JavaScript Important Interview Questions

**JavaScript Link**

# <https://www.simplilearn.com/tutorials/javascript-tutorial/javascript-interview-questions>

<https://www.javatpoint.com/javascript-interview-questions>

### **1) What is JavaScript?**

**JavaScript** is a scripting language. It is different from Java language. It is object-based, lightweight, cross-platform translated language. It is widely used for client-side validation. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

**JavaScript** was developed by **Netscape** in 1995 as a scripting programming language for web browser only. **JavaScript** is used to build **interactive websites** with **dynamic** features and to **validate form data**. JavaScript is **high-level** and **dynamic** programming language with first class functions, supported by all modern web browsers. Apart from web browser, JavaScript is also used to build scalable web applications using [Node JS](https://tutorial.techaltum.com/nodejs.html) / Deno / Bun.

**JavaScript** is also known as the **Programming Language of Web** as it is the only programming language for Web browsers. 100% websites are using JavaScript Programming language for frontend.

JS is **single thread** with support of asynchronous events and callbacks functions to improve performance. Unlike Other scripting languages, it is very fast as modern web browsers use *Just in Time* compilers, although earlier till 2009, JavaScript was interpreted in browsers.

### **2) List some features of JavaScript.**

Some of the features of JavaScript are:

* Lightweight
* Interpreted programming language
* Good for the applications which are network-centric
* Complementary to Java
* Complementary to HTML
* Open source
* Cross-platform

### **3) Who developed JavaScript, and what was the first name of JavaScript?**

JavaScript was developed by Brendan Eich, who was a Netscape programmer. Brendan Eich developed this new scripting language in just ten days in the year September 1995. At the time of its launch, JavaScript was initially called Mocha. After that, it was called Live Script and later known as JavaScript.

### **4) List some of the advantages of JavaScript.**

Some of the advantages of JavaScript are:

* Server interaction is less
* Feedback to the visitors is immediate
* Interactivity is high
* Interfaces are richer

### **5) List some of the disadvantages of JavaScript.**

Some of the disadvantages of JavaScript are:

* No support for multithreading
* No support for multiprocessing
* Reading and writing of files is not allowed
* No support for networking applications.

### **6) Define a named function in JavaScript.**

The function which has named at the time of definition is called a named function. For example

1. function msg()
2. {
3. document.writeln("Named Function");
4. }
5. msg();

### **7) Name the types of functions**

The types of function are:

* **Named** - These type of functions contains name at the time of definition. For Example:
  1. function display()
  2. {
  3. document.writeln("Named Function");
  4. }
  5. display();
* **Anonymous** - These type of functions doesn't contain any name. They are declared dynamically at runtime.
  1. var display=function()
  2. {
  3. document.writeln("Anonymous Function");
  4. }
  5. display();

### **8) Define anonymous function**

It is a function that has no name. These functions are declared dynamically at runtime using the function operator instead of the function declaration. The function operator is more flexible than a function declaration. It can be easily used in the place of an expression. For example:

1. var display=function()
2. {
3. alert("Anonymous Function is invoked");
4. }
5. display();

### **9) Can an anonymous function be assigned to a variable?**

Yes, you can assign an anonymous function to a variable.

### **10) In JavaScript what is an argument object?**

The variables of JavaScript represent the arguments that are passed to a function.

### **11) Define closure.**

In JavaScript, we need closures when a variable which is defined outside the scope in reference is accessed from some inner scope.

1. var num = 10;
2. function sum()
3. {
4. document.writeln(num+num);
5. }
6. sum();

### **12) If we want to return the character from a specific index which method is used?**

The JavaScript string charAt() method is used to find out a char value present at the specified index. The index number starts from 0 and goes to n-1, where n is the length of the string. The index value can't be a negative, greater than or equal to the length of the string. For example:

1. var str="Javatpoint";
2. document.writeln(str.charAt(4));

### **13) What is the difference between JavaScript and JScript?**

Netscape provided the JavaScript language. Microsoft changed the name and called it JScript to avoid the trademark issue. In other words, you can say JScript is the same as JavaScript, but Microsoft provides it.

### **14) How to write a hello world example of JavaScript?**

A simple example of JavaScript hello world is given below. You need to place it inside the body tag of HTML.

1. **<script** type="text/javascript"**>**
2. document.write("JavaScript Hello World!");
3. **</script>**

### **15) What are the key differences between Java and JavaScript? / How is JavaScript different from Java?**

JavaScript is a lightweight programming language (most commonly known as scripting language) developed by Netscape, Inc. It is used to make web pages interactive. It is not a part of the Java platform. Following is a list of some key differences between Java and JavaScript

**A list of key differences between Java and JavaScript**

|  |  |
| --- | --- |
| **Java** | **JavaScript** |
| Java is a complete and strongly typed programming language used for backend coding. In Java, variables must be declared first to use in the program, and the type of a variable is checked at compile-time. | JavaScript is a weakly typed, lightweight programming language (most commonly known as scripting language) and has more relaxed syntax and rules. |
| Java is an object-oriented programming (OOPS) language or structured programming languages such as C, C++, or .Net. | JavaScript is a client-side scripting language, and it doesn't fully support the OOPS concept. It resides inside the HTML documents and is used to make web pages interactive (not achievable with simple HTML). |
| Java creates applications that can run in any virtual machine (JVM) or browser. | JavaScript code can run only in the browser, but it can now run on the server via Node.js. |
| The Java code needs to be compiled. | The JavaScript code doesn't require to be complied. |
| Java Objects are class-based. You can't make any program in Java without creating a class. | JavaScript Objects are prototype-based. |
| Java is a Complete and Standalone language that can be used in backend coding. | JavaScript is assigned within a web page and integrates with its HTML content. |
| Java programs consume more memory. | JavaScript code is used in HTML web pages and requires less memory. |
| The file extension of the Java program is written as ".Java" and it translates source code into bytecodes which are then executed by JVM (Java Virtual Machine). | The JavaScript file extension is written as ".js" and it is interpreted but not compiled. Every browser has a JavaScript interpreter to execute the JS code. |
| Java supports multithreading. | JavaScript doesn't support multithreading. |
| Java uses a thread-based approach to concurrency. | JavaScript uses an event-based approach to concurrency. |

### **16) How to use external JavaScript file?**

I am assuming that js file name is message.js, place the following script tag inside the head tag.

1. **<script** type="text/javascript" src="message.js"**></script>**

### **17) Is JavaScript case sensitive language?**

Yes, JavaScript is a case sensitive language. For example:

1. Var msg = "JavaScript is a case-sensitive language"; //Here, var should be used to declare a variable
2. function display()
3. {
4. document.writeln(msg); // It will not display the result.
5. }
6. display();

### **18) What is BOM?**

**BOM** stands for Browser Object Model. It provides interaction with the browser. The default object of a browser is a window. So, you can call all the functions of the window by specifying the window or directly.

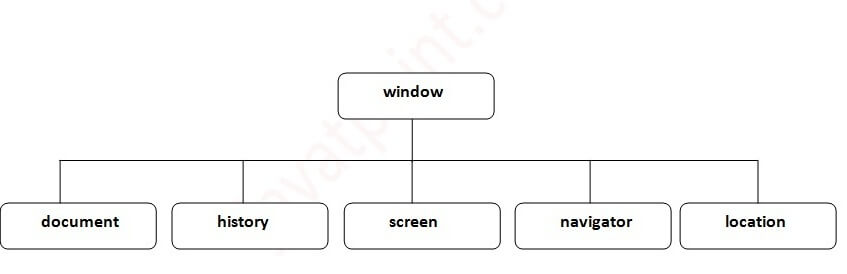
**For example:**

1. window.alert("hello javatpoint");

is same as:

1. alert("hello javatpoint");

The window object provides various properties like document, history, screen, navigator, location, innerHeight, innerWidth,



### **19) What is DOM? What is the use of document object?**

**DOM** stands for Document Object Model. A document object represents the HTML document. It can be used to access and change the content of HTML.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

As mentioned earlier, it is the object of window. So

1. window.document

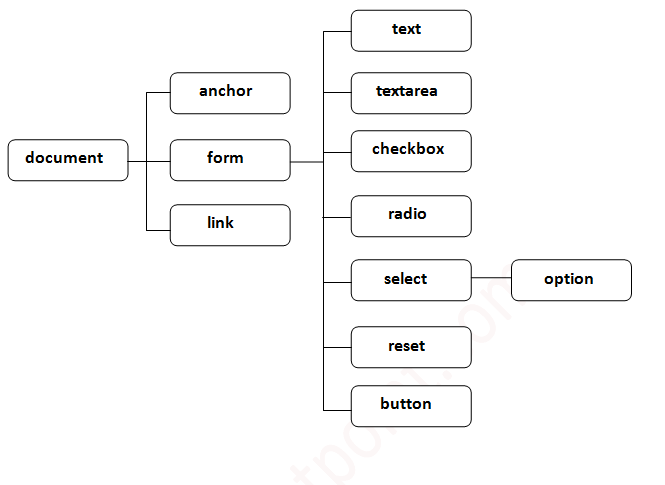
Is same as

1. document

According to W3C - *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

## **Properties of document object**

Let's see the properties of document object that can be accessed and modified by the document object.



