100 JavaScript Interview Questions and Answers by Sarthaksavvy

Please follow me on instagram at @sarthaksavvy
You can visit my YouTube Channels also:
***English** Web Dev Channel : https://www.youtube.com/@bitfumes*
***Hindi** Web Dev Channel : https://www.youtube.com/@code-jugaad*
Subscribe to my newsletters: https://bitfumes.com/newsletters
(If you are reading this on GitHub then please follow me for more contents like this)

These are curated lists of JavaScript questions with answers that I created from my own experience. It is really hard to create 100 questions from beginning to advanced level so as a human I might make mistakes so If you found any issue on these questions then please write me on email or DM me on instagram.

Question 1: What is the difference between let, const, and var in JavaScript?
Answer:
var has function scope, while let and const have block scope.
let allows reassignment, whereas const does not.

Question 2: Explain the concept of hoisting in JavaScript.
Answer:
Hoisting is a JavaScript behaviour where variable and function declarations are moved to the top of their containing scope during compilation.

Question 3: What is closure in JavaScript and how is it used?
Answer:
A closure is the combination of a function bundled with references to its surrounding state. Closures are used for data encapsulation, creating private variables, and implementing the module pattern.

Question 4: How does prototypal inheritance work in JavaScript?
Answer:
Objects in JavaScript can inherit properties and methods from other objects through their prototype chain.
Each object has a prototype object, and properties/methods are looked up in the prototype chain if not found on the object itself.

Question 5: What is the purpose of “this” keyword in JavaScript?
Answer:
this refers to the object it belongs to in the current context.
In a function, this refers to the global object; in a method, it refers to the owner object; in a constructor, it refers to the new instance.

Question 6: Explain the concept of event delegation in JavaScript.
Answer:
Event delegation is a technique where a single event listener is attached to a common ancestor instead of individual elements.
This is particularly useful when dealing with dynamically generated elements.

Question 7: What is the purpose of the bind method in JavaScript?
Answer:
The bind method is used to create a new function with a specified value and initial arguments. It is often used to set the context of a function.

Question 8: What is the event loop in JavaScript?

Answer: The event loop is a mechanism that allows JavaScript to execute asynchronous tasks by placing them in the callback queue and executing them in a loop.
It ensures non-blocking behaviour in JavaScript.

Question 9: Explain the difference between == and === in JavaScript.
Answer: == is a loose equality operator that performs type coercion, while === is a strict equality operator that compares both value and type.

Question 10: How does the async/await syntax work in JavaScript?
Answer:
async/await is a syntactic sugar for working with asynchronous code.
The async keyword is used to define a function that returns a promise, and await is used to pause the execution of the function until the promise is resolved.

Question 11: What is the difference between let and const when declaring variables?
Answer:
The main difference is that variables declared with const cannot be reassigned after declaration, whereas variables declared with let can.

Question 12: Explain the purpose of the map function in JavaScript.
Answer:
The map function is used to create a new array by applying a provided function to every element in the original array.

Question 13: What is the significance of the use strict mode in JavaScript?
Answer:
Use strict is a pragma that enforces a stricter set of parsing and error handling rules in JavaScript.
It helps catch common coding mistakes and prevents the use of certain error-prone features.

Question 14: How does the localStorage differ from sessionStorage in the context of web storage in JavaScript?
Answer:
Both localStorage and sessionStorage are web storage options with similar APIs.
The main difference is that data stored in localStorage persists even when the browser is closed and reopened, while data in sessionStorage is cleared when the session ends.

Question 15: What is a callback function, and why is it used in JavaScript?
Answer:
A callback function is a function passed as an argument to another function, which is then invoked inside the outer function.
Callbacks are commonly used in asynchronous operations and event handling to execute code once a task is complete or an event occurs.

Question 16: Explain the concept of destructuring in JavaScript.
Answer:
Destructuring is a feature in ES6 that allows you to extract values from arrays or properties from objects and assign them to variables in a concise way.

Question 17: What is the purpose of the Promise object in JavaScript, and how does it differ from callbacks?
Answer:

Promise is an object representing the eventual completion or failure of an asynchronous operation.
Promises provide a more structured way to handle asynchronous code compared to callbacks, making it easier to manage complex asynchronous workflows.

