Assignment Questions 4

Q.1 Explain Hoisting in JavaScript

Ans: Hoisting in JavaScript is a behavior where variable and function declarations are moved to the top of their containing scope during compilation, allowing them to be accessed and used before their actual declaration in the code. Variable declarations are hoisted but initialized with `undefined`, while function declarations are fully hoisted, including their body. It is recommended to declare variables and functions at the top of their scope for better code clarity.

Q.2 Explain Temporal Dead Zone?

Ans: The Temporal Dead Zone (TDZ) is a period in JavaScript where variables declared with `let` and `const` exist but cannot be accessed or assigned a value until they are declared. Accessing variables within the TDZ results in a `ReferenceError`. The TDZ serves as a safety measure to catch potential errors caused by using variables before they are properly initialized.

Q.3 Difference between var & let?

Ans: 1. Scope: Variables declared with `var` are function-scoped, meaning they are accessible within the function they are declared in, or globally if declared outside any function. On the other hand, variables declared with `let` are block-scoped, meaning they are limited to the block they are declared in (e.g., within an `if` statement or a loop).

2. Hoisting: Variables declared with `var` are hoisted to the top of their scope during the compilation phase, which means you can access them before their actual declaration in the code. However, their value will be `undefined` until the assignment is encountered. `let` variables are also hoisted, but they reside in the Temporal Dead Zone (TDZ) until their actual declaration is reached, and accessing them before that results in a `ReferenceError`.

Q.4 What are the major features introduced in ECMAScript 6?

Ans:1. Arrow Functions: A concise syntax for defining functions.

2. Block-Scoped Variables: `let` and `const` keywords for block-level variable scope.

3. Classes: Syntax for creating object-oriented classes.

4. Modules: Improved code organization and reusability with import and export statements.

5. Template Literals: Enhanced string literals with variable interpolation.

6. Destructuring Assignment: Extracting values from arrays or objects into variables.

7. Enhanced Object Literals: Shorthand syntax and additional features for defining objects.

8. Promises: Built-in mechanism for handling asynchronous operations.

9. Default Parameters: Assigning default values to function parameters.

10. Spread and Rest Operators: Expanding arrays/objects or gathering function arguments.

Q.5 What is the difference between `let` and `const` in ES6?

Ans: 1. Mutable vs. Immutable: Variables declared with `let` are mutable, meaning their values can be reassigned or modified. On the other hand, variables declared with `const` are immutable, and their values cannot be changed once assigned. However, it's important to note that for objects and arrays, `const` prevents reassignment of the variable itself, but the properties or elements of the object or array can still be modified.

2. Block Scope: Both `let` and `const` have block scope, meaning they are only accessible within the block they are declared in (a block is typically defined by curly braces `{}`). This differs from `var`, which has function scope.

3. Hoisting: Variables declared with `let` and `const` are hoisted to the top of their block scope, but unlike `var`, they are not initialized to a default value (`undefined`). Instead, they enter the Temporal Dead Zone (TDZ) and cannot be accessed before they are declared.

4. Redeclaration: While `let` allows redeclaration of a variable within a different block scope, `const` does not permit redeclaration. `const` variables must be assigned a value at the time of declaration and cannot be reassigned later.

Q.6 What is template literals in ES6 and how do you use them?

Ans: Template literals, introduced in ES6, are a way to create strings in JavaScript using backticks (` `) instead of single or double quotes. They allow for easy interpolation of variables and expressions within the string using `${}` notation. Template literals also support multiline strings without the need for escape characters or concatenation. They provide a concise and readable syntax for working with strings in JavaScript.

Q.7 What's difference between map & forEach?

Ans: `- `map()` returns a new array containing the results of applying a provided function to each element of the original array. It is used for transforming and mapping values from one array to another.

- `forEach()` does not return anything and is used to execute a provided function on each element of the array. It is commonly used for performing actions on array elements, such as logging or updating values.

Q.8 How can you destructure objects and arrays in ES6?

Ans:

1. Object Destructuring:

To destructure an object, you use curly braces `{}` and specify the variable names that correspond to the object's properties.

Example: const { property1, property2 } = object;

2. Array Destructuring:

To destructure an array, you use square brackets `[]` and specify the variable names to receive the array's values.

Example: const [element1, element2] = array;

Q.9 How can you define default parameter values in ES6 functions?

Ans: In ES6, you can define default parameter values for function parameters using the following syntax: javascript

function functionName(parameter1 = defaultValue1, parameter2 = defaultValue2) {

1. When defining a function, you can assign a default value to any parameter by using the assignment operator (`=`) followed by the desired default value.

2. If a value is not provided for a parameter when the function is called, the default value will be used instead.

Q.10 What is the purpose of the spread operator (`...`) in ES6?

Ans: The spread operator (`...`) in ES6 is used to unpack elements from an array or an object. It allows you to spread the elements into individual components, making it easier to work with arrays, objects, and function arguments.