<html lang="es">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="CSS/style.css">

<title>INVESTIGACIÓN</title>

</head>

<body>

<div id="container">

<header id="header">

<div id="logo">

<h1>INVESTIGACIÓN</h1>

</div>

</header>

<section class="content">

<aside id="left" class="sticky-menu">

<div>

<ul>

<a href="#tilulo1"><li>Prototipos y herencias de prototipos</li></a>

<a href="#tilulo2"><li>POO</li></a>

<a href="#tilulo3"><li>Expresiones regulares</li></a>

<a href="#tilulo4"><li>Modulos: Import y export</li></a>

<a href="#tilulo5"><li>Settimeout y setinterval</li></a>

<a href="#tilulo6"><li>Operaciones sincrónicas y asincrónicas</li></a>

<a href="#tilulo7"><li>Callbacks y promesas </li></a>

<a href="#titulo8"><li>MSets y maps </li></a>

</ul>

</div>

</aside>

<div id="right">

<h2 id="tilulo1">Prototipos y herencias de prototipos</h2>

<p>

En el contexto de la programación y la orientación a objetos, los conceptos de "prototipos" y "herencia de prototipos" están relacionados con la forma en que algunos lenguajes de programación manejan la creación y estructuración de objetos y clases.<br>

<br>

Prototipos:<br>

En lugar de utilizar clases para crear objetos como en otros lenguajes orientados a objetos (como Java o C++), algunos lenguajes, como JavaScript, utilizan un modelo basado en prototipos. En este enfoque, en lugar de definir clases, se crean objetos prototipo y luego se crean nuevos objetos basados en estos prototipos.<br>

<br>

Herencia de prototipos:<br>

La herencia de prototipos es el mecanismo mediante el cual un objeto puede heredar propiedades y métodos de otro objeto prototipo. Cuando se busca una propiedad o método en un objeto, si no se encuentra en el propio objeto, el motor de JavaScript (en el caso de este lenguaje) busca en la cadena de prototipos hasta encontrar la propiedad o método en cuestión.<br>

<h2 id="tilulo2">POO</h2>

<p>

POO significa "Programación Orientada a Objetos". Es un paradigma de programación que se basa en la organización del código en torno a objetos y clases. Los objetos son instancias individuales que representan entidades con características y comportamientos específicos, mientras que las clases son plantillas que definen cómo se crean esos objetos y qué propiedades y métodos poseen.

</p>

<h2 id="tilulo3">Expresiones regulares</h2>

<p>

Las expresiones regulares, también conocidas como "regex" o "regexp", son secuencias de caracteres que definen un patrón de búsqueda. Se utilizan en el procesamiento de cadenas de texto para buscar y manipular patrones específicos de caracteres. Las expresiones regulares son una herramienta poderosa y flexible que permite realizar búsquedas, coincidencias y transformaciones complejas en cadenas de texto.

</p>

<h2 id="tilulo4">Modulos: Import y export</h2>

<p>

Los módulos, en el contexto de la programación, se refieren a unidades de código independientes que pueden ser reutilizadas y compartidas entre diferentes partes de una aplicación. Permiten una mejor organización y modularidad del código, lo que facilita el mantenimiento y la colaboración en proyectos más grandes.<br>

<br>

Exportar:<br>

La declaración `export` se utiliza para exponer elementos (variables, funciones, clases, etc.) desde un módulo para que puedan ser utilizados en otros módulos.<br>

<br>

Importar:<br>

La declaración `import` se utiliza para traer elementos exportados desde otros módulos y hacer uso de ellos en el módulo actual.<br>

</p>

<h2 id="tilulo5">Settimeout y setinterval</h2>

<p>

SetTimeout:<br>

La función `setTimeout` se utiliza para ejecutar una función después de un cierto período de tiempo (en milisegundos) ha transcurrido. Toma dos argumentos: la función que se ejecutará y el tiempo en milisegundos que se debe esperar antes de ejecutarla.<br>

