

Unidad 3

JavaScript

Introducción

JavaScript es un lenguaje de programación dinámico y versátil que se utiliza principalmente para crear y controlar contenido dinámico en sitios web. Desde su creación por Brendan Eich en 1995, JavaScript ha evolucionado para convertirse en una de las tecnologías fundamentales del desarrollo web, junto con HTML y CSS.

Origen y Evolución

Originalmente desarrollado en solo diez días por Brendan Eich en Netscape Communications, JavaScript fue diseñado para ser un lenguaje ligero, fácil de aprender y capaz de interactuar con HTML. Inicialmente conocido como Mocha, luego LiveScript, y finalmente JavaScript, este lenguaje rápidamente ganó popularidad debido a su capacidad para crear interactividad en las páginas web, como validaciones de formularios y efectos visuales.

Con el tiempo, JavaScript ha evolucionado significativamente. La introducción del estándar ECMAScript (ES) en 1997 por Ecma International estableció una base sólida para su desarrollo continuo. Las versiones sucesivas de ECMAScript han añadido características avanzadas, como la programación orientada a objetos, módulos, y funciones asíncronas, ampliando las capacidades y el alcance de JavaScript.

Características Principales

- **Interactividad en el Cliente:** JavaScript permite a los desarrolladores agregar interactividad directamente en el navegador del usuario. Esto incluye la manipulación del Document Object Model (DOM), la validación de formularios, la creación de animaciones y la gestión de eventos del usuario.
- **Programación Asíncrona:** Con la introducción de Promises y `async/await`, JavaScript ha mejorado significativamente en la gestión de operaciones asíncronas, como las solicitudes a servidores, permitiendo un manejo más eficiente y comprensible del código asíncrono.
- **Extensibilidad:** JavaScript puede integrarse con numerosas bibliotecas y frameworks, como React, Angular, y Vue.js, que facilitan la creación de aplicaciones web complejas y escalables. Además, con la llegada de Node.js, JavaScript se ha expandido al desarrollo del lado del servidor.
- **Compatibilidad y Accesibilidad:** JavaScript es compatible con todos los navegadores modernos, lo que garantiza que el código escrito en este lenguaje funcionará en

cualquier dispositivo con acceso a la web. Esto, junto con su accesibilidad y facilidad de aprendizaje, lo convierte en una opción popular tanto para principiantes como para desarrolladores experimentados.

JavaScript Puede Cambiar el Contenido HTML


Uno de los muchos métodos de JavaScript para trabajar con HTML es **getElementById()**.

El siguiente ejemplo "encuentra" un elemento HTML (con id="demo") y cambia el contenido del elemento (innerHTML) a "Hello JavaScript":

<pre><!DOCTYPE html> <html> <body> <h2>Que puede hacer js</h2> <p id="demo">Cambiar el contenido HTML</p> <button type="button" onclick='document.getElementById("demo").innerHTML = "Hola!!!">Click Aca!</button> </body> </html></pre>	<h3>Que puede hacer js</h3> <p>Cambiar el contenido HTML</p> <p><input type="button" value="Click Aca!"/></p>
--	---

JavaScript Puede Cambiar los Valores de los Atributos HTML

En este ejemplo, JavaScript cambia el valor del atributo src (fuente) de una etiqueta

<pre><!DOCTYPE html> <html> <body> <button onclick="document.getElementById('myImage').src='https://www.w3schools.com/js/pic_bulbon.gif'">Encender</button> <button onclick="document.getElementById('myImage').src='https://www.w3schools.com/js/pic_bulboff.gif'">Apagar</button> </body> </html></pre>	 <p><input type="button" value="Encender"/> <input type="button" value="Apagar"/></p>
--	--

JavaScript Puede Cambiar los Estilos HTML (CSS)

Cambiar el estilo de un elemento HTML es una variante de cambiar un atributo HTML

<pre><!DOCTYPE html> <html> <body> <h2>Que puede hacer js</h2> <p id="demo">Cambiar este texto.</p> <button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Clic</button> </body> </html></pre>	
---	--

JavaScript Puede Ocultar y Mostrar Elementos HTML

Ocultar elementos HTML se puede hacer cambiando el estilo de visualización:

```
document.getElementById("demo").style.display = "none";
```

```
document.getElementById("demo").style.display = "block";
```

¿Cómo insertar un js?

