**1.Recursion and stack:**

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        function factorial(num)

        {

            if(num==0 || num==1)

            {

                return 1;

            }

            return num\*factorial(num-1);

        }

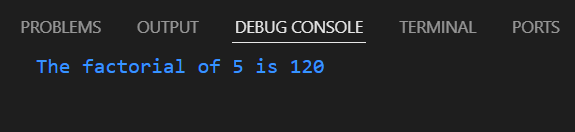
        console.log("The factorial of 5 is "+factorial(5));

    </script>

</body>

</html>

**Output:**

****

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

       function fibonacci(num)

       {

        if(num==0 || num==1)

        {

            return num;

        }

        return fibonacci(num-1)+fibonacci(num-2);

       }

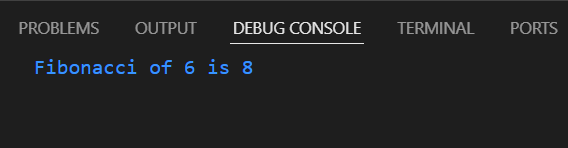
       console.log("Fibonacci of 6 is "+fibonacci(6));

    </script>

</body>

</html>

**Output:**

****

**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 lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        function stair(num)

        {

            if(num==0 || num==1)

            {

                return 1;

            }

            if(num==2)

            {

                return 2;

            }

            return stair(num-1)+stair(num-2)+stair(num-3);

        }

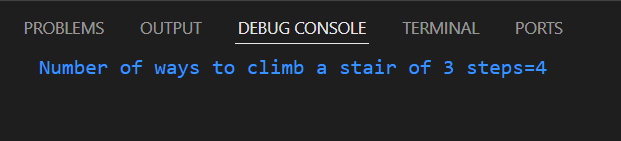
        console.log("Number of ways to climb a stair of 3 steps="+stair(3));

    </script>

</body>

</html>

**Output:**

****

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        function helper(arr,n)

        {

            if(n==0)

            {

                return arr;

            }

            arr=arr.flat();

            return helper(arr,n-1);

        }

        let arr=[1,2,[3,4],[5,6,[7,8],9]];

        let n=arr.length;

        let arr1=helper(arr,n);

        console.log(arr1);

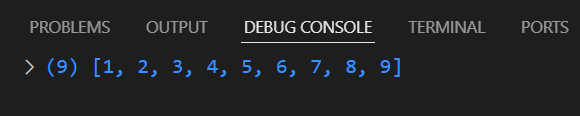
        document.write(arr1);

    </script>

</body>

</html>

**Output:**

****

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let count=0;

        function towerOfHanoi(n,from,to,aux)

        {

            if(n==0)

            {

                return;

            }

            towerOfHanoi(n-1,from,aux,to);

            count++;

            console.log("Move "+n+" from "+from+" to "+to);

            towerOfHanoi(n-1,aux,to,from);

        }

        towerOfHanoi(3,'a','b','c');

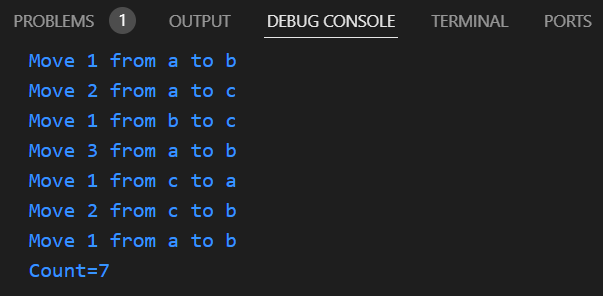
        console.log("Count="+count);

    </script>

</body>

</html>

**Output:**

****

**2.JSON and variable length arguments/spread syntax:**

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let sum=0;

        let value=+prompt("Enter a value=");

        function solve(num)

        {

            sum+=num;

            return sum;

        }

        for(i=0;i<value;i++)

        {

            let n=+prompt("Enter a number=");

            ans=solve(n);

