

Task-1

Variable and Datatypes

Aim:

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

Theoretical Background:

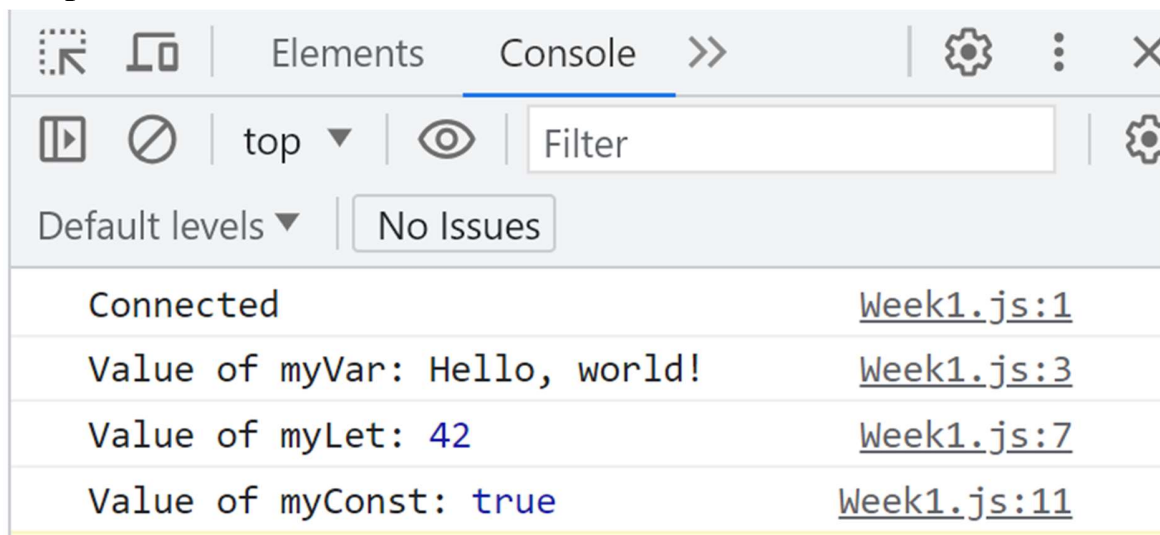
Variable in javascript are containers that holds reusable data. It is the basic unit of storage in a program. In JavaScript, all the variables must be declared before they can be used. JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript:

->Primitive and Non-primitive

Source Code:

```
console.log("Connected");  
var myVar = "Hello, world!";  
console.log("Value of myVar:", myVar);  
//document.getElementById("demo").innerHTML=myVar  
let myLet = 42;  
console.log("Value of myLet:", myLet);  
//document.getElementById("demo").innerHTML=myLet;  
  
const myConst = true;  
console.log("Value of myConst:", myConst);
```

Output:



Connected	Week1.js:1
Value of myVar: Hello, world!	Week1.js:3
Value of myLet: 42	Week1.js:7
Value of myConst: true	Week1.js:11

Task-2

Operators and Expressions

Aim:

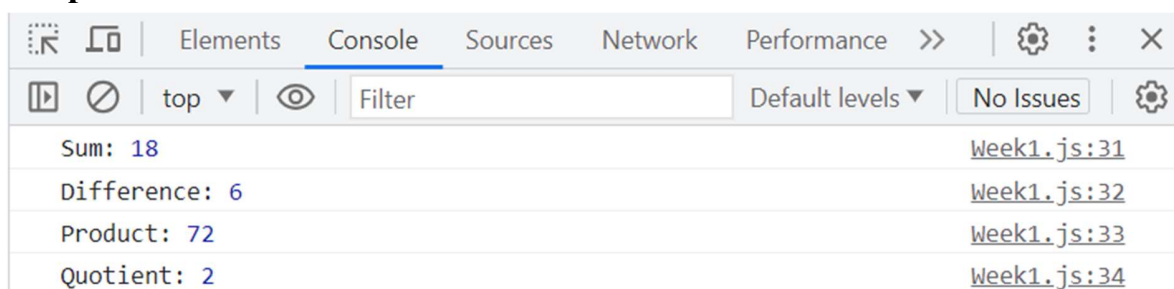
Write a function that takes two numbers as arguments and returns their sum, difference, product, and quotient using arithmetic operators.

Theoretical Background: JavaScript operators are symbols that are used to perform operations on operands. JavaScript's expression is a valid set of literals, variables, operators, and expressions that evaluate a single value that is an expression. This single value can be a number, a string, or a logical value depending on the expression.

Source Code:

```
function performOperations(num1, num2) {  
    var sum = num1 + num2;  
    var difference = num1 - num2;  
    var product = num1 * num2;  
    var quotient = num1 / num2;  
  
    return {  
        sum: sum,  
        difference: difference,  
        product: product,  
        quotient: quotient  
    };  
}  
  
var result = performOperations(12, 6);  
console.log("Sum:", result.sum);  
console.log("Difference:", result.difference);  
console.log("Product:", result.product);  
console.log("Quotient:", result.quotient);
```

Output:



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: The control flow is the order in which the computer executes statements in a script. Code is run in order from the first line in the file to the last line, unless the computer runs across the (extremely frequent) structures that change the control flow, such as conditionals and loops.

Source Code:

```
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."
}
```

Output:

127.0.0.1:5500 says

Enter your age:

You are a minor.

