**History Of JavaScript/ECMA script/Live script**

JavaScript was **invented by Brendan Eich in 1995**. It was developed for Netscape 2, and became the ECMA-262 standard in 1997. After Netscape handed JavaScript over to ECMA, the Mozilla foundation continued to develop JavaScript for the Firefox browser. Mozilla's latest version was 1.8

"JavaScript" is a trademark of **Oracle Corporation** in the United States.

JavaScript is a cross-platform, object-oriented scripting language used to make webpages interactive (e.g., having complex animations, clickable buttons, popup menus, etc.). There are also more advanced server side versions of JavaScript such as Node.js, which allow you to add more functionality to a website than downloading files (such as realtime collaboration between multiple computers). Inside a host environment (for example, a web browser), JavaScript can be connected to the objects of its environment to provide programmatic control over them.

This means that in the browser, JavaScript can change the way the webpage (DOM) looks. And, likewise, Node.js JavaScript on the server can respond to custom requests sent by code executed in the browser.

**What is binary and assembly level language?**

The major difference between machine language and assembly language is that **machine language is referred to as a binary language**. It can be run on a computer directly. While an assembly language is a low-level programming language that must be converted into machine code using software called an assembler.

The most commonly used assembly languages include ARM, MIPS, and x86.

**Translators**

Translators are used to translate assembly level and high level language into Machine level language.

* Assemblers.
* Interpreter.
* Compiler.

**How JavaScript is executed in Browsers without any interpreter?**

Since, In every browser there will be a inbuilt JavaScript Engine we can JavaScript in browsers

Ex: Chrome -V 8, InternetExplorer-chatra.

**Important Points:**

JS is a Case Sensitive language.

JS is a loosely typed language.

JS is a scripting language.

JS is a Interpreted language.

JS is a Object-oriented/Object Based Programming language.

**How to declare a variable in JS?**

We can declare variable by using keywords like var,let,const.

**var:**

if we are using var we can redeclare and reinitialize the variable.

Ex: var a=10 **✓**

var a= 20 **✓** ☓

**let:**

if we are using let we cannot redeclare but we can reinitialize the variable.

Ex: let a=10 **✓**

let a= 20 ☓

a=30 **✓**

**const:**

if we are using const we cannot redeclare /reinitialize the variable.

Ex: let a=10 **✓**

let a= 20 ☓

a=30 ☓

**DataTypes In JavaScript:**

**Primitive datatype:**

String: It is one of the datatype We can specify by using ‘’ /” ” .

Boolean: true/false.

Number:

Undefined: whenever we didn’t declare /initialize the variable.

Null:

**Non Primitive datatype:**

Arrays:

Object:

Functions:

**Example**

var a=" Nithish"

var a="jagadeesh"//we can redeclare and reintialize

let b=56

    b=66 // we cannot declare again but we can reintialize

const c=true // we cannot reintialize and redeclare

var e=null

console.log(typeof(a)) //this will reflect in console(developer side)//String

document.write("nithish")//this will reflect on webpages(client side)

console.log(typeof(b))//number

console.log(typeof(c))//boolean

console.log(typeof(d))//undefined

console.log(typeof(e))//object

**Functions in JavaScript:**

**typeof(variable/data):** it will return the datatype.

Ex:var a=”nithi”

Console.log( typeof(a)) //string

**How to change the datatype explicitely?**

var a=10

console.log(typeof(String(a))) //string

**Difference between document.write and console.log**

**document.write:** it is reflected in webpage(client side).

**console.log:** it is reflected in a console window(developer side).

**Inline Embedded javascript v/s external javascript.**

JavaScript code that is embedded with an HTML document is referred to as inline JavaScript. On the other hand, the HTML document may refer to a separate file that contains the JavaScript program(.js file), in which case codit is referred to as external JavaScript.

**Internal execution:**

<html lang="en">

<head>

    <title>Nithish</title>

</head>

<body>

    <Script>document.write("kk")</Script>

</body>

</html>

**External execution:**

<html lang="en">

<head>

    <title>Nithish</title>

</head>

<body>

    <Script src="./Demo.js"></Script>

</body>

</html>

**Demo.js (External Javascript file)**

var a=" ";

console.log(typeof(a))

document.write("nithish")

**JavaScript libraries:**

Libraries are developed by JS developers which contains reusable functons and reusable code .

By using JS libraries we can achieve codeOptimization.

**Ex:**Bootsrap,JQuery,Load# etc..,

**Javascript Frameworks:**

* **Node JS** ------ for developing webapplication.
* **React JS** ------ for developing webapplication.
* **React Native** ------ for developing client- server application.
* **Angular JS** ------ for developing single page web application(gmail,map).
* **Electron JS** ------ for developing stand-alone application.
* **Tensor flow** ----- for developing Artificial intelligence and machine learning application.

**Conditional statements**

1. if .
2. if else.
3. else if ladder.
4. switch.

**1.if statement:** whenever the given condition is true then,if block will execute.

var pageTitle="Doctors-Dashboard"

if(pageTitle==="Doctors-Dashboard")

console.log("Dashboard page is displayed")

**2.if else statement:** whenever the given condition is true then,if block will execute otherwise else block will executes.

var pageTitle="Doctors-Dashboard"

if(pageTitle==="Doctors-Dashboard")

console.log("Dashboard page is displayed")

else

console.log("Dashboard page is not displayed")

**3.else if statement:** whenever the given condition is true then, that respective block will executes otherwise else block will executes.

var pageTitle="Doctors-login"

if(pageTitle==="Doctors-login")

console.log("Still in Home page,not navigated")

else if(pageTitle==="Doctors-Dashboard")

console.log("Dashboard page is displayed")

else

console.log("Dashboard page is not displayed")

**4.switch statement:** Here it will search for the particular value in case block ,then that particular will executes if there is no matching case block then default block will executes.

var Dashboardname="Admin-dashboard"

switch(Dashboardname){

    case "Doctors-dashboard":console.log("Doctors-dashboard is displayed")

    break

    case "Admin-dashboard":console.log("admin-dashboard is displayed")

    break

    case "Patient-dashboard":console.log("Doctors-dashboard is displayed")

    break

     default: console.log("invalid page")

}

**Loops :**

1. for. 🡪 (to prints everything even it is undefined)
2. while.
3. do while.
4. for in🡪 (it will skip index of unassigned/undefined value)
5. for of🡪(to prints everything even it is undefined)
6. forEach( ) 🡪(it will skip unassigned/undefined value)

**1.for loop:**  Entry controlled loop ,Before executing statements itself its checking for condition.

In for loop if we wan to make infinite loop we no need to specify condition,only 2 semi-colon(;) is mandatory.

var doc=["nithish",121,"sathish",122,"suresh",123,"mahesh",124]

for (let index = 0; index < doc.length; index++) {

    if(doc[index]==="mahesh"){

    console.log("Doctor's profile has been added successfully and his id is "+doc[++index] )

    var flag =true

    break;

    }

    index++;

}

if(flag ==false)

console.log("doctor profile is not added")

**2.while loop:** Entry controlled loop ,Before executing statements itself its checking for condition. Here providing condition is mandatory.

var doc=["nithish",121,"sathish",122,"suresh",123,"mahesh",124]

var index=0

while(index < doc.length){

    if(doc[index]==="mahesh"){

        console.log("Doctor's profile has been added successfullyand his id is "+doc[++index] )

        var flag=true

        }

       index= index+2

    }

    if(flag ==false)

    console.log("doctor profile is not added")

