JavaScript & MERN

150 Interview Questions & Detailed Answers

A comprehensive guide covering:
• JavaScript Basics
• Functions & Scope
• Arrays & Objects
• DOM & Async JS
• Advanced JavaScript
• Node.js & Express

Table of Contents

Section 1: JavaScript Basics (1–25)

1. What is JavaScript?

JavaScript is a high-level, interpreted programming language that conforms to the ECMAScript specification. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. Originally designed for client-side scripting to make web pages interactive, it has evolved (via Node.js) to run on the server-side as well.

JavaScript engine (like V8 in Chrome) executes the code. Unlike compiled languages like C++, JS is JIT (Just-In-Time) compiled, meaning it is compiled to machine code during execution rather than beforehand.

```
// Example of basic interaction
console.log('Hello World');
document.body.style.background = 'blue';
```

2. Difference between var, let, and const?

The main differences lie in scope, hoisting, and reassignability:

1. **var**: Function-scoped. It is hoisted to the top of its scope and initialized with 'undefined'. It can be redeclared and reassigned. 2. **let**: Block-scoped (only exists within {}). It is hoisted but remains in the 'Temporal Dead Zone' until the declaration line is reached. Can be reassigned but not redeclared. 3. **const**: Block-scoped. Must be initialized during declaration. It cannot be reassigned (though objects declared with const can be mutated).

```
if (true) {
  var a = 10; // Function scoped
  let b = 20; // Block scoped
}
console.log(a); // 10
console.log(b); // Error: b is not defined
```

3. What is hoisting?

Hoisting is the JavaScript engine's behavior of moving variable and function declarations to the top of their containing scope (global or function) during the compilation phase, before code execution.

While declarations are hoisted, initializations are not. This means a 'var' variable can be accessed before it's written but will be 'undefined'. Function declarations are fully hoisted, meaning you can call them before they appear in the code.

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

greet(); // 'Hello'
function greet() { console.log('Hello'); }
```

4. Difference between == and ===?

== (Abstract Equality): Performs type coercion before comparing. If types differ, JS tries to convert them to a common type (usually number) before checking equality.

=== (Strict Equality): Checks both the value and the data type. No coercion occurs. It is generally recommended to always use === to avoid unexpected bugs caused by automatic type conversion.

```
console.log(5 == '5');  // true (string '5' becomes number 5)
console.log(5 === '5'); // false (different types)
```

5. What are primitive data types?

Primitives are data types that are not objects and do not have methods. They are immutable. In JavaScript, there are 7 primitive types: 1. **String**: Text data. 2. **Number**: Integers and floating-point. 3. **BigInt**: For integers larger than 2^53 - 1. 4. **Boolean**: true or false. 5. **Undefined**: Variable declared but not assigned. 6. **Null**: Intentional absence of value. 7. **Symbol**: Unique identifiers.

```
let type = typeof 'Hello'; // 'string'
let empty = null;
```

6. What is NaN?

NaN stands for 'Not-a-Number'. It is a property of the global object and results from an arithmetic operation that cannot produce a valid number (e.g., dividing zero by zero or multiplying a string by a number).

Interestingly, the type of NaN is actually 'number'. To check if a value is NaN, you should use `Number.isNaN()` because `NaN === NaN` evaluates to false.

```
console.log('abc' / 2); // NaN
console.log(typeof NaN); // 'number'
```

7. What is type coercion?

Type coercion is the automatic or implicit conversion of values from one data type to another (such as strings to numbers). This happens in operations involving different types.

It can be 'explicit' (using `Number()`) or 'implicit' (using `==` or `+` with strings).

```
const val = '5' - 1; // 4 (Implicit coercion to Number)
const str = '5' + 1; // '51' (Implicit coercion to String)
```

8. What is the typeof operator?

The `typeof` operator returns a string indicating the type of the operand. It is useful for debugging and type checking variables dynamically.

```
typeof 42; // 'number'
typeof 'blubber'; // 'string'
typeof true; // 'boolean'
typeof {}; // 'object'
```

9. What is null?

Null represents the intentional absence of any object value. It is one of JavaScript's primitive values. It is often treated as 'falsy' for boolean operations.

```
let user = null; // Explicitly empty
```

10. What is undefined?

Undefined means a variable has been declared but has not yet been assigned a value. It is also the return value of functions that do not explicitly return anything.

