TOP 50 JAVASCRIPT INTERVIEW QUESTIONS

With Example Answers

GRAPHICODAR

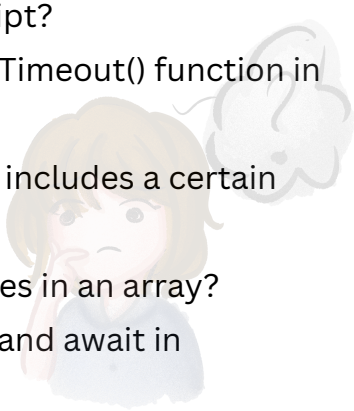
TOP 50 JAVASCRIPT INTERVIEW QUESTIONS WITH EXAMPLE ANSWERS

Review these common JavaScript interview questions and answers and practice your coding fundamentals with this guide for your next interview.

Preparing for a JavaScript interview requires a lot of work. It's important to be well-known in the fundamentals but you also should have some grasp on how to debug JavaScript code, what some of the advanced functions are and how to build projects in it.

COMMON JAVASCRIPT INTERVIEW QUESTIONS

1. What are the different data types in JavaScript?
2. What is hoisting in JavaScript?
3. What is the difference between null and undefined?
4. What are closures in JavaScript?
5. What is a callback function in JavaScript?
6. What are promises in JavaScript?
7. What is the purpose of the setTimeout() function in Javascript?
8. How can you check if an array includes a certain value?
9. How can you remove duplicates in an array?
10. What is the purpose of async and await in JavaScript?



JAVASCRIPT FUNDAMENTALS

1. What is JavaScript?

A high-level, interpreted programming language called JavaScript makes it possible to create interactive web pages and online apps with dynamic functionality. Commonly referred to as the universal language, Javascript is primarily used by developers for front-end and back-end work.

2. What are the different data types in JavaScript?

Primitive data types can store only a single value. To store multiple and complex values, **non-primitive data types** are used.

Note: To know the type of a JavaScript variable, we can use the **typeof** operator.

JavaScript has six primitive data types:

- Number
- String
- Boolean
- Null
- Undefined
- Symbol

It also has two compound data types:

- Object
- Array

3. What is hoisting in JavaScript?

Hoisting is a JavaScript concept that refers to the process of moving declarations to the top of their scope. This means that variables and functions can be used before they are declared, as long as they are declared before they are used in a function.

For example, the following code will print **"Hello, world!"** even though the function is declared after calling function.

```
hoistedFunction();

// Outputs " Hello world! " even  when the function is declared after calling function

 hoistedFunction(){
  console.log(" Hello world! ");
 }
```

4. What is the difference between null and undefined?

null is an assignment value that represents no value or an empty value, while **undefined** is a variable that has been declared but not assigned a value.

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

var y = null;
console.log(y);   // Output : null
```

5. Why do we use the word “debugger” in JavaScript?

The word “**debugger**” is used in JavaScript to refer to a tool that can be used to step through JavaScript code line by line. This can be helpful for debugging JavaScript code, which is the process of finding and fixing errors in JavaScript code.

To use the **debugger**, you need to open the JavaScript console in your browser. Then, you can use **debugger** commands to comb through your code line by line.

It's essential to know debugging techniques as well as the more general ideas behind code optimization and speed improvement. In addition to operating smoothly, efficient code significantly enhances the user experience.

For example, the following code will print the value of the x variable at each step of the **debugger**.

```
var x = 10;
debugger;
x = x + 1;
debugger;
console.log(x);
```

6. What is the purpose of the “this” keyword in JavaScript?

The **this** keyword refers to the object that is executing the current function or method.

It allows access to object properties and methods within the context of that object.

```
const person ={
   name : “Jonh”,
   greet : funtion(){
      console.log(“Hello, “ + this.name);
   }
}

person.greet();  // Output : Hello, Jonh
```

7. What is the difference between == and === operators in JavaScript?

The equality **==** operator is a comparison operator that compares two values and returns true if they are equal. The strict equality **===** operator is also a comparison operator, but it compares two values and returns true only if they are equal and of the same type.

For example, the following code will return true, because the values of the x and y variables are equal.

