SIMBOLOS EMPLEADOS EN JAVASCRIPT

1. Operadores Aritméticos

- + : Suma
- - : Resta
- * : Multiplicación
- / : División
- % : Módulo (resto de división)
- ++ : Incremento (incrementa en 1)
- -- : Decremento (decrece en 1)

2. Operadores de Asignación

- = : Asignación
- += : Asignación con suma
- -= : Asignación con resta
- *= : Asignación con multiplicación
- /= : Asignación con división
- %= : Asignación con módulo
- **= : Asignación con exponenciación (Introducido en ES6)

3. Operadores de Comparación

- == : Igualdad (no estricta)
- === : Igualdad estricta (sin conversión de tipo)
- != : Desigualdad (no estricta)
- !== : Desigualdad estricta
- > : Mayor que
- < : Menor que
- >= : Mayor o igual que
- <= : Menor o igual que
- ?? : Coalescencia nula (Introducido en ES11)

4. Operadores Lógicos

- && : Y lógico (AND)
- || : O lógico (OR)
- ! : No lógico (NOT)

- ?? : Operador de fusión nula (nullish coalescing)

5. Operadores Bit a Bit

- & : Y bit a bit
- | : O bit a bit
- ^ : XOR bit a bit
- ~ : Negación bit a bit (complemento a uno)
- << : Desplazamiento a la izquierda
- >> : Desplazamiento a la derecha
- >>> : Desplazamiento a la derecha sin signo

6. Operadores de Tipo

- typeof : Devuelve el tipo de una variable
- instanceof : Comprueba si un objeto es instancia de una clase o constructor

7. Operadores de Acceso

- . : Acceso a una propiedad u objeto
- [] : Acceso a propiedades o elementos (cuando se usa un índice o nombre de propiedad dinámico)

8. Operador Ternario (Condicional)

- ? : : Operador ternario (expresión condicional corta)

  let resultado = (condicion) ? valor1 : valor2;

9. Operadores de Desestructuración (Introducido en ES6)

- {} : Desestructuración de objetos

  const {nombre, edad} = persona;

- [] : Desestructuración de arrays

  const [primer, segundo] = array;

10. Operadores de Spread/Rest (Introducido en ES6)

- ... : Spread y Rest
    - **Spread**: Copiar los elementos de un array u objeto

```
let arr = [1, 2, 3];
let arr2 = [...arr];
```

    - **Rest**: Agrupar los elementos sobrantes

```
function sumar(...numeros) {
  return numeros.reduce((a, b) => a + b, 0);
}
```

11. Operadores de Funciones Generadoras

- function* : Declaración de una función generadora
- yield : Pausa la ejecución de una función generadora y devuelve un valor

12. Palabras Clave y Símbolos Relacionados con Clases y Objetos

- class : Definición de una clase
- extends : Herencia de una clase
- super() : Llamada al constructor de la clase padre
- constructor() : Constructor de una clase
- this : Referencia al objeto actual
- static : Método o propiedad estática
- new : Crea una nueva instancia de un objeto
- delete : Elimina una propiedad de un objeto
- in : Verifica si una propiedad existe en un objeto
- instanceof : Verifica si un objeto es instancia de una clase o constructor

13. Símbolos de Control de Flujo

- if : Estructura condicional
- else : Alternativa en la condición
- else if : Segunda condición en la estructura condicional
- switch : Estructura de selección múltiple
- case : Caso dentro de un switch
- break : Sale del bloque de control
- continue : Salta a la siguiente iteración de un bucle
- return : Retorna un valor desde una función

- throw : Lanza una excepción
- try : Intenta ejecutar un bloque de código
- catch : Captura excepciones
- finally : Código que se ejecuta después de try y catch

14. Delimitadores

- ; : Final de una declaración o expresión
- , : Separador de elementos (en listas, parámetros, etc.)
- . : Acceso a propiedad de un objeto
- [] : Definición de un array o acceso a un índice
- {} : Definición de un objeto o bloque de código
- () : Agrupación de expresiones o parámetros de función

15. Comentarios

- // : Comentario de una línea
- /* ... */ : Comentario de varias líneas

16. Símbolos de Regulares (Expresiones Regulares)

- ^ : Inicio de una cadena
- $: Fin de una cadena
- . : Coincide con cualquier carácter excepto saltos de línea
- * : Cero o más repeticiones
- + : Una o más repeticiones
- ? : Cero o una repetición
- [] : Conjunto de caracteres (rangos)
- () : Agrupación
- | : O (alternancia)
- \ : Escape de caracteres especiales

17. Operadores de Módulos de Importación/Exportación (ES6)

- import : Importar módulos
- export : Exportar módulos
- export default : Exportación predeterminada
- as : Alias para un módulo o propiedad

18. Símbolos de Promesas (ES6+)

- new Promise() : Crear una promesa
- .then() : Manejar la resolución de una promesa
- .catch() : Manejar el rechazo de una promesa
- .finally() : Ejecución final después de que la promesa se resuelve o se rechaza

19. Símbolos de Métodos Especiales

- Symbol() : Crea un valor único y primitivo
- Symbol.iterator : Protocolo de iterador
- Symbol.toStringTag : Protocolo para personalizar el etiquetado de objetos

---

Resumen de Caracteres Especiales

- \n : Salto de línea
- \t : Tabulador
- \\ : Barra invertida
- \' : Comilla simple
- \" : Comilla doble
- \b : Retroceso

Ejemplos:

