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

JavaScript es un lenguaje de programación único por varias razones, y aquí hay algunas características que lo diferencian de otros lenguajes:

1. **Lenguaje de Programación del Lado del Cliente:**
    - **Explicación:** JavaScript es principalmente un lenguaje de programación del lado del cliente, lo que significa que se ejecuta en el navegador del usuario, proporcionando interactividad y dinamismo en las páginas web sin la necesidad de comunicarse constantemente con el servidor.
    - **Ejemplo:** Cambiar el contenido de una página web o realizar validaciones en formularios en tiempo real sin necesidad de recargar la página.
2. **Asincronía y Callbacks:**
    - **Explicación:** JavaScript es conocido por su naturaleza asincrónica y la utilización extensiva de callbacks. Esto permite realizar operaciones sin bloquear la ejecución del resto del código, lo que es esencial para operaciones como la manipulación de eventos y solicitudes a servidores.
    - **Ejemplo:** La función **setTimeout** que permite ejecutar un código después de un cierto período de tiempo sin detener la ejecución del resto del programa.
3. **Prototipos en lugar de Clases:**
    - **Explicación:** JavaScript utiliza un modelo de programación basado en prototipos en lugar de clases, como en otros lenguajes orientados a objetos. La herencia se logra mediante la relación entre objetos y prototipos en lugar de clases.
    - **Ejemplo:** Crear un objeto y extender sus propiedades y métodos utilizando prototipos.
4. **Flexibilidad y Debilidad de Tipos:**
    - **Explicación:** JavaScript es un lenguaje de tipado débil y dinámico, lo que significa que no se requiere declarar el tipo de variable al definirla y las conversiones de tipos se realizan automáticamente según el contexto.
    - **Ejemplo:** La misma variable puede contener un número en un momento y una cadena de texto en otro.
5. **Comunidad Activa y Ecosistema:**
    - **Explicación:** JavaScript tiene una comunidad de desarrolladores muy activa y un vasto ecosistema de bibliotecas y marcos de trabajo que facilitan el desarrollo de aplicaciones.
    - **Ejemplo:** El uso de bibliotecas como React o frameworks como Node.js que amplían las capacidades de JavaScript.

6. **Uso en Diferentes Contextos:**

    - **Explicación:** Aunque JavaScript se originó como un lenguaje para la web, su versatilidad ha llevado a su adopción en entornos más allá de los navegadores, como en el desarrollo de aplicaciones móviles (usando frameworks como React Native) y en el desarrollo del lado del servidor (con Node.js).
    - **Ejemplo:** Crear aplicaciones móviles utilizando JavaScript y React Native.

En resumen, JavaScript se destaca por su capacidad para ejecutarse en el navegador del usuario, su enfoque asincrónico, su sistema de prototipos, su flexibilidad en cuanto a tipos, una comunidad activa y un ecosistema diverso que lo hacen adecuado para una variedad de contextos de desarrollo.

Para saber más:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

https://javascript.info/

¿Cuáles son algunos tipos de datos JS?

En JavaScript, hay varios tipos de datos que se utilizan para almacenar y manipular información. A Continuación va un listado de cada tipo con ejemplos de cada.

```
//Arreglo (Array): Uso: Almacena una colección ordenada de elementos.
let frutas = ["manzana", "banana", "naranja"];


//Objeto (Object): Uso: Almacena datos en pares clave-valor, permitiendo organizar información de manera estructurada.
let persona = {
  nombre: "Ana",
  edad: 30,
  ciudad: "Madrid"
};
// Boolean:  Uso: Representa valores de verdadero o falso.
var truthy = true;
var notTruthy = false;


// Null: Uso: Indica la ausencia intencional de un valor o un objeto no existente.
var nully = null;


// Undefined (indefinido). Uso: Indica que una variable no ha sido inicializada y no tiene un valor asignado.
var notDefined;


// Number (numérico): Uso: Se utiliza para representar valores numéricos, ya sean enteros o decimales.
var age = 12;
```

```
// String (cadena caracteres). Uso: Almacena y manipula texto.
var name = "Kristine";
var nameTwo = 'Jordan';

// Symbol. Uso: Se utiliza para crear identificadores únicos y prevenir colisiones en nombres de propiedades de objetos.
var mySym = Symbol('foo');
```

Para saber más:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects

¿Cuáles son las tres funciones de String en JS?

