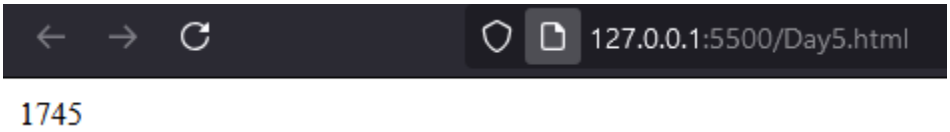
2. JSON and variable length arguments/spread syntax:

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

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Tower of Hanoi</title>
</head>
<body>
 <script>
 function task(arr) {
 {
 return arr.reduce((total,current)=>total+current,0);
 }
 }
 document.writeln(task([4,34,56,787,768,46,46,4]));
 </script>
</body>
</html>
```

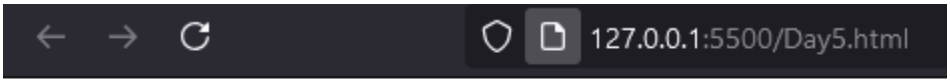
Output



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>Tower of Hanoi</title>
</head>
<body>
 <script>
 function task(...arr) {
 {return arr.reduce((total,current)=>total+current,0);} }
 document.writeln(task(4,34,56,787,768,46,46,4));
 </script>
</body></html>
```

Output

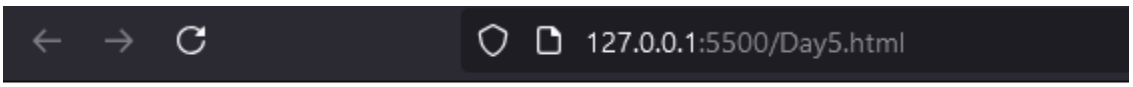


1745

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

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
</head>
<body>
    <script>
        const ob={
            name:"Thiyaneshwar",
            age:19,
            address:{
                city:"coimbatore",
            },
             h:[3,4,56,7,8]
        };
        document.writeln(JSON.stringify(ob));
        </script>
</body>
</html>
```

Output

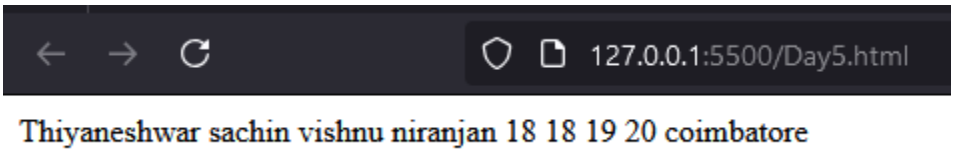


{"name":"Thiyaneshwar","age":19,"address":{"city":"coimbatore"},"h":[3,4,56,7,8]}

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

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Document</title>
</head>
<body>
   <script>
         let name=["Thiyaneshwar "," sachin "," vishnu "," niranjan "];
         var age=[" 18 "," 18 "," 19 "," 20 "];
      document.writeln(...name,...age,"coimbatore");
      </script>
</body>
</html>
```

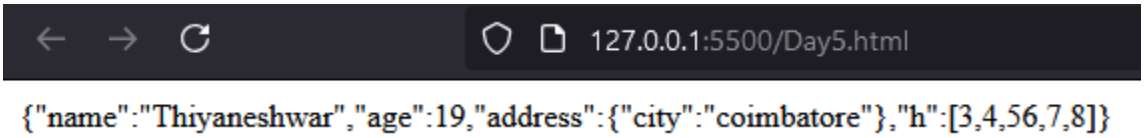
Output



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

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Document</title>
</head>
<body>
   <script>
      const ob={
         name:"Thiyaneshwar",
         age:19,
         address:{
            city:"coimbatore",
         },
          h:[3,4,56,7,8]
      };
      let clone=JSON.parse(JSON.stringify(ob));
      document.writeln(JSON.stringify(clone));</script></body></html>
```

Output



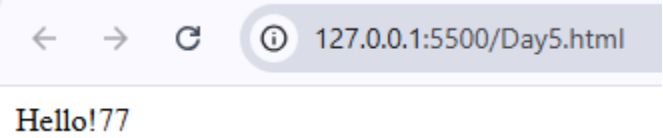
{"name":"Thiyaneshwar","age":19,"address":{"city":"coimbatore"},"h":[3,4,56,7,8]}

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>tasks</title>
</head>
<body>
 <script>
 function task() {
 {
   let c=77;
 return {
   implement:function ()
   {
      document.writeln("Hello!"+c);
   }
 }
 }
 }
 task().implement();
 </script>
</body>
</html>
```

