**Q.1** Explain Hoisting in JavaScript

Hoisting in JavaScript is a behavior where variable and function declarations are moved to the top of their containing scope during the compilation phase. This means that you can use variables and call functions before they are actually declared in your code. However, only the declarations are hoisted, not the assignments or initializations. It's important to declare variables at the top of their scope and define functions before using them to avoid confusion and unexpected behavior.

**Q.2** Explain Temporal Dead Zone?

The Temporal Dead Zone (TDZ) is a behavior in JavaScript that prevents accessing variables before they are declared. It applies to variables declared with let and const keywords and throws a ReferenceError if accessed within the TDZ. It promotes better code practices by enforcing variable declaration before use and prevents potential errors caused by using variables before they are initialized.

**Q.3** Difference between var & let?

In JavaScript, var and let have significant differences in terms of scoping, hoisting, and redeclaration. var is function-scoped, which means variables declared with var are accessible throughout the entire function in which they are declared. They are hoisted to the top of their scope during the compilation phase, allowing them to be accessed before their actual declaration in the code. This can sometimes lead to unexpected behavior. var also allows redeclaration within the same scope, which can potentially overwrite variables. On the other hand, let is block-scoped, meaning variables declared with let are limited to the block in which they are defined (such as within a loop or conditional statement). Unlike var, let variables are not hoisted to the top of their scope. Instead, they are subject to the Temporal Dead Zone (TDZ), which prevents accessing them before their declaration in the code. Redeclaring a variable with let in the same block or scope will result in a SyntaxError.

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

ECMAScript 6 (ES6) introduced major features such as let and const for variable declarations, arrow functions for concise function syntax, classes for object-oriented programming, template literals for string interpolation, destructuring assignment for easy value extraction, spread and rest operators for array manipulation, enhanced object literals, modules for modular code organization, promises for handling asynchronous operations, iterators for looping, and generators for pausing and resuming function execution. These additions enhanced JavaScript with improved syntax, modularity, and asynchronous handling.

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

The key difference between let and const in JavaScript lies in their reassignability and the level of mutability they allow. With let, you can declare a variable and assign a value to it, and later on, you can change that value by reassigning it. This means that the variable can be updated or modified throughout your code. It provides flexibility and allows for variable values to be changed as needed. On the other hand, const is used to declare variables that have a constant or fixed value. Once a value is assigned to a const variable, it cannot be reassigned or changed. It represents an unchangeable value throughout the execution of the program. This ensures that the variable maintains a consistent value and prevents accidental modifications. It's important to note that the immutability of a const variable refers to the variable reference itself, not necessarily to the value it holds. If the value is an object or an array, its properties or elements can still be modified. However, the restriction of const is on reassigning the variable itself.

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

Template literals in ES6 are a way to work with strings in JavaScript by using backticks ( ) instead of single or double quotes. They allow for easy embedding of expressions using ${expression} syntax within the backticks. This enables variables, function calls, and operations to be directly included in the string. Template literals also support multiline strings without manual line breaks. They provide a concise and readable syntax for string interpolation and multiline text.

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

The main difference between map() and forEach() is that map() returns a new array with the results of the provided callback function applied to each element, while forEach() does not return anything and simply executes the callback function for each element. map() is used when you want to transform elements and collect the results, while forEach() is useful when you want to perform an action on each element without creating a new array.

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

In ECMAScript 6 (ES6), object and array destructuring offer a concise way to extract values from objects and arrays into individual variables. With object destructuring, you can use curly braces {} and specify the property names that match the object's properties to create new variables. The variables will hold the corresponding property values. Additionally, you can assign default values to handle cases when the property is undefined. Array destructuring, on the other hand, involves using square brackets [] to specify the order in which you want to extract values from an array. By providing variable names within the brackets, you can assign values to those variables based on the corresponding positions in the array. The rest operator (...) can be used in array destructuring to capture remaining elements into a new array. Both object and array destructuring provide a cleaner and more readable approach to extracting values from complex data structures in JavaScript.

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

In ES6 (ECMAScript 6), you can define default parameter values for function parameters by assigning a value directly to the parameter within the function's parameter list. This enables you to specify default values that will be used if no argument or an undefined value is passed for that parameter. The default values can be any valid JavaScript expression, allowing for flexibility in providing fallback values or performing calculations based on other parameters. By defining default parameter values, you make certain function parameters optional, providing a convenient way to handle different use cases without explicitly checking for undefined values within the function body.

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

The spread operator (...) in ES6 is used for array manipulation, function arguments, object manipulation, and cloning arrays and objects. It allows you to expand arrays into individual elements, pass arrays as arguments, merge objects, and create shallow copies of arrays and objects. It provides a concise and versatile way to work with arrays and objects in JavaScript.