Javascript

JavaScript es un lenguaje de programación de alto nivel que se utiliza principalmente para crear contenido interactivo en páginas web. Originalmente, fue desarrollado por Netscape bajo el nombre de "LiveScript", pero luego fue renombrado como JavaScript cuando fue licenciado a Sun Microsystems, que luego fue adquirida por Oracle. JavaScript es un componente fundamental de la web moderna y es ampliamente utilizado para agregar funcionalidades dinámicas a los sitios web, como animaciones, validación de formularios, manipulación del DOM (Document Object Model), comunicación asíncrona con el servidor mediante AJAX (Asynchronous JavaScript and XML), entre otras cosas. JavaScript es un lenguaje de programación versátil y poderoso que ha evolucionado significativamente desde su creación en 1995.

Programación básica

Scope

El "scope" (alcance o ámbito) en JavaScript se refiere a la accesibilidad y visibilidad de variables dentro de un programa. En otras palabras, determina dónde en el código una variable es válida y puede ser utilizada.

Global Scope (Ámbito global)

Las variables declaradas fuera de cualquier función tienen alcance global. Estas variables son accesibles desde cualquier parte del código, incluyendo dentro de funciones, a menos que sean sombreadas por una variable local con el mismo nombre dentro de una función.

```javascript
const globalVariable = 'Soy global';

function myFunction() {
```

```
    console.log(globalVariable);
}

myFunction(); //Soy global
```

Local Scope (Ámbito local)

Las variables declaradas dentro de una función tienen alcance local. Estas variables son visibles sólo dentro de la función en la que están definidas, y no son accesibles desde fuera de esa función.

```
JavaScript
function myFunction() {
    const localVariable = 'Soy local';
    console.log(localVariable);
}

myFunction(); // Imprimirá: Soy local
console.log(localVariable); // Error
```

Es importante tener en cuenta que JavaScript utiliza "**lexical scope**", lo que significa que el ámbito de una variable se determina por su ubicación física en el código durante la escritura, y no por dónde se llama la función en tiempo de ejecución.

Variables

En JavaScript, hay varios tipos de variables que se utilizan para almacenar diferentes tipos de datos. Aquí están los principales tipos de variables:

Var

Esta fue la primera forma de declarar variables en JavaScript. Sin embargo, tiene algunos problemas, como el hoisting (elevación) y el ámbito de función en lugar de ámbito de bloque. No se recomienda su uso en código moderno. Este tipo de variable tiene un scope global.

```javascript
var x = 5;
```

Let

Introducido en ECMAScript 6 (ES6), `let` permite la declaración de variables con ámbito de bloque, lo que significa que solo están disponibles dentro del bloque en el que se declaran.

```javascript
let y = 10;
```

Const

También introducido en ES6, const se utiliza para declarar variables cuyo valor no cambiará a lo largo del tiempo. Las variables declaradas con `const` deben inicializarse con un valor y no pueden ser reasignadas.

```javascript
const PI = 3.14159;
```

Tipos de datos

Además de los tipos de variables nombrados anteriormente, JavaScript también contiene unos tipos de datos que se pueden asignar a las variables.

Primitivos

Number

Representa valores numéricos, ya sean enteros o de punto flotante.

```JavaScript
const value = 5;
const valueFloat = 5.1;
```

String

Representa datos de texto.

```JavaScript
const value = "Soy un tipo de dato primitivo"
```

Boolean

Representa un valor verdadero `(true)` o falso `(false)`.

```JavaScript
const on = true;
const off = false;
```

Big int

Introducido para menajer enteros más grandes que lo que puede representar el tipo de dato `number`. Generalmente utilizado para aplicaciones criptográficas o matemáticas avanzadas.

```javascript
const bigIntNumber = 12345678901234567890123456789012345678901234567890n;
const bigIntFromNumber = BigInt(9007199254740991);
```