Hay tres formas comunes de llamar a JavaScript en una página web:

- Incrustado en HTML: Puedes escribir JavaScript directamente dentro de las etiquetas `<script>` en tu archivo HTML. Por ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Incrustado</title>
</head>
<body>
  <script>
    alert("¡Hola, mundo!");
  </script>
</body>
</html>
```

- Archivo Externo: Puedes escribir tu código JavaScript en un archivo separado con extensión `.js` y luego enlazarlo en tu archivo HTML usando la etiqueta `<script>` con el atributo `src`. Por ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Externo</title>
  <script src="mi_script.js"></script>
</head>
<body>
  <!-- Contenido HTML -->
</body>
</html>
```

- **Manejadores de Eventos:** Puedes llamar a funciones JavaScript cuando ocurren eventos específicos en la página, como hacer clic en un botón o cargar la página. Esto se hace usando atributos de eventos HTML o añadiendo oyentes de eventos a elementos HTML a través de JavaScript. Por ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript con Manejadores de Eventos</title>
</head>
<body>
  <button onclick="saludar()">Haz clic aquí</button>

  <script>
    function saludar() {
      alert("¡Hola, mundo!");
    }
  </script>
</body>
</html>
```

Posibilidades de Visualización en JavaScript

JavaScript puede "mostrar" datos de diferentes maneras:

- Escribiendo en un elemento HTML, usando `innerHTML`.

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
```

- Escribiendo en la salida HTML usando `document.write()`.

```
<h1>Título</h1>
<p>Párrafo.</p>

<script>
document.write(5 + 6);
</script>
```

- Escribiendo en un cuadro de alerta, usando `window.alert()`.

```
<script>
window.alert(5 + 6);
</script>
```

- Escribiendo en la consola del navegador, usando `console.log()`.

```
<script>
console.log(5 + 6);
</script>
```

- Por impresora.

```
<button onclick="window.print()">Print this page</button>
```

Variables, tipos de datos y operadores.

JavaScript Variables

Las variables en JavaScript son contenedores que almacenan datos que pueden ser manipulados y utilizados a lo largo de un programa. La declaración de variables en JavaScript se realiza mediante las palabras clave `var`, `let` y `const`, cada una con sus propias características y usos específicos.

Declaración de Variables

var: Esta es la forma más antigua de declarar variables en JavaScript. Las variables declaradas con `var` tienen un ámbito de función, lo que significa que están disponibles dentro de la función donde se declaran.

```
var nombre = "Juan";
```

let: Introducida en ECMAScript 6 (ES6), `let` permite declarar variables con un ámbito de bloque. Esto significa que la variable sólo está disponible dentro del bloque `{ }` donde se declara.

```
let edad = 25;
```

const: También introducida en ES6, `const` se utiliza para declarar variables cuyo valor no debe cambiar a lo largo del programa. Similar a `let`, tiene un ámbito de bloque.

```
const pi = 3.14159;
```

Asignación de Valores

Las variables pueden ser asignadas a diferentes tipos de datos, como números, cadenas de texto, booleanos, objetos, arreglos, funciones, entre otros.

```
let numero = 10;
let texto = "Hola, Mundo!";
let esVerdadero = true;
let objeto = { nombre: "Ana", edad: 30 };
let arreglo = [1, 2, 3, 4, 5];
```

Reglas de Nomenclatura

Las variables en JavaScript deben seguir ciertas reglas de nomenclatura:

- Deben comenzar con una letra, un signo de dólar \$, o un guion bajo _.
- No pueden comenzar con un número.
- No pueden usar palabras reservadas de JavaScript (como class, return, if, etc.).

Ámbito de Variables

El ámbito de una variable determina dónde puede ser utilizada dentro del código:

- Ámbito global: Las variables declaradas fuera de cualquier función tienen un ámbito global y están disponibles en cualquier parte del código.
- Ámbito de función: Las variables declaradas dentro de una función con var están disponibles solo dentro de esa función.
- Ámbito de bloque: Las variables declaradas con let o const dentro de un bloque {} están disponibles solo dentro de ese bloque.

