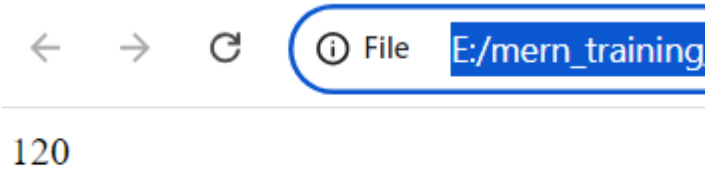
1) Task 1: Implement a function to calculate the factorial of a number using recursion.

```
<!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(n)
        {
            if(n==1 || n==0){
                return 1;
            }
            return n*factorial(n-1);
        }
        document.write(factorial(5));
    </script>
</body>
</html>
```

Output:

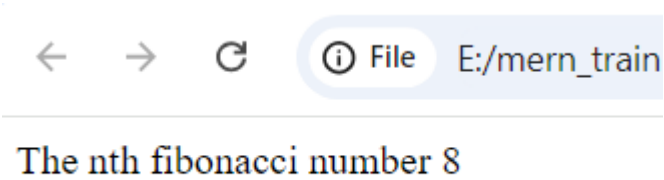


2) Task 2: Write a recursive function to find the nth Fibonacci number.

```
<!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 fibo(n)
        {
            if(n==0){
                return 0;
            }
```

```
            if(n==1 || n==2)
            {
                return 1;
            }
            return fibo(n-2)+fibo(n-1);
        }
        document.write("The nth fibonacci number " + fibo(6));
    </script>
</body>
</html>
```

Output:

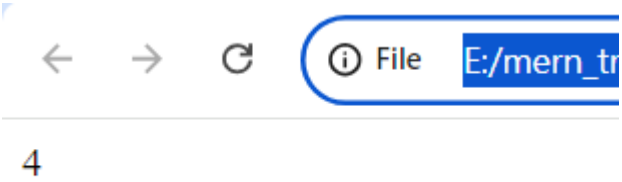


The nth fibonacci number 8

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 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 steps(n)
        {
            if(n==0){
                return 1;
            }
            if(n==1){
                return 1;
            }
            if(n==2)
            {
                return 2;
            }
            return steps(n-3)+steps(n-2)+steps(n-1);
        }
        document.write(steps(3));
    </script>
</body>
</html>
```

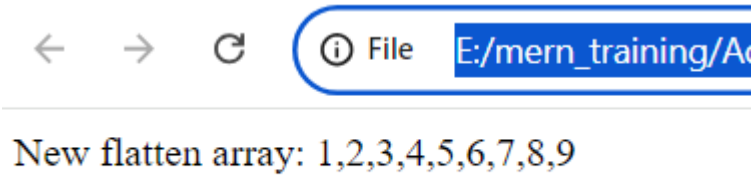
Output:



4) 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 solve(arr,n)
        {
            if(n==0)
        {
            return arr;
        }
            arr=arr.flat();
            return solve(arr,n-1);
        }
        let arr=[1,2,[3,4],[5,6,[7,8],9]];
        let len=arr.length;
        let newArray=solve(arr,len);
        document.write("New flatten array: "+newArray);
    </script>
</body>
</html>
```

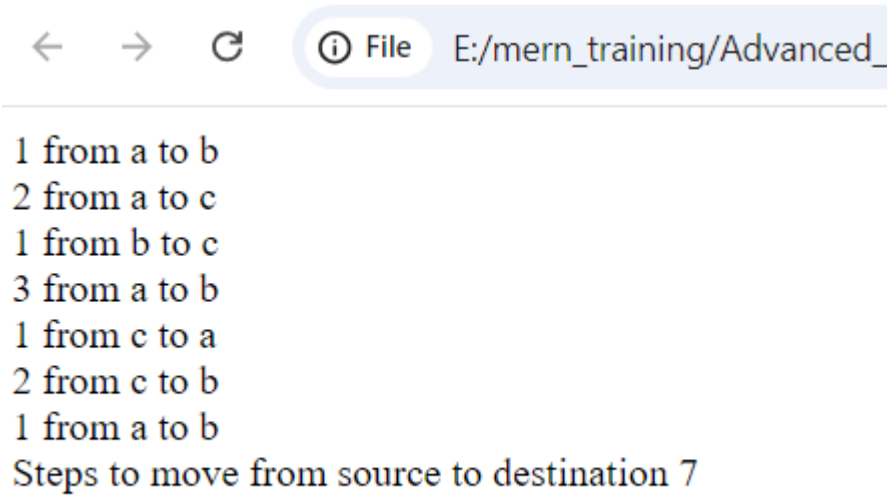
Output:



5) 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 solve(s,d,h,n)
        {
            if(n==0)
        {
            return;
        }
            solve(s,h,d,n-1);
            count++;
            document.write(n+" from "+s+" to "+d);
            document.write("<br>");
            solve(h,d,s,n-1)
        }
        solve('a','b','c',3);
        document.write("Steps to move from source to destination " +count);
    </script>
</body>
</html>
```

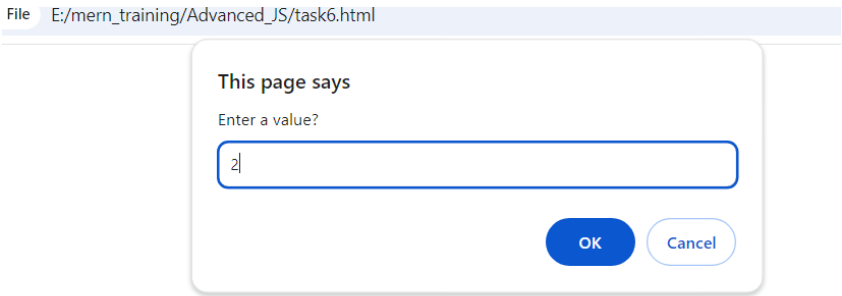
Output:

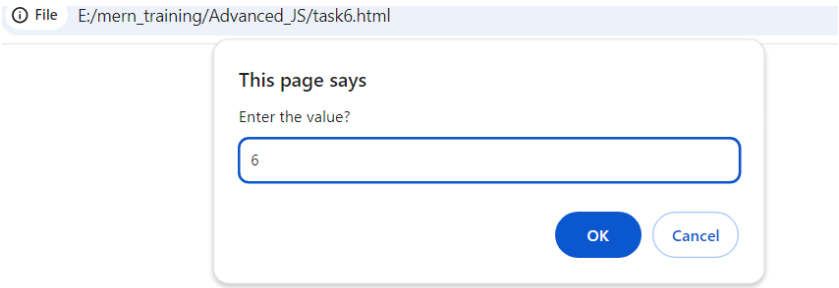


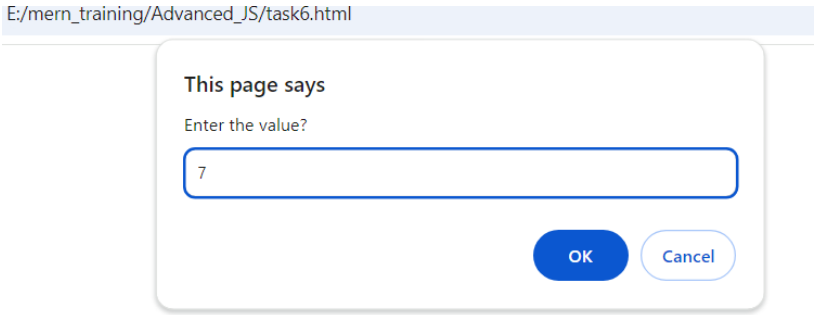
Task 6: Write a function that takes an arbitrary number of arguments and returns their sum.

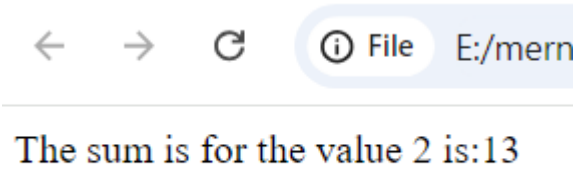
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        let val=+prompt("Enter a value?","0");
        let sum=0;
        function solve(val){
            sum=sum+val;
            return sum;
        }
        for(i=1;i<=val;i++)
        {
            let n=+prompt("Enter the value?");
            res=solve(n);
        }
        document.write("The sum is for the value "+val+" is:"+res);
    </script>
</body>
</html>
```

Output:







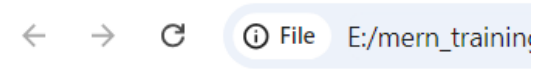


The sum is for the value 2 is:13

Task 7: 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 sum(...arr)
        {
            let s=0;
            for(i=0;i<arr.length;i++){
                s=s+arr[i];
            }
            return s;
        }
        let arr=[1,2,3,4];
        document.write("The sum of the given array :"+sum(...arr));
    </script>
</body>
</html>
```

Output:



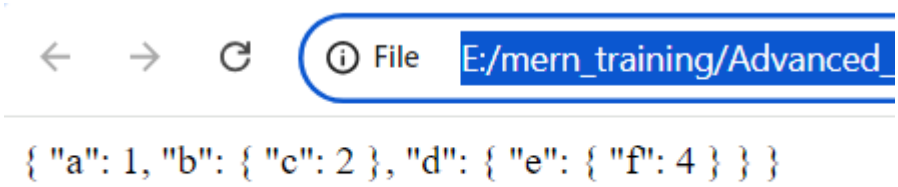
The sum of the given array :10

Task 8: 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 obj={
            a:1,
            b:{c:2},
            d:{e:{f:4}}
        };
        let cloneObj=JSON.parse(JSON.stringify(obj));
        document.write(JSON.stringify(cloneObj,null,2));

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

Output:

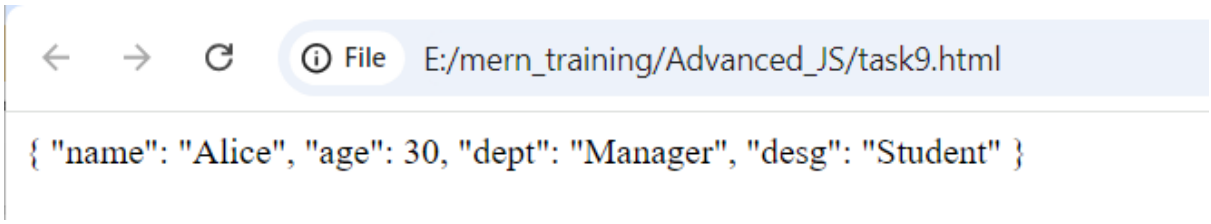


Task 9: Write a function that returns a new object, merging two provided objects using the spread syntax.

```
<!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 merge(obj1,obj2)
        {
            return {...obj1,...obj2};
        }
```

```
        let obj1={
            name:"John",
            age:30,
            dept:"Manager"
        };
        let obj2={
            name:"Alice",
            desg:"Student"
        };
        document.write(JSON.stringify(merge(obj1,obj2),null,2));
    </script>
</body>
</html>
```

Output:



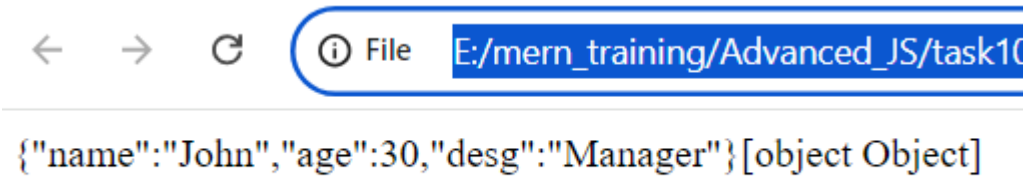
Task 10: Serialize a JavaScript object into a JSON string and then parse it back into an object.

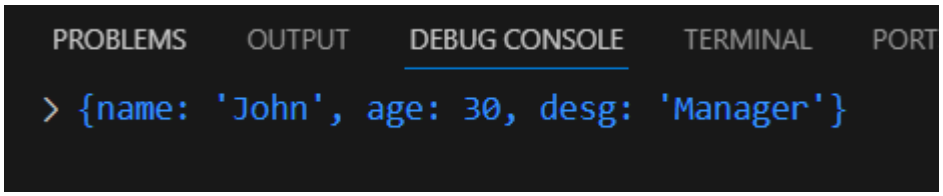
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        let obj={
            name:"John",
            age:30,
            desg:"Manager"
        }
        let str=JSON.stringify(obj);
        let jParse=JSON.parse(str);
        document.write(str);
        document.write(jParse);
        console.log(jParse);

    </script>
```

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

Output:

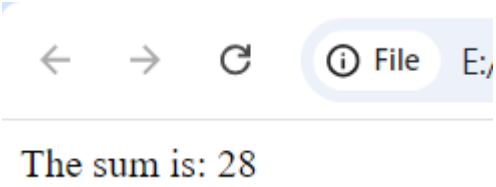




Task 11: Create a function that returns another function, capturing a local variable.

```
<!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 val1=12;
        function add()
        {
            let val2=10;
            function sum(num)
            {
                let s=val1+val2+num;
                document.write("The sum is: "+s);
            }
            sum(6);
        }
        add();
    </script>
</body>
</html>
```

Output:

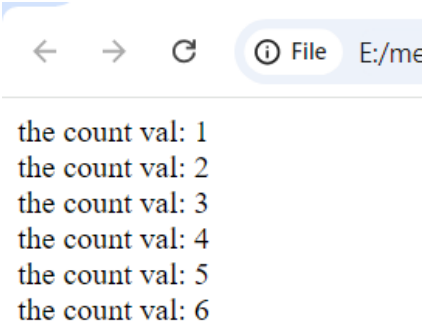


The sum is: 28

Task 12: 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>
        let c=0;
        let val=5;
        while(c<=val)
    {
        function count()
        {
            c++;
            function display()
            {
                document.write("the count val: "+c);
                document.write("<br>");
            }
            display();
        }
        count();
    }
    </script>
</body>
</html>
```

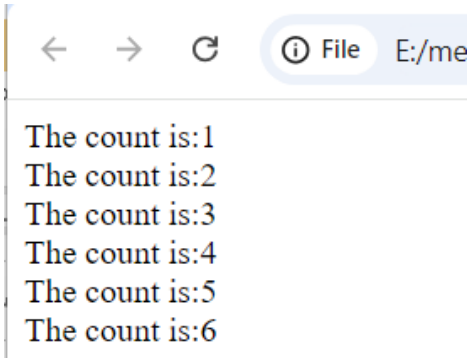
Output:



Task 13: Write a function to create multiple counters, each with its own separate 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>
        let c=0;
        let val=6;
        while(c<=val)
    {
        function counter()
        {
            function count1()
            {
                c++;
                document.write("The count is:"+c);
                document.write("<br>");
            }
            function count2()
            {
                c++;
                document.write("The count is:"+c);
                document.write("<br>");
            }
            count1();
            count2();
        }
        let counter1=counter();
    }
    </script>
</body>
</html>
```

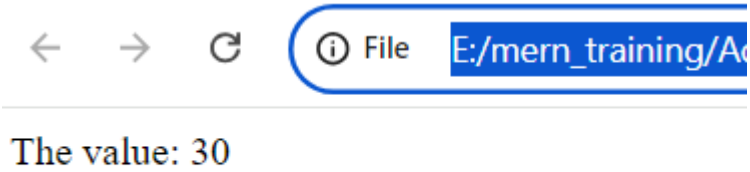
Output:



Task 14: Use closures to create private variables within a function.

```
<!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 val1=5;
        function mul()
        {
            let val2=6;
            function mul1()
            {
                let m=val1*val2;
                document.write("The value: "+m);
            }
            mul1();
        }
        mul();
    </script>
</body>
</html>
```

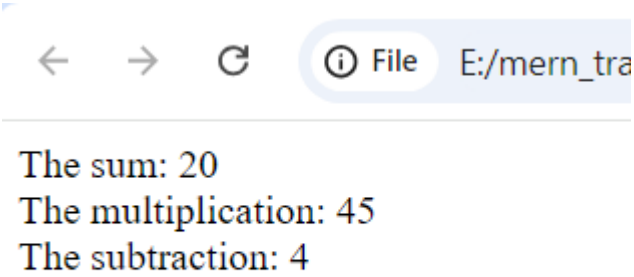
Output:



Task 15: Build a function factory that generates functions based on some input using closures.

```
<!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 solve(choice)
        {
            return function(num1,num2)
            {
                if(choice=="add"){
                    return num1+num2;
                }
                else if(choice=="mul"){
                    return num1*num2;
                }
                else if(choice=="sub")
                {
                    return num1-num2;
                }
            }
        }
        let sum=solve("add");
        let multiply=solve("mul");
        let subtract=solve("sub");
        document.write("The sum: "+sum(10,10));
        document.write("<br>");
        document.write("The multiplication: "+multiply(5,9));
        document.write("<br>");
        document.write("The subtraction: "+subtract(6,2));
    </script>
</body>
</html>
```