Symbol

Introducido en ECMAScript 6 (ES6), representa un identificador único e inmutable.

```javascript
// Crear un Symbol
const mySymbol = Symbol();

// Crear un Symbol con una descripción opcional
const anotherSymbol = Symbol('descripcion');

// Utilizar Symbols como propiedades de un objeto
const obj = {};

obj[mySymbol] = 'Valor asociado al Symbol';
obj[anotherSymbol] = 'Otro valor asociado al Symbol';

// Acceder al valor asociado al Symbol
console.log(obj[mySymbol]); // Imprime: Valor asociado al Symbol
console.log(obj[anotherSymbol]); // Imprime: Otro valor asociado al Symbol
```

```
// Los Symbols son únicos, incluso si tienen la misma descripción
const symbol1 = Symbol('test');
const symbol2 = Symbol('test');

console.log(symbol1 === symbol2); // Imprime: false

// Para crear Symbols globales que sean iguales en todo el código
const globalSymbol = Symbol.for('globalSymbol');
const anotherGlobalSymbol = Symbol.for('globalSymbol');

console.log(globalSymbol === anotherGlobalSymbol); // Imprime: true
```

Undefined

Representa una variable que ha sido declarada pero no inicializada o una propiedad que no está definida.

JavaScript

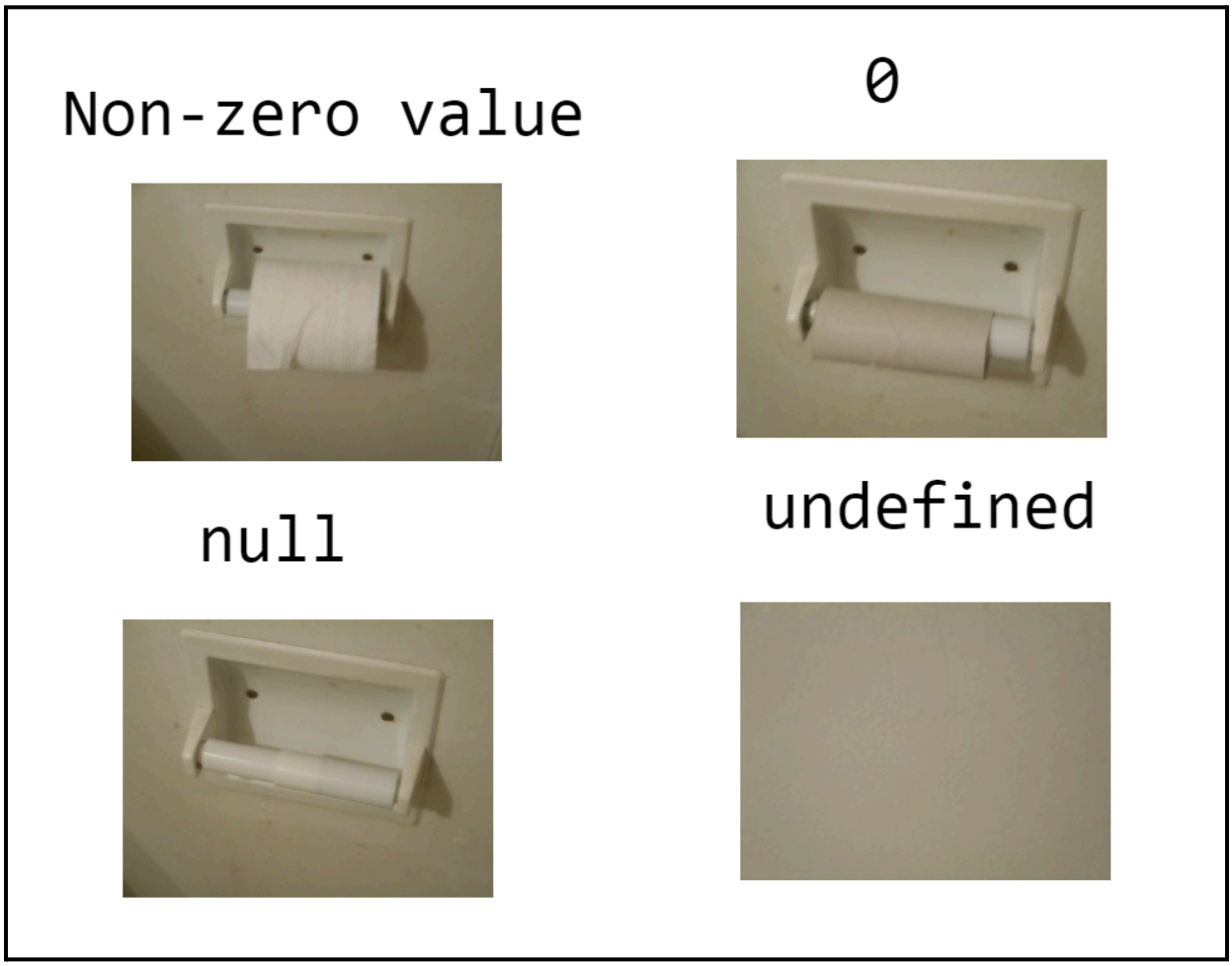
```
// Declarar una variable sin inicializarla
let variableUndefined;

// Acceder a la variable sin definir
console.log(variableUndefined); // Imprime: undefined

// También podemos obtener undefined al intentar acceder a una propiedad que no está definida en un objeto
const obj = {};
console.log(obj.undefinedProperty); // Imprime: undefined
```