Cuidados con el const

Esto esta permitido:

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// You can change an element:
cars[0] = "Toyota";

// You can add an element:
cars.push("Audi");
```

Pero no podemos hacer esto:

```
const cars = ["Saab", "Volvo", "BMW"];

cars = ["Toyota", "Volvo", "Audi"]; // ERROR
```

Operadores en JavaScript

Los operadores en JavaScript son símbolos utilizados para realizar operaciones sobre valores y variables. Estos operadores se pueden categorizar en diferentes tipos según su funcionalidad, como operadores aritméticos, de asignación, de comparación, lógicos, de cadena, condicionales y de tipo. A continuación, se presenta una descripción detallada de cada tipo de operador en JavaScript.

Operadores Aritméticos

Estos operadores se utilizan para realizar operaciones matemáticas entre números.

Suma (+): Añade dos valores.

```
let a = 5;  
let b = 3;  
let resultado = a + b; // resultado es 8
```

Resta (-): Resta un valor de otro.

```
let resultado = a - b; // resultado es 2
```

Multiplicación (*): Multiplica dos valores.

```
let resultado = a * b; // resultado es 15
```

División (/): Divide un valor por otro.

```
let resultado = a / b; // resultado es 1.6666...
```

Módulo (%): Devuelve el resto de la división de dos valores.

```
let resultado = a % b; // resultado es 2
```

Exponenciación (**): Eleva un valor a la potencia de otro.

```
let resultado = a ** b; // resultado es 125
```

Incremento (++): Incrementa el valor en uno.

```
a++; // a es 6
```

Decremento (--): Decrementa el valor en uno.

```
b--; // b es 2
```

Operadores de Asignación

Estos operadores se utilizan para asignar valores a variables.

Asignación (=): Asigna un valor a una variable.

```
let x = 10;
```

Asignación de Suma (+=): Añade y asigna un valor a la variable.

```
x += 5; // x es 15
```

Asignación de Resta (-=): Resta y asigna un valor a la variable.

```
x -= 3; // x es 12
```

Asignación de Multiplicación (*=): Multiplica y asigna un valor a la variable.

```
x *= 2; // x es 24
```

Asignación de División (/=): Divide y asigna un valor a la variable.

```
x /= 4; // x es 6
```

Asignación de Módulo (%=): Calcula el módulo y asigna el resultado a la variable.

```
x %= 5; // x es 1
```

Operadores de Comparación

Estos operadores se utilizan para comparar dos valores.

Igualdad (==): Compara si dos valores son iguales.

```
5 == '5'; // true
```

Identidad (===): Compara si dos valores son iguales y del mismo tipo.

```
5 === '5'; // false
```

Desigualdad (!=): Compara si dos valores no son iguales.

```
5 != '5'; // false
```

Desigualdad Estricta (!==): Compara si dos valores no son iguales o no son del mismo tipo.

```
5 !== '5'; // true
```

Mayor Que (>): Compara si un valor es mayor que otro.

```
10 > 5; // true
```

Menor Que (<): Compara si un valor es menor que otro.

```
10 < 5; // false
```

Mayor o Igual Que (>=): Compara si un valor es mayor o igual que otro.

```
10 >= 5; // true
```

Menor o Igual Que (<=): Compara si un valor es menor o igual que otro.

```
10 <= 5; // false
```


Operadores Lógicos

Estos operadores se utilizan para realizar operaciones lógicas y devuelven un valor booleano (true o false).

AND Lógico (&&): Devuelve true si ambas expresiones son verdaderas.

```
(5 > 3) && (2 < 4); // true
```

OR Lógico (||): Devuelve true si una de las expresiones es verdadera.

```
(5 > 3) || (2 > 4); // true
```

NOT Lógico (!): Niega una expresión lógica.

```
!(5 > 3); // false
```

Operadores de Cadena

Estos operadores se utilizan para concatenar cadenas de texto.

Concatenación (+): Une dos cadenas de texto.

```
let saludo = "Hola, " + "Mundo!"; // "Hola, Mundo!"
```

Concatenación Asignativa (+=): Añade una cadena a otra y asigna el resultado a la variable.

