Name Rohan Lalchandani
Class: D15A   Roll no.: 25

Lab Exercise 5

Title: Assignment onAdvanced Javascript

Objectives:

1. Write a javascript code to implement a basic calculator using Promises that performs addition, subtraction, multiplication, and division using promises. The calculator should accept two numbers and an operation and return the result as a promise. In case of invalid operation (wrong operator or divide by zero) let it reject with an error message
2. Creating a custom iterator for an array of numbers. The iterator should produce squares of all numbers in the array.
3. Execute generator function that generates prime numbers using JavaScript generators. The generator should produce prime numbers up to a specified limit.

Theory

1. What is a Promise in JavaScript, and what problem does it solve?

A **Promise** in JavaScript is an object representing the eventual completion (or failure) of an asynchronous operation and its resulting value. It allows developers to write asynchronous code in a cleaner, more readable way, avoiding the "callback hell" that occurs when multiple callbacks are nested within each other.

A Promise has three possible states:

- **Pending**: The initial state, neither fulfilled nor rejected.
- **Fulfilled**: The operation was completed successfully, and the promise has a value.
- **Rejected**: The operation failed, and the promise has a reason for the failure.

The primary problem that promises solve is the **synchronization of asynchronous operations** without deeply nesting callbacks. With promises, developers can chain `.then()`, `.catch()`, and `.finally()` methods to handle asynchronous events in a sequential, readable manner.

2. What is an iterator in JavaScript, and what is its primary purpose?

An **iterator** in JavaScript is an object that allows sequential access to a collection of items, one at a time. It follows the **Iterator Protocol**, which requires an object to have a `next()` method that returns an object with two properties:

- `value`: The current value in the sequence.
- `done`: A boolean indicating whether the iteration is complete.

The primary purpose of an iterator is to provide a **standardized way to traverse data structures** (like arrays, maps, sets, or even custom objects) without needing to understand the internal structure. Iterators enable the use of the `for...of` loop, allowing developers to write cleaner, more abstract code for iterating over collections.

3. What is the purpose of the yield keyword in a generator function?

The **yield** keyword is used inside a generator function in JavaScript to pause the execution of the function and return a value to the caller. The generator function can then be resumed later, continuing its execution from where it left off.

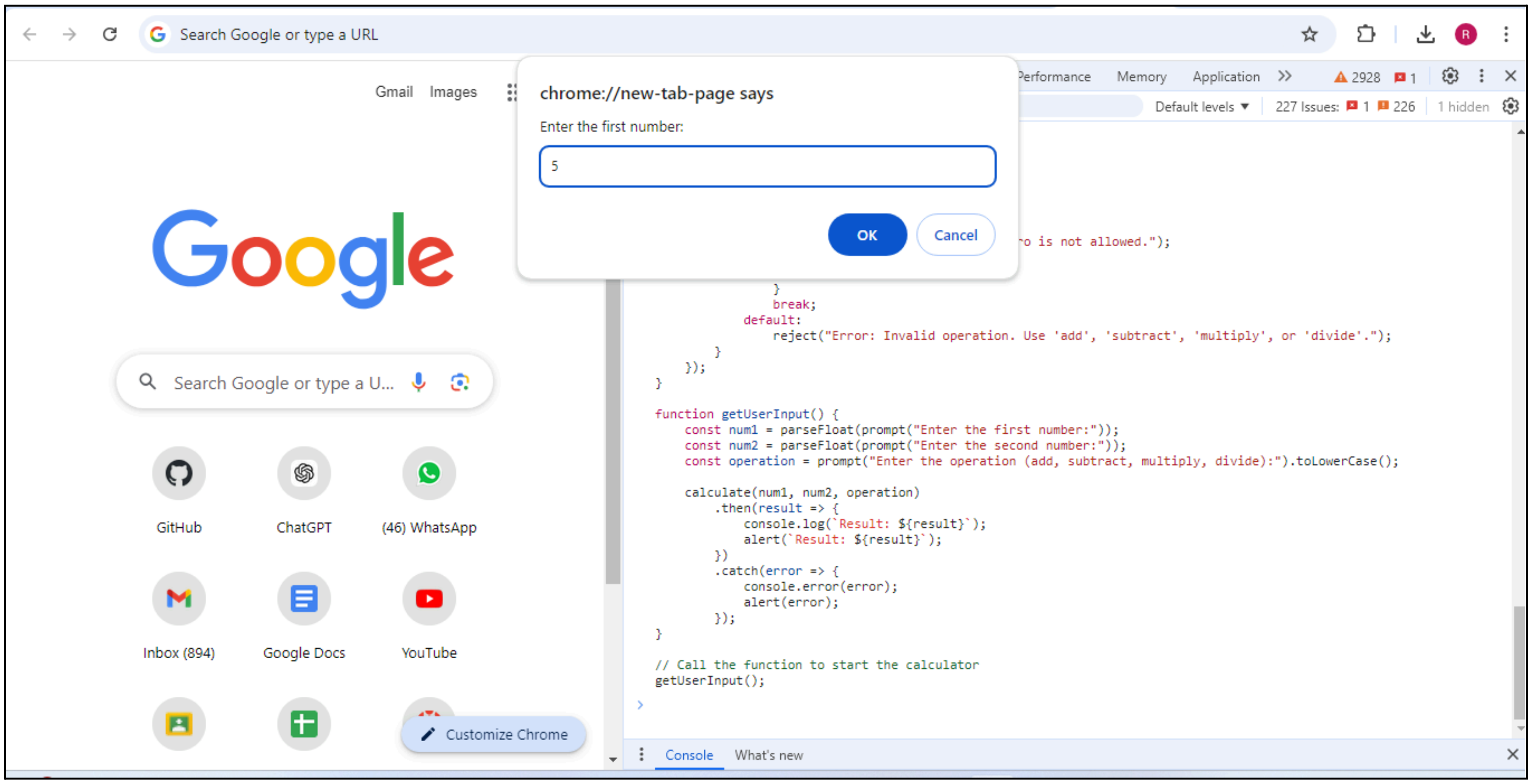
The purpose of `yield` is to allow the function to produce a series of intermediate results or values over time, rather than returning all results at once. This makes it particularly useful for creating **iterators** or handling asynchronous operations where multiple results may be produced in a sequence.