## **Methods of document object**

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| write("string") | writes the given string on the doucment. |
| writeln("string") | writes the given string on the doucment with newline character at the end. |
| getElementById() | returns the element having the given id value. |
| getElementsByName() | returns all the elements having the given name value. |
| getElementsByTagName() | returns all the elements having the given tag name. |
| getElementsByClassName() | returns all the elements having the given class name. |

### **Accessing field value by document object**

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

Here, **document** is the root element that represents the html document.

**form1** is the name of the form.

**name** is the attribute name of the input text.

**value** is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

1. **<script** type="text/javascript"**>**
2. function printvalue(){
3. var name=document.form1.name.value;
4. alert("Welcome: "+name);
5. }
6. **</script>**
8. **<form** name="form1"**>**
9. Enter Name:**<input** type="text" name="name"**/>**
10. **<input** type="button" onclick="printvalue()" value="print name"**/>**
11. **</form>**

### **20) What is the use of window object?**

The window object is created automatically by the browser that represents a window of a browser. Window is the object of browser, **it is not the object of javascript**. The javascript objects are string, array, date etc.

The window object is used to display the popup dialog box. Let's see with description.

|  |  |
| --- | --- |
| **Method** | **Description** |
| alert() | displays the alert box containing the message with ok button. |
| confirm() | displays the confirm dialog box containing the message with ok and cancel button. |
| prompt() | displays a dialog box to get input from the user. |
| open() | opens the new window. |
| close() | closes the current window. |
| setTimeout() | performs the action after specified time like calling function, evaluating expressions. |

### **21) What is the use of history object?**

The history object of a browser can be used to switch to history pages such as back and forward from the current page or another page. There are three methods of history object.

1. history.back() - It loads the previous page.
2. history.forward() - It loads the next page.
3. history.go(number) - The number may be positive for forward, negative for backward. It loads the given page number.

### **22) How to write a comment in JavaScript?**

There are two types of comments in JavaScript.

1. Single Line Comment: It is represented by // (double forward slash)
2. Multi-Line Comment: Slash represents it with asterisk symbol as /\* write comment here \*/

### **23) How to create a function in JavaScript?**

**JavaScript functions** are used to perform operations. We can call JavaScript function many times to reuse the code.

#### **Advantage of JavaScript function**

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

To create a function in JavaScript, follow the following syntax.

1. function function\_name(){
2. //function body
3. }

### **24) What are the different data types present in JavaScript?**

There are two types of data types in JavaScript:

* Primitive data types
* Non- Primitive data types

## **JavaScript primitive data types**

There are five types of primitive data types in JavaScript. They are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| String | represents sequence of characters e.g. "hello" |
| Number | represents numeric values e.g. 100 |
| Boolean | represents boolean value either false or true |
| Undefined | represents undefined value |
| Null | represents null i.e. no value at all |

The primitive data types are as follows:

**String:** The string data type represents a sequence of characters. It is written within quotes and can be represented using a single or a double quote.

**Example:**

1. var str1 = "Hello JavaTpoint"; //using double quotes
2. var str2 = 'Hello Javatpoint'; //using single quotes

**Number:** The number data type is used to represent numeric values and can be written with or without decimals. JavaScript integers are only accurate up to 15 digits:

**Example:**

1. var x = 5; //without decimal
2. var y = 5.0; //with decimal

**Boolean:** The Boolean data type is used to represent a Boolean value, either false or true. This data type is generally used for conditional testing.

**Example:**

1. var x = 5;
2. var y =  6;
3. var z =  5;
4. (x == y) // returns false
5. (x == z) //returns true

**BigInt:** The BigInt data type is used to store numbers beyond the Number data type limitation. This data type can store large integers and is represented by adding "n" to an integer literal.

**Example:**

1. var bigInteger =  123456789012345678901234567890n;
2. // This is an example of bigInteger.

**Undefined:** The Undefined data type is used when a variable is declared but not assigned. The value of this data type is undefined, and its type is also undefined.

**Example:**

1. var x; // value of x is undefined
2. var y = undefined; // You can also set the value of a variable as undefined.

**Null:** The Null data type is used to represent a non-existent, null, or an invalid value i.e. no value at all.

**Example:**

1. var  x = null;

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

**Example:**

1. var symbol1 = Symbol('symbol');

**typeof:** The typeof operator is used to determine what type of data a variable or operand contains. It can be used with or without parentheses (typeof(x) or typeof x). This is mainly used in situations when you need to process the values of different types.

**Example:**

1. typeof 10;  // Returns: "number"
2. typeof 10.0;  // Returns: "number"
3. typeof 2.5e-4;  // Returns: "number"
4. typeof Infinity;  // Returns: "number"
5. typeof NaN;  // Returns: "number". Despite being "Not-A-Number"

// Strings

1. typeof '';  // Returns: "string"
2. typeof 'Welcome to JavaTpoint';  // Returns: "string"
3. typeof '12';  // Returns: "string". Number within quotes is typeof string

// Booleans

1. typeof true;  // Returns: "boolean"
2. typeof false;  // Returns: "boolean"

// Undefined

1. typeof undefined;  // Returns: "undefined"
2. typeof undeclaredVariable; // Returns: "undefined"

// Null

1. typeof Null;  // Returns: "object"

// Objects

1. typeof {name: "John", age: 18};  // Returns: "object"

// Arrays

1. typeof [1, 2, 3];  // Returns: "object"

// Functions

1. typeof function(){};  // Returns: "function"

**Non-Primitive data types**

In the above examples, we can see that the primitive data types can store only a single value. To store multiple and complex values, we have to use non-primitive data types.

The non-primitive data types are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| Object | represents instance through which we can access members |
| Array | represents group of similar values |
| RegExp | represents regular expression |

**Object:** The Object is a non-primitive data type. It is used to store collections of data. An object contains properties, defined as a key-value pair. A property key (name) is always a string, but the value can be any data type, such as strings, numbers, Booleans, or complex data types like arrays, functions, and other objects.

**Example:**

// Collection of data in key-value pairs

1. var obj1 = {
2. x:  123,
3. y:  "Welcome to JavaTpoint",
4. z: function(){
5. return this.x;
6. }
7. }

**Array:** The Array data type is used to represent a group of similar values. Every value in an array has a numeric position, called its index, and it may contain data of any data type - numbers, strings, Booleans, functions, objects, and even other arrays. The array index starts from 0 so that the first array element is arr[0], not arr[1].

**Example:**

1. var colors = ["Red", "Yellow", "Green", "Orange"];
2. var cities = ["Noida", "Delhi", "Ghaziabad"];
3. alert(colors[2]);   // Output: Green
4. alert(cities[1]);   // Output: Delhi

### **25) What is the difference between == and ===?**

The == operator checks equality only whereas === checks equality, and data type, i.e., a value must be of the same type.

### **26) How to write HTML code dynamically using JavaScript?**

The innerHTML property is used to write the HTML code using JavaScript dynamically. It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Let's see a simple example:

1. document.getElementById('mylocation').innerHTML="<h2>This is heading using JavaScript</h2>";

We are going to create the html form when user clicks on the button.