**3.do while loop:** Exit controlled loop, it will executes once before checking for condition.

var doc=["nithish",121,"sathish",122,"suresh",123,"mahesh",124]

var index=0

do {

    if(doc[index]==="mahesh"){

        console.log("Doctor's profile has been added successfully and his id is"+doc[++index])

        var flag=true

        break;

        }

index=index+2

} while (index < doc.length);

if(flag==false)

console.log("doctor profile is not added")

**4.for in loop:** JS developers developed this loop in such a way that, it should iterates an array from 0th index to array.Length based on index value .

var doc= new Array("nithish","sathish","suresh","mahesh")

for (const index in doc) {

    console.log(index+ "--->"+doc[index])

    if(doc[index]==="mahesh"){

        console.log("Doctor's profile has been added successfully ")

        var flag=true

        break;

        }

    }

    if(flag==false)

    console.log("doctor profile is not added")

**5.for of loop:** JS developers developed this loop in such a way that, it should iterates one by one value from an array from 0th index to array.Length directly .

var doc=["nithish",121,"sathish",122,"suresh",123,"mahesh",124]

for (const names of doc) {

    if(names==="mahesh"){

        console.log("Doctor's profile has been added successfully")

        var flag=true

        break;

        }

    }

    if(flag==false)

    console.log("doctor profile is not added")

**6.forEach() :** It is an predefined method of Array By using array reference we are calling this method by passing callback function which takes three argumments (value,index,array) as a parameter .

**Functions :**

1. Standard function/function declaration.
2. Function expression.
3. Immediate invoke functions.
4. Arrow functions.

**1.Standard function/function declaration:**

**function doc(name){**

**console.log("doctor name is "+name);**

**}**

**2. Function Expression:**

**var doc=function (name){**

**console.log("doctor name is "+name);**

**}**

**3. Immediate invoke function:**

**(function (name){**

**console.log("doctor name is "+name);**

**})(“nithish”)**

This function invoke immediately once after declared.

**Avoid polluting the global namespace.**

Because our application could include many functions and global variables from different source files, it's important to limit the number of global variables. If we have some initiation code that we don't need to use again, we could use the IIFE pattern. As we will not reuse the code again, using IIFE in this case is better than using a function declaration or a function expression.

**4. Arrow function:**

**a) Var doc= (name)=>{console.log("doctor name is "+name)**

**console.log("Hi”)**

**}**

**b) Var doc= (name)=>console.log("doctor name is "+name)// if it is single statement then, no need of braces and no need of using return keyword for single return statement.**

**Difference between function declaration and function expression:**

|  |  |
| --- | --- |
| **Function declaration:**  **doc("nithish")//we can call it before also**  **function doc(name){**  **console.log("doctor name is "+name);**  **}**  **doc("sathish")** | **Function expression:**  **//doc("nithish") if we call it before it will through error i.e. doc is not a function**  **var doc=function (name){**  **console.log("doctor name is "+name);**  **}**  **doc("nithish")** |

**Callback Functions:** Calling one function inside another function is known as callback functions.

function doctor(dname,pname,operation){

    console.log (dname+'is added successfully')

    operation(pname)

}

function patient(pname){

    console.log (pname+' is admitted successfully')

}

doctor("nithish","vinod",patient)

Output

nithishis added successfully

vinod is admitted successfully

**Closure Property:**

Hiding /closing accessibility of an variable outside the functions.

Or

A closure can be defined as a JavaScript feature in which the inner function has access to the outer function variable. In [JavaScript](https://www.javatpoint.com/javascript-tutorial), every time a closure is created with the creation of a function.

The closure has three scope chains listed as follows:

* Access to its own scope.
* Access to the variables of the outer function.
* Access to the global variables.

**Example:** (Access to the variables of the outer function.)

**function fun()**

**{**

**var a = 4; // 'a' is the local variable, created by the fun()**

**function innerfun(b) // the innerfun() is the inner function, or a closure**

**{**

**return a\*b;**

**}**

**return innerfun;**

**}**

**var output = fun();**

**document.write(output(5)); // 20**

**Explination:** Whenever we declare variable inside a functions, it belongs to function scope itself whether it is var,let,const anyting.but we can access variable from global scope inside the functions.

Whenever we declare variable inside a block, it belongs to that scope itself when they declare with let and const keyword but var is gloabally accessible .but we can access variable from global scope inside the blocks.

|  |  |
| --- | --- |
| **Functions** | **Blocks** |
| **var a = 10;**  **let b=20;**  **const c=30;**  **function add( ) {**  **var x = 2;**  **let y=2;**  **const z=3;**  **console.log(a);//10**  **console.log(b)//20**  **console.log(c);//30**  **console.log(x);//2**  **console.log(y)//2**  **console.log(z);//3**  **}**  **console.log(a);//10**  **console.log(b)//20**  **console.log(c);//30**  **console.log(x);//undefined**  **console.log(y)//undefined**  **console.log(x);//undefined** | **var a = 2;**  **let b=2;**  **const c=3;**  **if (Math.random() > 0.5) {**  **var x = 1;**  **let y=2;**  **const z=3;**  **console.log(a);//10**  **console.log(b)//20**  **console.log(c);//30**  **console.log(x);//2**  **console.log(y)//2**  **console.log(z);//3**  **} else {**  **var x = 2;**  **let y=2;**  **const z=3;**  **}**  **console.log(a);//10**  **console.log(b)//20**  **console.log(c);//30**  **console.log(x);//2 ,since it is var we can access**  **console.log(y)//undefined**  **console.log(x);//undefined** |

**Note:**Variables created without a declaration keyword (var, let, or const) are always global, even if they are created inside a function.

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

**Object :**

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc. JavaScript is an object-based language. Everything is an object in JavaScript.

**In JavaScript, almost "everything" is an object.**

* Booleans can be objects (if defined with the new keyword)
* Numbers can be objects (if defined with the new keyword)
* Strings can be objects (if defined with the new keyword)
* Dates are always objects
* Maths are always objects
* Regular expressions are always objects
* Arrays are always objects
* Functions are always objects
* Objects are always objects

All JavaScript values, except primitives, are objects.

**JavaScript Primitives**

A primitive value is a value that has no properties or methods.

3.14 is a primitive value

A primitive data type is data that has a primitive value.

JavaScript defines 7 types of primitive data types:

**Examples**

* **string**
* **number**
* **boolean**
* **null**
* **undefined**
* **symbol**
* **bigint**

**Immutable**

Primitive values are immutable (they are hardcoded and cannot be changed).

if x = 3.14, you can change the value of x, but you cannot change the value of 3.14.

**Objects are Variables**

JavaScript variables can contain single values:

Example

let person = "John Doe";

JavaScript variables can also contain many values.

Objects are variables too. But objects can contain many values.

Object values are written as **name : value** pairs (name and value separated by a colon).

**Example**

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

A JavaScript object is a collection of **named values**

It is a common practice to declare objects with the const keyword.