Output

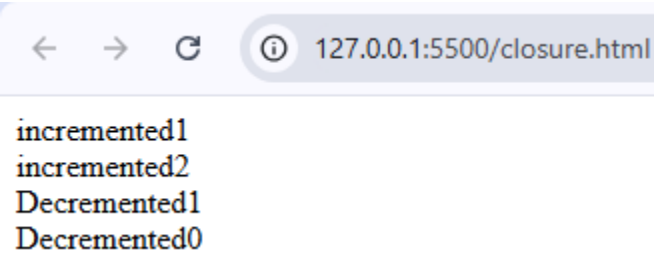


Hello!77

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

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Closure</title>
</head>
<body>
   <script>
   function count() {
      let x =0;
     return {
     increment: function(){
         x++;
         document.writeln("incremented"+x);
      },
      decrement:function() {
         x--;
         document.writeln("Decremented"+x);
      }
   };}
      const inc=count();
      inc.increment();
      inc.increment();
      inc.decrement();
      inc.decrement();
</script>
</body>
</html>
```

Output



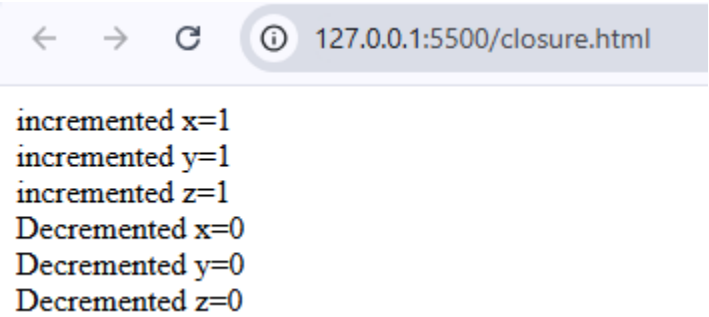
Task 3: Write a function to create multiple counters, each with its own separate count.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Closure</title>
</head>
<body>
   <script>
       function counter(){
   return {
      count:function() {
       let x =0;
      return {
      increment: function(){
           x++;
           document.writeln("incremented x="+x+"<br>");
       },
       decrement:function() {
           x--;
           document.writeln("Decremented x="+x+"<br>");
       }
    };},
   count2:function(){
      let y=0;
       return {
      increment2: function(){
           y++;
           document.writeln("incremented y="+y+"<br>");
       },
       decrement2:function() {
           y--;
           document.writeln("Decremented y="+y+"<br>");
       }
    };
    },
    count3:function() {
      let z=0;
       return {
      increment3: function(){
```

```
            z++;
            document.writeln("incremented z="+z+"<br>");
        },
        decrement3:function() {
            z--;
            document.writeln("Decremented z="+z+"<br>");
        }
    };
    }
     }

   }
        const inc=counter().count();
        const dec=counter().count();
        const inc2=counter().count2();
        const dec2=counter().count2();
        const inc3=counter().count3();
        const dec3=counter().count3();
        inc.increment();
        inc2.increment2();
        inc3.increment3();
        inc.decrement();
        inc2.decrement2();
        inc3.decrement3();
</script>
</body>
</html>
```

