Variables Assignments

Task 1: Variable Declaration and Assignment

1. Describe the differences between var, let, and const for variable declaration.

   Ans. The difference between var, let and const is that:

   - **Var:**
     - Var is a Local and Functional Scope.
     - The Functional scope means they can be accessed with in the function they were defined in.
     - Var is Hoisted. Its value is undefined until assigned.

   - **Let:**
     - Let is a blocked Scoped Variables.
     - It is introduced in ES6.
     - Let is block scoped it means it can only be accessed with in the block where they were declared.
     - It can reassign a value but cannot be redeclared.
     - It is not Hoisted.

   - **Const:**
     - Const is a blocked Scoped Variables.
     - It is introduced in ES6.
     - A Const is a type of a variable whose value cannot be changed.
     - It cannot reassign a value and cannot be redeclared as an element.
     - It is not Hoisted.

2. Provide examples of each keyword to demonstrate their behavior in different scenarios?

   Ans. **Var Example**

```
<html>
<body>
   <h1>Var Example</h1>
   <p>In this example firstName, lastName, and completeName are used for
variables.</p>
   <p id="name"></p>
   <script>
      var firstName = 'Muhammad';
      var lastName = 'Saqib';
      var completeName = firstName + " " + lastName;
      document.getElementById("name").innerHTML =
         "The Complete Name is: " + completeName;
   </script>
</body>
</html>
```

   Let Example

```
<!DOCTYPE html>
<html>
<body>
   <h1>Let Example</h1>
   <p>In this example, num1, num2, Addition, Subtraction, Multiply, and Division are
variables.</p>
   <script>
      let num1 = 6;
      let num2 = 2;
      let Addition = num1 + num2;
      let Subtraction = num1 - num2;
      let Multiply = num1 * num2;
      let Division = num1 / num2;
      document.write("Add", Addition);
      document.write("Subtract", Subtraction);
      document.write("Multiply", Multiply);
      document.write("Divide", Division);
   </script>
</body>
</html>
```

Const Example

```
<html>
<body>
<h1>Const Example</h1>

<p id="location"></p>

<script>
const city = "Bahawalpur";
const province = "Punjab";
const country = 'Pakistan';
const fullLocation = city + ', ' + province + ', ' + country;
document.getElementById("location").innerHTML =
"The Location is: " + fullLocation;
</script>

</body>
</html>
```

3. Discuss the importance of choosing the appropriate variable declaration based on the use case.

   Ans. Selecting the correct variable declaration (var, let, or const) is essential for writing clean, effective, and error-free code. All three keywords behave differently, so the correct choice depends on the particular application.

Var

- Avoid using var (Use Only When Necessary)
- Function-scoped, which could lead to bugs.
- Can be redeclared, with the increased chances of overwriting variables.

Let

- Modern Style and Blocked Scoped.
- Block-scoped (It only lives inside {}).
- Can be redeclared but not reassigned within the same scope.
- Hoisted but not initialized (Throws ReferenceError)

Const

- Const is used for Fixed Values. The value should remain constant.
- Enhances code reliability and security.
- Most suitable for things like API URLs, config variables, and constants.

TASK 2 : Hoisting

1. Define hoisting and how it affects variable declarations in JavaScript.

Ans. **Hoisting:** In JavaScript, hoisting is a behavior in which a function or a variable can be used before declaration.

There are two Type:
- Variable Hoisting
- Function Hoisting

- How Hoisting Affects Variable Declarations?

Var
- When we declare a variable using var it is hoisted to the top of its current scope.
- Var is hoisted but initialized as undefined until assigned.

Let & Const
- When we declare a variable using let or const, it is hoisted to the top of its current scope. The variable does not have a default when it is hoisted.

2. Discuss the hoisting behavior of var, let, and const variables.

Ans. The hoisting behavior of var, let, and const variables is that:

- **Var**
    - Var is hoisted but initialized as undefined until assigned.

- **Let**
    - When declaring with let, the variable is hoisted but remains in a limited stage known as the Temporal Dead Zone until assigned a value.

    - If you attempt to use it prior to the declaration, you will receive a ReferenceError.

- **Const**
    - Hoisted but not initialized, the same as Let remains in the Temporal Dead Zone.
    - If you attempt to use it prior to the declaration, you will receive a ReferenceError.

3. Provide examples to illustrate hoisting with each variable type.

Ans.

- **Var Example**
  ```
  console.log(a); // Output: undefined
  var a = 10;
  console.log(a); // Output: 10
  ```

- **Let Example**
  ```
  console.log(name); // ReferenceError: Cannot access 'name' before initialization

  let name = 'Ali';
  console.log(name); // Output: Ali
  ```

- **Const Example**
  ```
  console.log(c); //ReferenceError: Cannot access 'c' before initialization
  const c = 30;
  console.log(c); // Output: 30
  ```

Task 3: Scopes in JavaScript

1. Define the concept of scope in programming.

Ans. Scope in programming defines the accessibility and lifetime of variables in a program. It determines where a variable can be accessed or modified within the code.

In JavaScript, there are three main types of scope:

- **Global Scope** – Accessible anywhere in the program.
- **Function Scope** – Accessible only inside a function.
- **Block Scope** – Accessible only inside {} (blocks).

2. Explain global scope and how global variables are accessed throughout the code.

Ans. Global scope means that a variable can be accessed anywhere in the program. A variable declared outside any function or block becomes a global variable.

How It's Work:

- Declared outside functions and blocks
- Accessible anywhere in the script
- Can be modified from any part of the program

3. Describe function scope and how it affects variables declared inside functions.

Ans. Function Scope means variables are declared inside a function, they have a local scope and are accessible only within that function. They cannot be accessed outside the function. The types of variables are called local or functional variables.

Function Scope Works

- Variables declared with var, let, and const inside a function stay in that function.
- They cannot be accessed from outside the function.
- Each function creates its own scope, so variables inside one function don't affect variables in another function.

4. Introduce block scope and discuss its impact on variables declared using let and const.

Ans. Blocked Scope Variable means variables are accessible only with in the block {} they are defined in which can be smaller than a function scope. It cannot be accessed outside the block.

- **Applied to:**
    - Let
    - Const

5. Provide examples to demonstrate each type of scope.

Ans. **Functional Example:**

Var:

```
function myFunction() {

  var carName = "Volvo";   // Function Scope

}
```

Let:

```
function myFunction() {

  let firstName = "Saqib";   // Function Scope

}
```

Const:

```
function myFunction() {

  const animalName = "Lion";   // Function Scope

}
```

Block Scope Example

```
{
    let blockVar = "I am inside a block";
    console.log(blockVar); // Accessible inside block
}
console.log(blockVar); // Error: blockVar is not defined
```