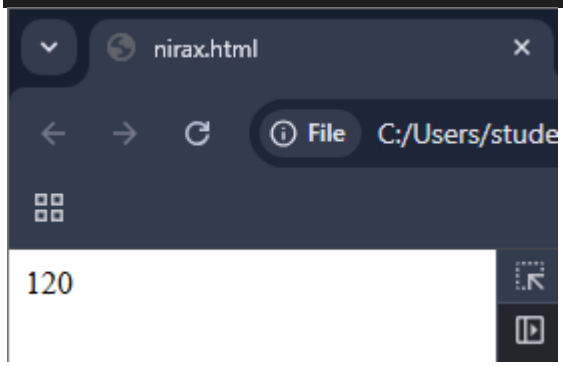
Recursion and stack:

Task 1:

Implement a function to calculate the factorial of a number using recursion.

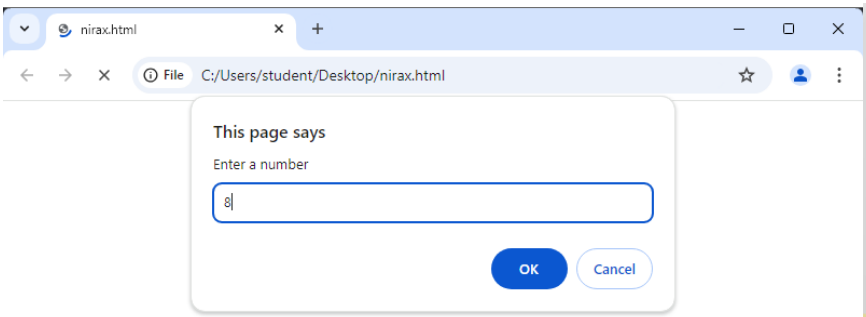
```
<html>
   <body>
    <script>
        function fun(x){
            if(x===0)return 1;
            return x*fun(x-1);
        }
        document.writeln(fun(5));
    </script>
   </body>
</html>
```

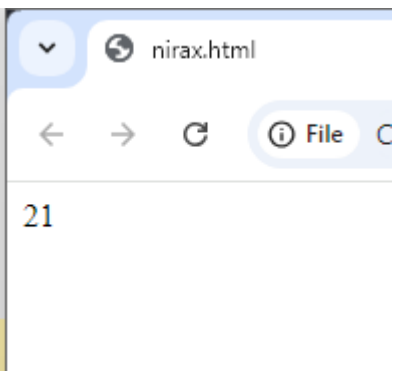


Task 2:

Write a recursive function to find the nth Fibonacci number.

```
<html>
   <body>
    <script>
        function fibo(n){
            if(n <= 1)
                return n;
            return fibo(n-1)+fibo(n-2);
        }
        let n=parseInt(prompt("Enter a number"));
        document.writeln(fibo(n))

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

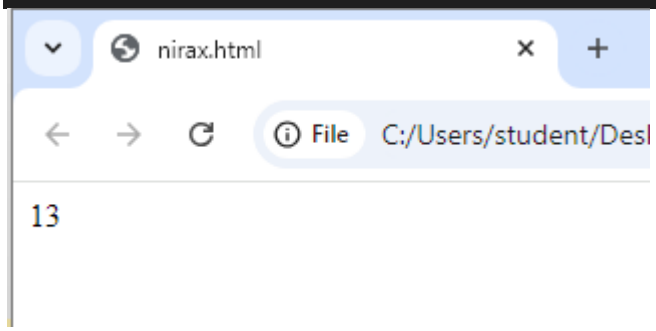




Task 3:

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.

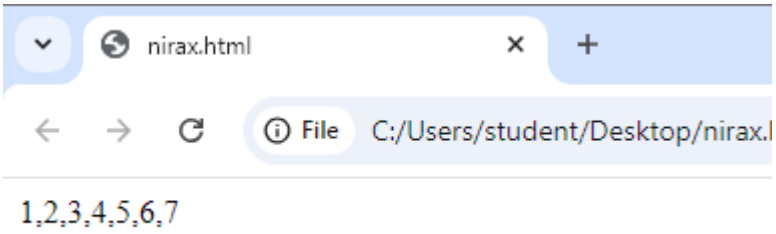
```
<html>
   <body>
    <script>
        function climb(x){
            if(x===0) return 1;
            if(x< 0) return 0;
            return climb(x-1)+climb(x-2)+climb(x-3);
        }
        document.writeln(climb(5));
    </script>
   </body>
</html>
```



Task 4:

Write a recursive function to flatten a nested array structure.

```
<html>
   <body>
    <script>
        function flatten(arr) {
            let result = [];

            for (let element of arr) {
              if (Array.isArray(element)) {
                result = result.concat(flatten(element));
              } else {
                result.push(element);
              }
            }

            return result;
        }
        document.writeln(flatten([1,2,[3,4,5],6,7]));
    </script>
   </body>
</html>
```

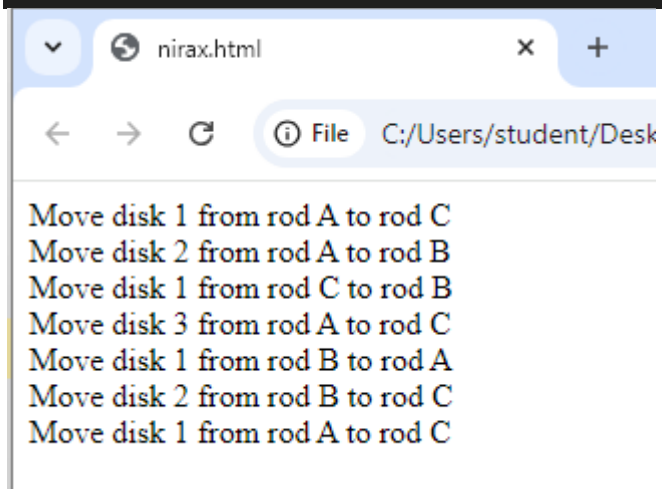


1,2,3,4,5,6,7

Task 5:

Implement the recursive Tower of Hanoi solution.