Output:

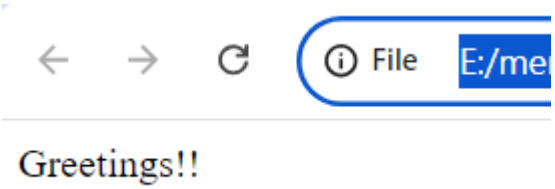


Task 16: Create a new promise that resolves after a set number of seconds and returns a greeting.

```
<!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 mypromise=new Promise(function(resolve,reject){
            setTimeout(function(){
                resolve("Greetings!!");
            },5000);
        })
        mypromise.then(function(value){
            document.write(value);
        })

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

Output:



Task 17: 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>
        let url='https://jsonplaceholder.typicode.com/users';
        fetch(url).then(response=>response.text()).then(data=>document.write(d
ata));
    </script>
</body>
</html>
```

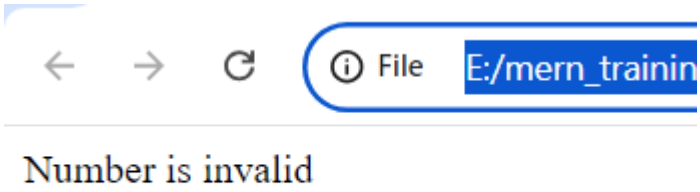
Output:

Task 18: 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 myPromise=new Promise(function(resolve,reject){
            let val=10;
            if(val>15){
                resolve("Number is valid");
            }
            else{
                resolve("Number is invalid");
            }
        });
        myPromise.then(function(value){
            document.write(value);
        },
        function(error)
        {
```

```
            document.write(error);
        }
        );
    </script>
</body>
</html>
```

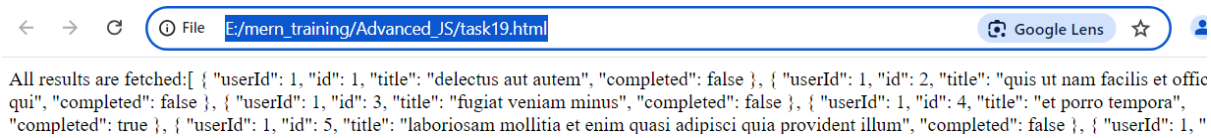
Output:



Task 19: 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>
        let urls=[
            'https://jsonplaceholder.typicode.com/todos',
            'https://jsonplaceholder.typicode.com/users'
    ];
    Promise.all(urls.map(url=>fetch(url).then(response=>response.text()))).
    then(result=>{
        document.write("All results are fetched:",result);
    }).catch(error=>{
        document.write("Error in fetching the results. ",error);
    });
    </script>
</body>
</html>
```

Output:



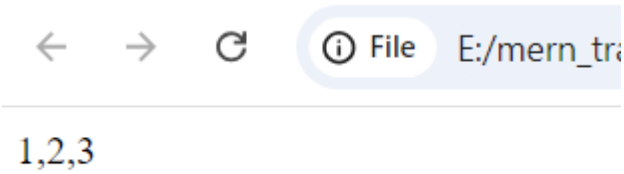
All results are fetched:[{ "userId": 1, "id": 1, "title": "delectus aut autem", "completed": false }, { "userId": 1, "id": 2, "title": "quis ut nam facilis et offic
qui", "completed": false }, { "userId": 1, "id": 3, "title": "fugiat veniam minus", "completed": false }, { "userId": 1, "id": 4, "title": "et porro tempora",
"completed": true }, { "userId": 1, "id": 5, "title": "laboriosam mollitia et enim quasi adipisci quia provident illum", "completed": false }, { "userId": 1, "

Task 20: 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 myPromise=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))
        ]).catch(error=>document.write("Promise rejected! ",error));

        myPromise.then(res=>document.write(res));
    </script>
</body>
</html>
```

Output:



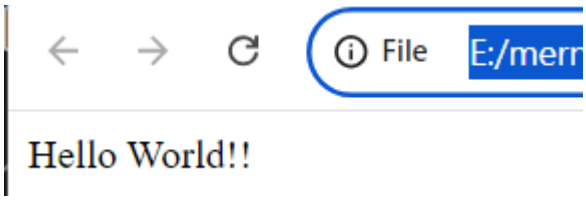
1,2,3

Task 21: 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>
        async function solve()
        {
            return "Hello World!!";
        }
        solve().then((value)=>document.write(value),(error)=>document.write(er
ror));
        </script>
</body>
```

```
</html>
```

