NAME:SHREYA SREE PV

DEPT: EEE-“B”

ROLL NO:717823E253

MERN STACK TRAINING

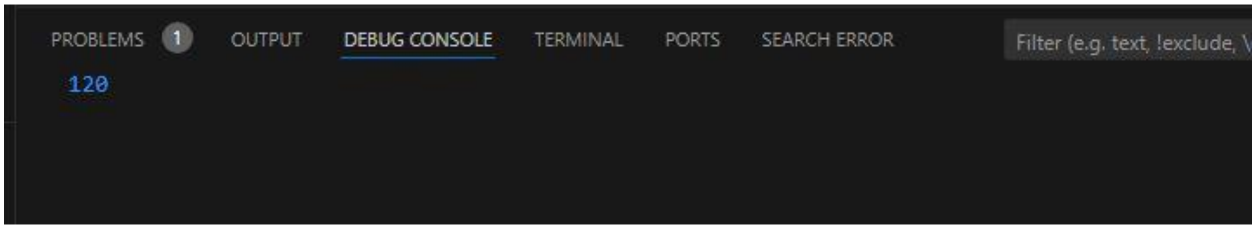
TASK(1-35)

1. Recursion and stack

1.1Task 1: Implement a function to calculate the factorial of a number using recursion.

```
<!DOCTYPE html>
<html>
    <head></head>
    <title>webpage</title>
    <body>
        <script>
        function fun(n){
            if(n===0||n===1){
                return 1;
            }
            else{
            return n*fun(n-1);
            }
        }
        console.log(fun(5));
        </script>
          </body>
          </html>
```

OUTPUT:

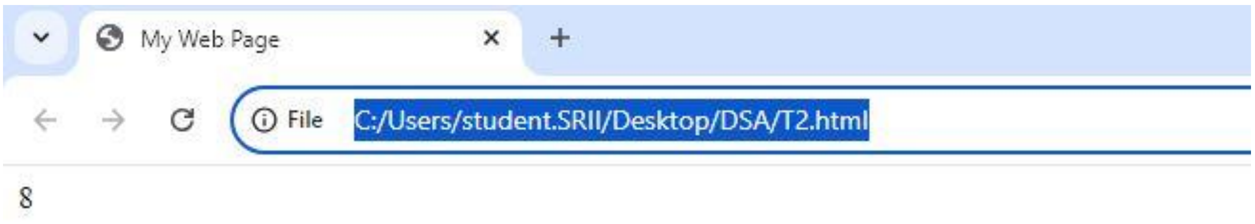


```
120
```

1.2 Task 2: Write a recursive function to find the nth Fibonacci number.

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            My Web Page
        </title>
    </head>
    <body>
        <script>
           function fib (n){
            if(n === 0){
                return 0;
            }else if(n === 1){
                return 1;
            }else{
               return fib(n-1) + fib(n-2);
            }
           }
           document.writeln(fib(6))
        </script>
    </body>
</html>
```

OUTPUT:

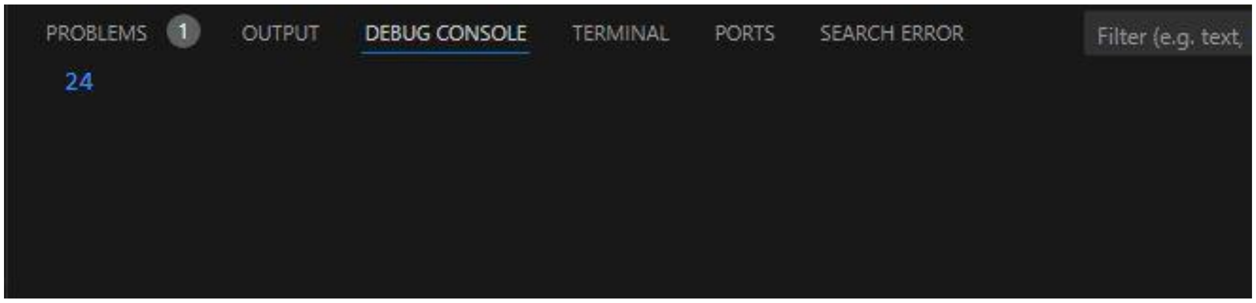


```
8
```

1.3. Task 3: Create a function to determine the total number of ways one can climb a staircase with 1, 2, or 3 steps at a time using recursion.

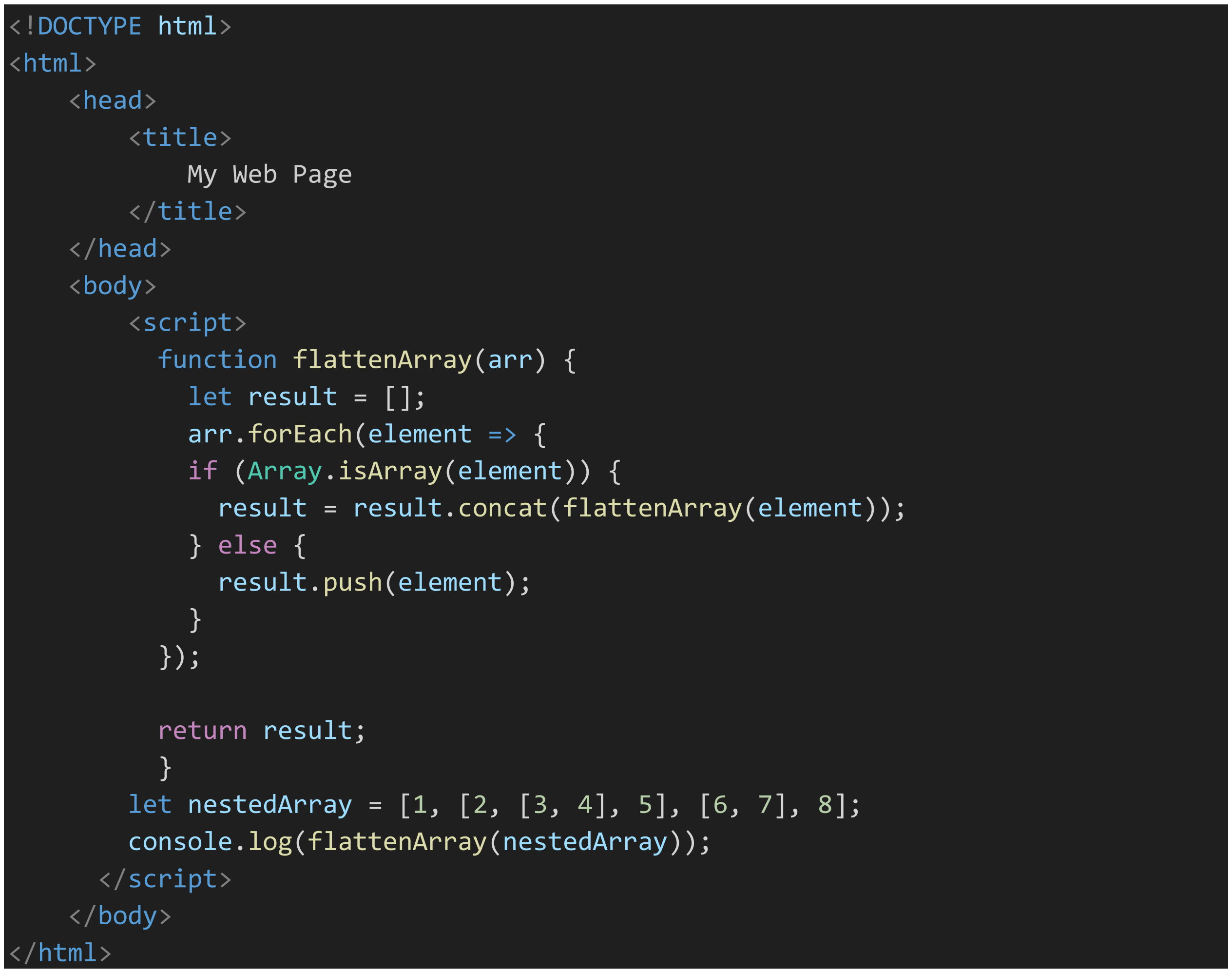
```
<!DOCTYPE html>
<html>
    <head>
        <title>Climbing Stairs</title>
    </head>
    <body>
        <script>
            function climb(n) {

                if (n === 0||n===1) return 1;

                if (n === 2) return 2;


                return climb(n - 1) + climb(n - 2) + climb(n - 3);
            }

            let result = climb(6);
            console.log(result);

        </script>
    </body>
</html>
```

OUTPUT:



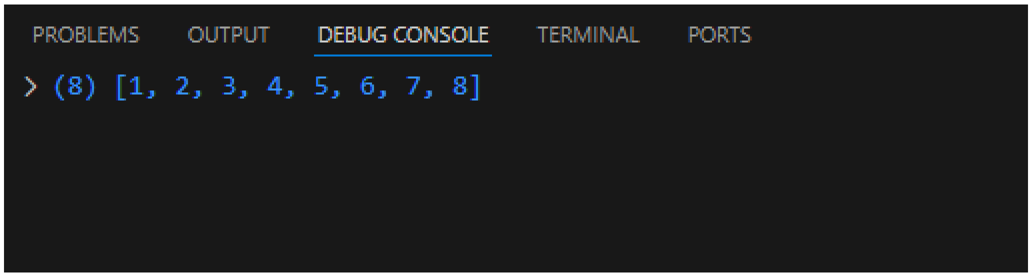
```
24
```

1.4. Task 4: Write a recursive function to flatten a nested array structure.



```
<!DOCTYPE html>
<html>
    <head>
        <title>
            My Web Page
        </title>
    </head>
    <body>
        <script>
          function flattenArray(arr) {
            let result = [];
            arr.forEach(element => {
            if (Array.isArray(element)) {
              result = result.concat(flattenArray(element));
            } else {
              result.push(element);
            }
          });

          return result;
          }
        let nestedArray = [1, [2, [3, 4], 5], [6, 7], 8];
        console.log(flattenArray(nestedArray));
      </script>
    </body>
</html>
```

OUTPUT:

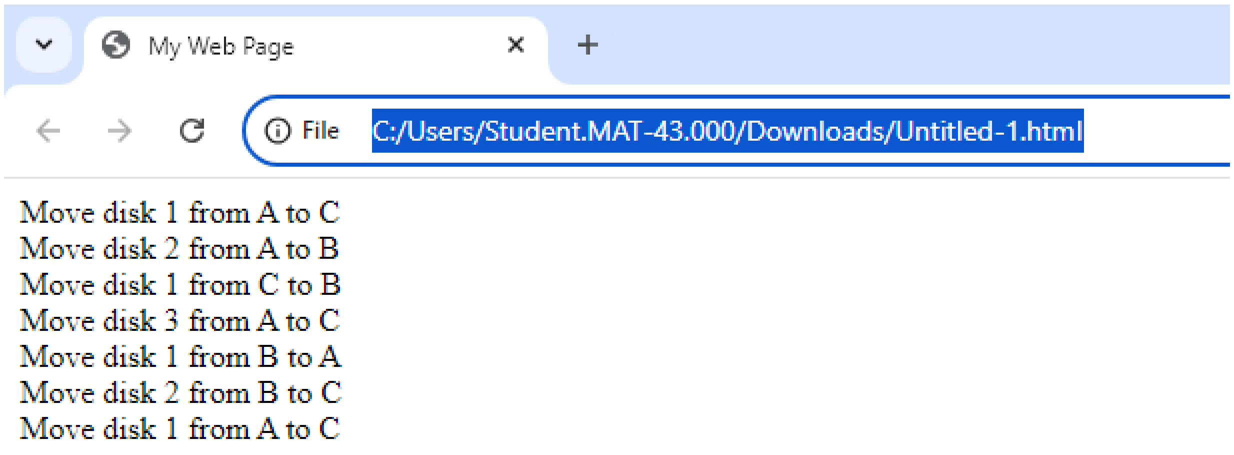


```
> (8) [1, 2, 3, 4, 5, 6, 7, 8]
```

1.5. Task 5: Implement the recursive Tower of Hanoi solution

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            My Web Page
        </title>
    </head>
    <body>
        <script>
            function towerOfHanoi(n, source, destination, auxiliary) {
                if (n === 1) {
                    document.writeln(`Move disk 1 from ${source} to
${destination}`, "<br>");
                    return;
            }
            towerOfHanoi(n - 1, source, auxiliary, destination);
            document.writeln(`Move disk ${n} from ${source} to ${destination}`,
"<br>");
            towerOfHanoi(n - 1, auxiliary, destination, source);
            }
            const a = 3;
            towerOfHanoi(a, 'A', 'C', 'B');
        </script>
    </body>
</html>
```

