1. ¿Qué diferencia a JavaScript de cualquier otro lenguaje de programación?

JavaScript se destaca por varias características que lo distinguen de otros lenguajes de programación:

- **Interpretado en el navegador:** JavaScript es un lenguaje interpretado que se ejecuta en el navegador web del cliente. Esto permite que las interacciones del usuario con la página web sean dinámicas y receptivas sin necesidad de recargar la página.

- **Cliente y servidor:** A diferencia de muchos otros lenguajes que son específicos del lado del servidor, como Python o Java, JavaScript puede ejecutarse tanto en el cliente como en el servidor. Esto permite una mayor coherencia en el desarrollo de aplicaciones web, ya que se puede utilizar el mismo lenguaje en ambos entornos.

- **Asincronía:** JavaScript es inherentemente asincrónico, lo que significa que puede manejar múltiples tareas simultáneamente sin bloquear la ejecución del código. Esto es esencial para realizar operaciones no bloqueantes, como solicitudes de red o eventos del usuario.

- **Funciones de orden superior:** JavaScript permite que las funciones sean tratadas como ciudadanos de primera clase, lo que significa que pueden ser asignadas a variables, pasadas como argumentos y devueltas como valores de otras funciones. Esto facilita el desarrollo de patrones de diseño como la programación funcional.

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.

2. ¿Cuáles son algunos tipos de datos JS?

JavaScript es un lenguaje de programación de tipado dinámico, lo que significa que no es necesario declarar explícitamente el tipo de una variable. Algunos tipos de datos en JavaScript incluyen:

- **Number:** Números, tanto enteros como de punto flotante.
- **String:** Secuencias de caracteres, como "hola mundo".
- **Boolean:** Valores que pueden ser verdaderos o falsos.
- **Array:** Una colección ordenada de elementos o valores.
- **Object:** Una colección de pares clave-valor.
- **Null:** Un valor especial que indica la ausencia de valor.
- **Undefined:** Un valor que indica que una variable no está definida.
- **Symbol**: Representa un identificador único.
- **Function**: Representa una función.

Estos tipos de datos permiten a los desarrolladores trabajar con una amplia gama de datos y realizar diversas operaciones con ellos.

3. ¿Cuáles son las tres funciones de String en JS?

En JavaScript, las cadenas (strings) tienen varias funciones integradas que pueden utilizarse para manipularlas. Algunas de las funciones más comunes son:

- **`length()`:** Esta función devuelve la longitud de una cadena, es decir, el número de caracteres que contiene.

Ejemplo:

```
const texto = "Hola mundo";
console.log(texto.length); //Output: 10
```

- **`charAt()`:** Esta función devuelve el carácter en la posición especificada dentro de una cadena.

Ejemplo:

```
const texto = "Hola mundo";
console.log(texto.charAt(0)); // Salida: "H"
```

- **`concat()`:** Esta función concatena dos o más cadenas y retorna una nueva cadena.

  Ejemplo:

```
const texto1 = "Hola";
const texto2 = "mundo";
console.log(texto1.concat(" ", texto2));
// Salida: "Hola mundo"
```

4. ¿Qué es un condicional?

Un condicional es una estructura de control que permite ejecutar cierto bloque de código si se cumple una condición especificada. Estas condiciones se evaluan como verdaderas o falsas y determinan qué acción tomar.

En JavaScript, los condicionales más comunes son el **`if, else if`** y **`else`**.

Aquí tienes un ejemplo:

```
const edad = 18;

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

Este código imprimirá "Eres mayor de edad" si la variable **`edad`** es mayor o igual a 18, de lo contrario, imprimirá "Eres menor de edad".

5. ¿Qué es un operador ternario?

El operador ternario es una forma abreviada de escribir una declaración **`if...else`** en una sola línea.

El operador evalúa una expresión y devuelve un resultado basado en si esa expresión es verdadera o falsa. Tiene la siguiente sintaxis:

```
condición ? SiVerdadero : SiFalso;
```

Este operador evalúa la condición. Si es verdadera, devuelve la expresión **`SiVerdadero.`** De lo contrario, si es falsa, devuelve la expresión **`SiFalso`**.

Por ejemplo:

```
const edad = 20;
const mensaje = edad >= 18 ? "Mayor de edad" : "Menor de edad";
console.log(mensaje);
```

Este código asigna el mensaje "Mayor de edad" a la variable **`mensaje`** si la edad es mayor o igual a 18, de lo contrario, asigna el mensaje "Menor de edad".

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.

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

- **Declaración de función:** Se define utilizando la palabra clave **`function`** seguida del nombre de la función y el cuerpo de la función. Pueden ser invocadas antes de su declaración debido al hoisting.

Ejemplo:

```
     function nombreDeFuncion() {
          // cuerpo de la función
}
```

- **Expresión de función:** Se asigna a una variable y se puede utilizar como cualquier otra variable. No pueden ser invocadas antes de su declaración.

Ejemplo:

```
     var nombreDeFuncion = function() {
          // cuerpo de la función
};
```

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 a disponibilidad del código. Las declaraciones de función son útiles cuando se necesita que una función esté disponible en todo el ámbito actual, mientras que las expresiones de función brindan más flexibilidad y se pueden asignar a variables o pasar como argumentos a otras funciones.

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

La palabra clave **`this`** en JavaScript se refiere al contexto de ejecución actual. Dependiendo de cómo se use, **`this`** puede hacer referencia a diferentes objetos:

- En una función, **`this`** se refiere al objeto global (en el navegador, esto suele ser **`window`**).
- En un método de un objeto, **`this`** se refiere al objeto que invocó el método.
- En un constructor, **`this`** se refiere a la instancia recién creada del objeto.

Ejemplo:

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

persona.saludar(); // Output: Hola, mi nombre es Juan
```

En este caso, **`this`** hace referencia al objeto **`persona`**, ya que la función **`saludar`** se invoca en el contexto de **`persona`**.