Operators

Arithmetic Operators

JavaScript **Arithmetic Operators** are the operators that operate upon the numerical values and return a numerical value. Any kind of arithmetic operations performance required these operators.

JavaScript **Assignment Operators list:** There are so many arithmetic operators as shown in the table with the description.

OPERATOR NAME	USAGE	OPERATION
Addition Operator	a+b	Add two numbers or concatenate the string
Subtraction Operator	a-b	Difference between the two operators
Multiplication Operator	a*b	Multiply two number

Division Operator	a/b	Find the quotient of two operands
Modulus Operator	a%b	Find the remainder of two operands
Exponentiation Operator	a**b	Raise the Left operator to the power of the right operator
Increment Operator	a++ ++a	Return the operand and then increase by one Increase operand by one and then return
Decrement Operator	a– –a	Return operand and then decrease by one Decrease operand by one and then return
Unary Plus(+)	+a	Converts NaN to number
Unary Negation (-)	-a	Converts operand to negative

Addition (+)

The addition operator takes two numerical operands and gives their numerical sum. It also concatenates two strings or numbers.

Example:

```
// Number + Number => Addition
let x = 1 + 2
console.log(x)
// Number + String => Concatenation
let y =  5 + "hello"
console.log(y)
```

Output: 3

5hello

Subtraction (-)

The subtraction operator gives the difference between two operands in the form of numerical value.

Example:

```
// Number - Number => Subtraction
let x = 10 - 7
```

```
console.log(x)

let y = "Hello" - 1

console.log(y)
```

Output:

```
3

NaN
```

Multiplication (*)

The multiplication operator gives the product of operands where one operand is a multiplicand and another is multiplier.

Example:

```
// Number * Number => Multiplication

let x = 3 * 3

let y = -4 * 4

console.log(x)

console.log(y)
```

```
let a = Infinity * 0

let b = Infinity * Infinity

console.log(a)

console.log(b)

let z = 'hi' * 2

console.log(z)
```

Output:

```
9

-16

NaN

Infinity

NaN
```

Division (/)

The division operator provides the quotient of its operands where the right operand is the divisor and the left operand is the dividend.

Example:

```
// Number / Number => Division
```

```
let x = 5 / 2

let y = 1.0 / 2.0

console.log(x)

console.log(y)



let a = 3.0 / 0

let b = 4.0 / 0.0

console.log(a)

console.log(b)

let z = 2.0 / -0.0

console.log(z)
```

Output:

```
2.5

0.5

Infinity

Infinity

-Infinity
```

Modulus (%)

The modulus operator returns the remainder left over when a dividend is divided by a divisor. The modulus operator is also known as the **remainder operator.** It takes the sign of the dividend.

Example:

```
// Number % Number => Modulus of the number
let x = 9 % 5
let y = -12 % 5
let z = 1 % -2
let a = 5.5 % 2
let b = -4 % 2
let c = NaN % 2
console.log(x)
console.log(y)
console.log(z)
console.log(a)
console.log(b)
console.log(c)
```

Output:

```
4
-2
1
1.5
0
NaN
```

Exponentiation (**)

The exponentiation operator gives the result of raising the first operand to the power of the second operand. The exponentiation operator is right-associative.

In JavaScript, it is not possible to write an ambiguous exponentiation expression i.e. you cannot put an unary operator (+ / – / ~ / ! / delete / void) immediately before the base number.

Example:

```
// Number ** Number => Exponential of the number

// let x = -4 ** 2 // This is an incorrect expression

let y = -(4 ** 2)
let z = 2 ** 5

let a = 3 ** 3

let b = 3 ** 2.5

let c = 10 ** -2

let d = 2 ** 3 ** 2

let e = NaN ** 2


console.log(y)

console.log(z)

console.log(a)

console.log(b)

console.log(c)

console.log(d)

console.log(e)
```

Output:

```
-16

32

27

15.588457268119896
```

0.01

512

NaN

Increment (++)

The increment operator increments (adds one to) its operand and returns a value.

If used postfix with the operator after the operand (for example, x++), then it increments and returns the value before incrementing.

If used prefix with the operator before the operand (for example, ++x), then it increments and returns the value after incrementing.