```
let x;
console.log(x); // undefined
```

11. Is JavaScript single-threaded?

Yes, JavaScript is single-threaded. This means it has one Call Stack and one Memory Heap. It executes code in order and must finish executing a piece of code before moving to the next.

However, it handles concurrent operations (like API calls) using the Event Loop, Web APIs, and Callback Queues, giving the illusion of multi-threading.

12. What is a loop?

A loop is a programming structure that repeats a sequence of instructions until a specific condition is met. Common JS loops include `for`, `while`, and `do...while`.

```
for(let i=0; i<3; i++) {
  console.log(i);
}
```

13. What is a ternary operator?

The ternary operator is the only JavaScript operator that takes three operands. It is frequently used as a concise alternative to an `if...else` statement.

```
const status = age >= 18 ? 'adult' : 'minor';
```

14. What is a boolean?

A Boolean represents a logical entity and can have two values: `true` and `false`.

15. What is BigInt?

BigInt is a special numeric type that provides support for integers of arbitrary length. A BigInt is created by appending `n` to the end of an integer literal.

```
const huge = 9007199254740991n;
```

16. What is Infinity in JS?

Infinity is a numeric value representing a number greater than any other number. `-Infinity` represents a number lower than any other number.

```
console.log(1 / 0); // Infinity
```

17. What is parseInt()?

The `parseInt()` function parses a string argument and returns an integer of the specified radix (the base in mathematical numeral systems).

```
parseInt('10.5'); // 10
```

18. What is parseFloat()?

The `parseFloat()` function parses an argument and returns a floating-point number. Unlike parseInt, it keeps decimals.

```
parseFloat('10.5'); // 10.5
```

19. What is isNaN()?

The global `isNaN()` function converts the argument to a number and then checks if it is NaN. `Number.isNaN()` is stricter and does not convert the type.

```
isNaN('hello'); // true (converts to NaN)
Number.isNaN('hello'); // false (not a number type)
```

20. What is console.log()?

It is a function that writes a message to the debugging console. It is primarily used for debugging purposes.

21. What is a comment?

Comments are lines of code ignored by the compiler. Single line uses `//`, multi-line uses `/* ... */`.

22. What is scope?

Scope determines the accessibility (visibility) of variables. JS has Global Scope, Function Scope, and Block Scope.

23. What is a variable?

A variable is a container for storing data values. In JS, we declare them using var, let, or const.

24. What is JSON?

JSON (JavaScript Object Notation) is a lightweight data interchange format. It is easy for humans to read and write and easy for machines to parse and generate.

```
{ "name": "John", "age": 30 }
```

25. What is typeof null?

It returns `'object'`. This is a well-known historical bug in JavaScript that was never fixed to maintain backward compatibility.

```
console.log(typeof null); // 'object'
```

Section 2: Functions & Scope (26–50)

26. What is a function?

A function is a block of code designed to perform a particular task. A function is executed when 'something' invokes it (calls it).

```
function add(a, b) { return a + b; }
```

27. What is a function declaration?

A function declaration uses the `function` keyword followed by the name. These are hoisted completely to the top of their scope.

```
function sayHello() { return 'Hi'; }
```

28. What is a function expression?

A function expression allows us to define a function inside an expression (often assigned to a variable). These are NOT hoisted.

```
const sayHello = function() { return 'Hi'; };
```

29. What is an arrow function?

Arrow functions (ES6) provide a concise syntax. Crucially, they do not bind their own `this`. They inherit `this` from the parent scope (lexical scoping).

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

30. What is a callback?

A callback is a function passed as an argument to another function, which is then invoked inside the outer function to complete some kind of routine or action.

```
function process(callback) {
  callback();
}
```

31. What is global scope?

Variables declared outside any function have Global Scope. They can be accessed from anywhere in the script.

32. What is block scope?

Variables declared with `let` or `const` inside a `{}` block cannot be accessed from outside the block.

33. What is function scope?

Variables defined inside a function are not accessible (visible) from outside the function.

34. What is closure?

A closure is the combination of a function bundled together with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function’s scope from an inner function.

```
function makeCounter() {
  let count = 0;
  return function() {
    return count++;
  };
}
```

35. What is hoisting in functions?

Function Declarations are hoisted completely (body and all). Function Expressions (var x = function...) hoist the variable `x` as undefined, but not the function body.

36. What is recursion?

Recursion is a technique where a function calls itself to solve a problem. It requires a base case to stop the loop.

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

37. What is an anonymous function?

A function without a name. Often used as arguments to other functions or in IIFEs.

38. What is a default parameter?

Default function parameters allow named parameters to be initialized with default values if no value or `undefined` is passed.

