Datos y operadores en JavaScript

Un tipo de datos sirve para definir el tipo y el comportamiento de los datos: En otras palabras, con esto podemos decirle al compilador o al intérprete cómo pretendemos usar una parte de los datos. La mayoría de los lenguajes de programación admiten tipos de datos básicos como números, booleanos, cadenas, etc.
En JavaScript podemos decir que admite ocho tipos de datos: número, BigInt, booleano, string o cadena, valor nulo, indefinido, símbolo y objeto.

- **Número (number)**: El tipo number representa tanto números enteros como de punto flotante.
- **BigInt**: En JavaScript, el tipo "number" no puede representar valores enteros mayores que (253-1) (eso es 9007199254740991), o menor que -(253-1) para negativos. Es una limitación técnica causada por su representación interna. Para la mayoría de los propósitos es suficiente, pero a veces necesitamos números realmente grandes. Por ejemplo, para criptografía o marcas de tiempo de precisión de microsegundos.
- **String**: Un string en JavaScript es una cadena de caracteres y debe colocarse entre comillas.
- **Boolean (tipo lógico)**: El tipo booleano tiene sólo dos valores posibles: true y false.
- **El valor "null" (nulo)**: El valor especial null no pertenece a ninguno de los tipos descritos anteriormente.
- **El valor "undefined" (indefinido)**: El valor especial undefined también se distingue. Hace un tipo propio, igual que null. El significado de undefined es "valor no asignado".
- **Object y Symbol**: El tipo object (objeto) es especial. Todos los demás tipos se llaman "primitivos" porque sus valores pueden contener una sola cosa (ya sea una cadena, un número o lo que sea). Por el contrario, los objetos se utilizan para almacenar colecciones de datos y entidades más complejas. El tipo symbol

(símbolo) se utiliza para crear identificadores únicos para los objetos.

Para simplificarlo un poco más los tipos de datos en JavaScript se pueden clasificar ampliamente en dos categorías: tipos de datos primitivos y tipos de datos no primitivos. Donde un objeto es un tipo de datos no primitivo o complejo, y el resto son tipos de datos primitivos.
Pero algo que debemos tener en cuenta en JavaScript, y que a veces complica las cosas un poco, es que se trata de un lenguaje de tipo dinámico, lo que significa que los tipos de variables se comprueban durante el tiempo de ejecución. La misma variable puede contener valores de diferentes tipos en cualquier momento.

Por ejemplo:

```
// x es un string o cadena
let x = "Hola Mundo";
// x es un número
x = 100;
// x es un booleano
x = true;
```

Esto hace que si bien el programa no se romperá durante su tiempo de ejecución como ocurre en Java, si podemos encontrar comportamientos inesperados si no tenemos claro cómo trabajar con los diferentes tipos de operadores.

Tipos de operadores de JavaScript

Estos son los diversos operadores que admite JavaScript:

- Operadores Aritméticos
- Operadores de Asignación
- Operadores de cadena
- Operadores de comparación
- Operadores lógicos
- Operadores Bitwise
- Operadores especiales

1. Operadores Aritméticos

Con ellos podemos utilizar operadores aritméticos para realizar cálculos matemáticos en los operandos. JavaScript proporciona los siguientes operadores aritméticos:

Operador de suma (+) para realizar sumas en los operandos.

```
let a = 12;
let b = 10;
let result = a+b;
console.log(result)
console.log(1+2);
console.log(a+6);

Output
22
3
18
```

Operador de sustracción (-) para restar el operando derecho del operando izquierdo.

```
let a = 10;
let b = 4;
let result = a-b;
console.log(result);
console.log (23-20);

Output:
6
3
```

Operador de multiplicación (*) para multiplicar los operandos.

```
let a = 10;
let b = 4;
let result = a*b;
console.log(result);
console.log(23*20);

Output:
40
460
```

Operador de división (/) para realizar la división en los operandos.

```
let a = 10;
let b = 4;
let result = a/b;
console.log(result);
console.log(40/20);

Output:
2.5
2
```

Operador de módulo (%). En informática, la operación módulo obtiene el resto entero de la división de un número entre otro.

```
let a = 10;
let b = 4;
let result = a%b;
console.log(result);
console.log(40%20);

Output:
2
0
```

Operador de exponenciación (**) para calcular la base a la potencia del exponente (base^exponente).

```
let a = 3;
console.log(a**4);

Output:
81
```

Operador de incremento (++) para aumentar el valor entero en uno.

```
let a = 3;
// Value of a=4 and returns 4
console.log(++a);
// Value of a=5 and returns 4
console.log(a++);
// Value of a=5 and returns 5
console.log(a);

Output:
4
4
5
```

Operador de decremento (--) que disminuye el valor entero en uno.

```
let a = 3;
// Value of a=2 and returns 2
console.log(--a);
// Value of a=1 and returns 2
console.log(a--);
// Value of a=1 and returns 1
console.log(a);

Output:
2
2
1
```

Operador unario más (+) para intentar convertir el operando en un número si aún no lo es

```
console.log(typeof("10"));
console.log(typeof(+"10"));
console.log(typeof(false));
console.log(typeof(+false));

Output:
string
number
boolean
number
```

Operador de negación unaria (-), que devuelve la negación de su operando.

```
let a = 10;
console.log(-a);

Output:
```

-10

2. Operadores de asignación

Un operador de asignación asigna un valor a su operando izquierdo basándose en el valor de su operando derecho. Utilizamos operadores de asignación para asignar valores a las variables. JavaScript proporciona los siguientes operadores de asignación:
Operador de asignación (=) para asignar el valor del operando derecho al operando izquierdo.

```
// Assigning 10 to a
let a = 10;
console.log(a);

Output:
10
```

Operador de asignación de suma (+=) para sumar los valores de los operandos izquierdo y derecho y asignar el resultado al operando izquierdo.

```
let a = 10;
let b = 5;
// Equivalent to a = a+b
a += b;
console.log(a);

Output:
15
```

Operador de asignación de resta (-=) para restar el operando derecho del operando izquierdo y asignará el resultado al operando izquierdo.

```
let a = 10;
let b = 5;
// Equivalent to a = a-b
a -= b;
console.log(a);

Output:
5
```

Operador de asignación de multiplicación (*=) para multiplicar los valores de los operandos izquierdo y derecho y asigne el resultado al operando izquierdo.

```
let a = 10;
let b = 5;
// Equivalent to a = a*b
a *= b;
console.log(a);
Output:
50
```

Operador de asignación de división (/=) para dividir el valor del operando izquierdo por el valor del operando derecho y asignar el resultado al operando izquierdo.

```
let a = 10;
let b = 5;
// Equivalent to a = a/b
a /= b;
console.log(a);

Output:
2
```

Operador de asignación de resto (%=), divide el operando izquierdo por el operando derecho y asigna el resto al operando izquierdo.

```
let a = 10;
let b = 5;
// Equivalent to a = a%b
a %= b;
console.log(a);

Output:
0
```

Operador de Asignación de Exponenciación (=)**, eleva el operando izquierdo a la potencia del operando derecho y asigna el resultado al operando izquierdo.

```
let a = 10;
let b = 5;
// Equivalent to a = a**b
a **= b;
console.log(a);

Output:
100000
```

Nota: Los operadores de asignación bit a bit o Bitwise (>=, >>>=, &=, ^=, |=, &&=, ||=, ??=) que veremos más adelante funcionan de manera similar.

3. Operadores de cadenas string

El operador de concatenación (+) se usa para concatenar (agregar) cadenas de texto o también llamados strings.

```
let result = "If" + "Geek" + "Then";
console.log(result);

Output:
IfGeekThen
```

4. Operadores de comparación

Los operadores de comparación se utilizan para comparar operandos y devuelven un valor lógico (verdadero o falso) sobre la base de la comparación. JavaScript proporciona los siguientes operadores de comparación:

Operador igual (==), devuelve verdadero si los operandos son iguales. Solo compara los valores de los operandos, ignorando su tipo al comparar.

```
console.log(2==4);
console.log("2"==2);

Output:
false
true
```

Operador no igual (!=), devuelve verdadero si los operandos no son iguales. También ignora el tipo de operandos al comparar.

```
console.log(2!=4);
console.log(2!="2");

Output:
true
false
```

Operador de igualdad estricta (===), devuelve verdadero si los operandos son iguales. Compara tanto los valores como los tipos de operandos mientras compara.

console.log(2===4);
console.log("2"===2);

Output:
false
false

Operador estricto no igual (!==), devuelve verdadero si los operandos no son iguales. También compara ambos: los valores y el tipo de operandos mientras compara.

console.log(2 !== 4);
console.log(2 !== "2");

Output:
true
true

Operador mayor que (>), devuelve verdadero si el operando izquierdo es mayor que el operando derecho.

console.log(10>4);
console.log(5>5);

Output:
true
false

Operador mayor que o igual (>=), devuelve verdadero si el operando izquierdo es mayor o igual que el operando derecho.

console.log(10 >= 4);

console.log(5 >= 5);

Output:
true
true

Operador menor que (<), devuelve verdadero si el operando izquierdo es menor que el operando derecho.

console.log(10<4);
console.log(5<5);

Output:
false
false

Operador menor o igual (<=), devuelve verdadero si el operando izquierdo es menor o igual que el operando derecho.

console.log(10 <= 4);
console.log(5 <= 5);

Output:
false
true

5. Operadores Lógicos

AND lógico (&&)
Uso: expr1 && expr2
Devuelve expr1 si se puede convertir a falso; de lo contrario, devuelve expr2. Cuando se usa con valores booleanos, && devuelve verdadero si ambos operandos son verdaderos; de lo contrario, devuelve falso.

console.log(true || false);

Output:
true

NO lógico (!)
Uso: !expr
Devuelve falso si su único operando se puede convertir en verdadero; de lo contrario, devuelve verdadero.

console.log(!true);
Output:
false

6. Operadores bitwise

Una operación bit a bit o bitwise opera sobre números binarios a nivel de sus bits individuales. Es una acción primitiva rápida, soportada directamente por los procesadores. JavaScript proporciona los siguientes operadores bitwise
Operador bitwise AND(&)
Este operador realiza una operación AND booleana en cada bit de sus argumentos enteros.

// In binary-
// 4: 100
// 1: 001
console.log(4 & 1);
Output: 0

Operador Bitwise OR (|)
Este operador realiza una operación OR booleana en cada bit de sus argumentos enteros.

console.log(4 | 1);
Output:5

Operador Bitwise XOR (^)
Este operador realiza una operación OR exclusiva booleana en cada bit de sus argumentos enteros.

console.log(4 ^ 1);
Output:5

Operador Bitwise NOT (~)
Este operador invierte todos los bits del operando.

console.log(~4);
Output:-5

Nota: JavaScript convierte números a enteros de 32 bits con signo antes de realizar una operación bit a bit. Y cuando se realiza la operación, vuelve a convertir el resultado en números de JavaScript de 64 bits.

Operador de desplazamiento a la izquierda (<<)
Este operador desplaza todos los bits de su primer operando hacia la izquierda el número de lugares especificado en el segundo operando.

console.log(4<<1);
Output: 8

Operador de desplazamiento a la derecha (>>)
Este operador desplaza todos los bits de su primer operando a la derecha el número de lugares especificado en el segundo operando.

*console.**log**(4>>1);*
Output: 2

7. Operadores Especiales

Operador Ternario

El operador ternario es la abreviatura de la instrucción if-else. Asigna valor a una variable en base a una condición, la sintaxis de la misma es:

condition ? value1 : value2

Si la condición es verdadera, el operador devuelve el valor de value1. De lo contrario, devuelve el valor de value2.

```
let result = (200>100) ? "Yes" : "No";
console.log(result);
Output:Yes
```

Operador Typeof
Se utiliza para encontrar el tipo de datos de un valor o variable.

```
console.log(typeof(100));
console.log(typeof("100"));

Output:
number
string
```

8. Tipo de datos de objeto en JavaScript

En JavaScript, los objetos son colecciones de pares clave-valor, donde la clave es una cadena y el valor puede ser cualquier tipo de datos.
Puede crear un objeto vacío en JavaScript usando la sintaxis de 'constructor de objetos' (nuevo Objeto()) o la sintaxis de 'objeto literal' (llaves {...}).

```
let obj1 = new Object();
let obj2 = {};
```

Cada objeto contiene una lista opcional de propiedades, donde una propiedad es un par clave:valor. Puede acceder a los valores del objeto mediante la notación de puntos o la notación tipo matriz (corchetes).

```
let obj = {
"key1" : "value1",
"key2" : "value2"
}
console.log(obj.key1);
console.log(obj["key2"]);

Output:
value1
value2
```