Null

Representa la ausencia intencional de algún valor o referencia a un objeto.



No primitivos / Derivados / Referencia

Object

Es una colección de pares clave-valor, donde las claves son cadenas y los valores pueden ser cualquier tipo de dato, incluyendo otros objetos.

```javascript
const person = {
```

```
    name: 'John',
    age: 30,
    email: 'john@example.com',
    address: {
        city: 'New York',
        country: 'USA'
    },
    hobbies: ['reading', 'traveling', 'cooking'],
    sayHello: function() {
        console.log('Hello, my name is ' + this.name + '.');
    }
};
```

Array

Es una colección ordenada de valores que pueden ser de cualquier tipo de datos. Los elementos de un array se acceden mediante un índice numérico.

```JavaScript
// Crear un array de números
const numbers = [1, 2, 3, 4, 5];

// Acceder a un elemento del array
console.log(numbers[0]); // Imprime: 1
```

Functions

Es un tipo especial de objeto que puede ser invocado (llamado) para realizar una tarea específica.

```JavaScript
// Definir una función
function greet(name) {
    console.log('Hola, ' + name + '!');
}

// Llamar a la función
greet('Juan'); // Imprime: Hola, Juan!
```

Date

Representa una fecha y hora en JavaScript.

```JavaScript
// Crear un objeto Date con la fecha y hora actuales
const currentDate = new Date();

// Imprimir la fecha y hora actual
console.log(currentDate); // Wed Apr 10 2024 10:17:18 GMT-0500 (Colombia Standard Time)
```

RegExp (Expresión regular)

Representa un patrón de búsqueda de texto.

```JavaScript
// Crear una expresión regular para buscar palabras que contengan 'JavaScript'
const regex = /JavaScript/;

// Definir una cadena de texto para buscar coincidencias con la expresión regular
const text = 'JavaScript es un lenguaje de programación poderoso';
```

```
// Verificar si la expresión regular coincide con la cadena de texto
console.log(regex.test(text)); // Imprime: true
```

Estos son los principales tipos de datos primitivos y no primitivos en JavaScript. Los datos primitivos son inmutables, lo que significa que una vez que se ha creado una variable con un valor primitivo, no se puede cambiar ese valor. Por otro lado, los datos no primitivos (objetos) son mutables y se pasan por referencia, lo que significa que si se modifica un objeto, esos cambios se reflejarán en todas las referencias al mismo objeto.

Operadores

Los operadores son símbolos especiales o palabras reservadas que se utilizan para realizar operaciones en los datos. En JavaScript, hay varios tipos de operadores que se utilizan para diferentes propósitos.