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

1. **<script** type="text/javascript" **>**
2. function showcommentform() {
3. var data="Name:**<input** type='text' name='name'**><br>**Comment:**<br><textarea** rows='5' cols='80'**></textarea>**
4. **<br><input** type='submit' value='Post Comment'**>**";
5. document.getElementById('mylocation').innerHTML=data;
6. }
7. **</script>**
8. **<form** name="myForm"**>**
9. **<input** type="button" value="comment" onclick="showcommentform()"**>**
10. **<div** id="mylocation"**></div>**
11. **</form>**

### **27) How to write normal text code using JavaScript dynamically?**

The innerText property is used to write the simple text using JavaScript dynamically. Let's see a simple example:

1. document.getElementById('mylocation').innerText="This is text using JavaScript";

### **28) How to create objects in JavaScript?**

There are 3 ways to create an object in JavaScript.

1. By object literal
2. By creating an instance of Object
3. By Object Constructor

## **1) JavaScript Object by object literal**

Let's see a simple code to create an object using object literal.

1. **<script>**
2. emp={id:102,name:"Shyam Kumar",salary:40000}
3. document.write(emp.id+" "+emp.name+" "+emp.salary);
4. **</script>**

## **2) By creating instance of Object**

The syntax of creating object directly is given below:

1. var objectname=new Object();

Here, **new keyword** is used to create object.

Let’s see the example of creating object directly.

1. **<script>**
2. var emp=new Object();
3. emp.id=101;
4. emp.name="Ravi Malik";
5. emp.salary=50000;
6. document.write(emp.id+" "+emp.name+" "+emp.salary);
7. **</script>**

## **3) By using an Object constructor**

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

1. **<script>**
2. function emp(id,name,salary){
3. this.id=id;
4. this.name=name;
5. this.salary=salary;
6. }
7. e=new emp(103,"Vimal Jaiswal",30000);
9. document.write(e.id+" "+e.name+" "+e.salary);
10. **</script>**

## **Defining method in JavaScript Object**

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

1. **<script>**
2. function emp(id,name,salary){
3. this.id=id;
4. this.name=name;
5. this.salary=salary;
6. // adding property in the function with same name as method
7. this.changeSalary=changeSalary;
8. function changeSalary(otherSalary){
9. this.salary=otherSalary;
10. }
11. }
12. e=new emp(103,"Sonoo Jaiswal",30000);
13. document.write(e.id+" "+e.name+" "+e.salary);
14. e.changeSalary(45000);
15. document.write("**<br>**"+e.id+" "+e.name+" "+e.salary);
16. **</script>**

#### **Output of the above example**

103 Sonoo Jaiswal 30000  
103 Sonoo Jaiswal 45000

### **29) How to create an array in JavaScript?**

There are 3 ways to create an array in JavaScript.

1. By array literal
2. By creating an instance of Array
3. By using an Array constructor

## **1) JavaScript array literal**

The syntax of creating array using array literal is given below:

1. var arrayname=[value1,value2.....valueN];

As you can see, values are contained inside [ ] and separated by , (comma).

Let's see the simple example of creating and using array in JavaScript.

1. **<script>**
2. var emp=["Sonoo","Vimal","Ratan"];
3. for (i=0;i**<emp.length**;i++){
4. document.write(emp[i] + "**<br/>**");
5. }
6. **</script>**

## **2) JavaScript Array directly (new keyword)**

The syntax of creating array directly is given below:

1. var arrayname=new Array();

Here, **new keyword** is used to create instance of array.

Let's see the example of creating array directly.

1. **<script>**
2. var i;
3. var emp = new Array();
4. emp[0] = "Arun";
5. emp[1] = "Varun";
6. emp[2] = "John";
8. for (i=0;i**<emp.length**;i++){
9. document.write(emp[i] + "**<br>**");
10. }
11. **</script>**

## **3) JavaScript array constructor (new keyword)**

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

The example of creating object by array constructor is given below.

1. **<script>**
2. var emp=new Array("Jai","Vijay","Smith");
3. for (i=0;i**<emp.length**;i++){
4. document.write(emp[i] + "**<br>**");
5. }
6. **</script>**

## **JavaScript Array Methods**

Let's see the list of JavaScript array methods with their description.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [concat()](https://www.javatpoint.com/javascript-array-concat-method) | It returns a new array object that contains two or more merged arrays. |
| [copywithin()](https://www.javatpoint.com/javascript-array-copywithin-method) | It copies the part of the given array with its own elements and returns the modified array. |
| [entries()](https://www.javatpoint.com/javascript-array-entries-method) | It creates an iterator object and a loop that iterates over each key/value pair. |
| [every()](https://www.javatpoint.com/javascript-array-every-method) | It determines whether all the elements of an array are satisfying the provided function conditions. |
| [flat()](https://www.javatpoint.com/javascript-array-flat-method) | It creates a new array carrying sub-array elements concatenated recursively till the specified depth. |
| [flatMap()](https://www.javatpoint.com/javascript-array-flatmap-method) | It maps all array elements via mapping function, then flattens the result into a new array. |
| [fill()](https://www.javatpoint.com/javascript-array-fill-method) | It fills elements into an array with static values. |
| [from()](https://www.javatpoint.com/javascript-array-from-method) | It creates a new array carrying the exact copy of another array element. |
| [filter()](https://www.javatpoint.com/javascript-array-filter-method) | It returns the new array containing the elements that pass the provided function conditions. |
| [find()](https://www.javatpoint.com/javascript-array-find-method) | It returns the value of the first element in the given array that satisfies the specified condition. |
| [findIndex()](https://www.javatpoint.com/javascript-array-findindex-method) | It returns the index value of the first element in the given array that satisfies the specified condition. |
| [forEach()](https://www.javatpoint.com/javascript-array-foreach-method) | It invokes the provided function once for each element of an array. |
| [includes()](https://www.javatpoint.com/javascript-array-includes-method) | It checks whether the given array contains the specified element. |
| [indexOf()](https://www.javatpoint.com/javascript-array-indexof-method) | It searches the specified element in the given array and returns the index of the first match. |
| [isArray()](https://www.javatpoint.com/javascript-array-isarray-method) | It tests if the passed value is an array. |
| [join()](https://www.javatpoint.com/javascript-array-join-method) | It joins the elements of an array as a string. |
| [keys()](https://www.javatpoint.com/javascript-array-keys-method) | It creates an iterator object that contains only the keys of the array, then loops through these keys. |
| [lastIndexOf()](https://www.javatpoint.com/javascript-array-lastindexof-method) | It searches the specified element in the given array and returns the index of the last match. |
| [map()](https://www.javatpoint.com/javascript-array-map-method) | It calls the specified function for every array element and returns the new array |
| [of()](https://www.javatpoint.com/javascript-array-of-method) | It creates a new array from a variable number of arguments, holding any type of argument. |
| [pop()](https://www.javatpoint.com/javascript-array-pop-method) | It removes and returns the last element of an array. |
| [push()](https://www.javatpoint.com/javascript-array-push-method) | It adds one or more elements to the end of an array. |
| [reverse()](https://www.javatpoint.com/javascript-array-reverse-method) | It reverses the elements of given array. |
| [reduce(function, initial)](https://www.javatpoint.com/javascript-array-reduce-method) | It executes a provided function for each value from left to right and reduces the array to a single value. |
| [reduceRight()](https://www.javatpoint.com/javascript-array-reduceright-method) | It executes a provided function for each value from right to left and reduces the array to a single value. |
| [some()](https://www.javatpoint.com/javascript-array-some-method) | It determines if any element of the array passes the test of the implemented function. |
| [shift()](https://www.javatpoint.com/javascript-array-shift-method) | It removes and returns the first element of an array. |
| [slice()](https://www.javatpoint.com/javascript-array-slice-method) | It returns a new array containing the copy of the part of the given array. |
| [sort()](https://www.javatpoint.com/javascript-array-sort-method) | It returns the element of the given array in a sorted order. |
| [splice()](https://www.javatpoint.com/javascript-array-splice-method) | It add/remove elements to/from the given array. |
| [toLocaleString()](https://www.javatpoint.com/javascript-array-tolocalestring-method) | It returns a string containing all the elements of a specified array. |
| [toString()](https://www.javatpoint.com/javascript-array-tostring-method) | It converts the elements of a specified array into string form, without affecting the original array. |
| [unshift()](https://www.javatpoint.com/javascript-array-unshift-method) | It adds one or more elements in the beginning of the given array. |
| [values()](https://www.javatpoint.com/javascript-array-values-method) | It creates a new iterator object carrying values for each index in the array. |

### **30) What does the isNaN() function?**

The isNaN() function returns true if the variable value is not a number. For example:

1. function number(num) {
2. if (isNaN(num)) {
3. return "Not a Number";
4. }
5. return "Number";
6. }
7. console.log(number('1000F'));
8. // expected output: "Not a Number"
10. console.log(number('1000'));
11. // expected output: "Number"

### **31) What is the output of 10+20+"30" in JavaScript?**

3030 because 10+20 will be 30. If there is numeric value before and after +, it treats as binary + (arithmetic operator).

