|  |  |
| --- | --- |
| Basics **Question 1 . What is JavaScript, and what is its primary use in web development?**  Ans. JavaScript is a high-level, interpreted programming language used primarily for enhancing the interactivity and functionality of web pages. It allows developers to manipulate the Document Object Model (DOM) of a web page, handle user interactions, validate input, and make asynchronous requests to servers, making web applications dynamic and responsive.  **Question 2. Explain the difference between let, const, and var when declaring variables.**  Ans. 1.var is function-scoped and hoisted to the top of the function or global context. It can be reassigned and used before declaration. 2. let is block-scoped, meaning it's limited to the block it's declared in (e.g., inside a loop or an if statement). It's not hoisted and can be reassigned. 3.const is also block-scoped but represents a constant value that cannot be reassigned after initialization. It's not hoisted and should be used when the variable's value won't change.  **Question 3. How do you write a comment in JavaScript?**  Ans. You can write comments in JavaScript using two methods: Single-line comments using //, e.g., // This is a comment. Multi-line comments using /\* \*/  **Question 4. What is the purpose of the console.log() method in JavaScript?**  Ans. The console.log() method is used for debugging and logging messages to the browser's console. It's a helpful tool for developers to inspect values, variables, and the flow of their code during development.  **Question 5. What is the significance of the typeof operator in JavaScript?**  Ans. The typeof operator is used to determine the data type of a value or expression in JavaScript. It returns a string representing the data type, such as "number", "string", "object", "function", or "undefined".  **Question 6. How can you create a function in JavaScript?**  Ans. "You can create a function in JavaScript using the function keyword or by using arrow functions (introduced in ES6). For example: Function declaration:  Javascript  function add(a, b) {  return a + b;  }  Arrow function:const add = (a, b) => a + b;  **Question 7. What is a JavaScript event handler, and how do you use it?**  Ans. A JavaScript event handler is a function that responds to specific events, such as user interactions (e.g., clicks, keypresses). You can attach event handlers to HTML elements using event attributes (e.g., onclick, onmouseover) or through JavaScript code using methods like addEventListener().  **Question 8.How do you add an element to an array in JavaScript?**  Ans. You can add an element to an array in JavaScript using various methods, such as: push(): Adds elements to the end of the array. unshift(): Adds elements to the beginning of the array. splice(): Adds elements at a specific position by specifying the index. concat(): Combines arrays to create a new array.  **Question 9. Explain the difference between == and === in JavaScript for equality comparison.**  Ans. (== loose equality) compares values for equality after performing type coercion if necessary. For example, 1 == '1' is true, (===strict equality) compares both values and types. It only returns true if both the value and type are the same. For example, 1 === '1' is false  **Question 10. What is the DOM (Document Object Model) in web development?**  Ans. The Document Object Model (DOM) is a programming interface provided by browsers that represents the structure of a web page as a hierarchical tree of objects. It allows JavaScript to interact with and manipulate the content and structure of a web page dynamically. Developers can access, modify, or create HTML elements and attributes using the DOM.  **Question 11. How do you select an HTML element in JavaScript using its id attribute?**  Ans. You can select an HTML element by its id attribute using the getElementById() method, like this: const element = document.getElementById('myElementId');  **Question 12. What is an anonymous function in JavaScript?**  Ans. An anonymous function is a function that does not have a name. It can be defined and used without assigning it a name. Anonymous functions are often used as callbacks or for immediately invoked function expressions (IIFE).  **Question 13. How do you use setTimeout() in JavaScript to delay code execution?**  Ans. "The setTimeout() function is used to execute a specified function or code block after a specified delay (in milliseconds). Here's an example of its usage:  Javascript  setTimeout(function () {  console.log('Delayed code executed.');  }, 2000); // Executes after a 2-second delay"  **Question 14. Explain the purpose of the null value in JavaScript.