1. **Using object literals:**

Literals are smaller and simpler ways to define objects.We simple define the property and values inside curly braces as shown below:

|  |
| --- |
| **//creating js objects with object literal**  **let car = {**  **name : 'GT',**  **maker : 'BMW',**  **engine : '1998cc'**  **start : function(){**  **console.log('Starting the engine...');**  **};**  **//property accessor**  **console.log(car.name); //dot notation**  **console.log(car['maker']); //bracket notation**  **car.start();//invoking fnction**  **To add new Property**  **Car.stop= function(){**  **console.log('Stopping the engine...');**  **};**  **Car.stop()// Stopping the engine...**  **To modify Property**  **Car.stop= function(){**  **console.log('switchoff the engine...');**  **};**  **Car.stop()//switchoff the engine...**  **To delete Property**  **delete Car.stop**  **Car.stop()//undefined** |

1. **Creating object with a constructor:**

One of the easiest ways to instantiate an object in JavaScript. Constructor is nothing but a function and with help of new keyword, constructor function allows to create multiple objects of same flavor as shown below:

|  |  |
| --- | --- |
| **//simple function**  **function vehicle(name,maker,engine){**  **this.name = name;**  **this.maker = maker;**  **this.engine = engine;**  **}**  **//new keyword to create an object**  **let car  = new vehicle('GT','BMW','1998cc');**  **//property accessors**  **console.log(car.name);**  **console.log(car.maker);**  **console.log(car['engine']);**   1. **Using new Keyword:**   By using new Keyword we can create object.  **var doctor= new Object()**  **doctor.name="nithish"**  **doctor.age=25**  **doctor.patient=()=>"vinod is a patient"**  **console.log(doctor.patient());**  **console.log(doctor.age);**   1. **Using es6 classes:**   ES6 supports class concept like any other Statically typed or object oriented language. So, object can be created out of a class in javascript as well as shown below:   |  | | --- | | **class Vehicle {**  **constructor(name, maker, engine) {**  **this.name = name;**  **this.maker =  maker;**  **this.engine = engine;**  **}**  **}**    **let car1 = new Vehicle('GT', 'BMW', '1998cc');**    **console.log(car1.name);  //GT** |   **Array:**  An array is a special variable, which can hold more than one value:  Type1:  **const cars = ["Saab", "Volvo", "BMW"];**  **Type2:**  **const cars = []; cars[0]= "Saab"; cars[1]= "Volvo"; cars[2]= "BMW";**  **Type3:**  **const cars = new Array("Saab", "Volvo", "BMW");**  **Diffrence of Arraydeclaration by usin var and const**  Arrays declared with var can be initialized at any time.  You can even use the array before it is declared:  **cars = ["Saab", "Volvo", "BMW"]; var cars;**  The keyword const is a little misleading.  It does NOT define a constant array. It defines a constant reference to an array.  Because of this, we can still change the elements of a constant array.  Elements Can be Reassigned  You can change the elements of a constant array:  **Example**  **// You can create a constant array: const cars = ["Saab", "Volvo", "BMW"];  // You can change an element: cars[0] = "Toyota";  // You can add an element: cars.push("Audi");**  **Associative Arrays**  Many programming languages support arrays with named indexes.  Arrays with named indexes are called associative arrays (or hashes).  JavaScript does not support arrays with named indexes.  In JavaScript, arrays always use numbered indexes.  **The Difference Between Arrays and Objects**  In JavaScript, arrays use numbered indexes.  In JavaScript, objects use named indexes.  **How to Recognize an Array**  A common question is: How do I know if a variable is an array?  The problem is that the JavaScript operator **typeof** returns "object"  **Solutions:**  **const fruits = ["Banana", "Orange", "Apple"];**  **Array.isArray(fruits); //true**  **fruits instanceof Array; //true**  **Array Methods**   1. **concat()** 2. **copyWithin()** 3. **entries()** 4. **every()** 5. **fill()** 6. **filter()** 7. **find()** 8. **findIndex()** 9. **forEach()** 10. **includes()** 11. **indexOf()** 12. **join()** 13. **keys()** 14. **lastIndexOf()** 15. **length** 16. **map()** 17. **pop()** 18. **push()** 19. **reduce()** 20. **reduceRight()** 21. **reverse()** 22. **shift()** 23. **slice()** 24. **some()** 25. **sort()** 26. **splice()** 27. **toString()** 28. **unshift()** 29. **valueOf()** 30. **Array.concat() -** Returns a new array comprised of this array joined with other array(s) and/or value(s).   **Example:**  **let arr1 = [1, 2, 3];**  **let arr2 = [4, 5, 6];**  **let arr3 = arr1.concat(arr2);**  **console.log(arr3); // Output: [1, 2, 3, 4, 5, 6]**   1. **Array.copyWithin() -** Copies a sequence of array elements within the array.   **Example:**  **let arr = [1, 2, 3, 4, 5];**  **arr.copyWithin(2, 0);// 2 is target index and 0 is start index**  **console.log(arr); // Output: [1, 2, 1, 2, 3]**   1. **Array.entries() -** Returns a new Array Iterator object that contains the key/value pairs for each index in the array.   **Example:**  **let arr = ["a", "b", "c"];**  **let iterator = arr.entries();**  **console.log(iterator.next().value); // Output: [0, "a"]**   1. **Array.every() -** Checks if every element in an array pass a test.   **Example:**  **function isPositive(num) {**  **return num >= 0;**  **}**  **let arr = [1, 2, 3, 4, 5];**  **let result = arr.every(isPositive);**  **console.log(result); // Output: true**   1. **Array.fill() -** Fills all the elements of an array from a start index to an end index with a static value.   **Example:**  **let arr = [1, 2, 3, 4, 5];**  **arr.fill(0, 2, 4);**  **console.log(arr); // Output: [1, 2, 0, 0, 5]**   1. **Array.filter() -** Creates a new array with every element in an array that pass a test.   **Example:**  **function isEven(num) {**  **return num % 2 === 0;**  **}**  **let arr = [1, 2, 3, 4, 5];**  **let result = arr.filter(isEven);**  **console.log(result); // Output: [2, 4]**   1. **Array..ind() -** Returns the value of the first element in an array that pass a test.   **Example:**  **function isBigEnough(num) {**  **return num >= 10;**  **}**  **let arr = [12, 5, 8, 130, 44];**  **let result = arr.find(isBigEnough);**  **console.log(result); // Output: 12**   1. **Array.findIndex() -** Returns the index of the first element in an array that pass a test.   **Example:**  **function isBigEnough(num) {**  **return num >= 10;**  **}**  **let arr = [12, 5, 8, 130, 44];**  **let result = arr.findIndex(isBigEnough);**  **console.log(result); // Output: 0**   1. **Array..flat() -** Creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.   **Example:**  **let arr = [1, 2, [3, 4, [5, 6]]];**  **let result = arr.flat(2);**  **console.log(result); // Output: [1, 2, 3, 4, 5, 6]**   1. **Array.flatMap() -** Maps each element using a mapping function, then flattens the result into a new array.   **Example:**  **let arr = [1, 2, 3, 4];**  **let result = arr.flatMap(x => [x \* 2]);**  **console.log(result); // Output: [2, 4, 6, 8]**   1. **Array.forEach() -** Calls a function for each element in the array.   **Example:**  **let arr = ["a", "b", "c"];**  **arr.forEach(function(element) {**  **console.log(element);**  **});**  **// Output:**  **// a**  **// b**  **// c**   1. **Array.includes() -** Checks if an array contains the specified element.   **Example:**  **let arr = [1, 2, 3];**  **let result = arr.includes(2);**  **console.log(result); // Output: true**   1. **Array.indexOf() -** Searches the array for the specified item, and returns its position.   **Example:**  **let arr = [1, 2, 3];**  **let result = arr.indexOf(2);**  **console.log(result); // Output: 1**   1. **Array.join() -** Joins all elements of an array into a string.   **Example:**  **let arr = [1, 2, 3];**  **let result = arr.join();**  **console.log(result); // Output: 1,2,3**   1. **Array.keys() -** Returns a new Array Iterator object that contains the keys for each index in the array.   **Example:**  **let arr = ["a", "b", "c"];**  **let iterator = arr.keys();**  **console.log(iterator.next().value); // Output: 0**   1. **Array.lastIndexOf() -** Searches the array for the specified item, starting at the end, and returns its position.   **Example:**  **let arr = [1, 2, 3, 2];**  **let result = arr.lastIndexOf(2);**  **console.log(result); // Output: 3**   1. **Array..map() -** Creates a new array with the results of calling a function for each array element.   **Example:**  **let arr = [1, 2, 3];**  **let result = arr.map(x => x \* 2);**  **console.log(result); // Output: [2, 4, 6]**   1. **Array.pop() -** Removes the last element of an array, and returns that element.   **Example:**  **let arr = [1, 2, 3];**  **let result = arr.pop();**  **console.log(arr); // Output: [1, 2]**   1. **Array.push() -** Adds one or more elements to the end of an array and returns the new length of the array.   **Example:**  **let arr = [1, 2, 3];**  **let result = arr.push(4);**  **console.log(result); // Output: 4**   1. **Array.reduce()** - Reduce the values of an array to a single value (going left-to-right).   **Example:**  **let arr = [1, 2, 3, 4];**  **let result = arr.reduce((acc, curr) => acc + curr);**  **console.log(result); // Output: 10**   1. **Array.reduceRight() -** Reduce the values of an array to a single value (going right-to-left).   **Example:**  **let arr = [1, 2, 3, 4];**  **let result = arr.reduceRight((acc, curr) => acc + curr);**  **console.log(result); // Output: 10**   1. **Array.reverse() -** Reverses the order of the elements in an array.   **Example:**  **let arr = [1, 2, 3];**  **let result = arr.reverse();**  **console.log(result); // Output: [3, 2, 1]**   1. **Array.shift() -** Removes the first element of an array, and returns that element.   **Example:**  **let arr = [1, 2, 3];**  **let result = arr.shift();**  **console.log(arr); // Output: [2, 3]**   1. **Array.slice() -** Selects a part of an array, and returns the new array.   **Example:**  **let arr = [1, 2, 3, 4, 5];**  **let result = arr.slice(2, 4);**  **console.log(result); // Output: [3, 4]**   1. **Array.some() -** Checks if any of the elements in an array pass a test.   **Example:**  **function isBigEnough(num) {**  **return num >= 10;**  **}**  **let arr = [12, 5, 8, 130, 44];**  **let result = arr.some(isBigEnough);**  **console.log(result); // Output: true**   1. **Array.sort() -** Sorts the elements of an array.   **Example:**  **let arr = [3, 1, 2];**  **let result = arr.sort();**  **console.log(result); // Output: [1, 2, 3]**   1. **Array..splice() -** Adds/Removes elements from an array.   **Example:**  **let arr = [1, 2, 3, 4, 5];**  **let result = arr.splice(2, 2);**  **console.log(result); // Output: [3, 4]**   1. **Array.toString() -** Converts an array to a string, and returns the result.   **Example:**  **let arr = [1, 2, 3];**  **let result = arr.toString();**  **console.log(result); // Output: 1,2,3**  **String Methods:** |