OUTPUT:



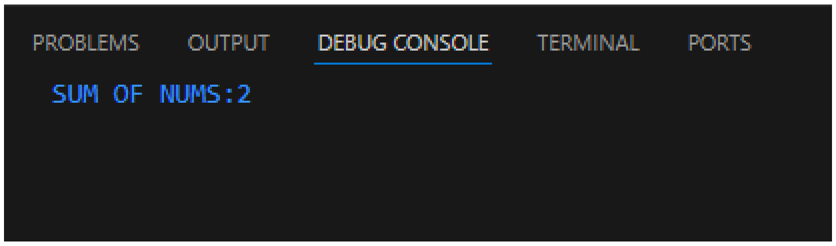
```
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

2. JSON and variable length arguments/spread syntax

2.1 Task 1: Write a function that takes an arbitrary number of arguments and returns their sum.

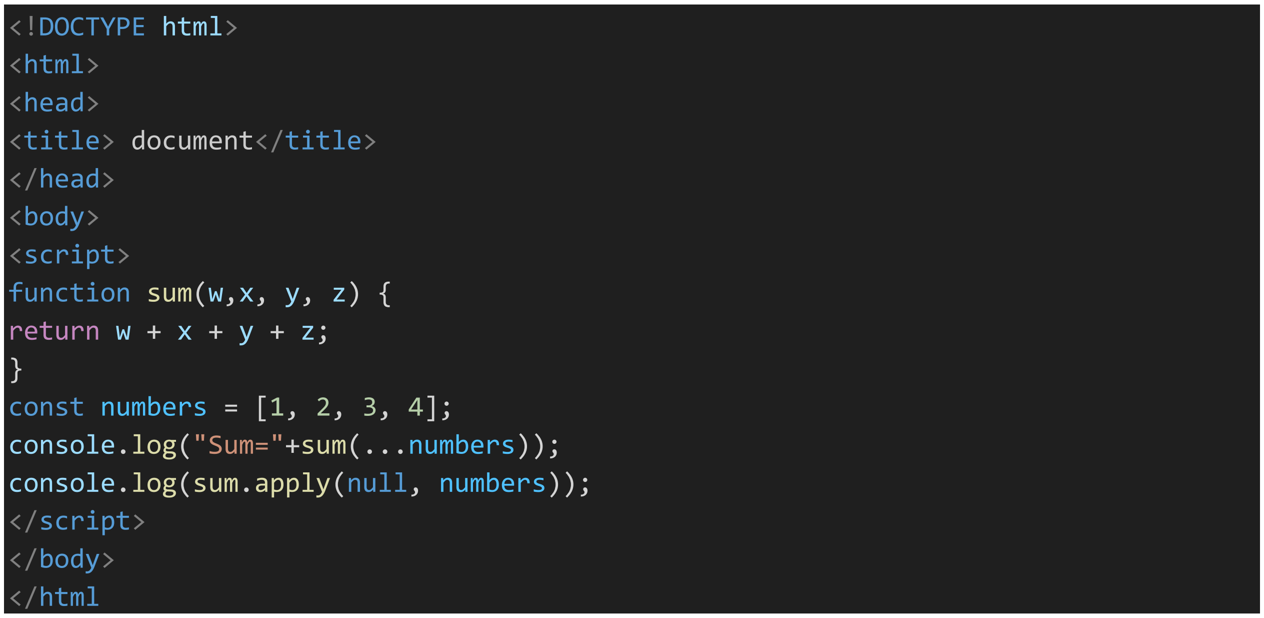
```
<!DOCTYPE html>
<html>
<head>
<title> document</title>
</head>
<body>
<script>
function sumAll(... args) {
let sum = 0;
for (let arg of args)
sum += arg;
return sum;
}
const numbers = [1, -2, 3];
console.log("SUM OF NUMS:"+sumAll(...numbers));
</script>
</body>
</html>
```

OUTPUT:



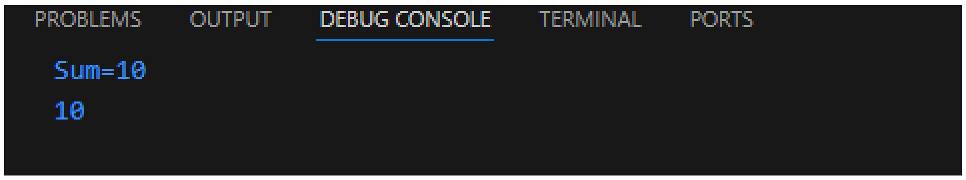
```
SUM OF NUMS:2
```

2.2 Task 2: Modify a function to accept an array of numbers and return their sum using the spread syntax.



```
<!DOCTYPE html>
<html>
<head>
<title> document</title>
</head>
<body>
<script>
function sum(w,x, y, z) {
return w + x + y + z;
}
const numbers = [1, 2, 3, 4];
console.log("Sum="+sum(...numbers));
console.log(sum.apply(null, numbers));
</script>
</body>
</html
```

OUTPUT:

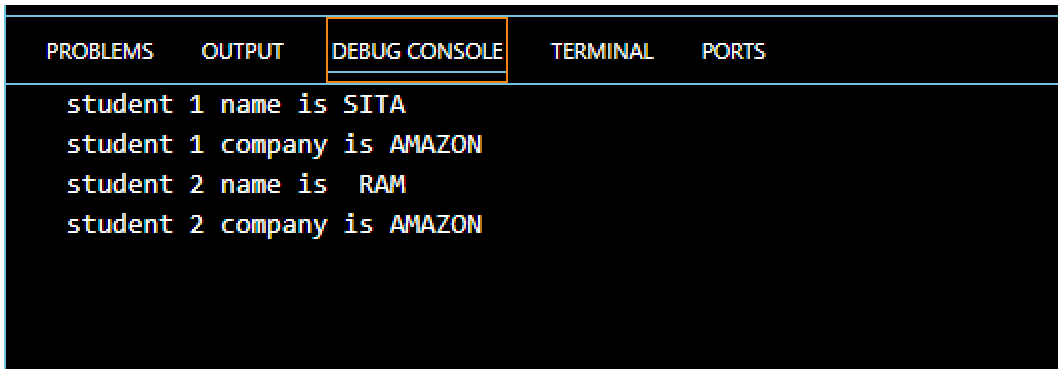


```
Sum=10
10
```

2.3 Task 3: Create a deep clone of an object using JSON methods.

```html
<!DOCTYPE html>
<html >
<head>
<title>My webpage</title>
</head>
<body>
<script>
let student1 = {
name: "RAM",
company: "AMAZON"
}
let student2 = { ...student1 };
student1.name = "SITA"
console.log("student 1 name is", student1.name);
console.log("student 1 company is", student1.company);
console.log("student 2 name is ", student2.name);
console.log("student 2 company is", student2.company);
</script>
</body>
</html>
```

OUTPUT:



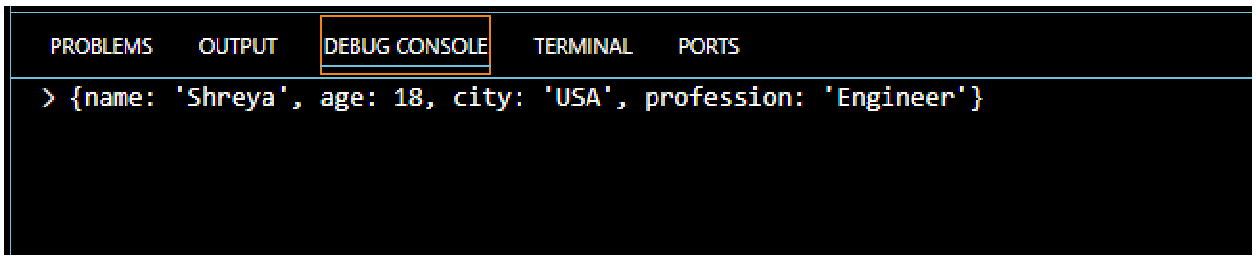
```
student 1 name is SITA
student 1 company is AMAZON
student 2 name is  RAM
student 2 company is AMAZON
```

2.4 Task 4: Write a function that returns a new object, merging two provided objects using the spread syntax

```html
<!DOCTYPE html>
<html>
<head>
<title> document</title>
</head>
<body>
```

```
<script>
function mergeObj(obj1, obj2) {
return { ...obj1, ...obj2 };
}
const obj1 = { name: "Shreya", age: 18 };
const obj2 = { city: "USA", profession: "Engineer" };
const merged1 = mergeObj(obj1, obj2);
console.log(merged1);
</script>
</body>
</html>
```

OUTPUT:

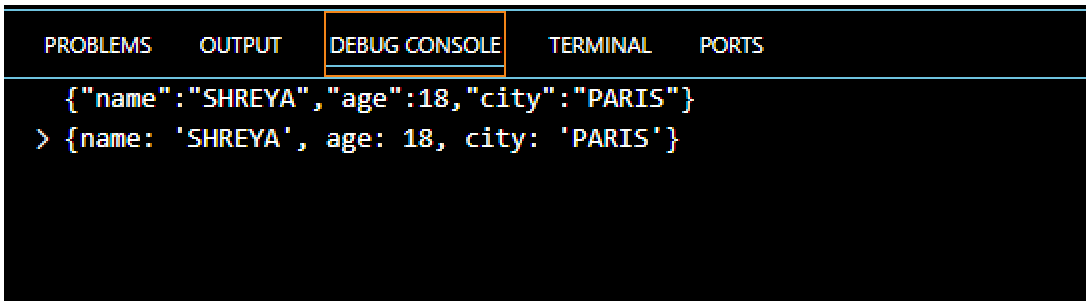


```
> {name: 'Shreya', age: 18, city: 'USA', profession: 'Engineer'}
```

2.5Task 5: Serialize a JavaScript object into a JSON string and then parse it back into an object.

```
<!DOCTYPE html>
<html>
<head>
<title> document</title>
</head>
<body>
<script>
const person = { name: "SHREYA", age: 18, city: "PARIS" };
const jsonStr= JSON.stringify(person);
console.log(jsonStr);
const parsedObj = JSON.parse(jsonStr);
console.log(parsedObj);
</script>
</body>
</html>
```

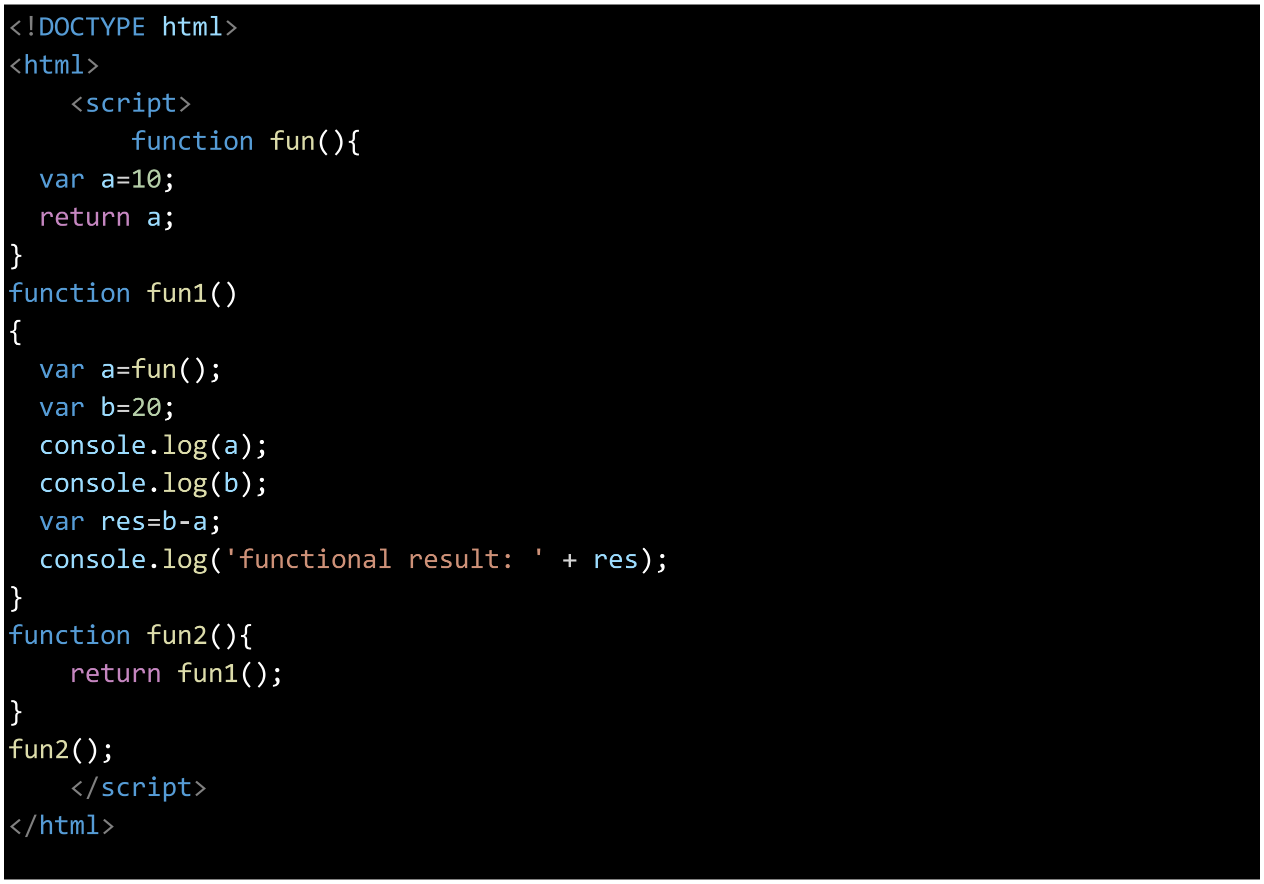
OUTPUT:



```
  {"name":"SHREYA","age":18,"city":"PARIS"}
> {name: 'SHREYA', age: 18, city: 'PARIS'}
```

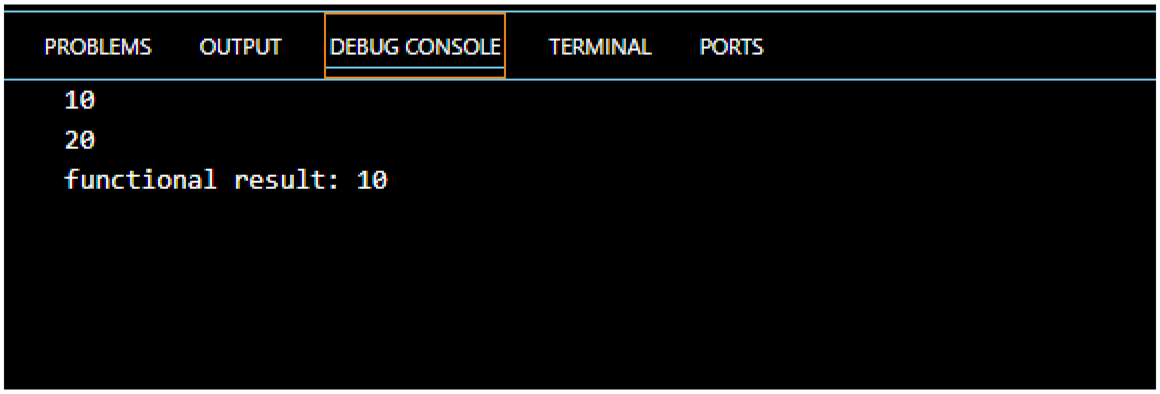
3. Closure

3.1Task 1: Create a function that returns another function, capturing a local variable.



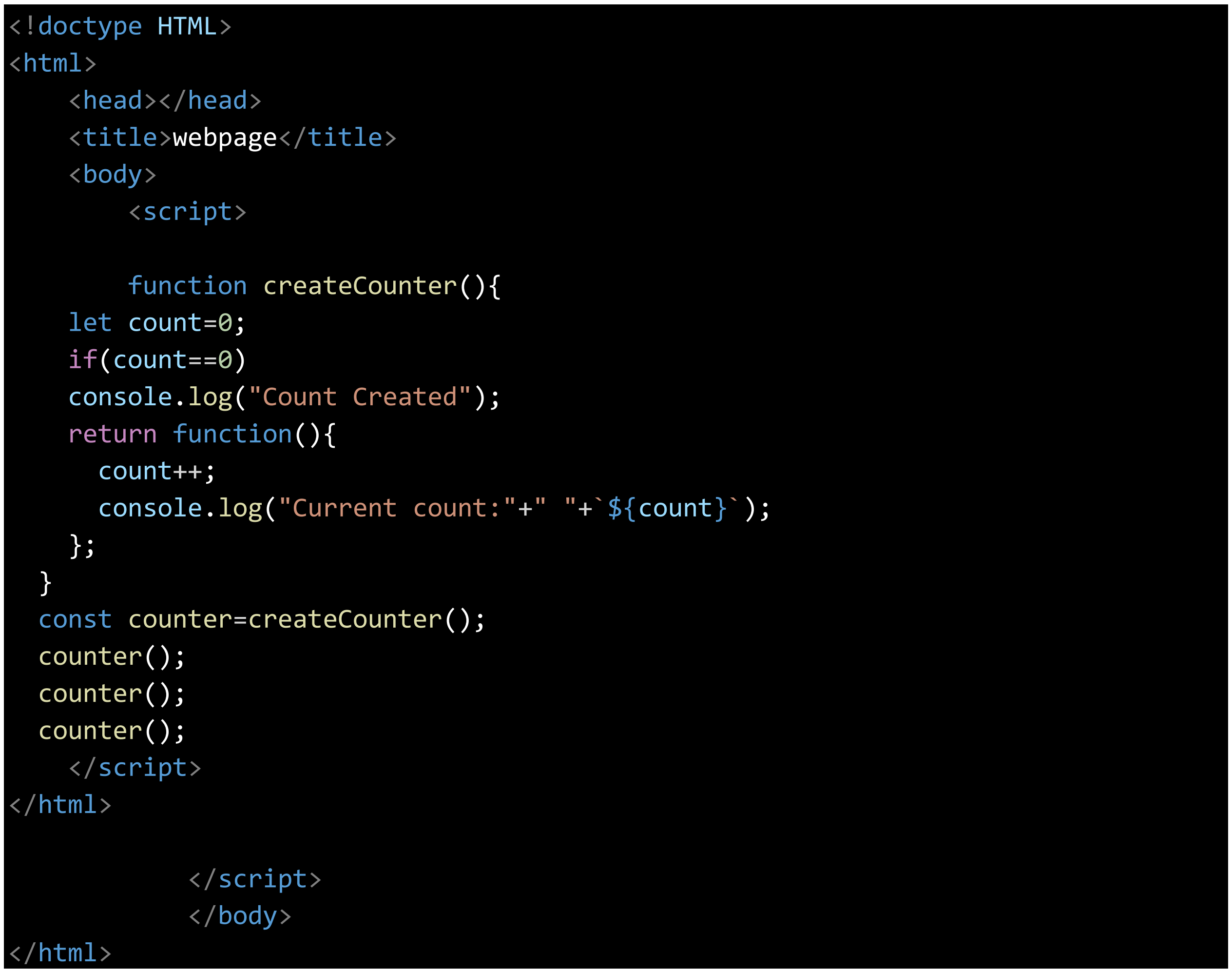
```
<!DOCTYPE html>
<html>
    <script>
        function fun(){
  var a=10;
  return a;
}
function fun1()
{
  var a=fun();
  var b=20;
  console.log(a);
  console.log(b);
  var res=b-a;
  console.log('functional result: ' + res);
}
function fun2(){
    return fun1();
}
fun2();
    </script>
</html>
```

OUTPUT:



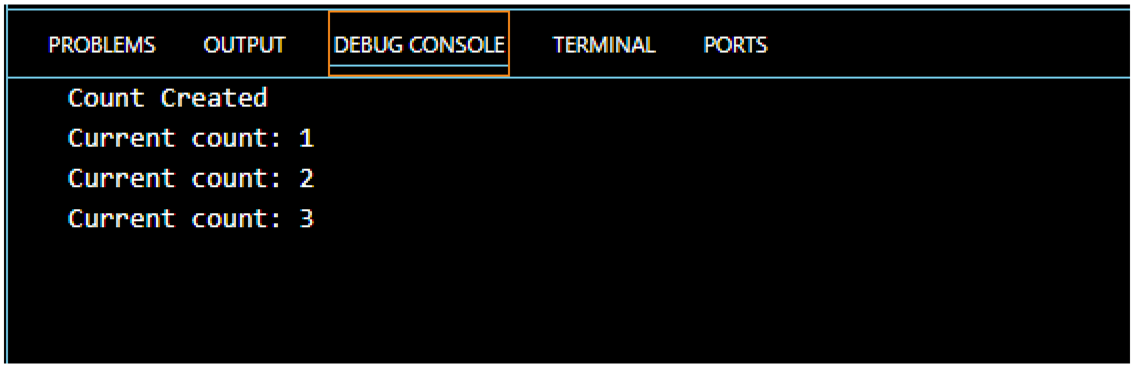
```
10
20
functional result: 10
```

3.2Task 2: Implement a basic counter function using closure, allowing incrementing and displaying the current count.



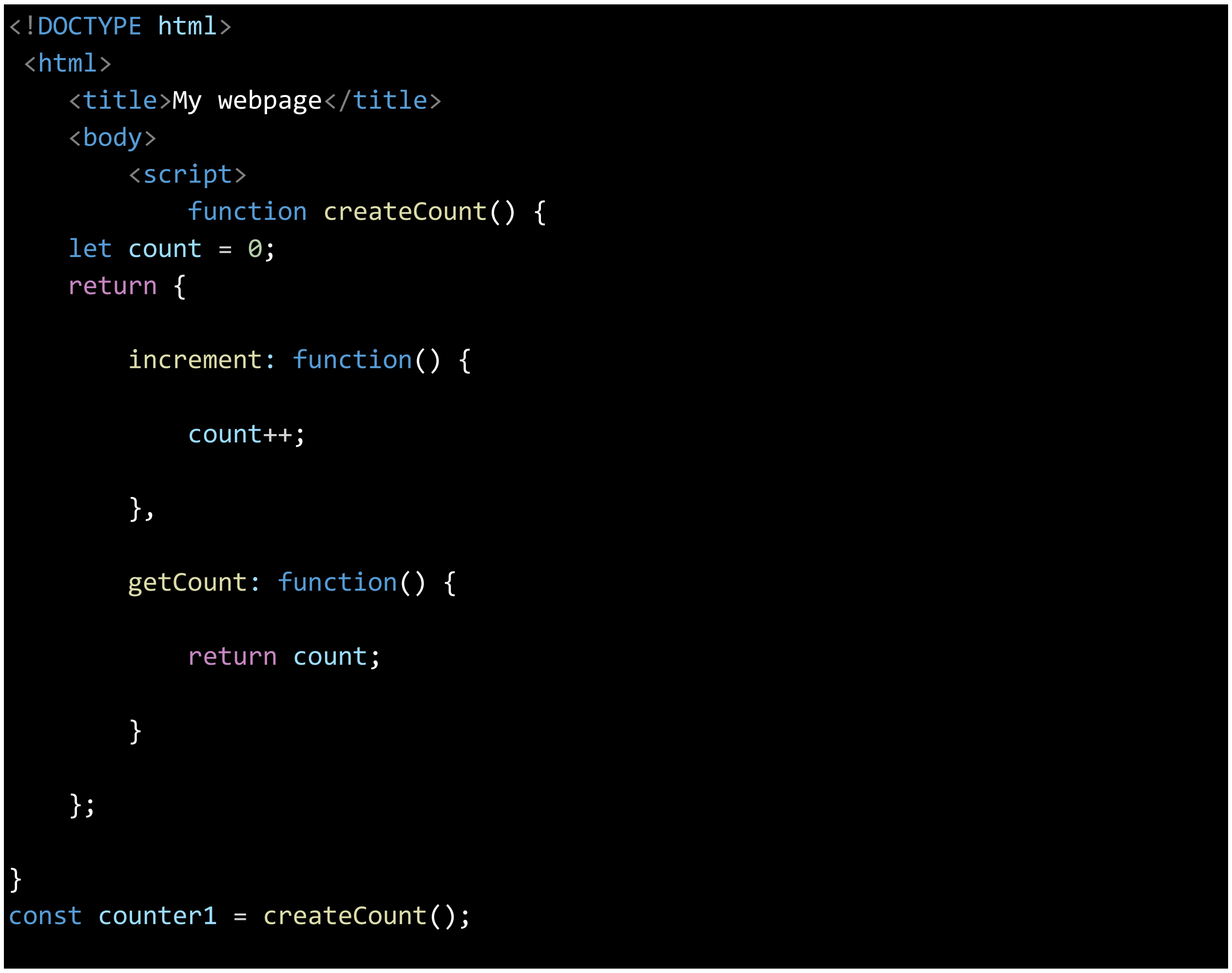
```
<!doctype HTML>
<html>
    <head></head>
    <title>webpage</title>
    <body>
        <script>

        function createCounter(){
    let count=0;
    if(count==0)
    console.log("Count Created");
    return function(){
      count++;
      console.log("Current count:"+" "+`${count}`);
    };
  }
  const counter=createCounter();
  counter();
  counter();
  counter();
    </script>
</html>