1. function display()
2. {
3. document.writeln(10+20+"30");
4. }
5. display();

### **32) What is the output of "10"+20+30 in JavaScript?**

102030 because after a string all the + will be treated as string concatenation operator (not binary +).

1. function display()
2. {
3. document.writeln("10"+20+30);
4. }
5. display();

### **33) Difference between Client side JavaScript and Server side JavaScript?**

**Client-side JavaScript** comprises the basic language and predefined objects which are relevant to running JavaScript in a browser. The client-side JavaScript is embedded directly by in the HTML pages. The browser interprets this script at runtime.

**Server-side JavaScript** also resembles client-side JavaScript. It has a relevant JavaScript which is to run in a server. The server-side JavaScript are deployed only after compilation.

### **34) In which location cookies are stored on the hard disk?**

The storage of cookies on the hard disk depends on the OS and the browser.

The Netscape Navigator on Windows uses a cookies.txt file that contains all the cookies. The path is c:\Program Files\Netscape\Users\username\cookies.txt

The Internet Explorer stores the cookies on a file username@website.txt. The path is: c:\Windows\Cookies\username@Website.txt.

### **35) What's the difference between event.preventDefault() and event.stopPropagation() methods in JavaScript?**

In JavaScript, the event.preventDefault() method is used to prevent the default behavior of an element.

**For example:** If you use it in a form element, it prevents it from submitting. If used in an anchor element, it prevents it from navigating. If used in a contextmenu, it prevents it from showing or displaying.

On the other hand, the event.stopPropagation() method is used to stop the propagation of an event or stop the event from occurring in the bubbling or capturing phase.

### **36) What is the real name of JavaScript?**

The original name was **Mocha**, a name chosen by Marc Andreessen, founder of Netscape. In September of 1995, the name was changed to LiveScript. In December 1995, after receiving a trademark license from Sun, the name JavaScript was adopted.

### **37) How can you check if the event.preventDefault() method was used in an element?**

When we use the event.defaultPrevent() method in the event object returns a Boolean indicating that the event.preventDefault() was called in a particular element.

### **38) What is the difference between undefined value and null value?**

**Undefined value:** A value that is not defined and is known as undefined value. For example:

1. int number; //Here, a number has an undefined value.

**Null value:** A value that is explicitly specified by the keyword "null" is known as a null value. For example:

1. String str=null; //Here, str has a null value.

### **39) How to set the cursor to wait in JavaScript?**

The cursor can be set to wait in JavaScript by using the property "cursor". The following example illustrates the usage:

1. **<script>**
2. window.document.body.style.cursor = "wait";
3. **</script>**

### **40) What is this [[[]]]?**

This is a three-dimensional array.

1. var myArray = [[[]]];

### **41) Are Java and JavaScript same?**

No, Java and JavaScript are the two different languages. Java is a robust, secured and object-oriented programming language whereas JavaScript is a client-side scripting language with some limitations.

### **42) What is negative infinity?**

Negative Infinity is a number in JavaScript which can be derived by dividing the negative number by zero. For example:

1. var num=-5;
2. function display()
3. {
4. document.writeln(num/0);
5. }
6. display();
7. //expected output: -Infinity

### **43) What is the difference between View state and Session state?**

"View state" is specific to a page in a session whereas "Session state" is specific to a user or browser that can be accessed across all pages in the web application.

### **44) What are the pop-up boxes available in JavaScript?**

* Alert Box
* Confirm Box
* Prompt Box

#### **Example of alert() in JavaScript**

1. **<script** type="text/javascript"**>**
2. function msg(){
3. alert("Hello Alert Box");
4. }
5. **</script>**
6. **<input** type="button" value="click" onclick="msg()"**/>**

#### **Example of confirm() in JavaScript**

1. **<script** type="text/javascript"**>**
2. function msg(){
3. var v= confirm("Are u sure?");
4. if(v==true){
5. alert("ok");
6. }
7. else{
8. alert("cancel");
9. }
10. }
11. **</script>**
12. **<input** type="button" value="delete record" onclick="msg()"**/>**

#### **Example of prompt() in JavaScript**

1. **<script** type="text/javascript"**>**
2. function msg(){
3. var v= prompt("Who are you?");
4. alert("I am "+v);
5. }
6. **</script>**
7. **<input** type="button" value="click" onclick="msg()"**/>**

### **45) How can we detect OS of the client machine using JavaScript?**

The **navigator.appVersion** string can be used to detect the operating system on the client machine.

### **46) How to submit a form using JavaScript by clicking a link?**

Let's see the JavaScript code to submit the form by clicking the link.

1. **<form** name="myform" action="index.php"**>**
2. Search: **<input** type='text' name='query' **/>**
3. **<a** href="javascript: submitform()"**>**Search**</a>**
4. **</form>**
5. **<script** type="text/javascript"**>**
6. function submitform()
7. {
8. document.myform.submit();
9. }
10. **</script>**

### **47) Is JavaScript faster than ASP script?**

Yes, because it doesn't require web server's support for execution.

### **48) How to change the background color of HTML document using JavaScript?**

1. **<script** type="text/javascript"**>**
2. document.body.bgColor="pink";
3. **</script>**

### **49) How to handle exceptions in JavaScript?**

By the help of try/catch block, we can handle exceptions in JavaScript. JavaScript supports try, catch, finally and throw keywords for exception handling.

### **50) How to validate a form in JavaScript?**

1. **<script>**
2. function validateform(){
3. var name=document.myform.name.value;
4. var password=document.myform.password.value;
6. if (name==null || name==""){
7. alert("Name can't be blank");
8. return false;
9. }else if(password.length**<6**){
10. alert("Password must be at least 6 characters long.");
11. return false;
12. }
13. }
14. **</script>**
15. **<body>**
16. **<form** name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" **>**
17. Name: **<input** type="text" name="name"**><br/>**
18. Password: **<input** type="password" name="password"**><br/>**
19. **<input** type="submit" value="register"**>**
20. **</form>**

### **51) How to validate email in JavaScript?**

1. **<script>**
2. function validateemail()
3. {
4. var x=document.myform.email.value;
5. var atposition=x.indexOf("@");
6. var dotposition=x.lastIndexOf(".");
7. if (atposition**<1** || dotposition**<atposition**+2 || dotposition+2**>**=x.length){
8. alert("Please enter a valid email address \n atpostion:"+atposition+"\n dotposition:"+dotposition);
9. return false;
10. }
11. }
12. **</script>**
13. **<body>**
14. **<form** name="myform"  method="post" action="#" onsubmit="return validateemail();"**>**
15. Email: **<input** type="text" name="email"**><br/>**
17. **<input** type="submit" value="register"**>**
18. **</form>**

### **52) What is this keyword in JavaScript?**

The this keyword is a reference variable that refers to the current object. For example:

1. var address=
2. {
3. company:"Javatpoint",
4. city:"Noida",
5. state:"UP",
6. fullAddress:function()
7. {
8. return this.company+" "+this.city+" "+this.state;
9. }
10. };
11. var fetch=address.fullAddress();
12. document.writeln(fetch);

### **53) What is the requirement of debugging in JavaScript?**

JavaScript didn't show any error message in a browser. However, these mistakes can affect the output. The best practice to find out the error is to debug the code. The code can be debugged easily by using web browsers like Google Chrome, Mozilla Firebox.