**1. charAt()**

**2. charCodeAt()**

**3. concat()**

**4. indexOf()**

**5. lastIndexOf()**

**6. match()**

**7. replace()**

**8. search()**

**9. slice()**

**10. split()**

**11. substr()**

**12. substring()**

**13. toLowerCase()**

**14. toUpperCase()**

**15. trim()**

**16. startsWith()**

**17. endsWith()**

**18. includes()**

**19. repeat()**

**20. toLocaleLowerCase()**

**21. toLocaleUpperCase()**

**22. localeCompare()**

**23. normalize()**

**24. padEnd()**

**25. padStart()**

**26. trimLeft()**

**27. trimRight()**

**28. valueOf()**

**1. charAt() :** The charAt() method returns the character at the specified index in a string.

**Syntax:**

**string.charAt(index)**

**Example:**

**const str = 'JavaScript';**

**console.log(str.charAt(0));**

**Output: 'J'**

**2. charCodeAt() :**The charCodeAt() method returns the Unicode of the character at the specified index in a string.

**Syntax:**

**string.charCodeAt(index)**

**Example:**

**const str = 'JavaScript';**

**console.log(str.charCodeAt(0));**

**Output: 74**

**3. concat() :**The concat() method is used to join two or more strings.

**Syntax:**

**string.concat(string2, string3, ..., stringN)**

**Example:**

**const str1 = 'Hello';**

**const str2 = 'World';**

**console.log(str1.concat(' ', str2));**

**Output: 'Hello World'**

**4. indexOf() :**The indexOf() method returns the position of the first occurrence of a specified value in a string.

**Syntax:**

**string.indexOf(searchValue, start)**

**Example:**

**const str = 'Hello World';**

**console.log(str.indexOf('World'));**

**Output: 6**

**5. lastIndexOf() :**The lastIndexOf() method returns the position of the last occurrence of a specified value in a string.

**Syntax:**

**string.lastIndexOf(searchValue, start)**

**Example:**

**const str = 'Hello World, Hello World';**

**console.log(str.lastIndexOf('World'));**

**Output: 14**

**6. match() :**The match() method searches a string for a match against a regular expression, and returns the matches, as an Array object.

**Syntax:**

**string.match(regexp)**

**Example:**

**const str = 'Hello World';**

**console.log(str.match(/Hello/));**

**Output: [ 'Hello' ]**

**7. replace() :**The replace() method searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced.

**Syntax:**

**string.replace(searchValue, newValue)**

**Example:**

**const str = 'Hello World';**

**console.log(str.replace('Hello', 'Hi'));**

**Output: 'Hi World'**

**8. search() :**The search() method searches a string for a specified value, and returns the position of the match.

**Syntax:**

**string.search(regexp)**

**Example:**

**const str = 'Hello World';**

**console.log(str.search('World'));**

**Output: 6**

**9. slice() :**The slice() method extracts a part of a string and returns the extracted part in a new string.

**Syntax:**

**string.slice(start, end)**

**Example:**

**const str = 'Hello World';**

**console.log(str.slice(6, 11));**

**Output: 'World'**

**10. split() :**The split() method splits a String object into an array of strings by separating the string into substrings.

**Syntax:**

**string.split(separator, limit)**

**Example:**

**const str = 'Hello World';**

**console.log(str.split(' '));**

**Output: [ 'Hello', 'World' ]**

**11. substr() :**The substr() method returns the characters in a string beginning at the specified location through the specified number of characters.

**Syntax:**

**string.substr(start, length)**

**Example:**

**const str = 'Hello World';**

**console.log(str.substr(6, 5));**

**Output: 'World'**

**12. substring() :**The substring() method returns the characters in a string between two specified indices.

**Syntax:**

**string.substring(start, end)**

**Example:**

**const str = 'Hello World';**

**console.log(str.substring(6, 11));**

**Output: 'World'**

**13. toLowerCase() :**The toLowerCase() method converts a string to lowercase letters.

**Syntax:**

**string.toLowerCase()**

**Example:**

**const str = 'Hello World';**

**console.log(str.toLowerCase());**

**Output: 'hello world'**

**14. toUpperCase() :**The toUpperCase() method converts a string to uppercase letters.

**Syntax:**

**string.toUpperCase()**

**Example:**

**const str = 'Hello World';**

**console.log(str.toUpperCase());**

**Output: 'HELLO WORLD'**

**15. trim() :**The trim() method removes whitespace from both ends of a string.

**Syntax:**

**string.trim()**

**Example:**

**const str = ' Hello World ';**

**console.log(str.trim());**

**Output: 'Hello World'**

**16. startsWith() :**The startsWith() method determines whether a string begins with the characters of a specified string.

