# Otros operadores de JavaScript

Los operadores que vimos en las lecciones anteriores son los que se utilizan con más frecuencia en JavaScript, sin embargo, existen muchos otros operadores que, aunque no se utilizan tanto, vale la pena conocerlos:

## Operador coma

Se utiliza para evaluar y ejecutar múltiples expresiones en secuencia, en una misma línea. Ahora bien, debes tener en cuenta que, aunque se evalúa cada expresión separada por una coma, sólo se devuelve el valor de la última expresión. Por ejemplo:

```
let valor = (5 + 2, 3 * 9, 10 + 20);
console.log(valor); // Devuelve 30
```

En el ejemplo anterior, aunque todas las expresiones han sido evaluadas, el operador coma sólo ha devuelto el valor de la última expresión.

## Operador Nullish coalescing

Este operador surge con la especificación ES2020, y es una herramienta muy útil para manejar valores nulos o indefinidos. Con este operador podemos seleccionar un valor de una lista en función de su disponibilidad, es decir, si el valor existe y no es *null* ni *undefined*. En términos más simples, devuelve el primer valor definido de una lista o el valor predeterminado si todos los valores de la lista son *null* o *undefined*.

La sintaxis básica del operador Nullish Coalescing es la siguiente:

*valor1 ?? valor2*

Veamos algunos ejemplos para entender mejor cómo funciona:

```
const nombre = null;
const valorPorDefecto = 'Usuario';
const nombreUsuario = nombre ?? valorPorDefecto;
console.log(nombreUsuario); // Resultado: Usuario
```

En este ejemplo, la variable `nombre` es nula, por lo que el operador Nullish Coalescing devuelve `valorPorDefecto`. Como resultado, el valor de `nombreUsuario` será "Usuario".

El operador nullish coalescing (unión nula) es un operador lógico muy similar al operador OR, sin embargo, difiere del operador OR (||) en cómo manejan los valores falsy (falsos). El operador Nullish Coalescing devuelve el segundo valor sólo si el primer valor es null o undefined, mientras que el operador OR devuelve el segundo valor si el primer valor es falsy (null, undefined, 0, "", false, NaN).

## Operador de Asignación lógica nula

Este operador permite asignar un valor predeterminado a una variable sólo si no tiene un valor asignado. La sintaxis básica del operador de Asignación Lógica Nula es la siguiente:

*variable ??= valor;*

Veamos un ejemplo para entender cómo funciona:

```
let nombre = null;
nombre ??= 'Usuario';
console.log(nombre); //  Usuario
```

En este ejemplo, la variable `nombre` es null, por tanto, el operador de Asignación Lógica Nula asigna 'Usuario' a la variable `nombre`. Como resultado, el valor será "Usuario". Veamos otro ejemplo:

```
let edad = 21;
edad ??= 18;
console.log(edad); // 21
```

En este caso, la variable `edad` tiene un valor asignado (21), por lo que el operador de Asignación Lógica Nula no realiza ninguna asignación y el valor permanece igual.

## Asignación a nivel de bit

El operador de asignación a nivel de bit es una herramienta que permite manipular los bits individuales de un número. En realidad, no se trata de un sólo operador sino de una serie de operadores que sirven para realizar diferentes acciones, y son los siguientes:

**Operador de desplazamiento a la izquierda (<<=)**: Desplaza los bits de un número hacia la izquierda y asigna el resultado.

```
let num = 5; // Representación binaria: 00000101
num <<= 2;
console.log(num); // Resultado: 20
```

En este ejemplo, el valor de `num` se desplaza dos bits hacia la izquierda utilizando el operador `<<=`. La representación binaria de `num` cambia a 00010100, que en decimal es 20.

**Operador de desplazamiento a la derecha (>>=)**: Desplaza los bits de un número hacia la derecha y asigna el resultado. Veamos otro ejemplo:

```
const edad = 21;
const edadPorDefecto = 18;
const edadUsuario = edad ?? edadUsuario;
console.log(edadUsuario); // 21
```

En este ejemplo, la variable `edad` tiene un valor definido (21), por lo que el operador Nullish Coalescing devuelve ese valor. El valor de `edadUsuario` será 21.

```
let num = 20; // Representación binaria: 00010100
num >>= 2;
console.log(num); // Resultado: 5
```

En este ejemplo, el valor de `num` se desplaza dos bits hacia la derecha utilizando el operador `>>=`. La representación binaria de `num` cambia a 00000101, que en decimal es igual a 5.

**- Operador de desplazamiento a la derecha sin signo (>>>=)**: Desplaza los bits de un número hacia la derecha y asigna el resultado rellenando con ceros.

```
let num = -10; // Representación binaria: 11110110
num >>>= 2;
console.log(num); // Output: 1073741821
```

En este caso, el valor de `num` se desplaza dos bits hacia la derecha utilizando el operador `>>>=`. La representación binaria de `num` cambia a 00111101 11111111 11111111 11111101, que en decimal es 1073741821.

## Operador de encadenamiento opcional

El operador Optional chaining o de encadenamiento opcional surge con la especificación ES2020 y permite acceder a las propiedades de un objeto de forma segura sin provocar errores si alguna de las propiedades en la cadena de acceso es null o undefined. Si alguna de las propiedades es null o undefined, el operador devuelve undefined en lugar de lanzar un error. La sintaxis de este operador es la siguiente:

objeto? Propiedad

objeto? Método ()

Como puedes ver, el operador de Encadenamiento Opcional se coloca después del objeto y antes de la propiedad o método al que se desea acceder. Si el objeto es null o undefined, la expresión completa devuelve undefined. Si el objeto no es null ni undefined, se accede a la propiedad o se llama al método.

```
const persona = {
  nombre: 'Thomas',
  edad: 22,
  direccion: {
    calle: 'Calle Principal',
    ciudad: 'Los Ángeles',
  },
};
const ciudad = persona.direccion.ciudad; // Forma convencional
console.log(ciudad); // Devuelve Los Angeles
const ciudadSegura = persona?.direccion?.ciudad; // Con operador de Encadenamiento Opcional
console.log(ciudadSegura); // Devuelve Los Angeles
```

Al utilizar la forma convencional, si alguna de las propiedades en la cadena de acceso es null o undefined, se lanzaría un error. Sin embargo, al utilizar el operador de Encadenamiento Opcional, si alguna de las propiedades es null o undefined, la expresión completa devuelve undefined en lugar de lanzar un error.