CHECKPOINT 7

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

JavaScript se distingue de otros lenguajes de programación por varias características clave:

- **Interpretado y de lado del cliente:** JavaScript se ejecuta en el navegador del cliente, lo que significa que no requiere compilación previa como otros lenguajes de programación (por ejemplo, C++ o Java). Esto permite una rápida iteración y desarrollo en el lado del cliente.

- **Lenguaje de scripting orientado a objetos:** JavaScript es un lenguaje de scripting que está basado en prototipos, lo que significa que su sistema de objetos difiere de los lenguajes orientados a objetos clásicos como Java o C++. Esto puede llevar a un estilo de programación diferente y a la necesidad de entender los prototipos y la herencia prototípica.

- **Flexibilidad y dinamismo:** JavaScript es un lenguaje flexible que permite la manipulación dinámica de objetos en tiempo de ejecución. Esto significa que puedes agregar, eliminar o modificar propiedades y métodos de objetos en cualquier momento, lo que proporciona una gran flexibilidad, pero también puede llevar a errores si no se maneja correctamente.

- **Asincronía y manejo de eventos**: JavaScript está diseñado para manejar eventos de manera asincrónica, lo que lo hace adecuado para el desarrollo de aplicaciones web interactivas y reactivas. Esta capacidad se basa en la naturaleza de un solo subproceso del navegador y la ejecución de código no bloqueante.

- **Amplio soporte en la web:** JavaScript es el lenguaje de programación principal para el desarrollo web, lo que significa que tiene un amplio soporte en todos los navegadores modernos. Esto lo hace indispensable para el desarrollo frontend y cada vez más popular para el desarrollo backend (con Node.js).

- **Ecosistema y bibliotecas:** JavaScript cuenta con un vasto ecosistema de bibliotecas y frameworks, como React, Angular, Vue.js, Express.js, entre otros, que facilitan el desarrollo de aplicaciones web complejas y eficientes.

En resumen, JavaScript se destaca por su capacidad de ejecución en el navegador del cliente, su flexibilidad y dinamismo, su manejo de eventos asíncronos y su amplio soporte en la web, lo que lo convierte en un lenguaje fundamental para el desarrollo web moderno.

¿Cuáles son algunos tipos de datos JS?

JavaScript admite varios tipos de datos, que se pueden clasificar en dos categorías principales: tipos de datos primitivos y tipos de datos compuestos.

Tipos de datos primitivos:

- **Number:** Representa números enteros o de punto flotante. **Ejemplo:** let x = 10;
- **String:** Representa cadenas de texto. **Ejemplo:** let mensaje = "Hola mundo";
- **Boolean:** Representa un valor verdadero o falso. **Ejemplo:** let esVerdadero = true;
- **Null:** Representa un valor nulo o vacío. **Ejemplo:** let valorNulo = null;
- **Undefined:** Representa un valor no definido. **Ejemplo:** let valorIndefinido;
- **Symbol:** Introducido en ECMAScript 6, representa un identificador único. **Ejemplo:** let sym = Symbol();

Tipos de datos compuestos:

- **Object:** Representa una colección de pares de clave-valor. **Ejemplo:** let persona = { nombre: "Juan", edad: 30 };
- **Array:** Representa una lista ordenada de elementos. **Ejemplo:** let numeros = [1, 2, 3, 4, 5];
- **Function:** Representa una función. **Ejemplo:** function sumar(a, b) { return a + b; }

Además de estos tipos de datos, JavaScript también proporciona otros tipos más avanzados, como los objetos de fecha (Date), las expresiones regulares (RegExp) y las clases (introducidas en ECMAScript 6). Estos tipos de datos permiten a los desarrolladores trabajar con una amplia gama de datos y realizar diversas operaciones en ellos.

¿Cuáles son las tres funciones de String en JS?

JavaScript proporciona una amplia gama de funciones para manipular cadenas (strings). Aquí hay tres funciones importantes de la clase String en JavaScript:

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

   let mensaje = "Hola mundo";

   let longitud = mensaje.length; // Devuelve 11

2. **charAt(index):** Esta función devuelve el carácter en la posición especificada (índice) dentro de una cadena. El índice comienza en cero. **Por ejemplo:**

   let mensaje = "Hola mundo";
   let primerCaracter = mensaje.charAt(0); // Devuelve "H"
   let quintoCaracter = mensaje.charAt(4); // Devuelve "a"

3. **indexOf(substring):** Esta función devuelve el índice de la primera aparición de la subcadena especificada dentro de una cadena. Si la subcadena no se encuentra, devuelve -1. **Por ejemplo:**

```
let mensaje = "Hola mundo";
let indice = mensaje.indexOf("mundo"); // Devuelve 5
```

Estas son solo tres de las muchas funciones disponibles en JavaScript para manipular cadenas. Otras funciones útiles incluyen substring(), slice(), concat(), toLowerCase(), toUpperCase(), trim(), entre otras.

¿Qué es un condicional?

