Javascript Interview Questions and Answers

**1. What is JavaScript?**

Answer: JavaScript is a programming language primarily used for web development. It allows developers to add interactivity and dynamic features to web pages. JavaScript is executed in web browsers and can manipulate HTML, CSS, and the Document Object Model (DOM). It is also used for server-side development and mobile app development.

**2. What are the data types in JavaScript?**

Answer: There are several types of data type in Javascript. They are “Number” which

Represents numeric values.“String” which Represents sequences of characters. “Boolean” Represents logical values of true or false. “Undefined” Represents variables that are declared but not assigned a value yet. “Null” Represents the intentional absence of any object value. “Object” Represents a collection of key-value pairs. “Array” which Represents an ordered list of values.“Symbol” Represents unique identifiers. “BigInt” which Represents arbitrary precision integers.

**3. What is the difference between null and undefined?**

Answer: In JavaScript, the difference between null and undefined is that null is an intentional value indicating the absence of an object, while undefined is assigned automatically to variables that have been declared but not assigned a value. For example, if a variable is declared but not assigned, its value will be undefined. On the other hand, null is a value that can be explicitly assigned to a variable to indicate that it has no object value.

**4. What is hoisting in JavaScript?**

Answer: Hoisting in JavaScript is a behavior where variable and function declarations are moved to the top of their containing scope during the compilation phase. This allows variables and functions to be used before they are declared in the code. It's important to note that only the declarations are hoisted, not the variable assignments or function expressions.

**5. What are closures in JavaScript?**

Answer: In JavaScript, closures are a powerful and fundamental concept. A closure is created when a function is defined within another function and has access to its outer function's variables, even after the outer function has finished executing. This means that the inner function "remembers" the environment in which it was created, including the variables and parameters of its outer function. Closures are formed by the combination of a function and its lexical environment, which consists of the variables, functions, and scopes available to it at the time of its creation. The inner function retains access to these variables and can continue to use them even after the outer function has been completed.

**6. What is the difference between == and === in JavaScript?**

Answer: The main difference is that “==” performs type coercion, allowing for loose equality comparisons where values with different types can be considered equal after type conversion. On the other hand, “===” does not perform type coercion and requires both the value and the type to be the same for the comparison to evaluate as true. It is generally recommended to use “===” for precise and predictable comparisons, as it avoids unexpected behavior caused by implicit type conversions.

**7. What are the different ways to define a function in JavaScript?**

Answer: In JavaScript, there are multiple ways to define a function. One way is using a function declaration, which involves using the function keyword followed by the function name, parameters in parentheses, and the function body enclosed in curly braces. Another approach is using function expressions, where we assign a function to a variable or a constant. This can be done by using the function keyword without a name or with a name. Lastly, there are arrow functions, introduced in ES6, which provide a shorter syntax for defining functions using the => arrow symbol. "Remember, in a face-to-face interview, the focus is often on your understanding and explanation of concepts rather than demonstrating code directly.

**8. What is the event loop in JavaScript?**

Answer: In Javascript event loop is responsible for handling asynchronous operations and ensuring the smooth execution of code in the event-driven environment of the browser or Node.js runtime. The event loop works by continuously monitoring the execution stack and the task queue. The event loop ensures that JavaScript can handle non-blocking I/O operations and asynchronous tasks without stopping the main thread. It allows for the handling of events, timers, and callbacks in an efficient and responsive manner. Once the asynchronous operation completes, the associated callback or event is placed in the task queue.

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

Answer: In JavaScript, this keyword means the object that is currently executing the code, or the object that a function is invoked. It allows access to properties and methods of the current object within the function. “this” keyword provides context and enables dynamic binding of functions to different objects. It allows functions to be reused with different objects, promoting code reusability and flexibility. When “this” is used in the global scope it refers to the global object. When a function is invoked as a method of an object, “this” refers to the object itself. When a function is invoked as a standalone function, “this” refers to the global object. When a function is used as a constructor with the “new” keyword, “this” refers to the newly created object instance.

**10. What are the different ways to handle asynchronous operations in JavaScript?**

Answer: There are multiple ways to handle asynchronous operations in JavaScript to ensure smooth execution and avoid blocking the main thread. Callback, promises, async/await, and event-driven programming are the commonly used asynchronous operations. Callbacks are a traditional approach for handling asynchronous operations. Functions accept a callback function as an argument, which is executed once the operation completes. Promises provide a more structured and readable way to handle asynchronous operations. Promises represent the eventual completion (or failure) of an asynchronous operation and allow the chaining of operations using methods like “then()” and “catch()”. The async keyword is used to define an asynchronous function, and the await keyword is used to pause the execution and wait for a promise to resolve before proceeding.

**11. How does prototypal inheritance work in JavaScript?**

Answer: In JavaScript, prototypal inheritance is the mechanism by which objects inherit properties and methods from other objects. Every object in JavaScript has a prototype, which serves as a blueprint for that object. When a property or method is accessed on an object, and it doesn't exist on that object itself, JavaScript looks up the prototype chain to find the property or method on higher-level objects. Each object in JavaScript has a prototype property that references another object. This object is the prototype of the current object and provides the properties and methods that the object inherits. constructor functions can be used to create objects with a shared prototype. The “new” keyword creates a new object, sets its prototype to the constructor function's “prototype” property, and executes the constructor function to initialize the object.