Generator functions are defined using `function*` syntax, and calling them returns an iterator. The `yield` keyword provides control over the iteration process, enabling a function to be paused and resumed.

Code with Screenshot:

Q.1 Code:
https://github.com/Rohan-Lalchandani08/IP-EXP5/blob/main/script1.js
Q.2 Code:
https://github.com/Rohan-Lalchandani08/IP-EXP5/blob/main/script2.js
Q.3 Code:
https://github.com/Rohan-Lalchandani08/IP-EXP5/blob/main/script3.js

Q.1 Screenshot:



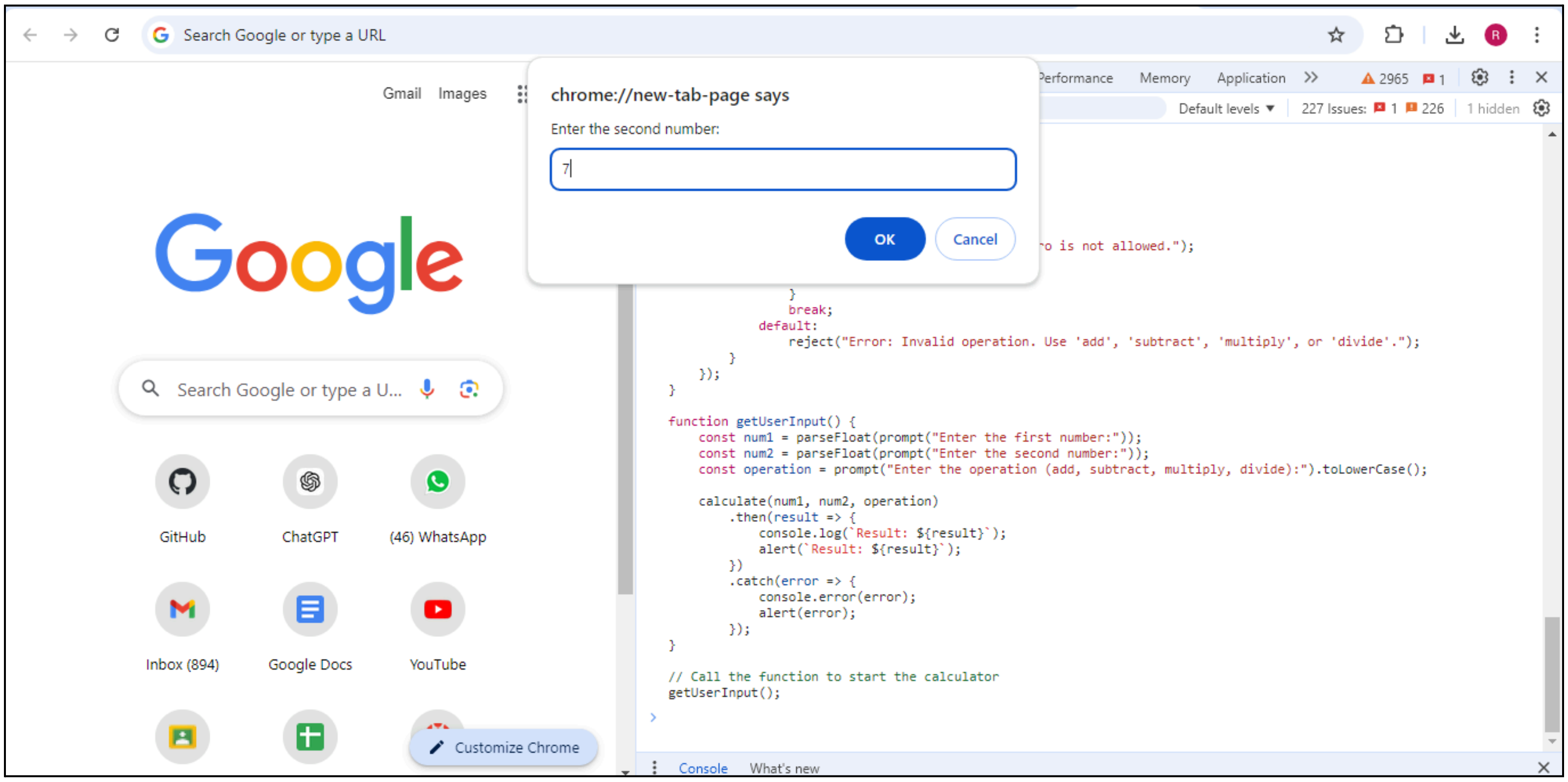
chrome://new-tab-page says

Enter the first number:

5

OK Cancel

```
                    o is not allowed.");
                    }
                    break;
                default:
                    reject("Error: Invalid operation. Use 'add', 'subtract', 'multiply', or 'divide'.");
            }
        });
}

function getUserInput() {
    const num1 = parseFloat(prompt("Enter the first number:"));
    const num2 = parseFloat(prompt("Enter the second number:"));
    const operation = prompt("Enter the operation (add, subtract, multiply, divide):").toLowerCase();

    calculate(num1, num2, operation)
        .then(result => {
            console.log(`Result: ${result}`);
            alert(`Result: ${result}`);
        })
        .catch(error => {
            console.error(error);
            alert(error);
        });
}

// Call the function to start the calculator
getUserInput();
```



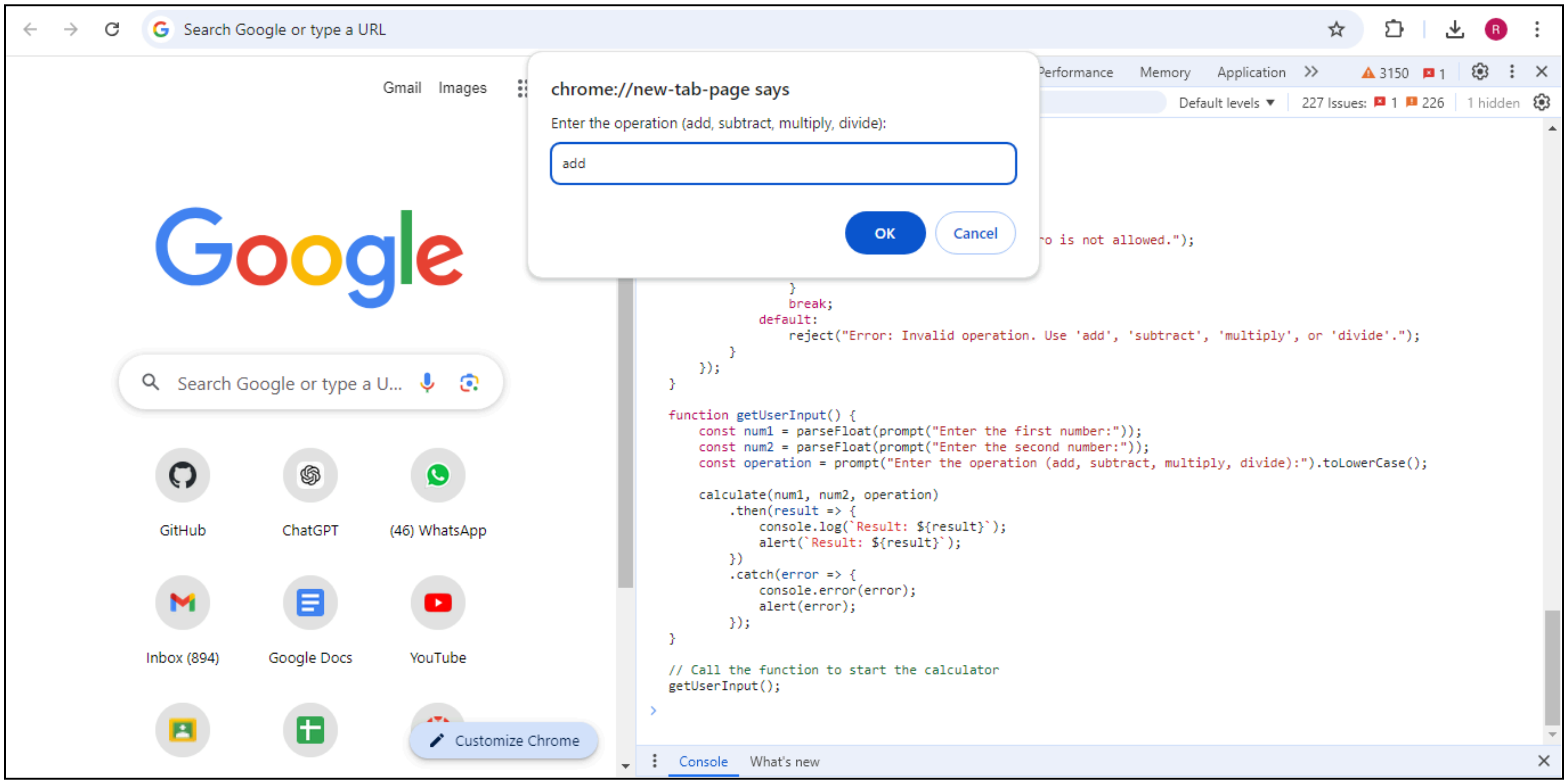
chrome://new-tab-page says

Enter the second number:

7

OK Cancel

```
                    o is not allowed.");
                    }
                    break;
                default:
                    reject("Error: Invalid operation. Use 'add', 'subtract', 'multiply', or 'divide'.");
            }
        });
}

function getUserInput() {
    const num1 = parseFloat(prompt("Enter the first number:"));
    const num2 = parseFloat(prompt("Enter the second number:"));
    const operation = prompt("Enter the operation (add, subtract, multiply, divide):").toLowerCase();

    calculate(num1, num2, operation)
        .then(result => {
            console.log(`Result: ${result}`);
            alert(`Result: ${result}`);
        })
        .catch(error => {
            console.error(error);
            alert(error);
        });
}

// Call the function to start the calculator
getUserInput();
```



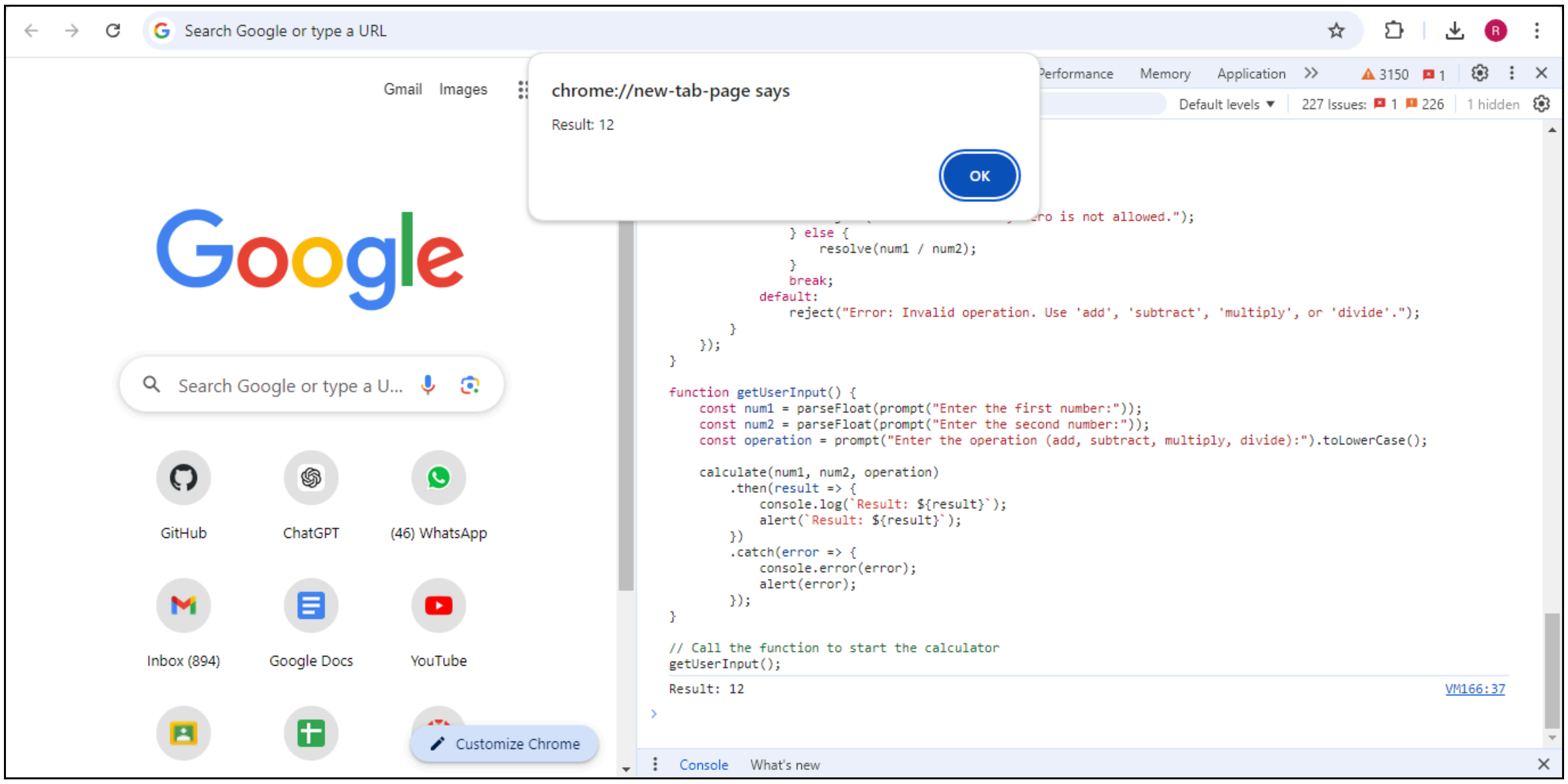
chrome://new-tab-page says

Enter the operation (add, subtract, multiply, divide):

add

OK Cancel

```
                                                  ro is not allowed.");
                }
                break;
            default:
                reject("Error: Invalid operation. Use 'add', 'subtract', 'multiply', or 'divide'.");
        }
    });
}

function getUserInput() {
    const num1 = parseFloat(prompt("Enter the first number:"));
    const num2 = parseFloat(prompt("Enter the second number:"));
    const operation = prompt("Enter the operation (add, subtract, multiply, divide):").toLowerCase();

    calculate(num1, num2, operation)
        .then(result => {
            console.log(`Result: ${result}`);
            alert(`Result: ${result}`);
        })
        .catch(error => {
            console.error(error);
            alert(error);
        });
}

// Call the function to start the calculator
getUserInput();
>
```