            </script>
            </body>
</html>
```

OUTPUT:



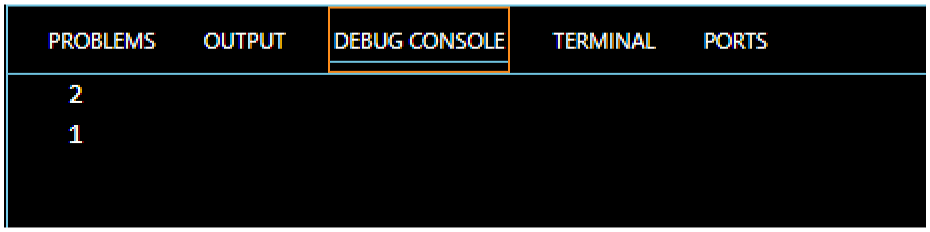
```
Count Created
Current count: 1
Current count: 2
Current count: 3
```

3.3 Task 3: Write a function to create multiple counters, each with its own separate count.



```
<!DOCTYPE html>
 <html>
    <title>My webpage</title>
    <body>
        <script>
            function createCount() {
    let count = 0;
    return {

        increment: function() {

            count++;

        },

        getCount: function() {

            return count;

        }

    };

}
const counter1 = createCount();
const counter2 = createCount();

counter1.increment();

counter1.increment();

console.log(counter1.getCount());

counter2.increment();

console.log(counter2.getCount());

        </script>

    </body>

 </html>
```

OUTPUT:

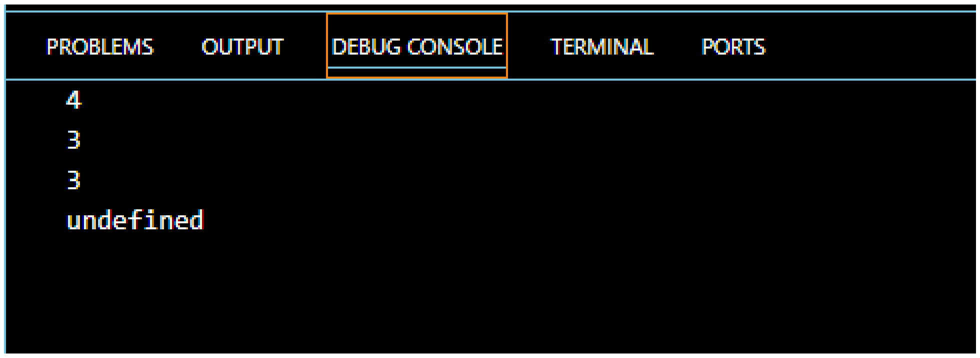


```
2
1
```

3.4Task 4: Use closures to create private variables within a function.

```
<!DOCTYPE html>
<html>
    <script>
        function Count(){
            let c=3;
            return {
                increment:function(){
                    c++;
                    return c; },
                decrement:function(){
                    c--;
                    return c; },
                getCount:function(){
                    return c; }}
        }
        let count=Count();
        console.log(count.increment());
        console.log(count.decrement());
        console.log(count.getCount());
        console.log(count.c);
    </script>
</html>
```

OUTPUT:

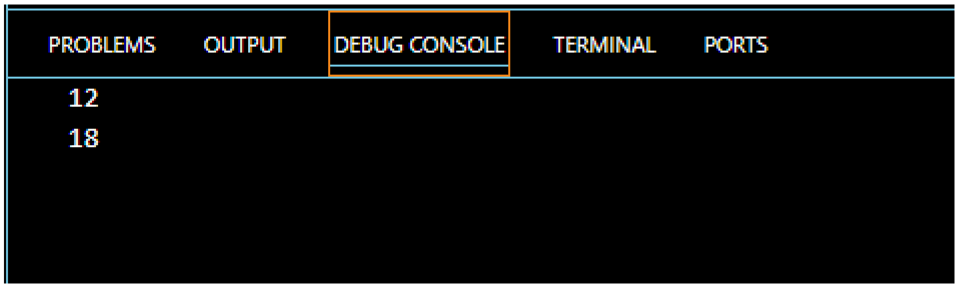


```
4
3
3
undefined
```

3.5Task 5: Build a function factory that generates functions based on some input using closures.

```
<!DOCTYPE html>
 <html>
    <title>My webpage</title>
    <body>
        <script>
           function factory(factor) {
    return function(number) {
        return number * factor;

    };

}
const double = factory(4);
const triple = factory(6);

console.log(double(3));

console.log(triple(3));

        </script>

    </body>

 </html>
```

OUTPUT:



```
12
18
```

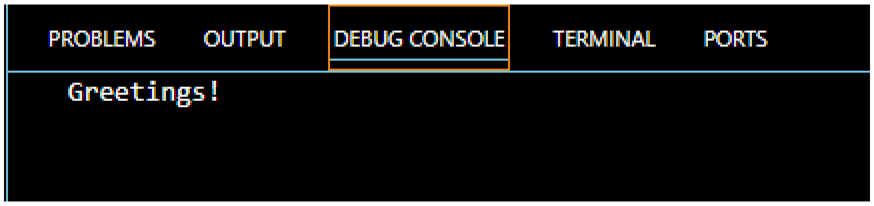
4. Promise, Promises chaining

4.1.Task 1: Create a new promise that resolves after a set number of seconds and returns a greeting.

```
<!DOCTYPE html>
<html>
    <script>
      function myPromise(){
        return new Promise((resolve)=>{
            setTimeout(()=>{
                resolve();
               console.log("Greetings!");
            },
            3000); })
 }
      myPromise();
    </script>
```

```
</html>
```

OUTPUT:

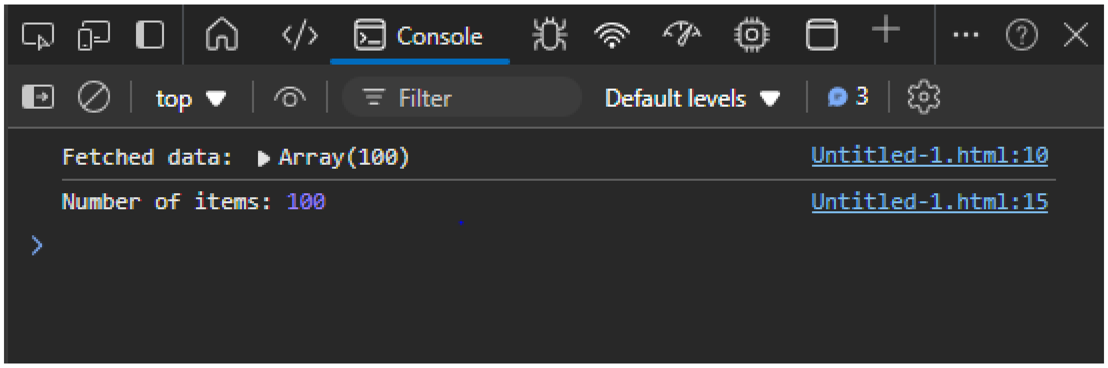


```
Greetings!
```

4.2. Task 2: Fetch data from an API using promises, and then chain another promise to process this data.

```
<html><head>
<title>My Webpage</title>
</head>
<body>
<script>
function fetchData(url) {
return fetch(url)
.then(response => response.json())
.then(data => {
console.log('Fetched data:', data);
return data;
})
.then(data => {
const count = data.length;
console.log('Number of items:', count);
})
.catch(error => {
console.log('Error:', error);
});
}
const apiUrl = 'https://jsonplaceholder.typicode.com/posts';
fetchData(apiUrl);
</script>
</body>
</html>
```

OUTPUT:

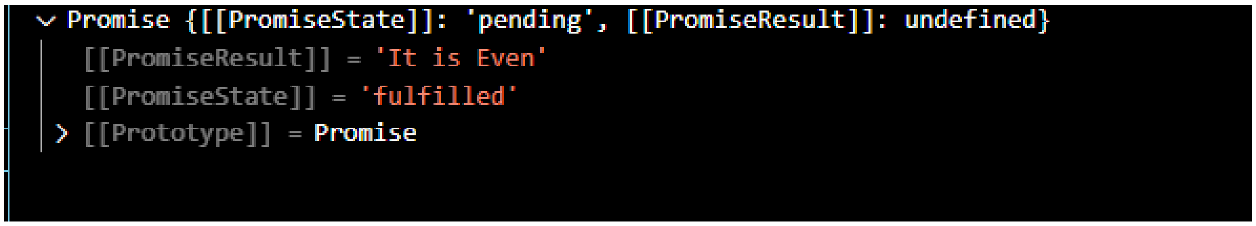


```
Fetched data:  ▶ Array(100)                    Untitled-1.html:10
Number of items: 100                           Untitled-1.html:15
```

4.3. Task 3: Create a promise that either resolves or rejects based on a random number

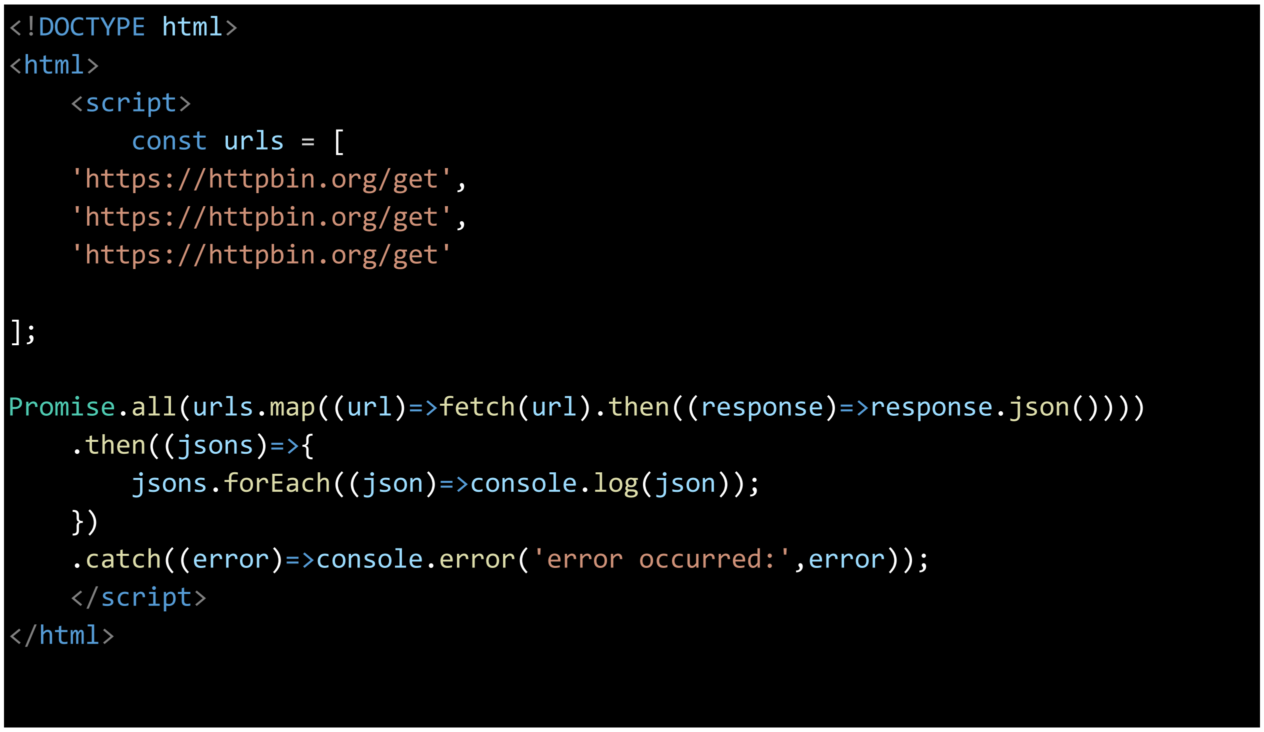
```
<!DOCTYPE html>
<html>
    <script>
        var data=new Promise((resolve,reject)=>{
            setTimeout(()=>{
                var n=parseInt(prompt("Enter Number:"));
                if(n%2==0)
                  resolve("It is Even");
                else
                    reject("It is Odd");
            },5000);
        })
        console.log(data);
    </script>
</html>
```

OUTPUT:



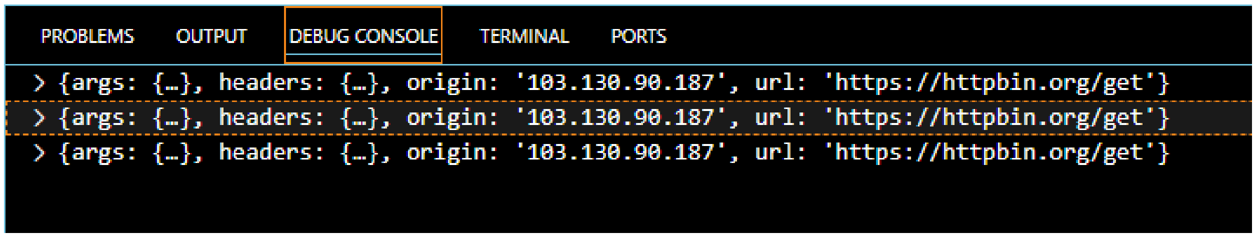
```
⌄ Promise {[[PromiseState]]: 'pending', [[PromiseResult]]: undefined}
    [[PromiseResult]] = 'It is Even'
    [[PromiseState]] = 'fulfilled'
  › [[Prototype]] = Promise
```

4.4. Task 4: Use Promise.all to fetch multiple resources in parallel from an API.



