# JavaScript Operators

Javascript operators are used to perform different types of mathematical and logical computations.

## Examples:

- The **Assignment Operator** = assigns values
- The **Addition Operator** + adds values
- The **Multiplication Operator** * multiplies values
- The **Comparison Operator** > compares values

## JavaScript Assignment

The **Assignment Operator** (`=`) assigns a value to a variable:

### Assignment Examples

```
let x = 10;
```

```
// Assign the value 5 to x
let x = 5;
// Assign the value 2 to y
let y = 2;
// Assign the value x + y to z:
let z = x + y;
```

## JavaScript Addition

The **Addition Operator** (`+`) adds numbers:

### Adding

```
let x = 5;
let y = 2;
let z = x + y;
```

## JavaScript Multiplication

The **Multiplication Operator** (`*`) multiplies numbers:

### Multiplying

```
let x = 5;
let y = 2;
let z = x * y;
```

## Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators

## JavaScript Arithmetic Operators

**Arithmetic Operators** are used to perform arithmetic on numbers:

### Arithmetic Operators Example

```
let a = 3;
let x = (100 + 50) * a;
```

| Operator | Description |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| ** | Exponentiation (ES2016) |
| / | Division |
| % | Modulus (Division Remainder) |
| ++ | Increment |
| -- | Decrement |

# JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

The **Addition Assignment Operator** (`+=`) adds a value to a variable.

## Assignment

```
let x = 10;
x += 5;
```

| Operator | Example | Same As |
|---|---|---|
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| **= | x **= y | x = x ** y |

# JavaScript Comparison Operators

| Operator | Description |
|---|---|
| == | equal to |
| === | equal value and equal type |
| != | not equal |
| !== | not equal value or not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |

| | |
|---|---|
| ? | ternary operator |

## JavaScript String Comparison

All the comparison operators above can also be used on strings:

### Example

```
let text1 = "A";
let text2 = "B";
let result = text1 < text2;
```

Note that strings are compared alphabetically:

### Example

```
let text1 = "20";
let text2 = "5";
let result = text1 < text2;
```

## JavaScript String Addition

The `+` can also be used to add (concatenate) strings:

### Example

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

The `+=` assignment operator can also be used to add (concatenate) strings:

### Example

```
let text1 = "What a very ";
text1 += "nice day";
```

The result of text1 will be:

```
What a very nice day
```

**Note:** When used on strings, the + operator is called the concatenation operator.

## Adding Strings and Numbers

Adding two numbers, will return the sum as a number like 5 + 5 = 10.

Adding a number and a string, will return the sum as a concatenated string like 5 + "5" = "55".

### Example

```
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
```

The result of *x*, *y*, and *z* will be:

```
10
55
Hello5
```

Note: If you add a number and a string, the result will be a string!

## JavaScript Logical Operators

| Operator | Description |
|---|---|
| && | logical and |
| \|\| | logical or |
| ! | logical not |

## JavaScript Type Operators

| Operator | Description |
|---|---|
| typeof | Returns the type of a variable |
| instanceof | Returns true if an object is an instance of an object type |

## JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

| Operator | Description | Example | Same as | Result | Decimal |
|---|---|---|---|---|---|
| & | AND | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| \| | OR | 5 \| 1 | 0101 \| 0001 | 0101 | 5 |
| ~ | NOT | ~ 5 | ~0101 | 1010 | 10 |
| ^ | XOR | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | left shift | 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | right shift | 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| >>> | unsigned right shift | 5 >>> 1 | 0101 >>> 1 | 0010 | 2 |

The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.
Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.
~00000000000000000000000000000101 will return
11111111111111111111111111111010

# JavaScript Data Types

## JavaScript has 8 Datatypes

String
Number
Bigint
Boolean
Undefined
Null
Symbol
Object

## The Object Datatype

The object data type can contain both **built-in objects**, and **user defined objects**:

Built-in object types can be:objects, arrays, dates, maps, sets, intarrays, floatarrays, promises, and more.

## Examples

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

Note: A JavaScript variable can hold any type of data.

# The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

```
let x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
let x = "16" + "Volvo";
```

Note: When adding a number and a string, JavaScript will treat the number as a string.

### Example

```
let x = 16 + "Volvo";
```

### Example

```
let x = "Volvo" + 16;
```

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

### JavaScript:

```
let x = 16 + 4 + "Volvo";
```

Result:

```
20Volvo
```

### JavaScript:

```
let x = "Volvo" + 16 + 4;
```

Result:

```
Volvo164
```

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".

In the second example, since the first operand is a string, all operands are treated as strings.

## JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

### Example

```
let x;         // Now x is undefined
x = 5;         // Now x is a Number
x = "John";    // Now x is a String
```

## JavaScript Strings

A string (or a text string) is a series of characters like "John Doe". Strings are written with quotes. You can use single or double quotes:

### Example

```
// Using double quotes:
let carName1 = "Volvo XC60";

// Using single quotes:
let carName2 = 'Volvo XC60';
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

### Example

```
// Single quote inside double quotes:
let answer1 = "It's alright";

// Single quotes inside double quotes:
let answer2 = "He is called 'Johnny'";

// Double quotes inside single quotes:
let answer3 = 'He is called "Johnny"';
```

## JavaScript Numbers

All JavaScript numbers are stored as decimal numbers (floating point).

Numbers can be written with, or without decimals:

### Example

```
// With decimals:
let x1 = 34.00;

// Without decimals:
let x2 = 34;
```

## Exponential Notation

Extra-large or extra small numbers can be written with scientific (exponential) notation:

### Example

```
let y = 123e5;     // 12300000
let z = 123e-5;    // 0.00123
```

## Note

Most programming languages have many number types:

Whole numbers (integers):
byte (8-bit), short (16-bit), int (32-bit), long (64-bit)

Real numbers (floating-point):
float (32-bit), double (64-bit).

**Javascript numbers are always one type:
double (64-bit floating point)**

## JavaScript BigInt

All JavaScript numbers are stored in a 64-bit floating-point format.

JavaScript BigInt is a new datatype (ES2020) that can be used to store integer values that are too big to be represented by a normal JavaScript Number.

### Example

```
let x = BigInt("123456789012345678901234567890");
```

## JavaScript Booleans

Booleans can only have two values: `true` or `false`.

### Example

```
let x = 5;
let y = 5;
let z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

Booleans are often used in conditional testing.

## JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called `cars`, containing three items (car names):

### Example

```
const cars = ["Saab", "Volvo", "BMW"];
```

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

## JavaScript Objects

JavaScript objects are written with curly braces `{}`. Object properties are written as name:value pairs, separated by commas.

### Example

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

## The typeof Operator

You can use the JavaScript `typeof` operator to find the type of a JavaScript variable.

The `typeof` operator returns the type of a variable or an expression:

### Example

```
typeof ""             // Returns "string"
typeof "John"         // Returns "string"
typeof "John Doe"     // Returns "string"
```

### Example

```
typeof 0              // Returns "number"
typeof 314            // Returns "number"
typeof 3.14           // Returns "number"
typeof (3)            // Returns "number"
typeof (3 + 4)        // Returns "number"
```

## Undefined

In JavaScript, a variable without a value, has the value `undefined`. The type is also `undefined`.

### Example

```
let car; // Value is undefined, type is undefined
```

Any variable can be emptied, by setting the value to `undefined`. The type will also be `undefined`.

### Example

```
car = undefined; // Value is undefined, type is undefined
```

## Empty Values

An empty value has nothing to do with `undefined`.

An empty string has both a legal value and a type.

### Example

```
let car = "";    // The value is "", the typeof is "string"
```