JavaScript Interview Questions and Answers for Freshers

1) What is the difference between Java and JavaScript?

Java and ***JavaScript*** are both programming languages, but they have different ***origins***, ***syntax***, and ***use cases***.

Java is an object-oriented programming language. And it was designed to be ***platform-independent.*** It means that Java programs can run on any system or machine which has a ***Java Virtual Machine (JVM)*** installed. ***Java*** is commonly used for developing ***desktop, web***, and ***Android mobile applications***.

On the other hand, ***JavaScript*** is a scripting language. And it was designed to add additional functionality to the webpage and add interactivity and dynamic behavior to web pages. It is executed on the ***client-side.*** JavaScript is primarily used for creating ***interactive*** web pages and web applications.

2) What is JavaScript?

JavaScript is a powerful scripting language which is used for creating dynamic and interactive websites. It can be used for ***Frontend*** and ***Backend Development***. And JavaScript is a ***high-level*** and most popular programming language. It is executed on the web browser. And now it can also run on the server using ***Node.JS*** runtime environment.

3) What are the data types supported by JavaScript?

JavaScript supports several data types, including:

1). Primitive Data Types

- **Number:** Number is used for numeric values, including integers and decimals.

- ***String:*** String is used for text values.
- **Boolean:** Boolean is used for logical values, either true or false.
- **Undefined:** It is used for uninitialized variables or missing property values.
- **Null:** Null in JavaScript is used for representing intentional absence of any object value.
- **Symbol (added in ECMAScript 6):** Symbol in JavaScript is used for creating unique identifiers.

2). Complex Data Types

Object: Object in JavaScript are used for storing collections of data, including ***key-value pairs*** and ***arrays***. Objects are essentially containers for other ***data types*** and object can also contain functions

4) What are the best features of JavaScript?

JavaScript is a ***high-level***, and ***interpreted*** programming language which is widely used for ***creating interactive*** and ***dynamic web pages***. Here are some key features of JavaScript:

- Object-Oriented
- Dynamic Typing
- Interactivity
- Cross-Platform
- Client-Side Execution
- Easy to Learn
- Versatility

5) Is JavaScript a case-sensitive language explains with Example?

JavaScript is a ***case-sensitive*** language. It means that the language ***keywords***, ***variables***, ***function names***, and any other ***identifiers*** must always be typed with a consistent capitalization of letters.

For example, the variable ***"myJSVariable"*** and ***"myjsvariable"*** would be considered as two different variables in JavaScript

6) Advantages and Disadvantages of JavaScript

Advantages

- **Simple:** JavaScript is simple and easy to use language for beginner. Users and Developers both find the structure to be straightforward.
- **Speed:** JavaScript is a ***"interpreted"*** language, and no matter where JavaScript is hosted, it is always run in a client environment to reduce bandwidth usage and speed up execution.
- **Fast speed:** JavaScript is executed on the ***client side*** that's why it is very fast.
- **Easy to learn:** JavaScript is easy to learn. Any one who have basic knowledge of programming can easily lean JavaScript.
- **Versatility:** JavaScript refers to lots of skills. So it can be used in a wide range of applications.
- **Browser Compatible:** JavaScript supports all modern browsers. So, it can execute on any browser and produce same result.
- **Server Load:** It reduces the server load because it executes on the ***client side***.

- **Rich interfaces:** JavaScript provides the drag and drop functionalities which can provides the rich look to the web pages.

- **Popularity:** JavaScript is a very popular scripting language because it is used everywhere on the web.

Disadvantages:

- **Security vulnerabilities:** JavaScript can pose security risks if not used properly. **Malicious code** can be injected into a web page and executed by the browser, which can lead to data theft and other security issues.

- **Browser compatibility issues:** JavaScript code can behave differently on different web browsers, which can lead to ***compatibility issues***. So developers need to test their code on multiple browsers to ensure that it works as expected and it does not contains any issues.

- **Performance issues:** It can sometimes be slower than other programming languages, especially when dealing with large amounts of data. This problem can lead to performance issues and slow page loading times.

- **Limited control over hardware:** JavaScript has limited access to hardware resources such as the ***file system*** and ***network ports***. Which can make it difficult to create certain types of applications.

7) How can you create an object in JavaScript?