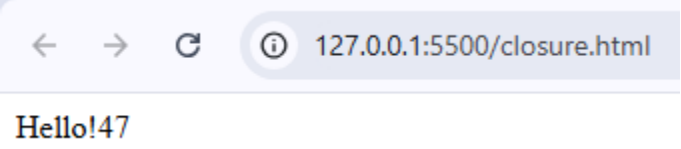
Output



Task 4: 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>tasks</title>
</head>
<body>
 <script>
 function task() {
 {
   let c=77;
 return {
   implement:function ()
   {
      let h=47;
      document.writeln("Hello!"+h);
   }
 }
 }
 }
 task().implement();
 </script>
</body>
</html>
```

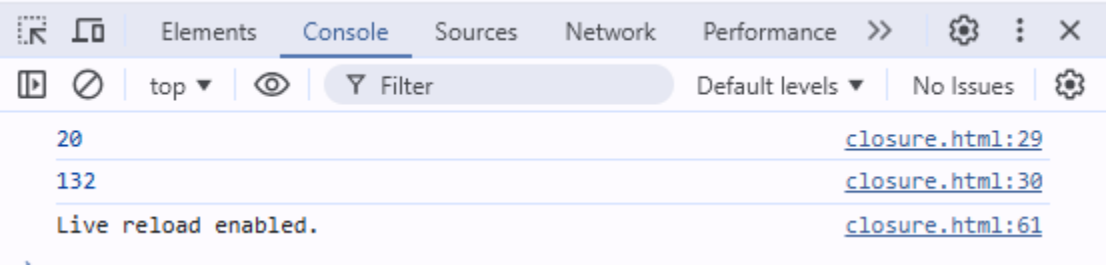
Output



Task 5: 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>JavaScript Tasks</title>
</head>
<body>
    <script>
       function createMathFunction(operator, value) {
  return function(input) {
    if (operator === 'add') {
      return input + value;
    } else if (operator === 'subtract') {
      return input - value;
    } else if (operator === 'multiply') {
      return input * value;
    } else if (operator === 'divide') {
      return input / value;
    } else {
      return input;
    }
  };
}
const add5 = createMathFunction('add', 5);
const multiplyBy3 = createMathFunction('multiply', 3);
console.log(add5(15));
console.log(multiplyBy3(44));
</script>
</body>
</html>
```

Output

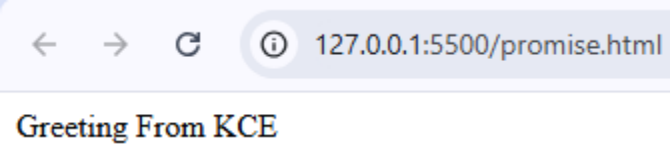


4. Promise, Promises chaining:

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>
        function myfirst()
        {
            return new Promise(()=>{
                setTimeout(() => {
                    document.writeln("Greeting From KCE");},3000)
           })
        }
        myfirst();
    </script>
</body>
</html>
```

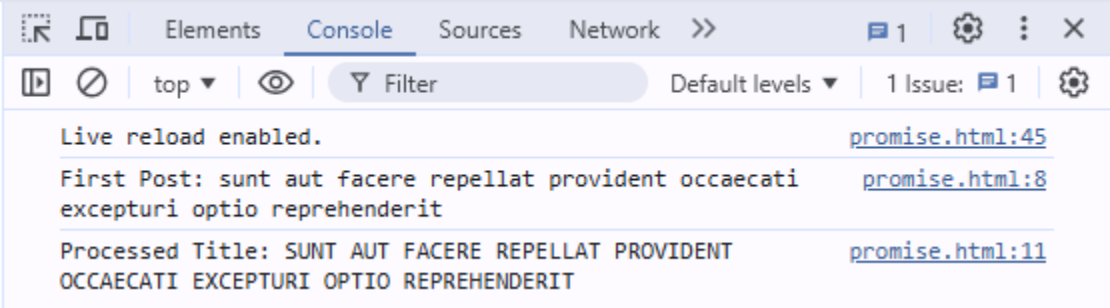
Output



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>
```

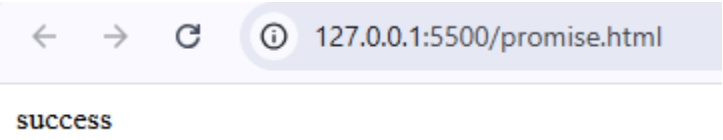
Output



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 har=new Promise((Resolve,Reject)=>
           {const x=98;
           if(x==98)
           Resolve('success');
           else Resolve('rejected')
        });
        har.then(result=>{
           document.writeln(result);
        })
        .catch(error=>
        {
           document.writeln(error);
        }
        )
    </script>
</body>
</html>
```

Output



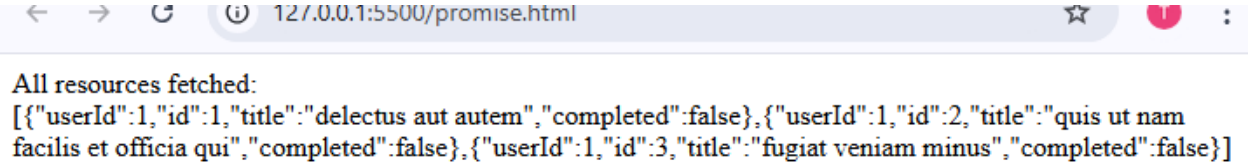
success

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>
```