        }

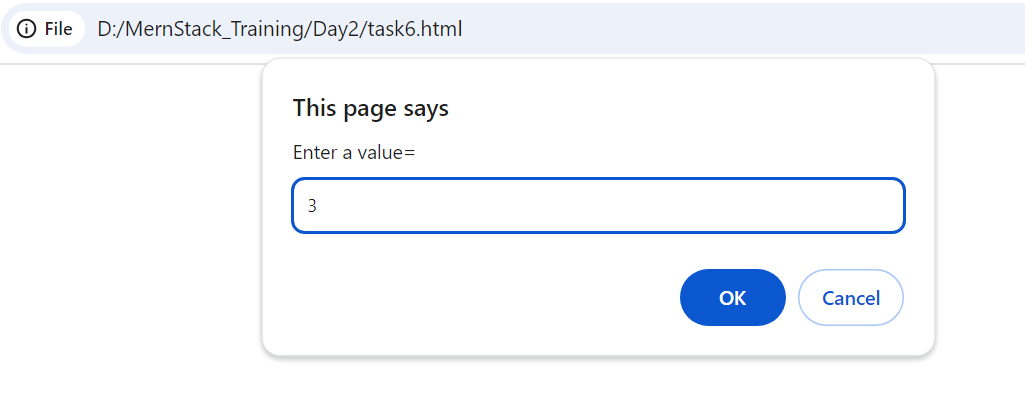
        document.write("sum="+ans);

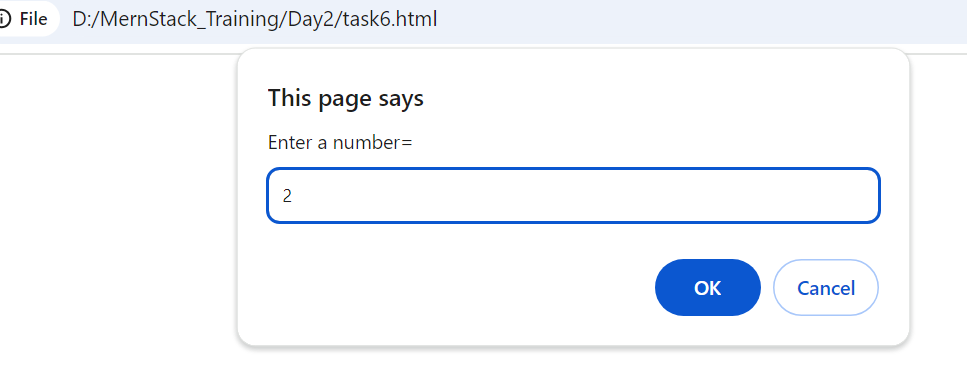
    </script>

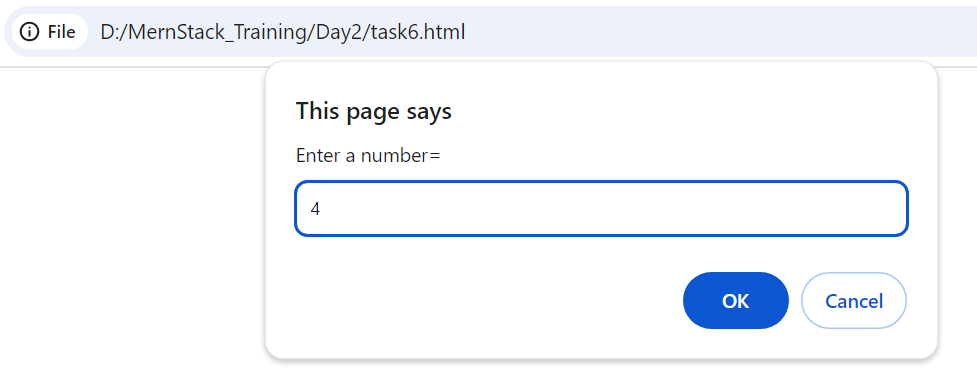
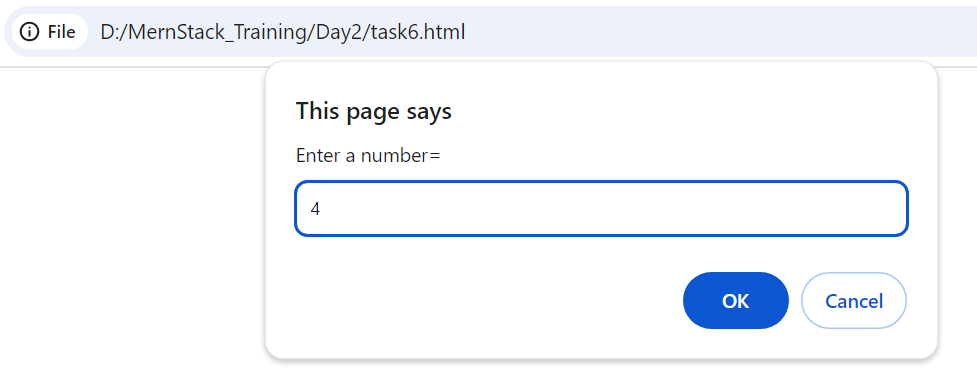
</body>