**  Ans. The null value in JavaScript represents the intentional absence of any object value or the absence of a value. It is often used to indicate that a variable or property should have no value or is not initialized with any meaningful data.  **Question 15. What are the benefits of using strict mode in JavaScript?**  Ans. Strict mode, enabled by adding "use strict"; at the beginning of a script or function, offers several benefits, including:1.Catches common coding mistakes and "unsafe" actions. 2.Prevents the use of undeclared variables. 3. Disallows reserved words as variable or function names. 4. Restricts certain features, such as assigning to read-only properties. 5. Makes code more optimized for modern JavaScript engines.   |  | | --- | | function notStrictFunction() {  undeclaredVariable = 10; // This will not throw an error in non-strict mode  return undeclaredVariable;  }    console.log(notStrictFunction()); // Outputs 10  "use strict";  function strictFunction() {  undeclaredVariable = 10; // This will throw an error in strict mode  return undeclaredVariable;  }  // This line will not be reached due to the error above  console.log(strictFunction()); |   **Question 16. How can you prevent the default behavior of an HTML form submit event?**  Ans. To prevent the default behavior of an HTML form submit event, you can use the event.preventDefault() method within an event handler. This prevents the form from being submitted, allowing you to perform custom actions instead.  **Question 17. How do you concatenate strings in JavaScript?**  Ans. "You can concatenate strings in JavaScript using the + operator or the concat() method. Here are examples of both methods: 1. Using the + operator:  Javascript  const str1 = 'Hello';  const str2 = 'World';  const concatenated = str1 + ' ' + str2; // Concatenates ""Hello"" and ""World"" 2. Using the concat() method: const str1 = 'Hello';  const str2 = 'World';  const concatenated = str1.concat(' ', str2); // Concatenates ""Hello"" and ""World"""  **Question 18. Explain the purpose of the JSON.stringify() and JSON.parse() methods in JavaScript.**  Ans. "JSON.stringify(): This method is used to convert a JavaScript object into a JSON-formatted string. It serializes the object, including its properties and values, into a string representation. This is commonly used when sending data to a server or storing data in a file. 1.  Javascript  const person = { name: 'Alice', age: 30 };  const jsonString = JSON.stringify(person);  // jsonString is '{""name"":""Alice"",""age"":30}'  JSON.parse(): This method is used to parse a JSON-formatted string and convert it into a JavaScript object. It takes a JSON string as input and returns a corresponding JavaScript object.  Javascript  const jsonString = '{""name"":""Bob"",""age"":25}';  const person = JSON.parse(jsonString);  // person is { name: 'Bob', age: 25 }"  **Question 19. What is the difference between null and undefined in JavaScript?**  Ans. null is a value in JavaScript that represents the intentional absence of any object value or a value that indicates no value or no object. It is explicitly assigned to indicate the absence of a value. undefined is a primitive value in JavaScript that indicates a variable has been declared but has not been assigned a value. It is the default value for function parameters that are not provided and for uninitialized variables.  **Question 20. What will be the output of the following code?**  Javascript  function foo() {  setTimeout(() => {  console.log('Timeout');  }, 0);  Promise.resolve().then(() => {  console.log('Promise');  });  console.log('Function');  }  foo();  console.log('End');  Ans. Expected Output:  Function  End  Promise  Timeout  Explanation: This demonstrates JavaScript's event loop behavior. The order of execution is such that the synchronous code (Function and End) is executed first, then microtasks (Promise), and finally the setTimeout callback (Timeout)." Intermediate **Question 1.What is scope in JavaScript, and how does it work?**  Ans. Scope refers to the context in which variables are declared and accessed. JavaScript has function scope and block scope. Function scope means variables declared inside a function are only accessible within that function. Block scope (introduced with let and const in ES6) restricts variable access to the block in which they are defined.  **Question 2. How do you handle asynchronous operations in JavaScript?