```
function multiply(a, b = 1) { return a * b; }
```

39. What is the rest operator?

The rest parameter syntax `...args` allows a function to accept an indefinite number of arguments as an array.

```
function sum(...numbers) {
  return numbers.reduce((a,b)=>a+b);
}
```

40. What is the spread operator?

The spread operator `...` allows an iterable (like an array) to be expanded in places where zero or more arguments or elements are expected.

```
const arr1 = [1, 2];
const arr2 = [...arr1, 3, 4]; // [1,2,3,4]
```

41. What is IIFE?

Immediately Invoked Function Expression. It is a function that runs as soon as it is defined. It is used to create a private scope.

```
(function() { console.log('I run now'); })();
```

42. What is lexical scope?

Lexical scope means that a variable defined outside a function can be accessible inside another function defined after the variable declaration. Inner functions contain the scope of parent functions.

43. What are higher-order functions?

A function that accepts another function as an argument OR returns a function. Examples: `map`, `filter`, `reduce`.

44. What is currying?

Currying is the process of converting a function that takes multiple arguments into a sequence of functions that each take a single argument.

```
const add = a => b => a + b;
add(5)(3); // 8
```

45. What is memoization?

An optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.

46. What is a pure function?

A function where the return value is determined only by its input values, without observable side effects.

47. What is the arguments object?

An Array-like object accessible inside functions that contains the values of the arguments passed to that function. (Note: Not available in arrow functions).

48. Can functions return functions?

Yes, because functions are first-class citizens in JavaScript.

49. What is temporal dead zone?

The period between entering the scope and the actual declaration of a `let` or `const` variable.

50. What is shadowing?

Variable shadowing occurs when a variable declared within a certain scope has the same name as a variable declared in an outer scope.

Section 3: Arrays & Objects (51–75)

51. What is an array?

An ordered collection of values (items). In JS, arrays are objects and can hold different data types.

```
let fruits = ['Apple', 'Banana'];
```

52. What is indexing?

Arrays are zero-indexed, meaning the first element is at position 0.

53. What is push()?

Adds one or more elements to the end of an array and returns the new length.

```
arr.push(4);
```

54. What is pop()?

Removes the last element from an array and returns that element.

```
arr.pop();
```

55. What is shift()?

Removes the first element from an array and returns that element.

56. What is unshift()?

Adds one or more elements to the beginning of an array.

57. What is map()?

Creates a new array populated with the results of calling a provided function on every element in the calling array. It does not mutate the original array.

```
const num = [1,2];
const dbl = num.map(x => x*2); // [2,4]
```

58. What is filter()?

Creates a shallow copy of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

```
const nums = [1, 10, 5];
const big = nums.filter(x => x > 5); // [10]
```

59. What is reduce()?

Executes a user-supplied 'reducer' callback function on each element of the array, passing in the return value from the calculation on the preceding element. It reduces the array to a single value.

```
const sum = [1,2,3].reduce((acc, curr) => acc + curr, 0); // 6
```

60. What is find()?

Returns the first element in the provided array that satisfies the provided testing function.

61. What is some()?

Tests whether at least one element in the array passes the test implemented by the provided function.

62. What is every()?

Tests whether all elements in the array pass the test implemented by the provided function.

63. What is destructuring?

A JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

```
const [a, b] = [10, 20];
const { name } = userObj;
```

64. What is an object?

A collection of related data and/or functionality, usually consisting of several variables and functions (which are called properties and methods when they are inside objects).

65. What is dot notation?

Accessing properties using a dot: `obj.key`.

66. What is bracket notation?

Accessing properties using brackets: `obj['key']`. Necessary if keys have spaces or are dynamic variables.

67. What is Object.keys()?

Returns an array of a given object's own enumerable property names.

68. What is Object.values()?

Returns an array of a given object's own enumerable property values.

69. What is Object.entries()?

Returns an array of a given object's own enumerable string-keyed property [key, value] pairs.

70. What is Object.freeze()?

Freezes an object: other code cannot delete or change any properties.

71. What is Object.seal()?

Seals an object: prevents new properties from being added, but existing properties can still be changed.

72. What is Array.isArray()?

Determines whether the passed value is an Array.

73. What is JSON.stringify()?

Converts a JavaScript object or value to a JSON string.

74. What is JSON.parse()?

Parses a JSON string, constructing the JavaScript value or object described by the string.

75. Difference between array and object?

Arrays are ordered lists indexed by numbers. Objects are unordered collections indexed by keys (strings/symbols).

Section 4: DOM & Async JS (76–100)

76. What is the DOM?

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects.

77. What is DOM manipulation?

The process of using JavaScript to add, remove, and modify elements of the website.

78. What is querySelector()?

Returns the first element within the document that matches the specified selector, or group of selectors.

79. What is an event?

Events are things that happen to HTML elements. When JavaScript is used in HTML pages, JavaScript can 'react' on these events (like clicks, mouseovers, keypresses).

80. What is addEventListener()?

A method that attaches an event handler to the specified element without overwriting existing event handlers.