Las funciones **length**, **repeat** y **concat** son funciones de cadena (String) en JavaScript que proporcionan funcionalidades específicas para manipular y trabajar con cadenas de texto.

1. **length:**
    - **Uso:** Devuelve la cantidad de caracteres en una cadena.
    - **Ejemplo:**

```
var str = 'The quick brown fox jumped over the lazy dog';
str.length; // 44
```

2. **repeat:**
    - **Uso:** Devuelve una nueva cadena que consiste en la cadena original repetida **n** veces.

Ejemplo:

```
str.repeat(5); // "The quick brown fox jumped over the lazy dogThe quick brown fox jumped over the lazy dogThe quick brown fox jumped over the lazy dogThe quick brown fox jumped over the lazy dogThe quick brown fox jumped over the lazy dog"
```

3. **concat:**
    - **Uso:** Combina dos o más cadenas y devuelve una nueva cadena resultante de la concatenación.
    - **Ejemplo:**

```
str.concat(' again and again'); // "The quick brown fox jumped over the lazy dog again and again"
```

Estas funciones son herramientas útiles para realizar diversas operaciones en cadenas de texto. La documentación proporcionada por MDN ofrece información detallada, ejemplos y casos de uso específicos para cada función, por lo que es un recurso valioso para aprender más sobre ellas.

Para saber más:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String

¿Qué es un condicional?

Un condicional en JavaScript es una estructura de control que permite ejecutar un bloque de código si se cumple una condición específica. En otras palabras, los condicionales son herramientas que le permiten a tu programa tomar decisiones en tiempo de ejecución.

Sintaxis básica de un condicional:

```
if (condicion) {
    // Bloque de código a ejecutar si la condición es verdadera
  } else {
    // Bloque de código a ejecutar si la condición es falsa
  }
```

Ejemplo práctico:

Supongamos que queremos mostrar un mensaje diferente en una página web según la edad de un usuario.

```
  let edad = 25;

if (edad >= 18) {
  console.log("Eres mayor de edad. Puedes ingresar al sitio.");
} else {
  console.log("Eres menor de edad. Acceso restringido.");
}
```

En este ejemplo, la condición **edad >= 18** verifica si la variable **edad** es mayor o igual a 18. Si es verdadera, se ejecutará el primer bloque de código; de lo contrario, se ejecutará el segundo bloque.

Para saber más:

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Introduction

¿Qué es un operador ternario?

Un operador ternario en JavaScript es una forma concisa de escribir una estructura condicional en una sola línea. Este operador permite tomar decisiones basadas en una condición de manera más compacta que utilizando la estructura **if-else**. El operador ternario se denomina "ternario" porque toma tres operandos: la condición, el resultado si la condición es verdadera y el resultado si la condición es falsa.

Sintaxis del operador ternario:

```
condicion ? resultado_si_verdadero : resultado_si_falso;
```

Ejemplo práctico:

Supongamos que queremos determinar si una persona tiene edad suficiente para alquilar un coche y mostrar un mensaje correspondiente.

```
function ageVerification(age) {
    return age > 25 ? "can rent a car" : "can't rent a car";
}
console.log(ageVerification(50)); \\"can rent a car"
console.log(ageVerification(10)); \\ "can't rent a car"
```

En este ejemplo, la condición **edad >= 25** verifica si la variable **edad** es mayor a 25. Si es verdadera, se asigna el mensaje "can rent a car" de lo contrario, se asigna el mensaje "can't rent a car". El resultado se almacena en la variable **age**, y luego se imprime en la consola.

