

# ISS JavaScript Tutorial

## What is JavaScript?

JavaScript is a programming language used primarily to add interactivity and dynamic functionality to web pages. It can be used to manipulate HTML and CSS, create animations, respond to user input, and communicate with servers to update content dynamically. It is a versatile language that is widely used on the web, and a fundamental understanding of JavaScript is essential for any web developer.

JavaScript is a Turing-complete programming language. That is, it can solve any task that can be done in other popular programming languages such as C or Python. Additionally, with the help of external libraries and frameworks, JavaScript can also be used to create full-fledged web applications, games, and other complex software.

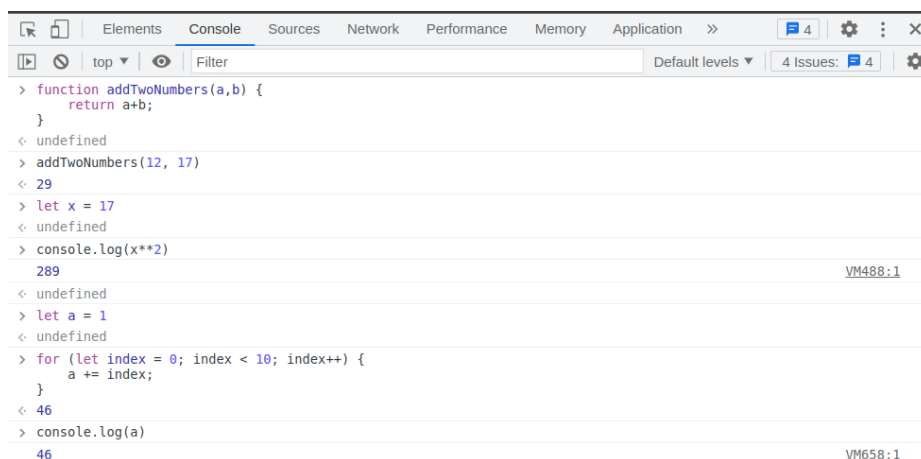
JavaScript is one of the most popular scripting languages (lightweight programming languages) on the internet and works in all major browsers. It can be run independently or can be embedded directly into HTML pages.

Is Java the same as JavaScript? No

- Java - Programming language, used for developing large-scale enterprise applications, desktop applications, mobile apps, etc.
- JS - Lightweight scripting language, used for the development of interactive webpages

## Running JavaScript

JavaScript requires a JavaScript engine for interpretation and execution. All modern browsers come equipped with a JS engine. You can run JS in the console window of your browser (*An example is shown below*)



```
> function addTwoNumbers(a,b) {  
  return a+b;  
}  
< undefined  
> addTwoNumbers(12, 17)  
< 29  
> let x = 17  
< undefined  
> console.log(x**2)  
289  
< undefined  
> let a = 1  
< undefined  
> for (let index = 0; index < 10; index++) {  
  a += index;  
}  
< 46  
> console.log(a)  
46
```

**Additional information:** It is also possible to run JavaScript outside a web browser using a JavaScript runtime environment such as node.js

It is also possible to execute JavaScript code by including it with an HTML file and executing it using a web browser. This can be done using Inline JS or External JS.

1. **External JavaScript** refers to code that is stored in a separate file with a .js extension and is linked to an HTML document using the `<script>` tag. This code is placed in a separate file and loaded into the HTML document using the tag below.

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

The `<script>` tag for an external JavaScript file should be placed in the head or body section of an HTML document, depending on the specific requirements of the JavaScript code.

- By placing the `<script>` tag in the head section, the JavaScript code is loaded and parsed before the rest of the page is rendered, which can improve performance. However, it's important to keep in mind that certain references to HTML elements using selectors (covered below) may not be available since they haven't been rendered yet. This can be avoided by placing the code that references these elements after they have been loaded, or by using the `window.onload` event to delay the execution of the JavaScript code until the entire page has been loaded.
- Placing the `<script>` tag at the end of the body will allow the HTML content to be loaded and rendered before the JavaScript code is executed. This can improve initial page load times.

Note: You can link multiple scripts in an HTML document by using multiple `<script>` tags at different locations within the document.

