

Task-1: Variables and Data Types

Aim: Declare a variable using var, let, and const. Assign different data types to each variable and print their values.

Theoretical Background:

Variables in JavaScript are containers that hold reusable data. It is the basic unit of storage in a program. The value stored in a variable can be changed during program execution.

JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type: String, Number, Boolean, Undefined, Null
2. Non-primitive (reference) data type: Object, Array, RegExp

Code:

```
//using var
var a = "Hello";
console.log("Value of a:", a);
//using let
let b = 1;
console.log("Value of b:", b);
//using const
const c = false;
console.log("Value of c:", c);
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE
[Running] node "c:\Users\Adminis
Value of a: Hello
Value of b: 1
Value of c: false
```

Task-2: Operators and Expressions

Aim: Write a function that takes two numbers as arguments and returns their sum, difference, product, and quotient using arithmetic operators.

Theoretical Background:

Operators are used to assign values, compare values, perform arithmetic operations, and more. There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

Code:

```
function performOperations(n1, n2) {  
  var sum = n1 + n2;  
  var diff = n1 - n2;  
  var product = n1 * n2;  
  var quotient = n1 / n2;  
  return{  
    sum: sum,  
    diff: diff,  
    product: product,  
    quotient: quotient  
  };  
}  
  
var result = performOperations(20, 4);  
console.log("Sum:", result.sum);  
console.log("Difference:", result.diff);  
console.log("Product:", result.product);  
console.log("Quotient:", result.quotient);
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  
[Running] node "c:\Users\Admini  
Sum: 24  
Difference: 16  
Product: 80  
Quotient: 5
```

Task-3: Control Flow

Aim: Write a program that prompts the user to enter their age. Based on their age, display different messages:

- If the age is less than 18, display "You are a minor."
- If the age is between 18 and 65, display "You are an adult."
- If the age is 65 or older, display "You are a senior citizen."

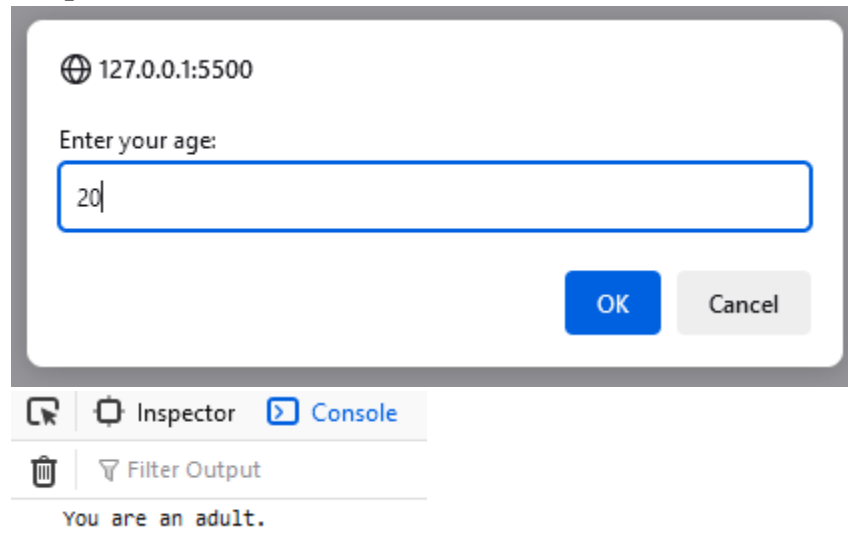
Theoretical Background:

Control flow statements are used to control the flow of execution in a program. They are used to make decisions, execute loops, and handle errors. There are three types of control flow statements in JavaScript: conditional statements, loops, and try/catch statements.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    var age = parseInt(prompt("Enter your age:"));
    if (age < 18) {
      console.log("You are a minor.");
      //document.getElementById("demo").innerHTML="You are
a minor."
    }
    else if (age >= 18 && age <= 65) {
      console.log("You are an adult.");
      //document.getElementById("demo").innerHTML="You are
an adult."
    }
    else {
```

