**Java Script – Introduction**

* JavaScript is the world's most popular programming language of the web.
* Since it is the scripting language, it doesn’t rely on the structure.
* JavaScript is easy to learn.

## [What is JavaScript?](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#what_is_javascript)

* JavaScript is a cross-platform, object-oriented scripting language used to make webpages interactive (e.g., having complex animations, clickable buttons, popup menus, etc.).
* More advanced server side versions of JavaScript such as **Node.js**, which allow you to add more functionality to a website such as realtime collaboration between multiple computers
* JavaScript can be connected to the objects of its environment to provide programmatic control over them.
* JavaScript contains a standard library of objects, such as *Array, Date*, and *Math,* and a core set of language elements such as operators, control structures, and statements.
* Client-side JavaScript extends the core language by supplying objects to control a browser and its Document Object Model (DOM).
* Server-side JavaScript extends the core language by supplying objects relevant to running JavaScript on a server

**Why Study JavaScript?**

JavaScript is one of the **3 languages** all web developers **must** learn:

   1. [**HTML**](https://www.w3schools.com/html/default.asp) to define the content of web pages

   2. [**CSS**](https://www.w3schools.com/css/default.asp)**/Bootstrap** to specify the layout of web pages

   3. **JavaScript** to program the behaviour of web pages

**Points to Remember**

* JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.
* Add a semicolon at the end of each executable statement, but it is optional.
* Ending statements with semicolon is not required, but highly recommended.

**JavaScript Can Change HTML Content**

Important JavaScript HTML methods is **getElementById( ),** to connect JS with HTML element

**Example**

document.getElementById("demo").innerHTML = "Hello JavaScript";

**Explanation:**

The example "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

**Program Example:**

<h2> JavaScript Display </h2>

<p **id="demo"**>JavaScript can change HTML content. </p>

<button type="button" onclick='document.getElementById**("demo"**).innerHTML = "Hello JavaScript!"'>Click Me!</button>

**JavaScript can change HTML attribute values**

<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>

<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='/Basics/asset/RedRose.jpg

'">Red Rose</button>

<img id="myImage" src="pic\_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='/Basics/asset/YellowRose.jpg '">Yellow Rose</button>

</body>

</html>

**Where shall we keep script?**

* In HTML,

JavaScript code is inserted between <script> and </script> tags.

* Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.
* Java script file can be stored externally and it can be attached with HTML page when required.
* Old JavaScript examples may use a type attribute: **<script type="text/javascript">.**
* The type attribute is not required. **JavaScript is the default scripting language in HTML.**

**JavaScript function is placed in the <head> section of an HTML page**

<!DOCTYPE html>

<html>

<head>

**<script>**

function myFunction() {

document.getElementById("demo").innerHTML = "Paragraph changed.";

}

**</script>**

</head>

<body>

<h2>JavaScript is in Head Section</h2>

**<p id="demo">A Paragraph.</p>**

**<button type="button" onclick="myFunction()">Try it-JS in Head</button>**

</body>

</html>

**JavaScript function is placed in the <head> section of an HTML page**

<!DOCTYPE html>

<html>

<body>

<h2>Demo JavaScript in Body</h2>

**<p id="demo">A Paragraph.</p>**

<button type="button" onclick="myFunction()">Try it</button>

**<script>**

**function myFunction() {**

**document.getElementById("demo").innerHTML = "Paragraph changed."; }**

**</script>** </body> </html>

**External JavaScript**

* External scripts are practical when the same code is used in many different web pages.
* JavaScript files have the **file extension .js.**
* External scripts cannot contain <script> tags.
* To use an external script, put the name of the script file in the **src (source) attribute of a <script> tag:**

**External file:** myScript.js

function myFunction() {

document.getElementById("demo").innerHTML = "Paragraph changed.";

}

**Example**

<script src="myScript.js"></script>

You can place an external script reference in <head> or <body> as you like.

**External JavaScript Advantages**

* It separates HTML and code
* It makes HTML and JavaScript easier to read and maintain
* Cached JavaScript files can speed up page loads
* To add several script files to one page - use several script tags:

## JavaScript Display Possibilities (Output statements)

JavaScript can "display" data in different ways:

* Writing into an HTML element, using innerHTML.
* Writing into the HTML output using document.write().
* Writing into an alert box, using window.alert().
* Writing into the browser console, using console.log().

**Display using innerHTML**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = 5 + 6;  
</script>

**Display using document.write**

<script>  
document.write(5 + 6);  
</script>

<h1>My First Web Page</h1>

<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>

**Display using window.alert**

* In JavaScript, the window object is the global scope object. This means that variables, properties, and methods by default belong to the window object.
* So one can skip the **window** keyword and is optional.

**Example:**

<script>  
window.alert(5 + 6); // alert( 5 + 6);  
</script>

**Display using console.log**

For debugging purposes, we can use console.log() method in the browser to display data.

<script>  
console.log(5 + 6);  
</script>

**Note:**

Changing the innerHTML property of an HTML element is a common way to display data in HTML.