2. **Inline JavaScript** code is written directly in the HTML file using the `<script>` tag.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Inline JavaScript Example</title>
  </head>
  <body>
    <h1>Hello World</h1>
    <script>
      alert('Hello World!');
    </script>
  </body>
</html>
```

## Basic Syntax

You'll find many similarities in the syntax of C and JavaScript. Here is an overview of the JavaScript syntax:

- JavaScript statements end with a semicolon (;) like C, but it is optional in many cases.
- JavaScript uses curly braces ({} ) for code blocks, like C.
- Variables are declared using the `var` , `let` , or `const` keyword, followed by the variable name and an optional initial value. Like C, JavaScript has various data types including numbers, strings, booleans, objects, and arrays.
- JavaScript also has operators, such as +, -, \*, /, %, and ++/--. They work the same way as in C, with a few differences like the `===` operator for strict equality testing.
- Like C, JavaScript has conditional statements such as `if/else` , `switch/case` , and ternary operator.
- JavaScript has loops, including `for` , `while` , and `do/while` loops, that are similar to C loops.

- JavaScript also has functions, which are declared using the `function` keyword followed by the function name and a set of parentheses containing the function parameters. Like C, functions can take parameters, and they can return a value.

### Defining Variables:

- Case sensitive; Can only start with a letter/ underscore character
- `var`, `const`, `let` keywords can be used for defining variables.
  - `var`: Used to declare a variable that has global/ functional scope.
  - `const`: Declares a constant variable that cannot be re-assigned.
  - `let`: Variable can be accessed only within the block it is defined in.

The `let` and `const` keywords were added to JavaScript in 2015 and thus may not work in older browsers.

```
let a = 5;
const pi = 3.14159;
var public_key = "43DckKWJ87nSozT4pQfIBp1"
```

### Operators:

Most operators in JavaScript are similar to those in C with the notable exception of the strict equality operator `===`

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2,5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Operator	Example	Is The 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

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5"  x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

### Conditional Statements and Loops:

The syntax of if-else and switch case is pretty similar to that of C.

```
let num = 5;
if (num > 10) {
  console.log("The number is greater than 10");
} else {
  console.log("The number is less than or equal to 10");
}
```

For loops can be written in JavaScript as follows:

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

JavaScript also has `while` and `do...while` loops.

### Functions:

A function can be defined in JavaScript as follows,

```
function addNumbers(num1, num2) {
  return num1 + num2;
}

/* The function above can be accessed as follows */
let sum = addNumbers(3, 5);
console.log(sum); // Output: 8
```

## ▼ Example: Prime Checker

# Prime Number Checker

Enter a number:

17 is prime!

```
<!DOCTYPE html>
<html>
  <head>
    <title>Prime Number Checker</title>
  </head>
  <body>
    <h1>Prime Number Checker</h1>
    <label for="number-input">Enter a number:</label>
    <input type="number" id="number-input" />
    <button onclick="checkPrime()">Check</button>
    <p id="result"></p>

    <script>
      function isPrime(n) {
        if (n <= 1) {
          return false;
        }

        for (let i = 2; i <= Math.sqrt(n); i++) {
          if (n % i === 0) {
            return false;
          }
        }

        return true;
      }

      function checkPrime() {
        const input = document.getElementById('number-input').value;
        const result = document.getElementById('result');

        if (isPrime(input)) {
          result.innerText = `${input} is prime!`;
        } else {
          result.innerText = `${input} is not prime.`;
        }
      }
    </script>
  </body>
</html>
```

## Selectors

In JavaScript, selectors are used to identify and manipulate elements on a webpage. Selectors enable you to target particular elements based on their ID, class, tag name, or other characteristics. These methods allow you to interact with and modify the content and presentation of the webpage, creating dynamic and interactive web experiences.

Here are a few of the most commonly used selectors:

1. **getElementById()**: This method selects an element based on its ID attribute.

```
document.getElementById("element-id");
```

```
let myElement = document.getElementById("my-element");
myElement.textContent = "New text content";
```

2. **getElementsByClassName()**: This method selects elements based on their class name.

