Top 54 JavaScript Coding Interview Questions 2024

Link: https://akdeepknowledge.com/javascript-interview-coding-questions/

JavaScript Interview Coding Questions

JavaScript Interview Questions

Interview Questions

1. What do you understand about JavaScript?

JavaScript is a versatile programming language is mainly utilized to enhance the interactivity of web pages. Unlike static HTML and CSS, JavaScript adds dynamic elements to websites, enabling features like interactive forms, animations, and real-time updates without reloading the page.

---

2. What's the difference between JavaScript and Java?

Aspect	JavaScript	Java
Type	Scripting language	Object-oriented programming language
Platform	Client-side scripting, runs in browsers	Platform-independent, runs on Java Virtual Machine
Syntax	C-like syntax	C/C++-like syntax
Usage	Web development, client-side scripting	General-purpose programming, server-side applications
Origin	Developed by Netscape in 1995	Developed by Sun Microsystems in 1995

---

3. What are the different types of data available in JavaScript?

JavaScript supports several data types following.

- **Primitive types:** `number`, `string`, `boolean`, `null`, `undefined`, `symbol`.
- **Reference types:** `object` (arrays, functions, objects).

4. What are the features of JavaScript?

JavaScript offers a wide range of features are.

- Dynamic typing
- Prototypal inheritance
- First-class functions
- Closures
- Asynchronous programming with Promises
- Event-driven programming

---

5. What benefits does JavaScript offer compared to other web technologies?

Advantages of JavaScript

- Cross-platform compatibility
- Rich interface and user experience
- Reduced server load with client-side processing
- Enhanced interactivity and responsiveness
- Large ecosystem with numerous libraries and frameworks

---

6. How can an object be created in JavaScript?

In JavaScript, you can create an object using either object literals or constructor functions.

```
// Using object literals
let myObject = {
  key1: value1,
  key2: value2,
  // Additional key-value pairs
};
// Using constructor functions
function Person(name, age) {
  this.name = name;
  this.age = age;
}
let person1 = new Person('John', 30);
```

7. How can an array be created in JavaScript?

In JavaScript, arrays can be created using array literals or the Array constructor.

```
// Using array literals
let myArray = [element1, element2, ...];

// Using the Array constructor
let myArray = new Array(element1, element2, ...);
```

---

8. What are some of the pre-existing methods available in JavaScript?

JavaScript provides a variety of built-in methods for different data types and objects. Some common ones include:

- **Array:** `push()`, `pop()`, `shift()`, `unshift()`, `splice()`
- **String:** `charAt()`, `toUpperCase()`, `toLowerCase()`, `split()`, `substring()`
- **Object:** `hasOwnProperty()`, `toString()`, `valueOf()`
- **Math:** `random()`, `round()`, `floor()`, `ceil()`, `sqrt()`

---

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

In JavaScript, variables can have either global or local scope. Global variables are accessible throughout the entire program, while local variables are limited to the function in which they are declared.

---

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

The "this" keyword in JavaScript refers to the object to which it belongs. Its value is determined by how a function is called and can vary depending on the context of the function invocation.

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

In JavaScript, variable names follow certain conventions.

- Begin with a letter, underscore, or dollar sign.
- Variables can contain letters, digits, underscores, and dollar signs in JavaScript.
- Cannot use reserved words or keywords.

---

12. What is Callback in JavaScript?

A callback in JavaScript is a function passed as an argument to another function, which is then executed after a certain event or operation. Callbacks are commonly used in asynchronous programming to handle tasks such as fetching data from a server or executing code after a timer expires.

---

13. How do you debug a JavaScript code?

JavaScript code can be debugged using various tools and techniques are following.

- Browser developer tools like Chrome DevTools or Firefox Developer Tools, which offer features like breakpoints, stepping through code, and inspecting variables.
- `console.log()` statements to log information to the browser console.
- Debugging tools provided by integrated development environments (IDEs) like Visual Studio Code or WebStorm.

---

14. What distinguishes a Function declaration and Function expression?

Aspect	Function Declaration	Function Expression
Syntax	Starts with the `function` keyword followed by a name	Can be named or anonymous, starts with `function` keyword
Hoisting	Entire function is hoisted to the top of the scope	Only the variable declaration is hoisted, not the function
Usage	Can be called before its declaration	Cannot be called before its definition
Example	`function myFunction() { /* function body */ }`	`const myFunction = function() { /* function body */ };`

---

15. How can you include JavaScript code in an HTML file?

JavaScript code can be added to an HTML file in several ways.