```
<!DOCTYPE html>
<html>
    <script>
        const urls = [
    'https://httpbin.org/get',
    'https://httpbin.org/get',
    'https://httpbin.org/get'

];

Promise.all(urls.map((url)=>fetch(url).then((response)=>response.json())))
    .then((jsons)=>{
        jsons.forEach((json)=>console.log(json));
    })
    .catch((error)=>console.error('error occurred:',error));
    </script>
</html>
```

OUTPUT:

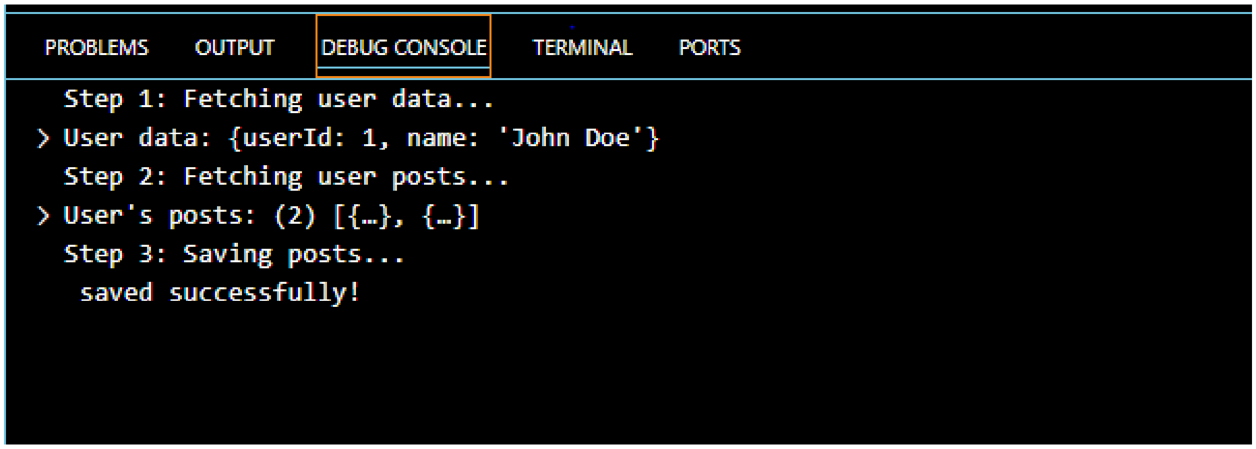


```
› {args: {…}, headers: {…}, origin: '103.130.90.187', url: 'https://httpbin.org/get'}
› {args: {…}, headers: {…}, origin: '103.130.90.187', url: 'https://httpbin.org/get'}
› {args: {…}, headers: {…}, origin: '103.130.90.187', url: 'https://httpbin.org/get'}
```

4.5. Task 5: Chain multiple promises to perform a series of asynchronous actions in sequence.

```
<!dOCTYPE HTML>
<html>
  <head></head>
  <title>My webpage</title>
  <body>
    <script>
    function step1() {
  return new Promise((resolve) => {
    console.log("Step 1: Fetching user data...");
    setTimeout(() => resolve({ userId: 1, name: "John Doe" }), 1000);
  });
}

function step2(user) {
  return new Promise((resolve) => {
    console.log("Step 2: Fetching user posts...");
    setTimeout(() => resolve([{ id: 1, title: "Post 1" }, { id: 2, title: "Post
2" }]), 1000);
  });
}

function step3(posts) {
  return new Promise((resolve) => {
    console.log("Step 3: Saving posts...");
    setTimeout(() => resolve(" saved successfully!"), 1000);
  });
}
step1()
  .then(user => {
    console.log("User data:", user);
    return step2(user);
  })
  .then(posts => {
    console.log("User's posts:", posts);
    return step3(posts);
  })
  .then(message => {
    console.log(message);
  })
  .catch(error => {
    console.error("Error:", error);
  });
  </script>
  </body>
  </html>
```

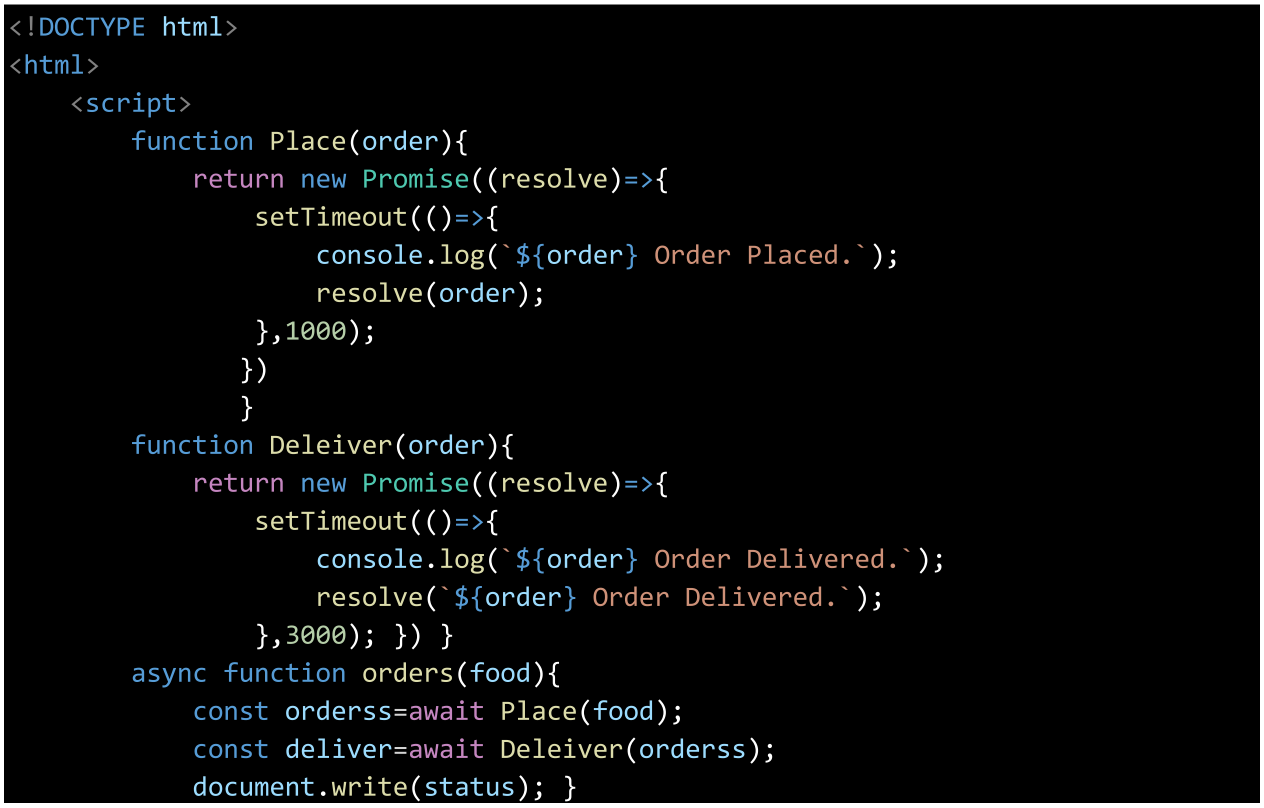
OUTPUT:



```
  Step 1: Fetching user data...
> User data: {userId: 1, name: 'John Doe'}
  Step 2: Fetching user posts...
> User's posts: (2) [{…}, {…}]
  Step 3: Saving posts...
   saved successfully!
```

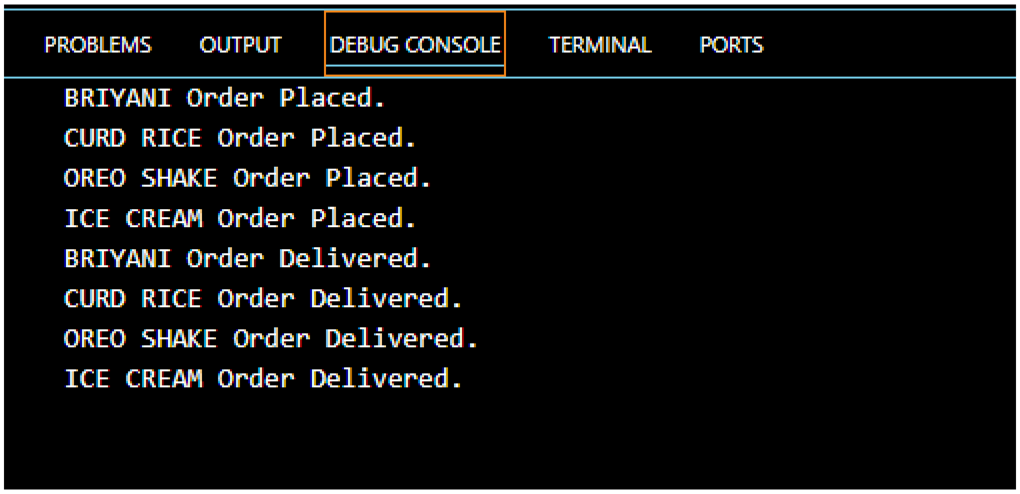
5. Async/await:

5.1.Task 1: Rewrite a promise-based function using async/await.