**Syntax:**

**string.startsWith(searchString, position)**

**Example:**

**const str = 'Hello World';**

**console.log(str.startsWith('Hello'));**

**Output: true**

**17. endsWith() :**The endsWith() method determines whether a string ends with the characters of a specified string.

**Syntax:**

**string.endsWith(searchString, length)**

**Example:**

**const str = 'Hello World';**

**console.log(str.endsWith('World'));**

**Output: true**

**18. includes() :**The includes() method determines whether a string contains the characters of a specified string.

**Syntax:**

**string.includes(searchString, position)**

**Example:**

**const str = 'Hello World';**

**console.log(str.includes('llo'));**

**Output: true**

**19. repeat() :**The repeat() method returns a new string with a specified number of copies of the string it was called on.

**Syntax:**

**string.repeat(count)**

**Example:**

**const str = 'Hello ';**

**console.log(str.repeat(3));**

**Output: 'Hello Hello Hello '**

**20. toLocaleLowerCase() :**The toLocaleLowerCase() method returns the value of the string converted to lower case according to any locale-specific case mappings.

**Syntax:**

**string.toLocaleLowerCase(locale)**

**Example:**

**const str = 'Hello World';**

**console.log(str.toLocaleLowerCase());**

**Output: 'hello world'**

**21. toLocaleUpperCase() :**The toLocaleUpperCase() method returns the value of the string converted to upper case according to any locale-specific case mappings.

**Syntax:**

**string.toLocaleUpperCase(locale)**

**Example:**

**const str = 'Hello World';**

**console.log(str.toLocaleUpperCase());**

**Output: 'HELLO WORLD'**

**22. localeCompare() :**The localeCompare() method returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.

**Syntax:**

**string.localeCompare(compareString, locale)**

**Example:**

**const str1 = 'a';**

**const str2 = 'b';**

**console.log(str1.localeCompare(str2));**

**Output: -1**

**23. normalize() :**The normalize() method returns the Unicode Normalization Form of a given string.

**Syntax:**

**string.normalize(form)**

**Example:**

**const str = '\u01c4';**

**console.log(str.normalize('NFC'));**

**Output: 'Ǆ'**

**24. padEnd() :**The padEnd() method pads the current string with a given string (repeated, if needed) so that the resulting string reaches a given length.

**Syntax:**

**string.padEnd(targetLength, padString)**

**Example:**

**const str = 'Hello';**

**console.log(str.padEnd(10, ' World'));**

**Output: 'Hello World'**

**25. padStart() :**The padStart() method pads the current string with a given string (repeated, if needed) so that the resulting string reaches a given length.

**Syntax:**

**string.padStart(targetLength, padString)**

**Example:**

**const str = 'World';**

**console.log(str.padStart(10, 'Hello '))**

**Output: 'Hello World'**

**26. trimLeft() :**The trimLeft() method removes whitespace from the left end of a string.

**Syntax:**

**string.trimLeft()**

**Example:**

**const str = ' Hello World ';**

**console.log(str.trimLeft());**

**Output: 'Hello World '**

**27. trimRight() :**The trimRight() method removes whitespace from the right end of a string**.**

**Syntax:**

**string.trimRight()**

**Example:**

**const str = ' Hello World ';**

**console.log(str.trimRight());**

**Output: ' Hello World'**

**28. valueOf() :**The valueOf() method returns the primitive value of a String object.

**Syntax:**

**string.valueOf()**

**Example:**

**const str = 'Hello World';**

**console.log(str.valueOf());**

**Output: 'Hello World'**

**Math Methods:**

Math Methods in JavaScript are pre-defined methods which allow us to perform mathematical operations and calculations. Some of the most commonly used Math methods in JavaScript are:

**1. Math.abs() :** This method returns the absolute value of a number.

**Example:**

**let number = -5;**

**console.log(Math.abs(number));**

**Output: 5**

**2. Math.ceil() :** This method returns the smallest integer greater than or equal to a given number.

**Example:**

**let number = 5.4;**

**console.log(Math.ceil(number));**

**Output: 6**

**3. Math.floor() :** This method returns the largest integer less than or equal to a given number.

**Example:**

**let number = 5.4;**

**console.log(Math.floor(number));**

**Output: 5**

**4. Math.max()** : This method returns the largest of zero or more numbers.

**Example:**

**let numbers = [5, 10, 20, -1];**

**console.log(Math.max(...numbers));**

**Output: 20**

**5. Math.min() :** This method returns the smallest of zero or more numbers.

**Example:**

**let numbers = [5, 10, 20, -1];**

**console.log(Math.min(...numbers));**

**Output: -1**

**6. Math.random() :** This method returns a random number between 0 (inclusive) and 1 (exclusive).

**Example:**

**console.log(Math.random());**

**Output: 0.6336849379852232**

**7. Math.round() :** This method rounds a number to the nearest integer.

**Example:**

**let number = 5.4;**

**console.log(Math.round(number));**

**Output: 5**

**8. Math.sqrt() :** This method returns the square root of a number.

**Example:**

**let number = 25;**

**console.log(Math.sqrt(number));**

**Output: 5**

**9. Math.pow() :** This method returns the value of a number raised to a given power.

**Example:**

**let number = 5;**

**let power = 3;**

**console.log(Math.pow(number, power));**

**Output: 125**

**10. Math.trunc() :** This method removes the fractional part of a number and returns the integer part.

**Example:**

**let number = 5.4;**

**console.log(Math.trunc(number));**

**Output: 5**

**11. Math.imul()** is a built-in function in JavaScript used to return the result of a 32-bit integer multiplication. It is used to calculate the product of two 32-bit integers.

**Example 1:**

**let x = 5;**

**let y = 4;**

**let z = Math.imul(x, y);**

**console.log(z);**

**Output:20**

**Date Methods:**

**1. getDate() :**Returns the day of the month (1-31)

**Example:**

**let d = new Date();**

**let n = d.getDate();**

**console.log(n);**

**Output: 12**

**2. getDay():** Returns the day of the week (0-6)

**Example:**

**let d = new Date();**

**let n = d.getDay();**

**console.log(n);**

**Output: 5**

**3. getFullYear():** Returns the year (4 digits)

**Example:**

**let d = new Date();**

**let n = d.getFullYear();**

**console.log(n);**

**Output: 2020**

**4. getMonth():** Returns the month (0-11)

**Example:**

**let d = new Date();**

**let n = d.getMonth();**

**console.log(n);**

**Output: 7**

**5. getTime():**Returns the time for the date as the number of milliseconds since January 1, 1970

**Example:**

**let d = new Date();**

**let n = d.getTime();**

**console.log(n);**

**Output: 1597118080577**

**6. getHours():** Returns the hour (0-23)

**Example:**

**let d = new Date();**

**let n = d.getHours();**

**console.log(n);**

**Output: 16**

**7. getMinutes():**Returns the minute (0-59)

**Example:**

**let d = new Date();**

**let n = d.getMinutes();**

**console.log(n);**

**Output: 31**

**8. getSeconds():** Returns the second (0-59)

**Example:**

**let d = new Date();**

**let n = d.getSeconds();**

**console.log(n);**

**Output: 6**

**9. toDateString():** Returns the date portion of a Date object as a human-readable string

**Example:**

**let d = new Date();**

**let n = d.toDateString();**

**console.log(n);**

**Output: Sun Aug 16 2020**

**10. toTimeString():**  Returns the time portion of a Date object as a human-readable string

**Example:**

**let d = new Date();**

**let n = d.toTimeString();**

**console.log(n);**

**Output: 19:18:58 GMT+0100 (British Summer Time)**

**The bind(), call(), and apply()** methods are used to bind the this value of a function to the first argument of the methods.

