# JAVASCRIPT

[JavaScript](#), created by Brendan Eich in 1995, is one of the most widely used web development languages. It was designed to build dynamic web pages at first. A script is a JS program that may be added to the HTML of any web page. When the page loads, these scripts execute automatically. A language that was originally designed to build dynamic web pages may now be run on the server and on almost any device that has the JavaScript Engine installed.

After HTML and CSS, JavaScript is the third biggest web technology. JavaScript is a scripting language that may be used to construct online and mobile apps, web servers, games, and more. JavaScript is an object-oriented programming language that is used to generate websites and applications. It was created with the intention of being used in a browser. Even today, the server-side version of JavaScript known as Node.js may be used to create online and mobile apps, real-time applications, online streaming applications, and videogames. [Javascript frameworks](#), often known as inbuilt libraries, may be used to construct desktop and mobile programs. Developers may save a lot of time on monotonous programming jobs by using these code libraries, allowing them to focus on the production work of development.

## 1. What are the different data types present in javascript?

To know the type of a JavaScript variable, we can use the typeof operator.

1. Primitive types

String - It represents a series of characters and is written with quotes. A string can be represented using a single or a double quote.

Example :

```
var str = "Vivek Singh Bisht"; //using double quotes
var str2 = 'John Doe'; //using single quotes
```

- Number - It represents a number and can be written with or without decimals.

Example :

```
var x = 3; //without decimal
var y = 3.6; //with decimal
```

- BigInt - This data type is used to store numbers which are above the limitation of the Number data type. It can store large integers and is represented by adding "n" to an integer literal.

Example :

```
var bigInteger =  234567890123456789012345678901234567890;
```

- Boolean - It represents a logical entity and can have only two values : true or false. Booleans are generally used for conditional testing.

Example :

```
var a = 2;
var b =  3;
var c =  2;
(a == b) // returns false
(a == c) //returns true
```

- Undefined - When a variable is declared but not assigned, it has the value of undefined and it's type is also undefined.

Example :

```
var x; // value of x is undefined
var y = undefined; // we can also set the value of a variable as undefined
```

- Null - It represents a non-existent or a invalid value.

Example :

**var** z = null;

- Symbol - It is a new data type introduced in the ES6 version of javascript. It is used to store an anonymous and unique value.

Example :

**var** symbol1 = Symbol('symbol');

- typeof of primitive types :

**typeof** "John Doe" // Returns "string"

**typeof** 3.14 // Returns "number"

**typeof** true // Returns "boolean"

**typeof** 2345678901234567890123456789012345678901234567890n // Returns bigint

**typeof** undefined // Returns "undefined"

**typeof** null // Returns "object" (kind of a bug in JavaScript)

**typeof** Symbol('symbol') // Returns Symbol

2. Non-primitive types

- Primitive data types can store only a single value. To store multiple and complex values, non-primitive data types are used.

- Object - Used to store collection of data.
- Example:

```javascript
// Collection of data in key-value pairs

var obj1 = {

  x:  43,

  y:  "Hello world!",

  z: function(){

    return this.x;

  }

}

// Collection of data as an ordered list

var array1 = [5, "Hello", true, 4.1];
```

## 2. Explain Hoisting in javascript.

Hoisting is the default behaviour of javascript where all the variable and function declarations are moved on top.

```
Declaration        a = 1;
moves to top       alert(' a = ' + a);
                   var a;
```

This means that irrespective of where the variables and functions are declared, they are moved on top of the scope. The scope can be both local and global.