```
var x = 10;
var y = 10;
console.log(x == y);  // Output : true
console.log(x === y); // Output : false
```

8. What is the difference between “var” and “let” keywords in JavaScript?

The **var** and **let** keywords are both used to declare variables in JavaScript. However, there are some key differences between the two keywords.

The **var** keyword declares a global variable, which means that the variable can be accessed from anywhere in the code. The **let** keyword declares a local variable, which means that the variable can only be accessed within the block of code where it is declared.

```
{
  let x = 10;
  console.log(x);  // Output : 10
}
```

9. What are closures in JavaScript?

Closures (**closureFn**) are functions that have access to variables from an outer function even after the outer function has finished executing. They "remember" the environment in which they were created.

```
function outer(){
 var outerVar = "Hello";
 function inner(){
   console.log(outerVar):
 }
 return inner;
}
var clouserFn = outer();
clouserFn();   //Output : Hello
```

10. What is event delegation in JavaScript?

Event delegation is a technique where you attach a single event listener to a parent element, and that event listener handles events occurring on its child elements. It helps optimize performance and reduce memory consumption.

11. What is the difference between “let”, “const”, and “var”?

let and **const** were introduced in ES6 and have block scope. **let** is reassignable, and **const** is non-reassignable. **var** is function-scoped and can be redeclared and reassigned throughout the function.

```
let x = 5;
x = 10;
console.log(x);  //Output : 10

const y  = 5;
y = 10;    //Error: Assignment to constant variable
console.log(y);

var x = 5;
var x = 10;
console.log(x);   //Output : 10
```

12. What is implicit type coercion in JavaScript?

Implicit type coercion is a JavaScript concept that refers to the automatic conversion of a value from one type to another. In JavaScript, this conversion follows a priority order that typically begins with strings, then numbers, and finally Booleans. If you try to add a string to a number, JavaScript will implicitly coerce the number to a string before performing the addition operation because strings have the highest priority in type coercion.

For example, when you combine the number 5 with the string '10' using the addition operator, the result is the string '510'. This occurs because JavaScript will implicitly convert the number 5 to a string following the priority of coercion, and then concatenate it to the string '10'.

```
var x = 5;
var y = "10";
console.log(x + y);    //Output : "510"
```

13. Explain the concept of prototypes in JavaScript.

Prototypes are a mechanism used by JavaScript objects for inheritance. Every JavaScript object has a prototype, which provides properties and methods that can be accessed by that object.

```
function Person(name){
  this.name = name;
}

Person.prototype.greet : function(){
  console.log("Hello, " + this.name);
}

var person = new Person("Jonh");
person.greet();   //Output: Hello, Jonh
```

14. What is the output of the following code?

```
console.log(3 + 2 + “7”);
```

The output will be "**57**". The addition operation is performed from left to right, and when a string is encountered, it performs concatenation.

15. How can you clone an object in JavaScript?

There are multiple ways to clone an object in JavaScript. One common method is using the **Object.assign()** method or the **spread operator (...).**

```
const obj1 = {name: “Jonh”, age: 30};

// Using Object.assign()
const obj2 = Object.assign({},obj1);

// Using Spread Operator
const obj3 = {...obj1};

console.log(obj2);     //Output: { name: “Jonh”, age: 30}

console.log(obj3);     //Output: { name: “Jonh”, age:  30}
```

INTERMEDIATE CONCEPTS

16. What are higher-order functions in JavaScript?

Higher order functions are functions that can accept other functions as arguments or return functions as their results. They enable powerful functional programming patterns in JavaScript.

```
function multiplybyTwo(num){
  num * 2;
}
function applyOperation(num, operation){
 return operation(num);
}
const result = applyOperation(5, multiplybyTwo);
console.log(result);     //Output: 10
```

17. What is the purpose of the bind() method in JavaScript?

The **bind()** method is used to create a new function with a specified **this** value and an initial set of arguments. It allows you to set the context of a function permanently.

