Operadores en JavaScript

Los operadores se pueden clasificar de varias maneras. Se distinguen, por ejemplo, por el número de operandos sobre los que trabajan.
El operador de adición **+** es un operador binario típico (utiliza dos operandos), mientras el operador **typeof** es unario (utiliza solo un operando).
JavaScript ofrece una amplia gama de operadores que permiten manipular datos y realizar cálculos. Estos símbolos pueden ser unitarios, binarios o ternarios, y su función depende del contexto en el que se utilizan, por ejemplo el operador **+,** puede usarse también para concatenar.

Operadores de asignación

El operador de asignación básico es el signo igual **=**. Este operador asigna el valor del operando derecho al operando izquierdo.

```
const nombre = "Alejandra";
```

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

```
let anio = 2050;
let nuevoAnio = anio = 2051;
```

Es lo mismo que:

```
let anio = 2050;
anio = 2051;
let nuevoAnio = anio;
```

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 Número antes de realizar la operación.

El operador de suma convertirá todo en una cadena si alguno de los operandos es de tipo cadena, de lo contrario los convertirá en un número como el resto de 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.

En general, es una buena costumbre utilizar paréntesis para forzar la precedencia y el orden de las operaciones, no solo las aritméticas. La precedencia de las operaciones realizadas por el intérprete no siempre será tan intuitiva como la precedencia de las operaciones aritméticas conocidas en matemáticas.

```
console.log(2 + 2 * 2); // -> 6
console.log(2 + (2 * 2)); // -> 6
console.log((2 + 2) * 2); // -> 8
```

Los operadores aritméticos binarios básicos son la suma+, resta-, multiplicación*, división/, resto de división % y potencia ** . Su funcionamiento es análogo al que conocemos de las matemáticas

```
const x = 5;
const y = 2;
console.log("suma: ", x + y); // -> 7
console.log("resta: ", x - y); // -> 3
console.log("multiplicacion: ", x * y); // -> 10
console.log("division: ", x / y); // -> 2.5
console.log("division resto :", x % y); // -> 1
console.log("potencia.: ", x ** y); // -> 25
```

Ejemplo de precedencia y tipos de datos:

let resultado = 2 + 3 * 4; // Resultado: 14 (la multiplicación se realiza antes de la suma)
resultado = (2 + 3) * 4; // Resultado: 20 (los paréntesis fuerzan la suma antes de la multiplicación)

let numero = "5";
let resultado2 = numero * 2; // Resultado: 10 (la cadena "5" se convierte a número)

let texto = "Hola" + 2; // Resultado: "Hola2" (el número 2 se convierte a cadena y se concatena)

Operadores aritméticos unarios incompletos

También existen varios operadores aritméticos unarios (que operan sobre un solo operando). Entre ellos se encuentran el signo más.+y menos-.

```
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 el incremento unario **++** y decremento **--**

```
let n1 = 10;
let n2 = 10;

console.log(n1); // -> 10
console.log(n1++); // -> 10
console.log(n1); // -> 11

console.log(n2); // -> 10
console.log(++n2); // -> 11
console.log(n2); // -> 11

let n3 = 20;
let n4 = 20;

console.log(n3); // -> 20
console.log(n3--); // -> 20
console.log(n3); // -> 19

console.log(n4); // -> 20
console.log(--n4); // -> 19
console.log(n4); // -> 19
```

Tener en cuenta que el tipo Número es un tipo de punto flotante, lo que significa que los resultados de algunas de las operaciones pueden ser imprecisos.

```
console.log(0.2 + 0.1); // 0.30000000000000004
console.log(0.2 * 0.1); // 0.020000000000000004
console.log(0.3 / 0.1); // 2.9999999999999996
```

El número será preciso para los números enteros de hasta 252, pero las fracciones pueden no ser tan precisas, ya que muchas fracciones son imposibles de representar directamente en formato binario. Analizaremos cómo mitigar esto en un momento cuando presentemos los operadores de comparación.

Operadores de asignación compuestos

Los operadores aritméticos binarios se pueden combinar con el operador de asignación, lo que da como resultado la asignación de suma.**+=**, la resta**-=**, la multiplicación***=**, la asignación de división**/=**, la asignación resto **%=**, y la asignación potencia **=.

Cada uno de estos operadores toma un valor de la variable a la que se le 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. Por ejemplo, el fragmento de código siguiente:

```
x += 100;
```

Podría escribirse:

```
x = x + 100;
```

entonces tenemos:

```
let x = 10;
x += 2;
console.log(x); // -> 12
x -= 4;
console.log(x); // -> 8
x *= 3;
console.log(x); // -> 24
x /= 6;
console.log(x); // -> 4
x **= 3;
console.log(x); // -> 64
x %= 10;
console.log(x); // -> 4
```

Operadores lógicos

Los operadores lógicos trabajan con valores de tipo booleano (verdadero o falso). JavaScript nos proporciona tres operadores de este tipo:

- una conjunción, es decir, un AND lógico (&&)
- una alternativa, es decir OR lógico (||)
- una negación, es decir, un NO lógico (!)

Empecemos con la conjunción. Se trata de un operador binario que devuelve verdadero si ambos operandos son verdaderos. Utilizando oraciones lógicas, podemos imaginar una oración que consta de dos enunciados simples conectados por un AND, por ejemplo:

Buenos Aires es una ciudad **y** Buenos Aires esta en Argentina

En este caso, ambas afirmaciones son verdaderas y, tras combinarlas con AND, se crea una oración que también es verdadera. Si alguna de estas afirmaciones fuera falsa (o ambas lo fueran), la oración completa también sería falsa, por ejemplo:

Buenos Aires es una ciudad **y** Buenos Aires esta en Brasil

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

En el caso de una alternativa que también sea un operador binario, basta con que uno de los operandos sea verdadero para que el operador devuelva verdadero. Volviendo a nuestro ejemplo con oraciones lógicas, utilicemos una oración formada por dos enunciados conectados por un operador OR, por ejemplo:

Buenos Aires es una capital **O** Buenos Aires esta en Brasil

La frase puede no parecer demasiado elocuente o sensata, pero desde el punto de vista de la lógica es bastante correcta. Basta con que una de las afirmaciones sea verdadera para que toda la frase sea también verdadera. Si ambas afirmaciones son falsas, entonces la frase también será falsa, por ejemplo:

Buenos Aires es un barrio **O** Buenos Aires esta en Brasil

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

El operador de **negación** es un operador unario y cambia el valor lógico del operando por su opuesto, es decir, de falso a verdadero y de verdadero a falso. Mediante oraciones lógicas, podemos representarlo con la negación NOT. Tomemos como ejemplo una oración simple que sea verdadera:

Buenos aires es una capital

Añadiéndole la negación cambiamos su valor a falso:

Buenos aires NO es una capital

De la misma manera, funcionará a la inversa, convirtiendo una oración falsa en una verdadera.

```
console.log(!true); // -> false
console.log(!false); // -> true
```

Por supuesto, podemos conectar tantos operadores como necesitemos, creando "oraciones" más complejas. Como en el caso de los operadores aritméticos, aquí se determina la secuencia de acciones. La prioridad más alta es la negación.!, entonces conjunción&&, y finalmente la alternativa||Por supuesto, la precedencia se puede cambiar mediante paréntesis.

```
const a = false;
const b = true;
const c = false;
const d = true;
console.log(a && b && c || d); // -> true
console.log(a && b && (c || d)); // -> false
```

Ejemplo 1: console.log(a && b && c || d);

- Paso 1: Se evalúa a && b. Como a es false, toda la expresión a && b es false.
- Paso 2: Se evalúa (false) && c. Como false && cualquier_valor siempre es false, el resultado es false.
- Paso 3: Se evalúa false || d. Como d es true, la disyunción es true.

Por lo tanto, el resultado final es true.

Ejemplo 2: console.log(a && b && (c || d));
Paso 1: Se evalúa c || d. Como d es true, el resultado es true.
Paso 2: Se evalúa a && b && true. Como a es false, toda la expresión es false.
(En este caso, los paréntesis fuerzan a que se evalúe primero c || d antes de aplicar la conjunción con a y b.)

Por último, Al igual que en las matemáticas, los operadores lógicos tienen una jerarquía de evaluación. Esto significa que algunas operaciones se realizan antes que otras. En este caso, la precedencia es la siguiente:

- Negación (!): Tiene la mayor prioridad.
- Conjunción (&&): Tiene mayor prioridad que la disyunción.
- Disyunción (||): Tiene la menor prioridad.

Ejercicios:

1- Complete los operadores faltantes para obtener el resultado esperado (reemplace el símbolo de guión bajo con el operador apropiado):

```
console.log(2 _ 3 _ 1);     // expected 7
console.log(2 _ 4);      // expected 16
console.log(5 _ 1);      // expected 5
console.log(8 _ 2 _ 5 _ 2); // expected 39
```

2- Complete los operadores de comparación faltantes de tal manera que todas las expresiones resulten enverdadero(reemplace el símbolo de guión bajo con el operador apropiado):

```
console.log(4 * 5 _ 20);
console.log(6 * 5 _ "30");
console.log(-17 _ 0);
console.log(25 _ 1);
console.log(2 + 2 * 2 _ 4);
```

3- Complete los operadores de comparación faltantes de tal manera que todas las expresiones resulten enverdadero (reemplace el símbolo de guión bajo con el operador apropiado):

```
console.log(true _ false);
console.log(false _ false);
console.log(false _ false _ true);
console.log(true _ false _ false && true);
```