**12. What is the difference between let, const, and var in JavaScript?**

Answer: Variables declared with var have function scope or global scope, meaning they are accessible throughout the entire function or globally. “let” allows you to declare

block-scoped variables, which means the variable is only accessible within the block of

code where it is defined. “Let” variables can be reassigned but not redeclared within the same block scope. “const” is used to declare variables with block scope, just like “let”. “const” is constant, which means their values cannot be reassigned once they are assigned. it is recommended to use “const” for values that shouldn't be changed and “let” for values that need to be reassigned. Reserve the use of “var” for situations where you explicitly need its function scope behavior or when working with older codebases that rely on it.

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

Answer: To prevent the default behavior of an event in JavaScript, we use the “preventDefault()” method available on the event object. This method allows you to stop the default action associated with the event from occurring. As a React/MERN stack developer, we use this method mostly in the “form”. When a user clicks the submit button, the default behavior is to navigate to the URL specified in the “to” attribute of the link. However, by using “event.preventDefault()” within the event handler for the click event, you can prevent this default navigation behavior. This can be useful when you want to implement custom functionality or perform additional actions before allowing the default behavior to happen.

**14. What are the different ways to manipulate the DOM in JavaScript?**

Answer: There are several ways to manipulate the DOM in JavaScript. The followings are “getElementById”, which allows selecting an element by its unique “id” attribute. Then “querySelector and querySelectorAll”. “querySelector” returns the first matching element, while “querySelectorAll” returns a collection of all matching elements. “innerHTML”, allows to get or set the HTML content of an element. It provides a convenient way to modify the entire HTML content within an element, including its child elements. “createElement”, allows to dynamically create a new element. Once created, you can modify its properties, attributes, and content before adding it to the DOM. “appendChild”, appends a child element to another element. It is commonly used to add newly created elements or existing elements to the DOM tree as children of a specific parent element.

**15. Explain the concept of debouncing and throttling in JavaScript.**

Answer: Debouncing is a technique used to ensure that a function is executed only after a certain period of inactivity following an event. It is useful in scenarios where you want to delay the execution of a function until the user has stopped triggering an event. For example, when handling search input, debouncing can be used to delay making API requests until the user has finished typing. Throttling, on the other hand, limits the rate at which a function is executed. It ensures that the function is called at most once within a specified time interval. Throttling is useful in scenarios where you want to limit the frequency of executing a function, especially for events that can occur rapidly. For example, throttling can be applied to scroll events to ensure that an action is performed periodically while scrolling, without overwhelming the browser with continuous function calls.

ES6 Interview Questions and Answers

**1. What is ES6 (ECMAScript 2015)?**

Answer: ES6, also known as ECMAScript 2015, is a significant update to the JavaScript language. It introduced several new features and enhancements to improve the developer experience and make JavaScript code more expressive and powerful. One key features of ES6 are Template literals for improved string manipulation, Destructuring assignments for extracting values from arrays or objects, Spread syntax for expanding elements from an iterable, Classes for object-oriented programming, Modules for organizing and reusing code across multiple files, Promises for better handling of asynchronous operations.

**2. What are some key features introduced in ES6?**

Answer: ES6, ECMAScript 2015 introduced several key features to enhance JavaScript. These include block-scoped variables “let” and “const”, arrow functions, template literals, destructuring assignments, spread syntax, classes, modules, promises, enhanced object literals, and iterators/generators. These features improved the expressiveness, readability, and functionality of JavaScript, making it more efficient and enjoyable to work with.

**3. What are block-scoped variables in ES6 and how are they declared?**

Answer: With block-scoped variables, developers have more precise control over the scope of variables within specific blocks of code. The let keyword allows for the declaration of variables that are limited to the block in which they are defined. This means they are accessible only within that block and any nested blocks. They have block-level visibility and are not accessible outside of the block. Additionally, let

variables can be reassigned to different values within the block if necessary On the other hand, the const keyword also declares block-scoped variables, but with an added restriction. Once a value is assigned to a const variable, it cannot be reassigned. const variables typically used for declaring constants or values that should remain unchanged throughout their lifecycle. They are still block-scoped like let variables and are not accessible outside of the block in which they are defined.

**4. How does arrow function syntax differ from regular function syntax in ES6?**

Answer: Arrow function syntax in ES6 provides a more concise and simplified way of writing functions compared to regular function syntax. It uses a shorter syntax, omitting the function keyword and utilizing a fat arrow (=>) between the parameter list and the function body. Arrow functions do not have their own “this” value and instead, inherit the “this” value from the surrounding context. They also have implicit returns, allowing for concise one-line function bodies. However, it's important to note that arrow functions may not be suitable for all scenarios, particularly when dynamic scoping or this binding is required.

