Task-1

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

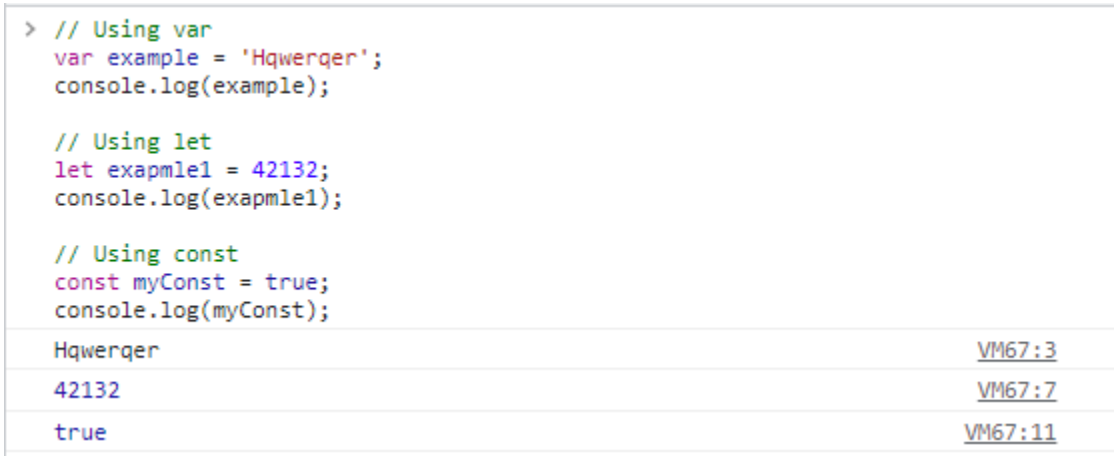
Source Code:

```
// Using var
var example = 'Hqwerqer';
console.log(example);
// Using let
let exapmle1 = 42132;
console.log(exapmle1);
// Using const
const myConst = true;
console.log(myConst);
```

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

Output:



Task-2

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

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
  };
}
var num1 = 10;
var num2 = 5;
var result = calculateOperations(num1, num2);
console.log(result);
```

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.

Output:

```
> 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
    };
  }
  var num1 = 10;
  var num2 = 5;
  var result = calculateOperations(num1, num2);

  console.log(result);
▶ {sum: 15, difference: 5, product: 50, quotient: 2}        VM46:18
```

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."

Source Code:

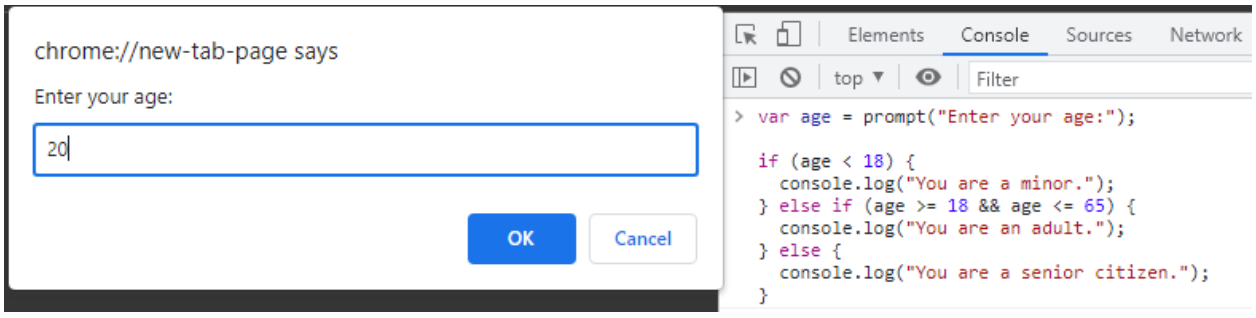
```
var age = prompt("Enter your age:");

if (age < 18) {
  console.log("You are a minor.");
} else if (age >= 18 && age <= 65) {
  console.log("You are an adult.");
} else {
  console.log("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.

Output:



```
> var age = prompt("Enter your age:");

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

Task-4

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

Source Code:

```
function findMinMaxSalary(salaries) {
  if (salaries.length == 0) {
    return {
      minSalary: null,
      maxSalary: null
    };
  }

  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 {
    minSalary: minSalary,
    maxSalary: maxSalary
  };
}
var salaries = [50000, 60000, 45000, 70000, 55000];
var result = findMinMaxSalary(salaries);
console.log("Minimum Salary: " + result.minSalary);
console.log("Maximum Salary: " + result.maxSalary);
```

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).

Output:

```
> function findMinMaxSalary(salaries) {
    if (salaries.length === 0) {
      return {
        minSalary: null,
        maxSalary: null
      };
    }

    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 {
      minSalary: minSalary,
      maxSalary: maxSalary
    };
  }
  var salaries = [50000, 60000, 45000, 70000, 55000];
  var result = findMinMaxSalary(salaries);

  console.log("Minimum Salary: " + result.minSalary);
  console.log("Maximum Salary: " + result.maxSalary);
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:

