CHECKPOINT 7

¿Qué diferencia a Javascript de cualquier otro lenguaje de programación?

JavaScript es un lenguaje de programación único por varias razones que lo distinguen de otros lenguajes. Aquí algunas de las diferencias más destacadas:

1. **Lenguaje de programación interpretado y de alto nivel**: JavaScript es interpretado, lo que significa que no necesita ser compilado antes de ser ejecutado. Esto lo hace más flexible y fácil de probar y depurar. Además, es de alto nivel, lo que significa que está diseñado para ser fácil de entender y escribir para los humanos.
2. **Lenguaje orientado a eventos y asíncrono**: Una de las características más distintivas de JavaScript es su modelo de ejecución asíncrona y orientada a eventos. Esto significa que puede manejar múltiples tareas simultáneamente sin bloquear la ejecución del código. Por ejemplo, puede realizar operaciones de entrada/salida (E/S) como solicitudes de red o manipulación de archivos sin detener la ejecución del resto del programa.
3. **Amplio uso en el desarrollo web**: JavaScript es el lenguaje principal para el desarrollo de aplicaciones web interactivas. Se utiliza para agregar interactividad a las páginas web, como validar formularios, crear animaciones, manipular el DOM (Document Object Model) y responder a eventos del usuario.
4. **Soporte multiplataforma y de lado del cliente**: JavaScript se ejecuta en el navegador del cliente, lo que significa que no requiere ninguna configuración especial del servidor. Esto permite que las aplicaciones web construidas con JavaScript sean altamente portátiles y puedan ejecutarse en una amplia variedad de dispositivos y plataformas.
5. **Extensibilidad y ecosistema vibrante**: JavaScript cuenta con una gran cantidad de bibliotecas y marcos de trabajo que facilitan el desarrollo de aplicaciones complejas de manera eficiente. Frameworks como React, Angular y Vue.js son ampliamente utilizados en el desarrollo web moderno.
6. **Tipado dinámico y débil**: A diferencia de otros lenguajes que son tipados estática o fuertemente tipados, JavaScript es dinámico y débilmente tipado. Esto significa que las variables no están asociadas a ningún tipo específico y pueden cambiar de tipo durante la ejecución del programa. Por ejemplo, una variable que contiene un número entero puede cambiar para almacenar una cadena más tarde.

En resumen, JavaScript es único por su naturaleza interpretada, orientada a eventos, amplio uso en el desarrollo web, soporte multiplataforma, extensibilidad y tipado dinámico. Estas características lo convierten en una herramienta poderosa y versátil para el desarrollo de aplicaciones web modernas.

¿Cuáles son algunos tipos de datos JS?

JavaScript es un lenguaje de programación dinámico que admite varios tipos de datos

para almacenar y manipular información. Aquí tienes algunos de los tipos de datos más comunes en JavaScript:

1. **Número (Number)**: Representa valores numéricos, ya sean enteros o de punto flotante. Por ejemplo:

```
let edad = 25;

let precio = 10.99;
```

2. **Cadena de texto (String)**: Representa una secuencia de caracteres encerrados entre comillas simples o dobles. Por ejemplo:

```
let nombre = "Juan";
let mensaje = '¡Hola mundo!';
```

3. **Booleano (Boolean)**: Representa un valor verdadero (true) o falso (false). Se utiliza para evaluar condiciones en las estructuras de control. Por ejemplo:

```
let esMayorDeEdad = true;
let tieneDescuento = false;
```

4. **Arreglo (Array)**: Representa una colección ordenada de elementos. Los elementos pueden ser de cualquier tipo de datos, incluidos otros arreglos. Por ejemplo:

```
let numeros = [1, 2, 3, 4, 5];
let colores = ["rojo", "verde", "azul"];
```

5. **Objeto (Object)**: Representa una colección de pares clave-valor. Cada valor se accede mediante su clave correspondiente. Por ejemplo:

```
let persona = {
   nombre: "María",
   edad: 30,
   ciudad: "Madrid"
};
```