```
<html>
   <body>
    <script>
        function towerOfHanoi(n, from_rod,  to_rod,  aux_rod)
        {
        if (n == 0)
        {
            return;
        }
        towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
        document.write("Move disk " + n + " from rod " + from_rod +
        " to rod " + to_rod+"<br/>");
        towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
    }
    var N = 3;
    towerOfHanoi(N, 'A', 'C', 'B');
</script>
   </body>
</html>
```

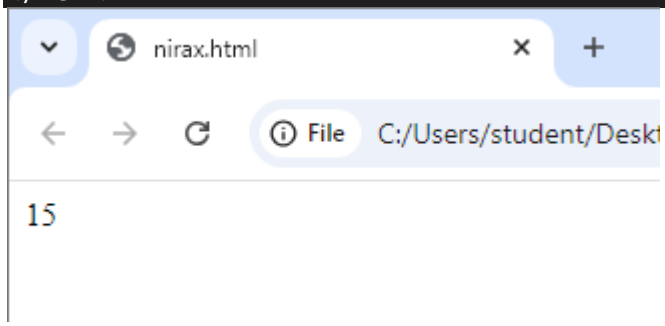


JSON and variable length arguments/spread syntax:

Task 1:

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

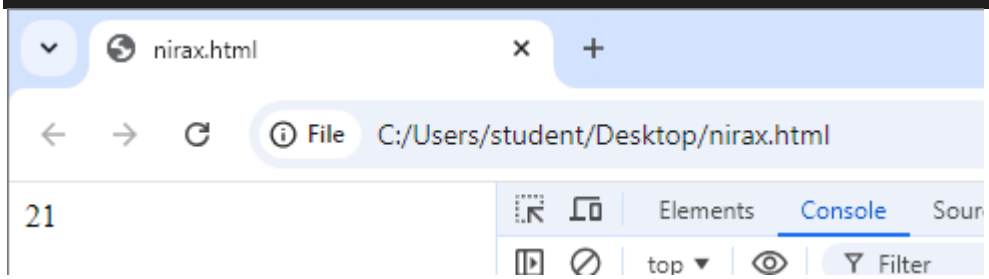
```
<html>
   <body>
    <script>
        function add(...arr){
            return arr.reduce((sum,cur)=>sum+cur,0);
        }
        document.writeln(add(1,2,3,4,5));
</script>
   </body>
</html>
```



Task 2:

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

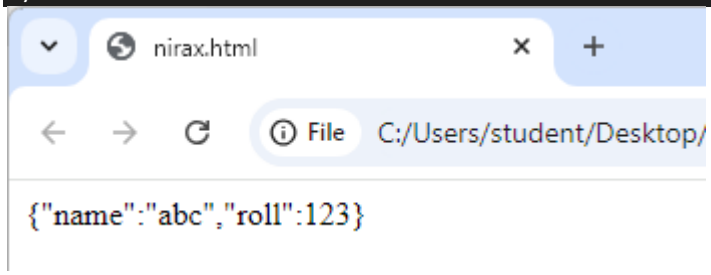
```
<html>
   <body>
    <script>
        function add(...arr){
            return arr.reduce((sum,cur)=>sum+cur,0);
        }
        function num(nums){
            return add(...nums);
        }
        document.writeln(num([1,2,3,4,5,6]));
</script>
   </body>
</html>
```



Task 3:

Create a deep clone of an object using JSON methods

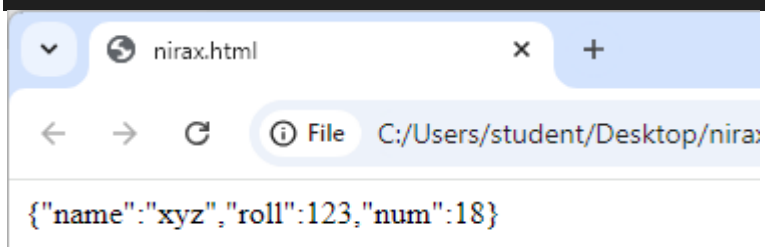
```
<html>
   <body>
    <script>
        const std={
            name:"abc",
            roll:123
        };
        const p={...std};
        document.writeln(JSON.stringify(p));
    </script>
   </body>
</html>
```



Task 4:

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

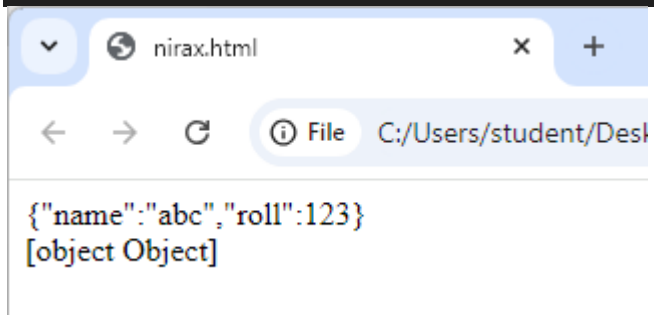
```
<<html>
   <body>
    <script>
        const std={
            name:"abc",
            roll:123
        };
        const player={
            name:"xyz",
            num:18
        }
        const p={...std,...player};
        document.writeln(JSON.stringify(p));
    </script>
   </body>
</html>
```



Task 5:

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

```
<html>
   <body>
    <script>
        const std={
            name:"abc",
            roll:123
        };
        const p=JSON.stringify({...std});
        document.writeln(p+"<br>");
        const x=JSON.parse(p);
        document.writeln(x);
    </script>
   </body>
</html>
```

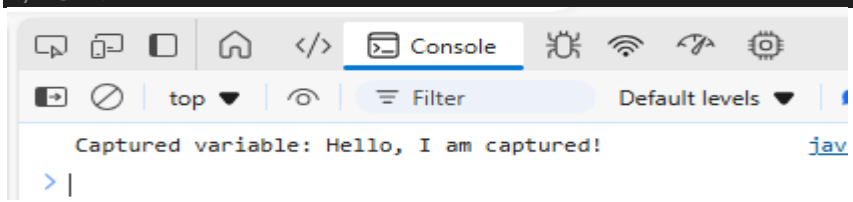


Closure:

Task 1:

Create a function that returns another function, capturing a local variable.

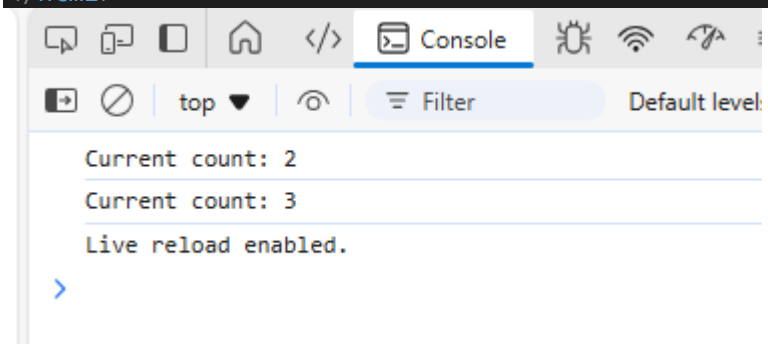
```
<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
</head>
<body>
    <script>
        function outer(outerVar) {
        return function inner() {
        console.log(`Captured variable: ${outerVar}`);
    };
}
const closure = outer("Hello, I am captured!");
closure();

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



Task 2:

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

```
<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
</head>
<body>
    <script>
        function createCounter() {
    let count = 0;
    return {
        increment: function() {
            count++;
        },
        getCount: function() {
            console.log(`Current count: ${count}`);
        }
    };
}

const counter = createCounter();
counter.increment();
counter.increment();
counter.getCount();

counter.increment();
counter.getCount();

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

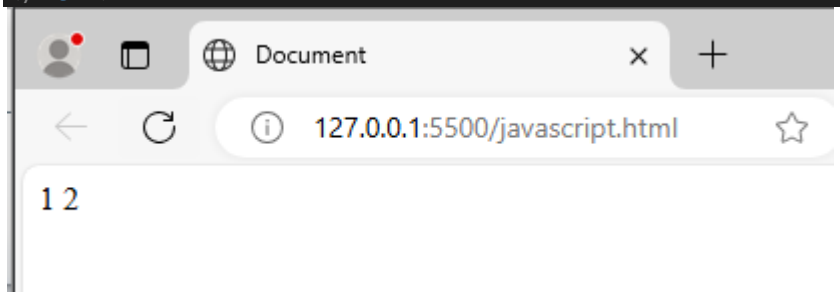


Task 3:

Write a function to create multiple counters, each with its own separate count

```
<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
</head>
<body>
    <script>
        function counter(){
            let cnt=0;
            return {
                inc:function(){
```

```
                cnt++;
            },
            getcnt:function (){
                return cnt;
            }
        };
    }
    const c1=counter();
    const c2=counter();

    c1.inc();
    document.writeln(c1.getcnt());
    c2.inc();
    c2.inc();
    document.writeln(c2.getcnt());

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

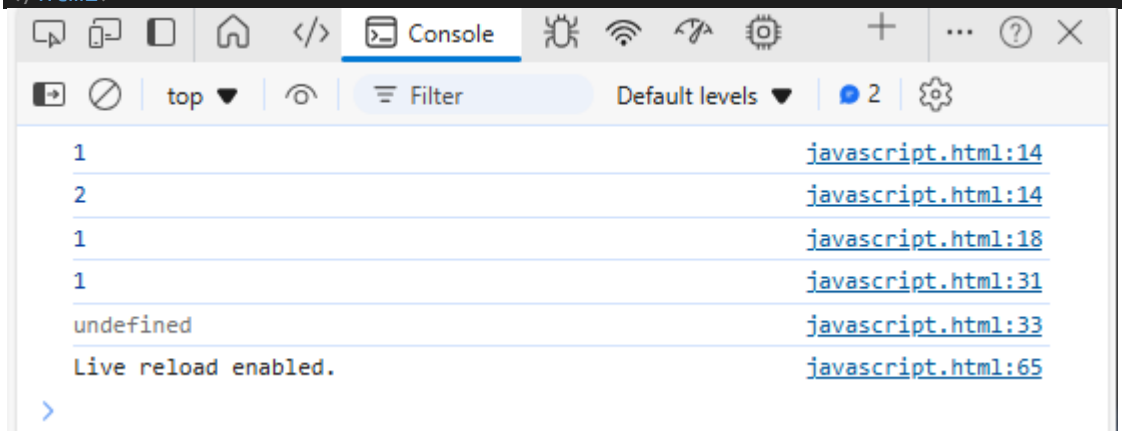


Task 4:

Use closures to create private variables within a function.

```
<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
</head>
<body>
    <script>
        function createCounter() {
  let count = 0;

  return {
    increment: function() {
      count++;
      console.log(count);
    },
    decrement: function() {
      count--;
      console.log(count);
    },
    getCount: function() {
      return count;
    }
  };
}
```

```
const counter = createCounter();

counter.increment();
counter.increment();
counter.decrement();
console.log(counter.getCount());

console.log(counter.count);



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

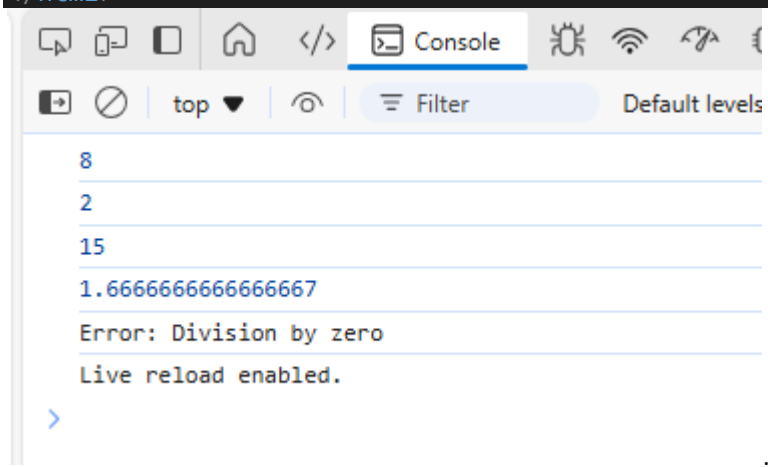


Task 5:

Build a function factory that generates functions based on some input using closures

```
<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
</head>
<body>
    <script>
        function createOperation(operator) {
  return function(a, b) {
    switch (operator) {
      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 operator';
    }
  };
}
```

```
const add = createOperation('add');
const subtract = createOperation('subtract');
const multiply = createOperation('multiply');
const divide = createOperation('divide');

console.log(add(5, 3));
console.log(subtract(5, 3));
console.log(multiply(5, 3));
console.log(divide(5, 3));
console.log(divide(5, 0));


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

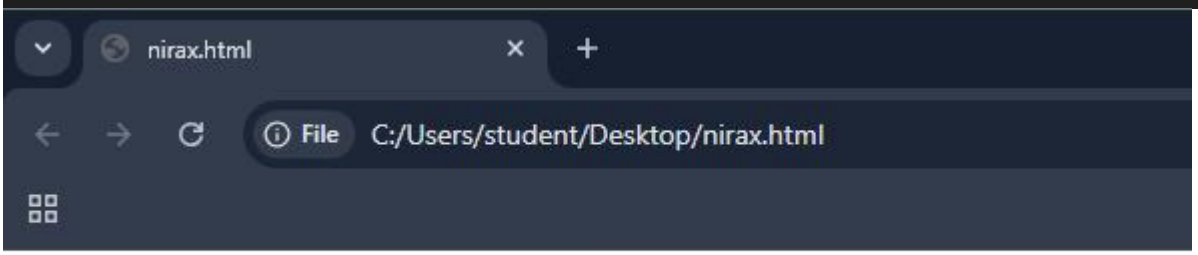


Promise, Promises chaining:

Task 1:

Create a new promise that resolves after a set number of seconds and returns a greeting

```
<html>
   <body>
    <script>
        function greetings(){
            return new Promise(()=>{
                setTimeout(()=>{
                    document.writeln("Good afternoon");
                },3000)
            })
        }
        let gd1=greetings();
    </script>
   </body>
</html>
```

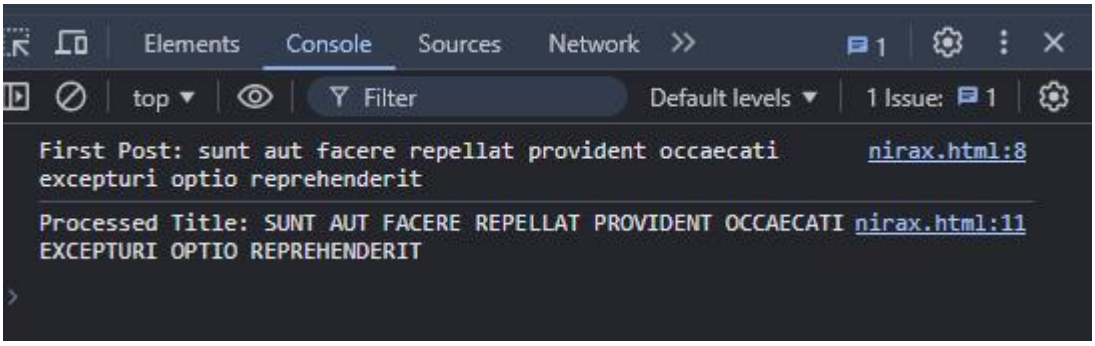


Good afternoon

Task 2:

Fetch data from an API using promises, and then chain another promise to process this data

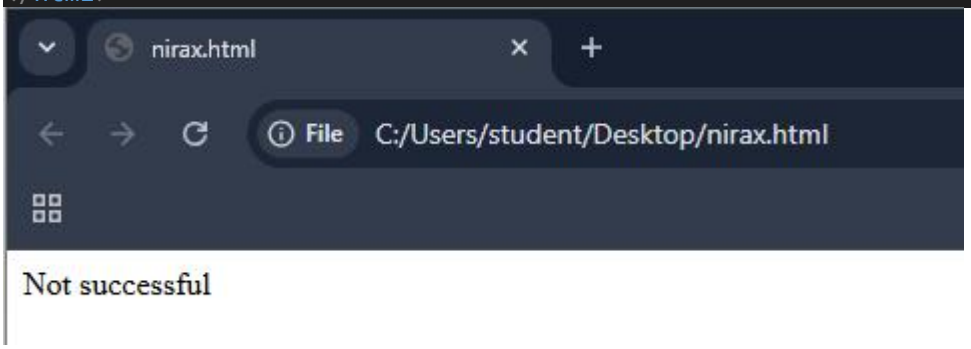
```
<html>
   <body>
    <script>
        fetch('https://jsonplaceholder.typicode.com/posts')
        .then(response => response.json())
        .then(data => {
        const firstPost = data[0];
        console.log('First Post:', firstPost.title);
        return firstPost.title.toUpperCase();  })
        .then(upperCaseTitle => {
        console.log('Processed Title:', upperCaseTitle);
        })
        .catch(error => {
        console.error('Error fetching data:', error);
        });

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



Task 3:

Create a promise that either resolves or rejects based on a random number.