```
document.getElementsByClassName("class-name");

let myElements = document.getElementsByClassName("my-class");
for (let i = 0; i < myElements.length; i++) {
  myElements[i].style.backgroundColor = "red";
}
```

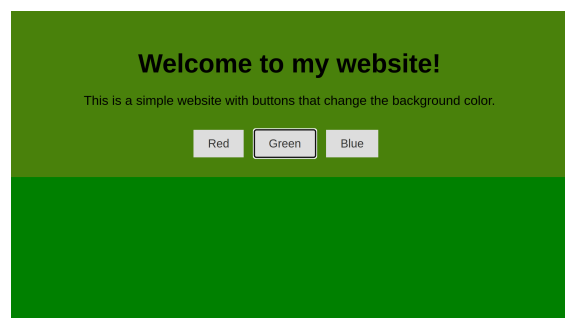
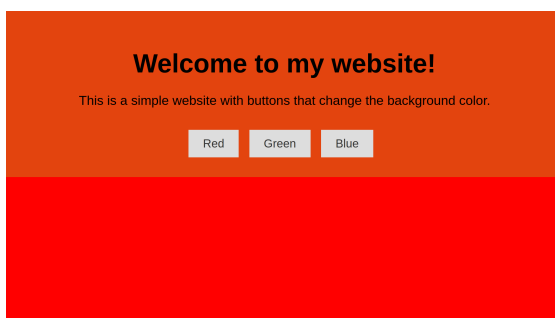
3. **getElementsByTagName()**: This method selects elements based on their tag name.

```
document.getElementsByTagName("tag-name");

let myElements = document.getElementsByTagName("p");
for (let i = 0; i < myElements.length; i++) {
  myElements[i].style.fontWeight = "bold";
}
```

There are other selectors such as `querySelector()` and `querySelectorAll()`

## ▼ Example 2: Background colour changer



Here is the code for the website,

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Button Background Changer</title>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <div class="container">
    <h1>Welcome to my website!</h1>
    <p>This is a simple website with buttons that change the background color.</p>
    <button id="redButton">Red</button>
```

```

    <button id="greenButton">Green</button>
    <button id="blueButton">Blue</button>
  </div>
  <script src="script.js"></script>
</body>
</html>

```

```

// script.js
const redButton = document.getElementById("redButton");
const greenButton = document.getElementById("greenButton");
const blueButton = document.getElementById("blueButton");

redButton.addEventListener("click", () => {
  document.body.style.backgroundColor = "red";
});

greenButton.addEventListener("click", () => {
  document.body.style.backgroundColor = "green";
});

blueButton.addEventListener("click", () => {
  document.body.style.backgroundColor = "blue";
});

```

```

/* style.css */
body {
  background-color: #f2f2f2;
  font-family: Arial, sans-serif;
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
  text-align: center;
}

h1 {
  font-size: 36px;
  margin-bottom: 20px;
}

p {
  font-size: 18px;
  margin-bottom: 30px;
}

button {
  background-color: #ddd;
  border: none;
  color: #333;
  cursor: pointer;
  font-size: 16px;
  padding: 10px 20px;
  margin-right: 10px;
}

button:hover {
  background-color: #555;
  color: #fff;
}

```

## Fetch Requests

- `fetch()` is a built-in function in JavaScript that allows you to make HTTP requests to servers and retrieve data from APIs.
- Example of a simple `GET` request using `fetch()`:

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

- Explanation of `GET` example:
  - We pass in the URL we need to make a request to in the `fetch()` function.
  - `fetch()` returns a promise which is handled using the `then()` function (In layman's terms, a promise basically provides a way to deal with operations that take time to complete {refer point 3 of additional knowledge - asynchronous operations}).
  - `json()` function is used to parse the obtained response to JSON.
  - `then()` uses the `console.log()` to log JSON data to console.
  - `catch()` is used to handle errors.
- Another type of HTTP request is the `POST` request. `GET` requests are typically used to retrieve data from a server, while `POST` requests are used to submit data to a server (can be to retrieve data based on some input data). You can read up about `POST` requests on your own.