**  Ans. Asynchronous operations in JavaScript can be handled using callbacks, promises, or the async/await syntax (ES6+). Callbacks are functions passed as arguments to be executed when an asynchronous task completes. Promises provide a structured way to handle async operations, allowing you to chain .then() and .catch() for handling success and errors. async/await is a syntax for working with promises in a more synchronous-like manner.  **Question 3. Explain the concept of closures in JavaScript.**  Ans. Closures occur when an inner function has access to the variables and scope of an outer (enclosing) function, even after the outer function has finished executing. Closures are commonly used for data encapsulation, maintaining private data, and creating factory functions. They preserve the scope chain of the enclosing function.  **Question 4. What is a callback function in JavaScript, and when would you use one?**  Ans. A callback function is a function that is passed as an argument to another function and is executed after the completion of that function. Callbacks are used for asynchronous operations, event handling, and to ensure that certain code runs after a specific task completes. They are a fundamental building block for handling async code in JavaScript.  **Question 5. How do you declare a JavaScript class and create instances of it?**  Ans. Classes can be declared in JavaScript using the class keyword. You can define a constructor and methods within the class. To create instances, use the new keyword followed by the class name. For example:  Javascript  class Person {  constructor(name) {  this.name = name;  }  sayHello() {  console.log(`Hello, my name is ${this.name}`);  }  }  const person = new Person('Alice');  person.sayHello(); // Output: Hello, my name is Alice"  **Question 6. Describe the event delegation pattern in JavaScript.**  Ans. Event delegation is a design pattern where a single event handler is attached to a common ancestor element of multiple child elements that you want to handle events for. It allows you to handle events on dynamically created elements and reduces the number of event listeners in your application. The event handler checks the target of the event to determine which child element triggered the event.  **Question 7. How do you loop through the properties of an object in JavaScript?**  Ans. You can loop through object properties using a for...in loop or by using Object.keys(), Object.values(), and Object.entries() methods introduced in ES6. Here's an example using for...in:  Javascript  const obj = { a: 1, b: 2, c: 3 };  for (const key in obj) {  if (obj.hasOwnProperty(key)) {  console.log(`${key}: ${obj[key]}`);  }  }  **Question 8. What is the difference between a shallow copy and a deep copy of an object?**  Ans. A shallow copy of an object creates a new object with its own properties, but if the properties are objects themselves, they are still references to the same objects in memory as the original object. A deep copy, on the other hand, creates a completely new object with its own properties and recursively copies nested objects, ensuring that they are also deep copies with no shared references.  **Question 9. Explain what the "this" keyword refers to in different contexts in JavaScript.**  Ans. The behavior of the this keyword in JavaScript depends on the context in which it is used: 1. In a method of an object, this refers to the object itself. 2. In a regular function (not in strict mode), this usually refers to the global object (e.g., window in a browser). 3. In an event handler, this often refers to the DOM element that triggered the event. 4. When using arrow functions, this retains the value of the enclosing lexical scope.  **Question 10. How do you handle errors in JavaScript using try...catch?**  Ans. Errors in JavaScript can be handled using a try...catch block. The try block contains code that might throw an error, and the catch block is executed if an error occurs. You can catch and handle specific types of errors using catch clauses for different error types.  Javascript  try {  // Code that may throw an error  const result = someFunction();  } catch (error) {  // Handle the error  console.error(error.message);  }  **Question 11. What is the purpose of the map() function in JavaScript, and how does it work?**  Ans. The map() function is used to create a new array by applying a provided function to each element of an existing array. It iterates through each item in the array, applies the function, and returns a new array with the results. It doesn't modify the original array.  Javascript  const numbers = [1, 2, 3];  const squared = numbers.map((num) => num \* num);  // Result: [1, 4, 9]"  **Question 12. How can you check if an element exists in an array in JavaScript?