```
<html>
   <body>
    <script>
        let prom=new Promise((resolve,reject)=>{
            const x=0;
            if(x==0) resolve("Success");
            else resolve("Not successful");
        });
        prom.then(result=>{
            document.writeln(result);
        })
        .catch(error=>{
            document.writelnerror(error);
        });

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

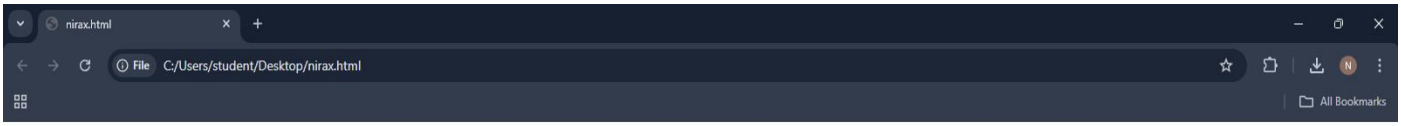


Task 4:

Use Promise.all to fetch multiple resources in parallel from an API.

```
<html>
   <body>
    <script>
        const urls = [
  'https://jsonplaceholder.typicode.com/todos/1',
  'https://jsonplaceholder.typicode.com/todos/2',
  'https://jsonplaceholder.typicode.com/todos/3'
];

function fetchData(url) {
  return fetch(url)
```

```
    .then(response => {
      if (!response.ok) {
        throw new Error(`HTTP error! Status: ${response.status}`);
      }
      return response.json();
    })
    .catch(error => {
      document.writeln(`Error fetching ${url}:`, error);
      throw error;
    });
}
Promise.all(urls.map(fetchData))
  .then(results => {
    document.writeln('All resources fetched:<br>', JSON.stringify(results));
  })
  .catch(error => {
    document.writeln('Error fetching resources:<br>', JSON.stringify(error));
  });
    </script>
   </body>
</html>
```



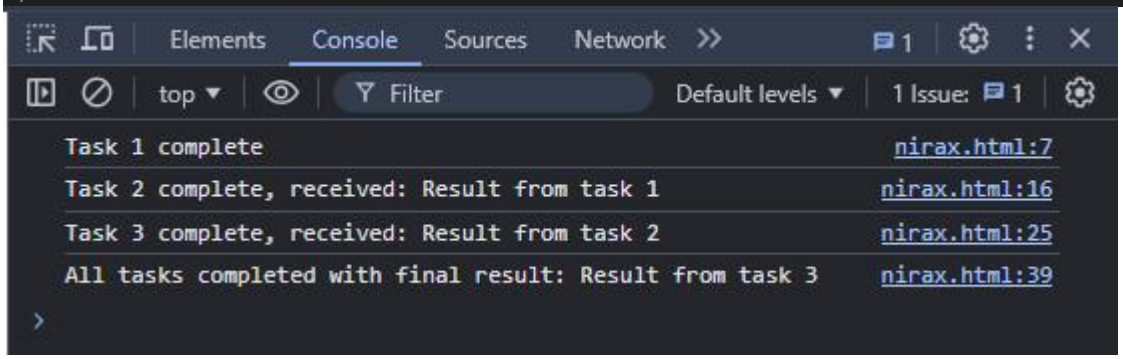
All resources fetched:
[{"userId":1,"id":1,"title":"delectus aut autem","completed":false},{"userId":1,"id":2,"title":"quis ut nam facilis et officia qui","completed":false},{"userId":1,"id":3,"title":"fugiat veniam minus","completed":false}]

Task 5:

Chain multiple promises to perform a series of asynchronous actions in sequence.

```
<html>
   <body>
    <script>
        function task1() {
            return new Promise((resolve, reject) => {
              setTimeout(() => {
                console.log("Task 1 complete");
                resolve("Result from task 1");
              }, 1000);
            });
          }

          function task2(resultFromTask1) {
            return new Promise((resolve, reject) => {
              setTimeout(() => {
                console.log("Task 2 complete, received:", resultFromTask1);
                resolve("Result from task 2");
              }, 1000);
            });
          }

          function task3(resultFromTask2) {
            return new Promise((resolve, reject) => {
              setTimeout(() => {
```

```
            console.log("Task 3 complete, received:", resultFromTask2);
            resolve("Result from task 3");
          }, 1000);
        });
      }

      task1()
        .then(result => {
          return task2(result);  // Pass result from task1 to task2
        })
        .then(result => {
          return task3(result);  // Pass result from task2 to task3
        })
        .then(result => {
          console.log("All tasks completed with final result:", result);
        })
        .catch(error => {
          console.error("An error occurred:", error);
        });

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



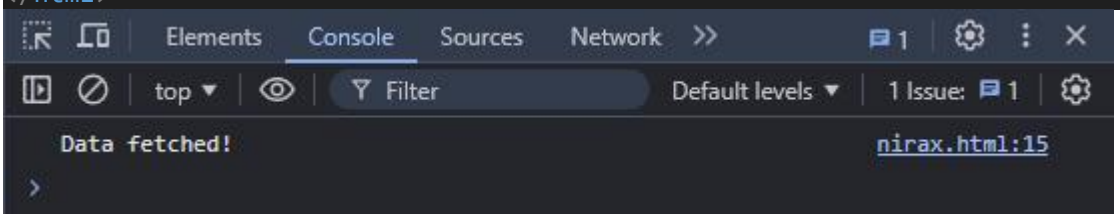
Async/await:

Task 1:

Rewrite a promise-based function using async/await

```
<html>
  <body>
    <script>
        async function fetchData() {
            return new Promise((resolve, reject) => {
              setTimeout(() => {
                resolve('Data fetched!');
              }, 2000);
            });
          }

          async function getData() {
            try {
              const result = await fetchData();
              console.log(result);
```

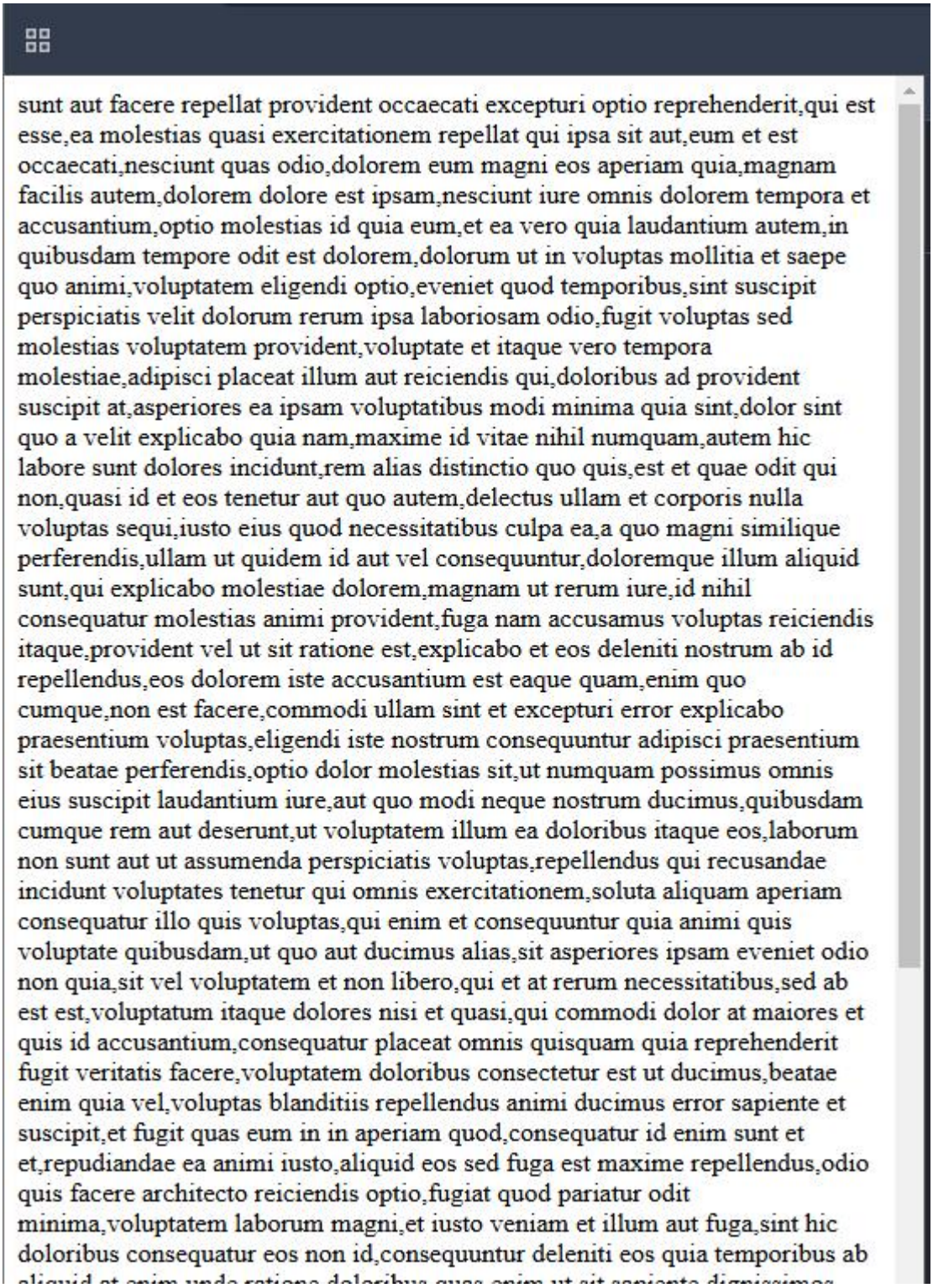
```
            } catch (error) {
              console.error(error);
            }
          }