1. **Operadores Aritméticos**

```javascript
let a = 5;
let b = 2;

console.log(a + b);  // Suma: 7
console.log(a - b);  // Resta: 3
console.log(a * b);  // Multiplicación: 10
console.log(a / b);  // División: 2.5
console.log(a % b);  // Módulo (residuo): 1
console.log(a++);    // Incremento: 5 (Luego a = 6)
console.log(a--);    // Decremento: 6 (Luego a = 5)
```

2. **Operadores de Asignación**

```javascript
let x = 10;
x += 5;  // x = x + 5 => 15
x -= 3;  // x = x - 3 => 12
x *= 2;  // x = x * 2 => 24
x /= 4;  // x = x / 4 => 6
x %= 3;  // x = x % 3 => 0
```

3. **Operadores de Comparación**

```javascript
console.log(5 == 5);   // true (igualdad no estricta)
console.log(5 === '5'); // false (igualdad estricta)
console.log(5 != 3);   // true (desigualdad no estricta)
console.log(5 !== '5'); // true (desigualdad estricta)
console.log(5 > 3);    // true (mayor que)
console.log(5 < 3);    // false (menor que)
console.log(5 >= 5);   // true (mayor o igual que)
console.log(5 <= 3);   // false (menor o igual que)
```

4. **Operadores Lógicos**

```javascript
let a = true;
let b = false;

console.log(a && b);  // false (AND lógico)
console.log(a || b);  // true (OR lógico)
console.log(!a);      // false (NOT lógico)
```

5. **Operadores Bit a Bit**

```javascript
let x = 5;  // 0101 en binario
let y = 3;  // 0011 en binario

console.log(x & y);  // AND bit a bit: 1 (0001)
console.log(x | y);  // OR bit a bit: 7 (0111)
console.log(x ^ y);  // XOR bit a bit: 6 (0110)
console.log(~x);     // Negación bit a bit: -6 (invertir bits)
console.log(x << 1); // Desplazamiento a la izquierda: 10 (1010)
console.log(x >> 1); // Desplazamiento a la derecha: 2 (0010)
```

6. **Operadores de Tipo**

```javascript
console.log(typeof 42);       // "number"
console.log(typeof 'hello');  // "string"
console.log(typeof true);     // "boolean"
console.log(typeof undefined); // "undefined"
console.log(typeof {});       // "object"
console.log('hello' instanceof String); // false
```

7. **Operadores de Acceso**

```javascript
let persona = { nombre: 'Juan', edad: 30 };
console.log(persona.nombre);  // Acceso a propiedad con '.'
console.log(persona['edad']); // Acceso a propiedad con '[]'

let array = [1, 2, 3];
console.log(array[1]);  // Acceso a índice del array: 2
```

8. **Operador Ternario (Condicional)**

```javascript
let edad = 18;
let mensaje = edad >= 18 ? "Eres mayor de edad" : "Eres menor de edad";
console.log(mensaje);  // "Eres mayor de edad"
```

9. **Operadores de Desestructuración**

```javascript
let persona = { nombre: 'Ana', edad: 25 };
let { nombre, edad } = persona;
console.log(nombre); // "Ana"
console.log(edad);   // 25

let array = [1, 2, 3];
let [primer, segundo] = array;
console.log(primer);  // 1
console.log(segundo); // 2
```

10. **Operadores de Spread/Rest**

```javascript
// Spread
let arr = [1, 2, 3];
let arr2 = [...arr, 4, 5];
console.log(arr2);  // [1, 2, 3, 4, 5]

// Rest
function sumar(...numeros) {
    return numeros.reduce((acc, num) => acc + num, 0);
}
console.log(sumar(1, 2, 3));  // 6
```

11. **Operadores de Funciones Generadoras**

```javascript
function* generarNumeros() {
    yield 1;
    yield 2;
    yield 3;
}

let generador = generarNumeros();
console.log(generador.next().value); // 1
console.log(generador.next().value); // 2
console.log(generador.next().value); // 3
```

12. **Palabras Clave y Símbolos Relacionados con Clases**

```javascript
class Animal {
    constructor(nombre) {
        this.nombre = nombre;
    }

    hablar() {
        console.log(`${this.nombre} hace un sonido`);
    }
}

let perro = new Animal('Rex');
perro.hablar();  // "Rex hace un sonido"
```

13. **Símbolos de Control de Flujo**

```javascript
let x = 3;

if (x > 5) {
    console.log("Mayor que 5");
} else {
    console.log("Menor o igual que 5");
}

for (let i = 0; i < 3; i++) {
    console.log(i);  // 0, 1, 2
}
```

14. **Delimitadores**

```javascript
let mensaje = "Hola Mundo";   // Uso de comillas dobles
let array = [1, 2, 3];        // Uso de corchetes []
let objeto = { clave: "valor" };  // Uso de llaves {}
```

15. **Comentarios**

```javascript
// Este es un comentario de una sola línea

/*
Este es un comentario
de varias líneas
*/
```

16. **Símbolos de Expresiones Regulares**

```javascript
let regex = /^a.*b$/;
console.log(regex.test("abc"));  // true
```

17. **Operadores de Módulos de Importación/Exportación**

```javascript
// export.js
export const saludo = "Hola";

// import.js
import { saludo } from './export.js';
console.log(saludo);  // "Hola"
```

18. **Símbolos de Promesas**

javascript

```javascript
let promesa = new Promise((resolve, reject) => {
    let exito = true;
    if (exito) {
        resolve("Operación exitosa");
    } else {
        reject("Error en la operación");
    }
});

promesa.then((mensaje) => {
    console.log(mensaje);  // "Operación exitosa"
}).catch((error) => {
    console.log(error);    // En caso de error
});
```