)
- Existe también del lado del servidor (node.js)

Incluirlo al final del body, luego de que ya se cargo el html con todos sus elementos.
<script type=”text/javascript” src= “js/main.js”></script>

DOM

El Document Object Model es una API (Application Programming Interface) para documentos HTML y XML. El DOM es un árbol de objetos.

- Representación estructurada del documento
- Permite modificar el contenido
- Es lo que conecta las páginas web con Javascript.

Las propiedades del DOM (HTML) se pueden leer/editar desde Javascript.

Eventos

Un evento es algo que ocurre en el sistema, originado por el usuario o otra parte del sistema y que se avisa al sistema. Las interfaces gráficas suelen programarse orientada a eventos.

JSON

JSON es una forma de escribir objetos en JS. Los objetos que permiten organizar variables y funciones. Estos objetos encapsulan datos y comportamiento.
Su estructura está compuesta por campos y métodos.

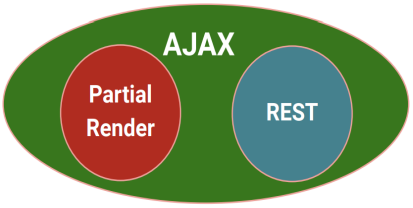
```
objeto = {
        "atributo1": "dato1",
        "atributo2": "dato2"
}
```

Un objeto es una forma de agrupar variables igual que un arreglo, pero un arreglo tienen que ser iguales, la cantidad puede cambiar, en un objeto le da nombre a cada una de esas subvariables

SPA (Single Page Application)

Es un tipo de aplicación web donde nunca se recarga completamente el navegador. Los recursos se cargan parcial y dinámicamente cuando lo requiera la página.

AJAX (Asynchronous Javascript And XML.)



Es una técnica que combina un set de tecnologías conocidas, para hacer mejorar la experiencia de usuario en las páginas web (más amigables y rápidas). No es un lenguaje o tecnología. Esta técnica es base de las aplicaciones web modernas.
Es una técnica de carga asíncrona de contenido dinámico y datos del server. Esta permite cambios dinámicos del lado del cliente y mejora la experiencia del usuario.

Partial render

Carga un fragmento HTML y lo inserta en nuestro html

Servicio REST

Consulta un objeto JSON y lo procesa del lado del cliente con Javascript

Promesas

Son un objeto que representa la terminación o el fracaso de una operación asíncrona. Es la forma de implementar el asincronismo en JS.

Orden de ejecución:

- Primero se hace la promesa
- Luego se sigue ejecutando
- Finalmente se ejecuta la resolución de la promesa

Promesas con Await/async

```
async function load2(event) {
	event.preventDefault();
	let container = document.querySelector("#use-ajax");  //DIV DONDE INSERTO EL HTML
	container.innerHTML = "<h1>Loading...</h1>";
	try {
		let response = await fetch(url); // url-> ej: “inicio.html”
		if (response.ok) {
			let data = await response.text()
			container.innerHTML = data;
		} else
			container.innerHTML = "<h1>Error - Failed URL!</h1>";
	} catch (error) {
		container.innerHTML = "<h1>Connection error</h1>";
	};
}
```

Método GET

```
//GET: Traer los objetos del mockapi
async function GET(){
    try {
        let response = await fetch(URL, {
            "method": "GET"
        }); //Get URL
        let data = await response.json();
        if (response.ok){
            console.log(data) //data va a ser el arreglo de objetos que traigo del servicio
            }
        }
    } catch (error) {
        console.log(error);
    }
}
```

Método POST

```
//POST: Subir al mockApi (solo el recurso)
async function POST(e){
    e.preventDefault();
    let dato1 = document.querySelector("#plin").value;
    let dato2 = document.querySelector("#plin").value;
    let objeto = {
        "variable": dato1,
        "variable": dato2
    }
    try {
        let response = await fetch(URL, {
            "method": 'POST',
            "headers": {
                'Content-Type': 'application/json'
            },
            "body": JSON.stringify(objeto)
        });
        let data = await response.json();
    } catch (error) {
        console.log(error);
    }
}
```

Método PUT

```
//PUT:Editar el mockapi (algo específico de la tabla)
async function PUT(id){ //id = boton.dataset.editarId
    let dato = document.querySelector("#plin").value;
    let dato = document.querySelector("#plin").value;
    let objeto = {
        "xxxx": dato,
        "xxxx": dato
    }
    try {
         let response = await fetch(`${URL}/${id}`, {
             "method": 'PUT',
             "headers": {
                  'Content-Type': 'application/json'
              },
              "body": JSON.stringify(objeto)
            });
            let data = await response.json();
        } catch {
            console.log("error")
        }
    } else{
        console.log("error")
    }
}
```

Método DELETE

```
//DELETE: Borrar un elemento específico del mockapi
async function DELETE(id){ //id = boton.dataset.borrarId
    try {
        let response = await fetch(`${URL}/${id}`, {
            "method":"DELETE"
        })
        let data = await response.json();
        document.getElementById(`${id}`).remove(); //Borra la fila con el id
    } catch {
        console.log("error")
    }
}
```