          getData();


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



Task 2:

Create an async function that fetches data from an API and processes it.

```
<html>
   <body>
    <script>
        async function fetchAndProcessData() {
            try {
              const response = await fetch('https://jsonplaceholder.typicode.com/posts');
              const data = await response.json();
              const processedData = data.map(post => post.title); // Example processing:
extracting titles
              document.writeln(processedData);
            } catch (error) {
              document.writeln('Error fetching data:', error);
            }
          }

          fetchAndProcessData();
    </script>
   </body>
</html>
```



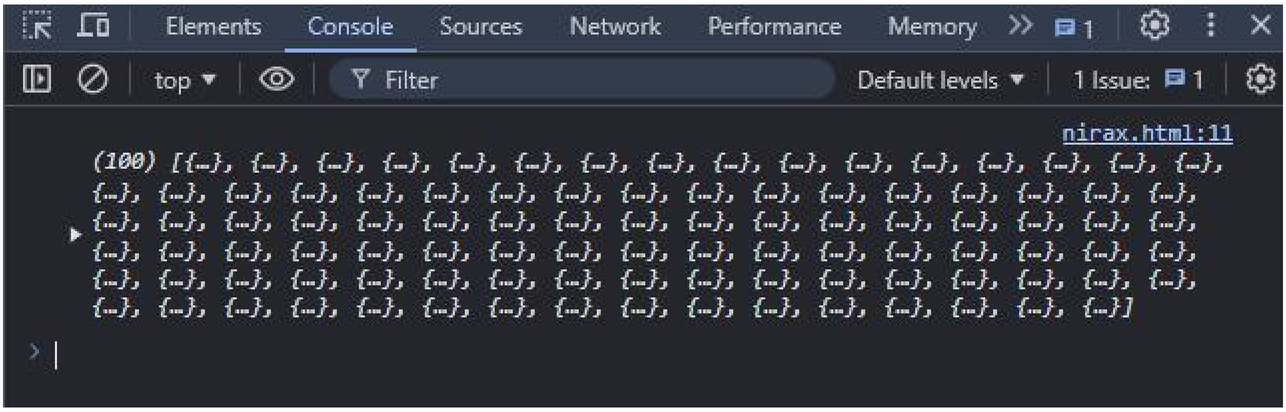
sunt aut facere repellat provident occaecati excepturi optio reprehenderit,qui est esse,ea molestias quasi exercitationem repellat qui ipsa sit aut,eum et est occaecati,nesciunt quas odio,dolorem eum magni eos aperiam quia,magnam facilis autem,dolorem dolore est ipsam,nesciunt iure omnis dolorem tempora et accusantium,optio molestias id quia eum,et ea vero quia laudantium autem,in quibusdam tempore odit est dolorem,dolorum ut in voluptas mollitia et saepe quo animi,voluptatem eligendi optio,eveniet quod temporibus,sint suscipit perspiciatis velit dolorum rerum ipsa laboriosam odio,fugit voluptas sed molestias voluptatem provident,voluptate et itaque vero tempora molestiae,adipisci placeat illum aut reiciendis qui,doloribus ad provident suscipit at,asperiores ea ipsam voluptatibus modi minima quia sint,dolor sint quo a velit explicabo quia nam,maxime id vitae nihil numquam,autem hic labore sunt dolores incidunt,rem alias distinctio quo quis,est et quae odit qui non,quasi id et eos tenetur aut quo autem,delectus ullam et corporis nulla voluptas sequi,iusto eius quod necessitatibus culpa ea,a quo magni similique perferendis,ullam ut quidem id aut vel consequuntur,doloremque illum aliquid sunt,qui explicabo molestiae dolorem,magnam ut rerum iure,id nihil consequatur molestias animi provident,fuga nam accusamus voluptas reiciendis itaque,provident vel ut sit ratione est,explicabo et eos deleniti nostrum ab id repellendus,eos dolorem iste accusantium est eaque quam,enim quo cumque,non est facere,commodi ullam sint et excepturi error explicabo praesentium voluptas,eligendi iste nostrum consequuntur adipisci praesentium sit beatae perferendis,optio dolor molestias sit,ut numquam possimus omnis eius suscipit laudantium iure,aut quo modi neque nostrum ducimus,quibusdam cumque rem aut deserunt,ut voluptatem illum ea doloribus itaque eos,laborum non sunt aut ut assumenda perspiciatis voluptas,repellendus qui recusandae incidunt voluptates tenetur qui omnis exercitationem,soluta aliquam aperiam consequatur illo quis voluptas,qui enim et consequuntur quia animi quis voluptate quibusdam,ut quo aut ducimus alias,sit asperiores ipsam eveniet odio non quia,sit vel voluptatem et non libero,qui et at rerum necessitatibus,sed ab est est,voluptatum itaque dolores nisi et quasi,qui commodi dolor at maiores et quis id accusantium,consequatur placeat omnis quisquam quia reprehenderit fugit veritatis facere,voluptatem doloribus consectetur est ut ducimus,beatae enim quia vel,voluptas blanditiis repellendus animi ducimus error sapiente et suscipit,et fugit quas eum in in aperiam quod,consequatur id enim sunt et et,repudiandae ea animi iusto,aliquid eos sed fuga est maxime repellendus,odio quis facere architecto reiciendis optio,fugiat quod pariatur odit minima,voluptatem laborum magni,et iusto veniam et illum aut fuga,sint hic doloribus consequatur eos non id,consequuntur deleniti eos quia temporibus ab

Task 3:

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

```
<html>
   <body>
    <script>
        async function fetchDataWithErrorHandling() {
            try {
              const response = await fetch('https://jsonplaceholder.typicode.com/posts');
              if (!response.ok) {
                throw new Error('Network response was not ok');
              }
```