6. **Undefined**: Representa un valor indefinido. Se asigna automáticamente a las variables que no han sido inicializadas o a las funciones que no tienen una declaración de retorno. Por ejemplo:

```
let direccion;
```

7. **Null**: Representa la ausencia intencional de cualquier valor o referencia a un objeto. Por ejemplo:

```
let resultado = null;
```

8. **Símbolo (Symbol)**: Introducido en ECMAScript 6, representa un identificador único e inmutable. Se utiliza principalmente como propiedades de objetos para evitar colisiones de nombres. Por ejemplo:

```
const simbolo = Symbol('descripcion');
```

Estos son algunos de los tipos de datos más utilizados en JavaScript. Comprender cómo trabajar con ellos es fundamental para desarrollar aplicaciones web eficientes y sólidas.

¿Cuáles son las tres funciones de String en JS?

En JavaScript, las cadenas de texto (strings) son primitivas que representan secuencias de caracteres. Existen muchas funciones integradas en JavaScript que pueden utilizarse para manipular cadenas de texto. Aquí tienes tres funciones de cadena comunes:

1. **length**: Esta propiedad devuelve la longitud de una cadena de texto, es decir, el número de caracteres que contiene. Por ejemplo:

```
let mensaje = "¡Hola mundo!";

console.log(mensaje.length); // Output: 12
```

2. **toUpperCase()**: Este método devuelve una nueva cadena de texto con todos los caracteres convertidos a mayúsculas. Por ejemplo:

```
let nombre = "juan";
console.log(nombre.toUpperCase()); // Output: JUAN
```

3. **toLowerCase()**: Este método devuelve una nueva cadena de texto con todos los caracteres convertidos a minúsculas. Por ejemplo:

```
let apellido = "PEREZ";
console.log(apellido.toLowerCase()); // Output: perez
```

Estas son solo tres de las muchas funciones y métodos disponibles para manipular cadenas de texto en JavaScript. Las cadenas de texto son muy versátiles y es posible realizar una amplia variedad de operaciones con ellas, como concatenación, búsqueda, reemplazo, entre otras.

¿Qué es un condicional?

Un condicional es una estructura de control en programación que permite ejecutar cierto bloque de código si se cumple una condición específica. En esencia, los condicionales permiten que un programa tome decisiones en función de diferentes situaciones.

En JavaScript, al igual que en muchos otros lenguajes de programación, el condicional más común es la declaración **if**. Este es un ejemplo básico de cómo funciona un condicional **if**:

```
let edad = 20;

if (edad >= 18) {

   console.log("Eres mayor de edad.");

}
```

En este ejemplo, la condición que se evalúa es **edad >= 18**, lo que significa "si la variable **edad** es mayor o igual a 18". Si esa condición se cumple (es decir, es verdadera), se ejecuta el bloque de código dentro de las llaves {} que sigue a la declaración **if**, y se imprime en la consola el mensaje "Eres mayor de edad.".

Los condicionales pueden tener otras formas, como **if...else**, y operadores ternarios, que permiten manejar múltiples casos de manera más compleja.

En resumen, un condicional en programación es una herramienta esencial que permite ejecutar diferentes fragmentos de código según se cumplan ciertas condiciones, lo que permite que los programas se adapten dinámicamente a diferentes situaciones.

¿Qué es un operador ternario?

Un operador ternario, también conocido como operador condicional, es un tipo especial de operador que toma tres operandos y se utiliza para realizar una evaluación condicional de manera concisa en una sola línea de código.

En JavaScript, el operador ternario tiene la siguiente sintaxis:

```
condición ? expresiónSiVerdadero : expresiónSiFalso
```

Donde:

- **condición** es la expresión que se evalúa.
- **expresiónSiVerdadero** es el valor que se devuelve si la condición es verdadera.
- **expresiónSiFalso** es el valor que se devuelve si la condición es falsa.

