**Conceptual Questions**

1. **Why are promises used in JavaScript? Explain the advantages of using promises over traditional callback functions.**

* **Promises are a powerful tool for handling asynchronous operations in JavaScript. They make asynchronous code more manageable and easier to read, and they can help to prevent callback hell. The advantages of using promises over traditional callback functions:**

1. **Improved readability: Promises can make asynchronous code more readable by making it clear which code is responsible for starting the asynchronous operation and which code is responsible for handling the result.**
2. **Reduced callback hell: Promises can help to prevent callback hell by allowing you to chain together multiple asynchronous operations without having to nest callbacks.**
3. **Error handling: Promises provide a standard way of handling errors in asynchronous operations**
4. **What is a closure in JavaScript? Provide an example.**

* **A closure is a function that has access to its own scope, the scope of the outer function, and the global scope. Closures are created every time a function is created. An example:**

**function createCounter() {**

**let count = 0;**

**return function() {**

**count++;**

**return count;**

**};**

**}**

**const counter = createCounter();**

**console.log(counter());**

**console.log(counter());**

1. **What is a callback function and why is it used in JavaScript?**

* **A callback function is a function passed into another function as an argument to be executed later. Callbacks are used in JavaScript for asynchronous operations like handling events, making API calls, and processing data.**

**4. What are async/await in JavaScript and how do they improve asynchronous**

**programming?**

* **async/await are syntactic sugar built on top of promises, making asynchronous code easier to write and read. async functions return a promise, and await pauses the execution of the function until the promise resolves or rejects. This allows writing asynchronous code in a synchronous manner.**

**async function fetchData() {**

**try {**

**const data = await someAsyncOperation();**

**console.log(data);**

**} catch (error) {**

**console.error(error);**

**}**

**}**

1. **Write the difference between ES6 and JS.**

* **ES6, also known as ECMAScript 2015, is the sixth edition of the ECMAScript standard, which is the specification for JavaScript. It introduced many new features and syntax enhancements that make JavaScript code more concise, readable, and powerful.**
* **Arrow Functions: ES6 introduced arrow functions, which are a more concise syntax for writing function expressions. They have a more compact syntax, lexical scoping of this, and do not bind their own this value. This makes arrow functions great for writing inline functions or for callbacks.**
* **Class Syntax: ES6 introduced a new syntax for creating classes in JavaScript, also known as class syntax. With class syntax, you can define classes more declaratively, making it easier to create objects with similar properties and methods. This syntax provides a more familiar and structured way of working with objects and inheritance.**
* **Let and Const Declarations: ES6 introduced two new ways to declare variables: let and const. let allows you to declare block-scoped variables, while const allows you to declare block-scoped variables that can't be reassigned. This helps to solve common issues and quirks associated with variable scoping in JavaScript.**
* **Modules: ES6 introduced native support for modules, allowing you to organize and separate your code into reusable and independent modules. Modules provide a way to encapsulate code and prevent pollution of the global namespace. This promotes better code organization and improves code reuse and maintainability.**
* **Template Literals: ES6 introduced template literals, tare a new way to define strings in JavaScript. Template literals allow you to embed expressions inside string literals using ${} syntax. This makes string interpolation and multiline strings much easier and more concise.**
* **Spread and Rest Operators: ES6 introduced the spread and rest operators, which allow you to work with arrays and objects in a more flexible way. The spread operator ... allows you to expand an iterable object into individual elements, making it useful for creating copies of arrays or concatenating multiple arrays. The rest operator ... allows you to collect multiple arguments into a single array, making it useful for functions with a variable number of arguments.**

**In summary, ES6 introduced various new features and syntax enhancements to JavaScript, such as arrow functions, class syntax, let and const declarations, modules, template literals, and spread and rest operators. These additions make JavaScript code more concise, readable, and powerful.**

1. **What are some of the major features introduced in ES6?**
   * **Some of the major features introduced in ES6:**

**1. Let and Const Keywords: `let` for block-scoped variable declaration. `const` for block-scoped, read-only variable declaration.**

**2. Arrow Functions: Shorter syntax for writing functions and lexical `this` binding.**

**3. Template Literals: Multi-line strings and string interpolation.**

**4. Destructuring Assignment: Extracting values from arrays or objects into distinct variables.**

**5. Default Parameter: Setting default values for function parameters.**

**6. Classes: Syntactical sugar over the prototype-based inheritance model.**

**7. Modules: Native support for module import/export.**

**8. Promises: Improved handling of asynchronous operations.**

**9. Rest and Spread Operators: Rest: Collects all remaining elements into an array and Spread: Expands an array into individual elements.**

**10. Enhanced Object Literals: Shorthand property names, method definitions, and computed property names.**

**11. Iterators and For...Of Loop: Custom iteration behavior with iterators and Simplified iteration over iterable objects using `for...of`.**

**12. Map and Set: New data structures for storing collections of unique values (Set) or key-value pairs (Map).**

**13. Symbols: A new primitive type for creating unique identifiers.**

**14. Block-Scoped Functions: Functions declared within a block are scoped to that block.**