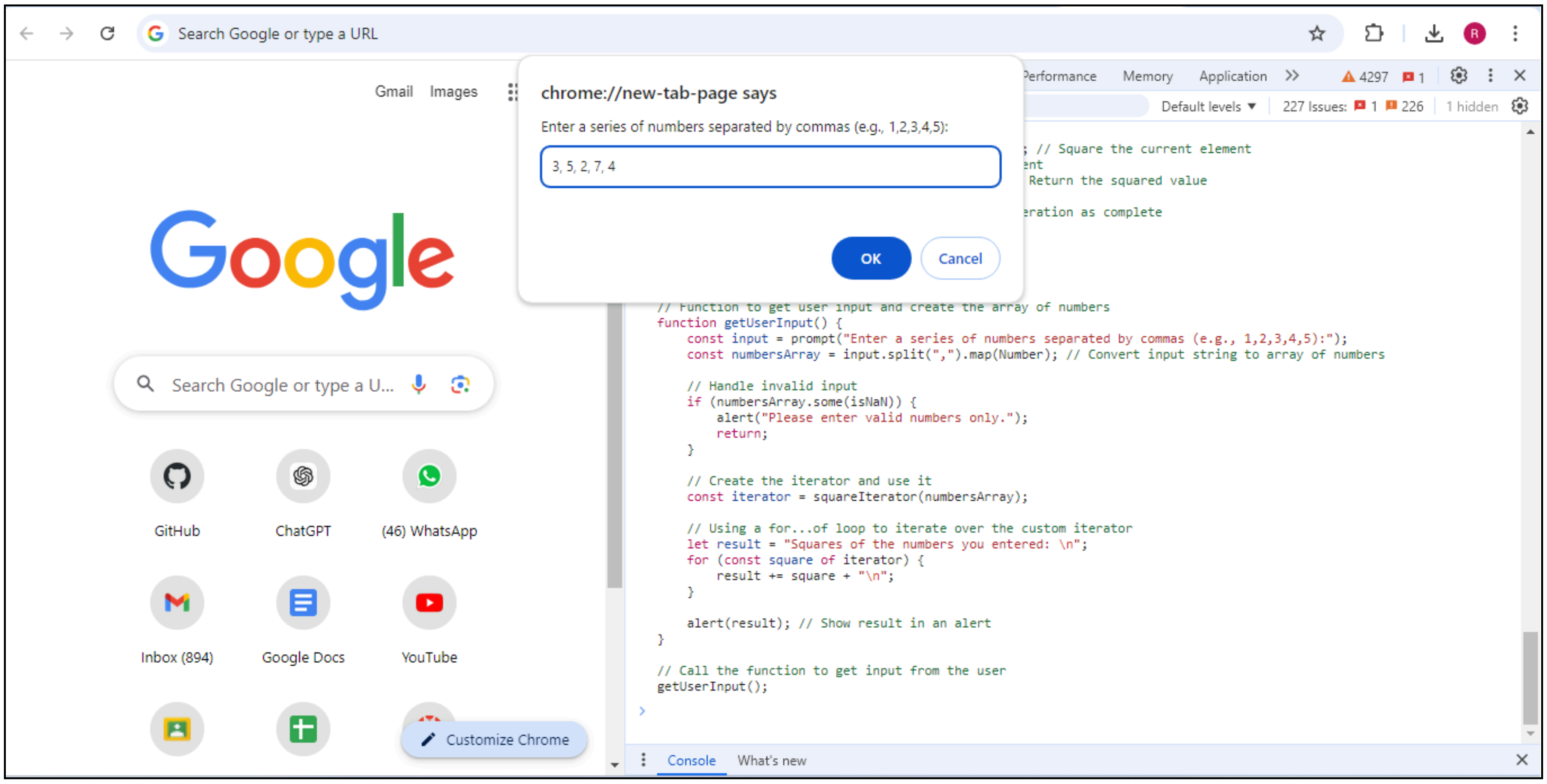
chrome://new-tab-page says

Result: 12

OK

```
                                                  ro is not allowed.");
                } else {
                    resolve(num1 / num2);
                }
                break;
            default:
                reject("Error: Invalid operation. Use 'add', 'subtract', 'multiply', or 'divide'.");
        }
    });
}

function getUserInput() {
    const num1 = parseFloat(prompt("Enter the first number:"));
    const num2 = parseFloat(prompt("Enter the second number:"));
    const operation = prompt("Enter the operation (add, subtract, multiply, divide):").toLowerCase();

    calculate(num1, num2, operation)
        .then(result => {
            console.log(`Result: ${result}`);
            alert(`Result: ${result}`);
        })
        .catch(error => {
            console.error(error);
            alert(error);
        });
}

// Call the function to start the calculator
getUserInput();
Result: 12                                                                       VM166:37
>
```