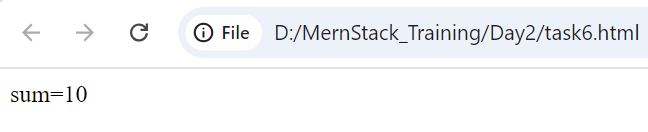
</html>

**Output:**

****

****

**** ****

****

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        function sumArray(...a)

        {

            sum=0;

            n=arr.length;

            for(i=0;i<n;i++)

            {

                sum=sum+a[i];

            }

            return sum;

        }

        let arr=[3,4,5,8];

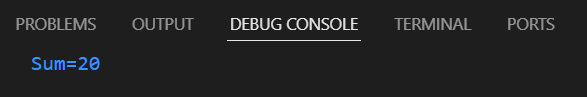
        console.log(sumArray(...arr));

    </script>

</body>

</html>

**Output:**

****

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let object={

            a:1,

            b:{c:3},

            d:{e:{f:7}}

        };

        obj=JSON.stringify(object);

        cloneObj=JSON.parse(obj);

        document.write("CloneObject="+JSON.stringify(cloneObj,null,2)+"<br>");

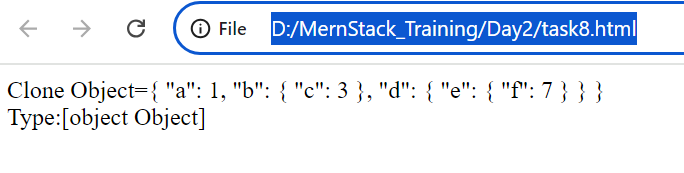
        document.write("Type:"+cloneObj);

    </script>

</body>

</html>

**Output:**

****

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        function mergeObj(object1,object2)

        {

            return {...object1,...object2};

        }

        let object1={

            name:"Sam",

            age:20,

            place:"Coimbatore"

        };

        let object2={

            name:"John",

            dept:"ECE",

            year:"III"

        };

        console.log("Merged Object:")

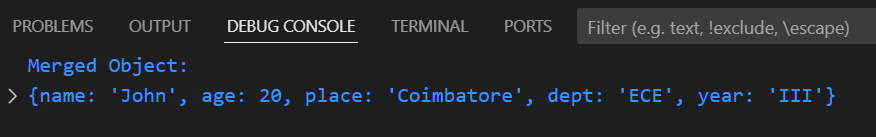
        console.log(mergeObj(object1,object2));

    </script>

</body>

</html>

**Output:**

****

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let object={

            name:"John",

            age:20,

            dept:"ECE"

        };

        let obj=JSON.stringify(object);

        console.log("JSON object is "+obj);

        let normalObj=JSON.parse(obj);

        console.log("Normal Object:")

