JavaScript

Web Development

JavaScript

JavaScript es el lenguaje de programación principal de la web, utilizado principalmente para agregar funcionalidad a los sitios web.

Es un lenguaje multiparadigma y de propósito general, con tipado dinámico, lo que significa que no se necesita declarar el tipo de las variables.

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

Paradigmas de Programación en JavaScript

Un paradigma es un estilo o forma de programar.

JavaScript es un lenguaje multiparadigma, ya que permite programar de diferentes maneras.

Los principales paradigmas que soporta son:

- Event-driven (Dirigido por eventos): Las funciones pueden responder a eventos, por ejemplo, cuando un usuario hace clic o desplaza la página.

- Functional (Funcional): Se pueden crear funciones “puras” que siempre devuelven el mismo resultado con los mismos argumentos y no producen efectos secundarios.

  JavaScript soporta funciones de primera clase y funciones de orden superior, lo que permite tratarlas como valores y pasarlas como argumentos.

Paradigmas de Programación en JavaScript

- Object-oriented (Orientado a objetos): Permite crear objetos personalizados y usar herencia entre ellos.
- Imperative (Imperativo): Los programas describen paso a paso cómo se realiza una tarea, usando bucles y condicionales.
- Declarative (Declarativo): Se describe el resultado deseado sin detallar el flujo de control. Ejemplo: usar forEach en lugar de un bucle for.

Tipos de Datos Primitivos

Los valores primitivos son los tipos de datos más básicos de JavaScript.

Existen 7 tipos primitivos:

- Number: Valores numéricos, incluyendo enteros y decimales.
- BigInt: Números enteros demasiado grandes para almacenarse en Number.
- Boolean: Valor binario true o false.
- String: Cadena de caracteres (texto).
- Symbol: Valor único generado dinámicamente.
- Null: Valor inexistente o nulo.
- Undefined: Valor que aún no ha sido definido o asignado.

Tipos de Datos Primitivos

JavaScript tiene el operador typeof, que devuelve el tipo de un valor como texto en minúsculas.

Ejemplo:

```
typeof "Hola"     // "string"
typeof 123        // "number"
typeof true       // "boolean"
typeof function(){} // "function"
```

https://developer.mozilla.org/en-US/docs/Glossary/Primitive

Declaración de Variables

JavaScript ofrece tres palabras clave para declarar variables:

- let: Declara una variable con alcance de bloque (block-scoped).

No puede usarse antes de ser inicializada.

- var: Declara una variable con alcance de función (function-scoped).

Se inicializa automáticamente como undefined cuando es hoisted.

- const: Declara una constante, similar a let, pero no puede reasignarse.

Declaración de Variables

let se usa comúnmente con tipos primitivos (número, string, boolean, etc.),

```
let edad = 18;

edad = 19; // permitido
```

Declaración de Variables

Const:

Se usa cuando no se va a reasignar la variable.

Pero eso no significa que su contenido no pueda cambiar, si es un objeto o un array.

```
const numeros = [1, 2, 3];
numeros.push(4); // ✅ permitido (modificamos el contenido)
console.log(numeros); // [1, 2, 3, 4]

// ❌ pero no puedes reasignar la variable completa:
numeros = [9, 8, 7]; // ❌ Error
```

Ámbitos (Scopes) y Hoisting

Block Scope:

Una variable declarada dentro de un bloque ({ }) sólo es accesible dentro de ese bloque.

```
{
  let x = 10;
  console.log(x); // 10
}
console.log(x); // Error: x no está definido
```

Ámbitos (Scopes) y Hoisting

Function Scope:

Una variable declarada dentro de una función solo existe dentro de esa función.

```
function test() {
  var y = 20;
  console.log(y); // 20
}
console.log(y); // Error
```

Ámbitos (Scopes) y Hoisting

Hoisting es el proceso mediante el cual JavaScript mueve las declaraciones al inicio del ámbito, antes de ejecutar el código.

- Las variables con var son inicializadas como undefined.

- Las variables con let o const son hoisted, pero no se inicializan, por lo que causan error si se usan antes.

Ámbitos (Scopes) y Hoisting

```
console.log(varNum); // undefined
console.log(letNum); // ReferenceError

var varNum = 5;
let letNum = 5;

console.log(varNum); // 5
console.log(letNum); // 5
```