Task-1

Aim:

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

Description:

Javascript has 8 datatypes:

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

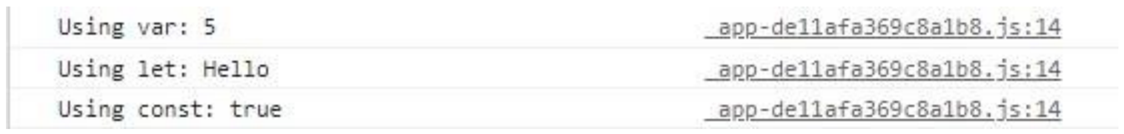
Source Code:

```
var a = 5; // integer
console.log("Using var: " + a);

let b = "Hello"; // string
console.log("Using let: " + b);

const c = true; // boolean
console.log("Using const: " + c);
```

Output:



```
Using var: 5                                  app-de11afa369c8a1b8.js:14
Using let: Hello                              app-de11afa369c8a1b8.js:14
Using const: true                             app-de11afa369c8a1b8.js:14
```

Task-2

Aim:

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

Description:

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 calculateOperations(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
  };
}
// Example usage var n1 = 10; var n2 =
375.2; var result =
calculateOperations(n1, n2);
```

console.log("Sum:", result.sum);

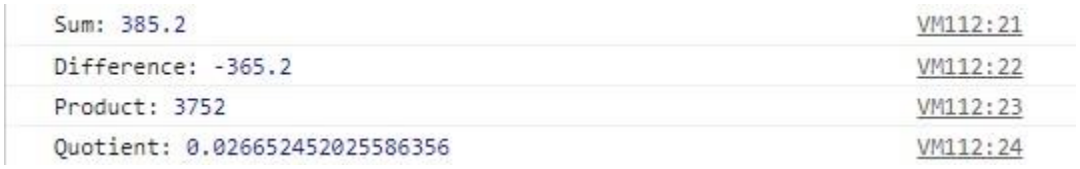
console.log("Difference:",

result.difference);

console.log("Product:", result.product);

console.log("Quotient:",

result.quotient); **Output:**



```
Sum: 385.2                                    VM112:21
Difference: -365.2                            VM112:22
Product: 3752                                 VM112:23
Quotient: 0.026652452025586356                VM112:24
```

Task-3

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."

Description:

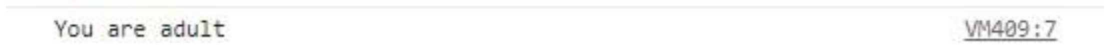
The prompt() method displays a dialog box that prompts the user for input. The prompt() method returns the input value if the user clicks "OK", otherwise it returns null.

Source code:

```
age=prompt("Enter your age");

if (age<18) {
    console.log("You are minor");
} if (age>=18 && age<=65){
console.log("You are adult")
} if (age>65){ console.log("You are
senior citizen")
}
```

Output:



```
You are adult                                                    VM409:7
```

Task-4

Aim:

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

Description:

The Array object, as with arrays in other programming languages, enables storing a collection of multiple items under a single variable name.

Source code:

```
function minMaxSalary(salaries) {
  if (salaries.length === 0) {
    return [null, null]; // Return [null, null] if the array is empty
  }

  let minSalary = maxSalary = salaries[0]; // Initialize minSalary and maxSalary
with the first element

  for (let i = 1; i < salaries.length; i++) {
    if (salaries[i] < minSalary) {
      minSalary = salaries[i];
    } else if (salaries[i] > maxSalary) {
      maxSalary = salaries[i];
    }
  }

  return [minSalary, maxSalary];
}

const salaries = [50000, 60000, 45000, 70000, 55000]; const [minimum,
maximum] = minMaxSalary(salaries); console.log("Minimum salary:", minimum);
console.log("Maximum salary:", maximum);
```
Output:

```
Minimum salary: 45000
Maximum salary: 70000
```

Task-5

Aim:

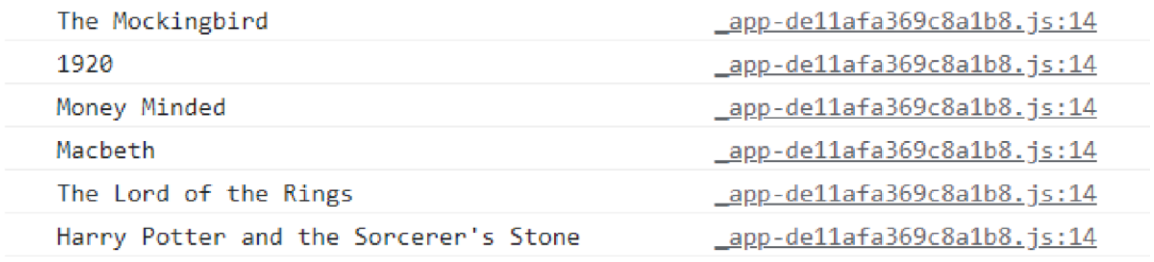
Create an array of your favorite books. Write a function that takes the array as an argument and displays each book title on a separate line.

Source code:

```
const favoriteBooks = [
  "The Mockingbird",
  "1920",
  "Money Minded",
  "Macbeth",
  "The Lord of the Rings",
  "Harry Potter and the Sorcerer's Stone"
];

function displayBooks(books) {
  for (let i = 0; i < books.length; i++) {
    console.log(books[i]);
  }
}
displayBooks(favoriteBooks);
```

Output:



```
The Mockingbird                          _app-de11afa369c8a1b8.js:14
1920                                     _app-de11afa369c8a1b8.js:14
Money Minded                             _app-de11afa369c8a1b8.js:14
Macbeth                                  _app-de11afa369c8a1b8.js:14
The Lord of the Rings                    _app-de11afa369c8a1b8.js:14
Harry Potter and the Sorcerer's Stone    _app-de11afa369c8a1b8.js:14
```

Task-6

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]

Description:

In JavaScript, the behavior of variables declared using var, let, and const differs in terms of scope.

Variables declared with var are function-scoped and can be accessed outside the function.

Variables declared with let and const are block-scoped and cannot be accessed outside the block (including functions) in which they are declared.

Variables defined with let and const are hoisted to the top of the block, but not initialized.

Meaning: The block of code is aware of the variable, but it cannot be used until it has been declared.

Using a let variable before it is declared will result in a ReferenceError. The variable is in a "temporal dead zone" from the start of the block until it is declared

Source code:

a)Scope:

i) **Using Var:** function

```
    testScope() {
  var variable = "Hello, world!";
} testScope(); console.log(variable); // Output:

"Hello, world!"
```

Output:

```
"Hello, world!"
```

ii) Using let:

```
function testScope() {
  let variable = "Hello, world!";
} testScope();

console.log(variable);
```

Output:

```
// Uncaught ReferenceError: variable is not defined
```

iii) Using const:

```
    function testScope() {
  const variable = "Hello, world!";
}

testScope();

console.log(

variable);
```

Output:

```
// Uncaught ReferenceError: variable is not defined
```

b)Hoisting:

case1:

```
x = 5; // Assign 5 to x
```

```
elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x; // Display x in the element

var x; // Declare x
```

Case2:

Using let and const variables:

```
carName = "Volvo";
let carName;
```

Output:

ReferenceError

```
carName = "Volvo";
const carName;
```

Output:

This code will not run.

Task-7

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.

Description:

When a web page is loaded, the browser creates a Document Object Model of the page.

With the object model, JavaScript gets all the power it needs to create dynamic HTML.

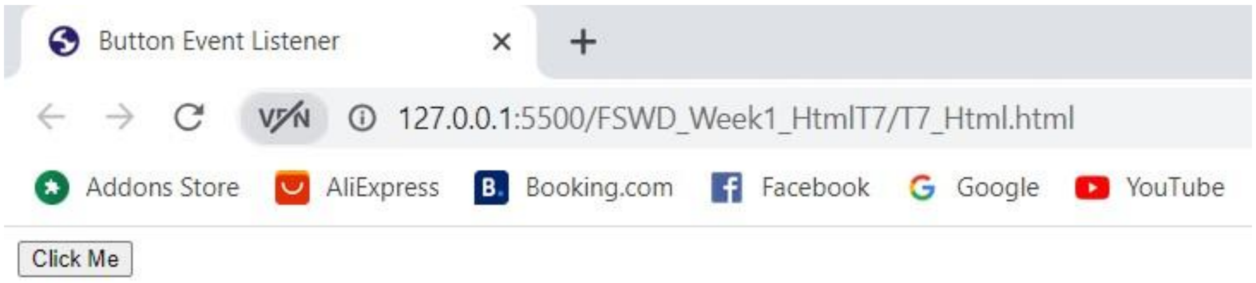
Source code:

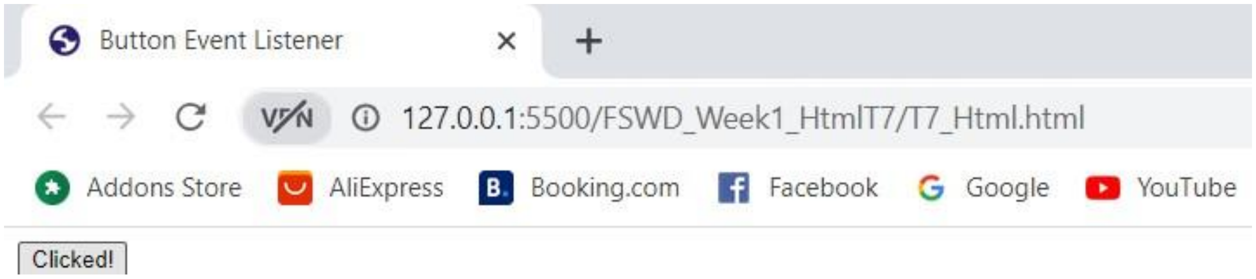
```
<!DOCTYPE html>
<html>
<head>
  <title>Button Event Listener</title>
</head>
<body>
  <button id="myButton">Click Me</button>

  <script> const button =
    document.getElementById("myButton");

    button.addEventListener("click", function() {
      button.textContent = "Clicked!";
    });
  </script>
</body>
</html>
```

Output:





Task-8

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.

Description:

JavaScript is a loosely-typed language. It does not give compile-time errors. So sometimes you will get a runtime error for accessing an undefined variable or calling undefined function etc. try catch block does not handle syntax errors.

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

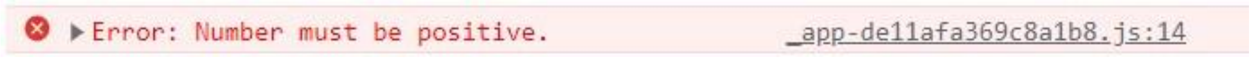
Source code:

```
function checkPositiveNumber(number) {
  if (number < 0) {
    throw new Error("Number must be positive.");
  }

  return number;
}

try {
  const inputNumber = -5; const result =
  checkPositiveNumber(inputNumber);
  console.log("Result:", result);
} catch (error) { console.error("Error:",
  error.message);
}
```

Output:



```
▶ Error: Number must be positive.                _app-de11afa369c8a1b8.js:14
```

Task-9

Aim:

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

Description:

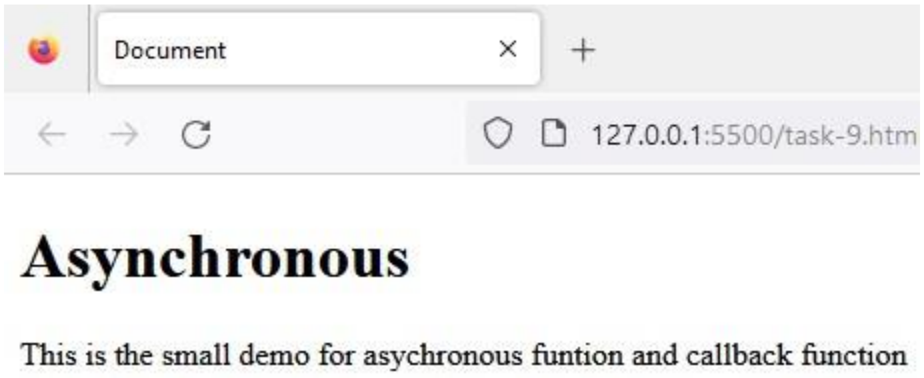
setTimeout() is an asynchronous function, meaning that the timer function will not pause execution of other functions in the functions stack. In other words, you cannot use setTimeout() to create a "pause" before the next function in the function stack fires.

Source 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>
   <h1>Asynchronous</h1>
   <p id="AD"></p>
   <script> let
    l=document.getElementById("AD");
    setTimeout(fun=>{
      l.innerHTML="This is the small demo for asychronous funtion and callback
function";
    },2532);
   </script>
</body>
</html>
```

Output:



Asynchronous

This is the small demo for asychronous funtion and callback function

Learning Outcome:

From above practicals I learnt the basics of javascript as well as how powerful this language is which is CO1. It also made me learnt to demonstrate js for front end as well as back end development.