Q.1 Explain Hoisting in JavaScript

Hoisting is a kind of default behaviour in which all the declarations either variable declaration or function declaration are moved at the top of the scope just before executing the program's code. However, it can be considered an advantage because all functions and variable declarations are placed to the top of their scope no matter where they are all declared anywhere in the whole program, even regardless of whether they are declared global or local.
Due to the concept of hoisting in JavaScript, we can call a function even before we define the function definition in our program's code.

JavaScript only hoists declaration, not initialization. This variable will be undefined until the line where its initialized is reached.

Hoisting with let and var

Variables defined with let and const are hoisted to the top of the block, but not initialized.

Using a let variable before it is declared will result in a ReferenceError.

Using a const variable before it is declared, is a syntax error.

Q.2 Explain Temporal Dead Zone?

A temporal dead zone is the block where a variable is inaccessible until the moment the computer initializes it with a value.

- A block can be defined as a pair of braces ({...}) used to accumulate multiple statements.
- Initialization occurs when one assigns an initial value to a variable.

If a user attempts to access a variable before its complete initialization. In such a scenario, JavaScript will subject a ReferenceError.

Q.3 Difference between var & let?

Var :

Var is a globally scoped variable

Var can be updated and re-declared within its scope

Var variable are initialized with undefined

var can be declared without initialization

Var keyword is used to declared var variable

Sytanx:

Var name = Rahul;

Hoisting is allowed with var.

Let:

Let is a block scoped variable

Let can be updated but not re-declared

Let can be declared without initialization

Let keyword is used to declare a let variable

Syntax:

let name = Rahul;

Hoisting not allowed with let.

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

let and const Keywords

Arrow Functions

Multi-line Strings

Default Parameters

Template Literals

Destructuring Assignment

Enhanced Object Literals

Promises

Classes

Modules

for...of Loop

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

Let:

Let is a block scoped variable

Let can be updated but not re-declared

Let can be declared without initialization

Let keyword is used to declare a let variable

Syntax:

let name = Rahul;

Hoisting not allowed with let.

const :

const is a block scope variable

const can neither be updatyed nor be re-declared

const variables are not initialized

const must be initialized during declaration

const keyword is used to declare const varaibles

Syntax:

const name = Rahul;

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

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

The template literals were called template strings before the introduction of ES6. Starting from ES6 (ECMAScript 6), we have Template Literals which are indicated by the backtick (` `) character. Template literals can also be used to hold the placeholders, that are indicated by the '$' sign and the {} braces such as (${expression}).

Back-Tics Syntax :

Template Literals use back-ticks (``) rather than the quotes ("") to define a string.

```
let text = `Hello World!`;
```

Quotes Inside Strings :

With template literals, you can use both single and double quotes inside a string.

```
let text = `He's often called "Johnny"`;
```

Multiline Strings :

Template literals allows multiline strings.

```
let text =
`The quick
brown fox
jumps over
the lazy dog`;
```

Interpolation :

Template literals provide an easy way to interpolate variables and expressions into strings.The method is called string interpolation.

The syntax is:

```
${...}
```

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

The map() method returns a new array, whereas

The forEach() method does not return a new array.

The map() method is used to transform the elements of an array, whereas The forEach() method is used to loop through the elements of an array.

The map() method can be used with other array methods, such as the filter () method, whereas

The forEach() method cannot be used with other array methods

The map() method returns the newly created array according to the provided callback function.

The forEach() method returns “undefined“.

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

The Destructuring assignment synatx is a javascript expression that makes it possible to unpack values from arrays, or proprtise from object, into distict variables

//array destructuring

```
const student = {name: 'Arun', position: 'First', rollno: '24'};
const {name, position, rollno} = student;

console.log(name); // Arun
console.log(position); // First
console.log(rollno); // 24
```

//array destructuring

```
const days = ["Monday","Tuesday","Wednesday","Thursady","Friday","Saturday","Sunady"]

//destructuring assignment

const [a,b, ,c,...rest] = days;
```

```
console.log(a); //Monday

console.log(b); // Tuesday

console.log(c); // Thursday

console.log(rest);  // ['Friday', 'Saturday', 'Sunady']
```

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

Default functions parameter allowed named parameter to be initialized with default values if no values or undefined is passed

```
function multiply(a, b = 1) {

  return a * b;

}


console.log(multiply(5, 2));  //10


console.log(multiply(5)); // 5
```

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

Spread Operator is a very simple and powerful feature introduced in the ES6 standard of JavaScript, which helps us to write nicer and shorter code. The   JavaScript spread operator is denoted by three dots (…).

the spread operator allows us to copy all elements from the existing array or object into another array or object.

```
let colors = ['Red', 'Yellow'];
let newColors = [...colors];


console.log(newColors);  //[ 'Red', 'Yellow' ]
```