```
let mensaje = "Hola";  
mensaje += ", Mundo!"; // "Hola, Mundo!"
```

Operador Condicional (Ternario)

Este operador toma tres operandos y actúa como un atajo para la declaración if-else.

Condicional (? :): Devuelve un valor si la condición es verdadera y otro si es falsa.

```
let edad = 18;  
let puedeVotar = (edad >= 18) ? "Sí" : "No"; // "Sí"
```

Operadores de Tipo

Estos operadores se utilizan para verificar el tipo de datos de una variable.

typeof: Devuelve el tipo de dato de una variable.

```
typeof 42; // "number"  
typeof "Hola"; // "string"
```

instanceof: Verifica si un objeto es una instancia de un tipo específico.

```
let fecha = new Date();  
fecha instanceof Date; // true
```

Funciones y objetos en JavaScript.

Las funciones en JavaScript son bloques de código reutilizables diseñados para realizar una tarea específica. Una función se puede definir una vez y luego invocarse múltiples veces a lo largo del programa, lo que ayuda a mantener el código organizado, modular y fácil de mantener.

Definición de Funciones

En JavaScript, una función se puede definir utilizando la palabra clave `function`, seguida por el nombre de la función, paréntesis y un bloque de código delimitado por llaves. Los paréntesis pueden incluir parámetros, que son variables que se utilizan para recibir valores cuando se llama a la función.

```
function saludar(nombre) {  
    console.log("Hola, " + nombre + "!");  
}
```

Invocación de Funciones

Para invocar o llamar a una función, simplemente se escribe su nombre seguido de paréntesis. Si la función acepta parámetros, se pasan los valores correspondientes dentro de los paréntesis.

```
saludar("Juan"); // "Hola, Juan!"
```

Funciones con Retorno

Las funciones pueden devolver un valor utilizando la palabra clave `return`. El valor especificado después de ***return*** se devuelve al lugar donde se invocó la función.

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

```
let resultado = sumar(3, 4); // resultado es 7
```

Funciones Anónimas

Las funciones anónimas son funciones sin nombre. A menudo se utilizan como valores de variables o como argumentos a otras funciones.

```
let multiplicar = function(a, b) {  
    return a * b;  
};
```

```
console.log(multiplicar(3, 4)); // 12
```

Funciones Flecha (Arrow Functions)

Las funciones flecha son una forma más concisa de escribir funciones anónimas introducida en ECMAScript 6 (ES6). Se utilizan el operador de flecha (\Rightarrow) y son especialmente útiles para funciones de una sola línea.

```
let restar = (a, b) => a - b;
```

```
console.log(restar(7, 2)); // 5
```

Funciones de Orden Superior

Las funciones de orden superior son funciones que pueden aceptar otras funciones como argumentos o devolver funciones como resultado. Estas son fundamentales en la programación funcional.

```
function aplicarOperacion(a, b, operacion) {  
  return operacion(a, b);  
}
```

```
let resultado = aplicarOperacion(5, 3, (x, y) => x * y);  
console.log(resultado); // 15
```

Funciones como Métodos

En JavaScript, las funciones también pueden ser propiedades de objetos. Cuando una función es una propiedad de un objeto, se llama método.

```
let persona = {  
  nombre: "Carlos",  
  saludar: function() {  
    console.log("Hola, " + this.nombre + "!");  
  }  
};
```

```
persona.saludar(); // "Hola, Carlos!"
```

Ámbito de las Funciones

Las variables definidas dentro de una función solo están disponibles dentro de esa función. Este se llama ámbito local. Las variables definidas fuera de cualquier función tienen ámbito global y están disponibles en cualquier parte del código.

```
function ejemplo() {  
  let local = "Solo accesible dentro de esta función";  
  console.log(local);  
}
```

```
ejemplo();  
console.log(local); // Error: local is not defined
```

Funciones Recursivas

Una función recursiva es una función que se llama a sí misma. La recursión es útil para resolver problemas que pueden dividirse en sub problemas similares.