Question 18: How does the fetch API work in JavaScript, and what are its advantages over traditional XMLHttpRequest?
Answer:
The fetch API is used to make network requests, typically to retrieve data from a server.
It provides a more modern and flexible interface for working with HTTP requests compared to XMLHttpRequest.

Question 19: Explain the concept of the event bubbling and event capturing phases in the DOM.
Answer:
Event bubbling is the process where the event starts from the target element and bubbles up to the root of the DOM.
Event capturing is the opposite, where the event starts from the root and trickles down to the target element.
Both phases are part of the event propagation in the DOM.

Question 20: What are arrow functions in JavaScript, and how do they differ from regular functions?
Answer:
Arrow functions are a concise way to write function expressions in JavaScript introduced in ES6.
They have a shorter syntax, do not bind their own this, and cannot be used as constructors.

Question 21: Explain the concept of the JavaScript Event Loop.
Answer:
The Event Loop is a core concept in JavaScript that handles the execution of code by continuously checking the call stack and message queue. It ensures the non-blocking nature of JavaScript, allowing asynchronous tasks to be executed.

Question 22: What are the differences between null and undefined in JavaScript?
Answer:
null is an explicitly assigned value that represents the absence of any object value.
undefined is a variable that has been declared but not assigned any value. It is also the default value of function parameters that are not passed.

Question 23: Explain the concept of IIFE (Immediately Invoked Function Expression) in JavaScript.
Answer:
An IIFE is a function expression that is defined and invoked immediately after its creation.
It is often used to create a private scope for variables, preventing them from polluting the global scope.

Question 24: What is the purpose of the Object.keys() method in JavaScript?
Answer:
The Object.keys() method returns an array of a given object's own enumerable property names.
It is commonly used to iterate over the properties of an object.

Question 25: How does the reduce method work in JavaScript, and what is its significance?
Answer:

The reduce method is used to reduce an array to a single value by executing a provided function for each element.
It is often used for operations like summing up values or transforming arrays.

Question 26: Explain the concept of the Same-Origin Policy in the context of JavaScript and AJAX.
Answer:
The Same-Origin Policy is a security measure that restricts web pages from making requests to a different domain than the one that served the web page.
It helps prevent potential security vulnerabilities, such as Cross-Site Scripting (XSS) attacks.

Question 27: What is a closure in JavaScript, and can you provide an example of its practical use?
Answer:
A closure is a function that has access to variables from its outer (enclosing) scope, even after the outer function has finished executing.
Practical use includes creating private variables, implementing data hiding, and maintaining state in functional programming.

Question 28: Explain the differences between localStorage and sessionStorage in terms of storage duration and scope.
Answer:
localStorage stores data with no expiration time and persists even when the browser is closed and reopened.
sessionStorage stores data for the duration of the page session and is cleared when the page is closed.

Question 29: How does JavaScript handle asynchronous operations, and what are the advantages of using Promises over callbacks?
Answer:
JavaScript uses asynchronous operations to perform tasks without blocking the main thread.
Promises provide a cleaner and more structured way to handle asynchronous code compared to callbacks, making it easier to reason about and maintain.

Question 30: Explain the concept of CORS (Cross-Origin Resource Sharing) in JavaScript.
Answer:
CORS is a security feature implemented by web browsers that controls how web pages in one domain can request and interact with resources hosted on another domain.
It is enforced to prevent potential security vulnerabilities associated with cross-origin requests.

Question 31: What is the purpose of the bind method in JavaScript, and how does it work?
Answer:
The bind method is used to create a new function with a specified this value and optional arguments.
It is often used to explicitly set the context of a function, ensuring that this refers to a specific object.

Question 32: Explain the concept of memoization in JavaScript.
Answer:
Memoization is an optimization technique where the results of expensive function calls are cached, and the cached result is returned when the same inputs occur again.

It is commonly used to improve the performance of recursive or repetitive function calls.