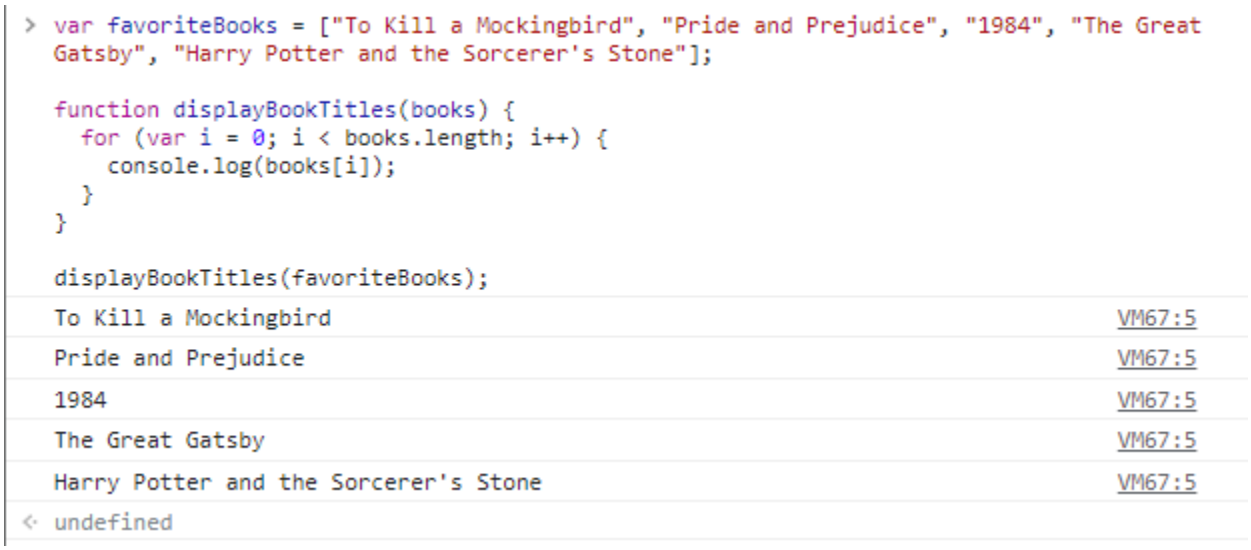
```
var favoriteBooks = ["To Kill a Mockingbird", "Pride and Prejudice", "1984", "The Great Gatsby", "Harry Potter and the Sorcerer's Stone"];

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

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 direct create objects.

Output:



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].

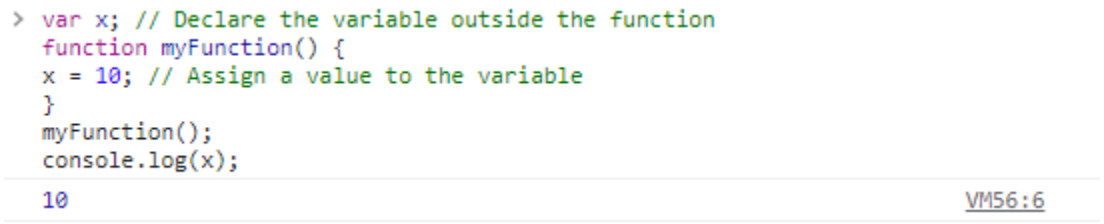
Source Code:

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

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.

Output:



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.

Source Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Button Text Change</title>
</head>
<body>
  <button id="myButton">Click me</button>

  <script>
    // Get the button element
    var button = document.getElementById("myButton");

    // Add event listener to the button
    button.addEventListener("click", function() {
      // Change the text of the button
      button.innerText = "Clicked!";
    });
  </script>
</body>
</html>
```

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.

Output:

index.html styles.css script.js Task 7 NEW

```
<!DOCTYPE html>
<html>
<head>
  <title>Button Text Change</title>
</head>
<body>
  <button id="myButton">Click me</button>

  <script>
    // Get the button element
    var button = document.getElementById("myButton");

    // Add event listener to the button
    button.addEventListener("click", function() {
      // Change the text of the button
      button.innerText = "Clicked!";
    });
  </script>
</body>
</html>
```

Click me

index.html styles.css script.js Task 7

```
<!DOCTYPE html>
<html>
<head>
  <title>Button Text Change</title>
</head>
<body>
  <button id="myButton">Click me</button>

  <script>
    // Get the button element
    var button = document.getElementById("myButton");

    // Add event listener to the button
    button.addEventListener("click", function() {
      // Change the text of the button
      button.innerText = "Clicked!";
    });
  </script>
</body>
</html>
```

Clicked!

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.

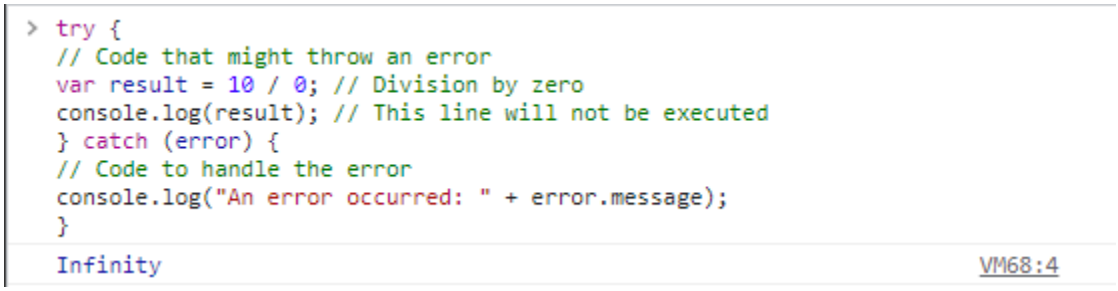
Source Code:

```
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);
}
```

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.

Output:



Task-9

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

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);
});
```

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.

Output:

```
> 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);
  });
<· ▶ Promise {<pending>}
  Processing data: Some fetched data                    VM72:10
```

Course Outcome (CO):

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