# Modulo 3. Operadores e interacciones con el usuario

Los operandos pueden ser tanto valores como variables.

Los operadores se pueden categorizar de varias maneras. Se distinguen, por ejemplo, por el número de operandos sobre los que trabajan. El operador de suma `+` es un **operador binario** típico (utiliza dos operandos), mientras que el operador `typeof` es **unario** (utiliza un solo operando). Podemos diferenciar entre **operadores de prefijo** (que ocurren antes del operando), **operadores de postfijo** (después del operando) y **operadores de infijo** (entre operandos). Sin embargo, es común categorizar a los operadores según el contexto en el que se usan: así tenemos **de asignación; aritmético; lógico;** u **operadores condicionales**. El mismo símbolo puede interpretarse como un operador diferente según el contexto.

En JavaScript, el símbolo `+` es un ejemplo. Si los operandos son números, el uso de este operador hará que el intérprete calcule su suma (es un operador de suma, clasificado como aritmético). Sin embargo, si los operandos son cadenas, el mismo símbolo se tratará como un operador de concatenación y el intérprete intentará unir ambas cadenas de caracteres.

## Operadores de asignacion

En este grupo se encuentran los operadores que permiten asignar valores a variables y constantes. El operador de asignación básico es el signo igual =

Si aparecen varios operadores de asignación en una secuencia, se aplica el orden de derecha a izquierda. Entonces la secuencia:

```
let year = 2050;
let newYear = year = 2051;
```

Es lo mismo que

```
let year = 2050;
year = 2051;
let newYear = year;
```

## Operadores de aritmeticos

Los operadores aritméticos expresan operaciones matemáticas y aceptan valores numéricos y variables. Todos los operadores aritméticos, excepto la suma, intentarán convertir implícitamente los valores al tipo Number antes de realizar la operación.

El operador de suma convertirá todo a String si alguno de los operandos es de tipo String, de lo contrario los convertirá a Number como el resto de los operadores aritméticos. El orden de las operaciones se respeta en JavaScript como en matemáticas, y podemos usar paréntesis como en matemáticas para cambiar el orden de las operaciones si es necesario.

Los operadores aritméticos binarios básicos son la suma `+`, la resta `-`, la multiplicación `*`, la división `/`, residuo de la división `%` y la potencia `**`. Su funcionamiento es análogo a lo que sabemos por las matemáticas.

```
const x = 5;
const y = 2;

console.log("suma: ", x + y); // -> 7
console.log("resta: ", x - y); // -> 3
console.log("multiplicación: ", x * y); // -> 10
console.log("división: ", x / y); // -> 2.5
console.log("residuo de la división: ", x % y); // -> 1
console.log("potencia: ", x ** y); // -> 25
```

## Operadores aritméticos unarios

También existen varios operadores aritméticos unarios (que operan en un solo operando). Entre ellos se encuentran los operadores de más `+` y menos `-`.

Ambos operadores convierten los operandos al tipo Number, mientras que el operador de menos (negativo) lo niega.

```
let str = "123";
let n1 = +str;
let n2 = -str;
let n3 = -n2;
```

```
let n4 = +"abcd";

console.log(`${str} : ${typeof str}`); // -> 123 : string
console.log(`${n1} : ${typeof n1}`); // -> 123 : number
console.log(`${n2} : ${typeof n2}`); // -> -123 : number
console.log(`${n3} : ${typeof n3}`); // -> 123 : number
console.log(`${n4} : ${typeof n4}`); // -> NaN : number
```

## Operadores unarios de incremento y decremento

Entre los operadores aritméticos, también tenemos a nuestra disposición los operadores de **incremento** `++` y **decremento** `--` unario, tanto en versiones de prefijo como de sufijo. Nos permiten aumentar (incrementar) o disminuir (decrementar) el valor del operando en 1.

Estos operadores en la versión de sufijo (es decir, el operador está en el lado derecho del operando) realizan la operación cambiando el valor de la variable, pero devuelven el valor antes del cambio. La versión de prefijo del operador (es decir, el operador se coloca antes del operando) realiza la operación y devuelve el nuevo valor. Esto es importante ya que el interprete lo vera de la siguiente manera:

```
console.log(n1++);
//console.log(n1);
//n1 = n1 + 1;

//Mientras que

console.log(++n1);
//n1 = n1 + 1;
//console.log(n1);
```

## Operadores de asignacion compuesta

**Los operadores aritméticos binarios** se pueden combinar con el **operador de asignación**, dando como resultado la asignación de suma `+=`, la asignación de resta `-=`, la asignación de la multiplicación `*=`, la asignación de la división `/=`, la asignación del residuo `%=` y la asignación de potencia `**=`.