- **Inline:** Using the `<script>` tag directly in the HTML file.
- **External file:** Linking an external JavaScript file using the `<script>` tag's `src` attribute.
- **Event attributes:** Adding JavaScript code directly to HTML elements using event attributes like `onclick` or `onload`.

---

16. What do you understand about cookies?

Cookies are tiny bits of information that websites store on the user's browser. They are commonly used for tracking user sessions, storing user preferences, and implementing features like shopping carts in e-commerce websites.

17. How would you create a cookie?

In JavaScript, you can create a cookie by setting a value to `document.cookie`.

Here's an example.

```
document.cookie = "username=John";
This code snippet sets a cookie named "username" with the value "John".
```

18. How would you read a cookie?

You can read a cookie using the `document.cookie` property.

Here's an example of reading the "username" cookie:

```
let username = document.cookie.split('; ').find(row => row.startsWith('username')).split('=')[1];

This retrieves the value of the "username" cookie.
```

19. How would you delete a cookie?

To delete a cookie, you can set its expiration date to a past date.

Here's an example.

```
document.cookie = "username=John; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
This sets the "username" cookie to expire immediately, effectively deleting it.
```

20. What's the difference between `let` and `var`?

Aspect	`let`	`var`
Scope	Block scope (limited to the block where it's defined)	Function scope (limited to the function where it's defined)
Hoisting	Not hoisted	Hoisted to the top of its scope
Re-declaration	Cannot be re-declared within the same scope	Can be re-declared within the same scope
Example	`let x = 10;`	`var y = 20;`

---

21. What are Closures in JavaScript?

Closures in JavaScript refer to the ability of a function to retain access to its surrounding scope even after the function has finished executing. This allows the function to access and manipulate variables from its enclosing scope.

22. What are the arrow functions in JavaScript?

Arrow functions provide a more concise way to express function expressions in JavaScript, making the code easier to read and write. They have a more compact syntax compared to traditional function expressions and lexically bind the `this` value, meaning they inherit the `this` value from the surrounding code.

Example of an arrow function

```
javascriptCopy code
const add = (a, b) => a + b;
```

23. What are the various methods to access an HTML element in JavaScript code?

HTML elements can be accessed in JavaScript using various methods:

- **By ID:** `document.getElementById()`
- **By class name:** `document.getElementsByClassName()`
- **By tag name:** `document.getElementsByTagName()`
- **Using CSS selectors:** `document.querySelector()` or `document.querySelectorAll()`
- **By name attribute:** `document.getElementsByName()`

---

24. What are the methods for defining a variable in JavaScript?

In JavaScript, you can define variables using the **var**, **let**, or **const** keywords.

- **`var`** is function-scoped and can be re-declared and reassigned.
- **`let`** is block-scoped, can be reassigned, but not re-declared in the same scope.
- **`const`** is also block-scoped but cannot be reassigned or re-declared.

---

25. What are Imports and Exports in JavaScript?

Imports and exports are part of the **ES6 module** system in JavaScript, allowing developers to organize code into separate files and reuse functionality across multiple files.

- **`import`** is used to import functionalities from other modules.
- **`export`** is used to export functionalities from a module to make them available for other modules to import.

---

26. What is the differences between Document and Window in JavaScript?

Aspect	Document	Window
Scope	Document object model (DOM)	Browser window object
Access	Represents the content of a webpage	Represents the browser window itself
Properties	Contains properties like body, title, URL	Contains properties like location, history, navigator
Methods	Contains methods like getElementById(), querySelector()	Contains methods like alert(), open(), close()

---

27. What are some of the JavaScript frameworks and their purposes?

JavaScript frameworks like React, Angular, and Vue.js are popular for building modern web applications. They provide tools and libraries to simplify development tasks, handle complex user interfaces, and manage application state efficiently.