<br>

<br>

SetInterval:<br>

La función `setInterval` se utiliza para ejecutar una función repetidamente en intervalos de tiempo especificados. Al igual que `setTimeout`, toma una función y un intervalo de tiempo en milisegundos como argumentos.<br>

</p>

<h2 id="tilulo6">Operaciones sincrónicas y asincrónicas</h2>

<p>

Operaciones Sincrónicas:<br>

Las operaciones sincrónicas son aquellas que se ejecutan en secuencia, una detrás de la otra. En otras palabras, una operación no comienza hasta que la operación anterior haya terminado. Esto significa que el programa espera a que cada operación se complete antes de pasar a la siguiente. Las operaciones sincrónicas son fáciles de entender y razonar, ya que el flujo del programa es lineal y predecible.<br>

<br>

<br>

Operaciones Asincrónicas:<br>

Las operaciones asincrónicas son aquellas que no bloquean el flujo del programa. En lugar de esperar a que una operación termine antes de pasar a la siguiente, se inician y se gestionan en segundo plano. Esto es especialmente útil cuando se trata de tareas que podrían llevar tiempo, como realizar llamadas a una API, leer un archivo grande o realizar operaciones de red.<br>

</p>

<h2 id="tilulo7">Callbacks y promesas </h2>

<p>

Callbacks:<br>

Un callback es una función que se pasa como argumento a otra función para que se llame en algún momento futuro, generalmente cuando se completa una operación asincrónica. Los callbacks son una forma común de manejar la programación asincrónica en JavaScript, pero a medida que las operaciones asincrónicas se vuelven más complejas, el código puede volverse difícil de leer y mantener debido a lo que se conoce como "callback hell" (anidación excesiva de callbacks).<br>

<br>

<br>

Promesas:<br>

Las promesas son una manera más estructurada y elegante de manejar la programación asincrónica. Una promesa representa un valor que puede estar disponible ahora, en el futuro o nunca. Proporciona una interfaz para trabajar con operaciones asíncronas de manera más legible y manejable, evitando la anidación excesiva de callbacks.

Las promesas también pueden encadenarse y ofrecen métodos como `.then()` y `.catch()` para manejar el flujo de control de manera más comprensible.<br>

</p>

<h2 id="titulo8">Sets y maps </h2>

<p>

Sets:<br>

Un Set es una colección de valores únicos. Esto significa que no puede contener duplicados, y cada valor solo puede aparecer una vez en el Set. Los Sets son útiles cuando necesitas almacenar una lista de elementos sin duplicados y no te importa su orden.

Los Sets tienen métodos para agregar, eliminar y verificar la presencia de elementos. Algunos de estos métodos incluyen `add()`, `delete()`, `has()` y `size`.<br>

<br>

<br>

Maps:<br>

Un Map es una colección de pares clave-valor, donde cada clave está asociada a un valor. A diferencia de los Sets, los Maps te permiten utilizar cualquier tipo de valor (objetos, funciones, etc.) como clave. Esto hace que los Maps sean útiles cuando necesitas una estructura para almacenar y acceder a información en base a una clave específica.

Los Maps tienen métodos para agregar, eliminar, obtener y verificar la presencia de elementos. Algunos de estos métodos incluyen `set()`, `get()`, `delete()` y `has()`.<br>

</p>

</div>

</section>

<footer>

<div id="logo">&copy;Sena</div>

<div id="f\_right">2023</div>

</footer>

</div>

<script>

window.addEventListener("scroll", function() {

var stickyMenu = document.querySelector(".sticky-menu");

var headerHeight = document.querySelector("#header").offsetHeight;

var scrollTop = window.scrollY;

if (scrollTop > headerHeight) {

stickyMenu.style.top = "0";

} else {

stickyMenu.style.top = (headerHeight - scrollTop) + "px";

}

});

window.onload = function() {

window.dispatchEvent(new Event('scroll'));

};

</script>

</body>

</html>