Example:

```
// Postfix
let a = 2;
b = a++; // b = 2, a = 3


// Prefix
```

```
let x = 5;

y = ++x; // x = 6, y = 6


console.log(a)

console.log(b)

console.log(x)

console.log(y)
```

Output:

```
3

2

6

6
```

Decrement (- -)

The decrement operator decrements (subtracts one from) its operand and returns a value.

If used postfix, with operator after operand (for example, x–), then it decrements and returns the value before decrementing.

If used prefix, with the operator before the operand (for example, –x), then it decrements and returns the value after decrementing.

Example:

```
// Prefix
let a = 2;
b = --a;
// Postfix
let x = 3;
y = x--;
console.log(a)
console.log(b)
console.log(x)
console.log(y)
```

Output:

```
1
1
```

2

3

Unary Negation(-)

This is a unary operator i.e. it operates on a single operand. It gives the negation of an operand.

Example:

```
let a = 3;

b = -a;


// Unary negation operator

// can convert non-numbers

// into a number

let x = "3";

y = -x;


console.log(a)

console.log(b)
console.log(x)

console.log(y)
```

Output:

```
3

-3

3

-3
```

Unary Plus(+)

This is a way to convert a non-number into a number. Although unary negation (-) also can convert non-numbers, unary plus is the fastest and preferred way of converting something into a number, because it does not perform any other operations on the number.

Example:

```
let a =  +4

let b = +'2'

let c = +true

let x = +false
let y = +null


console.log(a)

console.log(b)

console.log(c)

console.log(x)

console.log(y)
```

Output:

```
4

2

1

0

0
```

Comparison Operators

JavaScript **Comparison operators** are mainly used to perform the logical operations that determine the equality or difference between the values. **Comparison operators** are used in logical expressions to determine their equality or differences in variables or values.

JavaScript **Comparison Operators list:** There are so many comparison operators as shown in the table with the description.

OPERATOR NAME	USAGE	OPERATION
Equality Operator	a==b	Compares the equality of two operators
Inequality Operator	a!=b	Compares inequality of two operators
Strict Equality Operator	a===b	Compares both value and type of the operand

OPERATOR NAME	USAGE	OPERATION
Strict Inequality Operator	a!==b	Compares inequality with type
Greater than Operator	a>b	Checks if the left operator is greater than the right operator
Greater than or equal Operator	a>=b	Checks if the left operator is greater than or equal to the right operator
Less than Operator	a<b	Checks if the left operator is smaller than the right operator
Less than or equal Operator	a<=b	Checks if the left operator is smaller than or equal to the right operator

Equality (==)

This operator is used to compare the equality of two operands. If equal then the condition is true otherwise false.

Example: Below example illustrates the **(==)** operator

```
// Illustration of (==) operator
let val1 = 5;
let val2 = '5';

// Checking of operands
console.log(val1 == 5);
console.log(val2 == 5);
console.log(val1 == val1);

// Check against null and boolean value
console.log(0 == false);
console.log(0 == null);
```

Output:

```
true
true
```

```
true

true

false
```

Inequality(!=)

This operator is used to compare the inequality of two operands. If equal then the condition is false otherwise true.

Example: Below examples illustrate the **(!=)** operator in JavaScript.

```
// Illustration of (!=) operator
let val1 = 5;
let val2 = '5';

// Checking of operands
console.log(val1 != 6);
console.log(val2 != '5');
console.log(val1 != val2);

// Check against null and boolean value
console.log(0 != false);
console.log(0 != null);
```

Output:

```
true

false

false

false

true
```

Strict equality(===)

This operator is used to compare the equality of two operands with type. If both value and type are equal then the condition is true otherwise false.

Example:

```
// Illustration of (===) operator

let val1 = 5;

let val2 = '5';
```

```
// Checking of operands
console.log(val1 === 6);
console.log(val2 === '5');
console.log(val1 === val2);

// Check against null and boolean value
console.log(0 === false);
console.log(0 === null);
```

Output:

```
false
true
false
false
false
```

Strict inequality (!==)

This operator is used to compare the inequality of two operands with type. If both value and type are not equal then the condition is true otherwise false.

