**Task – 1**

**Aim:**

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

**Theoretical background:**

|  |  |  |
| --- | --- | --- |
| **var** | **let** | **const** |
| **The scope of a**[***var***](https://www.geeksforgeeks.org/javascript-var/)**variable is functional scope.** | **The scope of a**[***let***](https://www.geeksforgeeks.org/javascript-let/)**variable is block scope.** | **The scope of a *[const](https://www.geeksforgeeks.org/javascript-const/" \t "_blank)* variable is block scope.** |
| **It can be updated and re-declared into the scope.** | **It can be updated but cannot be re-declared into the scope.** | **It cannot be updated or re-declared into the scope.** |
| **It can be declared without initialization.** | **It can be declared without initialization.** | **It cannot be declared without initialization.** |

**Source Code:**

// Using var

var myVar = 'Hello';

console.log(myVar);

// Using let

let myLet = 123;

console.log(myLet);

// Using const

const myConst = true;

console.log(myConst);

**Learning Outcome:**

Understand the difference between var, let, and const in terms of scope and reassignment. Learn how to print variable values using console.log() for debugging or displaying information during program execution.

**Task – 2**

**Aim:**

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

**Theoretical background:**

This function allows you to perform arithmetic operations on two numbers in JavaScript and retrieve the sum, difference, product, and quotient.

Finally, the function returns an array containing the sum, difference, product, and quotient.

**Source Code:**

function Operations(num1, num2) {

    var sum = num1 + num2;

    var difference = num1 - num2;

    var mul = num1 \* num2;

    var div = num1 / num2;

    console.log(`Sum is : ${sum}`);

    console.log(`Difference is : ${difference}`);

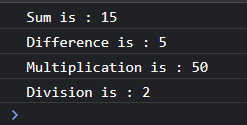
    console.log(`Multiplication is : ${mul}`);

    console.log(`Division is : ${div}`);

  }

Operations(10, 5);

**Output:**

****

**Learning Outcome:**

Understand how to define and use a function in JavaScript. Learn how to pass arguments to a function and retrieve them inside the function.Understand how to store and return multiple values using an array.

**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."

**Theoretical background:**

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:**

function ageCalculation() {

    var age = parseInt(prompt("Please 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 {

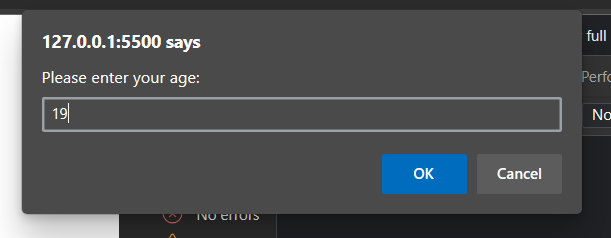
      console.log("You are a senior citizen.");

    }

}

ageCalculation();

**Output:**

****

****

**Learning Outcome:**

Understand how to use prompt() to get user input in JavaScript. Learn how to convert a string to an integer using parseInt().

**Task – 4**

**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 array is an object that represents a collection of similar type of elements.

There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword

**Source Code:**

function findMinMaxSalary(salaries) {

    //spread operator(...) : that allows an iterable (such as an array or a string) to be expanded into individual elements.

    var minSalary = Math.min(...salaries);

    var maxSalary = Math.max(...salaries);

    return [minSalary, maxSalary];

  }

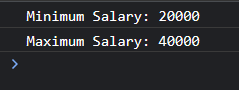
var salaries = [25000, 30000, 40000, 20000, 35000];

var result = findMinMaxSalary(salaries);

console.log("Minimum Salary: " + result[0]);

console.log("Maximum Salary: " + result[1]);

**Output:**



**Learning Outcome:**

The concept of function parameters and how they allow data to be passed into functions for processing. The concept that variables declared inside a function are accessible within that function but not outside of it.