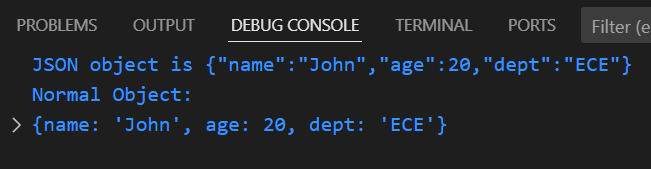
        console.log(normalObj);

    </script>

</body>

</html>

**Output:**

****

**3. Closure:**

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let a=10;

        function sum(num)

        {

            function solve(val)

            {

                sum1=a+num+val;

                return sum1;

            }

            return solve(10);

        }

        console.log("Sum="+(sum(10)));

    </script>

</body>

</html>

**Output:**

****

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        function count(num)

        {

            let val=0;

            while(val<num)

            {

                val++;

                function display()

                {

                    console.log("val="+val);

                }

                display();

            }

        }

        count(5);

    </script>

</body>

</html>

**Output:**



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

**Output:**

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

**Output:**

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

**Output:**

**4. Promise, Promises chaining:**

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

**<!DOCTYPE h­­­­­tml>**

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let promise=new Promise(function(myResolve,myReject)

        {

            setTimeout(function(){myResolve("Greetings!");},3000)

        });

        promise.then(function(value)

        {

            document.write(value);

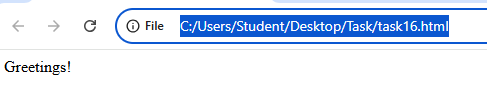
        });

    </script>

</body>

</html>

**Output:**



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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        const url = "https://jsonplaceholder.typicode.com/users/1";

        fetch(url).then(function myResponse(response)

        {

          if(response.ok)

          {

            return response.json();

          }

          else

          {

            throw new Error("Network Error");

          }

        }).then(function myResolve(data)

        {

          document.write(JSON.stringify(data));

        }).catch(function MyError(error)

        {

          document.write(error);

        });

    </script>

</body>

</html>

**Output:**



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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let promise=new Promise(function(resolve,reject){

            let num=15;

            if(num>10)

            {

                resolve("Number is valid");

            }

            else

            {

                reject("Number is invalid");

            }

        });

        promise.then(

            function(value){console.log(value);},

            function(error){console.log(error);}

        );

    </script>

</body>

</html>

**Output:**



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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        const url1="https://jsonplaceholder.typicode.com/users/1";

        const url2="https://jsonplaceholder.typicode.com/users/2";

        let user1=fetch(url1).

        then(function getUser1(response)

        {

            if(response.ok)

            {

                return response.json();

            }

            else{

                throw new Error("Network error");

            }

        });

        let user2=fetch(url2).

        then(function getUser1(response)

        {

            if(response.ok)

            {

                return response.json();

            }

            else{

                throw new Error("Network error");

            }

        });

        Promise.all([user1,user2]).then(

            function myResponse(result)

            {

                const[users1,users2]=result;

                document.write("User-1"+"<br>"+JSON.stringify(users1)+"<br>");

                document.write("User-2"+"<br>"+JSON.stringify(users2));

            }

        ).catch(function MyError(error)

        {

            document.write(error);

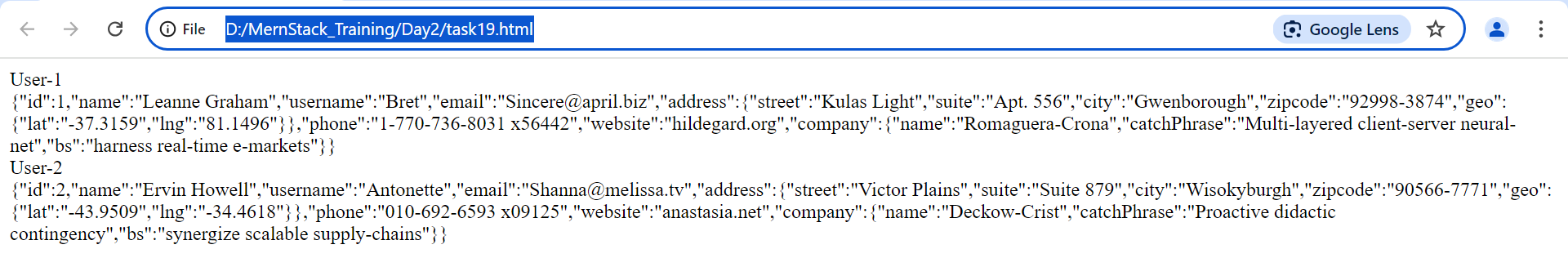
        });