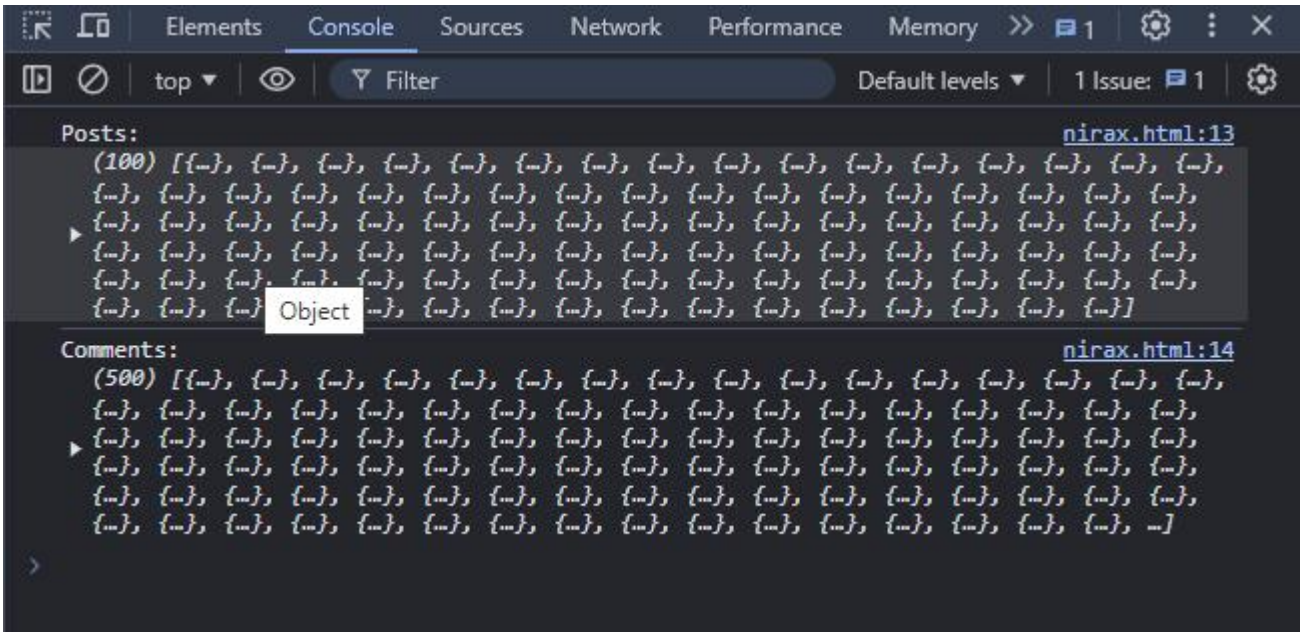
```
            const data = await response.json();
            console.log(data);
          } catch (error) {
            document.writeln('There was an error:', error.message);
          }
        }

        fetchDataWithErrorHandling();

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



Task 4:

Use async/await in combination with Promise.all.

```
<html>
   <body>
    <script>
        async function fetchMultipleResources() {
            try {
              const urls = [
                'https://jsonplaceholder.typicode.com/posts',
                'https://jsonplaceholder.typicode.com/comments'
              ];

              const [posts, comments] = await Promise.all(urls.map(url =>
fetch(url).then(response => response.json())));

              console.log('Posts:', posts);
              console.log('Comments:', comments);
            } catch (error) {
              console.error('Error fetching resources:', error);
            }
          }

        fetchMultipleResources();

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

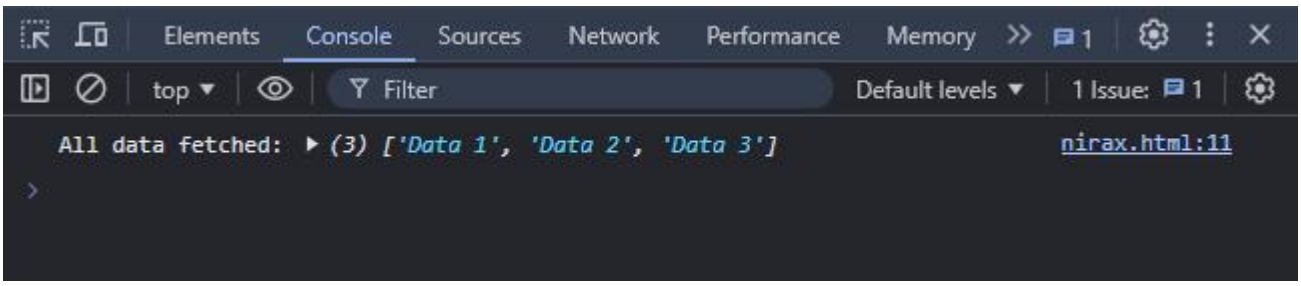


```
Posts:                                                                         nirax.html:13
▸ (100) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
Comments:                                                                      nirax.html:14
▸ (500) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, …]
>
```

Task 5:

Create an async function that waits for multiple asynchronous operations to complete before proceeding

```
<html>
   <body>
    <script>
        async function waitForAll() {
            const fetchData1 = new Promise(resolve => setTimeout(() => resolve('Data 1'),
2000));
            const fetchData2 = new Promise(resolve => setTimeout(() => resolve('Data 2'),
3000));
            const fetchData3 = new Promise(resolve => setTimeout(() => resolve('Data 3'),
1000));

            const results = await Promise.all([fetchData1, fetchData2, fetchData3]);

            console.log('All data fetched:', results);
          }

          waitForAll();
    </script>
   </body>
</html>
```



```
All data fetched: ▸ (3) ['Data 1', 'Data 2', 'Data 3']                        nirax.html:11
>
```

Modules introduction, Export and Import:

Task 1:

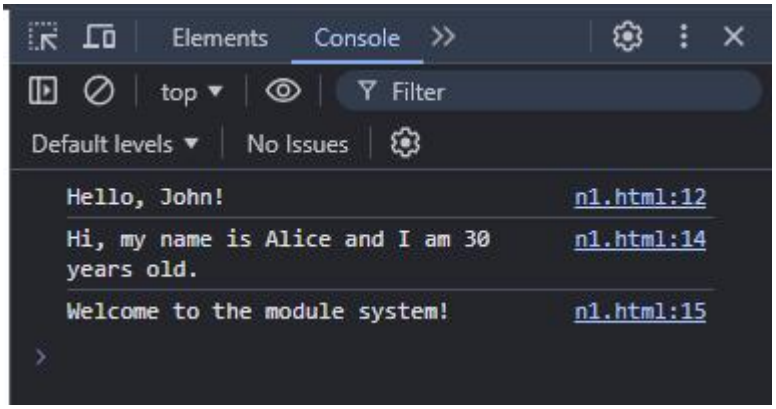
Create a module that exports a function, a class, and a variable.

```
export default function sayHello(name) {
  return `Hello, ${name}!`;
}

export class Person {
  constructor(name, age) {
      this.name = name;
      this.age = age;
  }
  greet() {
      return `Hi, my name is ${this.name} and I am ${this.age} years old.`;
  }
}
export const greetingMessage = 'Welcome to the module system!';
```

Task 2:

Import the module in another JavaScript file and use the exported entities

```
<!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 type="module">
        import sayHello, { Person, greetingMessage } from './module.js';

        console.log(sayHello('John'));
        const person = new Person('Alice', 30);
        console.log(person.greet());
        console.log(greetingMessage);
    </script>
</body>
</html>
```



Task 3:

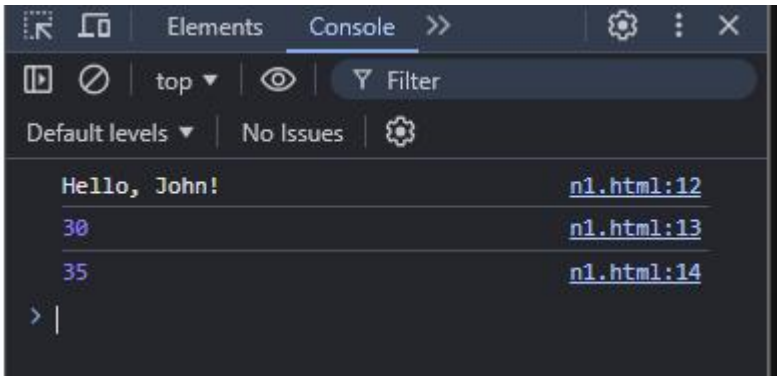
Use named exports to export multiple functions from a module.

```
export function sayHello(name) {
  return `Hello, ${name}!`;
}

export function add(a,b) {
  return a+b;
}
export function mul(a,b) {
  return a*b;
}
```

Task 4:

Use named imports to import specific functions from a module.

```
<!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 type="module">
        import {sayHello,add,mul} from './module.js';

        console.log(sayHello('John'));
        console.log(add(10,20));
        console.log(mul(5,7));
    </script>
</body>
</html>
```