```
const person ={
   name : “Jonh”,
   greet : funtion(){
      console.log(“Hello, “ + this.name);
   }
}

const greetFn = person.greet;
const  boundFn = greetFn.bind(person);
boundFn();    // Output : Hello, Jonh
```

18. What is the difference between function declarations and function expressions?

Function declarations are defined using the function keyword, while **function expressions** are defined by assigning a function to a variable. Function declarations are hoisted, while function expressions are not.

```
 // Function Declaration
function multiply(a, b){
  return a * b;
}
// Function Expression
const add = function(a, b){
 return a + b;
}
```

19. What are the different types of errors in JavaScript?

JavaScript can throw a variety of errors, including:

- **Syntax errors:** These errors occur when the JavaScript code is not syntactically correct.
- **Runtime errors:** These errors occur when the JavaScript code is executed and there is a problem.
- **Logical errors:** These errors occur when the JavaScript code does not do what it is supposed to do.

---

20. What is memoization in JavaScript?

Memoization is a technique that can be used to improve the performance of JavaScript code.

Memoization works by storing the results of expensive calculations in a cache. This allows the JavaScript code to avoid re-performing the expensive calculations if the same input is provided again.

For example, the following code calculates the factorial of a number. The factorial of a number is the product of all the positive integers from one to the number.

```
function factorial(n){
  if(n === 0){
    return 1;
  }else{
    return n * factorial(n - 1);
  }
}
```

This code can be memoized as follows:

```
function factorial(n){
  if(factorialCache[n] !== undefined){
    return factorialCache[n];
  }else{
    factorialCache[n] = n * factorial(n - 1);
    return factorialCache[n];
  }
}
```

21. What is recursion in JavaScript?

Recursion is a programming technique that allows a function to call itself. Recursion can be used to solve a variety of problems, such as finding the factorial of a number or calculating the Fibonacci sequence.

The following code shows how to use recursion to calculate the factorial of a number:

```
function factorial(n){
  if(n === 0){
    return 1;
  }else{
    return n * factorial(n - 1);
  }
}
```

22. What is the use of a constructor function in JavaScript?

A **constructor function** is a special type of function that is used to create objects. Constructor functions are used to define the properties and methods of an object.

The following code shows how to create a constructor function:

```
function Person(name, age){
  this.name = name;
  this. age = age;
}
```

23. What is the difference between a function declaration and a function expression in JavaScript?

A **function declaration** is a statement that defines a function. A **function expression** is an expression that evaluates to a function.

The following code shows an example of a function declaration. This code defines a function named factorial. The factorial function calculates the factorial of a number.

```
function factorial(n){
  if(n === 0){
    return 1;
  }else{
    return n * factorial(n - 1);
  }
}
```

The following code shows an example of a function expression:

```
var factorial = function(n){
  if(n === 0){
    return 1;
  }else{
    return n * factorial(n - 1);
  }
}
```

24. What is a callback function in JavaScript?

A **callback function** is a function passed as an argument to another function, which is then invoked inside the outer function. It allows asynchronous or event-driven programming.

```
function fetchData(callback){
 setTimeOut( functio(){
    const data = “Some Data”;
    callback(data);
 }, 2000);
}
function processData(data){
  console.log(“Data Received: ” + data);
}
fetchData(processData);  //Output: after 2
seconds: Data Received: Some Data
```

25. What are promises in JavaScript?

Promises are objects used for asynchronous operations. They represent the eventual completion or failure of an asynchronous operation and allow chaining and handling of success or error cases.