Operadores aritméticos

- Suma (+)
- Resta (-)
- Multiplicación (*)
- División (/)
- Módulo (%)

Operadores de asignación

Asignación (=)

Asigna el valor de la derecha a la variable de la izquierda:

```JavaScript
let x = 10;
```

Operadores abreviados de asignación

Son operadores aritméticos que realizan la operación y luego asignan el resultado a la variable:

- Suma (+=)
- Resta (-=)
- Multiplicación (*=)
- División (/=)
- Módulo (%=)

```JavaScript
let x = 10;

x += 5; // Equivalente a: x = x + 5;

console.log(x); // Output: 15
```

Operadores de comparación

- **Igualdad (==):** Comprueba si dos valores son iguales.
- **Igualdad estricta (===):** Comprueba si dos valores son iguales y del mismo tipo de datos.
- **No igua (!=):** Comprueba si dos valores no son iguales
- **No igual estricta (!==):** Comprueba si dos valores no son iguales o no son del mismo tipo.
- **Mayor que (>):** Compara si el primer valor es mayor que el segundo valor.
- **Menor que (<)**: Compara si el primer valor es menor que el segundo valor.

- **Mayor o igual que (>=):** Compara si el primer valor es mayor o igual al segundo valor.
- **Menor o igual que (<=):** Compara si el primer valor es menor o igual al segundo valor.

JavaScript

```
let num1 = 10;
let num2 = '10';

// Igualdad
console.log(num1 == num2); // Output: true

// Igualdad estricta
console.log(num1 === num2); // Output: false

// No igual
console.log(num1 != num2); // Output: false

// No igual estricta
console.log(num1 !== num2); // Output: true

let num3 = 10;
let num4 = 5;
let num5 = 5;

// Mayor que
console.log(num3 > num4); // Output: true

// Menor que
console.log(num4 < num3); // Output: true
```

```
// Mayor o igual que
console.log(num4 >= num5); // Output: true

// Menor o igual que
console.log(num4 <= num5); // Output: true
```

Operadores lógicos

- **And lógico (&&):** Devuelve verdadero si ambas expresiones son verdaderas.
- **Or lógico (||):** Devuelve verdadero si al menos una de las expresiones es verdadera.
- **Not lógico (!):** Niega el valor de una expresión.

JavaScript

```
let edad = 20;
let tieneLicencia = true;

// and logico
console.log(edad >= 18 && tieneLicencia) // Output: true
console.log(edad >= 18 && !tieneLicencia) // Output: false
console.log(edad >= 21 && tieneLicencia) // Output: false
console.log(edad >= 21 && !tieneLicencia) // Output: true


//or logico
console.log(edad >= 18 || tieneLicencia) // Output: true
console.log(edad >= 18 || !tieneLicencia) // Output: true
```

```
console.log(edad >= 21 || tieneLicencia) // Output: true
console.log(edad >= 21 || !tieneLicencia) // Output: false

// not logico
console.log(tieneLicencia) // Output: true
console.log(!tieneLicencia) // Output: false
```

Operador ternario

Evalúa una condición y devuelve una expresión si la condición es verdadera, y otra expresión si la condición es falsa.

```
JavaScript
let edad = 20;
let mensaje = (edad >= 18) ? 'Mayor de edad' : 'Menor de edad';
console.log(mensaje); // Output: Mayor de edad
```

Estructuras de control

Las estructuras de control en programación son construcciones que permiten controlar el flujo de ejecución de un programa. Son fundamentales para tomar decisiones, repetir acciones y ejecutar código en función de ciertas condiciones. En JavaScript, las estructuras de control más comunes son:

If...else

Esta estructura permite ejecutar un bloque de código si una condición es verdadera, y otro bloque de código si la condición es falsa.

```JavaScript
let edad = 18;

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

for

Esta estructura se utiliza para repetir un bloque de código un número específico de veces.

```JavaScript
for (let i = 0; i < 5; i++) {
    console.log(i); // Imprime los números del 0 al 4
}
```

While

Se utiliza para repetir un bloque de código mientras una condición especificada sea verdadera.

```JavaScript
let i = 0;
while (i < 5) {
    console.log(i); // Imprime los números del 0 al 4
    i++;
}
```

Do…while

Se utiliza para repetir un bloque de código una vez y luego repetirlo mientras una condición especificada sea verdadera.