Example 1:

```
hoistedVariable = 3;

console.log(hoistedVariable); // outputs 3 even when the variable is declared after it is initialized

var hoistedVariable;
```

Example 2:

```
hoistedFunction();  // Outputs " Hello world! " even when the function is declared after calling

function hoistedFunction(){
  console.log(" Hello world! ");
}
```

Example 3:

```javascript
// Hoisting takes place in the local scope as well

function doSomething(){
  x = 33;
  console.log(x);
  var x;
}
doSomething(); // Outputs 33 since the local variable "x" is hoisted inside the local
scope
```

Note - Variable initializations are not hoisted, only variable declarations are hoisted:

```javascript
var x;

console.log(x); // Outputs "undefined" since the initialization of "x" is not hoisted

x = 23;
```

Note - To avoid hoisting, you can run javascript in strict mode by using "use strict" on top of the code:

```javascript
"use strict";

x = 23; // Gives an error since 'x' is not declared

var x;
```

### 3. Why do we use the word "debugger" in javascript?

The debugger for the browser must be activated in order to debug the code. Built-in debuggers may be switched on and off, requiring the user to report faults. The remaining section of the code should stop execution before moving on to the next line while debugging.

### 4. Difference between " == " and " === " operators.

Both are comparison operators. The difference between both the operators is that "==" is used to compare values whereas, " === " is used to compare both values and types.

Example:

**var** x = 2;

**var** y = "2";

(x == y)  // Returns true since the value of both x and y is the same

(x === y) // Returns false since the typeof x is "number" and typeof y is "string

## 5. Difference between var and let keyword in javascript.

Some differences are

1. From the very beginning, the 'var' keyword was used in JavaScript programming whereas the keyword 'let' was just added in 2015.
2. The keyword 'Var' has a function scope. Anywhere in the function, the variable specified using var is accessible but in 'let' the scope of a

variable declared with the 'let' keyword is limited to the block in which it is declared. Let's start with a Block Scope.

3. In ECMAScript 2015, let and const are hoisted but not initialized. Referencing the variable in the block before the variable declaration results in a ReferenceError because the variable is in a "temporal dead zone" from the start of the block until the declaration is processed.

## 6. Explain Implicit Type Coercion in javascript.

Implicit type coercion in javascript is the automatic conversion of value from one data type to another. It takes place when the operands of an expression are of different data types.

- String coercion

String coercion takes place while using the ' + ' operator. When a number is added to a string, the number type is always converted to the string type.

**var** x = 3;

**var** y = "3";

x + y // Returns "33"

**var** x = 24;

**var** y = "Hello";

x + y   // Returns "24Hello";

Note - ' + ' operator when used to add two numbers, outputs a number. The same ' + '
operator when used to add two strings, outputs the concatenated string:

**var** name = "Vivek";

**var** surname = " Bisht";

name + surname     // Returns "Vivek Bisht"

Let's understand both the examples where we have added a number to a string,

When JavaScript sees that the operands of the expression x + y are of different types (
one being a number type and the other being a string type ), it converts the number type
to the string type and then performs the operation. Since after conversion, both the
variables are of string type, the ' + ' operator outputs the concatenated string "33" in the
first example and "24Hello" in the second example.

Note - Type coercion also takes place when using the ' - ' operator, but the difference
while using ' - ' operator is that, a string is converted to a number and then subtraction
takes place.

**var** x = 3;

Var y = "3";

x - y    //Returns 0 since the variable y (string type) is converted to a number type

- Boolean Coercion

Boolean coercion takes place when using logical operators, ternary operators, if
statements, and loop checks. To understand boolean coercion in if statements and
operators, we need to understand truthy and falsy values.

Truthy values are those which will be converted (coerced) to true. Falsy values are those which will be converted to false.

All values except false, 0, 0n, -0, "", null, undefined, and NaN are truthy values.

If statements:

Example:

**var** x = 0;

**var** y = 23;

**if**(x) { console.log(x) }   // The code inside this block will not run since the value of x is 0(Falsy)

**if**(y) { console.log(y) }    // The code inside this block will run since the value of y is 23 (Truthy)

- Logical operators:

Logical operators in javascript, unlike operators in other programming languages, do not return true or false. They always return one of the operands.

OR ( || ) operator - If the first value is truthy, then the first value is returned. Otherwise, always the second value gets returned.

AND ( && ) operator - If both the values are truthy, always the second value is returned. If the first value is falsy then the first value is returned or if the second value is falsy then

the second value is returned.