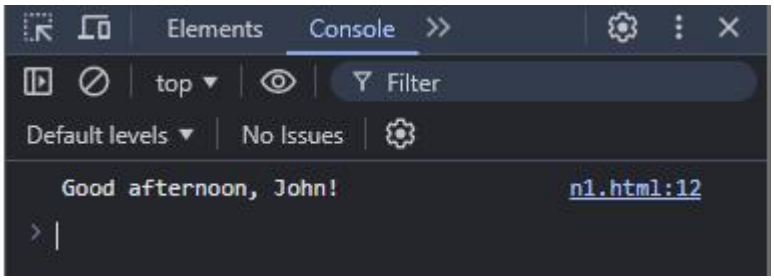


Task 5:

Use default export and import for a primary function of a module.

```
<!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 type="module">
        import Greetings from './module.js';

        console.log(Greetings('John'));
    </script>
</body>
</html>
```

```
export default function Greetings(name) {
  return `Good afternoon, ${name}!`;
}
```



Browser: DOM Basics:

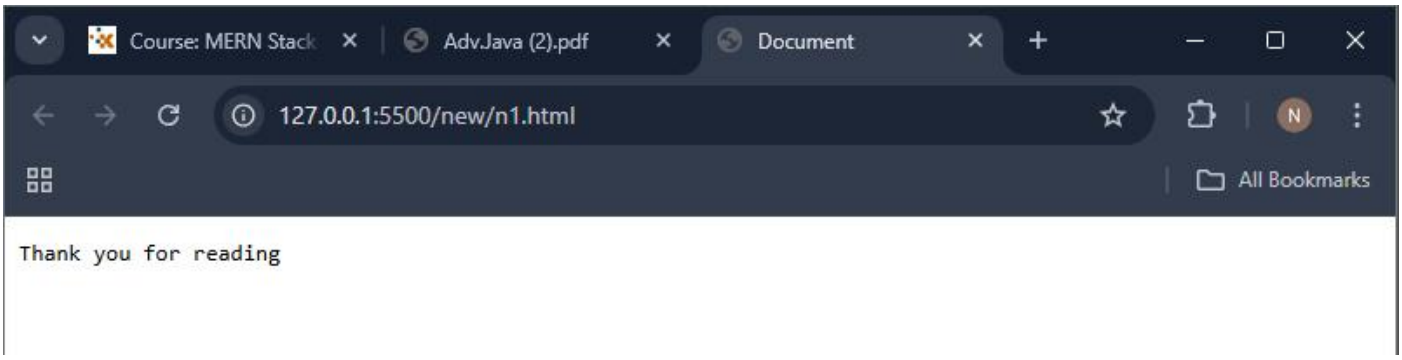
Task 1:

Select an HTML element by its ID and change its content using JavaScript.

```
<!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>
    <pre id="parah">
A paragraph is a series of related sentences developing a central idea, called the topic.
    </pre>
    <script src="module.js">
    </script>
</body>
</html>
```

Module.js file

```
document.getElementById("parah").innerHTML="Thank you for reading"
```



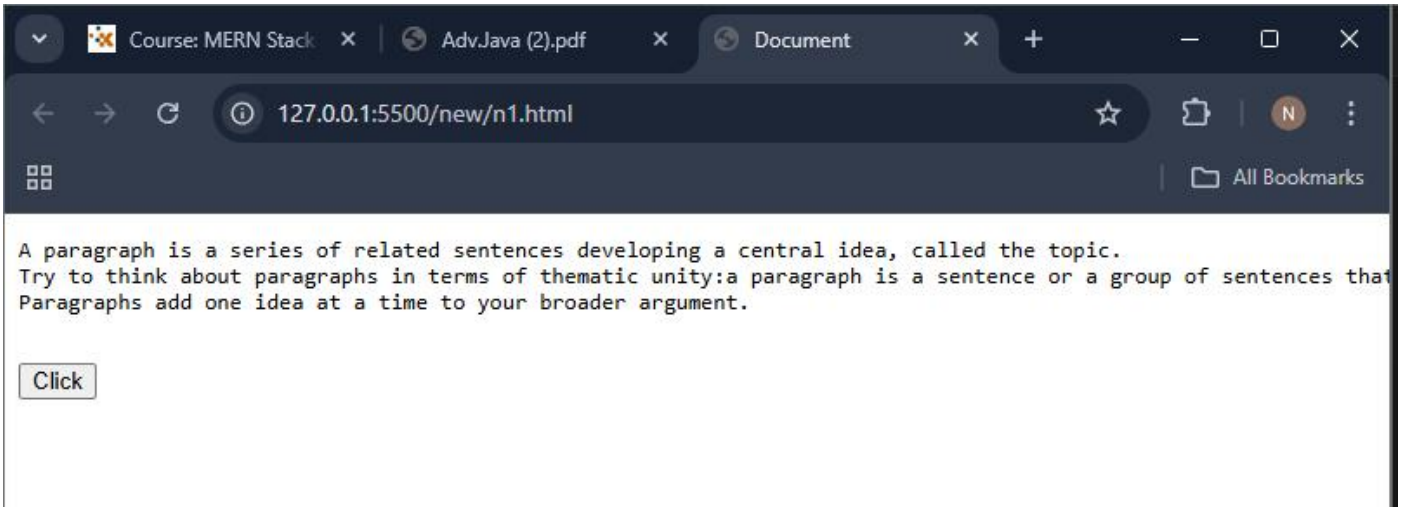
Task 2:

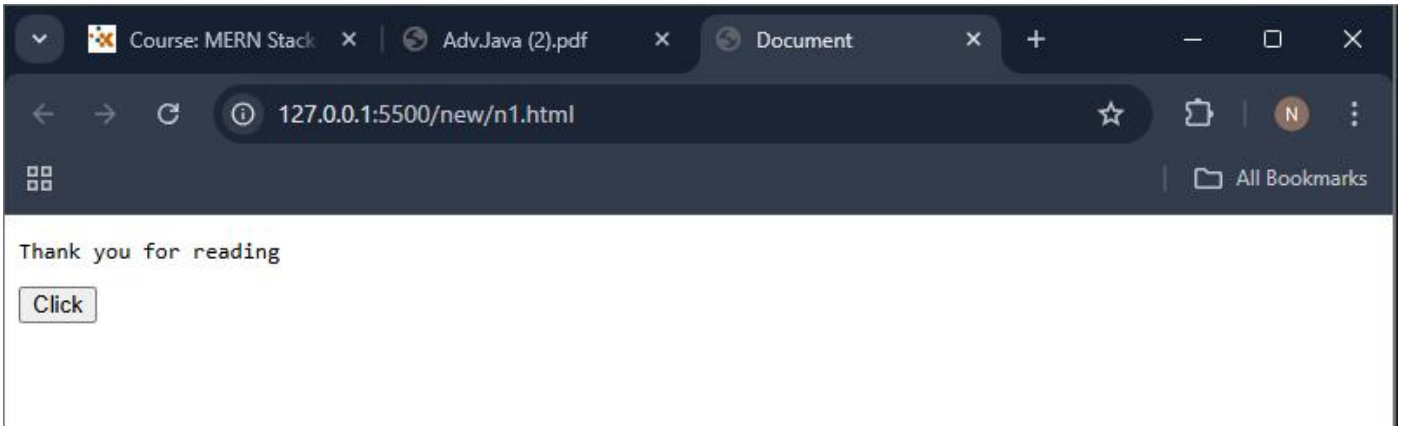
Attach an event listener to a button, making it perform an action when clicked.

```
<!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>
    <pre id="parah">
A paragraph is a series of related sentences developing a central idea, called the topic.
Try to think about paragraphs in terms of thematic unity:a paragraph is a sentence or a
group of sentences that supports one central, unified idea.
Paragraphs add one idea at a time to your broader argument.
    </pre>
    <script src="module.js">
    </script>
</body>
</html>
```

