**Assignment 4 – PPT Web Dev**

Q.1 Explain Hoisting in JavaScript

Hoisting is a JavaScript behavior that allows variables and function declarations to be moved to the top of their containing scope during the compilation phase, before the code is executed. This means that regardless of where variables and functions are declared in the code, they are conceptually moved to the top of their scope.

In JavaScript, variable declarations using var are hoisted, which means that the variable names are created at the top of their scope but their assignments are left in place. This allows you to use variables before they are actually declared in the code. However, the initial value of a hoisted variable is undefined until it is assigned a value in the code.

Q.2 Explain Temporal Dead Zone?

The Temporal Dead Zone (TDZ) is a behavior in JavaScript that occurs when accessing a variable before it is declared using the let or const keywords. It is a specific phase during the variable's lifecycle where accessing the variable results in a runtime error.

When a variable is declared with let or const, it is hoisted to the top of its block scope but remains uninitialized until the declaration statement is reached during program execution. Before the declaration statement, the variable is said to be in the TDZ. During this phase, any attempt to access the variable will result in a ReferenceError.

Q.3 Difference between var & let?

var has function scope and allows variables to be accessed before their declaration, while let has block scope and enforces variables to be declared before usage, eliminating the concept of hoisting.

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

Some of the major features introduced in ECMAScript 6 (ES6), also known as ECMAScript 2015, include:

- `let` and `const` declarations for block scoping variables.

- Arrow functions, providing concise syntax for defining functions.

- Classes and inheritance syntax, simplifying object-oriented programming.

- Template literals, allowing easier string interpolation and multiline strings.

- Destructuring assignment, enabling easy extraction of values from objects and arrays.

- Default function parameters, providing default values for function arguments.

- Rest and spread operators for handling variable-length arguments and array manipulation.

- Promises for handling asynchronous operations in a more structured manner.

- Modules, introducing a standardized way to organize and share JavaScript code.

- Enhanced object literals, offering concise syntax for creating objects and defining methods.

- Iterators and generators for custom iteration over data structures.

- Symbol data type, introducing a new primitive data type for unique identifiers.

- `let` and `const` block scoping variables.

- Arrow functions for concise function syntax.

- Classes and inheritance syntax for object-oriented programming.

- Template literals for improved string interpolation.

- Destructuring assignment for extracting values from objects and arrays.

- Default function parameters for setting default values for function arguments.

- Rest and spread operators for handling variable-length arguments and array manipulation.

- Promises for managing asynchronous operations.

- Modules for organizing and sharing JavaScript code.

- Enhanced object literals for creating objects and defining methods.

- Iterators and generators for custom iteration over data structures.

- Symbol data type for unique identifiers.

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

let allows reassignment of variables, while const creates variables that cannot be reassigned once assigned a value. However, it's important to note that const does not make the assigned value immutable, only the variable reference itself.

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

Template literals, introduced in ECMAScript 6 (ES6), provide an enhanced syntax for creating strings in JavaScript. They allow for easy string interpolation and multiline string creation by using backticks (```) as delimiters.

Eg.-

const name = "John";

const greeting = `Hello, ${name}!`;

console.log(greeting); // Output: Hello, John!

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

forEach() is used to iterate over elements in an array and perform an action on each element. It doesn't return a new array and doesn't modify the original array.

map() is used to iterate over elements in an array, apply a transformation or mapping function to each element, and return a new array with the transformed values. It doesn't modify the original array.

The choice between forEach() and map() depends on whether we need to perform an action on each element or transform the elements and retrieve a new array with the transformed values.

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

To destructure objects, we use curly braces {} and specify the variable names that correspond to the object's properties:

const person = { name: "John", age: 30 };

const { name, age } = person;

console.log(name); // Output: John

console.log(age); // Output: 30

In this example, the name and age variables are assigned the corresponding values from the person object using destructuring.

Now , To destructure arrays, we use square brackets [] and specify the variable names in the desired order:

const numbers = [1, 2, 3];

const [a, b, c] = numbers;

console.log(a); // Output: 1

console.log(b); // Output: 2

console.log(c); // Output: 3

In this example, the variables a, b, and c are assigned the values from the numbers array using destructuring.

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

function greet(name = "Guest") {

console.log(`Hello, ${name}!`);

}

greet(); // Output: Hello, Guest!

greet("John"); // Output: Hello, John!

In this example, the greet() function has a parameter name with a default value of "Guest". If the name argument is not provided when calling the function, the default value will be used.

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

The spread operator offers a concise and versatile syntax for working with arrays, objects, and function arguments in JavaScript. It simplifies tasks like array merging, object cloning, function parameter handling, and more.

The spread operator (`...`) in ES6 has multiple purposes and can be used in different contexts:

1. Array Spread

2. Object Spread

3. Function Arguments

4. Destructuring