**The bind()** method creates a new function that, when called, has its this keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

**The call()** method calls a function with a given this value and arguments provided individually.

**The apply()** method calls a function with a given this value and arguments provided as an array.

The difference between bind(), call(), and apply() is that bind() creates a new function, call() calls the function and apply() calls the function with an array of arguments.

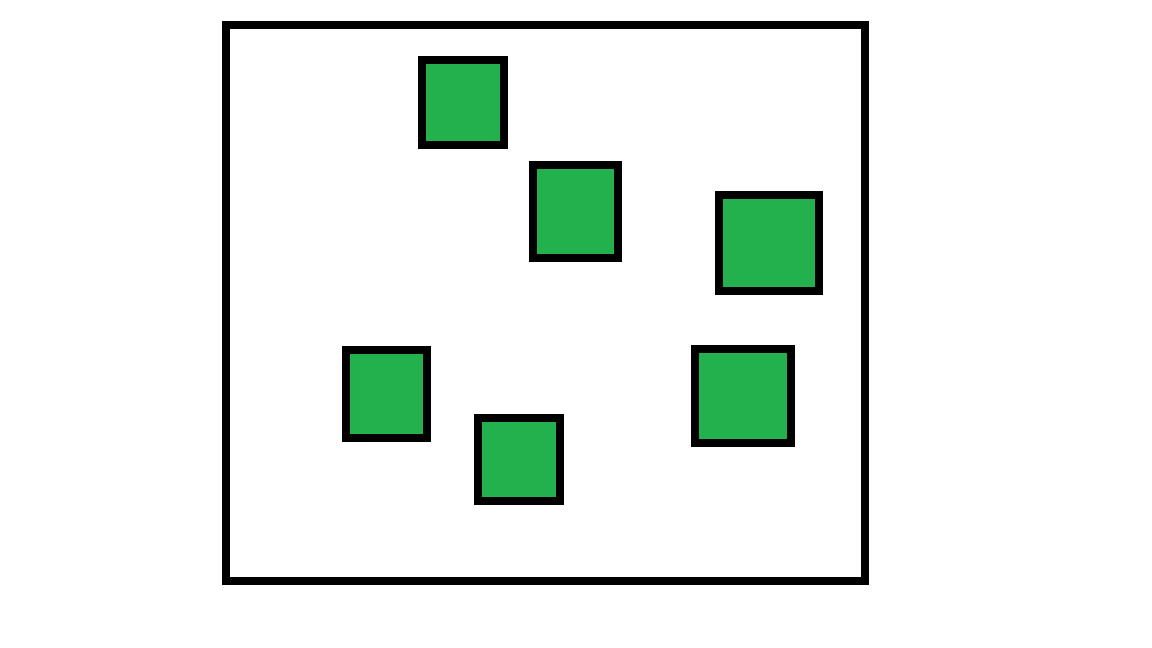
**Promises:**

Promises are used to handle asynchronous operations in JavaScript. They are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.

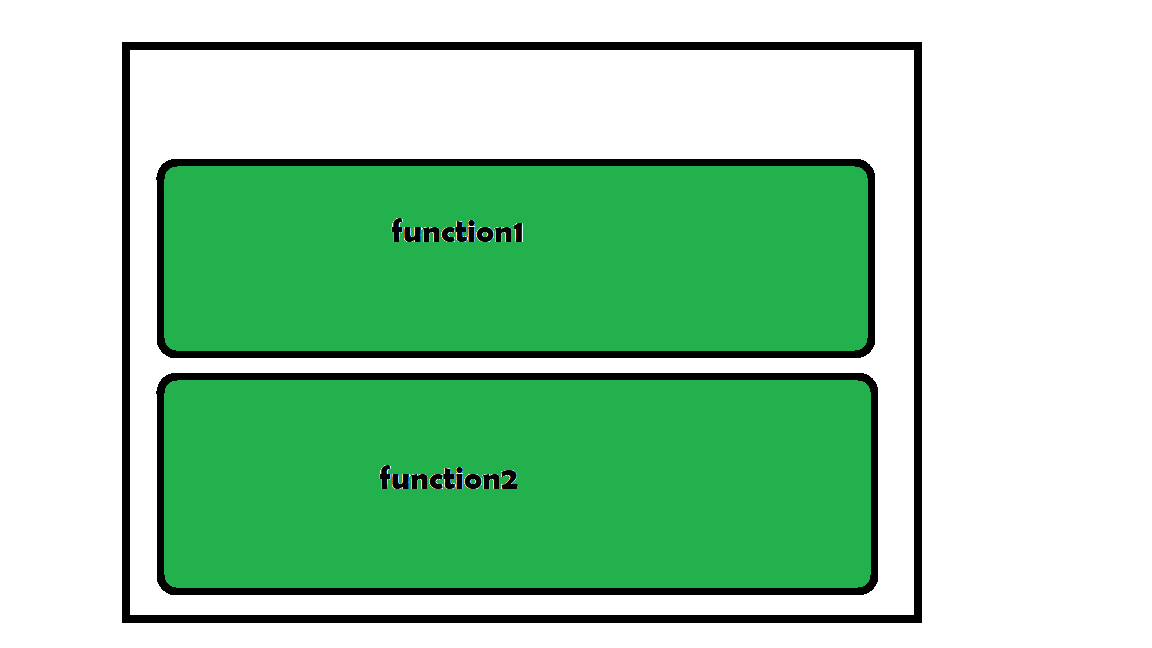
The await operator is used to wait for a Promise. It can only be used inside an async function.