```
btn.addEventListener('click', () => { alert('Clicked'); });
```

81. What is event bubbling?

A type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors (parents) of the target element in the same nesting hierarchy till it reaches the outermost DOM element.

82. What is event capturing?

The opposite of bubbling. The event is captured by the outermost element and propagated to the inner elements.

83. What is stopPropagation()?

Prevents further propagation of the current event in the capturing and bubbling phases.

84. What is asynchronous programming?

A programming pattern that enables your program to start a potentially long-running task and continue responsive to other events while that task runs, rather than having to wait until that task has finished.

85. What is setTimeout()?

Sets a timer which executes a function or specified piece of code once the timer expires.

86. What is setInterval()?

Repeatedly calls a function or executes a code snippet, with a fixed time delay between each call.

87. What is a Promise?

A Promise is an object representing the eventual completion or failure of an asynchronous operation. It is a cleaner way to handle async code than callbacks.

```
new Promise((resolve, reject) => { ... })
```

88. What are Promise states?

1. Pending (initial state) 2. Fulfilled (operation completed successfully) 3. Rejected (operation failed)

89. What is .then()?

Method used to schedule a callback to be executed when the Promise is successfully resolved.

90. What is .catch()?

Method used to schedule a callback to be executed when the Promise is rejected.

91. What is async?

The `async` keyword is placed before a function to ensure it returns a promise.

92. What is await?

The `await` keyword works only inside async functions. It makes JavaScript wait until the promise returns a result.

```
async function getData() {
  let data = await fetch(url);
}
```

93. What is fetch()?

The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses.

94. What is callback hell?

The situation where callbacks are nested within other callbacks several levels deep, making it difficult to understand and maintain the code.

95. What is the Event Loop?

The Event Loop is a mechanism that allows JavaScript to perform non-blocking I/O operations despite being single-threaded. It constantly checks if the Call Stack is empty; if so, it moves tasks from the Task Queue to the Call Stack.

96. What runs first: Promise or setTimeout?

Promises (Microtasks) run before setTimeout (Macrotasks). The Event Loop clears the Microtask queue before checking the Macrotask queue.

97. What is the microtask queue?

A queue for high-priority tasks like Promise callbacks and MutationObserver.

98. What is the macrotask queue?

A queue for tasks like setTimeout, setInterval, setImmediate, and I/O tasks.

99. What is innerHTML?

A property that gets or sets the HTML or XML markup contained within the element.

100. What is textContent?

A property that represents the text content of a node and its descendants. Unlike innerHTML, it does not parse HTML.

Section 5: Advanced JavaScript (101–125)

101. What is 'this'?

The `this` keyword refers to the object it belongs to. In a method, `this` refers to the owner object. Alone, it refers to the global object. In a function, it refers to the global object (or undefined in strict mode).

```
const obj = {
  name: 'Me',
  print: function() { console.log(this.name); }
};
```

102. What is a prototype?

Prototypes are the mechanism by which JavaScript objects inherit features from one another.

103. What is the prototype chain?

When accessing a property, JS looks at the object. If not found, it looks at the object's prototype, and so on, until null is reached.

104. What is __proto__?

An accessor property that exposes the [[Prototype]] of an object.

105. What are classes in JS?

Classes are templates for creating objects. They encapsulate data with code to work on that data. Introduced in ES6, they are syntactic sugar over JavaScript's existing prototype-based inheritance.