```
<!DOCTYPE html>
<html>
    <script>
        function Place(order){
            return new Promise((resolve)=>{
                setTimeout(()=>{
                    console.log(`${order} Order Placed.`);
                    resolve(order);
                },1000);
               })
               }
        function Deleiver(order){
            return new Promise((resolve)=>{
                setTimeout(()=>{
                    console.log(`${order} Order Delivered.`);
                    resolve(`${order} Order Delivered.`);
                },3000); }) }
        async function orders(food){
            const orderss=await Place(food);
            const deliver=await Deleiver(orderss);
            document.write(status); }
        orders("BRIYANI");
        orders("CURD RICE");
        orders("OREO SHAKE");
        orders("ICE CREAM");
    </script>
</html>
```

OUTPUT:

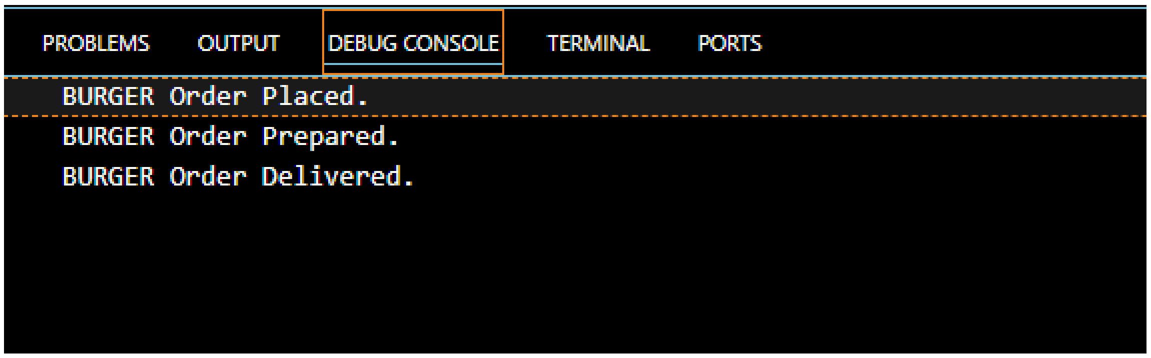


```
BRIYANI Order Placed.
CURD RICE Order Placed.
OREO SHAKE Order Placed.
ICE CREAM Order Placed.
BRIYANI Order Delivered.
CURD RICE Order Delivered.
OREO SHAKE Order Delivered.
ICE CREAM Order Delivered.
```

5.2. Task 2: Create an async function that fetches data from an API and processes it.

```
<!DOCTYPE html>
<html>
    <script>
        function PlaceFood(order){
            return new Promise((resolve)=>{
                setTimeout(()=>{
                    console.log(`${order} Order Placed.`);
                    resolve(order);
                },1000);
            })
        }
        function PrepareFood(order){
            return new Promise((resolve)=>{
                setTimeout(()=>{
                    console.log(`${order} Order Prepared.`);
                    resolve(order);
                },1000);
            })
        }
        function DeleiverFood(order){
            return new Promise((resolve)=>{
                setTimeout(()=>{
                    console.log(`${order} Order Delivered.`);
                    resolve(`${order} Order Delivered.`);
                },1000);
            })
        }
        async function orders(food){
            const orderss=await PlaceFood(food);
            const Prepare=await PrepareFood(orderss);
            const deliver=await DeleiverFood(Prepare);
            document.write(status);
        }
        orders("BURGER");
    </script>
</html>
```

OUTPUT:



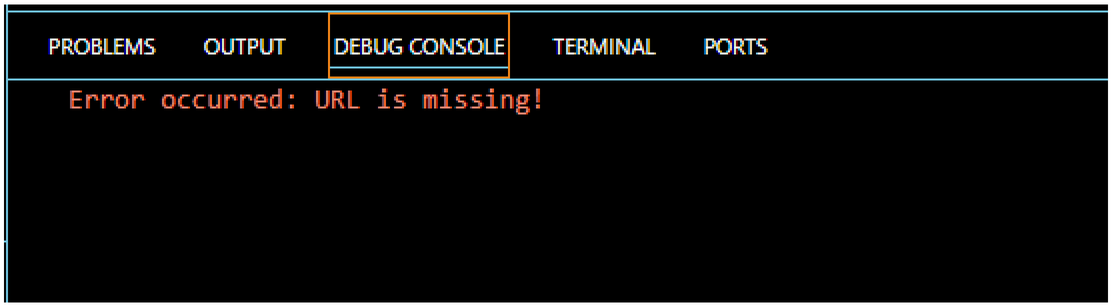
```
BURGER Order Placed.
BURGER Order Prepared.
BURGER Order Delivered.
```

5.3Task 3: Implement error handling in an async function using try/catch.

```
<!DOCTYPE html>

<html>
```

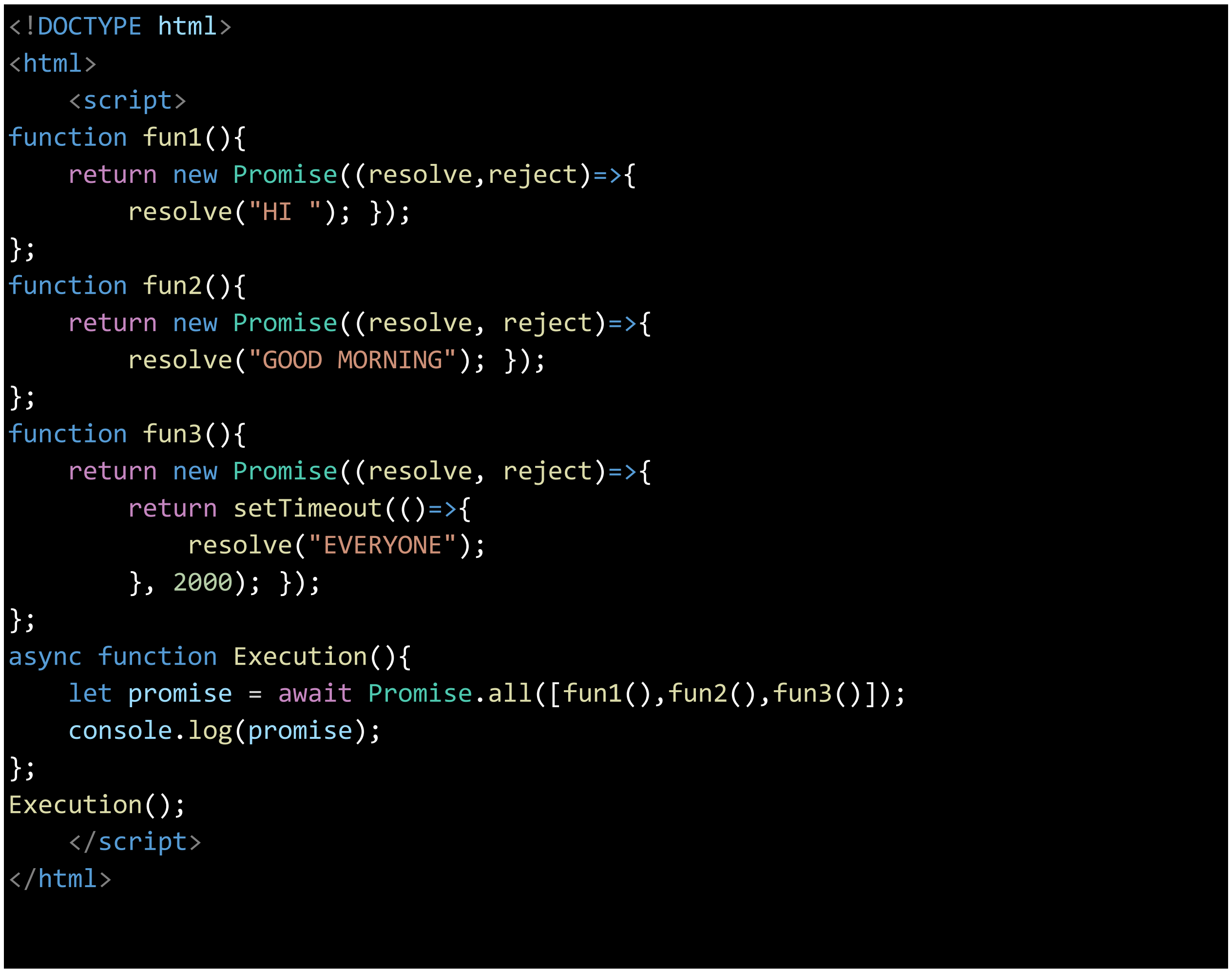
```
  <title>TASK 5.3</title>

  <body>

    <script>

async function fetchData() {

  throw new Error('URL is missing!');

}

async function fun() {

  try {

    const res = await fetchData();

    console.log('Data fetched:', res);

  } catch (error) {

    console.error('Error occurred:', error.message);

  }

}

fun();

      </script>

  </body>

</html>
```

OUTPUT:



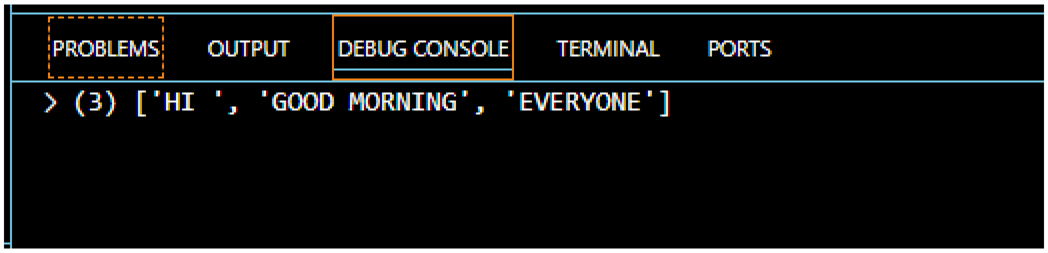
```
Error occurred: URL is missing!
```

5.4Task 4: Use async/await in combination with Promise.all.



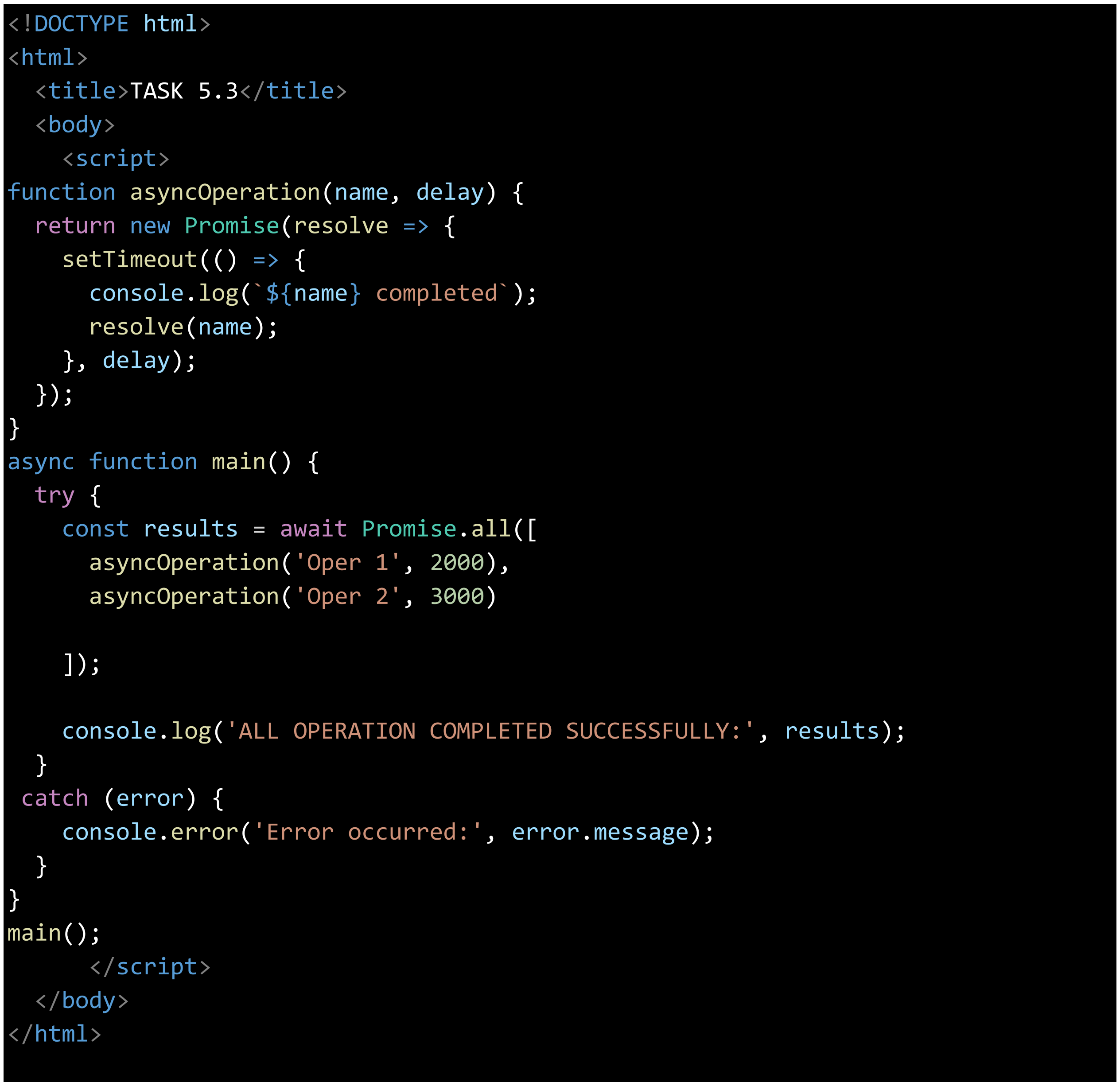
```
<!DOCTYPE html>
<html>
    <script>
function fun1(){
    return new Promise((resolve,reject)=>{
        resolve("HI "); });
};
function fun2(){
    return new Promise((resolve, reject)=>{
        resolve("GOOD MORNING"); });
};
function fun3(){
    return new Promise((resolve, reject)=>{
        return setTimeout(()=>{
            resolve("EVERYONE");
        }, 2000); });
};
async function Execution(){
    let promise = await Promise.all([fun1(),fun2(),fun3()]);
    console.log(promise);
};
Execution();
    </script>
</html>
```

OUTPUT:



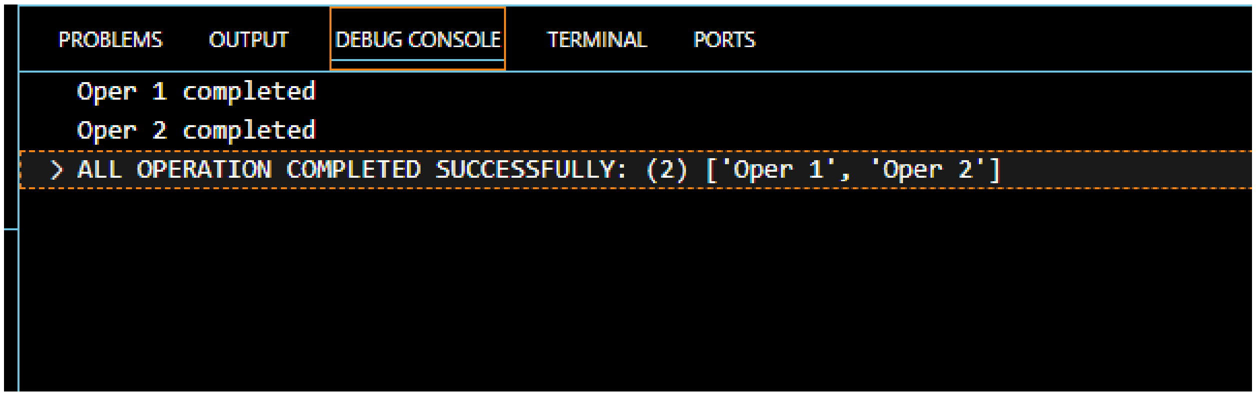
```
> (3) ['HI ', 'GOOD MORNING', 'EVERYONE']
```

5.5Task 5: Create an async function that waits for multiple asynchronous operations to complete before proceeding.