Almost everything is an object in JavaScript. And we can create objects using multiple ways and most common used are:

Using an Object Literal

```
const person = {
  firstName: "Raju",
  lastName: "WebDev",
  age: 20,
  eyeColor: "blue"
};
```

Using the JavaScript Keyword new

```
const person = new Object();
person.firstName = "Raju";
person.lastName = "WebDev";
person.age = 20;
person.eyeColor = "blue";
```

8) How can you create an Array in JavaScript?

The most common way to create an array in JavaScript is to define an array using the array literal. Given code below is used to create an array using array literal:

```
const codingSeries = ["HTML Simplified", "CSS Master", "JavaScript Doctory"];
```

9) What is a name function in JavaScript & how to define it?

A named function declares a name as soon as it is defined. It can be defined using function keyword as :

```
function myFunction() {
  // write code here
}
```

10) What is the 'this' keyword in JavaScript.

There can be different uses of ***this*** keyword in JavaScript. In JavaScript, ***this*** is a reference variable which refers to the current object.

***this* can also be used to:**

i) Invoke current class constructor

ii) Invoke current class method

iii) Return the current class object

iv) Pass an argument in the method call

v) Pass an argument in the constructor call

11) What is Callback function in JavaScript?

In JavaScript, a ***callback function*** is a function which is passed as an argument to another function, to be ***"called back"*** at a later time.

A function which accepts other functions as arguments is called a ***higher-order function***, which contains the logic for when the callback function gets executed. It's the combination of these two that allow us to extend our functionality.

12) Explain call() apply() and bind() methods in JavaScript.

call(), ***apply()***, and ***bind()*** are methods available in JavaScript which allows you to change the context in which a function is executed.

i) call() method is used to invoke a function with a specified ***this*** value and arguments provided individually. It accepts the this value as the ***first argument***, followed by the arguments to be passed to the function.

```
function greetHello(name) {
  console.log(`Hello, ${name}!`);
}

const person = {
  name: 'Geeks Help'
};

greetHello.call(person.name, person.name);

// Output: Hello, Geeks Help!
```

ii) apply() method is similar to ***call()***, but it accepts an array of arguments instead of individual arguments. The first argument is still the ***this*** value.

```
function greet(greeting, punctuation) {
  console.log(`${greeting}, ${this.name}${punctuation}`);
}

const person = {
  name: 'Raju'
};

greet.apply(person, ['Hello', '!']);

// Output: "Hello, Raju!"
```

iii) 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.

```
const person = {
  name: 'Raju WebDev'
};

function greetHello() {
  console.log(`Hello, ${this.name}!`);
}

const greetHelloToRaju = greetHello.bind(person);

greetHelloToRaju();

//Output: "Hello, Raju WebDev!"
```

13) What is the difference between exec() and test() methods in JavaScript?

i). exec(); The exec() method tests for a match in a string. If ***exec() method*** finds a match then it returns a ***result array***, otherwise it will ***return null***. In other words, The ***exec()*** method takes a string as the parameter. This string is which is to be checked for match in the given string.

Syntax: Here is the syntax of the ***exec() method*** of ***regular expressions in JavaScript***:

```
regularExpressionObj.exec(string);
```

14) What is currying in JavaScript with Example.

Currying in JavaScript transforms a function with multiple arguments into a nested series of functions, each taking a single argument. Currying helps you to avoid passing the same variable multiple times, and ***currying in JavaScript*** helps to create a ***higher order function***.

Here is the example of currying in JavaScript

```
function add(x) {
  return function (y) {
    return x + y;
  }
}

// Usage
const add5 = add(5);

console.log(add5(3));
//Output: 8

console.log(add5(7));
//Output: 12
```

15) What is closure in JavaScript with example.

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

Closures are the functions which refer to independent variables. In other words, the function defined in the closure ***'remembers'*** the environment in which it was created.

```
function numberGenerator() {
 // Local “free” variable that ends up within the closure
 let num = 1; function checkNumber() {
  console.log(num);
 }
 num++;
 return checkNumber;
}

let number = numberGenerator();
number();

// Output: 2
```

16) How to create a cookie using JavaScript?

A ***cookie*** is an amount of information that persists between a ***server-side*** and a ***client-side***. A web browser stores this information at the time of browsing in ***client's browser.***

A cookie contains the information generally as a string in the form of a ***name-value pair*** separated by ***semi-colons***. Cookie maintains the state of a user and remembers the user's information among all the web pages.