Example:

```
// Illustration of (!==) operator

let val1 = 5;

let val2 = '5';


// Checking of operands

console.log(val1 !== 6);

console.log(val2 !== '5');

console.log(val1 !== val2);


// Check against null and boolean value

console.log(0 !== false);

console.log(0 !== null);
```

Output:

```
true

false

true

true
```

```
true
```

Greater than (>)

This operator is used to check whether the left-side value is greater than the right-side value. If the value is greater then the condition is true otherwise false.

Example:

```
// Illustration of (>) operator
let val1 = 5;
let val2 = "5";

// Checking of operands
console.log(val1 > 0);
console.log(val2 > "10");
console.log(val1 > "10");
console.log(val2 > 0);
```

Output:

```
true
true
```

```
false

true
```

Greater than or equal (>=)

This operator is used to check whether the left side operand is greater than or equal to the right side operand. If the value is greater than or equal then the condition is true otherwise false.

Example:

```
// Illustration of (>=) operator
let val1 = 5;
let val2 = "5";

// Checking of operands
console.log(val1 >= 5);
console.log(val2 >= "15");
console.log(val1 >= "5");
console.log(val2 >= 15);
```

Output:

```
true

true

true

false
```

Less than operator(<)

This operator is used to check whether the left-side value is less than the right-side value. If yes then the condition is true otherwise false.

Example

```
// Illustration of (<) operator
let val1 = 5;
let val2 = "5";

// Checking of operands
console.log(val1 < 15);
console.log(val2 < "0");
console.log(val1 < "0");
console.log(val2 < 15);
```

Output:

```
true

false

false

true
```

Less than or equal operator(<=)

This operator is used to check whether the left side operand value is less than or equal to the right side operand value. If yes then the condition is true otherwise false.

Example:

```
// Illustration of (<=) operator

let val1 = 5;

let val2 = "5";

// Checking of operands

console.log(val1 <= 15);

console.log(val2 <= "0");

console.log(val1 <= "0");
console.log(val2 <= 15);
```

Output:

```
true

false

false

true
```

Logical Operators

JavaScript Logical operator allows us to compare variables or values. The logical operator is mostly used to make decisions based on conditions specified for the statements. It can also be used to manipulate a boolean or set termination conditions for loops.

In JavaScript, there are basically three types of logical operators.

OPERATOR NAME	**OPERATOR SYMBOL**	**OPERATION**
NOT	!	Converts operator to boolean and returns flipped value
AND	&&	Evaluates operands and return true only if all are true
OR	||	Returns true even if one of the multiple operands is true

!(NOT) Operator

It reverses the boolean result of the operand (or condition). It first converts the operand to a boolean type and then returns its flipped value.

Example:

```
// !(NOT) operator
let i = 0;
console.log((!i));
console.log(!!i);
```

Output:

```
true
false
```

Explanation

Since zero is treated as a falsy value therefore NOT operation on zero will return true and when this operation is performed again we get true as output.

&&(AND)

The && operator accepts multiple arguments and evaluates the operator from left to right. It returns true only if all the operands that are evaluated are true

Example:

```
// &&(AND) operator
let i = 0, j=2, k=3, l=8;
```

```
console.log(Boolean(i&&j&&k));

console.log(Boolean(j&&k&&l));
```

Output:

```
false

true
```

Explanation

In JavaScript, the value of 0 when converted to zero is considered false so when performing “and” operation on 0 a falsy value is returned and we get false output otherwise true.

||(OR)

The ‘OR’ operator is somewhat opposite of the ‘AND’ operator. It also evaluates the operator from left to right and returns true even if one operand is evaluated as true

Example:

```
// ||(OR) Operator

let i = 1;

let j = null;
let k = undefined;

let l = 0;

console.log(Boolean(j||k));

console.log(Boolean(i||l));
```

Output:

```
false

true
```

Explanation

null, undefined, and 0 are recognized as falsy values. So OR operation on null and undefined will give false whereas numbers except 0 are treated as true.