Question 33: What are the differences between == and === in JavaScript, and when would you use one over the other?
Answer:
== is the equality operator that performs type coercion, allowing values of different types to be considered equal.
=== is the strict equality operator that checks both value and type, and does not perform type coercion.
It is generally recommended to use === for more predictable and safer comparisons.

Question 34: Explain the concept of event delegation and why it is useful in JavaScript.
Answer:
Event delegation is a technique where a single event listener is attached to a common ancestor instead of individual elements.
It is useful for handling events on dynamically generated or large sets of elements, improving performance and reducing memory usage.

Question 35: What is the purpose of the typeof operator in JavaScript?
Answer:
The typeof operator is used to determine the data type of a variable or expression.
It returns a string representing the type, such as "number," "string," "boolean," etc.

Question 36: Explain the concept of the ternary operator in JavaScript and provide an example.
Answer:
The ternary operator (? :) is a shorthand for the if-else statement and allows for a concise way of writing conditional expressions.
Example: const result = condition ? trueValue : falseValue;

Question 37: What is the purpose of the async and await keywords in JavaScript?
Answer:
The async keyword is used to define asynchronous functions that return Promises.
The await keyword is used inside an async function to pause execution until the awaited Promise is resolved or rejected.

Question 38: Explain the difference between a shallow copy and a deep copy in JavaScript.
Answer:
A shallow copy creates a new object or array, but does not create new copies of nested objects or arrays. Changes to nested objects are reflected in both the original and the shallow copy.
A deep copy creates a new object or array and recursively copies all nested objects and arrays, ensuring that changes in the original do not affect the copy.

Question 39: What is the purpose of the Array.isArray() method in JavaScript?
Answer:
The Array.isArray() method is used to check if a given value is an array.
It returns true if the value is an array, and false otherwise.

Question 40: Explain the concept of the prototype chain in JavaScript.
Answer:
The prototype chain is a mechanism in JavaScript that allows objects to inherit properties and methods from other objects through their prototype link.

Objects in JavaScript have a prototype object, and if a property or method is not found on the object itself, it is looked up in the prototype chain

Question 41: Explain the concept of the "this" keyword in JavaScript and how its value is determined.
Answer:
The this keyword in JavaScript refers to the object it belongs to in the current execution context. Its value is determined by how a function is called, and it can be influenced by methods like call, apply, or bind.

Question 42: What is the purpose of the forEach method in JavaScript, and how does it differ from the map method?
Answer:
The forEach method is used to iterate over the elements of an array and execute a provided function for each element.
Unlike the map method, forEach does not return a new array, and it is mainly used for side effects.

Question 43: Explain the concept of the "event loop" in the context of asynchronous JavaScript.
Answer:
The event loop is a mechanism in JavaScript that continuously checks the call stack and message queue to manage the execution of code.
It ensures that asynchronous tasks, such as callbacks and Promises, are executed in a non-blocking manner.

Question 44: What is a closure in JavaScript, and how can it be used to create private variables?
Answer:
A closure is a function that retains access to variables from its outer (enclosing) scope, even after the outer function has finished executing.
It can be used to create private variables by encapsulating them within a function, making them inaccessible from outside the function.

Question 45: Explain the concept of "hoisting" in JavaScript.
Answer:
Hoisting is a JavaScript behavior where variable and function declarations are moved to the top of their containing scope during compilation.
Variables declared with var are hoisted and initialized with undefined, while function declarations are fully hoisted.

Question 46: What is the purpose of the Object.create() method in JavaScript?
Answer:
The Object.create() method is used to create a new object with a specified prototype object.
It allows for the creation of objects with a specific prototype, enabling prototype-based inheritance.

Question 47: Explain the difference between the apply() and call() methods in JavaScript.
Answer:
Both apply() and call() are used to invoke a function with a specified this value and arguments.
The main difference is in how arguments are passed: apply() takes an array of arguments, while call() takes individual arguments.

Question 48: What is the purpose of the splice() method in JavaScript, and how does it differ from slice()?
Answer:
The splice() method is used to change the contents of an array by removing or replacing existing elements and/or adding new elements.
slice() is used to create a shallow copy of a portion of an array without modifying the original array.