**Memory allocation in JavaScript:**

**1) Heap memory:** Data stored randomly and memory allocated.



**2) Stack memory:** Memory allocated in the form of stacks. Mainly used for functions.

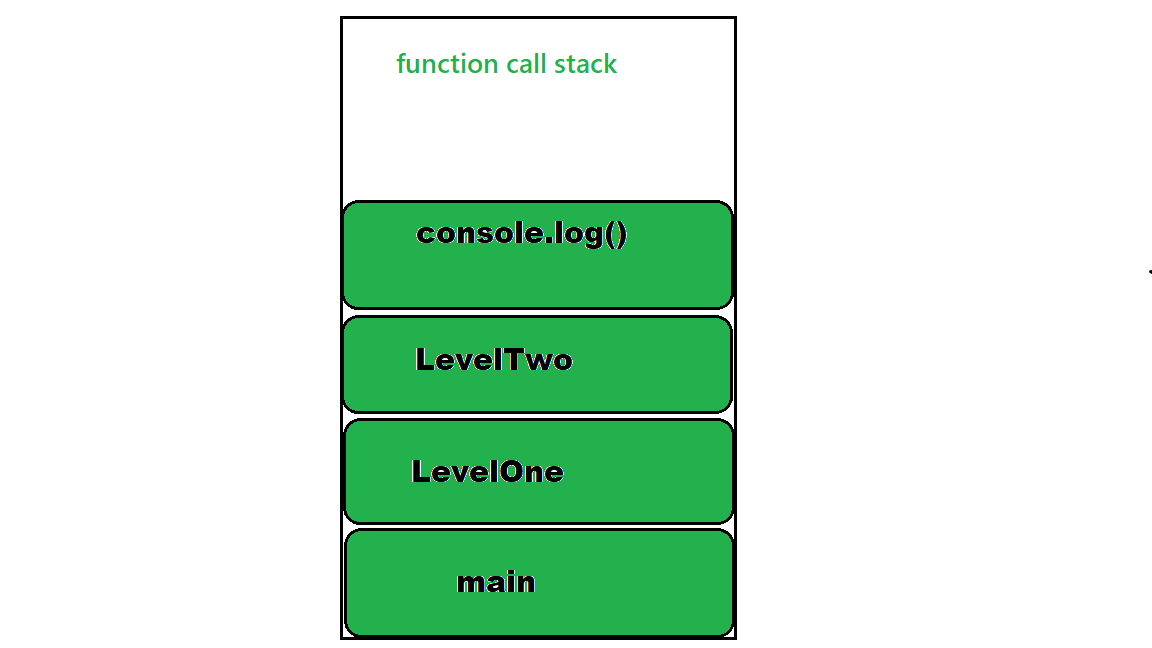


**Function call stack:** The function stack is a function which keeps track of all other functions executed in run time. Ever seen a stack trace being printed when you ran into an error in JavaScript. That is nothing but a snapshot of the function stack at that point when the error occurred.

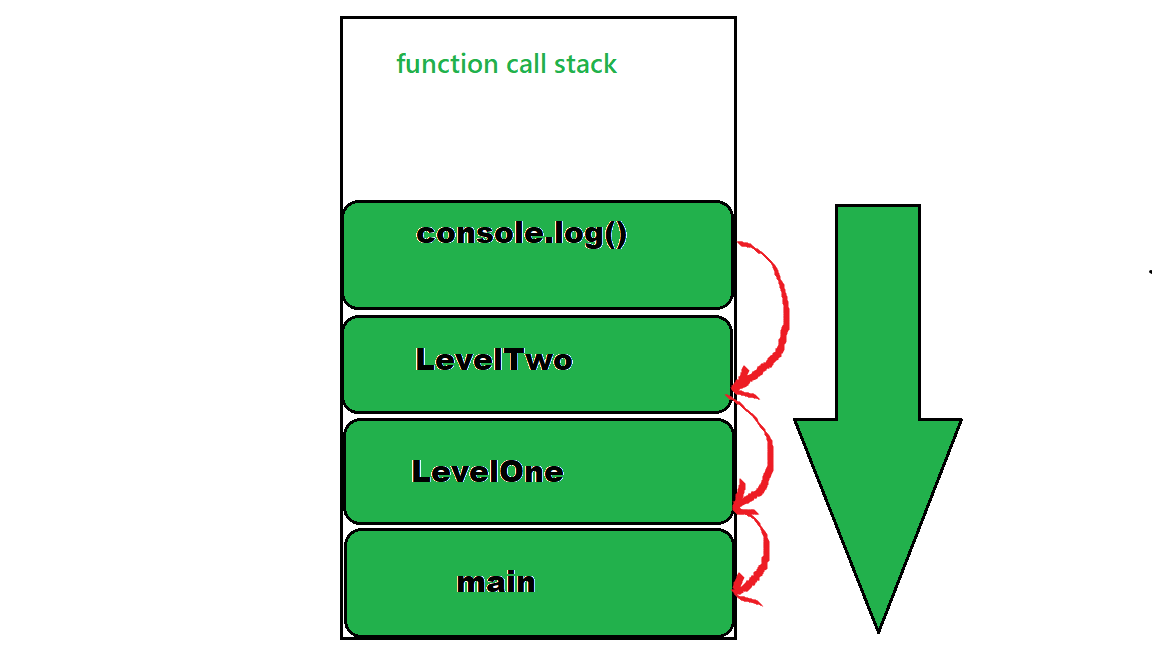
Example:

* Javascript

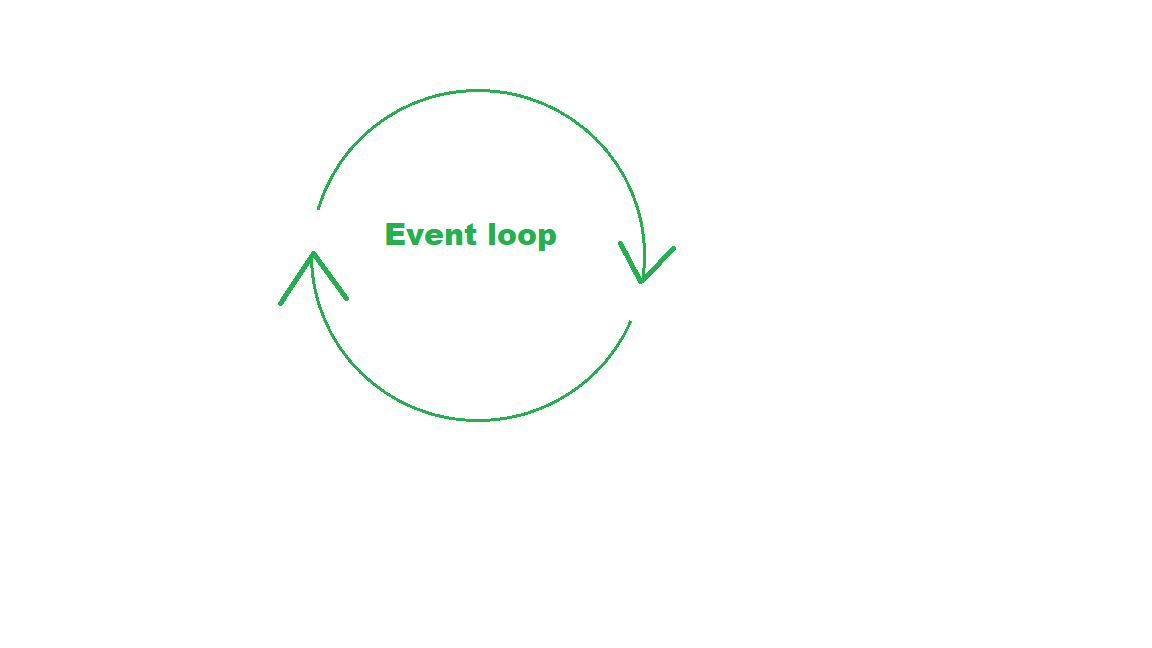
|  |
| --- |
| function LevelTwo() {     console.log("Inside Level Two!")  }    function LevelOne() {     LevelTwo()  }    function main() {     LevelOne()  }    main() |