```
function factorial(n) {  
  if (n === 0) {  
    return 1;  
  } else {  
    return n * factorial(n - 1);  
  }  
}
```

```
console.log(factorial(5)); // 120
```

Objetos en JavaScript

En JavaScript, un objeto es una colección de propiedades, y una propiedad es una asociación entre un nombre (o clave) y un valor. Los valores de las propiedades pueden ser cualquier tipo de datos, incluyendo otros objetos, lo que permite la creación de estructuras de datos complejas.

Creación de Objetos

Hay varias formas de crear objetos en JavaScript:

Literal de Objeto: Esta es la forma más común y directa de crear un objeto.

```
let persona = {  
  nombre: "Juan",  
  edad: 30,  
  saludar: function() {  
    console.log("Hola, soy " + this.nombre);  
  }  
};
```

Constructor de Objeto: Utiliza la función `Object`.

```
let persona = new Object();  
persona.nombre = "Juan";  
persona.edad = 30;  
persona.saludar = function() {  
  console.log("Hola, soy " + this.nombre);  
};
```

Funciones Constructoras: Permiten crear múltiples instancias de un objeto.

```
function Persona(nombre, edad) {  
  this.nombre = nombre;  
  this.edad = edad;  
  this.saludar = function() {  
    console.log("Hola, soy " + this.nombre);  
  };  
}
```

```
let juan = new Persona("Juan", 30);  
let maria = new Persona("Maria", 25);
```

Acceso a Propiedades

Las propiedades de los objetos pueden accederse usando la notación de punto o la notación de corchetes.

```
// Notación de punto  
console.log(persona.nombre); // "Juan"
```

```
// Notación de corchetes  
console.log(persona['nombre']); // "Juan"
```

Métodos de Objetos

Un método es una función que se define como una propiedad de un objeto. En el ejemplo anterior, saludar es un método del objeto persona.

```
persona.saludar(); // "Hola, soy Juan"
```

Propiedades y Métodos Dinámicos

En JavaScript, se pueden agregar y eliminar propiedades y métodos de un objeto en cualquier momento

```
persona.apellido = "Pérez";  
console.log(persona.apellido); // "Pérez"
```

```
delete persona.edad;  
console.log(persona.edad); // undefined.
```

Iteración sobre Propiedades

Se puede iterar sobre las propiedades de un objeto utilizando un bucle **for...in**.

```
for (let clave in persona) {  
    console.log(clave + ": " + persona[clave]);  
}  
// Salida:  
// nombre: Juan  
// saludar: function() { ... }  
// apellido: Pérez
```

Objetos Anidados

Las propiedades de un objeto pueden ser otros objetos, permitiendo estructuras de datos anidadas.

```
let estudiante = {  
  nombre: "Ana",  
  curso: {  
    nombre: "Matemáticas",  
    profesor: "Sr. Gómez"  
  }  
};  
  
console.log(estudiante.curso.nombre); // "Matemáticas"
```

Prototipos

Cada objeto en JavaScript tiene un prototipo, un objeto del cual hereda propiedades y métodos. Esto es fundamental para la herencia en JavaScript.

```
function Persona(nombre, edad) {  
  this.nombre = nombre;  
  this.edad = edad;  
}  
  
Persona.prototype.saludar = function() {  
  console.log("Hola, soy " + this.nombre);  
};  
  
let juan = new Persona("Juan", 30);  
juan.saludar(); // "Hola, soy Juan"
```

Clases (Introducidas en ES6)

Las clases proporcionan una forma más clara y simple de crear objetos y manejar herencia en JavaScript.

```
class Persona {  
  constructor(nombre, edad) {  
    this.nombre = nombre;  
    this.edad = edad;  
  }  
  
  saludar() {  
    console.log("Hola, soy " + this.nombre);  
  }  
}
```

```
let juan = new Persona("Juan", 30);
juan.saludar(); // "Hola, soy Juan"
```

Estructuras de Control en JavaScript

Las estructuras de control en JavaScript permiten dirigir el flujo de ejecución de un programa. Las principales estructuras de control incluyen condicionales (if, else if, else) y bucles (for, while, do...while). A continuación, se explica cómo funcionan estas estructuras y cómo se utilizan.

Condicionales: if, else if, else

Los condicionales se utilizan para ejecutar diferentes bloques de código en función de una condición.

if: La instrucción if ejecuta un bloque de código si una condición especificada es verdadera.