Given syntax is used to create a cookie:

```
document.cookie="name=value";
```

17) How to read a cookie using JavaScript?

Reading a cookie is just as simple as writing, because the value of the ***document.cookie*** object is the cookie. So you can use this string whenever you want to access the cookie. The ***document.cookie*** string will keep a list of ***name=value*** pairs separated by ***semicolons***, where name is the ***name of a cookie*** and value is ***its string value***.

To read a cookie using JavaScript, you can use the ***document.cookie*** property, which returns a string containing all the cookies associated with the current document.

```
let userName = document.cookie = "name=raju webdev";
console.log(userName);
```

18) How to delete a cookie using JavaScript?

To delete a cookie using JavaScript, you can set its expiry date to a date in the past. This will cause the cookie to expire immediately, effectively deleting it.

```
document.cookie = "name=; expires=Thu, 04 Feb 2024 00:00:00 IST ; path=/;";
```

19) Explain Hoisting in JavaScript.

Hoisting in JavaScript is a behavior where variables and functions are moved at the top of their respective scopes during the compilation phase. This means that they can be accessed and used before they actually declared in the code, no matter where they are declared.

20) Why do we use the word "debugger" in JavaScript?

The word ***"debugger"*** in JavaScript refers to the ***built-in debugging tool*** which is available in most modern web browsers. This tool allow developers to pause the execution of their ***JavaScript code***, examine the state of variables and objects, and step through the code ***line-by-line*** to find and fix errors or bugs in their code.

The term ***"debugger"*** comes from the process of ***"debugging,"*** which is the act of identifying and fixing errors or bugs in computer code. The term ***"debugging"*** refers to the process of finding and fixing errors in code.

21) Difference between var and let keywords in JavaScript.

In JavaScript, ***var*** and ***let*** both are used to declare variables, but they have some important differences in ***how they work***:

1). Scoping rules:

i) var is *function-scoped*, which means that a variable declared with ***var*** is accessible within the entire function it was declared in, regardless of where it was declared.

ii) let is *block-scoped*, which means that a variable declared with ***let*** is only accessible within the block. ***(i.e. between the curly braces {})*** it was declared in, including any nested blocks.

2). Hoisting:

i) Variables declared with ***var*** are hoisted to the top of their scope, which means that they can be accessed before they are declared, but their value will be ***undefined*** until they are assigned a value.

ii) Variables declared with ***let*** are not hoisted, which means that they cannot be accessed before they are declared.

3). Re-assignment:

i) Variables declared with ***var*** can be ***re-assigned*** to a new value within their scope.

ii) Variables declared with ***let*** can also be ***re-assigned***, but they cannot be ***re-declared*** within the same scope.

22) Explain Implicit Type Coercion in JavaScript.

In JavaScript, ***type coercion*** refers to the automatic conversion of a value from one data type to another data type. Implicit type coercion occurs when this conversion happens automatically, without the programmer explicitly specifying it in the code.

Implicit type coercion can happen in different contexts in JavaScript, but some common examples are:

1). String concatenation:

When a string is concatenated with another value that is not a string, JavaScript will implicitly convert the non-string value to a string before concatenating them together. For example, **"Hello" + 123** will be coerced to **"Hello123"**

2). Comparison operators:

When two values of different data types are compared using the **==** or != operators, JavaScript will implicitly convert one or both of the values to a common data type before making the comparison.

For example, **"123"** == **123** will be coerced to ***true***, because the string **"123"** is implicitly converted to the number **123** before the comparison.

23) JavaScript is a statically typed or a dynamically typed language?

JavaScript is a ***dynamically typed language***. It means that the ***data types*** of variables are determined at runtime, rather than being explicitly declared or enforced by the programmer. In JavaScript, a variable can hold a ***value*** of any ***data type***, and the data type of a variable can change dynamically based on the value assigned to it.

24) Explain passed by value and passed by reference.

In programming, when we pass a variable as an argument to a function, the value of the variable can be passed in two ways: ***passed by value*** and ***passed by reference***.

1). Passed by value:

When a variable is passed by value, a copy of the variable's value is made and this value is passed to the function. It means that any changes made to the variable within the function will not affect the original variable outside of the function.