    </script>

</body>

</html>

**Output:**



**Task 5:** Chain multiple promises to perform a series of asynchronous actions in

sequence.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let promise=Promise.all([

            new Promise((resolve,reject)=>{setTimeout(()=>resolve(1),1000);}),

            new Promise((resolve,reject)=>{setTimeout(()=>resolve(2),2000);}),

            new Promise((resolve,reject)=>{setTimeout(()=>resolve(3),3000);})

        ]

        );

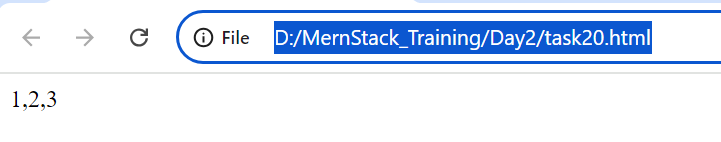
        promise.then(resolve=>document.write(resolve));

    </script>

</body>

</html>

**Output:**



**5. Async/await:**

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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let url="https://jsonplaceholder.typicode.com/posts/1";

        async function fetchUrl(url)

        {

            await fetch(url)

            .then(response=>response.json())

            .then(data=>{document.write(JSON.stringify(data));})

            .catch(error=>{document.write(error);});

        }

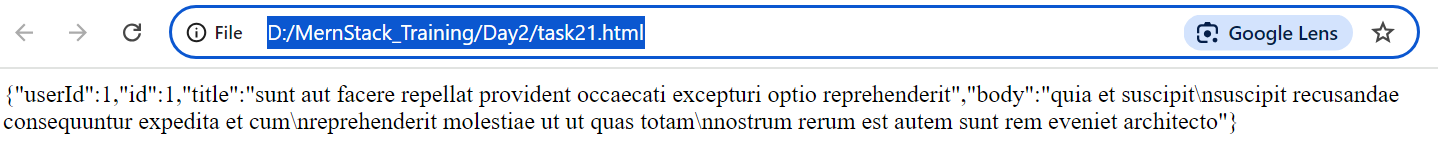
        fetchUrl(url);

    </script>

</body>

</html>

**Output:**



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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let url="https://jsonplaceholder.typicode.com/posts/1";

        async function myAsync(url)

        {

            fetch(url)

            .then(response=>response.json())

            .then(data=>document.write(JSON.stringify(data)))

            .catch(error=>document.write(error));

        }

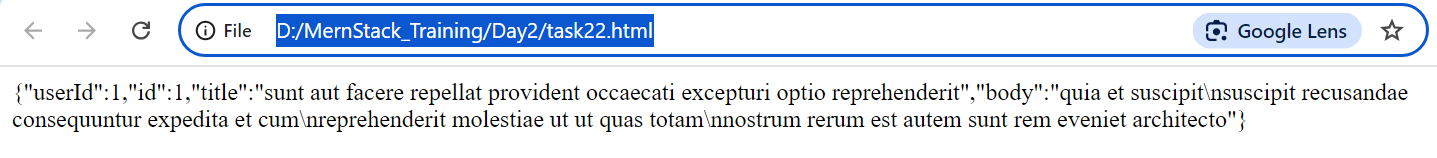
        myAsync(url);

    </script>

</body>

</html>

**Output:**



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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let url="https://jsonplaceholder.typicode.com/posts/1";

        async function fetchUrl(url)