## JavaScript Statements

JavaScript statements are composed of:

Comments, Keywords, Values, Expressions and Operators.

# **JavaScript Comments**

* JavaScript comments can be used to explain JavaScript code, and to make it more readable.
* JavaScript comments can also be used to prevent execution, when testing alternative code.

**Types of comment**

* Single line comment ---- Starts with //
* Multiline comment ---- /\* Multi line comment \*/

**JavaScript Keywords**

JavaScript statements often start with a keyword to identify the JavaScript action to be performed.

# **JavaScript Reserved Words**

In JavaScript you cannot use these reserved words as variables, labels, or function names:

abstract, arguments, await\*, boolean, break, byte, case, catch, char, class\*, const, continue, debugger, default, delete, do, double, else, enum\*, eval, export\*, extends\*, false, final, finally, float, for, function, goto, if, implements, import\*, in, instanceof, int, interface, let\*, long, native new, null, package, private, protected, public, return, short, static, super\*, switch, synchronized, this, throw, throws, transient, true, try, typeof, var, void, volatile, while, with, yield

Words marked with\* are new in ECMAScript 5 and 6. (After 2015)

## JavaScript Values

The JavaScript syntax defines two types of values:

* Fixed values
  + Fixed values are called **Literals**.
* Variable values
  + Variable values are called **Variables**.

## JavaScript Literals

The two most important syntax rules for fixed values are:

1. **Numbers** are written with or without decimals:

Example:

10.50  
1001

2. **Strings** are text, written within double or single quotes:

**Example:**

"New Delhi" (or) 'New Delhi'

‘e’ ( or ) “E”

‘5’ (or) “#”

**JavaScript Variables --** *Containers for storing the data*

4 Ways to Declare a JavaScript Variable:

* Using **var**
* Using **let -- block scope**
* Using **const -- block scope**
* Using nothing

// How to create variables:  
var x;  
let y;  
  
// How to use variables:  
x = 5;  
y = 6;  
let z = x + y;

### **Example using var**

var x = 5;  
var y = 6;  
var z = x + y;

### **Example using let**

let x = 5;  
let y = 6;  
let z = x + y;

### **Example without using var and let**

x = 5;  
y = 6;  
z = x + y;

**JavaScript constant**

* Constant values are declared in JavaScript using **const** keyword
* The value of the **const** will not be changed.
* Variables defined with **const** have block Scope.

**Example**

const VOTE\_AGE = 18;  
const PASS\_MARK = 40;  
let myage = VOTE\_AGE + 5;

VOTE\_AGE = VOTE\_AGE + 7; // Error

**When to Use JavaScript var?**

* Always declare JavaScript variables with **var, let**, or **const.**
* The **var** keyword is used in all JavaScript code from 1995 to 2015.
* The **let** and **const** keywords were added to JavaScript in 2015.
* If you want your code to run in older browsers, you must use **var.**

**JavaScript Let**

* The let keyword was introduced in ES6 (2015).
* Variables defined with let cannot be re-declared.
* Variables defined with let must be declared before use.
* Variables defined with let have block Scope

**Cannot be Redeclared**

Variables defined with **let** cannot be **redeclared**.

**Example**

**let x = "CG-VAK";  
let x = 1000;**  
// Syntax Error: 'x' has already been declared

Variables defined with **var** can be **redeclared**.

**Example**

**var x = "CG-VAK";  
var x = 1000;**

**Note:** The above re-declaration is possible using **var**, not with **let**.

**Block Scope**

* Before ES6 (2015), JavaScript had only **Global Scope** and **Function Scope**.
* ES6 introduced two important new JavaScript keywords: **let** and **const.**
* These two keywords provide **block Scope** in JavaScript.
* Variables declared inside a { } block cannot be accessed from outside the block:

**Example**

**{  
  let x = 2;  
}  
// x can NOT be used here**

**document.write (x); // Error will come**

* Variables declared with the var keyword can NOT have block scope.
* Variables declared inside a { } block can be accessed from outside the block.

**Example**

**{  
  var x = 2;  
}  
// x CAN be used here**

**Document.write (x); // Error will not come**

**Points to remember with constant**

## Use const to declare the following :

## •A new Array

## •A new Object

## •A new Function

## •A new RegExp

## Constant Objects and Arrays

* **const** does not define a constant value. But it defines a constant reference to a value.
* We can NOT Reassign a constant value / constant array / constant object
* We CAN Change the elements of constant array / properties of constant object

## Constant Arrays

* **You can change the elements of a constant array:**

**Example**

// You can create a constant array:  
const cars = ["Audi", "Volvo", "BMW"];  
  
// You can change an element:  
cars[0] = "Toyota";  
  
// You can add an element:  
cars.push("Hyundai");

* **You can NOT reassign the array:**

**Example**

const cars = ["Audi", "Volvo", "BMW"];  
  
cars = ["Toyota", "Volvo", "Audi"];    // ERROR

## Constant Objects

* **You can change the properties of a constant object.**

**Example**

// You can create a const object  
**const** car = {**type:**"Fiat", **model:**"500", **color:**"white"};

// You can change a property  
car**.color** = "red";  
  
// You can add a property:  
car**.owner** = "Johnson";

* **You can NOT reassign the object**

**Example**

const car = {type:"Fiat", model:"500", color:"white"};  
car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR

**Operators in JS**

* Arithmetic Operators
* Assignment Operators
* Comparison Operators
* Logical Operators
* Bitwise operators
* Special Operators
* **JavaScript Arithmetic Operators**

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

let a = 3;  
let x = (100 + 50) \* a;

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| \*\* | Exponentiation ([ES2016](https://www.w3schools.com/js/js_2016.asp)) |
| / | 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 |

**Adding JavaScript Strings**

The + operator can also be used to add (concatenate) strings.