```
function fetchData(){
  return new Promise(function(resolve, reject ){
    setTimeOut( functio(){
      const data = “Some Data”;
      callback(data);
    }, 2000);
  }
}
fetchData()
.then(function(data){
  console.log(“Data Received: ” + data);
})
.catch(function(error)){
  console.log(“Error Occured: ” + error);
});
)
```

26. What is the difference between synchronous and asynchronous programming?

In **synchronous programming**, the program execution occurs sequentially, and each statement blocks the execution until it is completed. In **asynchronous programming**, multiple tasks can be executed concurrently, and the program doesn't wait for a task to finish before moving to the next one.

Synchronous coding example:

```
console.log("Statement 1");
console.log("Statement 2");
console.log("Statement 3");
```

Asynchronous code example:

```
console.log("Statement 1");
 setTimeOut( functio(){
   console.log("Statement 2");
}, 2000);
console.log("Statement 3");
```

27. How do you handle errors in JavaScript?

Errors in JavaScript can be handled using **try-catch** blocks. The try block contains the code that may throw an error, and the catch block handles the error and provides an alternative execution path.

```
try{
 throw new Error("Something Went Wrong");
} catch(error){
console.log("Error Occured : " + error.message);
}
```

28. Explain the concept of event bubbling in JavaScript.

Event bubbling is the process where an event triggers on a nested element, and then the same event is propagated to its parent elements in the document object model (DOM) tree. It starts from the innermost element and goes up to the document root.

Example:

```
<div id="parent">
  <div id="Child">Click Me!</div>
</div>
```

When you click on the child element, both the child and parent event handlers will be triggered, and the output will be:

```
document.getElementById("child").addEventListe
ner("click", function(){
  console.log("Child clicked");
});


document.getElementById("parent").addEventLis
te ner("click", function(){
 console.log("Parent clicked");
});
```

29. What are arrow functions in JavaScript?

Arrow functions are a concise syntax for writing JavaScript functions. They have a more compact syntax compared to traditional function expressions and inherit the this value from their surrounding scope.

Example:

```
const multiply = (a, b) => a * b;
console.log(multiply(2, 3));    //Output: 6
const greet = name =>{
 console.log("Hello, " + name);
}
greet("Jonh");    //Output: Hello, Jonh
```

31. What is the purpose of the setTimeout() function in JavaScript?

The **setTimeout()** function is used to delay the execution of a function or the evaluation of an expression after a specified amount of time in milliseconds.

```
console.log("Start");
setTimeOut( functio(){
   console.log("Delayed");
}, 2000);
console.log("End");
```

Output after two seconds:

```
Start
End
Delayed
```

32. What is event delegation and why is it useful?

Event delegation is a technique where you attach a single event listener to a parent element to handle events occurring on its child elements. It's useful for dynamically created elements or when you have a large number of elements.

```
<ul id="mylist">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>


document.getElementById("mylist").addEventList
e  ner("click", function(event){
  if(event.target.nodeName === "LI"){
    console.log(event.target.textContent);
  }
});
```

33. How can you prevent the default behavior of an event in JavaScript?

You can use the **preventDefault()** method on the event object within an event handler to prevent the default behavior associated with that event.

```
<a href="#" id="myLink" >Click Me!<a/>

document.getElementById("myLink").addEventLi
ste  ner("click", function(event){
   event.target.preventDefault();
   console.log("Link clicked, but default behavior
prevented");
 });
```

34. What is the difference between localStorage and sessionStorage in JavaScript?

Both **localStorage** and **sessionStorage** are web storage objects in JavaScript, but they have different scopes and lifetimes.

- **localStorage** persists data even after the browser window is closed and is accessible across different browser tabs/windows of the same origin.
- **sessionStorage** stores data for a single browser session and is accessible only within the same tab or window.

```
localStorage.setItem("name", "Jonh");
console.log(localStorage.getItem("name"));

sessionStorage.setItem("name", "Jonh");
console.log(sessionStorage.getItem("name"));
```

35. How can you convert a string to lowercase in JavaScript?

You can use the **toLowerCase()** method to convert a string to lowercase in JavaScript.

```
const str = “Hello, World”;
console.log(str.toLowerCas());
 //Output: hello, world
```

ADVANCED CONCEPTS

36. What is the purpose of the map() function in JavaScript?

The **map()** function is used to iterate over an array and apply a transformation or computation on each element. It returns a new array with the results of the transformation.

