10. Tipos de Operadores

Los **operadores** son símbolos o palabras clave que permiten realizar operaciones sobre valores o variables. En JavaScript, hay varios tipos de operadores que cubren una amplia gama de necesidades, desde cálculos matemáticos hasta evaluaciones lógicas y manipulación de datos. En este capítulo, exploraremos los operadores más comunes y sus características.

---

1. Operadores Aritméticos y su Precedencia

Los **operadores aritméticos** se utilizan para realizar operaciones matemáticas básicas como suma, resta, multiplicación y división.

Operadores Aritméticos Comunes:

- `+`: Suma
- : Resta
- : Multiplicación
- `/`: División
- `%`: Módulo (resto de la división)
- `*`: Exponenciación

Ejemplo:

```
let a = 10;
let b = 3;

console.log(a + b); // 13
console.log(a - b); // 7
console.log(a * b); // 30
console.log(a / b); // 3.333...
console.log(a % b); // 1 (resto de 10 ÷ 3)
console.log(a ** b); // 1000 (10 elevado a 3)
```

Precedencia de Operadores

La **precedencia** determina el orden en que se evalúan las operaciones en una expresión. Los operadores con mayor precedencia se evalúan primero. Puedes usar paréntesis para forzar un orden específico.

Orden de Precedencia:

1. Paréntesis `()`
2. Exponenciación `*`
3. Multiplicación , División `/`, Módulo `%`
4. Suma `+`, Resta

Ejemplo:

```
let resultado = 10 + 5 * 2; // Multiplicación tiene mayor precedencia
console.log(resultado); // 20

resultado = (10 + 5) * 2; // Paréntesis cambian el orden
console.log(resultado); // 30
```

---

2. Operadores Relacionales

Los **operadores relacionales** comparan dos valores y devuelven un valor booleano (`true` o `false`). Son útiles en estructuras condicionales y bucles.

Operadores Relacionales Comunes:

- `>`: Mayor que
- `<`: Menor que
- `>=`: Mayor o igual que
- `<=`: Menor o igual que
- `==`: Igualdad (sin considerar el tipo)
- `===`: Igualdad estricta (considera el tipo)
- `!=`: Desigualdad (sin considerar el tipo)
- `!==`: Desigualdad estricta (considera el tipo)

Ejemplo:

```
let a = 5;
let b = "5";

console.log(a > 3); // true
console.log(a == b); // true (compara solo el valor)
console.log(a === b); // false (compara valor y tipo)
console.log(a != b); // false
console.log(a !== b); // true
```

---

3. Operadores de Incremento y Decremento

Los **operadores de incremento (`++`) y decremento (`--`)** aumentan o disminuyen el valor de una variable en 1.

Formas:

- **Prefijo**: El operador se aplica antes de que se evalúe la variable.
- **Postfijo**: El operador se aplica después de que se evalúe la variable.

Ejemplo:

```
let x = 5;

console.log(++x); // 6 (incremento prefijo)
console.log(x++); // 6 (incremento postfijo, pero se incrementa después)
console.log(x);   // 7

let y = 10;
console.log(--y); // 9 (decremento prefijo)
console.log(y--); // 9 (decremento postfijo, pero se decrementa después)
console.log(y);   // 8
```

4. Operador Unario

Un **operador unario** es un operador que actúa sobre un solo operando. En JavaScript, algunos ejemplos incluyen:

- `+`: Convierte un valor a número.
- : Convierte un valor a número y lo niega.
- `typeof`: Devuelve el tipo de dato de un valor.
- `delete`: Elimina una propiedad de un objeto.

Ejemplo:

```
let texto = "123";
console.log(+texto); // 123 (conversión a número)

let numero = 42;
console.log(-numero); // -42

console.log(typeof "hola"); // string
console.log(typeof 42);     // number

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

5. Operadores Lógicos

Los **operadores lógicos** se utilizan para combinar o modificar valores booleanos. Son esenciales para construir condiciones complejas.

Operadores Lógicos Comunes:

- `&&` (AND): Devuelve `true` si ambos operandos son verdaderos.
- `||` (OR): Devuelve `true` si al menos uno de los operandos es verdadero.
- `!` (NOT): Invierte el valor booleano de un operando.

Ejemplo:

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

console.log(a && b); // false (ambos deben ser true)
console.log(a || b); // true (al menos uno debe ser true)
console.log(!a);     // false (invierte el valor de a)
```

Evaluación Cortocircuitada

Los operadores `&&` y `||` usan **evaluación cortocircuitada**, lo que significa que no siempre evalúan ambos operandos.

- `&&`: Si el primer operando es `false`, no evalúa el segundo porque ya sabe que el resultado será `false`.
- `||`: Si el primer operando es `true`, no evalúa el segundo porque ya sabe que el resultado será `true`.

Ejemplo:

```
let resultado = 0 || "valor predeterminado"; // "valor predeterminado"
console.log(resultado);

resultado = 1 && "valor final"; // "valor final"
console.log(resultado);
```