```
let edad = 18;

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

else: La instrucción else se utiliza junto con if para ejecutar un bloque de código si la condición del if es falsa.

```
let edad = 16;

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

else if: La instrucción else if se utiliza para especificar una nueva condición si la condición anterior es falsa.

```
let nota = 85;

if (nota >= 90) {
  console.log("Excelente");
} else if (nota >= 70) {
  console.log("Bueno");
} else {
  console.log("Necesita mejorar");
}
```


Bucles: for, while, do...while

Los bucles permiten ejecutar un bloque de código varias veces.

for: El bucle for se utiliza cuando se conoce de antemano el número de iteraciones.

```
for (let i = 0; i < 5; i++) {  
  console.log("El número es " + i);  
}
```

El bucle for consta de tres partes:

Inicialización: let i = 0

Condición: i < 5

Incremento: i++

while: El bucle while ejecuta un bloque de código mientras una condición especificada sea verdadera.

```
let i = 0;
```

```
while (i < 5) {  
  console.log("El número es " + i);  
  i++;  
}
```

do...while: El bucle do...while es similar al bucle while, pero ejecuta el bloque de código al menos una vez antes de verificar la condición.

```
let i = 0;
```

```
do {  
  console.log("El número es " + i);  
  i++;  
} while (i < 5);
```

Bucles Anidados

Los bucles anidados son bucles dentro de otros bucles. Son útiles para trabajar con estructuras de datos multidimensionales como matrices bidimensionales.

```
for (let i = 0; i < 3; i++) {  
  for (let j = 0; j < 3; j++) {  
    console.log("i: " + i + ", j: " + j);  
  }  
}
```

Control de Bucles: break y continue

break: Termina el bucle actual inmediatamente.

```
for (let i = 0; i < 5; i++) {  
  if (i === 3) {  
    break;  
  }  
  console.log("El número es " + i);  
}  
// Salida: 0, 1, 2
```

continue: Salta a la siguiente iteración del bucle.

```
for (let i = 0; i < 5; i++) {  
  if (i === 3) {  
    continue;  
  }  
  console.log("El número es " + i);  
}  
// Salida: 0, 1, 2, 4
```

Strings, Arrays, BigInt, Number

Strings en JavaScript

Un string es una cadena de texto utilizada para representar palabras, frases, o cualquier tipo de texto. En JavaScript, los strings se pueden definir utilizando comillas simples ('), comillas dobles ("), o comillas invertidas (`) para plantillas literales.

```
let texto1 = 'Hola, mundo!';  
let texto2 = "Hola, JavaScript!";  
let texto3 = `La suma de 2 y 3 es ${2 + 3}`;
```

Métodos Comunes de Strings

- **length:** Devuelve la longitud del string.
let longitud = texto1.length; // 12
- **charAt():** Devuelve el carácter en una posición específica.
let caracter = texto1.charAt(0); // 'H'
- **toUpperCase()** y **toLowerCase():** Convierte el string a mayúsculas o minúsculas.
let mayusculas = texto1.toUpperCase(); // 'HOLA, MUNDO!'
let minusculas = texto1.toLowerCase(); // 'hola, mundo!'
- **substring():** Devuelve una subcadena del string.
let subcadena = texto1.substring(0, 4); // 'Hola'

Arrays en JavaScript

Un array es una colección de elementos, que pueden ser de cualquier tipo, almacenados en una única variable. Los arrays en JavaScript son dinámicos y pueden cambiar de tamaño.

Creación de Arrays

```
let numeros = [1, 2, 3, 4, 5];  
let mezcla = [1, 'dos', true, null];
```

Métodos Comunes de Arrays

- **length**: Devuelve el número de elementos en el array.
`let longitud = numeros.length; // 5`
- **push() y pop()**: Añade o elimina elementos al final del array.
`numeros.push(6); // [1, 2, 3, 4, 5, 6]`
`let ultimo = numeros.pop(); // 6, [1, 2, 3, 4, 5]`
- **shift() y unshift()**: Añade o elimina elementos al inicio del array.
`numeros.unshift(0); // [0, 1, 2, 3, 4, 5]`
`let primero = numeros.shift(); // 0, [1, 2, 3, 4, 5]`
- **map() y filter()**: Aplican una función a cada elemento del array o filtran elementos según una condición.
`let cuadrados = numeros.map(x => x * x); // [1, 4, 9, 16, 25]`
`let pares = numeros.filter(x => x % 2 === 0); // [2, 4]`

BigInt en JavaScript

BigInt es un tipo de dato que permite representar enteros con precisión arbitraria, útil para trabajar con números muy grandes que exceden el límite de los números enteros tradicionales en JavaScript.