To perform debugging, we can use any of the following approaches:

* Using console.log() method
* Using debugger keyword

### **54) What is the use of debugger keyword in JavaScript?**

JavaScript debugger keyword sets the breakpoint through the code itself. The debugger stops the execution of the program at the position it is applied. Now, we can start the flow of execution manually. If an exception occurs, the execution will stop again on that particular line. For example:

1. function display()
2. {
3. x = 10;
4. y = 15;
5. z = x + y;
6. debugger;
7. document.write(z);
8. document.write(a);
9. }
10. display();

### **55) What is the role of a strict mode in JavaScript?**

The JavaScript strict mode is used to generates silent errors. It provides "use strict"; expression to enable the strict mode. This expression can only be placed as the first statement in a script or a function.

Strict mode in JavaScript is a feature that was introduced in ECMAScript 5 (ES5) to make JavaScript code more robust by catching common coding mistakes and "unsafe" actions. When you enable strict mode in your JavaScript code, it enforces a stricter set of rules and generates errors or throws exceptions for conditions that would otherwise be silently ignored or cause unexpected behavior.

It's important to note that strict mode is opt-in, meaning you have to explicitly enable it. This is to ensure that older code that relies on non-strict behavior is not broken. However, it is generally recommended to use strict mode in all new JavaScript code as it promotes better coding practices and helps prevent subtle bugs.

For example:

1. "use strict";
2. x=10;
3. console.log(x);

### **57) What is the use of Math object in JavaScript?**

The JavaScript math object provides several constants and methods to perform a mathematical operation. Unlike date object, it doesn't have constructors. For example:

1. function display()
2. {
3. document.writeln(Math.random());
4. }
5. display();

### **58) What is the use of a Date object in JavaScript?**

The JavaScript date object can be used to get a year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

1. function display()
2. {
3. var date=new Date();
4. var day=date.getDate();
5. var month=date.getMonth()+1;
6. var year=date.getFullYear();
7. document.write("**<br>**Date is: "+day+"/"+month+"/"+year);
8. }
9. display();

### **59) What is the use of a Number object in JavaScript?**

The JavaScript number object enables you to represent a numeric value. It may be integer or floating-point. JavaScript number object follows the IEEE standard to represent the floating-point numbers.

1. function display()
2. {
3. var x=102;//integer value
4. var y=102.7;//floating point value
5. var z=13e4;//exponent value, output: 130000
6. var n=new Number(16);//integer value by number object
7. document.write(x+" "+y+" "+z+" "+n);
8. }
9. display();

### **60) What is the use of a Boolean object in JavaScript?**

The JavaScript Boolean is an object that represents value in two states: true or false. You can create the JavaScript Boolean object by Boolean() constructor.

1. function display()
2. {
3. document.writeln(10**<20**);//true
4. document.writeln(10**<5**);//false
5. }
6. display();

### **62) What is the use of a Set object in JavaScript?**

The JavaScript Set object is used to store the elements with unique values. The values can be of any type i.e. whether primitive values or object references. For example:

1. function display()
2. {
3. var set = new Set();
4. set.add("jQuery");
5. set.add("AngularJS");
6. set.add("Bootstrap");
7. for (let elements of set) {
8. document.writeln(elements+"**<br>**");
9. }
10. }
11. display();

### **63) What is the use of a WeakSet object in JavaScript?**

The JavaScript WeakSet object is the type of collection that allows us to store weakly held objects. Unlike Set, the WeakSet are the collections of objects only. It doesn't contain the arbitrary values. For example:

1. function display()
2. {
3. var ws = new WeakSet();
4. var obj1={};
5. var obj2={};
6. ws.add(obj1);
7. ws.add(obj2);
8. //Let's check whether the WeakSet object contains the added object
9. document.writeln(ws.has(obj1)+"**<br>**");
10. document.writeln(ws.has(obj2));
11. }
12. display()

### **64) What is the use of a Map object in JavaScript?**

The JavaScript Map object is used to map keys to values. It stores each element as key-value pair. It operates the elements such as search, update and delete on the basis of specified key. For example:

1. function display()
2. {
3. var map=new Map();
4. map.set(1,"jQuery");
5. map.set(2,"AngularJS");
6. map.set(3,"Bootstrap");
7. document.writeln(map.get(1)+"**<br>**");
8. document.writeln(map.get(2)+"**<br>**");
9. document.writeln(map.get(3));
10. }
11. display();

### **65) What is the use of a WeakMap object in JavaScript?**

The JavaScript WeakMap object is a type of collection which is almost similar to Map. It stores each element as a key-value pair where keys are weakly referenced. Here, the keys are objects and the values are arbitrary values. For example:

1. function display()
2. {
3. var wm = new WeakMap();
4. var obj1 = {};
5. var obj2 = {};
6. var obj3= {};
7. wm.set(obj1, "jQuery");
8. wm.set(obj2, "AngularJS");
9. wm.set(obj3,"Bootstrap");
10. document.writeln(wm.has(obj2));
11. }
12. display();

### **66) What are the falsy values in JavaScript, and how can we check if a value is falsy?**

Those values which become false while converting to Boolean are called falsy values.

1. const falsyValues = ['', 0, null, undefined, NaN, false];

We can check if a value is falsy by using the Boolean function or the Double NOT operator (!!).

Falsy values are the values that, when used in a conditional statement (like an **if** statement), will cause the condition to be **false**. Here are the common falsy values in JavaScript:

1. **false**: The boolean value **false**.
2. **0**: The number zero.
3. **-0**: Negative zero.
4. **0n**: BigInt zero.
5. **""**: An empty string.
6. **null**: A special value representing the absence of any object value.
7. **undefined**: A value representing an uninitialized or non-existent variable or object property.
8. **NaN**: A special value representing "Not-a-Number."

**Syntax to check falsy value:**

if (!value) {

// The value is falsy

}

### **67) What do you understand by hoisting in JavaScript?**

Hoisting is the default behavior of JavaScript where all the variable and function declarations are moved on top. In simple words, we can say that Hoisting is a process in which, 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:**

1. hoistedVariable = 12;
2. console.log(hoistedVariable); // outputs 12 even when the variable is declared after it is initialized
3. var hoistedVariable;

**Example2:**

1. hoistedFunction();  // Outputs " Welcome to JavaTpoint " even when the function is declared after calling
2. function hoistedFunction(){
3. console.log(" Welcome to JavaTpoint ");
4. }
5. Example3:
6. // Hoisting in a local scope
7. function doSomething(){
8. x = 11;
9. console.log(x);
10. var x;
11. }
12. doSomething(); // Outputs 11 since the local variable "x" is hoisted inside the local scope

### **3. What are the various data types that exist in JavaScript?**

These are the different types of data that JavaScript supports:

* Boolean - For true and false values
* Null - For empty or unknown values
* Undefined - For variables that are only declared and not defined or initialized
* Number - For integer and floating-point numbers
* String - For characters and alphanumeric values
* Object - For collections or complex values
* Symbols - For unique identifiers for objects

### **4. What are the features of JavaScript?**

These are the features of JavaScript:

* Lightweight, interpreted programming language
* Cross-platform compatible
* Open-source
* Object-oriented
* Integration with other backend and frontend technologies
* Used especially for the development of network-based applications

### **9. What are the scopes of a variable in JavaScript?**

The scope of a variable implies where the variable has been declared or defined in a JavaScript program. There are two scopes of a variable:

#### **Global Scope**

Global variables, having global scope are available everywhere in a JavaScript code.

#### **Local Scope**

Local variables are accessible only within a function in which they are defined.

### **10. What is the ‘this’ keyword in JavaScript?**

The [‘this’ keyword in JavaScript](https://www.simplilearn.com/tutorials/javascript-tutorial/javascript-this-keyword) refers to the currently calling object. It is commonly used in constructors to assign values to object properties.

### **11. What are the conventions of naming a variable in JavaScript?**

Following are the naming conventions for a variable in JavaScript:

* Variable names cannot be similar to that of reserved keywords. For example, var, let, const, etc.
* Variable names cannot begin with a numeric value. They must only begin with a letter or an underscore character.
* Variable names are case-sensitive.

### **12. What is Callback in JavaScript?**

In JavaScript, functions are objects and therefore, functions can take other functions as arguments and can also be returned by other functions.

