

Fresher Level Questions

1. What is JavaScript?

- **Theory:** JavaScript is a high-level, dynamic, and interpreted programming language commonly used for web development to create interactive effects within web browsers.

2. What are the data types supported by JavaScript?

- **Theory:** JavaScript has several data types, including:
 - Primitive types: `string`, `number`, `boolean`, `null`, `undefined`, `symbol`, and `bigint`.
 - Reference types: Objects, Arrays, Functions.

Code Example:

javascript

Copy code

```
let name = "John"; // string
let age = 30; // number
let isStudent = false; // boolean
let user = null; // null
let id; // undefined
let uniqueSymbol = Symbol('id'); // symbol
```

○

3. What is the difference between `==` and `===`?

- **Theory:**
 - `==` checks for value equality, performing type coercion if necessary.
 - `===` checks for both value and type equality without type coercion.

Code Example:

javascript

Copy code

```
console.log(5 == '5'); // true (type coercion)
console.log(5 === '5'); // false (no type coercion)
```

○

4. Explain Hoisting in JavaScript.

- **Theory:** Hoisting is JavaScript's behavior of moving variable and function declarations to the top of their containing scope during the compilation phase.

Code Example:

javascript

Copy code

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

○

5. What is a closure?

- **Theory:** A closure is a function that retains access to its lexical scope, even when the function is executed outside that scope.

Code Example:

javascript

Copy code

```
function outerFunction() {  
    let outerVariable = 'I am outside!';  
    return function innerFunction() {  
        console.log(outerVariable);  
    };  
}  
  
const innerFunc = outerFunction();  
innerFunc(); // Output: I am outside!
```

○

Intermediate Level Questions

6. What is the difference between `let`, `const`, and `var`?

- **Theory:**
 - `var`: Function-scoped, can be redeclared and updated.
 - `let`: Block-scoped, can be updated but not redeclared in the same scope.
 - `const`: Block-scoped, cannot be updated or redeclared.

Code Example:

javascript

Copy code

```
var x = 10;  
let y = 20;  
const z = 30;
```

```
// Redefinition  
var x = 15; // Allowed  
// let y = 25; // Error: Identifier 'y' has already been declared  
// const z = 35; // Error: Identifier 'z' has already been declared
```

○

7. Explain the concept of Promises.

- **Theory:** A Promise is an object representing the eventual completion (or failure) of an asynchronous operation and its resulting value.

Code Example:

javascript

Copy code

```
const myPromise = new Promise((resolve, reject) => {
    const success = true;
    if (success) {
        resolve("Operation succeeded!");
    } else {
        reject("Operation failed!");
    }
});

myPromise
    .then(result => console.log(result))
    .catch(error => console.log(error));

    ○
```

8. What is the purpose of the `this` keyword?

- **Theory:** The `this` keyword refers to the object that is currently executing the function. Its value can change depending on the context in which a function is called.

Code Example:

javascript

Copy code

```
const obj = {
    name: 'Alice',
    greet: function() {
        console.log(`Hello, my name is ${this.name}`);
    }
};

obj.greet(); // Output: Hello, my name is Alice
```

○

9. What are arrow functions, and how do they differ from regular functions?

- **Theory:** Arrow functions are a shorter syntax for writing functions in JavaScript. They do not have their own `this`, which is lexically inherited from the outer function.

Code Example:

javascript

Copy code

```
const add = (a, b) => a + b; // Arrow function
console.log(add(2, 3)); // Output: 5
```

```
function regularFunction() {  
    console.log(this); // 'this' refers to the global object or  
undefined in strict mode  
}
```

○

Experienced Level Questions

10. What is event delegation in JavaScript?

- **Theory:** Event delegation is a technique that uses a single event listener on a parent element to manage events for multiple child elements. It improves performance and memory efficiency.

Code Example:

```
javascript  
Copy code  
const parentElement = document.getElementById('parent');  
parentElement.addEventListener('click', function(event) {  
    if (event.target.matches('.child')) {  
        console.log('Child element clicked:', event.target);  
    }  
});
```

○

11. Explain the concept of asynchronous programming in JavaScript.

- **Theory:** Asynchronous programming allows code to run without blocking the main thread. It uses callbacks, Promises, and async/await syntax to handle asynchronous operations.

Code Example:

```
javascript  
Copy code  
async function fetchData() {  
    try {  
        const response = await  
fetch('https://api.example.com/data');  
        const data = await response.json();  
        console.log(data);  
    } catch (error) {  
        console.error('Error fetching data:', error);  
    }  
}  
fetchData();
```

○

12. What is the difference between `call`, `apply`, and `bind`?

- **Theory:**

- `call`: Calls a function with a given `this` value and arguments provided individually.
- `apply`: Calls a function with a given `this` value and arguments provided as an array.
- `bind`: Returns a new function that, when called, has its `this` keyword set to the provided value.

Code Example:

javascript

Copy code

```
function greet() {  
    console.log(`Hello, my name is ${this.name}`);  
}  
  
const person = { name: 'Bob' };  
  
greet.call(person); // Output: Hello, my name is Bob  
greet.apply(person); // Output: Hello, my name is Bob  
  
const boundGreet = greet.bind(person);  
boundGreet(); // Output: Hello, my name is Bob
```

○

13. What are JavaScript modules, and how do they work?

- **Theory:** JavaScript modules allow you to break up your code into separate files, making it easier to maintain and reuse. Modules can export variables, functions, or classes and import them into other modules.

