Operators & Expressions in JS

This lesson lists the commonly used JavaScript's expressions and operators, including assignment, comparison, arithmetic, bitwise, logical and string.

Operators

JavaScript operators can be categorized into two main categories i.e., *Unary* and *Binary* operators. Unary takes only one operand whereas binary takes two.

1. Binary Operators

Binary operators can further be divided into following types:

- *Arithmetic Operators* $(+, -, *, /)$

```
//Arithmetic Operators
console.log("****Arithmetic Operators****\n")
console.log("2 + 3 = " + (2 + 3))
console.log("2 - 3 = " + (2 - 3))
console.log("2 + 3 = " + (2 * 3))
console.log("6 + 3 = " + (6 / 3))
console.log("7 + 3 = " + (7 / 3))
```

- *Assignment Operators* (=, +=, -=, *=)

```
//Assignment Operators
console.log("\n****Assignment Operators****\n")
var x = 3;
console.log("x = " + x)
console.log("x += 1 gives x = " + (x+=1)) // adds 1
console.log("x -= 1 gives x = " + (x-=1)) // subtracts 1
console.log("x *= 3 gives x = " + (x*=3)) // multiplies 3 with x
```

- *Logical Operators* ($&&, ||, ! $)

```
//Logical Operators
console.log("\n****Logical Operators****\n")
console.log("1 OR 1 = " + (1 || 1)) // 1 OR 1
console.log("1 OR 0 = " + (1 || 0)) // 1 OR 0
console.log("0 OR 0 = " + (0 || 0)) // 0 OR 0
console.log("1 AND 1 = " + (1 && 1)) // 1 AND 1
console.log("1 AND 0 = " + (1 && 0)) // 1 AND 0
console.log("0 AND 0 = " + (0 && 0)) // 0 AND 0
console.log(!true)  // NOT TRUE
console.log(!1)     // NOT TRUE
console.log(!false) // NOT FALSE
console.log(!0)     // NOT FALSE
```

- *Comma Operator* (,): The Comma operator evaluates each operand from left to right and returns the value of right most operand.

```
//Comma operator
console.log("\n****Comma Operator****")
var a = 4;
a = (a++, a);
console.log("The value for expression with comma operator is: " + a) //returns 5
```

- *Comparison Operators* $(<, >, ==, !=)$

```
//Comparison operators
console.log("\n****Comparison Operators****")
console.log(1 > 2) //false
console.log(1 < 2) //true
console.log(1 == 1) //true
console.log(1 != 1) //false
```

- *Bitwise Operators* (&, |, ^)

```
//Bitwise Operator
```

```
console.log("\n****Bitwise Operators****")
console.log("Bitwise AND of 5 and 1: " + (5 & 1)) //returns 1


console.log("Bitwise OR of 5 and 1: " + (5 | 1)) // returns 5
console.log("Bitwise XOR of 5 and 1: " + (5 ^ 1)) //returns 4
```

- *String Operators* (+)

```
//String Operator
console.log("\n****String Operator****")
console.log("Concatenation" + " (+)" + " operator in action")
```

- *Conditional Operators* (? :)

```
//Conditional Operator
console.log("\n ****Conditional Operator****")
var num_of_months = 13
var ans = (num_of_months > 12) ? "Invalid" : "Valid"
console.log(ans) //Returns Invalid
```

Did you know?

A comma operator (`,`) is used when you want to evaluate an expression from left to right.

2. Unary Operators

As mentioned earlier, unary operators take only one operand in order to perform a specific operation. Some of the commonly used unary operators in JavaScript are:

- `typeof`: Returns the type of the given operand

- `delete`: Deletes an object, object's attribute or an instance in an array
- `void`: Specifies that an expression does not return anything
- *Increment Operators* : ++, --

> **Did you know?**
>
> The operator `===` is commonly referred as Deep Equals in JavaScript. The only difference between double equals `==` and deep equals is that the former does not perform type comparison but in fact, converts the type of one of the operands to make their types same. Deep equals, on the other hand, returns false if both types are not the same.

See the example given below for better understanding:

```
console.log(1 == 1); //returns true
console.log('1' == 1); //returns true
console.log(1 === 1); //returns true
console.log('1' === 1); //returns false
```

Expressions

Anything that evaluates to a value is called an *expression*. Some of the basic expressions and keywords used in JavaScript are mentioned below:

- `this`: points to the current object
- `super`: calls methods on an object's parent, for example, call parent's constructor
- `function`: used to define a function
- `function*`: used to define a generator function
- `async function`: used to define an async function

There is one special operator which we have left for now... let's discuss it in the next lesson!