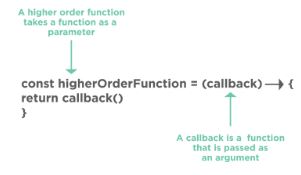


Fig: Callback function

A [callback](https://www.simplilearn.com/tutorials/javascript-tutorial/callback-function-in-javascript) is a [JavaScript function](https://www.simplilearn.com/tutorials/javascript-tutorial/javascript-functions) that is passed to another function as an argument or a parameter. This function is to be executed whenever the function that it is passed to gets executed.

### **14. What is the difference between Function declaration and Function expression?**

|  |  |
| --- | --- |
| **Function declaration** | **Function expression** |
| Declared as a separate statement within the main JavaScript code | Created inside an expression or some other construct |
| Can be called before the function is defined | Created when the execution point reaches it; can be used only after that |
| Offers better code readability and better code organization | Used when there is a need for a conditional declaration of a function |
| Example:  function abc() {      return 5;  } | Example:  var a = function abc() {      return 5;  } |

### **15. What are the ways of adding JavaScript code in an HTML file?**

There are primarily two ways of embedding JavaScript code:

* We can write JavaScript code within the script tag in the same HTML file; this is suitable when we need just a few lines of scripting within a web page.
* We can import a JavaScript source file into an HTML document; this adds all scripting capabilities to a web page without cluttering the code.

### **16. What do you understand about cookies?**

A cookie is generally a small data that is sent from a website and stored on the user’s machine by a web browser that was used to access the website. Cookies are used to remember information for later use and also to record the browsing activity on a website.

### **17. How would you create a cookie?**

The simplest way of creating a cookie using JavaScript is as below:

document.cookie = "key1 = value1; key2 = value2; expires = date";

### **18. How would you read a cookie?**

Reading a cookie using JavaScript is also very simple. We can use the document.cookie string that contains the cookies that we just created using that string.

The document.cookie string keeps a list of name-value pairs separated by semicolons, where ‘name’ is the name of the cookie, and ‘value’ is its value. We can also use the split() method to break the cookie value into keys and values.

### **19. How would you delete a cookie?**

To delete a cookie, we can just set an expiration date and time. Specifying the correct path of the cookie that we want to delete is a good practice since some browsers won’t allow the deletion of cookies unless there is a clear path that tells which cookie to delete from the user’s machine.

function delete\_cookie(name) {

  document.cookie = name + "=; Path=/; Expires=Thu, 01 Jan 1970 00:00:01 GMT;";

}

### **20. What’s the difference between let and var?**

Both let and var are used for variable and method declarations in JavaScript. So there isn’t much of a difference between these two besides that while var keyword is scoped by function, the let keyword is scoped by a block.

### **21. What are Closures in JavaScript?**

[Closures](https://www.simplilearn.com/tutorials/javascript-tutorial/javascript-closure) provide a better, and concise way of writing JavaScript code for the developers and programmers. Closures are created whenever a variable that is defined outside the current scope is accessed within the current scope.

Closures are a fundamental concept in JavaScript, and they occur when a function is defined within another function and has access to the outer function's variables. This allows the inner function to "remember" and access those variables even after the outer function has finished executing. Closures are a powerful and versatile feature in JavaScript, and they have several important use cases.

function outer() {

let outerVar = "I am from the outer function";

function inner() {

console.log(outerVar); // inner function can access outerVar

}

return inner;

}

const closureFunction = outer();

closureFunction(); // This will log "I am from the outer function"

In this example, **inner** is a closure because it's defined within the **outer** function and can access the **outerVar** variable even after **outer** has completed its execution.

function createCounter() {

let count = 0;

return {

increment: function() {

count++;

},

decrement: function() {

count--;

},

getCount: function() {

return count;

}

};

}

const counter = createCounter();

counter.increment();

counter.increment();

console.log(counter.getCount()); // Outputs 2

In this example, the **createCounter** function creates a closure that encapsulates the **count** variable, providing methods to interact with it while keeping it private from external code. Closures make it possible to maintain state and data isolation within functions, contributing to modular and maintainable code.

### **22. What are the arrow functions in JavaScript?**

Arrow functions are a short and concise way of writing functions in JavaScript. The general syntax of an arrow function is as below:

const helloWorld = () => {

  console.log("hello world!");

};

### **23. What are the different ways an HTML element can be accessed in a JavaScript code?**

Here are the ways an HTML element can be accessed in a JavaScript code:

* getElementByClass(‘classname’): Gets all the HTML elements that have the specified classname.
* getElementById(‘idname’): Gets an HTML element by its ID name.
* getElementbyTagName(‘tagname’): Gets all the HTML elements that have the specified tagname.
* querySelector(): Takes CSS style selector and returns the first selected HTML element.

### **25. What are Imports and Exports in JavaScript?**

Imports and exports help in writing modular code for our [JavaScript applications](https://www.simplilearn.com/applications-of-javascript-article). With the help of imports and exports, we can split a JavaScript code into multiple files in a project. This greatly simplifies the application source code and encourages code readability.

**calc.js file**

export const sqrt = Math.sqrt;

export function square(x) {

  return x \* x;

}

export function diag(x, y) {

  return sqrt(square(x) + square(y));

}

This file exports two functions that calculate the squares and diagonal of the input respectively.

**main.js file**

import { square, diag } from "calc";

console.log(square(4)); // 16

console.log(diag(4, 3)); // 5

Therefore, here we import those functions and pass input to those functions to calculate square and diagonal.

### **26. What is the difference between Document and Window in JavaScript?**

|  |  |
| --- | --- |
| Document | Window |
| The document comes under the windows object and can also be considered as its property. | Window in JavaScript is a global object that holds the structure like variables, functions, location, history, etc. |

### **27. What are some of the JavaScript frameworks and their uses?**

JavaScript has a collection of many frameworks that aim towards fulfilling the different aspects of the web application development process. Some of the prominent frameworks are:

* React - Frontend development of a web application
* Angular - Frontend development of a web application
* Node - Backend or server-side development of a web application

### **28. What is the difference between Undefined and Undeclared in JavaScript?**

|  |  |
| --- | --- |
| Undefined | Undeclared |
| Undefined means a variable has been declared but a value has not yet been assigned to that variable. | Variables that are not declared or that do not exist in a program or application. |

### **29. What is the difference between Undefined and Null in JavaScript?**

|  |  |
| --- | --- |
| Undefined | Null |
| Undefined means a variable has been declared but a value has not yet been assigned to that variable. | Null is an assignment value that we can assign to any variable that is meant to contain no value. |

### **30. What is the difference between Session storage and Local storage?**

|  |  |
| --- | --- |
| Session storage | Local storage |
| The data stored in session storage gets expired or deleted when a page session ends. | Websites store some data in local machine to reduce loading time; this data does not get deleted at the end of a browsing session. |

### **34. What are the scopes of a variable in JavaScript?**

The scope of variables in JavaScript is used to determine the accessibility of variables and functions at various parts of one’s code. There are three types of scopes of a variable, global scope, function scope, block scope

* **Global Scope:** It is used to access the variables and functions from anywhere inside the code.

Example:

var globalVariable = "Hello world";

function sendMessage(){

  return globalVariable; // globalVariable is accessible as it's written in global space

}

* **Function scope:** It is used to declare the function and variables inside the function itself and not outside.
* **Block Scope:** It uses let and const to declare the variables.

Example:

{

  let x = 45;

}

console.log(x); // Gives reference error since x cannot be accessed outside of the block

for(let i=0; i<2; i++){

  // do something

}

console.log(i); // Gives reference error since i cannot be accessed outside of the for loop block

### **39. Implicit Type Conversion in javascript (in detail with examples)**

When the value of one data type is automatically converted into another data type, it is called Implicit type conversion in javascript.

### **40. Is javascript a statically typed or a dynamically typed language?**

Yes, JavaScript is a dynamically typed language and not statically.

### **43. Immediately Invoked Function in JavaScript**

An Immediately Invoked Function also abbreviated as IIFE or IIFY runs as soon as it is defined. To run the function, it needs to be invoked otherwise the declaration of the function is returned.