**  Ans. To check if an element exists in an array, you can use methods like indexOf(), includes(), or find(). For example:  Javascript  const myArray = [1, 2, 3, 4];  const element = 3;  // Using indexOf()  const exists = myArray.indexOf(element) !== -1;  // Using includes()  const exists = myArray.includes(element);  // Using find()  const exists = myArray.find(item => item === element) !== undefined;  **Question 13. Explain the difference between null, undefined, and undeclared variables.**  Ans. null is a value in JavaScript that represents the intentional absence of any object value. It is a value that a variable can be explicitly assigned to.  undefined is a special value in JavaScript that represents the absence of a value, often the default value of uninitialized variables.  undeclared variables are variables that have not been declared using any keyword (var, let, const). Accessing an undeclared variable will result in a ReferenceError.  **Question 14. How do you implement inheritance in JavaScript using prototypes?**  Ans. Inheritance in JavaScript is achieved by setting an object's prototype to another object. The prototype property of a constructor function is used to create a prototype chain. Here's an example:  Javascript  function Animal(name) {  this.name = name;  }  Animal.prototype.eat = function () {  console.log(`${this.name} is eating.`);  };  function Dog(name, breed) {  Animal.call(this, name); // Call the parent constructor  this.breed = breed;  }  Dog.prototype = Object.create(Animal.prototype); // Inherit from Animal  Dog.prototype.constructor = Dog; // Set the correct constructor  const myDog = new Dog('Buddy', 'Golden Retriever');  myDog.eat(); // Output: Buddy is eating.  **Question 15. What is the event loop in JavaScript, and how does it work?**  Ans. The event loop is a core concept in JavaScript's concurrency model. It's responsible for managing the execution of code, handling asynchronous tasks, and maintaining responsiveness in web applications. The event loop continually checks the message queue for tasks, processes them in a non-blocking manner, and schedules callbacks for execution when the call stack is empty. This mechanism allows JavaScript to handle tasks like timers, I/O operations, and event handling in an asynchronous and non-blocking way.  **Question 16. Describe the concept of promises in JavaScript and how they help with asynchronous code.**  Ans. Promises are objects representing the eventual completion or failure of an asynchronous operation. They provide a structured way to work with asynchronous code, making it more readable and maintainable. A promise can be in one of three states: pending, fulfilled (resolved), or rejected. You can attach .then() and .catch() methods to a promise to handle success and error cases, respectively. Promises simplify error handling and allow you to chain multiple asynchronous operations together.  **Question 17. What is the purpose of the localStorage and sessionStorage objects in JavaScript?**  Ans. localStorage and sessionStorage are client-side storage mechanisms provided by web browsers for storing data in a key-value format. They are used to store data persistently (across browser sessions) or for the duration of a single session, respectively. They are often used to store user preferences, authentication tokens, or other application-specific data without the need for server-side storage.  **Question 18. How can you create and manipulate HTML elements dynamically in JavaScript?**  Ans. You can create and manipulate HTML elements dynamically using the Document Object Model (DOM) API in JavaScript. Here are some common methods: -document.createElement(): Create a new HTML element. -element.appendChild(): Add an element as a child to another element. -element.removeChild(): Remove a child element. -element.setAttribute(): Set an attribute of an element. -element.addEventListener(): Attach event listeners to elements. -element.innerHTML: Get or set the HTML content of an element. -element.style: Access and modify CSS styles of an element.  **Question 19. Explain the concept of currying in JavaScript functions.**  Ans. Currying is a functional programming technique where a function that takes multiple arguments is transformed into a series of unary (single-argument) functions. Each unary function takes one argument and returns another unary function or the final result. Currying allows you to partially apply arguments to a function, creating more specialized  Javascript  functions. // Non-curried function  function add(a, b) {  return a + b;  }  // Curried function  function curriedAdd(a) {  return function (b) {  return a + b;  };  }  const addTwo = curriedAdd(2);  const result = addTwo(3); // Result: 5"  **Question 20. What is a JavaScript IIFE (Immediately Invoked Function Expression)?