```
let numeroGrande = 1234567890123456789012345678901234567890n;  
let otroNumeroGrande = BigInt("1234567890123456789012345678901234567890");
```

Number en JavaScript

El tipo Number representa tanto números enteros como de punto flotante. JavaScript usa un único tipo de número, almacenado como un valor de punto flotante de doble precisión según el estándar IEEE 754.

Creación de Numbers

```
let entero = 42;  
let flotante = 3.14;
```

Propiedades y Métodos Comunes de Numbers

- **toFixed()**: Formatea un número utilizando notación de punto fijo.
`let formateado = flotante.toFixed(2); // '3.14'`
- **parseInt() y parseFloat()**: Convierte cadenas a enteros y números de punto flotante.
`let enteroDesdeCadena = parseInt("123"); // 123`
`let flotanteDesdeCadena = parseFloat("3.14"); // 3.14`
- **isNaN()**: Comprueba si un valor es NaN (Not-a-Number).
`let noEsNumero = isNaN("hola"); // true`

Manipulación del DOM (Document Object Model)

El DOM (Document Object Model) es una representación estructurada de un documento HTML o XML que permite a los lenguajes de programación acceder y modificar el contenido, la estructura y el estilo del documento de manera dinámica. En el contexto de JavaScript, la manipulación del DOM es una de las tareas más comunes y esenciales para crear páginas web interactivas.

Estructura del DOM

El DOM representa un documento como una jerarquía de nodos. Estos nodos pueden ser elementos, atributos, texto, comentarios, entre otros. La estructura del DOM se asemeja a un árbol, donde cada nodo tiene una relación padre-hijo con otros nodos.

Por ejemplo, un documento HTML simple:

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de DOM</title>
</head>
<body>
  <h1>Hola, mundo!</h1>
  <p>Este es un párrafo.</p>
</body>
</html>
```

Se representa en el DOM de la siguiente manera:

```
html
├── head
│   └── title
└── body
    ├── h1
    └── p
```

Selección de Elementos del DOM

Para manipular el DOM, primero debemos seleccionar los elementos que queremos modificar. JavaScript ofrece varios métodos para esto:

- **getElementById**: Selecciona un elemento por su ID.
`let titulo = document.getElementById('titulo');`
- **getElementsByClassName**: Selecciona todos los elementos que tienen una clase específica.
`let parrafos = document.getElementsByClassName('parrafo');`
- **getElementsByTagName**: Selecciona todos los elementos de un tipo de etiqueta específico.
`let divs = document.getElementsByTagName('div');`
- **querySelector y querySelectorAll**: Selecciona elementos utilizando selectores de CSS.
`let primerParrafo = document.querySelector('p');`
`let todosLosParrafos = document.querySelectorAll('p');`

Modificación de Elementos del DOM

Una vez seleccionados los elementos, podemos modificar su contenido, atributos y estilos.

- **Modificar Contenido**: Usando `innerHTML` o `textContent`.
`titulo.innerHTML = 'Nuevo Título';`
- **Modificar Atributos**: Usando `setAttribute` o accediendo directamente a la propiedad.
`let imagen = document.querySelector('img');`
`imagen.setAttribute('src', 'nueva-imagen.jpg');`
`imagen.alt = 'Descripción de la imagen';`
- **Modificar Estilos**: Accediendo a la propiedad `style`.
`titulo.style.color = 'red';`
`titulo.style.fontSize = '24px';`

Creación y Eliminación de Elementos del DOM

También podemos crear y eliminar elementos del DOM dinámicamente.