**5. What are template literals in ES6 and how are they used?**

Answer: Template literals in ES6 offer an enhanced approach for working with strings in

JavaScript. They use backticks (`) as delimiters and allow for the embedding of

expressions within ${} placeholders. Template literals provide a concise way to

interpolate variables into strings without requiring concatenation. Additionally, they

support multiline strings without the need for escape characters. The usage of template

literals results in more readable and flexible string manipulation in JavaScript.

**6. What is the destructuring assignment in ES6 and how does it work?**

Answer: Destructuring assignment in ES6 refers to the capability of unpacking values from arrays or objects and assigning them to variables. It provides a concise syntax for extracting specific values from complex data structures. Array destructuring uses square brackets [], while object destructuring uses curly braces {}. By utilizing this feature, you can assign multiple variables at once by matching the structure of the array or object. This simplifies the process of accessing and working with elements or properties within JavaScript.

**7. Explain the concept of default parameter values in ES6 functions.**

Answer: Default parameter values in ES6 functions provide a concise and efficient way to set fallback values for function parameters. By assigning a default value to a parameter using the assignment operator (=), you ensure that the function can still execute even if certain arguments are not provided or are explicitly undefined. This helps avoid errors and simplifies the handling of missing values within the function. With default parameter values, you can define the behavior of a function when specific arguments are not explicitly passed, making your code more robust and flexible.

**8. What is the purpose of the spread operator in ES6 and how is it used?**

Answer: The spread operator in ES6 serves multiple purposes. It enables the creation of copies or clones of arrays and objects, ensuring that modifications to the new instance do not affect the original. Additionally, it allows for the merging of arrays into a single array, simplifying tasks like concatenation or combining different arrays. Moreover, the spread operator facilitates the passing of array elements as individual arguments to functions, providing flexibility and convenience when working with varying argument lists. With its versatility and ability to handle iterable objects, the spread operator enhances code readability and conciseness.

**9. How are classes defined in ES6 and what are their benefits?**

Answer: In ES6, classes are defined using the “class” keyword, providing a structured syntax for creating objects based on a blueprint. The benefits of using classes include code organization, simplicity, and readability. Classes support inheritance and code reuse, promote encapsulation, and offer a more intuitive approach to working with prototypes. Overall, classes in ES6 enhance object-oriented programming in JavaScript by providing a clearer syntax and facilitating code modularity.

**10. What are modules in ES6 and how do they differ from previous module patterns?**

Answer: ES6 modules are a standardized way of organizing and sharing JavaScript code. They offer built-in support in JavaScript, with explicit export and import statements to define dependencies. ES6 modules are file-based, promoting code organization, and are executed in strict mode by default. They also allow for static analysis, enabling better tooling and optimization. Overall, ES6 modules provide a more structured and standardized approach to managing dependencies and organizing code compared to previous module patterns.

**11. What is the difference between “let”, “const”, and “var” in ES6?**

Answer: ES6 modules are a standardized way of organizing and sharing JavaScript code. They differ from previous module patterns in that they have native support in JavaScript, use explicit export and import statements for managing dependencies, are file-based for improved code organization, are executed in strict mode by default, and allow for static analysis. ES6 modules provide a more structured and standardized approach to managing dependencies and organizing code compared to previous module patterns.

**12. How can ES6 iterators and generators be used to control iteration?**

Answer: Iterators are objects that implement the Iterator protocol. They have a next() method which, when called, returns an object with value and done properties. By manually calling next(), you can control the iteration process and retrieve values one at a time. Generators, defined using the function syntax, are functions that can be paused and resumed during execution using the yield keyword. Generators provide a flexible approach to iteration, allowing you to yield values and control the flow of iteration. Both iterators and generators offer ways to customize and control the iteration process in JavaScript, providing flexibility and fine-grained control over the iteration flow.

**13. What are modules in ES6 and how do they differ from previous module patterns?**

Answer: ES6 modules have native support in JavaScript, eliminating the need for workarounds or external libraries. They use explicit export and import statements to define dependencies and control what is exposed from a module. This provides a standardized and clearer syntax for managing dependencies. Furthermore, ES6 modules are file-based, with each module typically residing in a separate file. This promotes code organization and separation of concerns. ES6 modules are executed in strict mode by default, ensuring better code quality and preventing common JavaScript pitfalls. Lastly, ES6 modules allow for static analysis, enabling better tooling support and optimization during the build process.

**14. Explain the concept of modules in ES6 and how they are used.**

Answer: modules in ES6 provide a standardized way of organizing and sharing JavaScript code. They differ from previous module patterns by offering native support in JavaScript, explicit export and import statements for defining dependencies, file-based organization, strict mode execution by default, and support for static analysis. ES6 modules promote code organization, improve maintainability, and provide a clearer syntax for managing dependencies in JavaScript applications.

**15. What are some new data structures introduced in ES6?**

Answer: ES6 introduced new data structures to JavaScript, including Set, Map, WeakSet, and WeakMap. These data structures enhance the capabilities of JavaScript by providing efficient ways to manage unique values, key-value pairs, and weak references. They offer improved solutions for data manipulation and provide more flexibility and efficiency in handling data within JavaScript applications.