Para saber más:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Conditional_operator

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

1. **Hoisting:**
    - Las declaraciones de funciones tienen hoisting, lo que significa que pueden ser invocadas antes de su declaración.
    - Las expresiones de funciones no tienen hoisting y deben ser definidas antes de su uso.
2. **Ubicación en el Código:**
    - Las declaraciones de funciones pueden ubicarse en cualquier lugar del código y serán hoisteadas al inicio del ámbito.
    - Las expresiones de funciones deben ser definidas antes de ser invocadas y seguirán el flujo normal del código.
3. **Sintaxis:**
    - La sintaxis de las declaraciones de funciones es más directa y no requiere un punto y coma al final.
        - Las expresiones de funciones terminan con un punto y coma y se asignan a una variable.

A continuación se explica con más detalle cada expresión por separado.

Declaración de Función en JavaScript:

Una declaración de función es una forma clásica y directa de definir una función en JavaScript. Utiliza la palabra clave **function** seguida del nombre de la función, una lista de parámetros entre paréntesis y el cuerpo de la función encerrado entre llaves. La característica distintiva de las declaraciones de funciones es que pueden ser invocadas incluso antes de que aparezcan en el código, gracias al concepto de hoisting.

Ejemplo:

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

 console.log(suma(2, 3)); // Resultado: 5
```

Expresión de Función en JavaScript:

Por otro lado, las expresiones de funciones son asignadas a variables o constantes. En lugar de usar la palabra clave **function** seguida de un nombre, se utiliza una sintaxis similar a la de una variable seguida de la palabra clave **function**. Las expresiones de funciones no tienen hoisting, lo que significa que deben ser definidas antes de ser llamadas.

Ejemplo:

```
const resta = function(a, b) {
   return a - b;
 };
console.log(resta(5, 2)); // Resultado: 3
```

Para saber más:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions

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

La palabra clave "this" en JavaScript se utiliza para referirse al contexto en el que se está ejecutando una función. En otras palabras, "this" se refiere al objeto al que pertenece la función en ese momento particular. El valor de "this" puede cambiar dependiendo de cómo se invoque la función.

Ejemplo básico:

En este ejemplo, al llamar a **persona.mostrar()**, "this" dentro de la función **mostrarNombre** se refiere al objeto **persona**, por lo que se imprime el nombre "Juan".

```
function mostrarNombre() {
    console.log(this.nombre);
  }
  const persona = {
    nombre: "Juan",
    mostrar: mostrarNombre
  };
  persona.mostrar(); //  Juan
```

Ejemplo 2:

En el contexto de este código, this se utiliza para acceder a las propiedades y métodos del objeto guide.

```
var guide = {
  title: 'Guide to Programming',
  content: 'Content goes here...',
  visibleToUser: function (viewingUserRole) {
    if (viewingUserRole === 'paid') {
      return true;
    } else {
      return false;
    }  },
  renderContent: function(userRole) {
    if (this.visibleToUser(userRole)) {
      console.log(this.title + "-" + this.content);
    } else {
      this.content = '';
      console.log(this.title + "-" + this.content);
    }  }}
user = { role: 'paid' };
guide.renderContent(user.role); // Guide to Programming - Content goes here...
```

Este código en JavaScript define un objeto guide que representa una guía de programación. Luego, se crea un objeto user con un atributo role que especifica el rol del usuario, y se utiliza el método renderContent del objeto guide para mostrar o no el contenido de la guía según el rol del usuario. Aquí una explicación paso a paso:); } } };

El objeto guide tiene propiedades como title, content y métodos como visibleToUser y renderContent.

visibleToUser es un método que toma un parámetro viewingUserRole y devuelve true si el rol es 'paid' y false en caso contrario.

renderContent es un método que toma un parámetro userRole. Si el usuario es visible según su rol (utilizando visibleToUser), imprime el título y contenido; de lo contrario, establece el contenido en vacío y lo imprime.

Se crea un objeto user con un atributo role igual a 'paid'.

Se llama al método renderContent del objeto guide pasando el rol del usuario como parámetro.

En este ejemplo, como el rol del usuario es 'paid', el método visibleToUser devuelve true, por lo que el contenido de la guía se imprime en la consola. La salida sería algo así como "Guide to Programming - Content goes here...".

Para saber más:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/this