Crear Elementos: Usando `createElement` y `appendChild`.

```
let nuevoParrafo = document.createElement('p');
nuevoParrafo.textContent = 'Este es un nuevo párrafo';
document.body.appendChild(nuevoParrafo);
```

Eliminar Elementos: Usando `removeChild`.

```
let parrafoAEliminar = document.querySelector('p');
parrafoAEliminar.parentNode.removeChild(parrafoAEliminar);
```

Eventos y Manipulación del DOM

Los eventos son fundamentales para la interacción del usuario con la página web. Podemos agregar manejadores de eventos para responder a acciones del usuario.

Agregar Eventos: Usando `addEventListener`.

```
titulo.addEventListener('click', () => {  
    alert('Título clickeado');  
});
```

Modificar DOM en Respuesta a Eventos:

```
let boton = document.querySelector('button');  
boton.addEventListener('click', () => {  
    let parrafo = document.createElement('p');  
    parrafo.textContent = 'Nuevo párrafo agregado';  
    document.body.appendChild(parrafo);  
});
```

Eventos y Manejo de Eventos en JavaScript

Los eventos en JavaScript son acciones o sucesos que ocurren en el navegador, los cuales pueden ser desencadenados por la interacción del usuario con la página web o por el sistema. El manejo de eventos es un aspecto crucial del desarrollo web interactivo, ya que permite a los desarrolladores crear aplicaciones dinámicas que responden a las acciones del usuario.

Tipos de Eventos

Existen muchos tipos de eventos en JavaScript, algunos de los más comunes incluyen:

- Eventos de Mouse: `click`, `dblclick`, `mouseover`, `mouseout`, `mousemove`, `mousedown`, `mouseup`.
- Eventos de Teclado: `keydown`, `keyup`, `keypress`.
- Eventos de Formulario: `submit`, `focus`, `blur`, `change`, `input`.
- Eventos de Documento/Carga: `DOMContentLoaded`, `load`, `resize`, `scroll`, `unload`.

Manejo de Eventos

El manejo de eventos implica la asociación de funciones a estos eventos para que se ejecuten en respuesta a las acciones del usuario. Hay varias formas de manejar eventos en JavaScript:

Atributos HTML: Definir manejadores de eventos directamente en los atributos HTML.

```
<button onclick="alert('Botón clickeado!')">Haz clic aquí</button>
```

Propiedades del DOM: Asignar funciones a las propiedades de eventos del DOM.

```
let boton = document.querySelector('button');
boton.onclick = function() {
  alert('Botón clickeado!');
};
```

Método addEventListener: Añadir eventos de manera más flexible y moderna, permitiendo agregar múltiples manejadores para el mismo evento.

```
let boton = document.querySelector('button');
boton.addEventListener('click', function() {
  alert('Botón clickeado!');
});
```

Ejemplo Práctico

Un ejemplo de manejo de eventos usando addEventListener:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Manejo de Eventos</title>
</head>
<body>
  <button id="miBoton">Haz clic aquí</button>
  <script>
    let boton = document.getElementById('miBoton');
    boton.addEventListener('click', function() {
      alert('Botón clickeado!');
    });
  </script>
</body>
</html>
```

Eventos y Funciones

Podemos pasar el objeto event a las funciones manejadoras para obtener información adicional sobre el evento:

```
let boton = document.getElementById('miBoton');
boton.addEventListener('click', function(event) {
    console.log('Tipo de evento: ', event.type);
    console.log('Elemento objetivo: ', event.target);
});
```

Eliminación de Manejadores de Eventos

Si necesitamos eliminar un manejador de eventos, podemos usar el método `removeEventListener`:

```
function mostrarAlerta() {
    alert('Botón clickeado!');
}
```

```
let boton = document.getElementById('miBoton');
boton.addEventListener('click', mostrarAlerta);
```

```
// Para eliminar el manejador de eventos
boton.removeEventListener('click', mostrarAlerta);
```

Prevención del Comportamiento Predeterminado

Algunos eventos tienen comportamientos predeterminados que pueden ser prevenidos usando `preventDefault`:

```
let enlace = document.querySelector('a');
enlace.addEventListener('click', function(event) {
    event.preventDefault(); // Prevenir la acción predeterminada de seguir el enlace
    alert('Enlace clickeado, pero la navegación fue prevenida.');
```