Code Example:

javascript

Copy code

```
// In file: module.js  
export const name = 'Charlie';  
export function greet() {  
    console.log(`Hello, ${name}`);  
}  
  
// In another file: app.js  
import { name, greet } from './module.js';  
greet(); // Output: Hello, Charlie
```

○

14. What is the purpose of the `prototype` in JavaScript?

- **Theory:** The `prototype` is an object that allows you to add properties and methods to a constructor function, enabling inheritance in JavaScript.

Code Example:

```
javascript
Copy code
function Person(name) {
    this.name = name;
}
Person.prototype.greet = function() {
    console.log(`Hello, my name is ${this.name}`);
};

const alice = new Person('Alice');
alice.greet(); // Output: Hello, my name is Alice
```

○

15. What is the difference between synchronous and asynchronous code?

- **Theory:**

- Synchronous code executes line by line, blocking the execution of subsequent code until the current operation completes.
- Asynchronous code allows other operations to run while waiting for a time-consuming task (like a network request) to complete.

Code Example:

```
javascript
Copy code
// Synchronous
console.log('Start');
console.log('End'); // 'End' will print after 'Start'

// Asynchronous
console.log('Start');
setTimeout(() => {
    console.log('Middle'); // This will print after 2 seconds
}, 2000);
console.log('End'); // 'End' will print immediately after 'Start'
```

○

These questions and answers cover a range of concepts and will help you prepare for interviews at various experience levels. Feel free to ask for more detailed explanations or additional topics!

You said:

aur bhee hai

ChatGPT said:

ChatGPT

Sure! Here are some additional JavaScript interview questions with their explanations and code examples:

Fresher Level Questions

16. What is an object in JavaScript?

- **Theory:** An object is a collection of key-value pairs where each key is a string (or Symbol) and each value can be of any data type, including other objects.

Code Example:

javascript

Copy code

```
const person = {
    name: 'John',
    age: 30,
    greet: function() {
        console.log(`Hello, my name is ${this.name}`);
    }
};
person.greet(); // Output: Hello, my name is John
```

○

17. What are arrays in JavaScript?

- **Theory:** An array is a special type of object used for storing multiple values in a single variable. Arrays are zero-indexed and can hold elements of any type.

Code Example:

javascript

Copy code

```
const fruits = ['Apple', 'Banana', 'Cherry'];
console.log(fruits[1]); // Output: Banana
```

○

Intermediate Level Questions

18. What is the event loop in JavaScript?

- **Theory:** The event loop is a mechanism that allows JavaScript to perform non-blocking I/O operations by offloading operations to the system kernel whenever possible. It enables asynchronous execution.

Code Example:

```
javascript
Copy code
console.log('Start');
setTimeout(() => {
    console.log('Timeout!');
}, 0);
console.log('End');
// Output: Start, End, Timeout!
```

○

19. Explain the use of `async/await`.

- **Theory:** `async/await` is syntactic sugar built on Promises, allowing you to write asynchronous code that looks synchronous. It makes it easier to read and manage asynchronous operations.

Code Example:

```
javascript
Copy code
const fetchData = async () => {
    try {
        const response = await
fetch('https://api.example.com/data');
        const data = await response.json();
        console.log(data);
    } catch (error) {
        console.error('Error:', error);
    }
};
fetchData();
```

○

20. What is destructuring in JavaScript?

- **Theory:** Destructuring is a syntax that allows unpacking values from arrays or properties from objects into distinct variables.

Code Example:

```
javascript
Copy code
const array = [1, 2, 3];
const [a, b] = array; // a = 1, b = 2

const obj = { x: 1, y: 2 };
const { x, y } = obj; // x = 1, y = 2
```

○

Experienced Level Questions

21. What are higher-order functions?

- **Theory:** A higher-order function is a function that either takes another function as an argument or returns a function as a result. They are commonly used in functional programming.

Code Example:

```
javascript
Copy code
function greet(name) {
  return function() {
    console.log(`Hello, ${name}`);
  };
}
const greetJohn = greet('John');
greetJohn(); // Output: Hello, John
```

○

22. What is the `bind` method and how does it work?

- **Theory:** The `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.

Code Example:

```
javascript
Copy code
function greet() {
  console.log(`Hello, ${this.name}`);
}
const person = { name: 'Alice' };
const boundGreet = greet.bind(person);
boundGreet(); // Output: Hello, Alice
```

○

23. What is throttling and debouncing?

- **Theory:**
 - **Throttling** limits the number of times a function can be executed over time. It ensures that a function is only called at most once in a specified time frame.
 - **Debouncing** ensures that a function is only called once after a specified time has passed since the last time it was invoked.

Code Example:

```
javascript
Copy code
// Throttle function
function throttle(fn, wait) {
    let lastTime = 0;
    return function(...args) {
        const now = Date.now();
        if (now - lastTime >= wait) {
            lastTime = now;
            fn.apply(this, args);
        }
    };
}

// Debounce function
function debounce(fn, delay) {
    let timeoutId;
    return function(...args) {
        clearTimeout(timeoutId);
        timeoutId = setTimeout(() => {
            fn.apply(this, args);
        }, delay);
    };
}
```

○

24. What are template literals in JavaScript?

- **Theory:** Template literals are string literals that allow embedded expressions, making it easier to create multiline strings and interpolate variables.

Code Example:

```
javascript
Copy code
const name = 'Alice';
const greeting = `Hello, ${name}!`;
console.log(greeting); // Output: Hello, Alice!

const multiline = `This is
a multiline
string.`;
console.log(multiline);
```

○

25. What is the purpose of the `with` statement?

- **Theory:** The `with` statement extends the scope chain for a statement. However, it is generally discouraged due to readability and performance issues.

