Don Bosco Institute of Technology
Kurla (W), Mumbai 400 070

Internet Programming Lab

Name: Umme Atiya Quraishi
Class: TEIT
Roll no.: 47
Date : 30/08/2024
Experiment no. 6
Title: Design a web page using JavaScript Variables

Theory:

In JavaScript, variables are fundamental building blocks that allow you to store, manipulate, and retrieve data throughout your program. Understanding how to declare and use variables effectively is crucial for any JavaScript developer. This theory covers the types of variables, their scope, hoisting, data types, and best practices.

1. What is a Variable?

A variable is a symbolic name associated with a value in memory. It acts as a container for storing data that can be referenced and manipulated in your code. Variables allow you to give meaningful names to values, making your code more readable and maintainable.

2. Declaring Variables

JavaScript provides three keywords to declare variables:

`var`: This is the original keyword used to declare variables. Variables declared with `var` are function-scoped or globally scoped and can be re-declared and updated.
javascript
Copy code
```
var name = "Alice";
```

-

`let`: Introduced in ES6 (ECMAScript 2015), `let` allows you to declare block-scoped variables. This means the variable is only accessible within the block where it is defined.

javascript
Copy code
```
let age = 25;
```

-

`const`: Also introduced in ES6, `const` is used to declare block-scoped constants. A variable declared with `const` cannot be re-assigned after its initial value is set.
javascript
Copy code
```
const birthYear = 1998;
```

-

3. Variable Scope

Variable scope determines where a variable can be accessed in your code:

Global Scope: A variable declared outside of any function or block is globally scoped and can be accessed anywhere in the code.
javascript
Copy code
```
var globalVar = "I am global!";
```

-

Function Scope: Variables declared with `var` inside a function are scoped to that function. They cannot be accessed outside the function.
javascript
Copy code
```
function myFunction() {
    var localVar = "I am local!";
}
```

-

Block Scope: Variables declared with `let` and `const` are scoped to the nearest enclosing block (e.g., a loop or an if statement).
javascript

Copy code

```
if (true) {
    let blockScopedVar = "I am block-scoped!";
}
```

-

4. Hoisting

Hoisting is a JavaScript mechanism where variables declared with `var`, `let`, or `const` are moved to the top of their scope during the compilation phase. However, only the declarations are hoisted, not the initializations.

Example:
javascript
Copy code

```
console.log(myVar); // Output: undefined
var myVar = "Hello";
```

- In this example, the declaration `var myVar;` is hoisted to the top, but the assignment occurs later, which is why it logs `undefined`.

5. Data Types

JavaScript supports various data types, which can be broadly categorized into:

- **Primitive Types**:
  - **String**: Represents text, e.g., `"Hello, World!"`.
  - **Number**: Represents both integer and floating-point numbers, e.g., `42`, `3.14`.
  - **Boolean**: Represents logical values: `true` or `false`.
  - **Undefined**: A variable that has been declared but not assigned a value is of type `undefined`.
  - **Null**: Represents an intentional absence of any value.
  - **Symbol** (ES6): A unique and immutable data type often used as object property keys.
  - **BigInt** (ES11): A numeric type that can represent integers with arbitrary precision.

- **Non-Primitive Types**:
    - **Object**: A collection of properties (key-value pairs). Objects can represent complex data structures.

javascript
Copy code

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

-

6. Best Practices for Using Variables

- **Use `let` and `const`**: Prefer `let` for variables that will change and `const` for constants to avoid unintentional reassignments.
- **Follow Naming Conventions**: Use descriptive variable names (e.g., `userName`, `totalAmount`) to improve code readability.
- **Declare Variables at the Top**: To avoid confusion, declare all variables at the beginning of their scope.
- **Avoid Global Variables**: Minimize the use of global variables to prevent naming conflicts and unexpected behavior.

Conclusion

Understanding JavaScript variables is essential for effective programming. By mastering variable declaration, scope, hoisting, and data types, developers can write cleaner, more efficient, and more maintainable code. This foundational knowledge lays the groundwork for advanced JavaScript concepts and techniques.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <style>
```

```
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        input, button {
            padding: 10px;
            margin: 10px 0;
        }
        #result {
            margin-top: 20px;
            font-size: 18px;
            color: green;
        }
    </style>
</head>
<body>

    <p>Enter two numbers:</p>

    <label for="num1">Number 1:</label>
    <input type="number" id="num1" placeholder="Enter first number"><br>

    <label for="num2">Number 2:</label>
    <input type="number" id="num2" placeholder="Enter second number"><br>

    <button id="calculateButton">Calculate Sum</button>

    <p id="result"></p>

    <script>
        // Variables
        let number1, number2, result;

        // Function to calculate sum
        function calculateSum() {
            // Get values from input fields
            number1 = parseInt(document.getElementById("num1").value);
```

```
        number2 = parseInt(document.getElementById("num2").value);

        // Operators and Conditions
        if (isNaN(number1) || isNaN(number2)) {
            document.getElementById("result").innerHTML = "Please enter valid
numbers.";
            return;
        }

        // Using Operators
        result = number1 + number2;

        // Using Loops (Example with for loop)
        let message = "";
        for (let i = 1; i <= result; i++) {
            if (i === result) {
                message += i;
            } else {
                message += i + ", ";
            }
        }

        // Display result using Events
        document.getElementById("result").innerHTML = `Sum is:
${result}<br>Numbers from 1 to sum: ${message}`;

        // Check if sum is greater than 10 using Conditions
        if (result > 10) {
            document.getElementById("result").innerHTML += "<br>That's a large sum!";
        } else {
            document.getElementById("result").innerHTML += "<br>That's a small sum.";
        }

        // Use of Class and Object
        let user = new User("John Doe");
        user.greet();
    }
```

```
        // Class and Object Example
        class User {
            constructor(name) {
                this.name = name;
            }

            // Class Method
            greet() {
                console.log("Welcome, " + this.name);
            }
        }

        // Event Listener
        document.getElementById("calculateButton").addEventListener("click",
calculateSum);

    </script>
</body>
</html>
```

Output:

Enter two numbers:

Number 1: 3

Number 2: 1

Calculate Sum

Sum is: 4
Numbers from 1 to sum: 1, 2, 3, 4
That's a small sum.