Q.2 Screenshot:



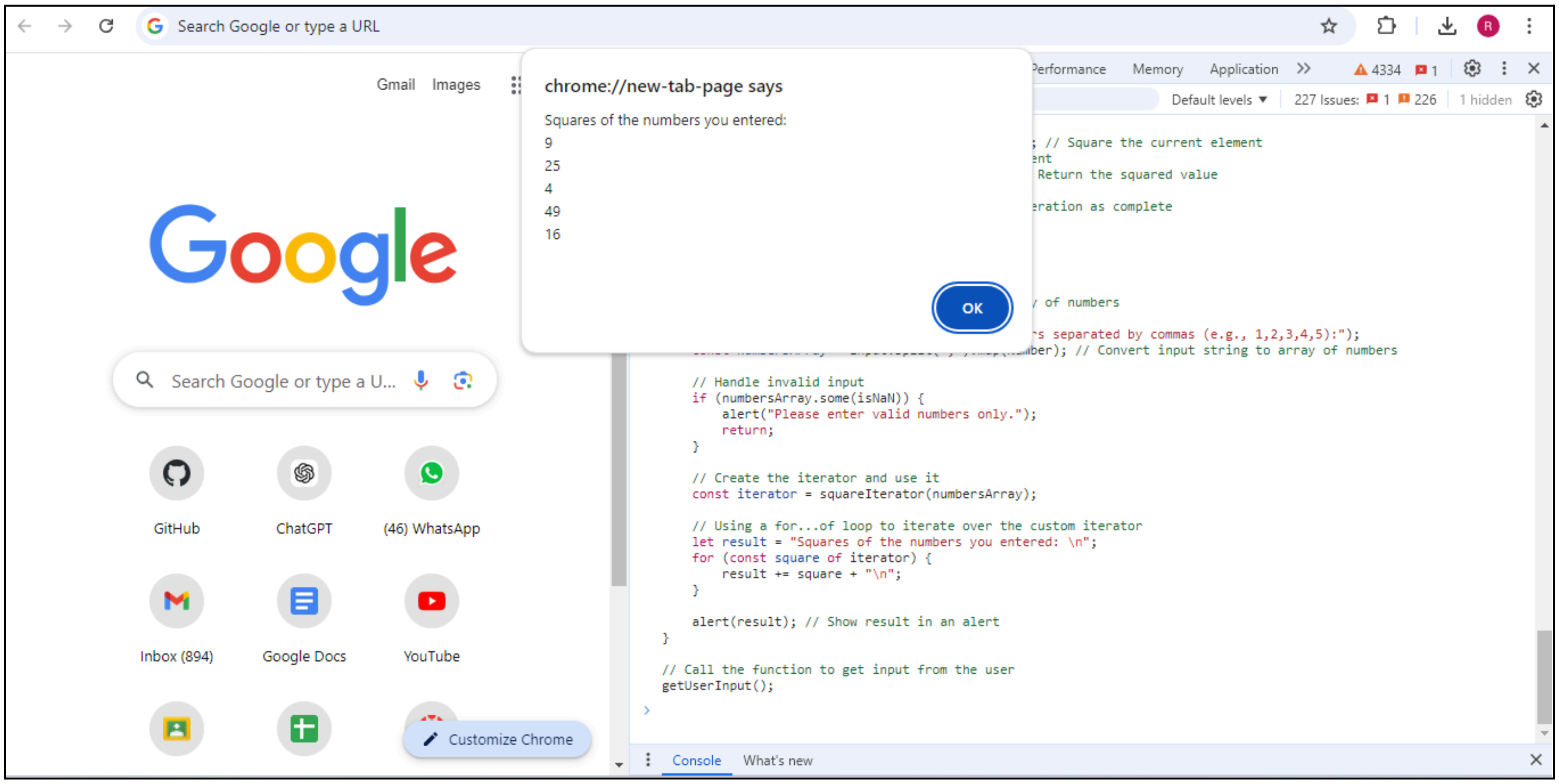
chrome://new-tab-page says

Enter a series of numbers separated by commas (e.g., 1,2,3,4,5):

3, 5, 2, 7, 4

OK Cancel

```
; // Square the current element
ent
 Return the squared value

eration as complete

// Function to get user input and create the array of numbers
function getUserInput() {
    const input = prompt("Enter a series of numbers separated by commas (e.g., 1,2,3,4,5):");
    const numbersArray = input.split(",").map(Number); // Convert input string to array of numbers

    // Handle invalid input
    if (numbersArray.some(isNaN)) {
        alert("Please enter valid numbers only.");
        return;
    }

    // Create the iterator and use it
    const iterator = squareIterator(numbersArray);

    // Using a for...of loop to iterate over the custom iterator
    let result = "Squares of the numbers you entered: \n";
    for (const square of iterator) {
        result += square + "\n";
    }

    alert(result); // Show result in an alert
}

// Call the function to get input from the user
getUserInput();
```