Bitwise Operators

JavaScript uses 32 bits Bitwise operands. A number is stored as a 64-bit floating-point number but the bit-wise operation is performed on a 32-bit binary number i.e. to perform a bit-operation JavaScript converts the number into a 32-bit binary number (signed) and performs the operation and converts back the result to a 64-bit number.

Below is a list of bitwise operators:

OPERATOR NAME	**USAGE**	**DESCRIPTION**
Bitwise AND(&)	a&b	Returns true if both operands are true
Bitwise OR(|)	a|b	Returns true even if one operand is true
Biwise XOR(^)	a^b	Returns true if both operands are different
Bitwise NOT(~)	a~b	Flips the value of the operand
Bitwise Left Shift(<<)	a<<b	Shifts the bit toward the left

OPERATOR NAME	USAGE	DESCRIPTION
Bitwise Right Shift(>>)	a>>b	Shifts the bit towards the right
Zero Fill Right Shift(>>>)	a>>>b	Shifts the bit towards the right but adds 0 from left

Bitwise AND (&)

It is a binary operator i.e. accepts two operands. Bit-wise AND (&) returns 1 if both the bits are set (i.e 1) and 0 in any other case.

A	B	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR (|)

It is a binary operator i.e. accepts two operands. Bit-wise OR (|) returns 1 if any of the operands is set (i.e. 1) and 0 in any other case.

A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise XOR (^)

It is a binary operator i.e. accepts two operands. Bit-wise XOR (^) returns 1 if both the operands are different and 0 in any other case.

A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise NOT (~)

It is a unary operator i.e. accepts single operands. Bit-wise NOT (~) flips the bits i.e 0 becomes 1 and 1 becomes 0.

A	OUTPUT
0	1
1	0

Below is an example of the JavaScript Bitwise Operators:

Example:

```
let a = 4;

let b = 1;


console.log("A & B = " + (a & b));

console.log("A | B = " + (a | b));

console.log("~A = " + (~a));
```

Output:

```
A & B = 0

A | B = 5

~A = -5
```

Below are a few bit-wise shift operators used in JavaScript:

Left Shift (<<)

It’s a binary operator i.e. it accepts two operands. The first operator specifies the number and the second operator specifies the number of bits to shift. Each bit is shifted towards the left and 0 bits are added from the right. The excess bits from the left are discarded.

A	**6 (00000000000000000000000000000110**
B	**1 (00000000000000000000000000000001)**
OUTPUT	**12 (00000000000000000000000000001100)**

`

Sign Propagating Right Shift (>>)

It’s a binary operator i.e. it accepts two operands. The first operand specifies the number and the second operand specifies the number of bits to shift. Each bit is shifted towards the right, the overflowing bits are discarded. This is Sign Propagating as the bits are added from the left depending upon the sign of the number (i.e. 0 if positive and 1 if negative)

A	**6 (00000000000000000000000000000110)**
B	**1 (00000000000000000000000000000001)**
OUTPUT	**3 (00000000000000000000000000000011)**

Zero Fill Right Shift (>>>):

It’s a binary operator i.e. it accepts two operands. The first operand specifies the number and the second operand specifies the number of bits to shift. Each bit is shifted towards the right, the overflowing bits are discarded. 0 bit is added from the left so its zero fill right shift.

A	6 (00000000000000000000000000000110)
B	1 (00000000000000000000000000000001)
OUTPUT	3 (00000000000000000000000000000011)

Example:

```
let a = 6;
let b = 1;

// AND Operation
console.log("A & B = " + (a & b));

// OR operation
console.log("A | B = " + (a | b));

// NOT operation
console.log("~A = " + (~a));

// Sign Propagating Right Shift
```

```
console.log("A >> B = " + (a >> b));


// Zero Fill Right Shift
console.log("A >>> B = " + (a >>> b));


// Left Shift
console.log("A << B = " + (a << b));
```

Output:

```
A & B = 0
A | B = 7
~A = -7
A >> B = 3
A >>> B = 3
A << B = 12
```

Assignment Operators

JavaScript **assignment operator** is **equal (=)** which assigns the value of the right-hand operand to its left-hand operand. That is if a = b assigns the value of b to a.

The simple assignment operator is used to assign a value to a variable. The assignment operation evaluates the assigned value. Chaining the assignment operator is possible in order to assign a single value to multiple variables. See the example.