---

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

Aspect	Undefined	Undeclared
Definition	Variable is declared but not initialized	Variable is not declared or defined
Example	`let x; console.log(x); // undefined`	`console.log(y); // ReferenceError: y is not defined`

29. What is the differences between Undefined and Null in JavaScript?

Aspect	Undefined	Null
Definition	Variable has been declared but not assigned a value	Variable explicitly assigned the value `null`
Example	`let x; console.log(x); // undefined`	`let y = null; console.log(y); // null`

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

Aspect	Session Storage	Local Storage
Scope	Limited to the current session	Persists even after the browser is closed
Expiration	Cleared when the session ends	Persists until explicitly cleared
Storage limit	Typically larger than cookies	Typically larger than cookies

31. What are the different data types that exist in JavaScript?

JavaScript supports several data types, including.

- **Primitive types:** `number`, `string`, `boolean`, `null`, `undefined`, `symbol`
- **Reference types:** `object` (arrays, functions, objects)

---

32. What is the 'this' keyword in JavaScript?

The "this" keyword in JavaScript refers to the object to which it belongs. Its value is determined by how a function is called and can vary depending on the context of the function invocation.

---

33. What is the difference between Call and Apply? (explain in detail with examples)

Aspect	Call Method	Apply Method
Syntax	`function.call(thisArg, arg1, arg2)`	`function.apply(thisArg, [argsArray])`
Arguments	Accepts individual arguments	Accepts arguments as an array
Example	`func.call(obj, arg1, arg2)`	`func.apply(obj, [arg1, arg2])`

---

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

In JavaScript, variables have either global or local scope. Global variables are accessible throughout the entire program, while local variables are limited to the function in which they are declared.

---

35. What are the arrow functions in JavaScript?

Arrow functions provide a concise method for writing function expressions in JavaScript. They provide a shorter syntax compared to traditional function expressions and lexically bind this value, making it easier to access the surrounding context.

---

36. Explain Hoisting in JavaScript. (with examples)

Hoisting is a mechanism in JavaScript where variables and function declarations are moved to the top of their containing scope during the compile phase. This enables variables and functions to be utilized before they are declared.

Example

```
console.log(x); // Output: undefined var x = 5;
```

In the above example, the variable x is hoisted to the top of its scope, but its value is not assigned until the line where it's declared. Therefore, `console.log(x)` outputs `undefined`.

---

37. Difference between "==" and "===" operators (with examples)

Aspect	== Operator	=== Operator
Equality Check	Checks for equality after type coercion	Checks for strict equality (no type coercion)
Example	`0 == '0' // true`	`0 === '0' // false`

---

38. Difference between var and let keyword

Aspect	var Keyword	let Keyword
Scope	Function-scoped	Block-scoped
Hoisting	Hoisted to the top of the function	Not hoisted
Re-declaration	Can be re-declared within the same scope	Cannot be re-declared within the same scope

---

39. Implicit Type Coercion in JavaScript (in detail with examples)

Implicit type coercion in JavaScript refers to the automatic conversion of one data type to another during operations.

Example

```
console.log(1 + '2'); // Output: '12' (number is coerced to string)
console.log('2' - 1); // Output: 1 (string is coerced to number)
```

In the first example, the number `1` is coerced into a string and concatenated with `'2'`. In the second example, the string `'2'` is coerced into a number and subtracted from `1`.

---

40. Is JavaScript a statically typed or a dynamically typed language?

JavaScript is a dynamically typed language. This means that variables in JavaScript can hold values of any data type, and their data types are determined at runtime rather than at compile time.

---

41. NaN property in JavaScript

NaN stands for "Not-a-Number" and is a special value in JavaScript used to represent an unrepresentable value resulting from an arithmetic operation. It is returned when a mathematical operation is performed on a non-numeric value.

Example

```
console.log(0 / 0); // Output: NaN
```

---

42. Passed by value and passed by reference

In JavaScript, primitive data types (like numbers and strings) are passed by value, while objects and arrays are passed by reference.

- **Passed by value:** A copy of the variable's value is passed to the function.
- **Passed by reference:** A reference to the variable's memory location is passed to the function.

Example

```
let num = 10;
function changeNum(val) {
  val = 20;
}
changeNum(num);
console.log(num); // Output: 10 (unchanged)

let arr = [1, 2, 3];
function changeArr(arr) {
  arr.push(4);
}
changeArr(arr);
console.log(arr); // Output: [1, 2, 3, 4] (modified)
```

In the first example, `num` remains unchanged because it's passed by value. In the second example, `arr` is modified because it's passed by reference.

---

43. Immediately Invoked Function in JavaScript

An Immediately Invoked Function Expression (IIFE) is a JavaScript function that is executed immediately after it is defined. It is typically used to create a private scope and avoid polluting the global namespace with variables.