**Syntax**

(function()

{

  // Do something;

})();

(function(name) {

console.log("Hello, " + name);

})("John"); // Outputs: Hello, John

### **44. Characteristics of javascript strict-mode**

* Strict mode does not allow duplicate arguments and global variables.
* One cannot use JavaScript keywords as a parameter or function name in strict mode.
* All browsers support strict mode.
* Strict mode can be defined at the start of the script with the help of the keyword ‘use strict’.

### **45. Higher Order Functions (with examples)**

Higher-order functions are the functions that take functions as arguments and return them by operating on other functions

Example:

function higherOrder(fn)

 {

  fn();

}

higherOrder(function() { console.log("Hello world") });

### **49. Advantages of using External JavaScript**

* External Javascript allows web designers and developers to collaborate on HTML and javascript files.
* It also enables you to reuse the code.
* External javascript makes Code readability simple.

### **50. What are object prototypes?**

Following are the different object prototypes in javascript that are used to inherit particular properties and methods from the Object.prototype.

1. Date objects are used to inherit properties from the Date prototype
2. Math objects are used to inherit properties from the Math prototype
3. Array objects are used to inherit properties from the Array prototype.

### **51. Types of errors in javascript**

Javascript has two types of errors, Syntax error, and Logical error.

### **52. What is memorization?**

In JavaScript, when we want to cache the return value of a function concerning its parameters, it is called memorization. It is used to speed up the application especially in case of complex, time consuming functions.

### **53. Recursion in a programming language**

Recursion is a technique in a programming language that is used to iterate over an operation whereas a function calls itself repeatedly until we get the result.

### **54. Use of a constructor function (with examples)**

Constructor functions are used to create single objects or multiple objects with similar properties and methods.

Example:

function Person(name,age,gender)

{

  this.name = name;

  this.age = age;

  this.gender = gender;

}

var person1 = new Person("Vivek", 76, "male");

console.log(person1);

var person2 = new Person("Courtney", 34, "female");

console.log(person2);

### **55. Which method is used to retrieve a character from a certain index?**

We can retrieve a character from a certain index with the help of charAt() function method.

### **56. What is BOM?**

BOM is the Browser Object Model where users can interact with browsers that is a window, an initial object of the browser. The window object consists of a document, history, screen, navigator, location, and other attributes. Nevertheless, the window’s function can be called directly as well as by referencing the window.

### **61. Promises in JavaScript**

Promises in JavaScript have four different states. They are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Pending** | **Fulfilled** | **Rejected** | **Settled** |
| Pending is an initial state of promise. It is the initial state of promise where it is in the pending state that neither is fulfilled nor rejected. | It is the state where the promise has been fulfilled that assures that the async operation is done. | It is the state where the promise is rejected and the async operation has failed. | It is the state where the promise is rejected or fulfilled. |

Example:

function sumOfThreeElements(...elements)

{

  return new Promise((resolve, reject)=>{

    if(elements.length > 3)

{

      reject("Just 3 elements or less are allowed");

    }

    else

{

      let sum = 0;

      let i = 0;

      while(i < elements.length)

{

        sum += elements[i];

        i++;

      }

      resolve("Sum has been calculated: "+sum);

    }

  })

}

### **65. What is the use of callbacks?**

* A callback function is used to send input into another function and is performed inside another function.
* It also ensures that a particular code does not run until another code has completed its execution.

### **72. Primitive data types**

The primitive data types are capable of displaying one value at a time. It consists of Boolean, Undefined, Null, Number, and String data types.

### **77. How to detect the OS of the client machine using JavaScript?**

The OS on the client machine can be detected with the help of navigator.appVersion string

### **80. How do you empty an array in JavaScript?**

There are a few ways in which we can empty an array in JavaScript:

* **By assigning array length to 0:**

var arr = [1, 2, 3, 4];

arr.length = 0;

* **By assigning an empty array:**

var arr = [1, 2, 3, 4];

arr = [];

* **By popping the elements of the array:**

var arr = [1, 2, 3, 4];

while (arr.length > 0) {

  arr.pop();

}

* **By using the splice array function:**

var arr = [1, 2, 3, 4];

arr.splice(0, arr.length);

### **81. What is the difference between Event Capturing and Event Bubbling?**

|  |  |
| --- | --- |
| **Event Capturing** | **Event Bubbling** |
| This process starts with capturing the event of the outermost element and then propagating it to the innermost element. | This process starts with capturing the event of the innermost element and then propagating it to the outermost element. |

### **85. What is the difference between Call and Apply?**

|  |  |
| --- | --- |
| **Call** | **Apply** |
| In the call() method, arguments are provided individually along with a ‘this’ value. | In the apply() method, arguments are provided in the form of an array along with a ‘this’ value. |

## **JSON**

**JSON** or **JavaScript Object Notation** is a popular text format. **JSON** is language independent, lightweight and can be used to exchange and store data. **JSON** syntax is both human and machine readable. **JSON** was build in 2001 but JavaScript starts supporting **JSON** in ECMA5. File extension for **JSON** file is .json.

## **What is Closure**

A **Closure** is combination of function and the lexical environment (surrounding state). Using **Closure**, a nested function (function inside another function) can access local variables of parent functions.

**Closure** are created every time a function is created inside another function. The child function can access variables of parent function. But parent function cannot access variables of child function because of scope.

function outer(){

var x=3;

function inner(){

var y=4;

return x+y;

}

return inner;

}

console.log( outer() )

In the above example, the function inner can access variables of outer.

## **Lexical Environment**

**Lexical Environment** is the scope of function. A variable inside function is accessible anywhere in the function. When a nested function is created, it can also access variable of parent function.

## **JavaScript Callback Function**

A function that is passed as an argument to another function is known as **callback function**. This means, **callback functions** are just any JavaScript function, but that can be used as an argument to another function.

**Callback Functions** are high order functions. A **high order functions** in javascript includes **callback functions** and **function that returns a function**.

### **Callback Function Example**

function main(x){

x();

}

function callback(){

console.log("hi");

}

main(callback);

In the above example, main is a function with parameter x. Inside main function, we are calling x. This means x should be a function, not any other datatype. Argument of main function is callback. The above example is **synchronous callback** as the code execute immediately.

**Synchronous callback** is a callback where code execute immediately, i.e. synchronously.

## **Asynchronous Callback**

**Asynchronous callback** is a callback where code within callback function is executed after main thread is completed. **Event Loop** handle all the asynchronous operations in JavaScript. For **asynchronous callback**, we can use any one of below techniques in javascript.

### **Asynchronous techniques in javascript**

1. setTimeout / setInterval
2. events
3. ajax/fetch
4. Promises
5. async await

**Event Loop** is the runtime model on which javascript is based. By default frontend javascript is synchronous. But using **Event Loop**, we can run asynchronous programs in javascript.

### **Iterate each element in querySelectorAll**

To **iterate each element in querySelectorAll**, we should use **forEach** method. forEach is a loop used to iterate [Arrays](https://tutorial.techaltum.com/javascript-arrays.html) or [Objects](https://tutorial.techaltum.com/javascript-objects.html). **querySelectorAll** returns a **Array Like** list, which supports **forEach** Method.

forEach methods first parameter is Callback Function with parameter in Callback.

<p> Para 1</p>

<p> Para 2</p>

<p> Para 3</p>

<script>

document.querySelectorAll('p').forEach(function(i){

console.log(i);

});

</script>

## **Javascript Events**

**Events** are the techniques used by a user to interact with webpage, like **mouse click**, **mouse hover**, **key press**, **keyup**, **right click**, **drag** **touch** etc. Javascript can handle **Keyboard based events**, **mouse based events** and **Touch Based events**. A fully **interactive website** is not possible without **JS Events**.

**Touch based events** and **Drag and Drop** are also supported in javascript after ES5(2011).

#### **onclick**

<script>

document.querySelector("button").onclick=function(){

alert(this.textContent);

}

</script>

<!--add event to element, can assign single function only-->

#### **Event Listener**

<script>

document.querySelector("button").addEventListener("click",function(){ alert(this.textContent) })

</script>

<!--recommended way of dealing with Javascript Events-->

## **Event Listeners**

**Event Listeners** are used to add and remove **JavaScript Events**. The most preferred and recommended way of dealing with **javascript events** is to use **event listeners**. These **event listeners**were introduced in ECMA 5. Thus all modern web browsers support them (*Except I.E 8 and below*).