```javascript
let i = 0;
do {
    console.log(i); // Imprime los números del 0 al 4
    i++;
} while (i < 5);
```

Switch

Se utiliza para seleccionar uno de varios bloques de código a ejecutar, dependiendo del valor de una expresión.

```javascript
let dia = 3;
let mensaje;

switch (dia) {
    case 1:
        mensaje = 'Lunes';
        break;
    case 2:
        mensaje = 'Martes';
        break;
    case 3:
        mensaje = 'Miércoles';
```

```
        break;
    default:
        mensaje = 'Otro día';
}

console.log(mensaje); // Output: Miércoles
```

Break y continue

Son palabras clave utilizadas dentro de bucles para controlar el flujo de ejecución. `break` se utiliza para salir de un bucle y `continue` se utiliza para saltar la iteración actual y continuar con la siguiente.

Funciones

Una función en JavaScript es un bloque de código que se define una vez y se puede ejecutar en cualquier momento que se necesite. Las funciones en JavaScript son objetos de primera clase, lo que significa que pueden ser tratadas como cualquier otro objeto, como se puede asignar a variables, pasarse como argumentos a otras funciones, devolverse como valores de otras funciones, etc.

JavaScript

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

// Llamando a la función
const resultado = sumar(5, 3);
console.log(resultado); // Imprime: 8
```

En este ejemplo, la función `sumar` toma dos parámetros `a` y `b`, y devuelve su suma. Cuando se llama a la función `sumar(5, 3)`, los valores `5` y `3` se pasan como argumentos a la función, que los suma y devuelve el resultado `8`. Luego, este resultado se asigna a la variable `resultado` y se imprime en la consola.

El funcionamiento de una función en JavaScript es bastante simple:

- **Declaración**: Se define la función utilizando la palabra clave `function`, seguida del nombre de la función y los parámetros entre paréntesis. En el cuerpo de la función se escribe el código que se ejecutará cuando se llame a la función.
- **Llamada**: Para ejecutar el código dentro de una función, se llama a la función utilizando su nombre seguido de paréntesis, y se pasan los valores de los parámetros necesarios.
- **Argumentos y Parámetros**: Los valores que se pasan a la función cuando se llama se denominan argumentos, mientras que los nombres que se utilizan en la definición de la función para representar esos valores se llaman parámetros.
- **Retorno**: Una función puede devolver un valor utilizando la palabra clave `return`. Cuando se ejecuta `return`, la función termina y devuelve el valor especificado. Si no hay una declaración `return` o si `return` no tiene ningún valor asociado, la función devuelve `undefined`.
- **Ámbito**: Las variables declaradas dentro de una función tienen un ámbito local y solo están disponibles dentro de esa función, a menos que sean declaradas con `var`, `let` o `const` en un ámbito superior.

Las funciones en JavaScript son fundamentales para organizar y reutilizar código, así como para permitir la modularidad y la abstracción en el desarrollo de aplicaciones.

Adicionalmente, hay dos conceptos importantes a entender cuando se habla de las funciones:

- **Parámetro**: Un parámetro es un nombre declarado en la lista de parámetros de una función. Los parámetros son como variables locales dentro de la función que reciben valores cuando la función es llamada o invocada. Estos valores son proporcionados por el código que llama a la función. Los parámetros se definen entre los paréntesis en la declaración de la función.
- **Argumento**: Un argumento es el valor real que se pasa a la función cuando es llamada. Es el valor que se suministra a un parámetro durante una invocación de función. Los argumentos se proporcionan en la lista de argumentos entre los paréntesis al llamar a la función.

```JavaScript
// Declaración de la función con dos parámetros: a y b
function sumar(a, b) {
    return a + b; // Utiliza los parámetros a y b para realizar la operación de suma
}

// Llamando a la función y pasando dos argumentos: 5 y 3
const resultado = sumar(5, 3);
// En esta llamada, 5 se asigna al parámetro 'a' y 3 se asigna al parámetro 'b'

console.log(resultado); // Imprime: 8
```

Las funciones pueden categorizarse de diferentes maneras:

Funciones puras

Una función pura es aquella que, dado el mismo conjunto de entradas, siempre devuelve el mismo resultado y no tiene efectos secundarios observables fuera de la función. En otras palabras, una función pura no depende de ningún estado externo y no

modifica ningún estado externo. Esto significa que, dadas las mismas entradas, una función pura siempre devolverá el mismo resultado, sin importar cuántas veces se llame y en qué contexto se llame.