Question 49: Explain the concept of "currying" in JavaScript.
Answer:
Currying is a technique in functional programming where a function with multiple arguments is transformed into a sequence of functions, each taking a single argument.
It allows for partial function application and the creation of more reusable and composable functions.

Question 50: What are the advantages and disadvantages of using arrow functions in JavaScript?
Answer:
Advantages of arrow functions include a shorter syntax, implicit binding of this, and no binding of their own arguments.
Disadvantages include the lack of a this value of their own and the inability to be used as constructors.

Question 51: How does the spread operator work in JavaScript, and what are its common use cases?
Answer:
The spread operator (`...`) in JavaScript is used to expand elements of an iterable (e.g., arrays) or object properties. It is commonly used for shallow copying arrays, merging arrays, and passing variable arguments to functions.

Question 52: Explain the concept of lexical scoping in JavaScript.
Answer:
Lexical scoping refers to the way the JavaScript interpreter resolves the scope of variables. It is determined by the placement of variables in the source code, and a variable defined in a function is accessible within that function and any nested functions.

Question 53: What is the difference between a deep equality check and a shallow equality check in JavaScript?
Answer:
Shallow equality checks compare the references of objects, while deep equality checks compare the values of nested objects. Shallow checks are faster but may not accurately reflect the content of complex data structures.

Question 54: How can you prevent the default behavior of an event in JavaScript?
Answer:
The `event.preventDefault()` method is used to prevent the default behavior of an event in JavaScript. It is commonly used in event handlers to stop the default action associated with an event, such as form submission or link navigation.

Question 55: What is the purpose of the memoize pattern in JavaScript?
Answer:
Memoization is a pattern in JavaScript used to cache the results of expensive function calls and return the cached result when the same inputs occur again. This helps improve the performance of functions by avoiding redundant computations.

Question 56: Explain the concept of the ternary operator in JavaScript and provide a use case.
Answer:
The ternary operator (`condition ? expr1 : expr2`) is a concise way to write conditional statements. It returns `expr1` if the condition is true and `expr2` otherwise. An example use case is assigning a value based on a condition, like setting a variable to "even" or "odd" based on the parity of another variable.

Question 57: How does the `event.stopPropagation()` method work in JavaScript, and when would you use it?
Answer:
The `event.stopPropagation()` method is used to stop the propagation of an event through the DOM hierarchy. It prevents the event from reaching parent or ancestor elements. This is useful when you want to handle an event only at a specific level in the DOM without triggering parent element handlers.

Question 58: What is the significance of the `isNaN()` function in JavaScript?
Answer:
The `isNaN()` function in JavaScript is used to determine whether a value is NaN (Not a Number). It returns `true` if the value is NaN, and `false` if the value can be converted to a number.

Question 59: How does the JavaScript `typeof` operator behave with different data types?
Answer:
The `typeof` operator in JavaScript returns a string indicating the type of the operand. It can be used to check the data type of a variable. Common return values include "number," "string," "boolean," "object," "function," and "undefined."

Question 60: Explain the concept of lexical scoping in JavaScript and provide an example.
Answer:
Lexical scoping means that the scope of a variable is determined by its position within the source code. An example is:

```javascript
function outerFunction() {
  const outerVar = "I am from outer function";

  function innerFunction() {
    console.log(outerVar); // Accessing outerVar from outer function
  }

  innerFunction();
}

outerFunction();
```

In this example, `innerFunction` has access to `outerVar` because of lexical scoping.

Question 61: What is the purpose of the rest parameter in JavaScript functions?
Answer:

The rest parameter (`...args`) allows a function to accept an indefinite number of arguments as an array. It is useful when the number of arguments is unknown or variable, enabling the function to work with a flexible number of parameters.

Question 62: How does the JavaScript call stack work, and what is its role in function execution?
Answer:
The call stack is a mechanism in JavaScript that keeps track of function calls. When a function is invoked, a new frame is added to the call stack. When the function completes, its frame is removed. This stack ensures the order of function execution, preventing the system from running out of memory due to infinite recursion.

Question 63: Explain the concept of the Observer pattern in JavaScript.
Answer:
The Observer pattern is a design pattern where an object, known as the subject, maintains a list of its dependents, called observers, that are notified of any state changes. It is commonly used for implementing distributed event handling systems.