Un condicional en JavaScript es una estructura de control que permite ejecutar cierto bloque de código si se cumple una condición especificada. Estas condiciones se evalúan como verdaderas o falsas y determinan qué acción tomar. Los condicionales son fundamentales para controlar el flujo de ejecución de un programa.

El condicional más básico en JavaScript es el if. Aquí hay un ejemplo simple:

```
let edad = 20;


if (edad >= 18) {
   console.log("Eres mayor de edad");
}
```

En este ejemplo, el bloque de código dentro del if se ejecutará si la variable edad es mayor o igual a 18.

Además del if, JavaScript también proporciona otras formas de condicionales, como else if y else, que permiten especificar diferentes bloques de código que se ejecutarán dependiendo de varias condiciones.

```
let hora = 15;
if (hora < 12) {
   console.log("Buenos días");
} else if (hora < 18) {
   console.log("Buenas tardes");
} else {
   console.log("Buenas noches");
}
```

En este ejemplo, se evalúan tres condiciones diferentes en orden. Si la primera no se cumple, se evalúa la siguiente y así sucesivamente. Si ninguna de las condiciones anteriores es verdadera, se ejecuta el bloque de código dentro del else.

Los condicionales en JavaScript son una herramienta poderosa para controlar el flujo de un programa y permiten que el código tome decisiones dinámicas basadas en las condiciones actuales.

¿Qué es un operador ternario?

Un operador ternario en JavaScript es un operador condicional que permite escribir expresiones condicionales de manera más concisa que utilizando la estructura if...else. El operador ternario toma tres operandos y se utiliza para tomar decisiones basadas en una condición.

La sintaxis del operador ternario es la siguiente:

condición ? expresión_verdadera : expresión_falsa

- Si la condición es verdadera, se evaluará y devolverá la expresión_verdadera.
- Si la condición es falsa, se evaluará y devolverá la expresión_falsa.

Por ejemplo, considera el siguiente código que utiliza un if...else para determinar si un número es par o impar:

```
let numero = 10;

let esPar;

if (numero % 2 === 0) {

   esPar = true;

} else {

   esPar = false;

}

console.log(esPar); // Devuelve true
```

El mismo código se puede escribir de manera más concisa utilizando el operador ternario:

```
let numero = 10;

let esPar = numero % 2 === 0 ? true : false;

console.log(esPar); // Devuelve true
```

Aquí, la condición numero % 2 === 0 se evalúa primero. Si es verdadera, la variable esPar se establece en true, de lo contrario se establece en false.

El operador ternario es especialmente útil cuando se desea asignar un valor a una variable basado en una condición en una sola línea de código. Sin embargo, es importante usarlo con moderación para mantener la legibilidad del código.

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

En JavaScript, hay dos formas principales de definir funciones: mediante una declaración de función y mediante una expresión de función. La diferencia clave entre ambas radica en cómo se comportan y cuándo están disponibles en el código.

1. Declaración de función:

   - Una declaración de función es una forma más tradicional de definir una función.

   - Se compone de la palabra clave function, seguida del nombre de la función, paréntesis para los parámetros y un bloque de código que contiene las instrucciones de la función.

   - Las declaraciones de función se pueden invocar en cualquier parte del código, incluso antes de que se haya declarado la función, debido a un concepto llamado elevación (hoisting). Esto significa que el intérprete de JavaScript moverá todas las declaraciones de función al inicio del ámbito actual antes de ejecutar el código.

   Ejemplo de declaración de función:

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

2. Expresión de función:

- Una expresión de función es cuando se asigna una función a una variable, o se utiliza como un valor en una expresión.

- Se compone de la palabra clave function, seguida opcionalmente de un nombre de función (en el caso de las funciones con nombre), paréntesis para los parámetros y un bloque de código que contiene las instrucciones de la función.

- Las expresiones de función solo están disponibles después de la línea en la que se han definido, ya que se comportan como cualquier otra asignación de variable.

  Ejemplo de expresión de función:

```
const sumar = function(a, b) {
  return a + b;
};
```

La principal diferencia práctica entre una declaración de función y una expresión de función radica en cómo se comportan en relación con la elevación y la disponibilidad en el código. Las declaraciones de función son útiles cuando se necesita que una función esté disponible en todo el ámbito actual (incluso antes de la declaración), mientras que las expresiones de función brindan más flexibilidad y se pueden asignar a variables o pasar como argumentos a otras funciones.

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

En JavaScript, la palabra clave this se refiere al objeto actual en el contexto de ejecución. Su valor depende de cómo se llama una función y dónde se invoca:

- En una función regular, this puede apuntar al objeto global o a un objeto específico.
- En un método de objeto, this se refiere al propio objeto.
- En un constructor de objeto, this se refiere al nuevo objeto creado.
- En eventos de DOM, this se refiere al elemento que activa el evento.

En resumen, this varía dependiendo del contexto de ejecución y puede tener diferentes valores en diferentes partes del código.