## Example 3: Chuck Norris API

### Chuck Norris Jokes

Get a new joke

Chuck Norris was asked to donate blood. He proceed to donate 6 quarts... Simply by roundhouse kicking the flabotomist.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Chuck Norris Jokes</title>
  </head>
  <body>
    <h1>Chuck Norris Jokes</h1>
    <button id="new-joke">Get a new joke</button>
    <div id="joke"></div>

    <script>
      const newJokeButton = document.querySelector('#new-joke');
      const jokeDiv = document.querySelector('#joke');

      function getJoke() {
        fetch('https://api.chucknorris.io/jokes/random')
          .then(response => response.json())
          .then(data => {
            jokeDiv.innerHTML = `<p>${data.value}</p>`;
          })
      }
    </script>
  </body>
</html>
```



```

        .catch(error => console.error(error));
    }
    getJoke(); // Get a joke on page load
    newJokeButton.addEventListener('click', getJoke);
</script>
</body>
</html>

```

## Additional Knowledge

### 1. “this” keyword

- a. Refers to the current execution context or the current object of which the function is a method.
- b. When a function is called without an explicit context, the keyword refers to the global context.
- c. Examples:

```

function sayHello() {
    console.log("Hello, " + this.name);
}

const person1 = { name: "John" };
const person2 = { name: "Jane" };

sayHello(); // Output: "Hello, undefined"
person1.sayHello = sayHello;
person1.sayHello(); // Output: "Hello, John"
person2.sayHello = sayHello;
person2.sayHello(); // Output: "Hello, Jane"

```

```

const person = {
    firstName: "John",
    lastName: "Doe",
    fullName: function() {
        return this.firstName + " " + this.lastName;
    }
};

console.log(person.fullName()); // Output: "John Doe"

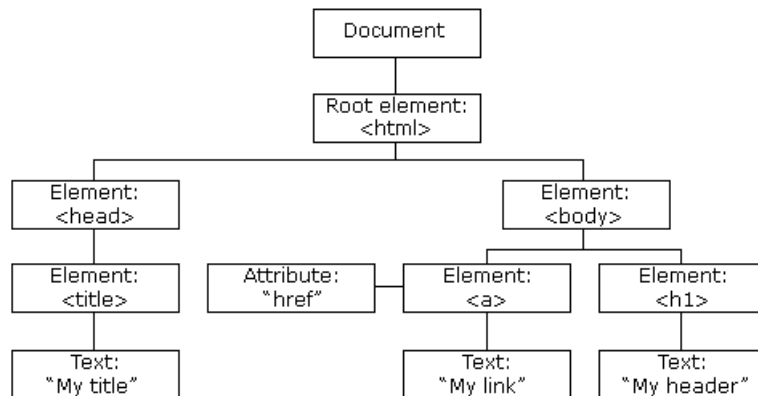
```

### 2. Javascript HTML Document Object Model

The Document Object Model (DOM) is a programming interface that allows web developers to access and manipulate the contents of a web page. The DOM represents the web page as a tree-like structure where each node corresponds to an element or object on the page. JavaScript can be used to access and modify these nodes, allowing developers to create dynamic and interactive web pages.

Using the DOM, developers can add or remove elements from the page, change the content of existing elements, and respond to user interactions such as clicks or mouse movements. The DOM also enables developers to create animations, validate form data, and perform other tasks that make web pages more interactive and engaging. By understanding the basics of the DOM, beginners can start building their own web applications and become proficient in web development.

## The HTML DOM Tree of Objects



### 3. Asynchronous Nature of JavaScript

Javascript code is read line by line unless it involves “function calls” or “asynchronous operations”.

```
writelnToConsole();

function writelnToConsole() {
  console.log("This line was reached.");
}
```

In the above code, although the function `writelnToConsole()` is called before it's defined, it gets executed.

When a function is called, the engine will jump to the function definition and start executing the code inside the function. Once the function has completed execution, the engine will return to the point in the code where the function was called and continue executing the remaining lines.

When the code includes asynchronous operations, such as fetching data from a server or waiting for user input, the engine does not wait for the operation to complete before moving on to the next line of code. Instead, it starts the operation and then continues executing the rest of the code. Once the operation is complete, the engine will execute the callback function associated with that operation.