Question 64: What is the purpose of the `defer` attribute in a script tag, and how does it differ from `async`?
Answer:
The `defer` attribute in a script tag is used to defer the execution of the script until the HTML parsing is complete. Unlike `async`, it ensures that scripts are executed in order and after the HTML document is fully parsed.

Question 65: How can you handle errors in asynchronous JavaScript code?
Answer:
In asynchronous JavaScript, errors can be handled using `try`, `catch`, and `finally` blocks. Additionally, the use of `Promise` rejection and `async/await` makes error handling more structured.

Question 66: What are the differences between `document.querySelector` and `document.getElementById` in JavaScript?
Answer:
`document.querySelector` selects the first element that matches a specified CSS selector, while `document.getElementById` specifically selects an element by its unique ID. The former is more versatile for selecting elements based on various conditions.

Question 67: Explain the concept of memoization in the context of optimizing recursive functions in JavaScript.
Answer:
Memoization is a technique where the results of expensive function calls are cached and reused when the same inputs occur again. In the context of optimizing recursive functions, memoization helps avoid redundant computations by storing and retrieving previously calculated results.

Question 68: What is the purpose of the `Symbol` data type in JavaScript, and how is it used?
Answer:
The `Symbol` data type in JavaScript represents a unique identifier. Symbols are often used as property keys to avoid naming conflicts in objects. They are guaranteed to be unique and are not enumerable in `for...in` loops.

Question 69: How does the `instanceof` operator work in JavaScript, and when is it useful?

Answer:
The `instanceof` operator checks whether an object is an instance of a particular class or constructor function. It returns `true` if the object is an instance of the specified type; otherwise, it returns `false`. It is useful for checking the type of objects, especially in object-oriented programming.

Question 70: Explain the concept of currying in JavaScript and provide an example.
Answer:
Currying is a technique in functional programming where a function with multiple arguments is transformed into a series of functions, each taking a single argument. Here's an example:
```javascript
function curry(fn) {
  return
```
Question 71: What is the purpose of the `window` object in the browser environment, and how is it different from the `document` object?
Answer:
The `window` object in the browser environment represents the global window containing the DOM and other properties. It serves as the global scope for JavaScript in a web page. The `document` object, on the other hand, represents the HTML document itself, providing methods and properties to manipulate the document's content.

Question 72: How does the JavaScript `call()` method differ from `apply()` and `bind()`?
Answer:
All three methods (`call()`, `apply()`, and `bind()`) are used to set the value of `this` in a function. The key difference is in how arguments are passed. `call()` and `apply()` immediately invoke the function, with `apply()` accepting arguments as an array. `bind()` returns a new function with `this` set, but doesn't immediately invoke the function.

Question 73: Explain the concept of the prototype-based inheritance in JavaScript.
Answer:
Prototype-based inheritance in JavaScript involves creating objects that act as prototypes for other objects. Objects inherit properties and methods from their prototype, forming a chain. When a property or method is accessed on an object, JavaScript looks up the prototype chain until it finds the property or determines it doesn't exist.

Question 74: What is the purpose of the `requestAnimationFrame` method in JavaScript?
Answer:
The `requestAnimationFrame` method is used for smooth animations in the browser. It schedules a function to be executed before the next repaint, aligning with the browser's rendering cycle. This helps create animations that are synchronized with the display refresh rate, reducing flickering and improving performance.

Question 75: How does the JavaScript `switch` statement work, and what are its limitations?
Answer:
The `switch` statement evaluates an expression against multiple possible case values. When a match is found, the associated block of code is executed. It is often used as an alternative to long sequences of `if-else` statements. Limitations include the inability to use complex conditions in cases and the absence of "fall-through" prevention without the `break` statement.

Question 76: What is the significance of the `fetch` API's `Response.ok` property in JavaScript?
Answer:

The `Response.ok` property is a boolean indicating whether a response was successful (status code in the range 200-299) or not. Checking `Response.ok` is a convenient way to determine if a network request was successful without manually inspecting the status code.