Cada uno de estos operadores toma un valor de la variable a la que se va a realizar la asignación (el operando izquierdo) y lo modifica realizando una operación

aritmética utilizando el valor del operando derecho. El nuevo valor se asigna al operando izquierdo.

```
x += 1;
//Es lo mismo que
x = x + 1;
```

## Operadores logicos

Los operadores lógicos funcionan con valores de tipo booleano (`true` o `false`). Por ahora, podemos suponer que funcionan con operandos de este tipo y devuelven valores de este tipo únicamente. JavaScript nos proporciona tres de estos operadores:

- una conjunción, es decir, AND (Y) lógico (símbolo `&&`)
- una disyunción, es decir, OR (O) lógico (símbolo `||`)
- una negación, es decir, NOT (NO) lógico (símbolo `!`)

Los operadores lógicos se suelen utilizar junto con los **operadores condicionales**, y son especialmente útiles en **instrucciones condicionales** (toma de decisiones) y en **bucles** (condiciones de fin de ciclo).

## Operadores de Asignación Compuesta

Al igual que los operadores aritméticos, los **operadores lógicos binarios** se pueden usar en combinación con el operador de asignación, creando una asignación AND lógica `&&=` y una asignación OR lógica `||=`.

```
let a = true;
console.log(a); // -> true
a &&= false;
console.log(a); // -> false
```

La instrucción `a &&= false` significa exactamente lo mismo que `a = a && false`.

Para las operaciones OR funciona de manera similar.

```
let b = false;
console.log(b); // -> false
b ||= true;
console.log(b); // -> true
```

Esta vez, la operación `b ||= true` se interpreta como `b = b || true`.

## Operadores de cadena

El único operador en este grupo es la **concatenación** `+`. Este operador convertirá todo a una **cadena** si alguno de los operandos es de tipo String. Finalmente, creará una nueva cadena de caracteres, adjuntando el operando derecho al final del operando izquierdo.

```
let greetings = "Hi";
console.log(greetings + " " + "Alice"); // -> Hi Alice

let sentence = "Happy New Year ";
let newSentence = sentence + 10191;

console.log(newSentence); // -> Happy New Year 10191
console.log(typeof newSentence); // -> string
```

## Operadores de asignacion compuesta

Este operador también se puede usar junto con el operador de reemplazo. Su funcionamiento es muy intuitivo.

```
let sentence = "Happy New ";
sentence += "Year ";
sentence += 10191;
console.log(sentence); // -> Happy New Year 10191
```

## Operadores de comparacion

Los operadores de comparación se utilizan para verificar la igualdad o desigualdad de valores. **Todos los operadores de comparación son binarios y todos**

**devuelven un valor lógico que representa el resultado de la comparación, `true` o `false`.**

Al igual que con otros operadores, JavaScript intentará convertir los valores que se comparan si son de diferentes tipos. Tiene sentido verificar la igualdad, o cuál es mayor, usando la representación numérica, y JavaScript en la mayoría de los casos convertirá los tipos a Number antes de la comparación. Hay dos excepciones a esto, las cadenas y el **operador de identidad (igualdad estricta)**. Las cadenas se comparan `char` por `char` (especificamente carácter Unicode por carácter Unicode usando sus valores).

Para verificar si los operandos son iguales, podemos usar el operador de **identidad** (igualdad estricta) `===` o el operador de **igualdad** ==.

El primero es más restrictivo y, para devolver verdadero, los operandos deben ser idénticos (es decir, deben ser iguales y del mismo tipo).

También tenemos operadores que nos permiten comprobar si uno de los operandos es mayor que `>`, menor que `<`, mayor o igual que `>=`, y menor o igual que `<=`. Estos operadores funcionan en cualquier tipo de operando, pero tiene sentido usarlos solo en números o valores que se convertirán correctamente en números.