Por ejemplo, considera el siguiente código que utiliza un operador ternario para determinar si una persona es mayor o menor de edad:

```
let edad = 20;
```

```
let mensaje = (edad >= 18)  "Eres mayor de edad." : "Eres menor de edad.";

console.log(mensaje); // Output: Eres mayor de edad.
```

En este caso, la condición **edad >= 18** se evalúa como verdadera, por lo que se devuelve el valor de **expresiónSiVerdadero**, que es "Eres mayor de edad.". Si la condición fuera falsa, se devolvería el valor de **expresiónSiFalso**, que en este caso sería "Eres menor de edad.".

El uso de operadores ternarios puede hacer que el código sea más conciso y legible, especialmente para evaluaciones simples. Sin embargo, es importante usarlos con moderación para no sacrificar la claridad del código en favor de la brevedad.

¿Cuál es la diferencia entre una declaración de función y una expresión de función?

La diferencia principal entre una declaración de función y una expresión de función en JavaScript radica en cómo son definidas y cómo se comportan en el código.

1. **Declaración de función (Function Declaration)**:
   - Se define utilizando la palabra clave **function** seguida por el nombre de la función y su cuerpo.
   - Puede ser invocada antes de su declaración, debido al concepto de elevación (hoisting) en JavaScript. Esto significa que una función declarada puede ser llamada en el código antes de que se haya declarado formalmente.
   - Puede ser utilizada como un bloque de código independiente y su alcance está dentro del contexto en el que se define.

Ejemplo de declaración de función:

```
function suma(a, b) {
   return a + b;
}
```

2. **Expresión de función (Function Expression)**:
   - Se define mediante una asignación de función a una variable (o a cualquier otro identificador).
   - No puede ser invocada antes de su declaración, ya que no está sujeta al hoisting. Debe ser definida antes de ser invocada en el código.
   - Puede ser anónima o tener un nombre, y su alcance está determinado por el contexto en el que se asigna.

Ejemplo de expresión de función:

```
let resta = function(a, b) {
   return a - b;
};
```

En resumen, una declaración de función se define con la palabra clave **function** y se puede invocar antes de su declaración debido al hoisting. Mientras que una expresión de función se define mediante asignación y no puede ser invocada antes de su declaración. Ambas formas tienen sus usos específicos en diferentes situaciones de programación

¿Qué es la palabra clave "this" en JS?

La palabra clave **this** en JavaScript es una variable especial que se refiere al objeto al que pertenece el contexto actual. El valor de **this** depende de cómo se llama la función y de dónde se utiliza.

Aquí hay algunas situaciones comunes en las que se utiliza **this**:

1. **En el contexto global**: Cuando **this** se utiliza fuera de cualquier función, se refiere al objeto global en el navegador (como **window** en el navegador web).

```
console.log(this === window); // Output: true
```

2. **En un método de objeto**: Cuando **this** se utiliza dentro de un método de objeto, se refiere al objeto que invocó el método.

```
let persona = {
    nombre: "Juan",
    saludar: function() {
        console.log("Hola, soy " + this.nombre);
    }
};

persona.saludar(); // Output: Hola, soy Juan
```

3. **En un constructor de función**: Cuando **this** se utiliza dentro de un constructor de función, se refiere a la instancia del objeto creado por ese constructor.

```
function Persona(nombre) {
    this.nombre = nombre;
    this.saludar = function() {
        console.log("Hola, soy " + this.nombre);
    };
}

let juan = new Persona("Juan");
juan.saludar(); // Output: Hola, soy Juan
```

4. **En funciones de flecha**: En las funciones de flecha, **this** se refiere al valor de **this** en el contexto que rodea la función de flecha, no a su propio valor.

```
let objeto = {
    metodo: function() {
```

```
        setTimeout(() => {
            console.log(this);
        }, 1000);
    }
};

objeto.metodo(); // Output: { metodo: f }
```

En resumen, **this** en JavaScript es una variable especial que se refiere al objeto al que pertenece el contexto actual. Su valor puede cambiar dependiendo de cómo se llama la función y de dónde se utiliza. Entender cómo **this** funciona es crucial para escribir código JavaScript eficaz y evitar errores comunes.