**  Ans. An IIFE is a JavaScript function that is defined and executed immediately after its declaration. It is typically wrapped in parentheses to prevent it from being treated as a regular function declaration. IIFE is often used to create private scopes for variables and to encapsulate code to avoid polluting the global scope.  (function () {  // Code here  })();  **Question 21. Describe the differences between the map(), filter(), and reduce() array methods in JavaScript.**  Ans. map() is used to create a new array by applying a provided function to each element of an existing array. It returns a new array with the results.  filter() is used to create a new array containing elements that satisfy a specified condition. It returns a new array with elements that pass the condition.  reduce() is used to accumulate values of an array into a single value by applying a provided function that processes each element sequentially. It returns a single result.  Javascript  const numbers = [1, 2, 3, 4, 5];  // Using map()  const squared = numbers.map((num) => num \* num); // Result: [1, 4, 9, 16, 25]  // Using filter()  const evenNumbers = numbers.filter((num) => num % 2 === 0); // Result: [2, 4]  // Using reduce()  const sum = numbers.reduce((accumulator, num) => accumulator + num, 0); // Result: 15"  **Question 22. How do you remove duplicates from an array in JavaScript?**  Ans. You can remove duplicates from an array in JavaScript using various methods. One common approach is to use a Set to store unique values, and then convert the set back to an array.  Javascript  const arrayWithDuplicates = [1, 2, 2, 3, 4, 4, 5];  const uniqueArray = [...new Set(arrayWithDuplicates)];  // Result: [1, 2, 3, 4, 5]  **Question 23. What is the JavaScript event loop, and how does it handle asynchronous tasks?**  Ans. The JavaScript event loop is a mechanism that manages the execution of code in a non-blocking and asynchronous manner. It consists of two main components: the call stack and the message queue. When a script starts, it enters the call stack. Asynchronous tasks, such as timers, I/O operations, and events, are offloaded to the browser's APIs. When an asynchronous task completes, it places a message in the message queue. The event loop continuously checks the message queue and moves messages to the call stack when it is empty, allowing asynchronous tasks to be processed in the order they were scheduled.  **Question 24. How do you use the bind() method in JavaScript to set the value of "this" in a function?**  Ans. The bind() method in JavaScript is used to create a new function with a specified value for the this keyword. It allows you to explicitly set the value of this within the function when it is called. Here's an example:  Javascript  const person = {  name: 'Alice',  sayHello: function () {  console.log(`Hello, my name is ${this.name}`);  },  };  const greet = person.sayHello;  const boundGreet = greet.bind(person); // Binding the ""this"" value  boundGreet(); // Output: Hello, my name is Alice  **Question 25. Explain what the "hoisting" behavior is in JavaScript.**  Ans. Hoisting is a JavaScript behavior where variable and function declarations are moved to the top of their containing scope during the compilation phase. This means that you can use a variable or function before it's declared in your code, but it's important to understand that only the declarations are hoisted, not the initializations.  Javascript  console.log(x); // Outputs ""undefined"" (declaration is hoisted)  var x = 10; // Initialization  foo(); // Outputs ""Hello, World!"" (function declaration is hoisted)  function foo() {  console.log('Hello, World!');  }  **Question 26. How does the "use strict" directive affect JavaScript code?**  Ans. use strict is a pragma that enforces stricter parsing and error handling in JavaScript code. When used at the beginning of a script or function, it enables a set of rules and restrictions that help catch common coding mistakes and promote better code quality. Some of the effects of strict mode include disallowing the use of undeclared variables, preventing the use of reserved words, and disabling potentially unsafe features.  **Question 27. What are JavaScript promises, and how do they work with async/await?