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: A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

Source Code:

```
function findMinMaxSalary(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 = [50000, 60000, 25000, 100000, 55000];
var result1 = findMinMaxSalary(salaries);

console.log("Minimum Salary:", result1.min);
console.log("Maximum Salary:", result1.max);
```

Output:

Elements

Console

Sources

Network

Performance

>>

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: JavaScript array is an object that represents a collection of similar type of elements. JavaScript is an object-based language. Everything is an object in JavaScript. JavaScript is template based not class based. Here, we don't create class to get the object. But, we directly create objects.

Source Code:

```
var favoriteBooks = [  
    "Its ends with us",  
    "Its start with us",  
    "Ugly",  
    "Wings of fire",  
];  
function displayBookTitles(books) {  
    for (var i = 0; i < books.length; i++) {  
        console.log(books[i]);  
    }  
}  
displayBookTitles(favoriteBooks);
```

Output:

Elements

Console

Sources

Network

Performance

>>

top ▾

Filter

Default levels ▾

No Issues

Its ends with us

Week1.js:89

Its start with us

Week1.js:89

Ugly

Week1.js:89

Wings of fire

Week1.js:89

Task-6

Scope and Hoisting

Aim:

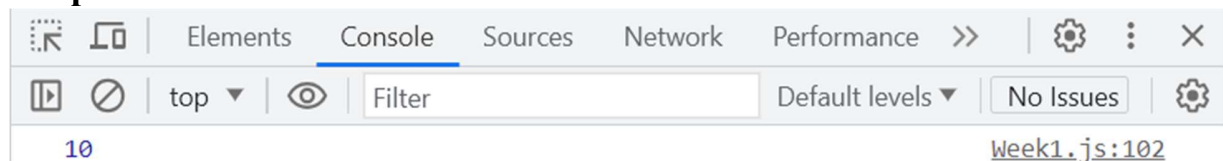
Declare a variable inside a function and try to access it outside the function. Observe the scope behavior and explain the results. [var vs let vs const].

Theoretical Background: In JavaScript, scope refers to the visibility and accessibility of variables, functions, and objects in some particular part of your code during runtime. Hoisting is a behavior in JavaScript where variable and function declarations are moved to the top of their respective scopes during the compilation phase before the code is executed.

Source Code:

```
var x; // Declare the variable outside the function
function myFunction() {
    x = 10; // Assign a value to the variable
}
myFunction();
console.log(x);
```

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: DOM manipulation refers to the process of manipulating the Document Object Model (DOM) using JavaScript. The DOM represents the structure of an HTML or XML document and provides a programming interface for interacting with and modifying the document.

Source Code:

```
var button = document.getElementById("myButton");
button.addEventListener("click", function() {
    button.textContent = "Button Clicked!";});
```

Output:

Button Clicked!

Task-8

Error Handling

Aim:

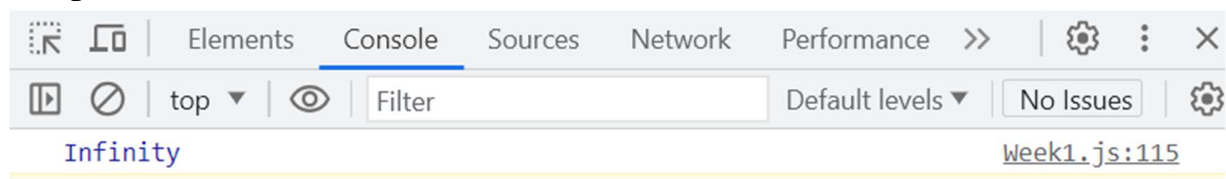
Write a function that takes a number as an argument and throws an error if the number is negative. Handle the error and display a custom error message.

Theoretical Background: Error handling in JavaScript involves the process of identifying and handling errors that occur during the execution of your code. By implementing proper error handling techniques, you can gracefully handle exceptions and prevent your program from crashing.

Source Code:

```
try {  
    // Code that might throw an error  
    var result = 10 / 0; // Division by zero  
    console.log(result); // This line will not be executed  
} catch (error) {  
    // Code to handle the error  
    console.log("An error occurred: " + error.message);  
}
```

Output:



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 in JavaScript allows you to execute code without blocking the execution of other tasks. It is particularly useful when dealing with time-consuming operations, network requests, file operations, and other tasks that may take some time to complete.

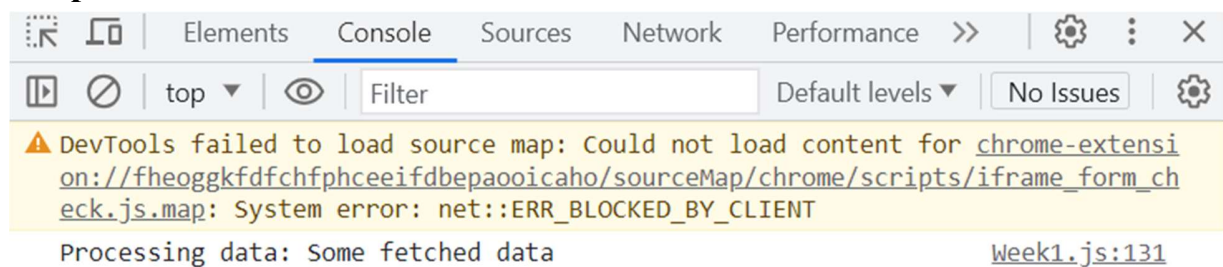
Source Code:

```
function fetchData() {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            var data = "Some fetched data";
            resolve(data);
        }, 2000);
    });
}

function processData(data) {
    console.log("Processing data: " + data);
}

fetchData()
    .then(processData)
    .catch(function(error) {
        console.log("Error occurred: " + error);
    });
```

Output:



Course Outcome:

CO4 : Demonstrate the use of JavaScript to fulfill the essentials of front-end development To back-end development.