Question 77: Explain the concept of memoization in JavaScript and how it can improve the performance of functions.
Answer:
Memoization involves caching the results of expensive function calls. When the same inputs occur again, the cached result is returned instead of recalculating. This can significantly improve the performance of functions, especially recursive ones, by avoiding redundant computations.

Question 78: How does the JavaScript `forEach` method work, and what are its advantages over traditional `for` loops?
Answer:
The `forEach` method is used to iterate over elements of an array. It takes a callback function as an argument and executes that function for each array element. Advantages over traditional `for` loops include cleaner syntax, implicit iteration, and less room for off-by-one errors.

Question 79: What is the purpose of the `DataView` object in JavaScript, and how is it used?
Answer:
The `DataView` object in JavaScript provides a way to read and write raw binary data in a buffer. It allows for more fine-grained control over the interpretation of data, enabling manipulation at the byte level. This is particularly useful when dealing with binary data structures or network protocols.

Question 80: How can you create and handle custom events in JavaScript?
Answer:
Custom events in JavaScript can be created using the `CustomEvent` constructor. They can be dispatched on elements using `dispatchEvent`. Event listeners are then used to handle these custom events. This allows for the implementation of a pub-sub (publish-subscribe) pattern.

Question 81: Explain the differences between the terms "undefined" and "null" in JavaScript.
Answer:
In JavaScript, `undefined` is a primitive value that represents the absence of a value or the absence of a defined property. `null` is also a primitive value, but it is explicitly assigned to indicate the absence of any object value. While similar in some contexts, they are distinct values in the language.

Question 82: What is the purpose of the `isNaN()` function in JavaScript, and how does it handle different data types?
Answer:
The `isNaN()` function is used to determine whether a value is NaN (Not a Number). It returns `true` if the value is NaN, and `false` if the value can be converted to a number. `isNaN()` handles different data types by attempting to convert the argument to a number before performing the check.

Question 83: How does the JavaScript `Array.from()` method work, and what is its use case?
Answer:
The `Array.from()` method creates a new array from an iterable or array-like object. It takes an optional mapping function and an optional `this` argument. This method is useful for converting iterable structures, such as strings or NodeList objects, into arrays.

Question 84: Explain the concept of the prototype chain in JavaScript and how it facilitates inheritance.
Answer:
The prototype chain is a mechanism in JavaScript that allows objects to inherit properties and methods from other objects. Each object has a prototype object, and when a property or method is accessed on an object, JavaScript looks up the prototype chain until it finds the property or determines it doesn't exist. This facilitates prototype-based inheritance.

Question 85: What is the purpose of the `requestAnimationFrame` method in JavaScript, and how does it contribute to smooth animations?
Answer:
The `requestAnimationFrame` method is used for smooth animations by synchronizing with the browser's rendering cycle. It ensures that animations are executed just before the next repaint, providing a smoother visual experience. This method is preferred over traditional timers for animations to optimize performance.

Question 86: How does the JavaScript `for...of` loop differ from the `for...in` loop, and when would you use each?
Answer:
The `for...of` loop iterates over iterable objects (arrays, strings, etc.) and gives direct access to the values, while the `for...in` loop iterates over the properties of an object and provides access to the property names. Use `for...of` when you need values, and `for...in` when you need property names.

Question 87: Explain the concept of the Observer pattern in JavaScript and provide a real-world example.
Answer:
The Observer pattern is a design pattern where an object, known as the subject, maintains a list of its dependents, called observers, that are notified of any state changes. A real-world example is implementing event listeners in a web application. The listeners (observers) are notified when a specific event (state change) occurs on an element (subject).

Question 88: What is the purpose of the `WeakMap` object in JavaScript, and how does it differ from a regular `Map`?
Answer:
The `WeakMap` object in JavaScript is a collection of key/value pairs where keys must be objects. Unlike a regular `Map`, a `WeakMap` allows garbage collection of keys that are not referenced elsewhere in the code, making it useful for scenarios where the keys are temporary or short-lived.

Question 89: How does the JavaScript `Array.splice()` method work, and what are its practical applications?
Answer:
The `Array.splice()` method in JavaScript changes the contents of an array by removing or replacing existing elements and/or adding new elements. It can be used for array manipulation, such as deleting elements, inserting elements at a specific position, or replacing elements with new values.