Code Example:

```
javascript
Copy code
const obj = { x: 10, y: 20 };
with (obj) {
  console.log(x + y); // Output: 30
}
```

○

These additional questions will further enhance your preparation for JavaScript interviews across various experience levels. If you need more questions or specific topics, feel free to ask!

4o mini

You said:

aur questions hai keya

ChatGPT said:

ChatGPT

Certainly! Here are more JavaScript interview questions along with their explanations and code examples:

Fresher Level Questions

26. What are functions in JavaScript?

- **Theory:** A function is a block of code designed to perform a particular task. It can take inputs (parameters) and return an output.

Code Example:

```
javascript
Copy code
function add(a, b) {
  return a + b;
}
console.log(add(5, 10)); // Output: 15
```

○

27. What is the difference between a function declaration and a function expression?

- **Theory:**

- **Function Declaration:** A named function defined with the `function` keyword. It can be called before it is defined due to hoisting.
- **Function Expression:** A function defined as part of a larger expression, like a variable assignment. It is not hoisted.

Code Example:

```
javascript
Copy code
// Function Declaration
function greet() {
    console.log('Hello!');
}
greet(); // Output: Hello!

// Function Expression
const sayHi = function() {
    console.log('Hi!');
};
sayHi(); // Output: Hi!
```

○

28. What is the purpose of the `return` statement?

- **Theory:** The `return` statement is used to end the execution of a function and specify the value to be returned to the function caller.

Code Example:

```
javascript
Copy code
function square(x) {
    return x * x;
}
console.log(square(4)); // Output: 16
```

○

Intermediate Level Questions

29. What are callback functions in JavaScript?

- **Theory:** A callback function is a function that is passed as an argument to another function and is executed after the completion of that function.

Code Example:

```
javascript
Copy code
function fetchData(callback) {
    // Simulating a data fetch with setTimeout
```

```
    setTimeout(() => {
      const data = { name: 'Alice', age: 25 };
      callback(data);
    }, 1000);
}

fetchData((data) => {
  console.log(data); // Output: { name: 'Alice', age: 25 }
});
```

○

30. What is the purpose of the `typeof` operator?

- **Theory:** The `typeof` operator returns a string indicating the type of the unevaluated operand.

Code Example:

javascript

Copy code

```
console.log(typeof 'Hello'); // Output: string
console.log(typeof 42); // Output: number
console.log(typeof true); // Output: boolean
console.log(typeof {}); // Output: object
```

○

31. Explain the difference between `slice()`, `splice()`, and `concat()`.

- **Theory:**

- `slice()`: Returns a shallow copy of a portion of an array into a new array.
- `splice()`: Changes the contents of an array by removing or replacing existing elements and/or adding new elements.
- `concat()`: Merges two or more arrays and returns a new array.

Code Example:

javascript

Copy code

```
const fruits = ['Apple', 'Banana', 'Cherry'];

// slice
const citrus = fruits.slice(1); // ['Banana', 'Cherry']

// splice
fruits.splice(1, 1, 'Orange'); // fruits is now ['Apple', 'Orange', 'Cherry']
```

```
// concat
const moreFruits = fruits.concat(['Mango', 'Pineapple']); // []
['Apple', 'Orange', 'Cherry', 'Mango', 'Pineapple']
```

○

Experienced Level Questions

32. What is the `instanceof` operator?

- **Theory:** The `instanceof` operator tests whether an object is an instance of a specific constructor or class.

Code Example:

javascript

Copy code

```
function Person(name) {
    this.name = name;
}
const john = new Person('John');
console.log(john instanceof Person); // Output: true
console.log(john instanceof Object); // Output: true
```

○

33. What are the benefits of using `let` and `const` over `var`?

- **Theory:**

- `let` and `const` provide block scoping, preventing variables from being accessible outside their block.
- `const` ensures that a variable's reference cannot be changed, providing immutability for reference types.

Code Example:

javascript

Copy code

```
if (true) {
    var x = 5; // function-scoped
    let y = 10; // block-scoped
    const z = 15; // block-scoped
}
console.log(x); // Output: 5
// console.log(y); // Error: y is not defined
// console.log(z); // Error: z is not defined
```

○

34. What is the purpose of the `new` keyword?

- **Theory:** The `new` keyword is used to create an instance of an object from a constructor function. It sets up the prototype chain and initializes properties.

Code Example:

javascript

Copy code

```
function Car(make, model) {
    this.make = make;
    this.model = model;
}
const myCar = new Car('Toyota', 'Camry');
console.log(myCar.make); // Output: Toyota
```

○

35. What is the difference between deep copy and shallow copy?

- **Theory:**

- **Shallow Copy:** Creates a new object, but the properties are references to the same objects as in the original.
- **Deep Copy:** Creates a new object and recursively copies all properties, ensuring that nested objects are copied rather than referenced.

Code Example:

javascript

Copy code

```
const original = { a: 1, b: { c: 2 } };

// Shallow Copy
const shallowCopy = { ...original };
shallowCopy.b.c = 3;
console.log(original.b.c); // Output: 3 (changed)

// Deep Copy using JSON
const deepCopy = JSON.parse(JSON.stringify(original));
deepCopy.b.c = 4;
console.log(original.b.c); // Output: 3 (unchanged)
```

○

36. What is a Promise chain?

- **Theory:** A Promise chain is a sequence of asynchronous operations where each operation is performed after the previous one is completed, using `then()` for chaining.