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

let resultado1 = suma(2, 3); // resultado1 = 5
let resultado2 = suma(2, 3); // resultado2 = 5
```

La función `suma` es pura porque siempre devuelve el mismo resultado para el mismo conjunto de entradas (`a` y `b`). No modifica ninguna variable fuera de su ámbito ni realiza ningún efecto observable.

Funciones impuras

Una función impura es aquella que puede tener efectos secundarios o depende de estados externos. Estos efectos secundarios pueden incluir modificar variables globales, realizar operaciones de entrada/salida, realizar solicitudes de red, modificar el DOM, imprimir en consola, entre otros.

```javascript
let total = 0;

function sumaConEfectoSecundario(a) {
    total += a;
    return total;
}
```

```
let resultado1 = sumaConEfectoSecundario(2); // resultado1 = 2
let resultado2 = sumaConEfectoSecundario(3); // resultado2 = 5
```

La función `sumaConEfectoSecundario` es impura porque modifica la variable global `total` cada vez que se llama. El resultado de la función depende del estado externo de `total`, por lo que no es una función pura. Además, produce un efecto observable, que es cambiar el valor de la variable `total` fuera de la función.

Clases

En JavaScript, las clases son una forma de definir objetos y trabajar con herencia. Introducidas en ECMAScript 2015 (también conocido como ES6), las clases proporcionan una sintaxis más clara y orientada a objetos para trabajar con prototipos y objetos.

Una clase en JavaScript puede contener propiedades y métodos que describen el comportamiento y las características de un tipo de objeto específico. Las clases pueden ser utilizadas para crear múltiples instancias de objetos con las mismas propiedades y métodos.

JavaScript
```
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
  }

  saludar() {
```

```
    console.log(`Hola, mi nombre es ${this.nombre} y tengo ${this.edad}
años.`);
  }
}

// Crear una instancia de la clase Persona
const persona1 = new Persona('Juan', 30);

// Llamar al método saludar()
persona1.saludar(); // Imprime: Hola, mi nombre es Juan y tengo 30 años.
```

El constructor es un método especial dentro de una clase que se ejecuta automáticamente cuando se crea una nueva instancia de la clase utilizando el operador `new`. El constructor se utiliza para inicializar las propiedades de un objeto. En el constructor, puedes asignar valores iniciales a las propiedades de la instancia utilizando la palabra clave `this`.

En el código anterior, el constructor de la clase `Persona` toma dos parámetros `(nombre y edad)` y los asigna a las propiedades `nombre` y `edad` del objeto.

Para crear un objeto a partir de una clase, se utiliza el operador `new` seguido del nombre de la clase y los argumentos necesarios para el constructor.

JavaScript
```
const persona1 = new Persona('Juan', 30);
```

En este ejemplo, se crea una instancia de la clase `Persona` con el nombre `persona1` y se pasan los valores `'Juan'` y `30` como argumentos para el constructor.

Una vez que se ha creado una instancia de la clase, se pueden acceder a sus propiedades y métodos utilizando la notación de punto (`.`).