Question 90: Explain the concept of the Proxy object in JavaScript and how it can be used to intercept and customize object operations.
Answer:

The `Proxy` object in JavaScript allows you to create a custom handler to intercept and customize fundamental operations on objects, such as property access, assignment, and method invocation. This provides a powerful way to control and modify object behavior.

Question 91: What is the purpose of the JavaScript `Array.filter()` method, and how does it work?
Answer:
The `Array.filter()` method creates a new array with elements that pass a provided function's test. It filters out elements for which the callback function returns `false`. This method is commonly used for selecting elements based on specific criteria from an array.

Question 92: How does the JavaScript `this` keyword behave in arrow functions compared to regular functions?
Answer:
In arrow functions, the value of `this` is lexically scoped, meaning it is inherited from the enclosing scope. In regular functions, `this` is dynamically scoped, determined by how the function is called. Arrow functions are particularly useful when you want to maintain the surrounding context's `this` value.

Question 93: Explain the concept of the "time complexity" of an algorithm in the context of JavaScript.
Answer:
Time complexity measures the amount of time an algorithm takes to complete as a function of the input size. It describes how the execution time grows with the size of the input. Common time complexities include constant time (`O(1)`), linear time (`O(n)`), logarithmic time (`O(log n)`), and quadratic time (`O(n^2)`).

Question 94: What is the purpose of the `Intl` object in JavaScript, and how can it be used for internationalization?
Answer:
The `Intl` object in JavaScript provides internationalization and localization support. It offers functionality for formatting numbers, dates, and strings according to specified locales. This helps developers create applications that are culturally aware and user-friendly for different regions and languages.

Question 95: How does the JavaScript `Array.every()` method work, and when would you use it?
Answer:
The `Array.every()` method tests whether all elements in an array pass a provided function's test. It returns `true` if the callback function returns `true` for every element, and `false` otherwise. It is useful for checking if all elements in an array satisfy a specific condition.

Question 96: What is the purpose of the `Object.freeze()` method in JavaScript, and how does it affect object properties?
Answer:
The `Object.freeze()` method in JavaScript is used to freeze an object, preventing any changes to its properties or the addition of new properties. Once an object is frozen, its properties become read-only and cannot be modified. This helps create immutable objects with fixed state.

Question 97: Explain the concept of the mediator pattern in JavaScript and provide an example.
Answer:
The mediator pattern is a behavioral design pattern where a mediator object centralizes communication between components, reducing direct connections between them. An example

is a chat application where individual users (components) communicate through a central chat room (mediator) instead of directly with each other.

Question 98: How does the JavaScript `Array.reduce()` method differ from the `Array.map()` method, and when would you choose one over the other?
Answer:
The `Array.reduce()` method reduces an array to a single value by applying a function to each element and accumulating the result. `Array.map()` creates a new array by applying a function to each element. Choose `reduce()` when you need to derive a single value, and `map()` when you want to transform each element into a new value.

Question 99: What is the purpose of the `document.createDocumentFragment()` method in JavaScript?
Answer:
The `document.createDocumentFragment()` method creates an empty `DocumentFragment` object. A `DocumentFragment` is a lightweight container that can hold multiple nodes. It is useful for efficiently appending or manipulating multiple elements before adding them to the DOM, reducing the number of reflows.

Question 100: How does the JavaScript `Promise.all()` method work, and what is its use case?
Answer:
The `Promise.all()` method takes an array of promises and returns a single promise that resolves when all the input promises have resolved or rejects when any of them rejects. It is commonly used when multiple asynchronous operations need to be performed concurrently, and you want to wait for all of them to complete.

Thanks for reaching out at this level.
If you have read every question then you are really an amazing learner. Keep learning.

Please follow me on instagram at @sarthaksavvy
You can visit my YouTube Channels also:
***English** Web Dev Channel : https://www.youtube.com/@bitfumes*
***Hindi** Web Dev Channel : https://www.youtube.com/@code-jugaad*
Subscribe to my newsletters: https://bitfumes.com/newsletters