**  Ans. JavaScript promises are objects representing the eventual completion (or failure) of an asynchronous operation. Promises have three states: pending, fulfilled (resolved), and rejected. Promises work seamlessly with async/await, which is a syntax introduced in ES2017 for working with asynchronous code in a more synchronous-like manner. You can use await within an async function to pause the execution until a promise is resolved or rejected. This makes asynchronous code more readable and easier to reason about.  **Question 28. How can you check if an object is an array in JavaScript?**  Ans. You can use the Array.isArray(obj) method to check if an object is an array. It returns true if the object is an array and false otherwise.  Javascript  const myArray = [1, 2, 3];  console.log(Array.isArray(myArray)); // Output: true  const myObject = { name: 'Alice' };  console.log(Array.isArray(myObject)); // Output: false"  **Question 29. Explain the concept of prototypal inheritance in JavaScript.**  Ans. In prototypal inheritance, objects inherit properties and methods from a prototype object. Every object in JavaScript has a prototype, which is another object. When you access a property or method on an object, JavaScript first looks for it on the object itself. If it's not found, it searches for it in the prototype chain by following the \_\_proto\_\_ link to the prototype object. This chain continues until the property or method is found or until it reaches the end of the prototype chain (the Object.prototype).  **Question 30. What is the difference between a shallow clone and a deep clone of an object in JavaScript?**  Ans. A shallow clone of an object creates a new object with its own properties, but if the properties are objects themselves, they still reference the same objects in memory as the original object. In contrast, a deep clone creates a completely new object with its own properties and recursively clones nested objects, ensuring that there are no shared references between the original and the cloned object. Deep clones create independent copies of the entire object hierarchy.  **Question 31. How do you create and manipulate DOM elements using JavaScript?**  Ans. You can create and manipulate DOM elements in JavaScript using various methods and properties. Here are some common operations:  -Creating new elements: Use document.createElement('elementName') to create a new HTML element.  -Modifying element properties: Use properties like element.textContent, element.innerHTML, or element.setAttribute() to set content and attributes.  -Appending elements: Use parentElement.appendChild(newElement) to add a new element as a child of another element.  -Removing elements: Use parentElement.removeChild(elementToRemove) to remove an element.  -Modifying CSS styles: Use element.style.property to change the style of an element.  -Adding event listeners: Use element.addEventListener('event', callback) to handle events like clicks."  **Question 32. Describe the purpose of the "debounce" and "throttle" functions in JavaScript.**  Ans. Debounce: Debouncing is a technique used to control the rate at which a function is executed. It ensures that a function is called only after a specified time has passed since the last time it was invoked. It's useful for scenarios like handling user input to avoid excessive function calls.  Throttle: Throttling limits the number of times a function can be called within a specific time interval. It ensures that a function can be executed at most once per interval. Throttling is helpful for optimizing resource-intensive operations like scrolling or resizing event handlers."  **Question 33. What is event delegation, and why is it useful in JavaScript?**  Ans. Event delegation is a technique in JavaScript where a single event handler is attached to a common ancestor element of multiple child elements. Instead of attaching event handlers to each child element individually, the ancestor element listens for events bubbling up from its descendants. This approach is useful for:  -Reducing the number of event handlers, improving performance.  -Dynamically handling events for elements created or removed at runtime.  -Simplifying event management for large or complex DOM structures.  **Question 34. How do you handle asynchronous operations using promises in JavaScript?**  Ans. Promises are used to manage asynchronous operations in JavaScript. You can create a promise using the Promise constructor, and it can be in one of three states: pending, resolved (fulfilled), or rejected. Promises are typically used with the .then() and .catch() methods to handle successful results and errors, respectively. Here's an example:  Javascript  const fetchData = () => {  return new Promise((resolve, reject) => {  // Asynchronous operation  if (/\* operation successful \*/) {  resolve(result);  } else {  reject(error);  }  });  };  fetchData()  .then(data => {  // Handle successful result  })  .catch(error => {  // Handle error  });  **Question 35. Explain the concept of a closure in JavaScript and provide an example.