Module.js file

```
const myb=document.createElement("button");
myb.textContent="Click";
myb.onclick=()=>{
document.getElementById("parah").innerHTML="Thank you for reading"
}
document.body.append(myb);
```





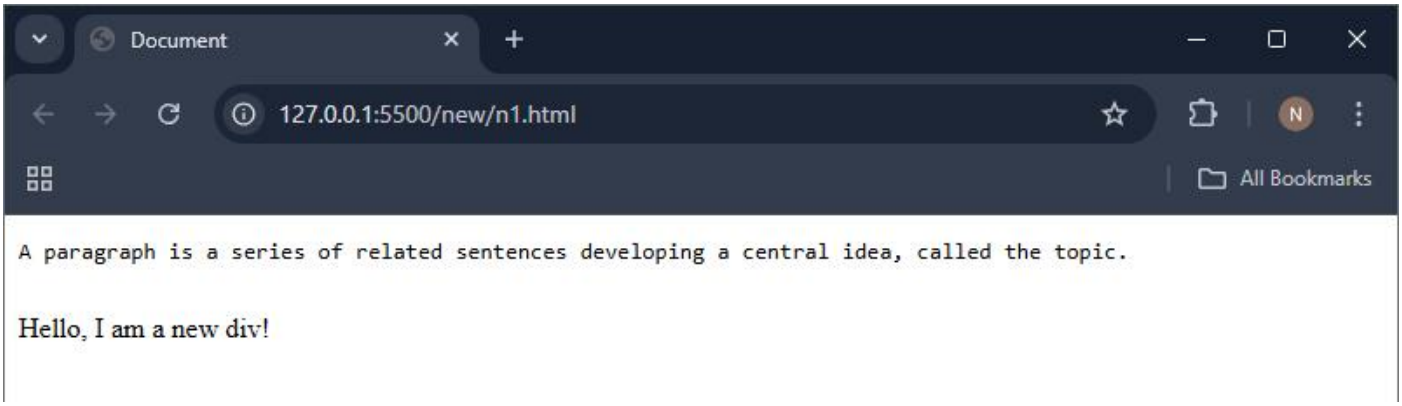
Task 3:

Create a new HTML element and append it to the DOM.

```
<!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>
    <pre id="parah">
A paragraph is a series of related sentences developing a central idea, called the topic.
    </pre>
    <script src="module.js">
    </script>
</body>
</html>
```

Module.js file

```
const newDiv = document.createElement("div");
newDiv.textContent = "Hello, I am a new div!";
document.body.appendChild(newDiv);
```



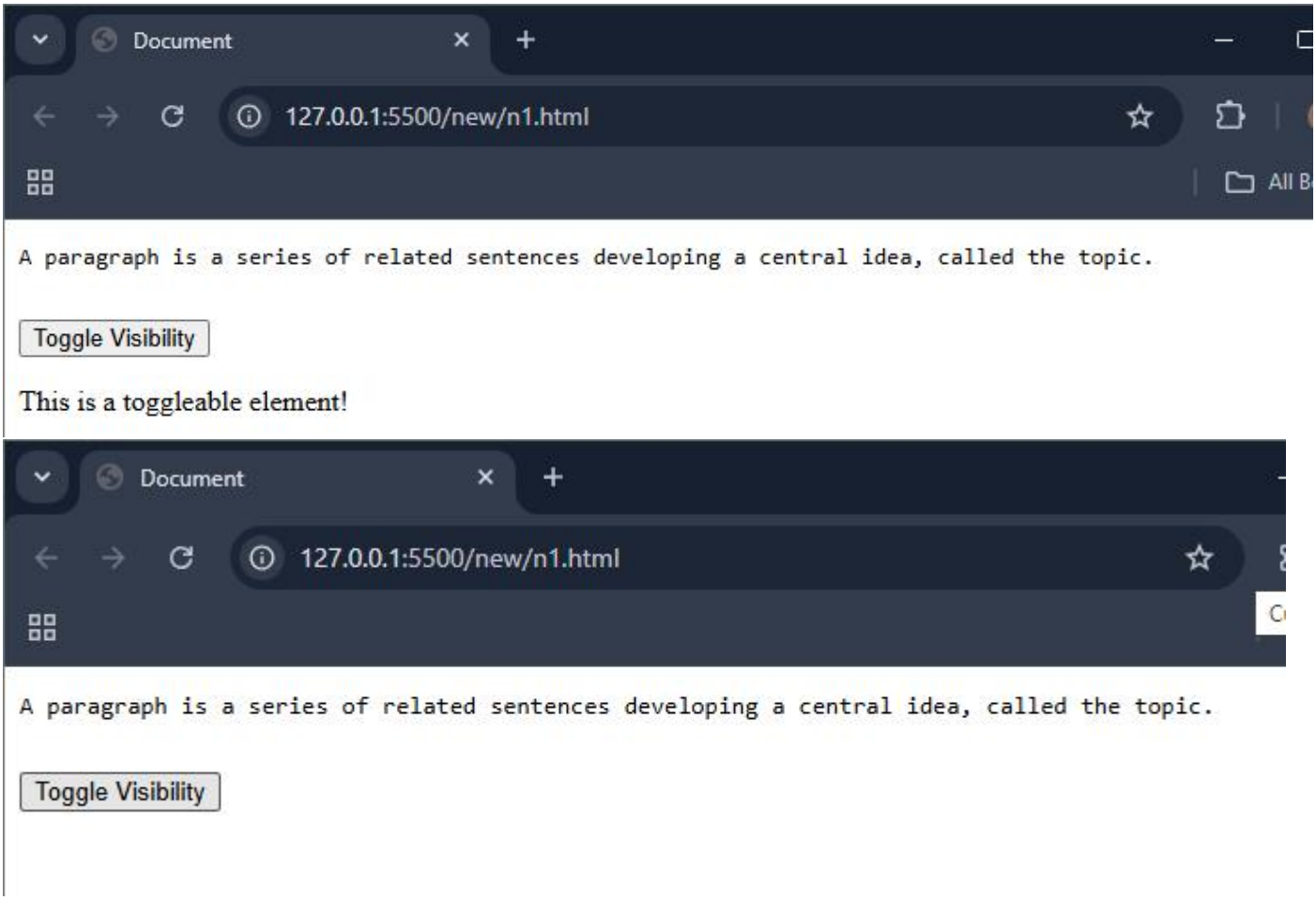
Task 4:

Implement a function to toggle the visibility of an element

```
<!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>
    <pre id="parah">
A paragraph is a series of related sentences developing a central idea, called the topic.
    </pre>
    <button onclick="toggleVisibility('myDiv')">Toggle Visibility</button>
    <div id="myDiv">
        <p>This is a toggleable element!</p>
    </div>

    <script src="module.js">

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

Module.js file

```
function toggleVisibility(elementId) {
  const element = document.getElementById(elementId);
  if (element.style.display === "none") {
      element.style.display = "block";
  } else {
      element.style.display = "none";
  }
}
```



Task 5:

Use the DOM API to retrieve and modify the attributes of an element.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM Attribute Example</title>
</head>
<body>
    <button id="myButton" class="btn" data-info="1234">Click Me</button>

    <script>
        const button = document.getElementById('myButton');
        button.addEventListener('click', function() {
            console.log('Current Class:', button.getAttribute('class'));
            if (button.getAttribute('class') === 'btn') {
                button.setAttribute('class', 'newClass');
                button.textContent = 'You clicked me!';
            } else {
                button.setAttribute('class', 'btn');
                button.textContent = 'Click Me';
            }
            console.log('Data-info:', button.getAttribute('data-info'));
            if (button.getAttribute('data-info') === '1234') {
                button.setAttribute('data-info', '5678');
            } else {
```

```
                button.setAttribute('data-info', '1234');
            }
            console.log('Updated Data-info:', button.getAttribute('data-info'));
        });
    </script>
</body>
</html>
```