Code Example:

javascript

Copy code

```
const fetchData = (data) => {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve(data);
        }, 1000);
    });
};

fetchData('Data 1')
    .then(result => {
        console.log(result); // Output: Data 1
        return fetchData('Data 2');
})
    .then(result => {
        console.log(result); // Output: Data 2
    });

```

○

More Advanced Questions

37. What is the difference between `Array.map()` and `Array.forEach()`?

- **Theory:**

- `Array.map()`: Creates a new array populated with the results of calling a provided function on every element in the calling array.
- `Array.forEach()`: Executes a provided function once for each array element but does not return a new array.

Code Example:

javascript

Copy code

```
const numbers = [1, 2, 3];

const doubled = numbers.map(num => num * 2); // [2, 4, 6]
console.log(doubled);

numbers.forEach(num => console.log(num * 2)); // Outputs: 2, 4, 6
```

○

38. What is the purpose of the `setTimeout()` function?

- **Theory:** The `setTimeout()` function sets a timer which executes a function or specified piece of code once the timer expires.

Code Example:

```
javascript
Copy code
console.log('Start');
setTimeout(() => {
    console.log('This runs after 2 seconds');
}, 2000);
console.log('End');
// Output: Start, End, This runs after 2 seconds
```

○

39. What are rest parameters and the spread operator?

○ **Theory:**

- **Rest Parameters:** Allows a function to accept an indefinite number of arguments as an array.
- **Spread Operator:** Allows an iterable to be expanded in places where 0 or more arguments or elements are expected.

Code Example:

```
javascript
Copy code
// Rest Parameters
function sum(...numbers) {
    return numbers.reduce((acc, num) => acc + num, 0);
}
console.log(sum(1, 2, 3, 4)); // Output: 10

// Spread Operator
const arr1 = [1, 2];
const arr2 = [3, 4];
const combined = [...arr1, ...arr2];
console.log(combined); // Output: [1, 2, 3, 4]
```

○

40. What is the use of eval() function?

- **Theory:** The eval() function evaluates JavaScript code represented as a string. It can execute code dynamically, but its use is generally discouraged due to security and performance issues.

Code Example:

```
javascript
Copy code
const result = eval('2 + 2');
console.log(result); // Output: 4
```

○

These additional questions cover various aspects of JavaScript and can help you prepare for interviews at different levels. If you have specific topics or areas you'd like to focus on, just let me know!

4o mini

You said:

aur acche acche questions mil sakte hai

ChatGPT said:

ChatGPT

Absolutely! Here's a collection of more advanced and interesting JavaScript interview questions that you can use for preparation:

Advanced JavaScript Interview Questions

41. What are closures in JavaScript?

- **Theory:** A closure is a function that retains access to its outer scope even after the outer function has finished executing. Closures are commonly used to create private variables or functions.

Code Example:

```
javascript
Copy code
function createCounter() {
    let count = 0; // private variable
    return function() {
        count += 1;
        return count;
    };
}

const counter = createCounter();
console.log(counter()); // Output: 1
console.log(counter()); // Output: 2
```

○

42. What is the concept of prototypal inheritance?

- **Theory:** Prototypal inheritance allows objects to inherit properties and methods from other objects. In JavaScript, every object has a prototype, and properties can be accessed on this prototype chain.

Code Example:

```
javascript
```

Copy code

```
const animal = {
  speak: function() {
    console.log('Animal speaks');
  }
};

const dog = Object.create(animal);
dog.speak(); // Output: Animal speaks
```

○

43. What is the difference between `null` and `undefined`?

- **Theory:**

- `null` is an assignment value that represents the intentional absence of any object value.
- `undefined` is a type that indicates a variable has been declared but has not yet been assigned a value.

Code Example:

javascript

Copy code

```
let a; // undefined
let b = null; // null
console.log(a); // Output: undefined
console.log(b); // Output: null
```

○

44. What is the difference between `==` and `===`?

- **Theory:**

- `==` (abstract equality) compares two values for equality after converting both values to a common type.
- `===` (strict equality) compares both value and type, without performing type conversion.

Code Example:

javascript

Copy code

```
console.log(0 == '0'); // Output: true
console.log(0 === '0'); // Output: false
```

○

45. What are JavaScript modules, and why are they used?

- **Theory:** JavaScript modules allow for code encapsulation, better organization, and reusability. They enable you to export variables and functions from one module and import them into another.

Code Example:

```
javascript
Copy code
// module.js
export const name = 'Alice';
export function greet() {
    console.log(`Hello, ${name}`);
}

// main.js
import { name, greet } from './module.js';
greet(); // Output: Hello, Alice
```

○

46. What is the `this` keyword, and how does its value change?

- **Theory:** The `this` keyword refers to the object from which a function was called. Its value can change based on the context in which a function is called.

Code Example:

```
javascript
Copy code
const obj = {
    name: 'John',
    greet: function() {
        console.log(`Hello, ${this.name}`);
    }
};
obj.greet(); // Output: Hello, John

const greetFunc = obj.greet;
greetFunc(); // Output: Hello, undefined (or global variable if in
non-strict mode)
```

○

47. What is the purpose of the `super` keyword?

- **Theory:** The `super` keyword is used to call functions on an object's parent. It is often used in class constructors to access the parent class's methods or properties.

Code Example:

```
javascript
Copy code
class Animal {
    constructor(name) {
```

```

        this.name = name;
    }
    speak() {
        console.log(`"${this.name}" makes a noise.`);
    }
}

class Dog extends Animal {
    speak() {
        super.speak(); // Call the parent's method
        console.log(`"${this.name}" barks.`);
    }
}

const dog = new Dog('Rover');
dog.speak(); // Output: Rover makes a noise. Rover barks.

```

○ 48. What is the difference between synchronous and asynchronous programming in JavaScript?

- Theory:
 - **Synchronous:** Code execution is sequential; one operation must complete before the next begins.
 - **Asynchronous:** Operations can occur independently of the main program flow, allowing other code to run while waiting for a result (e.g., network requests).

Code Example:

```

javascript
Copy code
console.log('Start'); // Synchronous

setTimeout(() => {
    console.log('Asynchronous operation'); // Asynchronous
}, 1000);

console.log('End'); // Synchronous
// Output: Start, End, Asynchronous operation

```

○ 49. What are Promises, and how do they work?

- **Theory:** A Promise is an object representing the eventual completion or failure of an asynchronous operation and its resulting value. Promises can be in one of three states: pending, fulfilled, or rejected.

Code Example:

javascript

Copy code

```
const myPromise = new Promise((resolve, reject) => {
    const success = true;
    if (success) {
        resolve('Operation was successful!');
    } else {
        reject('Operation failed.');
    }
});

myPromise
    .then(result => console.log(result)) // Output: Operation was
successful!
    .catch(error => console.error(error));
```

○

50. What is the `async` function and how does it work?

- **Theory:** An `async` function is a function that returns a Promise and allows the use of `await` within it, which pauses the execution of the function until the Promise is resolved or rejected.

Code Example:

javascript

Copy code

```
async function fetchData() {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    return data;
}
```

```
fetchData().then(data => console.log(data));
```

○

Additional Topics**51. What is the event delegation pattern?**

- **Theory:** Event delegation is a technique in which a single event listener is added to a parent element instead of multiple listeners on child elements. This allows you to manage events more efficiently.

Code Example:

javascript

Copy code

```
const list = document.getElementById('myList');
list.addEventListener('click', (event) => {
  if (event.target.tagName === 'LI') {
    console.log(`Item clicked: ${event.target.textContent}`);
  }
});
```

○

52. What is memoization in JavaScript?

- **Theory:** Memoization is an optimization technique that stores the results of expensive function calls and returns the cached result when the same inputs occur again.

Code Example:

javascript

Copy code

```
function memoizedAdd() {
  const cache = {};
  return function(x) {
    if (cache[x]) {
      return cache[x];
    }
    const result = x + 10; // Expensive operation
    cache[x] = result;
    return result;
  };
}

const add = memoizedAdd();
console.log(add(5)); // Output: 15 (calculated)
console.log(add(5)); // Output: 15 (cached)
```

○

53. What are Web Workers in JavaScript?

- **Theory:** Web Workers allow you to run scripts in background threads, enabling parallel processing without blocking the main thread, which helps in improving performance for heavy computations.

Code Example:

javascript

Copy code

```
// worker.js
self.onmessage = function(e) {
  const result = e.data * 2;
```

```
        self.postMessage(result);
    };

// main.js
const worker = new Worker('worker.js');
worker.onmessage = function(e) {
    console.log('Result from worker:', e.data);
};
worker.postMessage(10); // Sends 10 to the worker
```

○

54. What are service workers?

- **Theory:** Service workers are scripts that run in the background of a web page and enable features like caching, background sync, and push notifications, significantly enhancing web applications.

Code Example:

javascript

Copy code

```
// Registering a service worker
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/service-worker.js')
        .then(() => {
            console.log('Service Worker registered!');
        });
}
```

○

55. What is the difference between a stack and a queue?

- **Theory:**
 - **Stack:** A data structure that follows the Last In First Out (LIFO) principle. Elements can be added or removed from the top.
 - **Queue:** A data structure that follows the First In First Out (FIFO) principle. Elements are added at the end and removed from the front.

Code Example:

javascript

Copy code

```
// Stack
class Stack {
    constructor() {
        this.items = [];
    }
    push(element) {
        this.items.push(element);
```

```
        }
    pop() {
        return this.items.pop();
    }
}

// Queue
class Queue {
    constructor() {
        this.items = [];
    }
    enqueue(element) {
        this.items.push(element);
    }
    dequeue() {
        return this.items.shift();
    }
}
```

○

These questions cover a wide range of JavaScript topics and should help you prepare effectively for your interviews. If you'd like more questions or need help with specific topics, just let me know!

40 mini

You said:

aur bhee questions hai iss k alawa

ChatGPT said:

ChatGPT

Certainly! Here are more JavaScript interview questions covering various concepts, including advanced topics and practical scenarios:

More JavaScript Interview Questions

56. What is the difference between `let`, `const`, and `var`?

○ **Theory:**

- `var` is function-scoped and can be redeclared.
- `let` is block-scoped and cannot be redeclared in the same scope.
- `const` is also block-scoped but must be initialized and cannot be reassigned.

Code Example:

javascript

Copy code

```
var a = 10;
let b = 20;
const c = 30;

a = 15; // valid
b = 25; // valid
// c = 35; // Uncaught TypeError: Assignment to constant variable.

console.log(a, b, c); // Output: 15 25 30
```

○

57. What is event bubbling and event capturing?

○ **Theory:**

- **Event Bubbling:** The event starts from the target element and bubbles up to the root.
- **Event Capturing:** The event starts from the root and captures down to the target element.

Code Example:

html

Copy code

```
<div id="parent">
    <button id="child">Click me</button>
</div>

<script>
    document.getElementById('parent').addEventListener('click', () => {
        console.log('Parent clicked!');
    });

    document.getElementById('child').addEventListener('click', () => {
        console.log('Child clicked!');
    });
</script>
```

○

58. What are the different types of error handling in JavaScript?

○ **Theory:**

- **Try/Catch:** Handles exceptions in synchronous code.
- **Promise.catch():** Handles errors in asynchronous code.