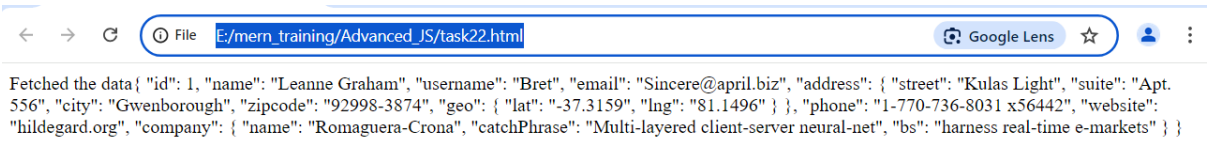
Output:



22) 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/users/1'
        async function name() {
            return
fetch(url).then(response=>response.text()).then((data)=>document.write("Fetche
d the data",data)).catch((error)=>document.write("Error in fetching the data
",error));
        }
        name().then();
    </script>
</body>
</html>
```

Output:

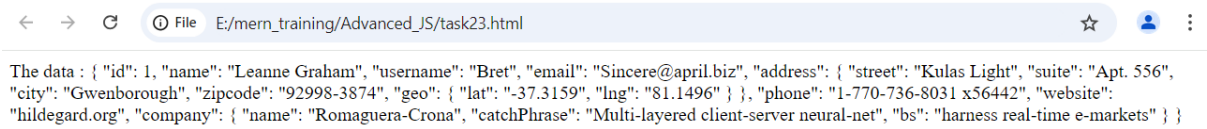


23) Implement error handling in an async function using try/catch.

```
<!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>
        async function name() {
            try{
                let url=await
fetch('https://jsonplaceholder.typicode.com/users/1');
                let data= await url.text();
                document.write("The data : ",data);
            }
            catch(error)
            {
                document.error("Error:",error);
            }
        }
        name().then();
    </script>
</body>
</html>
```

Output:



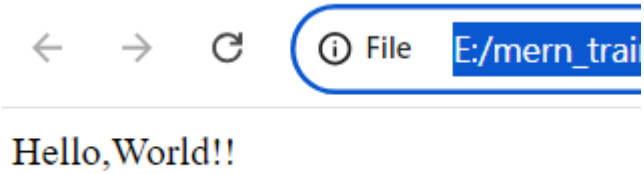
The data : { "id": 1, "name": "Leanne Graham", "username": "Bret", "email": "Sincere@april.biz", "address": { "street": "Kulas Light", "suite": "Apt. 556", "city": "Gwenborough", "zipcode": "92998-3874", "geo": { "lat": "-37.3159", "lng": "81.1496" } }, "phone": "1-770-736-8031 x56442", "website": "hildegard.org", "company": { "name": "Romaguera-Crona", "catchPhrase": "Multi-layered client-server neural-net", "bs": "harness real-time e-markets" } }

24) 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 promise1=()=>{
            return new Promise((resolve,reject)=>{
                resolve("Hello");
            })
        };
        let promise2=()=>{
            return new Promise((resolve,reject)=>{
                resolve("World!!");
            })
        }
        let myPromise=async()=>{
            let promise=await Promise.all([
```

```
                promise1().then(),
                promise2().then(),
            ]);
            document.write(promise);
        }
        myPromise();
    </script>
</body>
</html>
```

Output:



Hello,World!!

25) 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>
        let tasks=async()=>{
            let task1=async()=>{
                return new Promise((resolve)=>setTimeout(()=>resolve("Task1 is
completed"),1000));
            };
            let task2=async()=>{
                return new Promise((resolve)=>setTimeout(()=>resolve("Task2 is
completed"),2000));
            };
            let task3=async()=>{
                return new Promise((resolve)=>setTimeout(()=>resolve("Task3 is
completed"),3000));
            };
            try{
                let myPromise=await Promise.all([task1(),task2(),task3()]);
                document.write("All tasks are completed: ",myPromise);
            }
            catch(error){
                document.write("Error: ",error);
```

```
            }
        }
        tasks();

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

Output: