Operators

Arithmetic Operators

Arithmetic operators take numerical values as operands and evaluate them to a single numerical value. Some of the arithmetic operators are -

- **Addition(+)** - It adds the numerical operands and is also used for string concatenation. It will add numerically and string operand to a number if it can be converted to a number. Else it concatenates them.

- **Subtraction(-)** - It subtracts the two numeric operands. If any one of them is not a number or cannot be converted to a number, then **'NaN'** is printed.

  NaN refers to Not a Number

- **Division(/)** - It divides the first operand with the second operand. If the second operator is **'+0'** or **'-0**', then **'Infinity'** and **'-Infinity'** are printed, respectively. If they are not divisible, **'NaN'** is printed.

- **Multiplication(*)** - It multiplies the two numeric operands. Any number multiplied with Infinity prints **'Infinity'**. If the other number is zero, then **'NaN**’ is printed.

- **Remainder(%)** - It finds the remainder left after division. If one of the operands is **'Infinity'** or **'NaN',** then **'NaN'** is printed.

```
Examples :  var x = "Coding"  ;
            var y = " Ninjas"  ;
            var z = x + y ;
            console.log(z) ;     // Output Coding Ninjas

            z = x - y ;
            console.log(z) ;  // Output NaN

            z = x / y ;
            console.log(z) ;  // Output NaN

            x= 5 ; y = 0 ;
            z=x/y;
            console.log(z) ;  // Output Infinity

            x  = Infinity , y = 0 ;
            z= x*y;
            console.log(z) ;  // Output NaN
```

Assignment Operators

Assignment operators are used to assign the value of the right operand/expression to the left operand. The simplest assignment operator is equal (=), which sets the right operand value to the left operand.

```
Example :  var x=10;
           var y=x;
```

This will assign a value of 10 to y.

Other assignment operators are shorthand operations of other operators. They are called **compound assignment operators**. Some of them with their meaning (i.e. the extended version of these operations) are provided below.

Name	Shorthand operator	Meaning
Addition Assignment	x +=y	x = x + y
Division Assignment	x /=y	x = x / y
Exponentiation Assignment	x **=y	x = x ** y
Right Shift Assignment	x >>=y	x = x >> y
Bitwise XOR Assignment	x ^=y	x = x ^ y

Increment and Decrement Operator

The increment operator increments the value of the numeric operand by one.
The decrement operator decreases the value of the numeric operand by one.

There are two ways to use increment and decrement -

- **Using postfix (x++ or x--)** - the value is returned first, and then the value is incremented or decremented.

- **Using prefix (++x or --x)** - the value is first incremented or decremented and then returned.

```
Eg :  var x=10;
       console.log(++x); // Output 11
But ,
      x = 10 ;
      console.log(x++);  // Output 10

This is  because in post-increment/decrement value is changed after
the work is completed

     x = 1 ; y = 2;
    console.log( x++ + ++y) ; // Output 4 , not 5
```

Comparison Operators

The comparison operators are used to compare two values with each other.

The **equality operator (==)** is used to compare the two values, if they are equal or not. But the values are not directly compared. First, they are converted to the same data type and then the converted content is compared.

```
Example:  "1" == 1 evaluates to true, even though the first one is a String and the
other is a Number.
```

There is another **comparison operator (===)** known as the strict equality operator. It checks both the data type and the content. If the data type is not equal, it returns false.

```
So "1" === 1 now evaluates to false.
```

Other comparison operators are -

- **Inequality (!=)** - It returns the opposite result of the equality operator.
- **Strict Inequality (!==)** - It returns the opposite result of strict equality.
- **Greater Than (>)** - It returns true if the left operand is greater than the right one.
- **Greater Than or Equal (>=)** - It returns true if the left operand is greater than or equal to the right one.
- **Less Than (<)** - It returns true if the left operand is less than the right one.
- **Less Than or Equal (<=)** - It returns true if the left operand is less than or equal to the right one.

Logical Operators

The logical operators use boolean values as their operands. These operands are mostly expressions that evaluate to **'true'** or **'false'**.

● **Logical AND (&&)** - returns 'true' if both operands/expression are true, else 'false'. If the first expression is false, the second expression is not evaluated, and 'false' is returned.

```
Example :  false && true returns false
```

● **Logical OR (||)** - returns 'true' if any of the operand/expression is true, else 'false'.

```
Example : false || true returns true
```

● **Logical NOT (!)** - returns the opposite boolean value to which the expression is evaluated to.

```
Example :  !false returns true
           !true return false
```

Bitwise Operators

The bitwise operators treats operands as a sequence of 32 bits binary representation(0 and 1).

- **Bitwise AND (&) -** returns 1 for each bit position where both bits are 1s.
  Example : (5 & 13 = 5) is evaluated as (0101 & 1101 = 0101).

- **Bitwise OR (|) -** returns 1 for each bit position where either bit is 1.
  Example : (5 | 13 = 13) is evaluated as (0101 | 1101 = 1101).

- **Bitwise XOR (^) -** returns 1 for each bit position where either bit is 1 but not both.
  Example : (5 ^ 13 = 8) is evaluated as (0101 ^ 1101 = 1000).

- **Bitwise NOT (~)** - returns the inverted bits of operand. This means that 0 becomes 1 and vice versa.

- **Left Shift (<<)** - shifts bits to the left and insert 0s from right.

  Example : 3 in binary is 011
  3<<1 means shift 1 bits to the leftand insert zeroes from right
  011 - > 110 i.e 6

- **Sign-propagating Right Shift (>>)** - shifts bits to the right and insert either 0s or 1s from the left according to the sign of the number ('0' for positive and '1' for negative).

  Example : 3 in binary is 011
  3>>1 means shift 1 bits to the right and insert zeroes from left
  011 - > 001 i.e 1

- **Zero-fill Right Shift (>>>)** - shifts bits to the right and insert 0s from left.

  Example : 5 in binary is 101
  5>>>2 means shift 2 bits to the right and insert zeroes from left
  101 - > 001 i.e 1

NOTE: You need to be careful while working with operators because they work in precedence. So you might use round brackets to group them according to your priority of execution. Also, the round brackets (called as grouping operator) has the highest precedence.