```
        console.log("You are a senior citizen.");  
        //document.getElementById("demo").innerHTML="You are  
a senior citizen."  
    }  
    </script>  
</body>  
</html>
```

Output:

The screenshot shows a web browser window with a dialog box titled "127.0.0.1:5500". Inside the dialog box, there is a label "Enter your age:" followed by a text input field containing the number "20". Below the input field are two buttons: "OK" and "Cancel". Below the dialog box, there is a toolbar with icons for "Inspector" and "Console". The "Console" tab is selected, showing the output "You are an adult."

Task-4: Functions

Aim: Write a function that takes an array of salary as an argument and returns the min/max salary in the array.

Theoretical Background:

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code. There are mainly two advantages of JavaScript functions.

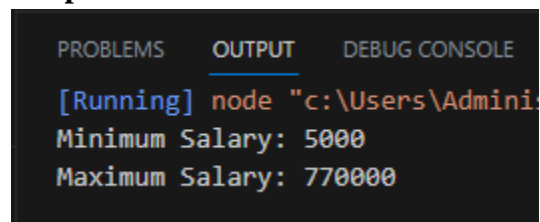
1. Code reusability: We can call a function several times so it save coding.
2. Less coding: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

Code:

```
function findingMinMaxSalary(salaries) {  
    var minSalary = salaries[0];
```

```
var maxSalary = salaries[0];
for (var i = 1; i < salaries.length; i++) {
    if (salaries[i] < minSalary) {
        minSalary = salaries[i];
    }
    if (salaries[i] > maxSalary) {
        maxSalary = salaries[i];
    }
}
return {
    min: minSalary,
    max: maxSalary
};
}
var salaries = [450000, 10000, 5000, 770000, 54000];
var result1 = findingMinMaxSalary(salaries);
console.log("Minimum Salary:", result1.min);
console.log("Maximum Salary:", result1.max);
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE
[Running] node "c:\Users\Admini...
Minimum Salary: 5000
Maximum Salary: 770000
```

Task-5: Arrays and Objects

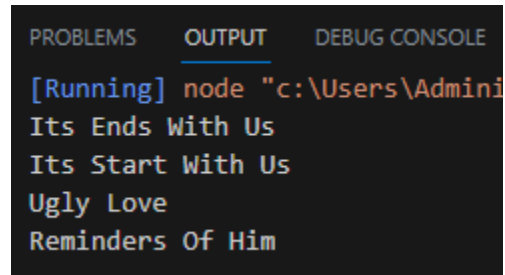
Aim: Create an array of your favorite books. Write a function that takes the array as an argument and displays each book title on a separate line.

Theoretical Background:

Both objects and arrays are considered “special” in JavaScript. Objects represent a special data type that is mutable and can be used to store a collection of data (rather than just a single value). Arrays are a special type of variable that is also mutable and can also be used to store a list of values.

Code:

```
<!DOCTYPE html>
var favBooks = [
    "Its Ends With Us",
    "Its Start With Us",
    "Ugly Love",
    "Reminders of Him",
];
function displayBookTitles(books) {
    for (var i = 0; i < books.length; i++) {
        console.log(books[i]);
    }
}
displayBookTitles(favBooks);
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE
[Running] node "c:\Users\Admini
Its Ends With Us
Its Start With Us
Ugly Love
Reminders Of Him
```

Task-6: Scope and Hoisting

Aim: Declare a variable inside a function and try to access it outside the function. Observe the scope behavior and explain the results. [var vs let vs const]

Theoretical Background:

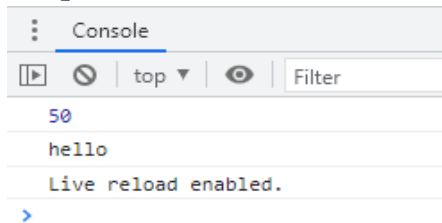
Scoping: Scoping is determining where variables, functions, and objects are accessible in your code during runtime. This means the scope of a variable (where it can be accessed) is controlled by the location of the variable declaration.

Hoisting: Hoisting is a concept that enables us to extract values of variables and functions even before initializing/assigning value without getting errors.

Code:

```
//task6
let x1

function f() {
  x1 = 50
  z1 = "hello"
}
f()
console.log(x1)
console.log(z1)
```

Output:**Task-7: DOM Manipulation**

Aim: Create an HTML page with a button. Write JavaScript code that adds an event listener to the button and changes its text when clicked.

Theoretical Background:

The DOM in dom manipulation in javascript stands for Document Object Model. The Document Object Model (DOM) is a tree-like structure illustrating the hierarchical relationship between various HTML elements. By manipulating the DOM, we can create web applications that update the data in a web page without refreshing the page and can change its layout without doing a refresh. Throughout the document, items can be deleted, moved, or rearranged.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
</head>
<body>
```

```
Enter Text:
<input type="text" name="Enter Text" id="idText">
<input type="button" value="Enter" id="idBtn"
onclick="changeText()">
<p id="idDemo"></p>
<script>
    function changeText() {
        var data = document.getElementById("idText").value;
        document.getElementById("idDemo").textContent =
data;
    }
</script>
</body>
</html>
```

Output:

Enter Text:

abc123

Task-8: Error Handling

Aim: Write a function that takes a number as an argument and throws an error if the number is negative. Handle the error and display a custom error message.

Theoretical Background:

JavaScript provides error-handling mechanism to catch runtime errors using try-catch-finally block.

try: wrap suspicious code that may throw an error in try block.

catch: write code to do something in catch block when an error occurs. The catch block can have parameters that will give you error information. Generally catch block is used to log an error or display specific messages to the user.

finally: code in the finally block will always be executed regardless of the occurrence of an error. The finally block can be used to complete the remaining task or reset variables that might have changed before error occurred in try block.

Code:

```
//Task-8
function checkPositiveNumber(number) {
    console.log(number)
    if (number < 0) {
        throw new Error("Negative numbers are not allowed.");
    }

    // If the number is positive, perform some other operations here
    console.log("Number:", number);
}

try {
    checkPositiveNumber(-6);
} catch (error) {
    console.error("Error:", error.message);
}
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

[Running] node "c:\Users\DELL\Desktop\FSWD\tasks.js"
8
Number: 8

[Done] exited with code=0 in 0.337 seconds

[Running] node "c:\Users\DELL\Desktop\FSWD\tasks.js"
-6
Error: Negative numbers are not allowed.

[Done] exited with code=0 in 0.638 seconds
```

Task-9: Asynchronous JavaScript

Aim: Write a function that uses `setTimeout` to simulate an asynchronous operation. Use a callback function to handle the result.

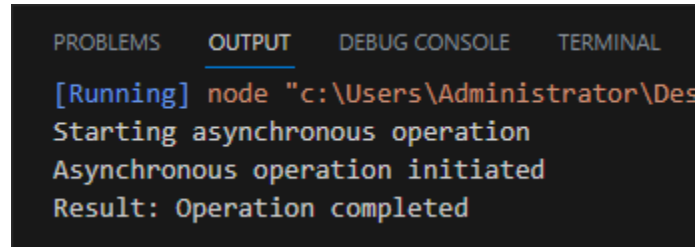
Theoretical Background:

Asynchronous programming is a technique that enables your program to start a potentially long-running task and still be able to be responsive to other events while that task runs, rather than having to wait until that task has finished. Once that task has finished, your program is presented with the result.

Code:

```
function simulateAsyncOperation(callback) {
    setTimeout(function() {
```

```
    var result = "Operation completed";  
    callback(result);  
    }, 2000);  
}  
function handleResult(result) {  
    console.log("Result:", result);  
}  
console.log("Starting asynchronous operation");  
simulateAsyncOperation(handleResult);  
console.log("Asynchronous operation initiated");
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
[Running] node "c:\Users\Administrator\Desktop\...  
Starting asynchronous operation  
Asynchronous operation initiated  
Result: Operation completed
```

Learning Outcome:

In this practical, I learnt about basics of JavaScript. It is used for making interactive webpages.