```javascript
console.log(persona1.nombre); // Imprime: Juan
console.log(persona1.edad); // Imprime: 30
persona1.saludar(); // Imprime: Hola, mi nombre es Juan y tengo 30 años.
```

En resumen, una clase en JavaScript proporciona una forma conveniente de definir la estructura y el comportamiento de los objetos. El constructor se utiliza para inicializar las propiedades del objeto, las propiedades son variables asociadas a un objeto, y los métodos son funciones asociadas a un objeto. Una vez que se ha creado una instancia de la clase, se pueden acceder a sus propiedades y métodos utilizando la notación de punto.

DOM (Document Object Model)

El DOM (Document Object Model) es una representación en forma de árbol de la estructura de un documento HTML o XML, que proporciona una interfaz para acceder y manipular los elementos y contenido de una página web mediante JavaScript u otros lenguajes de programación.

El DOM se organiza jerárquicamente, donde cada nodo en el árbol representa un elemento del documento, como un elemento HTML, un atributo, un texto o un comentario. Los elementos del DOM están interconectados entre sí, reflejando la estructura del documento original.

Árbol de nodos

El DOM se organiza en un árbol de nodos, donde cada nodo representa un elemento individual del documento, como un elemento HTML, un atributo, un texto o un

comentario. Los nodos están interconectados entre sí, reflejando la estructura del documento original.

Jerarquía de elementos

Los elementos del DOM están organizados en una jerarquía, donde cada elemento puede tener uno o varios hijos (nodos secundarios) y un padre (nodo padre). Esta jerarquía refleja la estructura anidada de los elementos HTML en el documento.

Interacción con JavaScript

El DOM proporciona una interfaz que permite a los scripts de JavaScript acceder y manipular los elementos y contenido de una página web de forma dinámica. Esto permite realizar acciones como cambiar el contenido de un elemento, modificar estilos CSS, agregar o eliminar elementos, manejar eventos de usuario y más.

Modelo de Objeto

Cada nodo en el DOM es representado como un objeto JavaScript, lo que facilita la interacción con el DOM utilizando JavaScript. Los elementos del DOM son objetos que tienen propiedades y métodos que pueden ser utilizados para acceder y manipular su contenido, atributos, estilos y más.

Actualización dinámica

El DOM se actualiza dinámicamente para reflejar los cambios en el documento HTML o XML, así como las acciones realizadas por los scripts de JavaScript. Esto significa que los cambios realizados en el DOM mediante JavaScript se reflejan inmediatamente en la página web sin necesidad de recargarla.

En resumen, el DOM es una representación estructurada y jerárquica del contenido de una página web, que proporciona una interfaz para acceder y manipular los elementos y contenido de la página mediante JavaScript u otros lenguajes de programación.

Permite a los desarrolladores crear aplicaciones web interactivas y dinámicas al interactuar con los elementos de la página de forma dinámica.

Eventos

Los eventos en JavaScript son acciones o sucesos que ocurren en el navegador web, como hacer clic en un botón, mover el ratón sobre un elemento, presionar una tecla del teclado, cargar una página, etc. Los eventos permiten que el código JavaScript responda a las acciones del usuario o a cambios en el estado del documento, lo que hace que las aplicaciones web sean interactivas y dinámicas.

Event Target (Objetivo del Evento)

El elemento HTML sobre el cual ocurre el evento se conoce como el objetivo del evento o "event target". Por ejemplo, si se hace clic en un botón, el botón es el objetivo del evento de clic

Event Listener (Oyente de Eventos)

Un "event listener" es una función que se adjunta a un elemento HTML y se ejecuta cuando ocurre un evento en ese elemento. Los event listeners se utilizan para definir qué sucede cuando se produce un evento específico. Puedes agregar uno o varios event listeners a un elemento.

Tipos de Eventos

Hay muchos tipos diferentes de eventos en JavaScript, que van desde eventos de interacción del usuario como clics de ratón y pulsaciones de teclas, hasta eventos relacionados con la carga y descarga de recursos, cambios en el estado del documento, cambios en los formularios, etc.

Event Object (Objeto de Evento)

Cuando ocurre un evento, se crea un objeto de evento que contiene información sobre el evento, como el tipo de evento, el objetivo del evento, las coordenadas del puntero del ratón, etc. Puedes acceder a este objeto de evento dentro de un event listener para obtener más información sobre el evento que ha ocurrido.

JavaScript

```
<body>
    <button id="miBoton">Haz clic aquí</button>

    <script>
        // Obtener el botón por su ID
        const boton = document.getElementById('miBoton');