        {

            try

            {

                let response=await fetch(url);

                let data=await response.json();

                document.write(JSON.stringify(data));

            }

            catch(error)

            {

                document.write(error);

            }

        }

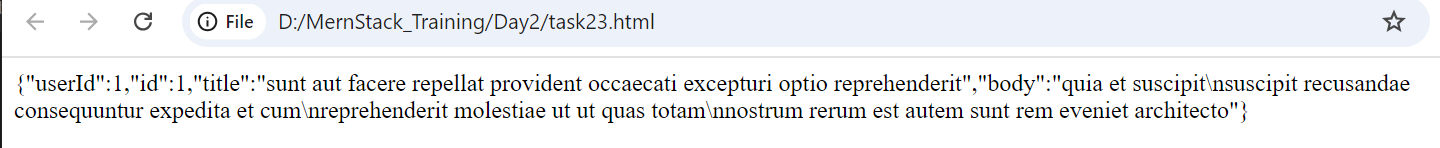
        fetchUrl(url);

    </script>

</body>

</html>

**Output:**



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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        let urls=[

            "https://jsonplaceholder.typicode.com/posts/1",

            "https://jsonplaceholder.typicode.com/posts/2",

            "https://jsonplaceholder.typicode.com/posts/3"

        ];

        async function fetchData(urlLink)

        {

            let response=await fetch(urlLink);

            let data=response.json();

            return data;

        }

        async function fetchUrls(urls)

        {

            try

            {

                let promises=[];

                for(let url of urls)

                {

                    promises.push(fetchData(url));

                }

                let results=await Promise.all(promises);

                for(let result of results)

                {

                    document.write(JSON.stringify(result)+"<br>");

                }

            }

            catch(error)

            {

                document.write(error);

            }

        }

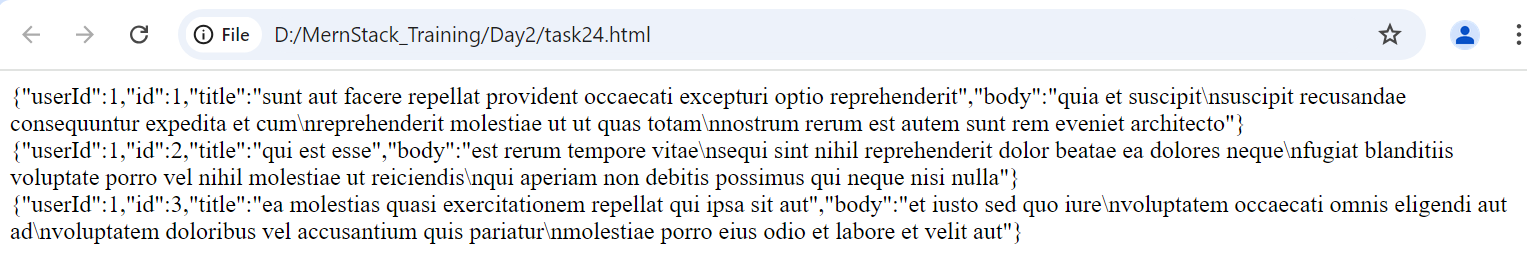
        fetchUrls(urls);

    </script>

</body>

</html>

**Output:**



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

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        async function fetchDetails() {

            let urls=[

                "https://jsonplaceholder.typicode.com/posts/1",

                "https://jsonplaceholder.typicode.com/posts/2",

                "https://jsonplaceholder.typicode.com/posts/3"

            ];

            async function fetchData(url) {

                let response=await fetch(url);

                let data=response.json();

                return data;

            }

            let promises=[];

            for(let url of urls)

            {

                promises.push(fetchData(url));

            }

            try

            {

                let results=await Promise.all(promises);

                for(let result of results)

                {

                    document.write(JSON.stringify(result)+"<br>");

                }

            }

            catch(error)

            {

                document.write(error);

            }

        }

        fetchDetails();

    </script>

</body>

</html>

**Output:**