**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.

**Theoretical background:**

Similar to C/C++/Java, we can pass the entire array as a parameter to a [function in JavaScript](https://www.scientecheasy.com/2022/02/function-in-javascript.html/). This method of array passing is called [call by reference in JavaScript](https://www.scientecheasy.com/2022/02/pass-by-value-in-javascript.html/). To pass an array argument to a function, simply pass the name of an array (a reference to an array) without brackets.

**Source Code:**

var favBooks = ["abc", "def", "ghi", "jkl", "mno"];

function displayBooks(books) {

  for (var i = 0; i < books.length; i++) {

    console.log(books[i]);

  }

}

displayBooks(favBooks);

**Learning Outcome:**

Learn the concept of iteration techniques, such as forEach or for loop, to access and display elements from an array. And concept of writing a function that takes an array as an argument.

**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]

**Theoretical background:**

In JavaScript, when a variable is declared with var inside a function, it has function-level scope and can be accessed outside the function. However, if the variable is declared with let or const, it has block-level scope and is not accessible outside the block or function it is declared in. This behavior prevents accidental variable leakage and promotes better scoping practices in JavaScript code.

**Source Code:**

//6.1 using let

function myFunction() {

    let y = 10;

    console.log(y); // Output 10

}

myFunction();

console.log(y); // Error: y is not defined

//6.2 using const

function myFunction() {

    const z = 20;

    console.log(z); // Output: 20

}

myFunction();

console.log(z); // Error: z is not defined

//6.3 using var

function myFunction() {

  var z = 40;

  console.log(z); // Output: 20

}

myFunction();

console.log(z); // Error: z is not defined

**Learning Outcome:**

Understanding of how variable accessibility varies based on their scope (function-scoped vs. block-scoped).

**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.

**Theoretical background:**

In an HTML page, JavaScript code can be used to select a button element and attach an event listener to it. When the button is clicked, the JavaScript code can modify the text content of the button element, providing dynamic interactivity to the user interface. This approach demonstrates how JavaScript can manipulate HTML elements and respond to user interactions, enhancing the user experience and enabling the creation of interactive web applications.

**Source Code:**

document.addEventListener("DOMContentLoaded", function(event) {

  var button = document.getElementById("btn");

  button.addEventListener("click", function(){

    button.innerHTML = "Button Clicked";

  });

});

**Learning Outcome:**

Gain practical knowledge of event handling in JavaScript and how to execute code in response to user interactions. learn how to dynamically modify HTML elements using JavaScript.

**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.

**Theoretical background:**

In JavaScript, a function can be defined to take a number as an argument, and by using conditional statements, an error can be thrown if the number is negative. The error can be handled using a try...catch block, where a custom error message can be displayed to provide meaningful feedback to the user or aid in debugging. This allows for robust error handling and controlled execution flow in response to exceptional cases.

**Source Code:**

function checkNumber(number) {

  if (number < 0) {

    throw new Error("The number cannot be negative.");

  }

  return number;

}

function handleInput() {

  var userInput = parseFloat(prompt("Enter a number:"));

  try {

    var result = checkNumber(userInput);

    alert("The number is: " + result);

  } catch (error) {

    alert("Error: " + error.message);

  }

}

**Learning Outcome:**

understanding of error handling mechanisms in JavaScript and how to handle exceptional cases.

**Task – 9**

**Aim:**

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

**Theoretical background:**

Asynchronous operations in JavaScript allow tasks to be executed independently, and setTimeout is used to simulate delays. Callback functions are employed to handle the results, enabling the program to continue executing other tasks while waiting for asynchronous operations to complete, thus preventing blocking and enhancing responsiveness.

**Source Code:**

//task-9

function timeOut(callback) {

  setTimeout(function() {

    var result = 42;

    callback(result);

  }, 2000);  //gap 2 seconds

}

function printResult(result) {

  console.log("Result:", result);

}

timeOut(printResult);

**Learning Outcome:**

Understanding of how asynchronous operations work in JavaScript and how to handle their results using callback functions.