Code Example:

javascript

Copy code

```
try {
    throw new Error('Something went wrong!');
} catch (error) {
    console.error(error.message); // Output: Something went wrong!
}
```

```
Promise.reject('Error occurred')
    .catch(error => console.error(error)); // Output: Error occurred
```

○

59. Explain the concept of "this" in different contexts (global, object, function).

○ **Theory:**

- In the global context, `this` refers to the global object (window in browsers).
- In an object method, `this` refers to the object itself.
- In a regular function, `this` refers to undefined in strict mode or the global object in non-strict mode.

Code Example:

javascript

Copy code

```
console.log(this); // Global object (window)
```

```
const obj = {
    name: 'John',
    greet: function() {
        console.log(this.name); // 'this' refers to obj
    }
};
obj.greet(); // Output: John

function showThis() {
    console.log(this); // undefined in strict mode
}
showThis();
```

○

60. What is the purpose of the `bind()`, `call()`, and `apply()` methods?

○ **Theory:**

- `bind()`: Creates a new function that, when called, has its `this` keyword set to the provided value.

- `call()`: Calls a function with a specified `this` value and arguments provided individually.
- `apply()`: Calls a function with a specified `this` value and arguments provided as an array.

Code Example:

javascript

Copy code

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

const person = { name: 'Alice' };

greet.call(person, 'Hello'); // Output: Hello, Alice
greet.apply(person, ['Hi']); // Output: Hi, Alice

const boundGreet = greet.bind(person);
boundGreet('Hey'); // Output: Hey, Alice
```

○

61. What is a "debounce" function?

- **Theory:** A debounce function limits the rate at which a function can fire. It ensures that the function is not called again until a specified amount of time has passed since the last time it was invoked.

Code Example:

javascript

Copy code

```
function debounce(func, delay) {
  let timeout;
  return function(...args) {
    clearTimeout(timeout);
    timeout = setTimeout(() => func.apply(this, args), delay);
  };
}

const handleResize = debounce(() => {
  console.log('Resized!');
}, 200);

window.addEventListener('resize', handleResize);
```

○

62. What is a "throttle" function?

- **Theory:** A throttle function ensures that a function is called at most once in a specified time interval, which helps improve performance by limiting the number of times a function can execute.

Code Example:

javascript

Copy code

```
function throttle(func, limit) {  
    let lastFunc;  
    let lastRan;  
  
    return function(...args) {  
        if (!lastRan) {  
            func.apply(this, args);  
            lastRan = Date.now();  
        } else {  
            clearTimeout(lastFunc);  
            lastFunc = setTimeout(() => {  
                if ((Date.now() - lastRan) >= limit) {  
                    func.apply(this, args);  
                    lastRan = Date.now();  
                }  
            }, limit - (Date.now() - lastRan));  
        }  
    };  
}  
  
const log = throttle(() => {  
    console.log('Logging!');  
}, 2000);  
  
window.addEventListener('scroll', log);
```

○

63. What are the key differences between arrow functions and regular functions?

- **Theory:**

- Arrow functions do not have their own `this`, which is lexically inherited from the surrounding code.
- They cannot be used as constructors and do not have the `arguments` object.

Code Example:

javascript

Copy code

```
const obj = {
    value: 42,
    regularFunction: function() {
        console.log(this.value);
    },
    arrowFunction: () => {
        console.log(this.value);
    }
};

obj.regularFunction(); // Output: 42
obj.arrowFunction(); // Output: undefined (or global value)
```

○

64. What is the `async/await` syntax, and how does it improve code readability?

- **Theory:** `async/await` is a syntax for working with Promises that allows you to write asynchronous code in a synchronous style, improving readability and reducing complexity.

Code Example:

javascript

Copy code

```
async function fetchData() {
    try {
        const response = await
fetch('https://api.example.com/data');
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error:', error);
    }
}

fetchData().then(data => console.log(data));
```

○

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

- **Theory:**
 - Object literals
 - Constructor functions
 - `Object.create()`
 - ES6 classes

Code Example:

```
javascript
Copy code
// Object literal
const obj1 = { name: 'Alice' };

// Constructor function
function Person(name) {
    this.name = name;
}
const obj2 = new Person('Bob');

// Object.create()
const obj3 = Object.create(obj1);

// ES6 class
class Animal {
    constructor(name) {
        this.name = name;
    }
}
const obj4 = new Animal('Cat');
```

○

66. Explain the concept of the "Event Loop" in JavaScript.

- **Theory:** The event loop is a mechanism that allows JavaScript to perform non-blocking I/O operations. It enables the execution of asynchronous code, managing the call stack and the message queue.

Code Example:

```
javascript
Copy code
console.log('Start');

setTimeout(() => {
    console.log('Timeout');
}, 0);

console.log('End');

// Output: Start, End, Timeout
```

○

67. What is the difference between deep copy and shallow copy?

- **Theory:**
 - **Shallow Copy:** Creates a new object that is a copy of the original object but shares references to nested objects.
 - **Deep Copy:** Creates a new object and recursively copies all nested objects.

Code Example:

javascript

Copy code

```
const original = { a: 1, b: { c: 2 } };

// Shallow copy
const shallowCopy = { ...original };
shallowCopy.b.c = 3;
console.log(original.b.c); // Output: 3 (shared reference)