To use event listeners, we use [addEventListener()](https://tutorial.techaltum.com/addeventlistener.html#addEventListener) function (to add event), and [removeEventListener()](https://tutorial.techaltum.com/addeventlistener.html#removeEventListener) function (to remove an event).

### **Advantage of using Event Listeners**

1. Can add multiple functions on same event.
2. Event can be removed using [removeEventListener](https://tutorial.techaltum.com/addeventlistener.html#removeEventListener) function.
3. [Event Propagation](https://tutorial.techaltum.com/addeventlistener.html#prop) can be changed.
4. Can use custom events, like [touch events](https://tutorial.techaltum.com/addeventlistener.html#touch).

## **Event Propagation**

**Event Propagation** is the order in which an events fire from child element to parent. By default events propagates from child to parent node, also known as bubbling.

Type of event propagations

* Bubbling
* Capturing

### **Event Bubbling**

**Event Bubbling** is when event fire on an elements and then bubbles up to parent Elements till it reach root node. By default all events are bubbling type. In example show, click events is used on both parent div, and child button. But button will call first.

<div id="div1">

<button id="button1">Button 1</button>

</div>

<script>

document.querySelector("#div1").addEventListener("click",function(){

alert("you clicked div");

});

document.querySelector("#button1").addEventListener("click",function(){

alert("you clicked button");

});

</script>

### **Event capturing**

**Event capturing** is when root node is fired first, and then Propagate downwards until it reach targeted element. Capturing is possible by using third boolean parameter (true) in event listener. Here is an example.

<div id="div2">

<button id="button2">Button 1</button>

</div>

<script>

document.querySelector("#div2").addEventListener("click",function(){

alert("you clicked div");

},true);

document.querySelector("#button2").addEventListener("click",function(){

alert("you clicked button");

},true);

</script>

## **AJAX**

**AJAX** stands for **Asynchronous JavaScript and XML**. In **JavaScript AJAX** is used to fetch data asynchronous from web server without refreshing webpage. For example, search results.

**Ajax** can send and receive data like *XML*, *JSON*, *TEXT* and *HTML* from web server without reloading.

### **JavaScript AJAX benefits**

1. Load data from Web Server.
2. Update webpage content without reloading
3. Communicate with web server asynchronously.
4. Send data to server in background.

**Asynchronous** means the request will not block other requests and run in parallel. This is the modern and most efficient way to send or receive data from server.

### **HTTP Status**

The **HTTP Status**property returns **HTTP Status Code**, which is an integer value. Here are some popular **HTTP Status Codes**.

1. **200** → Response is successful
2. **201** → Resource was created
3. **204** → Request is successful, but no data received.
4. **403** → Forbidden
5. **404** → Page Not Found
6. **500** → Internal Server Error

## **Fetch API**

**Fetch API** is [Promise](https://tutorial.techaltum.com/javascript-promise.html) based api used to fetch resource across web server, almost same like [AJAX](https://tutorial.techaltum.com/javascript-ajax.html) but with easy syntax and more features.

**Fetch API** can also send or receive data like *XML*, *JSON*, *TEXT* and *HTML* from web server without reloading. Fetch is also used to **fetch data from Web API** or REST APIs.

### **Fetch Vs Ajax**

| **Properties** | **Fetch** | **Ajax ( xhr)** |
| --- | --- | --- |
| **ECMA Version** | ES6 | ES5 |
| **Architecture** | Promise based | Event Based |
| **Complexity** | Easy | Complex |
| **Syntax** | Simple | Verbose |
| **Request / Response** | Supports | Supports but complex structure |
| **Cookies** | send cookies | Cookie less |

**fetch api** is based on **Request** and **Response**. Request is the incoming stream to server and Response is the outgoing stream from server.

Request is the URL of resource.

then handler is used to get response of request. then includes a callback with Response.

### **fetch example with then handler**

fetch(req).then(res=>res.text()).then(res=>console.log(res));

## **JS Promises**

**JavaScript Promises** are a modern approach to handle **asynchronous operations**. Although we can handle small small asynchronous operations using [Callback Functions](https://tutorial.techaltum.com/callback-function.html), but not large operations. **Promises** can handle long and nested asynchronous operations easily.

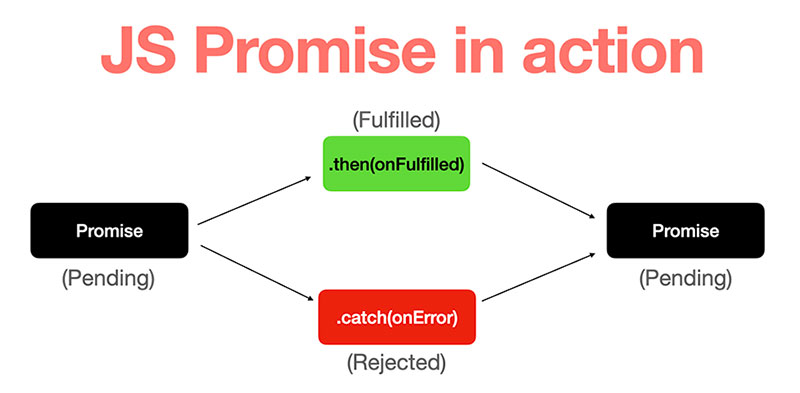
**Promises**are also helpful to handle long chaining of nested callback functions known as *callback hell* in javascript.

### **Build-in Promise in JavaScript**

1. [Fetch API](https://tutorial.techaltum.com/fetch-api.html)
2. [getBattery](https://tutorial.techaltum.com/javascript-navigator.html#getbattery)

## **Promise States**

A **Promise** is always in one of these states, pending, fulfilled and rejected. For **chaining in Promise**, we can use **then** or **catch** method.

Javascript Promise

|  |  |
| --- | --- |
| **Promise State** | **Explanation** |
| **pending** | The initial state, waiting for fulfill or reject |
| **fulfilled** | the operation has successfully done |
| **rejected** | the operation was failed |

A Promise is said to be resolved or settled when it is fulfilled or rejected. Once settled (resolved or rejected), a Promise cannot be changed. That means a Promise is immutable after settled.

The **then()** and **catch()** methods of Promise are used to handle callbacks that executes when it is settled or resolved.

### **Promise fulfilled**

function done(){

console.log("done");

}

function error(){

console.error("error");

}

var promise=new Promise((resolve,reject)=>{

resolve();

reject();

});

promise.then(done).catch(error);

## **Async and Await**

**Async and Await functions**are also used to handle **asynchronous operations**. They use **async** and **await** keywords to handle [Promises](https://tutorial.techaltum.com/javascript-promise.html) based functionality but in neater and cleaner way. Thus we can avoid using Promises and its chaining for asynchronous programming.

### **Asynchronous programming in JavaScript**

To run a code **asynchronously** in JavaScript, we can use any one of following.

1. setTimeout
2. setInterval
3. Events
4. Promises
5. Async and Await

## **Async**

**Async keyword** is used before function keyword. Async function will always return a promise. Even if the return value of async function is not a promise, it will still be wrapped in a promise.

## **Await**

**await** keyword or operator is used inside async function to wait for promise.  **await** will wait for promise to full filled or reject and return the output.

**await** is only used in **async function** or top level of module.

async function asyncFunction(){

let getData=function(){

return setTimeout(function(){

console.log("task done");

});

};

let res=await getData;

res();

console.log("function called");

}

asyncFunction();

console.log("app start");

1. app start
2. function called
3. task done

## **Cookies**

**JavaScript Cookies** are used to store data on client side by in string format. Data stored in cookies can be used when user visit again. Cookies was developed by Netscape for Navigator Browser. When user leaves a website, browser never save anything. Cookies can save upto 4kb text data on client side.

### **Cookies advantages**

1. Login using cookies (without password).
2. Product added in shopping cart.
3. Last visited products on e-commerce sites.
4. Ads Channels show relevant ads.
5. Sites like Google use cookies for securities.

### **Cookies Limitations**

1. Can store text data only.
2. Can store upto 4kb data only.

## **Create Cookies**

To **create cookies in javascript**, we use document.cookie property and assign key/value in strings. Key and values in string must be separated by = .

### **Create Cookie Example**

document.cookie="name=avinash";

document.cookie="location=delhi";