Example:

```
(function() { // IIFE body console.log('This is an IIFE'); })();
```

In this example, the function is defined and immediately invoked using parentheses ( ) around the function declaration.

44. Characteristics of JavaScript strict mode

JavaScript strict mode is a feature that enforces stricter parsing and error handling in JavaScript. Some characteristics of strict mode following.

- Disallows the use of undeclared variables.
- Throws errors for assignments to non-writable properties.
- Prevents the use of `with` statement.
- Disallows the use of `arguments.callee` and `arguments.caller`.
- Throws errors for duplicate property names in objects.

Strict mode is enabled by adding the directive `'use strict';` at the beginning of a script or function.

45. Higher Order Functions (with examples)

Higher Order Functions in JavaScript are functions that either take other functions as arguments or return functions as results.

Example

```
// Function taking another function as argument
function higherOrderFunction(callback) {
  callback();
}

// Function returning another function
function createMultiplier(factor) {
  return function(num) {
    return num * factor;
  };
}

// Usage of higher order functions
higherOrderFunction(function() {
  console.log('This is a callback function');
});

const double = createMultiplier(2);
console.log(double(5)); // Output: 10
```

In this example, `higherOrderFunction` takes a callback function as an argument, and `createMultiplier` returns a function that multiplies its argument by a factor.

---

46. Self-Invoking Functions

A self-invoking function, also known as an immediately-invoked function expression (IIFE), is a function that is automatically executed as soon as it is defined. It is wrapped inside parentheses ( ) to make it an expression and then immediately invoked using an additional pair of parentheses ( ).

Example.

```
(function() { console.log("This is a self-invoking function!"); })();
```

In this example, the function is defined and invoked immediately, printing the message "This is a self-invoking function!" to the console.

---

47. Difference between exec() and test() methods

The `exec()` and `test()` methods are both methods of the JavaScript `RegExp` object used for working with regular expressions.

`exec()`: Executes a search for a match in a specified string. If a match is found, it returns an array containing the matched text, along with information about the match. If no match is found, it returns `null`.

Example

```
let str = "Hello World!";
let pattern = /Hello/;
console.log(pattern.exec(str)); // Output: ["Hello", index: 0, input: "Hello World!", groups: undefined]
```

`test()`: Tests for a match in a string. It returns `true` if a match is found, otherwise `false`.

Example

```
let str = "Hello World!";
let pattern = /Hello/;
console.log(pattern.test(str)); // Output: true
```

In summary, **`exec()`** returns details about the match, while **`test()`** simply returns a boolean indicating whether a match exists.

---

48. Currying in JavaScript (with examples)

Currying is a technique in functional programming where a function with multiple arguments is transformed into a sequence of functions, each taking a single argument. This allows for partial application of the function, which can be useful for creating reusable and composable functions.

Example.

```
// Without currying
function add(a, b, c) {
  return a + b + c;
}
console.log(add(1, 2, 3)); // Output: 6

// With currying
function curryAdd(a) {
  return function(b) {
    return function(c) {
      return a + b + c;
    }
  }
}
console.log(curryAdd(1)(2)(3)); // Output: 6
```

In the curried version, `curryAdd` returns a series of functions, each taking one argument and returning another function until all arguments are collected and the final result is calculated.

---

49. Advantages of using External JavaScript

Using external JavaScript files has several advantages.

- **Separation of concerns:** Keeps HTML and JavaScript code separate, making code organization and maintenance easier.
- **Caching:** External JavaScript files can be cached by the browser, reducing page load times for subsequent visits.
- **Reuse:** External JavaScript files can be reused across multiple HTML files, promoting code reusability.
- **Improved maintainability:** Changes to JavaScript code can be made in a single file, which reflects across all HTML files that reference it.

---

50. What are object prototypes?

In JavaScript, every object has a prototype, which serves as a template for the object. The prototype contains properties and methods that are accessible to all instances of the object. When a property or method is accessed on an object, JavaScript first checks if the object has that property or method. If not found, it looks up the prototype chain until it finds it.