        // Agregar un event listener para el evento de clic
        boton.addEventListener('click', function(event) {
            // Acceder al objeto event.target para obtener el elemento que ha
disparado el evento
            const elemento = event.target;
            // Cambiar el texto del botón cuando se hace clic en él
            elemento.textContent = '¡Gracias por hacer clic!';
        });
    </script>
</body>
</html>
```

En este ejemplo, cuando haces clic en el botón, se ejecuta la función del `event listener` asociado al evento de `clic`. Dentro de esta función, accedemos al `event.target`, que es una referencia al elemento que ha disparado el evento (en este

caso, el botón). Luego, modificamos el texto del botón usando `elemento.textContent` para cambiar su contenido. Esto demuestra cómo puedes usar el event target para realizar acciones específicas basadas en el elemento que ha sido interactuado por el usuario.

API’s del navegador

Las API del navegador son conjuntos de funciones y métodos proporcionados por los navegadores web para permitir que los desarrolladores web accedan y manipulen diferentes aspectos del entorno del navegador y del sistema operativo subyacente. Estas API permiten a los desarrolladores crear aplicaciones web ricas e interactivas.

DOM API

Permite a los desarrolladores acceder y manipular la estructura del documento HTML y XML de una página web. Esto incluye la creación, eliminación y modificación de elementos HTML, así como la manipulación de atributos y estilos.

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM API Example</title>
</head>
<body>
    <div id="myDiv">Hello, World!</div>

    <script>
```

```
        // Acceder al elemento con id "myDiv" y cambiar su contenido
        const myDiv = document.getElementById('myDiv');
        myDiv.textContent = 'Hola, Mundo!';
    </script>
</body>
</html>
```

Asincronismo / XMLHttpRequest / Fetch

Permite a los desarrolladores realizar solicitudes HTTP asíncronas desde JavaScript, lo que posibilita la comunicación con servidores web para obtener o enviar datos sin tener que recargar la página.

Web Storage API

Proporciona mecanismos para almacenar datos de forma persistente en el navegador, como **localStorage** y **sessionStorage**, que permiten a los desarrolladores almacenar datos de forma local en el navegador del usuario.

```JavaScript
// Almacenar un valor en localStorage
localStorage.setItem('nombre', 'Juan');

// Obtener el valor almacenado
const nombre = localStorage.getItem('nombre');
console.log(nombre); // Imprime: Juan
```

Geolocation API

Permite a los desarrolladores obtener la ubicación geográfica del usuario mediante el navegador, utilizando información de GPS, redes móviles y direcciones IP.

JavaScript

```
// Obtener la ubicación actual del usuario
navigator.geolocation.getCurrentPosition(function(position) {
    console.log('Ubicación actual:');
    console.log('Latitud:', position.coords.latitude);
    console.log('Longitud:', position.coords.longitude);
});
```

Canvas API

Permite a los desarrolladores dibujar y manipular gráficos, imágenes y animaciones en tiempo real utilizando JavaScript en un lienzo HTML.

JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Canvas API Example</title>
</head>
<body>
    <canvas id="myCanvas" width="200" height="100" style="border:1px solid
black;"></canvas>

    <script>
        const canvas = document.getElementById('myCanvas');
        const ctx = canvas.getContext('2d');
        ctx.fillStyle = 'red';
        ctx.fillRect(10, 10, 150, 80);
```

```
    </script>
</body>
</html>
```

Web Audio API

Proporciona capacidades de procesamiento y manipulación de audio en tiempo real en el navegador, lo que permite a los desarrolladores crear aplicaciones de música, juegos y aplicaciones multimedia interactivas.

JavaScript

```javascript
// Crear un contexto de audio
const audioContext = new AudioContext();

// Crear un oscilador
const oscillator = audioContext.createOscillator();
oscillator.type = 'sine'; // Tipo de onda sinusoidal
oscillator.frequency.setValueAtTime(440, audioContext.currentTime); //
Frecuencia de 440 Hz

// Conectar el oscilador al destino de salida del contexto de audio
oscillator.connect(audioContext.destination);

// Comenzar a reproducir el sonido
oscillator.start();
```

Estas son solo algunas de las API más comunes disponibles en los navegadores web modernos. Hay muchas más API disponibles, cada una con sus propias funcionalidades y casos de uso específicos.