Assignment Operators List:

OPERATOR NAME	SHORTHAND OPERATOR	MEANING
Addition Assignment	a+=b	a=a+b
Subtraction Assignment	a-=b	a=a-b
Multiplication Assignment	a*=b	a=a*b
Division Assignment	a/=b	a=a/b
Remainder Assignment	a%=b	a=a%b
Exponentiation Assignment	a**=b	a=a**b
Left Shift Assignment	a<<=b	a=a<<b
Right Shift Assignment	a>>=b	a=a>>b
Bitwise AND Assignment	a&=b	a=a&b

OPERATOR NAME	SHORTHAND OPERATOR	MEANING
Bitwise OR Assignment	a|=b	a=a | b
Bitwise XOR Assignment	a^=b	a=a^b

Addition Assignment

This operator adds the value to the right operand to a variable and assigns the result to the variable. The types of the two operands determine the behavior of the addition assignment operator. Addition or concatenation is possible. In case of concatenation then we use the string as an operand.

Example:

```
let a = 2;
```

```
const b = 3;


// Expected output: 2
console.log(a);


// Expected output: 4
console.log(a = b + 1);
```

Output:

```
2
4
```

Subtraction Assignment

This operator subtracts the value of the right operand from a variable and assigns the result to the variable.

Example:

```
let yoo = 4;


// Expected output 3
```

```
console.log(foo = yoo - 1);
```

Output:

3

Multiplication Assignment

This operator multiplies a variable by the value of the right operand and assigns the result to the variable.

Example:

```
let yoo = 4;


// Expected output 3
console.log(foo = yoo - 1);
```

Output:

10

Division Assignment

This operator divides a variable by the value of the right operand and assigns the result to the variable.

Example:

```
let yoo = 10;

const moo = 2;


// Expected output 5

console.log(yoo = yoo / moo);


// Expected output Infinity

console.log(yoo /= 0);
```

Output:

```
5

Infinity
```

Remainder Assignment

This operator divides a variable by the value of the right operand and assigns the remainder to the variable.

Example:

```
let yoo = 50;


// Expected output 0
console.log(yoo %= 10);
```

Output:

0

Exponentiation Assignment

This operator raises the value of a variable to the power of the right operand.

Example:

```
let yoo = 50;


// Expected output 0
console.log(yoo %= 10);
```

Output:

4

Left Shift Assignment

This operator moves the specified amount of bits to the left and assigns the result to the variable.

Example:

```
let yoo = 5;


// Expected output 20(In Binary 10100)
console.log(yoo <<= 2);
```

Output:

20

Right Shift Assignment:

This operator moves the specified amount of bits to the right and assigns the result to the variable.

Example:

```
let yoo = 5;


// Expected Output 1(In binary 001)
```

```
console.log(yoo >>= 2);
```

Output:

1

Bitwise AND Assignment:

This operator uses the binary representation of both operands, does a bitwise AND operation on them, and assigns the result to the variable.

Example:

```
let yoo = 5;


// Expected output 0(In binary 000)
console.log(yoo &= 2);
```

Output:

0

Bitwise OR Assignment:

This operator uses the binary representation of both operands, does a bitwise OR operation on them, and assigns the result to the variable.

Example:

```
let yoo=5;


// Expected output 7(In binary 111)
console.log(yoo|=2);
```

Output:

7

Bitwise XOR Assignment:

This operator uses the binary representation of both operands, does a bitwise XOR operation on them, and assigns the result to the variable.

Example:

```
let yoo = 5;


// Expected output 7(In binary 111)
```

```
console.log(yoo ^= 2);
```

Output:

7

Ternary Operator

The “Question mark” or “conditional” operator in JavaScript is a ternary operator that has three operands. It is the simplified operator of if/else.

Examples:

Input: let result = (10 > 0) ? true : false;

Output: true

Input: let message = (20 > 15) ? "Yes" : "No";

Output: Yes

Syntax:

condition ? value if true : value if false

condition: Expression to be evaluated which returns a boolean value.

value if true: Value to be executed if the condition results in a true state.

value if false: Value to be executed if the condition results in a false state.