Example:

```javascript
var x = 220;

var y = "Hello";

var z = undefined;



x || y    // Returns 220 since the first value is truthy



x || z   // Returns 220 since the first value is truthy



x && y    // Returns "Hello" since both the values are truthy



y && z   // Returns undefined since the second value is falsy



if( x && y ){

  console.log("Code runs" ); // This block runs because x && y returns "Hello" (Truthy)
```

```
}
```

```
if( x || z ){

    console.log("Code runs");  // This block runs because x || y returns 220(Truthy)

}
```

- ● Equality Coercion

Equality coercion takes place when using ' == ' operator. As we have stated before

The ' == ' operator compares values and not types.

While the above statement is a simple way to explain == operator, it's not completely true

The reality is that while using the '==' operator, coercion takes place.

The '==' operator, converts both the operands to the same type and then compares them.

Example:

```
var a = 12;

var b = "12";

a == b // Returns true because both 'a' and 'b' are converted to the same type and then compared.
Hence the operands are equal.
```

Coercion does not take place when using the '===' operator. Both operands are not converted to the same type in the case of '===' operator.

Example:

var a = 226;
var b = "226";
a === b // Returns false because coercion does not take place and the operands are of different types. Hence they are not equal.

## 7. Is javascript a statically typed or a dynamically typed language?

**JavaScript is a dynamically typed language. In a dynamically typed language, the type of a variable is checked during run-time in contrast to a statically typed language, where the type of a variable is checked during compile-time.**

| Static Typing | Dynamic Typing |
|---|---|
| string name;<br>name = "John";<br>~~name = 34;~~ | var name;<br>name = "John";<br>name = 34; |
| Variables have types | Variables have no types |
| Values have types | Values have types |
| Variables cannot change type | Variables change type dramatically |

Since javascript is a loosely(dynamically) typed language, variables in JS are not associated with any type. A variable can hold the value of any data type.

For example, a variable that is assigned a number type can be converted to a string type:

var a = 23;
var a = "Hello World!";

## 8. What is NaN property in JavaScript?

NaN property represents the "Not-a-Number" value. It indicates a value that is not a legal number.

typeof of NaN will return a Number.

To check if a value is NaN, we use the isNaN() function,

 Note- isNaN() function converts the given value to a Number type, and then equates to NaN.

isNaN("Hello")  // Returns true

isNaN(345)   // Returns false

isNaN('1')  // Returns false, since '1' is converted to Number type which results in 0 ( a number)

isNaN(true) // Returns false, since true converted to Number type results in 1 ( a number)
isNaN(false) // Returns false
isNaN(undefined) // Returns true

## 9. Explain passed by value and passed by reference.

In JavaScript, primitive data types are passed by value and non-primitive data types are passed by reference.

For understanding passed by value and passed by reference, we need to understand what happens when we create a variable and assign a value to it,

```
var x = 2;
```

In the above example, we created a variable x and assigned it a value of "2". In the background, the "=" (assign operator) allocates some space in the memory, stores the value "2" and returns the location of the allocated memory space. Therefore, the variable x in the above code points to the location of the memory space instead of pointing to the value 2 directly.

Assign operator behaves differently when dealing with primitive and non-primitive data types,

Assign operator dealing with primitive types:

```
var obj = { name: "Vivek", surname: "Bisht" };
```

```
var obj2 = obj;
```

In the above example, the assign operator directly passes the location of the variable obj to the variable obj2. In other words, the reference of the variable obj is passed to the variable obj2.

```
var obj = #8711;  // obj pointing to address of { name: "Vivek", surname: "Bisht" }
```

```
var obj2 = obj;
```

```
var obj2 = #8711; // obj2 pointing to the same address
```

```
// changing the value of obj1
```

```
obj.name = "Akki";
```

```
console.log(obj2);
```

```
// Returns {name:"Akki", surname:"Bisht"} since both the variables are pointing to the same address.
```

From the above example, we can see that while passing non-primitive data types, the assigned operator directly passes the address (reference).

Therefore, non-primitive data types are always passed by reference.