```
const num= [1, 2, 3, 4, 5];
const squareNum = num.map(function(n)){
  return n * n;
}
console.log(squareNum);
 //Output: [1, 4, 9, 16, 25]
```

37. What is the difference between splice() and slice()?

- **splice()** is used to modify an array by adding, removing, or replacing elements at a specific position.
- **slice()** is used to create a new array that contains a portion of an existing array, specified by the starting and ending indices.

Example of splice():

```
const fruits= [”apple”, “banana”, “orange”];
fruits.splice(1, 1, “grape”);
// remove “banana” & add  “grape” at index 1
console.log(fruits);
 //Output: [”apple”, “grape”, “orange”]
```

Example of slice():

```
const num= [1, 2, 3, 4, 5];
const slicedNum = num.slice(1, 4);
// Extract elements from index 1 to 3
console.log(slicedNum);
 //Output: [2, 3, 4]
```

38. What is the purpose of the reduce() function in JavaScript?

The **reduce()** function is used to reduce an array to a single value by applying a function to each element and accumulating the result.

```
const num= [1, 2, 3, 4, 5];
const sum= num.reduce(function(acc, num){
  return acc + num;
}, 0);
console.log(sum);
 //Output: 15
```

39. How can you check if an array includes a certain value in JavaScript?

You can use the **includes()** method to check if an array includes a specific value. It returns true if the value is found, and false otherwise.

```
const fruits= [”apple”, “banana”, “orange”];
console.log(fruits.includes(“banana”));
//Output: true

console.log(fruits.includes(“grape”));
//Output: false
```

40. What is the difference between prototype and instance properties in JavaScript?

A **prototype** property is a property that is defined on the prototype object of a constructor function. **Instance** properties are properties that are defined on individual objects that are created by a constructor function.

Prototype properties are shared by all objects that are created by a constructor function. **Instance** properties are not shared by other objects.

41. What is the difference between an array and an object in JavaScript?

An **array** is a data structure that can store a collection of values. An **object** is a data structure that can store a collection of properties.

Arrays are indexed by numbers. Objects are indexed by strings. Arrays can only store primitive data types and objects. Objects can store primitive data types, objects and arrays.

42. How can you remove duplicates from an array in JavaScript?

One way to remove duplicates from an array is by using the Set object or by using the **filter()** method with the **indexOf()** method.

```
const num= [1, 2, 2, 3, 4, 4, 5];
const uniqueNum= [...new Set(num)]
console.log(uniqueNum);
```

43. What is the purpose of the fetch() function in JavaScript?

The **fetch()** function is used to make asynchronous HTTP requests in JavaScript. It returns a Promise that resolves to the response from the server.

Example:

```
fetch(URL)
.then(function(response){
 return response.json();
})
.then(function(data){
 console.log(data);
})
.catch(function(error){
  console.log("Error Occured: " + error);
});
```

44. What is a generator function in JavaScript?

A **generator function** is a special type of function that can be paused and resumed during its execution. It allows generating a sequence of values over time, using the **yield** keyword.