**  Ans. A closure is a JavaScript feature that allows a function to remember and access its outer (enclosing) function's variables and parameters even after the outer function has finished executing. Closures are created whenever a function is defined within another function. Here's an example:  Javascript  function outer() {  const outerVar = 'I am from outer function';  function inner() {  console.log(outerVar); // Accesses outerVar from the outer function  }  return inner;  }  const closureFunc = outer();  closureFunc(); // Outputs: ""I am from outer function""  **Question 36. How can you iterate over the properties of an object in JavaScript?**  Ans. You can iterate over the properties of an object in JavaScript using various methods, such as for...in loops or methods like Object.keys(), Object.values(), and Object.entries(). Here's an example using for...in:  Javascript  const person = {  name: 'Alice',  age: 30,  city: 'New York'  };  for (const key in person) {  if (person.hasOwnProperty(key)) {  console.log(`${key}: ${person[key]}`);  }  }  **Question 37. What is memoization, and how can it be implemented in JavaScript?**  Ans. Memoization is an optimization technique used to cache the results of expensive function calls and return the cached result when the same inputs occur again. It can improve the performance of functions that are called with the same arguments multiple times. You can implement memoization in JavaScript using an object to store the results. Here's a simple example:  Javascript  function memoize(fn) {  const cache = {};  return function (...args) {  const key = JSON.stringify(args);  if (cache[key]) {  return cache[key];  } else {  const result = fn(...args);  cache[key] = result;  return result;  }  };  }  // Example usage  const expensiveFunction = memoize(function (n) {  console.log(`Calculating for ${n}`);  return n \* 2;  });  console.log(expensiveFunction(5)); // Output: Calculating for 5, 10  console.log(expensiveFunction(5)); // Output: 10 (cached)  **Question 38. Describe the purpose of the "arguments" object in JavaScript functions.**  Ans. The "arguments" object is a special object available within all JavaScript functions. It contains an array-like list of the arguments passed to the function, regardless of the number of named parameters defined in the function's signature. It allows you to work with a variable number of arguments. However, it is not a true array but an "array-like" object and lacks array methods. Modern JavaScript encourages using the rest parameters syntax (...args) instead of the "arguments" object for better flexibility and compatibility with array methods.  **Question 39. How can you compare two JavaScript objects for equality?**  Ans. To compare two JavaScript objects for equality (i.e., having the same properties and values), you can't use == or === directly because they compare references, not object contents. You need to implement a custom comparison function that recursively checks the properties and values of the objects. Here's a simple example:  Javascript  function deepEqual(obj1, obj2) {  if (obj1 === obj2) {  return true;  }  if (typeof obj1 !== 'object' || typeof obj2 !== 'object' || obj1 === null || obj2 === null) {  return false;  }  const keys1 = Object.keys(obj1);  const keys2 = Object.keys(obj2);  if (keys1.length !== keys2.length) {  return false;  }  for (const key of keys1) {  if (!keys2.includes(key) || !deepEqual(obj1[key], obj2[key])) {  return false;  }  }  return true;  }  // Example usage  const objA = { x: 1, y: { z: 2 } };  const objB = { x: 1, y: { z: 2 } };  console.log(deepEqual(objA, objB)); // Output: true  **Question 40. What is a JavaScript generator, and how does it work?**  Ans. A JavaScript generator is a special type of function that allows you to pause and resume its execution. It's defined using the function\* syntax and contains one or more yield statements. Generators are used to generate a sequence of values lazily and iteratively. When you call a generator function, it returns an iterator object that can be used to control the execution of the generator. You can pause the generator using yield and resume it using the iterator's .next() method. Here's a simple example:  Javascript  function\* countToThree() {  yield 1;  yield 2;  yield 3;  }  const iterator = countToThree();  console.log(iterator.next().value); // Output: 1  console.log(iterator.next().value); // Output: 2  console.log(iterator.next().value); // Output: 3 |