Assignment -4

Q.1 Explain Hoisting in JavaScript

Ans. It is a phenomenon in JS where we can access functions and variables in Js before initialization, without any error.

- It is all because of GEC (Global Execution Context).
- The first phase is the memory allocated phase and the second phase is the initialization phase.
- So the concept of hoisting comes, when we access variables and function in js before initialization.
- All this happens because before execution of code the memory is allocated to variables and in case of function the whole code is put, as it is.
- So when we call variables before declaration / initialization it gives us undefined as only the first phase of GEC gets over which is memory allocation phase but not the second phase that is initialization phase.
- But if we call a function, it gives us the desired output because in case of function the whole code is put, as it is in GEC.

Q.2 Explain Temporal Dead Zone?

Ans. It is the period of time during which the let and const declarations cannot be accessed.

```
console.log(x);   // ReferenceError: x is not defined
let x = 5;
```

Q.3 Difference between var & let?

Ans. Variables declared by let are only available inside the block where they're defined.

- Variables declared by var are available throughout the function in which they're declared.
- We can declare a variable again even if it has been defined previously in the same scope.
- We cannot declare a variable more than once if we defined that previously in the same scope.
- Hoisting is allowed in the var, hoisting is not allowed in the var.
- Let is ES6 feature.

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

Ans.

- ES6 introduced several key features like const, let, arrow functions,
- It introduces several new features such as, block-scoped variables, new loop for iterating over arrays and objects, template literals, and many other enhancements to make JavaScript programming easier and more fun

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

Ans.

Let	**Const**
It can be updated but cannot be re-declared into the scope.	It cannot be updated and cannot re-declared into the scope.
It can be declared without initialization.	It cannot be declared without initialization.

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

Ans.

- Template literals are a new feature that was introduced in ECMAScript6, which offers a simple method for performing string interpolation and multiline string creation.

- `Any string ${jsExpression} can appear here`

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

Ans.

forEach	Map
The forEach() method does not returns a new array based on the given array.	The map() method returns an entirely new array.
The forEach() method returns "undefined".	The map() method returns the newly created array according to the provided callback function.
The forEach() method doesn't return anything hence the method chaining technique cannot be applied here.	With the map() method, we can chain other methods like, reduce(),sort() etc.
It is not executed for empty elements.	It does not change the original array.

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

Ans. destructure objects and arrays using destructuring assignment. It allows you to extract values from objects and arrays into individual variables or assign them to new object properties or array elements.

Eg: Arrays to be destructure:

```
// Array to be destructured
const numbers = [1, 2, 3, 4, 5];

// Destructuring assignment
const [first, second, ...rest] = numbers;

// Accessing the extracted values
console.log(first); // 1
console.log(second); // 2
console.log(rest); // [3, 4, 5]
```

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

Ans.

- Default parameter values allow you to specify a default value that is used if no argument is provided for that parameter or if the argument is explicitly passed as undefined.

```
function greet(name = 'Guest') {
  console.log(`Hello, ${name}!`);
}

greet(); // Hello, Guest!
greet('John'); // Hello, John!
```

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

Ans.

- The JavaScript spread operator is denoted by three dots (…). The spread operator helps the iterable objects to expand into individual elements. Iterable objects are those on which we can use a loop, for example, Array, Map, Set, etc.

```
- const numbers = [1, 2, 3];
-
- console.log(...numbers); // 1 2 3
-
- const mergedArray = [0, ...numbers, 4, 5];
-
- console.log(mergedArray); // [0, 1, 2, 3, 4, 5]
```