In JavaScript, ***primitive data types*** (such as numbers, strings, and booleans) are typically passed by value. Here's an example:

```
function increment(num) {
  num += 1;
}

let x = 5;
increment(x);
console.log(x);

// Output: 5
```

2). Passed by reference:

When a variable is passed by reference, a reference to the variable's memory location is passed to the function. It means that any changes made to the variable within the function also affect the original variable outside of the function.

In JavaScript, ***objects*** (including arrays and functions) are typically passed by reference. Here's an example:

```
function addItem(arr, item) {
  arr.push(item);
}
let myArray = [1, 2, 3];
addItem(myArray, 4);

console.log(myArray);

// Output: [1, 2, 3, 4]
```

25) Explain the Immediately invoked function expression JavaScript example.

An ***Immediately Invoked Function (IIFE)*** in JavaScript is a function which is executed immediately after it is defined. The purpose of an ***Immediately Invoked Function (IIFE)*** is to create a new scope for the function and to encapsulate its variables and functions to prevent them from polluting the global namespace.

```
(function() {
  // Code to be executed here
})();
```

26) What do you mean by strict mode in JavaScript and characteristics of JavaScript strict-mode

Strict Mode was a new feature in ***ECMAScript5*** which allows us to place a program, or a function, in a ***"strict"*** operating context. This strict context prevents certain actions from being taken and throws more exceptions. The statement ***"use strict";*** instructs the browser to use the Strict mode, which is a reduced and safer feature set of JavaScript.

Using Strict mode for the entire script: To invoke strict mode for an entire script, put the exact statement ***"use strict";* (or** *'use strict';***)** before any other statements.

Whole-script strict mode syntax

```
'use strict';
let strickCode = 'strict mode script!';
```

Characteristics of JavaScript 'Strict Mode'

i) Variable Declaration: In strict mode, all variables must be declared using ***var***, ***let***, or ***const keywords*** before they are used. It helps to catch undeclared variables, which can lead to unexpected behavior.

ii) Assignment to Read-Only Properties: In strict mode, attempting to assign a value to a ***read-only property***, such as a global object or a variable declared with const, will result in an error.

iii) Deleting Variables, Functions, or Function Arguments: Deleting variables, functions, or function arguments is not allowed in strict mode.

iv) Duplicate Parameter Names: In strict mode, defining a function with duplicate parameter names is not allowed.

v) Octal Literal Syntax: Octal literals (e.g., 0123) are not allowed in strict mode.

vi) With Statement: The use of the with statement is not allowed in strict mode. The with statement can make code less predictable and hard to optimize.

vii) Reserved Words as Variable Names: Using certain reserved words as variable names, such as eval or arguments, is not allowed in strict mode.

viii) This Binding in Functions: In strict mode, the ***this*** value is ***undefined*** in functions that are not methods or constructors.

27) Higher-order functions JavaScript example

Higher Order Functions (HOF) are functions which takes one or more functions as arguments or return a function as a result.

In JavaScript, functions are first-class citizens, which means that they can be assigned to variables, passed as arguments, and returned from functions.

Here are some examples of Higher Order Functions in JavaScript:

1). Map: The map function is used to transform an array by applying a function to each element of the array and creating a new array with the results.

```
const arr = [1, 2, 3, 4, 5];
const newArr = arr.map((num) => num * 2);
console.log(newArr);

//Output: [2, 4, 6, 8, 10]
```

2). Filter: The filter function is used to create a new array with all elements that pass the test implemented by the provided function.

```
const arr = [1, 2, 3, 4, 5];
const newArr = arr.filter((num) => num % 2 === 0);
console.log(newArr);

// Output: [2, 4]
```

3). Reduce: The reduce function is used to reduce an array to a single value by applying a function to each element of the array.

```
const arr = [1, 2, 3, 4, 5];
const sum = arr.reduce((acc, num) => acc + num, 0);
console.log(sum);

//Output: 15
```

4). Function as an argument

```
function higherOrderFunction(func) {
  console.log("Before calling the argument function");
  func();
  console.log("After calling the argument function");
}

function sayHello() {
  console.log("Hello!");
}

higherOrderFunction(sayHello);

// Output
    Before calling the argument function
    Hello!
    After calling the argument function
```

5). Function as a return value:

```
function greet() {
  return function() {
    console.log("Hello World!");
  };
}

const greeting = greet();
greeting();

//Output: Hello World!
```