// Deep copy using JSON
const deepCopy = JSON.parse(JSON.stringify(original));
deepCopy.b.c = 4;
console.log(original.b.c); // Output: 3 (independent copy)
```

○

68. What are higher-order functions?

- **Theory:** Higher-order functions are functions that either take other functions as arguments or return functions as their result. They enable functional programming concepts like composition and currying.

Code Example:

javascript

Copy code

```
function higherOrderFunction(fn) {
  return function(...args) {
    console.log('Before executing the function');
    const result = fn(...args);
    console.log('After executing the function');
    return result;
  };
}

const add = (a, b) => a + b;
const wrappedAdd = higherOrderFunction(add);
console.log(wrappedAdd(2, 3)); // Output: Before executing the
function, After executing the function, 5
```

○

69. Explain the difference between `==` and `===`.

o **Theory:**

- `==` (loose equality) compares values for equality after performing type conversion.
- `===` (strict equality) compares values for equality without type conversion.

Code Example:

javascript

Copy code

```
console.log(5 == '5');    // Output: true (type conversion)
console.log(5 === '5');   // Output: false (no type conversion)
```

o

70. What is a closure in JavaScript?

- o **Theory:** A closure is a function that has access to its outer enclosing scope, even after the outer function has finished executing. It allows for data encapsulation and maintaining state.

Code Example:

javascript

Copy code

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

const counter = outerFunction();
console.log(counter()); // Output: 1
console.log(counter()); // Output: 2
```

o

These questions should further enhance your preparation for JavaScript interviews! If you need more or specific areas, feel free to ask!

40 mini

You said:

abhi jo questions tumne mujha deya hai keya iss k alawa aur questions hai

ChatGPT said:

ChatGPT

Sure! Here are even more JavaScript interview questions, covering additional concepts and scenarios:

Additional JavaScript Interview Questions

71. What are Promises in JavaScript, and how do they work?

- **Theory:** A Promise is an object representing the eventual completion (or failure) of an asynchronous operation and its resulting value.

Code Example:

javascript

Copy code

```
const myPromise = new Promise((resolve, reject) => {
  const success = true; // Simulate success/failure
  if (success) {
    resolve('Operation succeeded!');
  } else {
    reject('Operation failed.');
  }
});
```

myPromise

```
.then(result => console.log(result)) // Output: Operation
succeeded!
.catch(error => console.log(error));
```

○

72. What is the purpose of the `finally()` method in Promises?

- **Theory:** The `finally()` method executes a specified block of code once a Promise is settled, regardless of its outcome (resolved or rejected).

Code Example:

javascript

Copy code

myPromise

```
.then(result => console.log(result))
.catch(error => console.log(error))
.finally(() => console.log('Promise settled.'));
```

○

73. What is the difference between `for...of` and `for...in` loops?

- **Theory:**

- `for...in` is used to iterate over the enumerable properties of an object.

- `for...of` is used to iterate over iterable objects like arrays, strings, and NodeLists.

Code Example:

javascript

Copy code

```
const arr = [1, 2, 3];
const obj = { a: 1, b: 2 };

for (const index in obj) {
    console.log(index); // Output: a, b
}

for (const value of arr) {
    console.log(value); // Output: 1, 2, 3
}
```

○

74. What is the spread operator, and how is it used?

- **Theory:** The spread operator (`...`) allows an iterable such as an array or string to be expanded in places where zero or more arguments or elements are expected.

Code Example:

javascript

Copy code

```
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
const combined = [...arr1, ...arr2]; // Combine arrays
console.log(combined); // Output: [1, 2, 3, 4, 5, 6]

const obj1 = { a: 1 };
const obj2 = { b: 2 };
const combinedObj = { ...obj1, ...obj2 }; // Combine objects
console.log(combinedObj); // Output: { a: 1, b: 2 }
```

○

75. What is the rest operator?

- **Theory:** The rest operator (`...`) collects all remaining elements into an array. It is used in function arguments to capture a variable number of arguments.

Code Example:

javascript

Copy code

```
function sum(...numbers) {
```

```
        return numbers.reduce((total, num) => total + num, 0);
    }

console.log(sum(1, 2, 3, 4)); // Output: 10
```

○

76. What is the difference between `null` and `undefined`?

- **Theory:**

- `null` is an assignment value representing "no value" or "empty".
- `undefined` is a type itself, indicating that a variable has been declared but not yet assigned a value.

Code Example:

javascript

Copy code

```
let a; // undefined
let b = null; // null

console.log(a); // Output: undefined
console.log(b); // Output: null
```

○

77. What is the `Object.freeze()` method?

- **Theory:** `Object.freeze()` prevents modifications to an object, including adding or removing properties and changing existing property values.

Code Example:

javascript

Copy code

```
const obj = { name: 'Alice' };
Object.freeze(obj);
obj.name = 'Bob'; // This will not work
console.log(obj.name); // Output: Alice
```

○

78. Explain the difference between synchronous and asynchronous code.

- **Theory:**

- Synchronous code runs sequentially, blocking further execution until the current task completes.
- Asynchronous code allows multiple tasks to run concurrently, enabling the program to continue executing other tasks while waiting for a long-running operation to finish.

Code Example:

javascript

Copy code

```
console.log('Start');

setTimeout(() => {
    console.log('Asynchronous task completed');
}, 1000);

console.log('End'); // This will execute before the asynchronous task
```

○

79. What are modules in JavaScript, and how are they used?

- **Theory:** Modules are reusable pieces of code that can export variables and functions from one module and import them into another, promoting code organization and reusability.