Example.

```
// Creating an object with a prototype
let animal = {
  type: 'Animal',
  sound: function() {
    console.log('Makes a sound');
  }
};

// Creating an instance of the object
let cat = Object.create(animal);
cat.type = 'Cat';

console.log(cat.type); // Output: 'Cat'
cat.sound(); // Output: 'Makes a sound'
```

In this example, `cat` inherits the properties and methods from the `animal` prototype object.

51. Types of errors in JavaScript

JavaScript errors can be categorized into several types.

- **Syntax errors:** Occur when the JavaScript engine encounters code that violates the language syntax rules. These errors prevent the code from executing and are typically detected during the parsing phase.
- **Runtime errors:** Occur during the execution of the code when an operation cannot be performed as expected. Examples include accessing undefined variables, type errors, and division by zero.
- **Logical errors:** Occur when the code does not produce the expected output due to incorrect logic or algorithmic mistakes. These errors are often more subtle and may not result in immediate failures or exceptions.
- **Reference errors:** Occur when trying to access a variable or function that is not defined or is out of scope. This typically happens when attempting to access properties of undefined or null values.

---

52. What is memorization?

Memorization is an optimization technique used in computer programming to speed up function execution by caching the results of expensive function calls and returning the cached result when the same inputs occur again.

Example

```
function memoize(func) {
  let cache = {};
  return function(...args) {
    let key = JSON.stringify(args);
    if (!(key in cache)) {
      cache[key] = func.apply(this, args);
    }
    return cache[key];
  };
}

// Example function to be memoized
function expensiveOperation(n) {
  console.log('Performing expensive operation...');
  return n * 2;
}
```

```
let memoizedOperation = memoize(expensiveOperation);

console.log(memoizedOperation(5)); // Output: Performing expensive operation... 10
console.log(memoizedOperation(5)); // Output: 10 (cached result)
```

In this example, the `expensiveOperation` function's results are cached based on the input argument `n`, and subsequent calls with the same argument retrieve the cached result instead of recomputing it.

---

53. Recursion in a programming language

Recursion is a programming technique where a function calls itself to solve a problem. This process continues until a base case is reached, which stops the recursive calls. Recursion is particularly useful for solving problems that can be broken down into smaller, similar subproblems.

Example of a recursive function to calculate factorial.

```
function factorial(n) {
  if (n === 0 || n === 1) {
    return 1; // Base case: factorial of 0 or 1 is 1
  } else {
    return n * factorial(n - 1); // Recursive call
  }
}

console.log(factorial(5)); // Output: 120
```

In this example, the `factorial` function calls itself with a smaller input (`n - 1`) until it reaches the base case (`n === 0` or `n === 1`), at which point it returns 1.

---

54. Use of a constructor function (with examples)

Constructor functions in JavaScript are used to create multiple instances of objects with similar properties and methods. They are typically defined with a capitalized name to distinguish them from regular functions.

Example of a constructor function to create objects representing cars.

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}

// Creating instances of Car objects
let car1 = new Car('Toyota', 'Corolla', 2020);
let car2 = new Car('Honda', 'Civic', 2018);

console.log(car1); // Output: Car { make: 'Toyota', model: 'Corolla', year: 2020 }
console.log(car2); // Output: Car { make: 'Honda', model: 'Civic', year: 2018 }
```

In this example, the `Car` constructor function is used to create two instances (`car1` and `car2`) of `Car` objects with different properties.

https://akdeepknowledge.com

Updated Questions 👇

Updated Post Link: https://akdeepknowledge.com/javascript-interview-coding-questions/

More Interview Questions 👇

- Python: https://akdeepknowledge.com/python-programming-interview-questions/
- SQL: https://akdeepknowledge.com/important-sql-interview-questions/
- Linux: https://akdeepknowledge.com/linux-commands-for-devops-interview-questions/
- Data Bricks: https://akdeepknowledge.com/databricks-interview-questions-and-answers/

Article And Free Course Tutorials 👇

Visit Blog Website: https://akdeepknowledge.com

Visit Coding Website:

Follow on My social media 👇

Subscribe YouTube Channel: https://www.youtube.com/@akdeepknowledge/videos

Facebook Page: https://www.facebook.com/ArsalanKhatri184

LinkedIn Group: https://www.linkedin.com/groups/9305818/

SlideShare: https://www.slideshare.net/ArsalanKhatri4

Medium: https://medium.com/@Arsalan_Khatri

CodePen: https://codepen.io/AK-Deep-Knowledge

GitHub: https://github.com/arsalankhatri184