Characteristics of Ternary Operator:

The expression consists of three operands: the condition, value if true, and value if false.

The evaluation of the **condition** should result in either true/false or a boolean value.

The **true** value lies between "**?**" & "**:**" and is executed if the condition returns true. Similarly, the **false** value lies after **":"** and is executed if the condition returns false.

Example 1:

```
function gfg() {

// JavaScript to illustrate

// Conditional operator

let PMarks = 40

let result = (PMarks > 39) ?
"Pass" : "Fail";


console.log(result);
}
gfg();
```

Output:

```
Pass
```

Example 2:

```
function gfg() {
// JavaScript to illustrate
// Conditional operator


let age = 60
let result = (age > 59) ?
"Senior Citizen" : "Not a Senior Citizen";


console.log(result);
}
```

```
gfg();
```

Output:

Senior Citizen

Example 3:

```
function gfg() {
// JavaScript to illustrate
// multiple Conditional operators


let marks = 95;
let result = (marks < 40) ? "Unsatisfactory" :
(marks < 60) ? "Average" :
(marks < 80) ? "Good" : "Excellent";


console.log(result);
}
gfg();
```

Output:

Excellent

Typeof Operator

In JavaScript, the **typeof operator** returns the data type of its operand in the form of a string. The operand can be any object, function, or variable.

Syntax:

typeof operand

OR

typeof (operand)

Note:

Operand is an expression representing the object or primitive whose type is to be returned. The possible types that exist in javascript are:

- **undefined**
- **Object**
- **boolean**
- **number**
- **string**
- **symbol**
- **function**

Example:

```
// "string"
console.log(typeof 'mukul')


// "number"
console.log(typeof 25)


// "undefined"
console.log(typeof variable)
```

Output:

```
string
number
undefined
```

Example:

Typeof Number, in this sample, we used '===' (strict equality comparison operator) which compare value and type both and then return true or false. For example- consider the first console.log(), the js starts compiling from left to right and it first calculates the type of 25 which is 'number', and then compares it with ‘number’ and then finally returns true or false accordingly.

```
//Number
console.log(typeof 25 === 'number');
console.log(typeof 3.14 === 'number');
console.log(typeof (69) === 'number');

// log base 10
console.log(typeof Math.LN10 === 'number');
console.log(typeof Infinity === 'number');

// Despite being "Not-A-Number"
console.log(typeof NaN === 'number');

// Wrapping in Number() function
console.log(typeof Number('100') === 'number');
```

Output:

```
true
true
true
```

```
true

true

true

true
```

Fun fact NaN which stands for not-a-number has a type of "number".

Example:

```
// string
console.log(typeof '' === 'string');
console.log(typeof 'bla' === 'string');


// ES6 template literal
console.log(typeof `template literal` === 'string');
console.log(typeof '1' === 'string');
console.log(typeof (typeof 1) === 'string');


// Wrapping inside String() function
console.log(typeof String(1) === 'string');
```

Output:

```
true

true

true

true

true

true
```

Example:

```
// Boolean

console.log(typeof true === 'boolean');

console.log(typeof false === 'boolean');


// Two calls of the ! (logical NOT) operator

// are equivalent to Boolean()

console.log(typeof !!(1) === 'boolean');
```

Output:

```
true

true

true
```

Example:

```
// Undefined

console.log(typeof undefined === 'undefined');

// Declared but undefined variable

console.log(typeof variable === 'undefined');
```

Output:

```
true

true
```

Example:

```
// Symbol

console.log(typeof Symbol() === 'symbol');

console.log(typeof Symbol('party') === 'symbol');

console.log(typeof Symbol.iterator === 'symbol');
```

Output:

```
true

true

true
```

Example:

```
// Object
console.log(typeof { b: 1 } === 'object');
console.log(typeof [1, 2, 9] === 'object');
console.log(typeof new Date() === 'object');
```

Output:

```
true
true
true
```

Example:

```
// function
console.log(typeof function () { } === 'function');


//classes too are objects
console.log(typeof class C { } === 'function');
console.log(typeof Math.sin === 'function');
```

Output:

```
true
true
true
```