```
<!DOCTYPE html>
<html>
  <title>TASK 5.3</title>
  <body>
    <script>
function asyncOperation(name, delay) {
  return new Promise(resolve => {
    setTimeout(() => {
      console.log(`${name} completed`);
      resolve(name);
    }, delay);
  });
}
async function main() {
  try {
    const results = await Promise.all([
      asyncOperation('Oper 1', 2000),
      asyncOperation('Oper 2', 3000)

    ]);

    console.log('ALL OPERATION COMPLETED SUCCESSFULLY:', results);
  }
 catch (error) {
    console.error('Error occurred:', error.message);
  }
}
main();
      </script>
  </body>
</html>
```

OUTPUT:



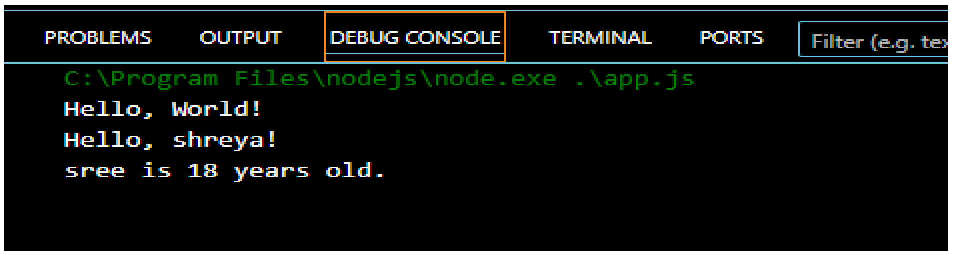
```
Oper 1 completed
Oper 2 completed
> ALL OPERATION COMPLETED SUCCESSFULLY: (2) ['Oper 1', 'Oper 2']
```

6. Modules introduction,Export and Import

6.1 Task 1: Create a module that exports a function, a class, and a variable

```
export const greeting = "Hello, World!";

// A simple function
export function greet(name) {
  return `Hello, ${name}!`;
}

// A simple class
export class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  getDetails() {
    return `${this.name} is ${this.age} years old.`;
  }
}
import { greeting, greet, Person } from './module.js';

console.log(greeting);
console.log(greet('shreya'));
const person = new Person('sree', 18);
console.log(person.getDetails());
```

OUTPUT:

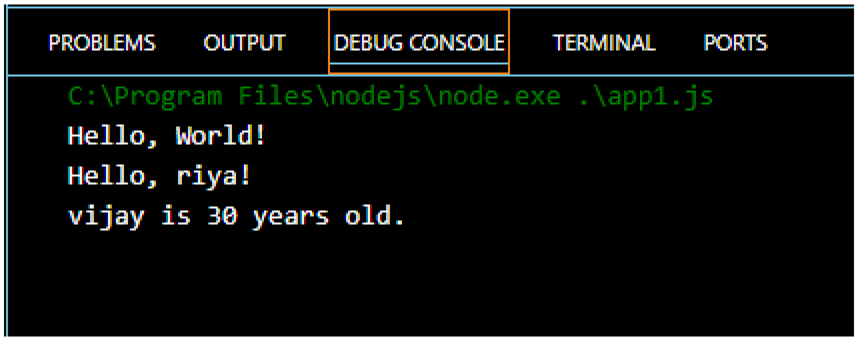


```
C:\Program Files\nodejs\node.exe .\app.js
Hello, World!
Hello, shreya!
sree is 18 years old.
```

6.2 Task 2: Import the module in another JavaScript file and use the exported entities.

```
export const greeting = "Hello, World!";

// A simple function
export function greet(name) {
  return `Hello, ${name}!`;
}

// A simple class
export class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  getDetails() {
    return `${this.name} is ${this.age} years old.`;
  }
}
import { greeting, greet, Person } from './module1.js';

console.log(greeting);

console.log(greet('riya'));

const person = new Person('vijay',30);
console.log(person.getDetails());
```

OUTPUT:



```
C:\Program Files\nodejs\node.exe .\app1.js
Hello, World!
Hello, riya!
vijay is 30 years old.
```

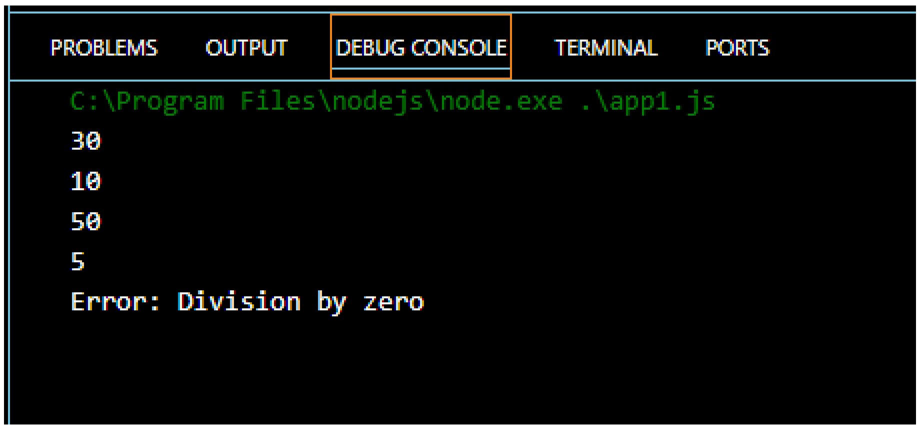
6.3 Task 3: Use named exports to export multiple functions from a module.

```
export function add(a, b) {
    return a + b;
  }
  export function subtract(a, b) {
    return a - b;
  }

  export function multiply(a, b) {
    return a * b;
  }
  export function divide(a, b) {
    if (b === 0) {
      return 'Error: Division by zero';
    }
    return a / b;
  }
```

6.4 Task 4: Use named imports to import specific functions from a module.

```
import { add, subtract, multiply, divide } from './module1.js';
console.log(add(10, 20));
console.log(subtract(30,20));
console.log(multiply(10,5));
console.log(divide(10,2));
console.log(divide(10, 0));
```

OUTPUT (3,4):

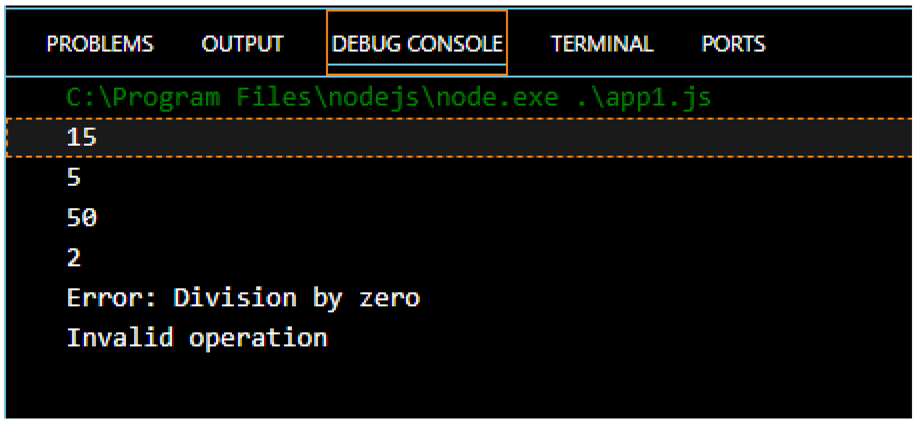


```
C:\Program Files\nodejs\node.exe .\app1.js
30
10
50
5
Error: Division by zero
```

6.5 Task 5: Use default export and import for a primary function of a module