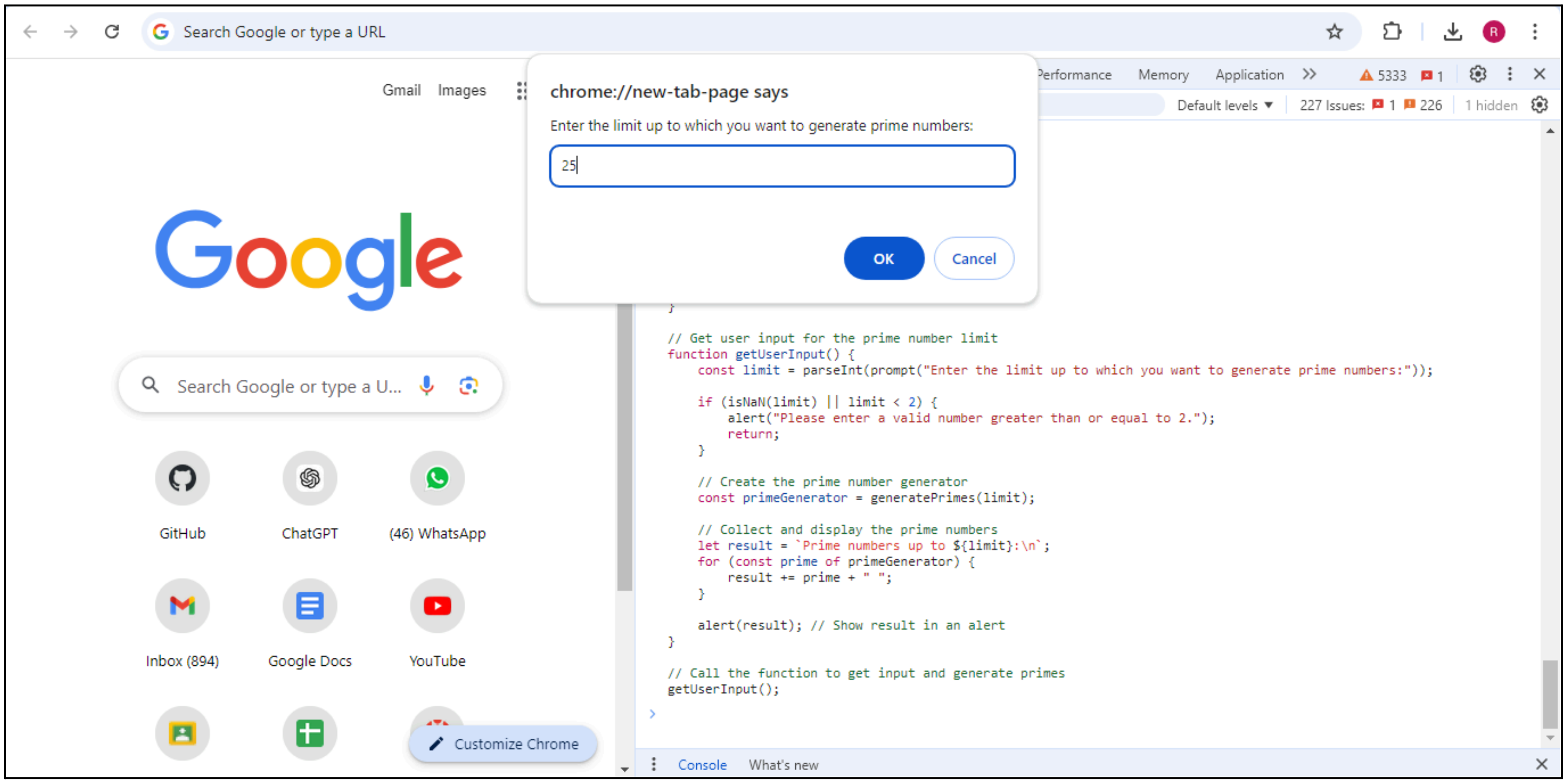
chrome://new-tab-page says

Squares of the numbers you entered:
9
25
4
49
16

OK

```
; // Square the current element
ent
 Return the squared value

eration as complete

/ of numbers
s separated by commas (e.g., 1,2,3,4,5):");
ber); // Convert input string to array of numbers

    // Handle invalid input
    if (numbersArray.some(isNaN)) {
        alert("Please enter valid numbers only.");
        return;
    }

    // Create the iterator and use it
    const iterator = squareIterator(numbersArray);

    // Using a for...of loop to iterate over the custom iterator
    let result = "Squares of the numbers you entered: \n";
    for (const square of iterator) {
        result += square + "\n";
    }

    alert(result); // Show result in an alert
}

// Call the function to get input from the user
getUserInput();
```