```
function generateNum(){
 yield 1;
 yield  2;
 yield 3;
}
const generator = generateNum();
console.log(generator.next().value);    //Output: 1
console.log(generator.next().value);    //Output: 2
console.log(generator.next().value);    //Output: 3
```

45. What are the different events in JavaScript?

There are many different events in JavaScript, but some of the most common events include:

- **Click:** The click event occurs when a user clicks on an HTML element.
- **Mouseover:** The mouseover event occurs when a user's mouse pointer moves over an HTML element.
- **Keydown:** The keydown event occurs when a user presses a key on the keyboard.
- **Keyup:** The keyup event occurs when a user releases a key on the keyboard.
- **Change:** The change event occurs when a user changes the value of an HTML input element.

46. What are the different ways to access an HTML element in JavaScript?

There are three main ways to access an HTML element in JavaScript:

1. Using **the getElementById() method:** The getElementById() method takes a string as an argument and returns the HTML element with the specified ID.
2. Using **the getElementsByTagName() method:** The getElementsByTagName() method takes a string as an argument and returns an array of all the HTML elements with the specified tag name.
3. Using **the querySelector() method:** The querySelector() method takes a CSS selector as an argument and returns the first HTML element that matches the selector.

47. What is the scope of a variable in JavaScript?

The **scope of a variable** in JavaScript is the part of the code where the variable can be accessed. Variables declared with the var keyword have a local scope, which means that they can only be accessed within the block of code where they are declared. Variables declared with the let keyword have a block scope, which means that they can only be accessed within the block of code where they are declared and any nested blocks. Variables declared with the const keyword have a global scope, which means that they can be accessed from anywhere in the code.

48. What are the different ways to create objects in JavaScript?

In JavaScript, there are several ways to declare or construct an object.

1. Object.
2. using Class.
3. create Method.
4. Object Literals.
5. using Function.
6. Object Constructor

49. What is the purpose of the window object in JavaScript?

The **window object** represents the browser window. The window object can be used to access the browser's features, such as the location bar, the status bar and the bookmarks bar.

50. What is the purpose of the async and await keywords in JavaScript?

The **async** and **await** keywords are used for handling asynchronous operations in a more synchronous-like manner.

The async keyword is used to define an asynchronous function, and the await keyword is used to pause the execution of an async function until a promise is fulfilled or rejected.

```
async function fetchData(){
try{
 const response = await fetch(URL);
 const data = await response.json();
 console.log(data);
}catch(error){
 console.log("Error Occured: " + error);
};
}
fetchData();
```

HOW TO PREPARE FOR A JAVASCRIPT INTERVIEW

In order to ace a JavaScript interview, you need to be ready for anything. It's important to practice your code, but you should also be able to clearly explain how different functions work, have real world experience working in JavaScript and understand how to debug.

7 WAYS TO PREPARE FOR A JAVASCRIPT INTERVIEW

1. Review JavaScript fundamentals.
2. Master key concepts.
3. Study common interview questions.
4. Master debugging skills.
5. Practice coding.
6. Build projects.
7. Mock interviews.

Fortunately, there are some basic steps you can take to be prepared and stand out from other applicants.

1. Review JavaScript Fundamentals

- Make sure you are well-versed in the foundational concepts of JavaScript, such as data types, variables, operators, control structures, functions and objects.

2. Master Key Concepts

- It's also important to study up on key JavaScript topics like promises, asynchronous programming, hoisting, scope, closures, prototypes and ES6 features. You should understand how each of these works.

3. Study Common Interview Topics

- Take the time to review JavaScript interview questions that are regularly asked, including those about closures, prototypes, callbacks, promises, AJAX (asynchronous JavaScript and XML), error handling and module systems. Most interviews follow a similar pattern. Preparing answers for those questions will help you stand out from other candidates.

4. Master Debugging Skills

- Interviewers for JavaScript will often look to assess your code debugging skills. Practice using IDEs or browser developer tools to find and correct common JavaScript code issues. Learn how to read error messages and review basic debugging techniques.

5. Practice Coding

- To develop your coding abilities, complete coding tasks and challenges. Concentrate on standard JavaScript data structures and algorithms such arrays, strings, objects, recursion and sorting techniques.

6. Build Projects

- Take on modest JavaScript projects to get experience and show that you can create useful applications. Showing off your portfolio at the interview is also beneficial. In addition, developers can also work on JavaScript projects to obtain practical experience and show that they are capable of building effective applications.

7. Mock Interviews

- With a friend or mentor, practice mock interviews paying particular attention to both behavioral and technical components. This enables you to hear useful criticism and become accustomed to the interview process.

  It's not just mastering the technical aspect of JavaScript, it's about your body language and how you explain your answers. Many companies are also assessing your ability to work within a team and pair program. The better you can explain your actions and thought process, the more likely you'll be to win over the interviewer.