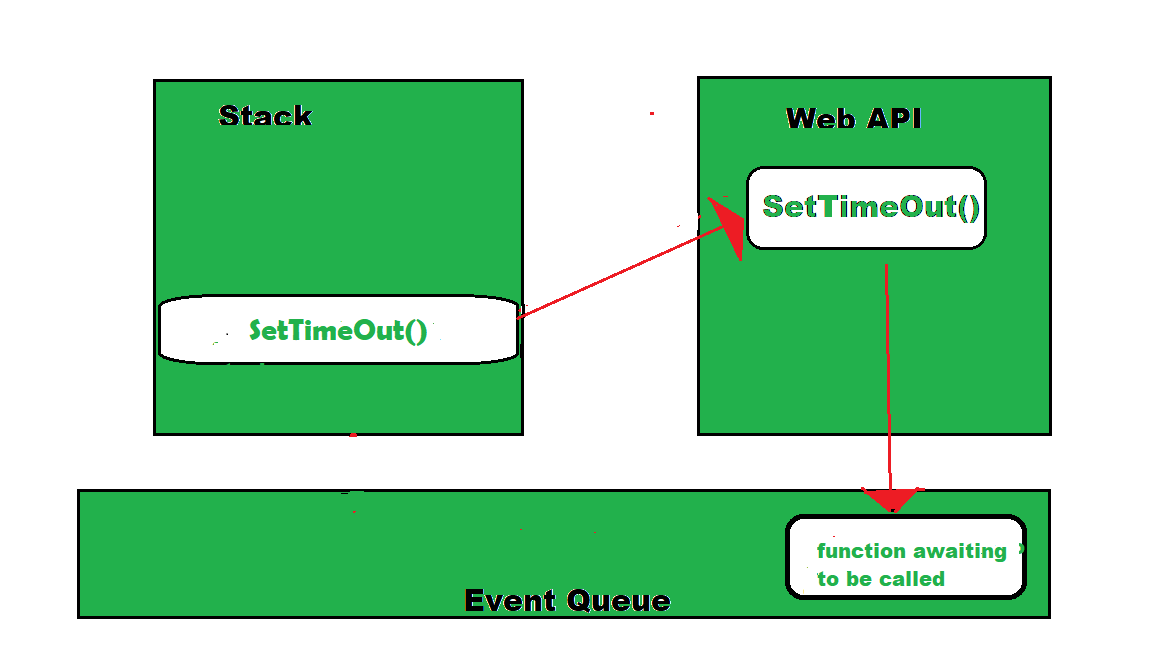
Order at which functions get executed i.e get popped out of the stack after a function’s purpose gets over as shown below:



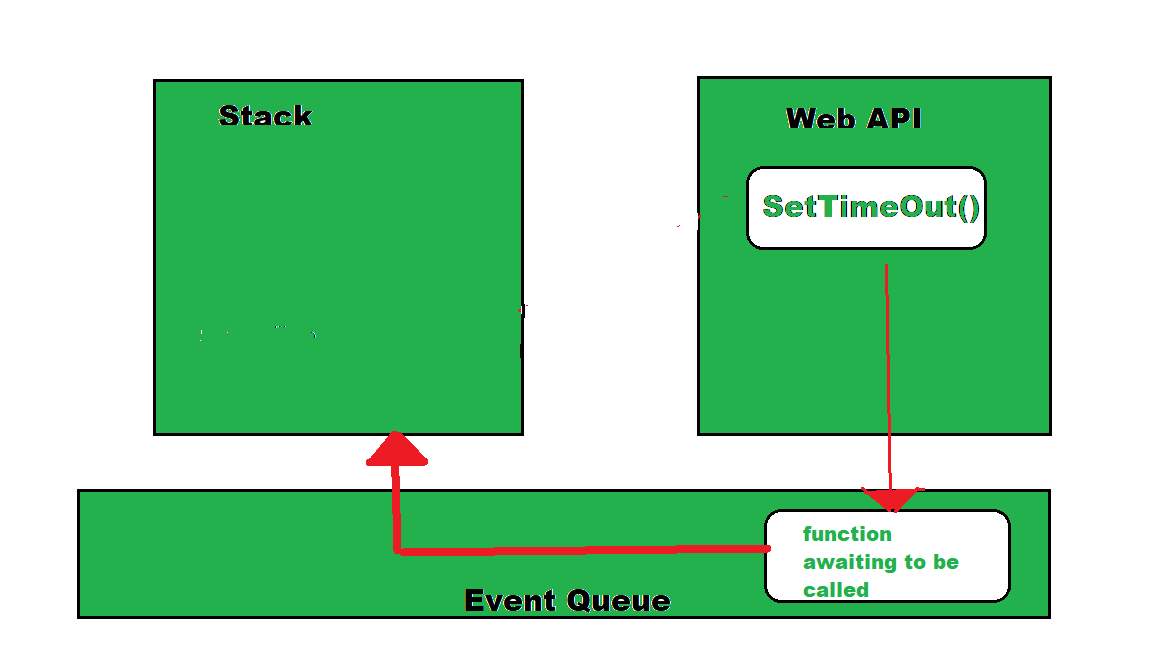
**Event loop:**An event loop is something that pulls stuff out of the queue and places it onto the function execution stack whenever the function stack becomes empty.



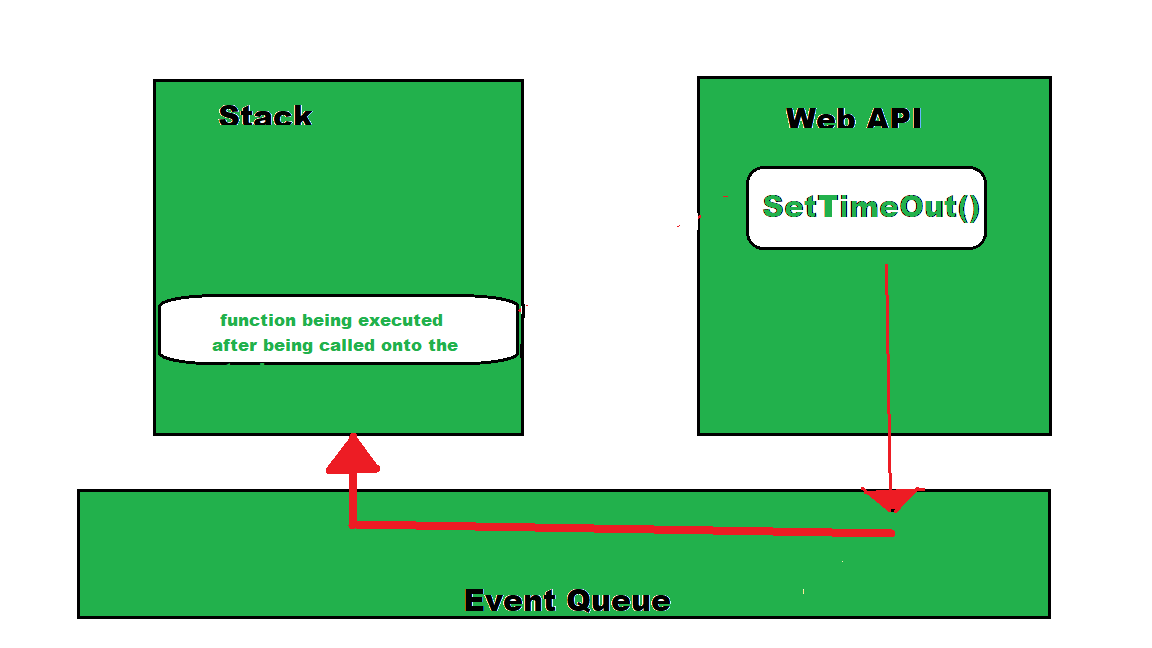
The event loop is the secret by which JavaScript gives us an illusion of being multithreaded even though it is single-threaded. The below illusion demonstrates the functioning of event loop well:



Here the callback function in the event queue has not yet run and is waiting for its time into the stack when the SetTimeOut() is being executed and the Web API is making the mentioned wait. When the function stack becomes empty, the function gets loaded onto the stack as shown below:



That is where the event loop comes into picture, it takes the first event from the Event Queue and places it onto the stack i.e in this case the callback function. From here, this function executes calling other functions inside it, if any.



This cycle is called the **event loop** and this how JavaScript manages its events.