Code Example:

javascript

Copy code

```
// module.js
export const greeting = 'Hello, World!';
export function greet() {
    console.log(greeting);
}
```

```
// main.js
import { greeting, greet } from './module.js';
console.log(greeting); // Output: Hello, World!
greet(); // Output: Hello, World!
```

○

80. What is the difference between `slice()` and `splice()` methods in arrays?

- **Theory:**

- `slice()` returns a shallow copy of a portion of an array without modifying the original array.
- `splice()` changes the contents of an array by removing or replacing existing elements and/or adding new elements.

Code Example:

javascript

Copy code

```
const arr = [1, 2, 3, 4, 5];

const sliced = arr.slice(1, 4); // [2, 3, 4]
console.log(sliced);
```

```
console.log(arr); // Original array remains unchanged  
  
arr.splice(1, 2); // Removes 2 elements starting from index 1  
console.log(arr); // Output: [1, 4, 5]
```

○

81. What is the `instanceof` operator, and how does it work?

- **Theory:** The `instanceof` operator checks whether an object is an instance of a specific constructor or class, returning `true` or `false`.

Code Example:

javascript

Copy code

```
class Person {}  
const person = new Person();  
console.log(person instanceof Person); // Output: true  
console.log(person instanceof Object); // Output: true
```

○

82. What are default parameters in functions?

- **Theory:** Default parameters allow named parameters to be initialized with default values if no value or `undefined` is passed.

Code Example:

javascript

Copy code

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

```
console.log(multiply(5)); // Output: 5 (5 * 1)  
console.log(multiply(5, 2)); // Output: 10 (5 * 2)
```

○

83. What is a Symbol in JavaScript?

- **Theory:** A Symbol is a unique and immutable primitive value that can be used as a key for object properties, providing a way to create private properties.

Code Example:

javascript

Copy code

```
const uniqueKey = Symbol('key');  
const obj = {  
    [uniqueKey]: 'value'
```

```
};

console.log(obj[uniqueKey]); // Output: value
```

○

84. Explain the concept of currying in JavaScript.

- **Theory:** Currying is the technique of converting a function that takes multiple arguments into a series of functions that each take a single argument.

Code Example:

javascript

Copy code

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

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

○

85. What is the difference between `map()`, `filter()`, and `reduce()` methods in arrays?

- **Theory:**

- `map()`: Creates a new array populated with the results of calling a provided function on every element in the calling array.
- `filter()`: Creates a new array with all elements that pass the test implemented by the provided function.
- `reduce()`: Executes a reducer function on each element of the array, resulting in a single output value.

Code Example:

javascript

Copy code

```
const numbers = [1, 2, 3, 4, 5];

const doubled = numbers.map(num => num * 2); // [2, 4, 6, 8, 10]
const evens = numbers.filter(num => num % 2 === 0); // [2, 4]
const sum = numbers.reduce((acc, num) => acc + num, 0); // 15

console.log(doubled, evens, sum);
```

○

86. What is the difference between block scope and function scope?

- **Theory:**

- Function scope means that variables declared within a function are only accessible within that function.
- Block scope means that variables declared with `let` or `const` are only accessible within the block in which they are defined.

Code Example:

javascript

Copy code

```
function example() {  
    var functionScoped = 'I am function scoped';  
    if (true) {  
        let blockScoped = 'I am block scoped';  
        console.log(blockScoped); // Output: I am block scoped  
    }  
    console.log(functionScoped); // Output: I am function scoped  
    // console.log(blockScoped); // Error: blockScoped is not  
defined  
}  
example();
```

○

87. What is `this` in JavaScript?

- **Theory:** The value of `this` is determined by how a function is called. In a method, `this` refers to the object that the method belongs to. In a regular function, it refers to the global object (or `undefined` in strict mode).

Code Example:

javascript

Copy code

```
const obj = {  
    name: 'Alice',  
    greet: function() {  
        console.log(`Hello, ${this.name}`);  
    }  
};  
obj.greet(); // Output: Hello, Alice  
  
function showThis() {  
    console.log(this);  
}  
showThis(); // Output: Window (or global object in Node.js)
```

○

88. What are higher-order functions?

- **Theory:** Higher-order functions are functions that take other functions as arguments or return functions as their result.

Code Example:

javascript

Copy code

```
function operation(fn, a, b) {  
    return fn(a, b);  
  
}  
  
function add(x, y) {  
    return x + y;  
}  
  
console.log(operation(add, 5, 3)); // Output: 8
```

○

89. Explain what `setTimeout()` and `setInterval()` do.

- **Theory:**

- `setTimeout()` executes a function after a specified number of milliseconds.
- `setInterval()` repeatedly executes a function at specified intervals.

Code Example:

javascript

Copy code

```
setTimeout(() => {  
    console.log('Executed after 2 seconds');  
, 2000);  
  
let count = 0;  
const intervalId = setInterval(() => {  
    count++;  
    console.log(`Count: ${count}`);  
    if (count === 5) clearInterval(intervalId); // Stops after 5  
counts  
, 1000);
```

○

90. What is a template literal in JavaScript?

- **Theory:** Template literals are string literals that allow embedded expressions. They use backticks (`) instead of single or double quotes and can span multiple lines.

Code Example:

```
javascript
Copy code
const name = 'Alice';
const greeting = `Hello, ${name}!`;
console.log(greeting); // Output: Hello, Alice!
```

○

Conclusion

These questions should give you a comprehensive understanding of JavaScript and prepare you for interviews from fresher to experienced levels. If you want even more questions or specific areas to focus on, just let me know!

40 mini
ChatGPT ca