```
class Car {
  constructor(brand) { this.brand = brand; }
}
```

106. Why don't arrow functions have 'this'?

Arrow functions capture the `this` value of the enclosing context where the function is defined.

107. Why can't arrow functions be constructors?

Since they don't have their own `this`, they cannot be initialized with `new`.

108. What is pass by value?

Primitive types are passed by value (copy). Changing the copy does not affect the original.

109. What is pass by reference?

Objects are passed by reference. Changing the reference affects the original object.

110. What is deep copy?

Creating a new object with new memory locations for all properties, including nested ones.

111. What is shallow copy?

Copying the top-level properties, but nested objects are still shared references.

112. What is spread for objects?

Copies own enumerable properties from a provided object onto a new object.

113. What is illegal shadowing?

Declaring a variable with `var` in a scope where a `let` or `const` variable with the same name already exists.

114. What is strict mode?

A way to opt in to a restricted variant of JavaScript (`'use strict';`). It eliminates some silent errors and prohibits some syntax.

115. Why use strict mode?

It makes it easier to write 'secure' JavaScript and prevents accidental globals.

116. What is call()?

Invokes a function with a given `this` value and arguments provided individually.

```
func.call(thisObj, arg1, arg2);
```

117. What is apply()?

Invokes a function with a given `this` value and arguments provided as an array.

```
func.apply(thisObj, [argsArray]);
```

118. What is bind()?

Creates a new function that, when called, has its `this` keyword set to the provided value.

```
const bound = func.bind(thisObj);
```

119. What is a constructor function?

A regular function used with the `new` keyword to create objects.

120. What is instanceof?

Tests if the prototype property of a constructor appears anywhere in the prototype chain of an object.

121. What is a Symbol?

A unique and immutable primitive value, often used to identify object properties without collision.

122. What is garbage collection?

The process by which the JS engine automatically frees up memory that is no longer in use (objects with no references).

123. What is tail recursion?

A recursive function where the recursive call is the last action in the function. Some engines optimize this to save stack space.

124. What is event delegation?

A pattern where you place a single event listener on a parent element instead of one on every child element.

125. What is memoization (Advanced)?

A specific form of caching involving caching the return value of a function based on its parameters.

Section 6: Node.js & Express (126–150)

126. What is Node.js?

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a web browser. It is built on Chrome's V8 JavaScript engine.

127. What is npm?

npm (Node Package Manager) is the default package manager for Node.js, consisting of a CLI and a registry of public packages.

128. What is module.exports?

A special object which is included in every JS file in the Node.js application by default. The module is a variable that represents the current module, and exports is an object that will be exposed as a module.

129. What is require()?

A built-in function to include external modules that exist in separate files.

```
const fs = require('fs');
```

130. What is the fs module?

The file system module allows you to work with the file system on your computer (read, write, delete files).

131. What is the http module?

Allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP). It can create a server.

132. What is non-blocking I/O?

Input/Output operations that do not block the execution of further operations. Node.js uses this to handle high concurrency.

133. What is event-driven architecture?

A software architecture pattern promoting the production, detection, consumption of, and reaction to events.

134. What is Express.js?

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It simplifies routing and middleware.

135. What is routing?

Routing refers to determining how an application responds to a client request to a particular endpoint (URI) and method (GET, POST, etc.).

```
app.get('/', (req, res) => res.send('Hi'));
```

136. What is middleware?

Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle.

```
app.use((req, res, next) => {
  console.log('Time:', Date.now());
  next();
});
```

137. What is express.json()?

It is a built-in middleware function in Express. It parses incoming requests with JSON payloads.

138. What is a REST API?

Representational State Transfer. An architectural style for an application program interface (API) that uses HTTP requests to access and use data.

139. What is CRUD?

Create, Read, Update, Delete. These are the four basic functions of persistent storage.

140. What is HTTP?

Hypertext Transfer Protocol. The foundation of data communication for the World Wide Web.

141. What are HTTP methods?

GET (retrieve), POST (submit), PUT (update), DELETE (remove), PATCH (partial update).

142. What are status codes?

Standard response codes given by web site servers on the Internet. 2xx (Success), 3xx (Redirect), 4xx (Client Error), 5xx (Server Error).

143. What is 401?

Unauthorized. The request has not been applied because it lacks valid authentication credentials.

144. What is 500?

Internal Server Error. The server encountered an unexpected condition.

145. What is CORS?

Cross-Origin Resource Sharing. A mechanism that uses additional HTTP headers to tell browsers to give a web application running at one origin, access to selected resources from a different origin.

146. What is JWT?

JSON Web Token. A compact, URL-safe means of representing claims to be transferred between two parties.

147. What is MongoDB?

A NoSQL database program that uses JSON-like documents with optional schemas.

148. What is Mongoose?

An Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

149. What is MVC?

Model-View-Controller. An architectural pattern that separates an application into three main logical components.

150. What is package.json?

A file that contains metadata about the project and lists the dependencies (packages) that the project needs to run.