## Operador ternario

Este operador funciona de la misma forma que un if. Es un operador condicional. Según el valor del primer operando (verdadero o falso), se devuelve el valor del segundo o tercer operando, respectivamente. Este operador se usa con mayor frecuencia para colocar uno de los dos valores en la variable dependiendo de una determinada condición.

```
let name = 1 > 2 ? "Alice" : "Bob";
console.log(name); // -> Bob
```

## Precedencia

El intérprete de JavaScript utiliza dos propiedades de operador para determinar la secuencia de operaciones: precedencia y asociatividad. La precedencia se puede tratar como una prioridad, con algunos operadores que tienen la misma precedencia

(por ejemplo, suma y resta). La asociatividad le permite especificar el orden de ejecución si hay varios operadores con las mismas prioridades uno al lado del otro.

| Precedencia | Operador | Asociatividad | Símbolo |
|---|---|---|---|
| 14 | Agrupamiento | n/a | ( … ) |
| 13 | Acceso al campo | ⇒ | … . … |
| 12 | Llamada de función | ⇒ | … ( … ) |
| 11 | Incremento postfijo | n/a | … ++ |
| | Decremento postfijo | n/a | … -- |
| 11 | NOT (NO) lógico | ⇐ | ! … |
| | Signo de más unario | ⇐ | + … |
| | Negación unaria | ⇐ | - … |
| | Incremento prefijo | ⇐ | ++ … |
| | Decremento prefijo | ⇐ | -- … |
| | Tipo de dato | ⇐ | typeof … |
| | Eliminar | ⇐ | delete … |
| 9 | Exponenciación | ⇐ | … ** … |

| | | | | |
|---|---|---|---|---|
| 9 | Exponenciación | | ⇐ | ... ** ... |
| 8 | Multiplicación | ⇒ | | ... * ... |
| | División | ⇒ | | ... / ... |
| | Residuo | ⇒ | | ... % ... |
| 7 | Suma | ⇒ | | ... + ... |
| | Resta | ⇒ | | ... - ... |
| 6 | Menor que | ⇒ | | ... < ... |
| | Menor o igual que | ⇒ | | ... <= ... |
| | Mayor que | ⇒ | | ... > ... |
| | Mayor o igual que | ⇒ | | ... >= ... |
| | instancia de | ⇒ | | ... instanceof ... |
| 5 | Igualdad | ⇒ | | ... == ... |
| | Desigualdad | ⇒ | | ... != ... |
| | Igualdad estricta | ⇒ | | ... === ... |
| | Desigualdad estricta | ⇒ | | ... !== ... |
| 4 | AND (Y) lógico | ⇒ | | ... && ... |
| 3 | OR (O) lógico | ⇒ | | ... \|\| ... |
| 2 | Condicional (ternary) | | ⇐ | ... ? ... : ... |
| 1 | Asignación | | ⇐ | ... = ...<br>... += ...<br>... *= ...<br>... y otros operadores de asignación |

Una flecha en la columna de asociatividad que apunta hacia el lado derecho significa asociatividad de izquierda a derecha, mientras que hacia el lado opuesto significa asociatividad de derecha a izquierda.

La abreviatura n/a significa no aplicable, porque en algunos operadores el término asociatividad no tiene sentido.

- **Agrupar** es simplemente usar paréntesis. Tienen prioridad sobre los demás operadores, por lo que podemos usarlos para forzar la ejecución de operaciones para que tengan prioridad;
- **Acceso al campo (acceso al miembro)** es el operador que se usa en la notación de puntos, que es cuando se accede a un campo de objeto seleccionado. Tiene prioridad sobre otros operadores (excepto los paréntesis), por ejemplo, la instrucción:

  `let x = myObject.test + 10;` significa que el valor del campo test del objeto myObject se obtendrá primero, luego le sumaremos un valor de 10, y el resultado irá a la variable x;
- La precedencia **llamada de función** nos dice que si llamamos a una función, esta acción tendrá prioridad sobre otras operaciones, excepto la agrupación entre paréntesis y el operador de acceso al campo (puntos). Así que en el ejemplo:

  `let y = 10 + myFunction() ** 2;` myFunction será llamada primero, el resultado devuelto será elevado a potencia 2, y solo entonces sumaremos 10 al total y guardaremos el resultado en la variable y.

## Interacción con el usuario

Las interacciones con el usuario son necesarias, por que con la data que nos ingrese, podemos devolver otra data o ejercer operaciones con la misma.

- cuadros de dialogo: Los cuadros de diálogo son partes integrales de los navegadores web y están disponibles en casi todos ellos, incluso en los más antiguos. Todos ellos son ventanas emergentes (o ventanas modales), lo que significa que cuando se muestra el cuadro de diálogo, no es posible interactuar con la página web en sí hasta que se cierra este cuadro de diálogo.