```
export default function calculate(a, b, operation) {
    switch (operation) {
      case 'add':
        return a + b;
      case 'subtract':
        return a - b;
      case 'multiply':
        return a * b;
      case 'divide':
        if (b === 0) {
          return 'Error: Division by zero';
        }
        return a / b;
      default:
        return 'Invalid operation';
    }
  }
import calculate from './module1.js';
console.log(calculate(10, 5, 'add'));
console.log(calculate(10, 5, 'subtract'));
console.log(calculate(10, 5, 'multiply'));
console.log(calculate(10, 5, 'divide'));
console.log(calculate(10, 0, 'divide'));
console.log(calculate(10, 5, 'unknown'));
```

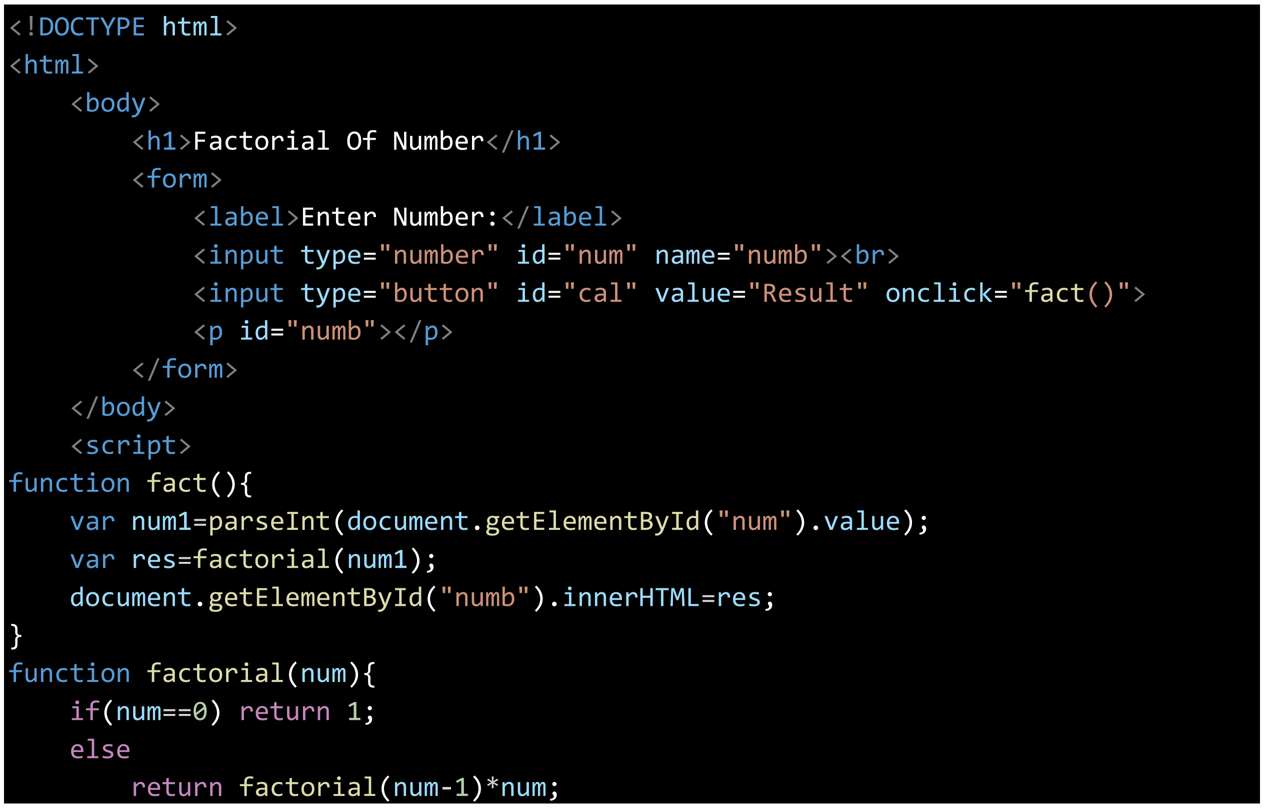
OUTPUT:



```
C:\Program Files\nodejs\node.exe .\app1.js
15
5
50
2
Error: Division by zero
Invalid operation
```

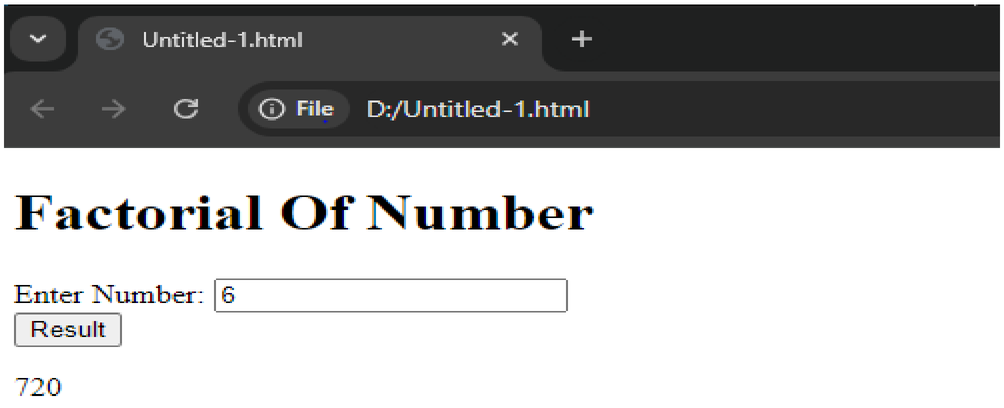
7.Browser DOM Basics

7.1Task 1: Select an HTML element by its ID and change its content using JavaScript.



```
<!DOCTYPE html>
<html>
    <body>
        <h1>Factorial Of Number</h1>
        <form>
            <label>Enter Number:</label>
            <input type="number" id="num" name="numb"><br>
            <input type="button" id="cal" value="Result" onclick="fact()">
            <p id="numb"></p>
        </form>
    </body>
    <script>
function fact(){
    var num1=parseInt(document.getElementById("num").value);
    var res=factorial(num1);
    document.getElementById("numb").innerHTML=res;
}
function factorial(num){
    if(num==0) return 1;
    else
        return factorial(num-1)*num;
}
    </script>
</html>
```

OUTPUT:



Factorial Of Number

Enter Number: 6

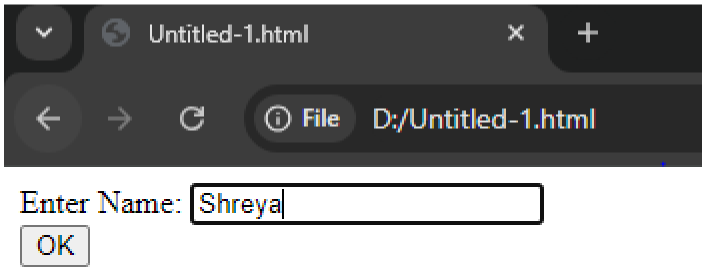
Result

720

7.2Task 2: Attach an event listener to a button, making it perform an action when clicked.

```
<!DOCTYPE html>
<html>
    <body>
        <form>
            <label>Enter Name:</label>
            <input type="text" id="nam" name="namm"><br>
            <input type="button" id="cal" value="OK" onclick="fun()">
            <p id="numm"></p>
        </form>
    </body>
    <script>
function fun(){

    var name=document.getElementById("nam").value;
    document.getElementById("numm").innerHTML=document.write(`Hello! ${name}`);
}
    </script>
</html>
```

OUTPUT:

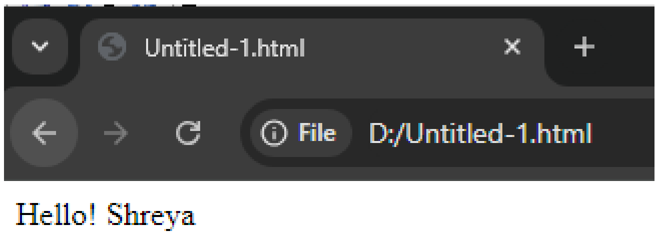


Untitled-1.html

File D:/Untitled-1.html

Enter Name: Shreya

OK



Untitled-1.html

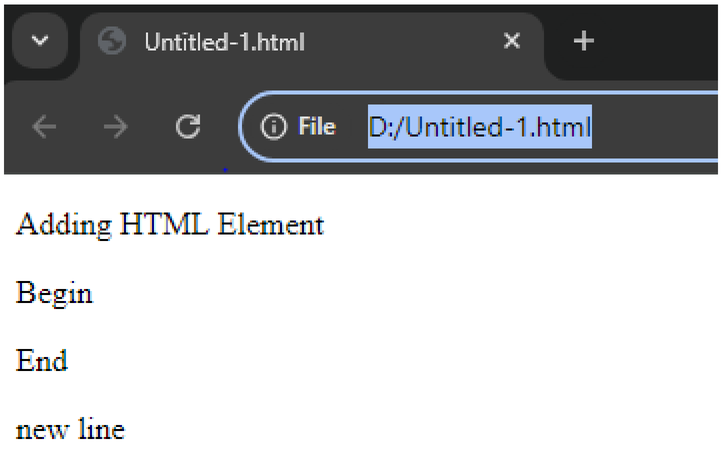
File D:/Untitled-1.html

Hello! Shreya

7.3Task 3: Create a new HTML element and append it to the DOM.

```
<!DOCTYPE html>
<html>
<body>
    <p>Adding  HTML Element</p>
    <div id="d">
        <p id="p1">Begin</p>
        <p id="p2">End</p>
    </div>
<script>
const a=document.createElement("p");
const node=document.createTextNode("new line");
a.appendChild(node);
const ele=document.getElementById("d");
ele.appendChild(a);
</script>
</body>
</html>
```

OUTPUT:

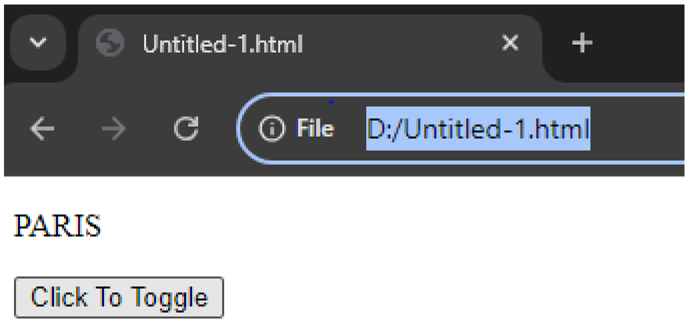


```
Adding HTML Element

Begin

End

new line
```

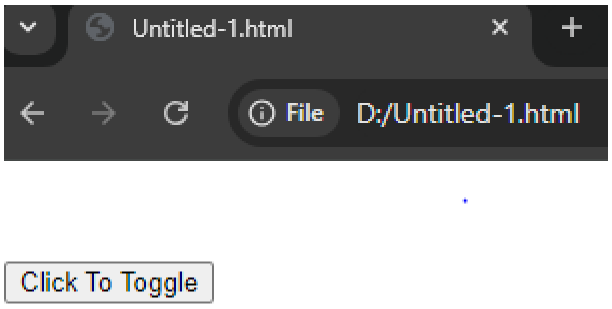
7.4 Task 4: Implement a function to toggle the visibility of an element.

```
<!DOCTYPE html>
<html lang="en">
<body>
        <p id="m">PARIS<br></p>
        <button onclick="toggleEle()">
            Click To Toggle
        </button>
    <script>
        function toggleEle(){
            const a=document.getElementById('m');
            const vi=window.getComputedStyle(a).visibility;
            if (vi==='hidden')
                a.style.visibility='visible';
             else
                a.style.visibility='hidden';
        }
    </script>
</body>

</html>
```

OUTPUT:



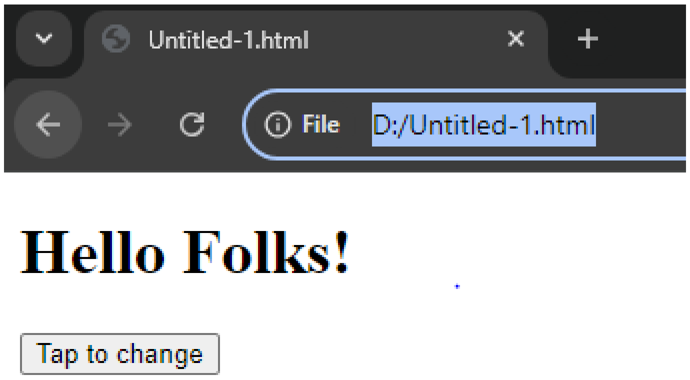
PARIS



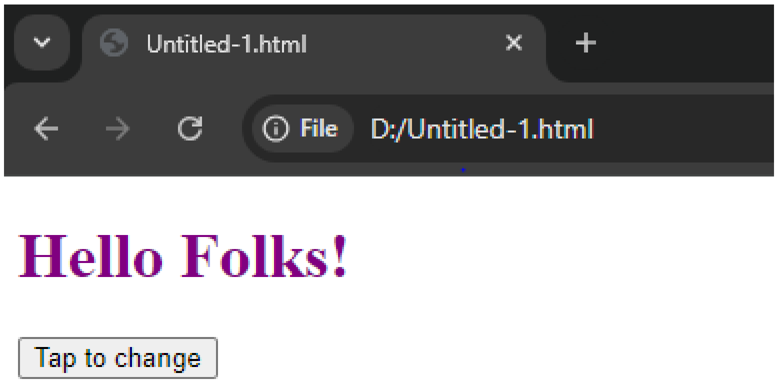
7.5Task 5: Use the DOM API to retrieve and modify the attributes of an element.

```
<!DOCTYPE html>
<html>
    <style>
        .attributee{
            color:purple;
        };
    </style>
    <h1 id="Id">Hello Folks!</h1>
    <button onclick="addAttribute()">Tap to change</button>
    <script>
        function addAttribute(){
            document.getElementById("Id").setAttribute("class","attributee");
        }
        </script>
        </html>
```

OUTPUT:



```
Hello Folks!
```



```
Hello Folks!
```