**Example**

let text1 = "CG-VAK ";  
let text2 = "Coimbatore";  
let text3 = text1 + " " + text2;

This can be written as

let text1 += text2; // text1 = text1 + text2; // Here + is concatenation operator

**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

* **JavaScript Comparison Operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| = = | equal to |
| = = = | equal value and equal type [Identical (equal and of same type)] |
| != | not equal |
| != = | not equal value or not equal type [Not identical] |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? : | ternary operator |

* **JavaScript Logical Operators**

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

* **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 |

**Note:**

* 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
* **Additional JS Assignment Operators**
* **Shift Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| <<= | x <<= y | x = x << y |
| >>= | x >>= y | x = x >> y |
| >>>= | x >>>= y | x = x >>> y |

* **Bitwise Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| &= | x &= y | x = x & y |
| ^= | x ^= y | x = x ^ y |
| |= | x |= y | x = x | y |

* **Logical Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| &&= | x &&= y | x = x && (x = y) |
| ||= | x ||= y | x = x || (x = y) |
| ??= | x ??= y | x = x ?? (x = y) |

**?? Operator**

?? - The **nullish coalescing (**??**)** operator is a logical operator that returns its right-hand side operand when its left-hand side operand is [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/null) or [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined), and otherwise returns its left-hand side operand.

**Exampe:**

const result1 = null ?? 'CG-VAK';

console.log(result1);

// Expected output: "CG-VAK"

const result2 = 10 ?? 100;

console.log(result2);

// Expected output: 0

const result3 = null ?? undefined;

console.log(result3);

// Expected output: undefined

## JavaScript Special Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| (?:) | Conditional Operator returns value based on the condition. It is like if-else. |
| , | Comma Operator allows multiple expressions to be evaluated as single statement. |
| delete | Delete Operator deletes a property from the object. |
| in | In Operator checks if object has the given property |
| instanceof | checks if the object is an instance of given type |
| new | creates an instance (object) |
| typeof | checks the type of object. |
| void | it discards the expression's return value. |
| yield | checks what is returned in a generator by the generator's iterator. |

**JavaScript has the following Datatypes**

1. String  
2. Number  
3. BigInt  
4. Boolean  
5. Undefined  
6. Null  
7. Object

**The Object Datatype**

The object data type can contain:

1. An object  
2. An array  
3. A date

**Examples**

// Numbers:  
let length = 16;  
let weight = 7.5;  
  
// Strings:  
let color = "Yellow";  
let lastName = ‘Arulseeli’;  
  
// Booleans  
let x = true;  
let y = false;  
  
// Object:  
const person = {firstName:"Thangam",lastName:"Arulseeli"};  
  
// Array object:  
const cars = ["Fiat", "Maruti", "BMW"];  
  
// Date object:  
const date = new Date("2023-02-20");

What will be the output of the following expression?

let x = 20 + "Volvo"; --- 20Volvo

let x = "20" + "Volvo"; --- 20Volvo

let x = 20 + 23 +  "Volvo"; --- 43Volvo

let x = "Volvo" + 20 + 23; --- Volvo2023

let x = 1 + (2 + “3” ); -- 33

let x = 1 + (2 + “3” ); -- 123

let x = 1 + “2” + 3 ; -- 123

* + Hoisting
  + Use Strict

**Note:**

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

## 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 = "CG-VAK";  // 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-piont):

## float (32-bit), double (64-bit).

## Javascript are always one type:

## double (64-bit floating point).

## JavaScript BigInt

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

JavaScript BigInt is a new datatype (2020) 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, which is used to group set of elements under the common variable.
* Array items are separated by commas.
* Array indexes are zero-based, which means the first item is [0], second is [1], and so on.
* The following code declares (creates) an array called cars, containing three items (car names):

**Example**

const cars = ["Fiat", "Volvo", "BMW"];

## JavaScript Objects

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

**Example**

**const person = {**

**firstName:"Mark",**

**lastName:"Antony",**

**age:50,**

**city:"Coimbatore"**

**};**

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 "CGVAK"         // Returns "string"  
typeof "New Delhi"     // Returns "string"

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.

**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"**

**Scope of the variables:**

let b=10;

//let b="CGVAK";

console.log(b);

if (b==10)

{

    let p=100;

    var q=200;

    const r=300;

}

console.log("Q="+q);

console.log("R="+r); // Error – Not defined

console.log("P="+p); // Error – Not defined