Q. 3 Screenshot

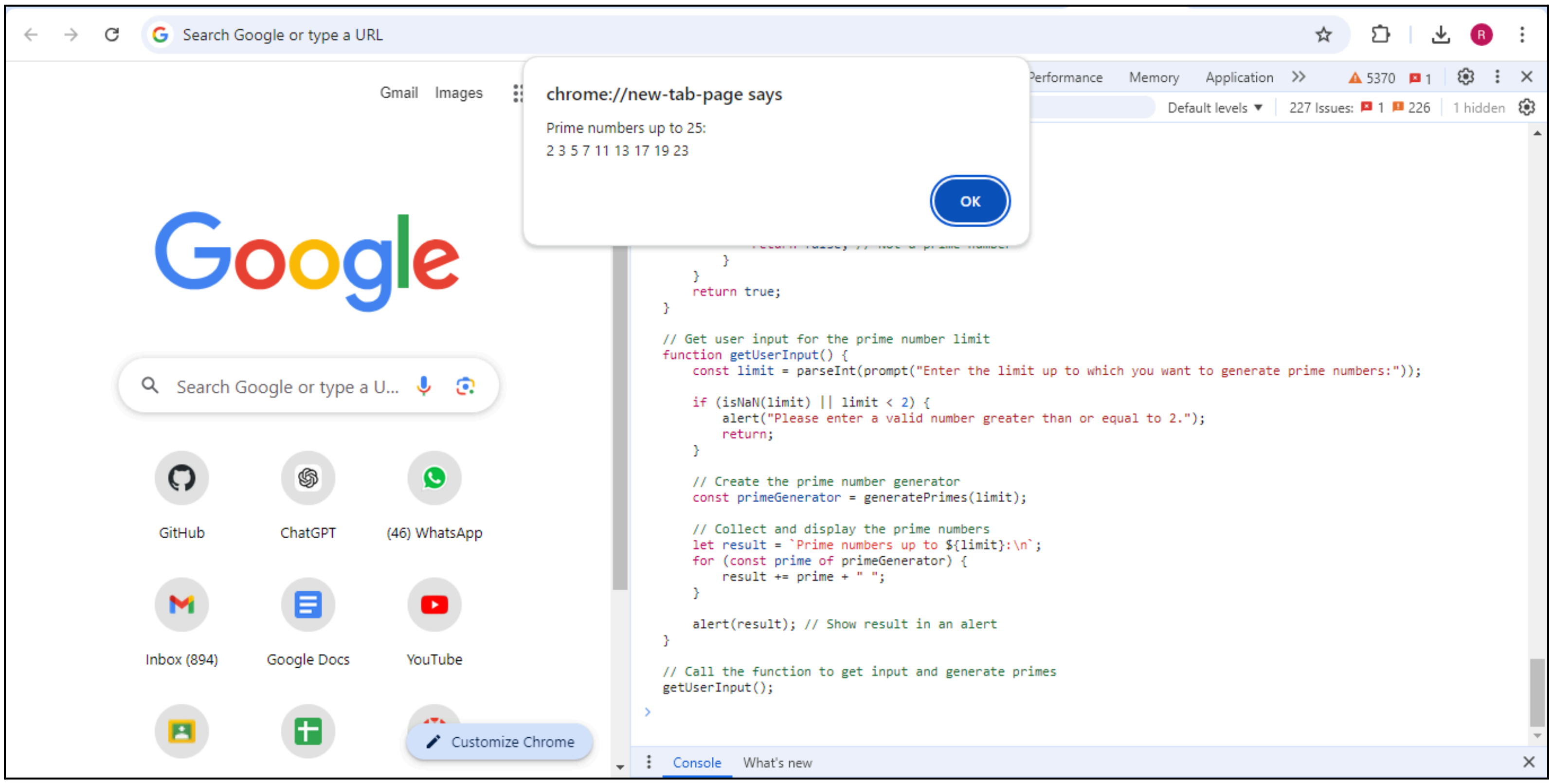


chrome://new-tab-page says

Enter the limit up to which you want to generate prime numbers:

25

```
}

// Get user input for the prime number limit
function getUserInput() {
    const limit = parseInt(prompt("Enter the limit up to which you want to generate prime numbers:"));

    if (isNaN(limit) || limit < 2) {
        alert("Please enter a valid number greater than or equal to 2.");
        return;
    }

    // Create the prime number generator
    const primeGenerator = generatePrimes(limit);

    // Collect and display the prime numbers
    let result = `Prime numbers up to ${limit}:\n`;
    for (const prime of primeGenerator) {
        result += prime + " ";
    }

    alert(result); // Show result in an alert
}

// Call the function to get input and generate primes
getUserInput();
```



chrome://new-tab-page says

Prime numbers up to 25:
2 3 5 7 11 13 17 19 23

```
            }
        }
        return true;
    }

// Get user input for the prime number limit
function getUserInput() {
    const limit = parseInt(prompt("Enter the limit up to which you want to generate prime numbers:"));

    if (isNaN(limit) || limit < 2) {
        alert("Please enter a valid number greater than or equal to 2.");
        return;
    }

    // Create the prime number generator
    const primeGenerator = generatePrimes(limit);

    // Collect and display the prime numbers
    let result = `Prime numbers up to ${limit}:\n`;
    for (const prime of primeGenerator) {
        result += prime + " ";
    }

    alert(result); // Show result in an alert
}

// Call the function to get input and generate primes
getUserInput();
```