Output



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}]

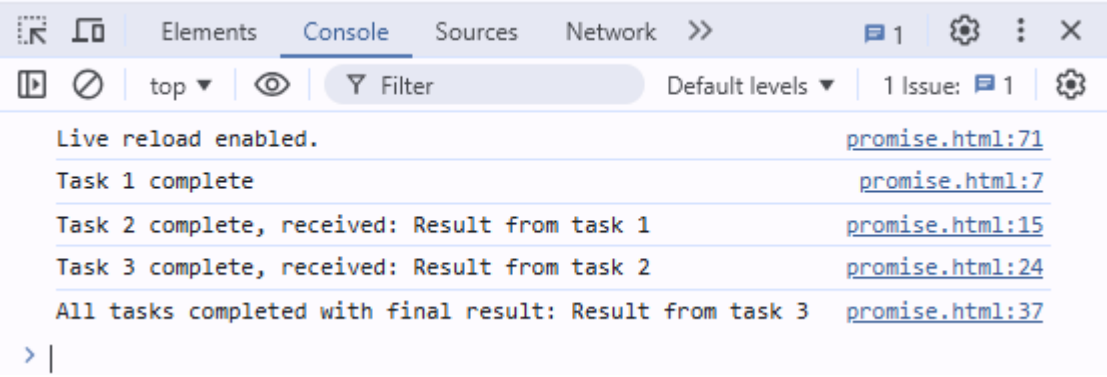
Task 5: Chain multiple promises to perform a series of asynchronous actions in sequence.

```
<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>
```

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>
      function myfirst()
      {
         return new Promise(()=>{
            setTimeout(() => {
               console.log("Hey now its you Time to Rock! ");},3000)
        })
      }
      async function first()
      {
       console.log("awaiting");
       const wait=await myfirst();
       wait();
      }
      first();
   </script>
</body>
</html>
```

Output

```
awaiting                                          promise.html:19
Live reload enabled.                              promise.html:53
Hey now its you Time to Rock!                     promise.html:14
>
```

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>
```

Output

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 aliquid at,enim unde ratione doloribus quas enim ut sit sapiente,dignissimos eum dolor ut enim et delectus in,doloremque officiis ad et non perferendis,necessitatibus quasi exercitationem odio,quam voluptatibus rerum veritatis,pariatur consequatur quia magnam autem omnis non amet,labore in ex et explicabo corporis aut quas,tempora rem veritatis voluptas quo dolores vero,laudantium voluptate suscipit sunt enim enim,odit et voluptates doloribus alias odio et,optio ipsam molestias necessitatibus occaecati facilis veritatis dolores aut,dolore veritatis porro provident adipisci blanditiis et sunt,placeat quia et porro iste,nostrum quis quasi placeat,sapiente omnis fugit eos,sint soluta et vel magnam aut ut sed qui,ad iusto omnis odit dolor voluptatibus,aut amet sed,ratione ex tenetur perferendis,beatae soluta recusandae,qui qui voluptates illo iste minima,id minus libero illum nam ad officiis,quaerat velit veniam amet cupiditate aut numquam ut sequi,quas fugiat ut perspiciatis vero provident,laboriosam dolor voluptates,temporibus sit alias delectus eligendi possimus magni,at nam consequatur ea labore ea harum

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>
```

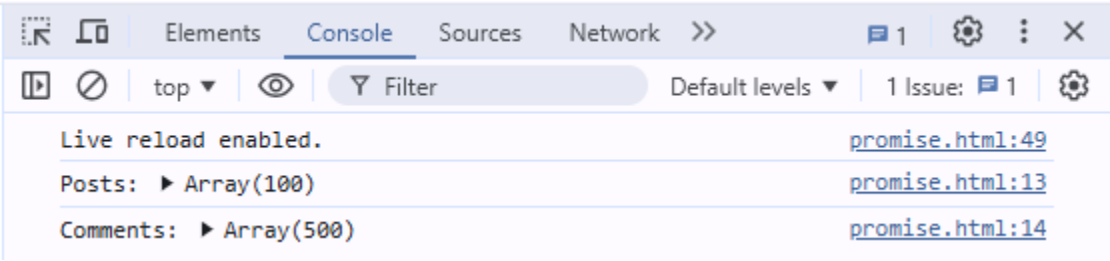
Output

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>
```

Output

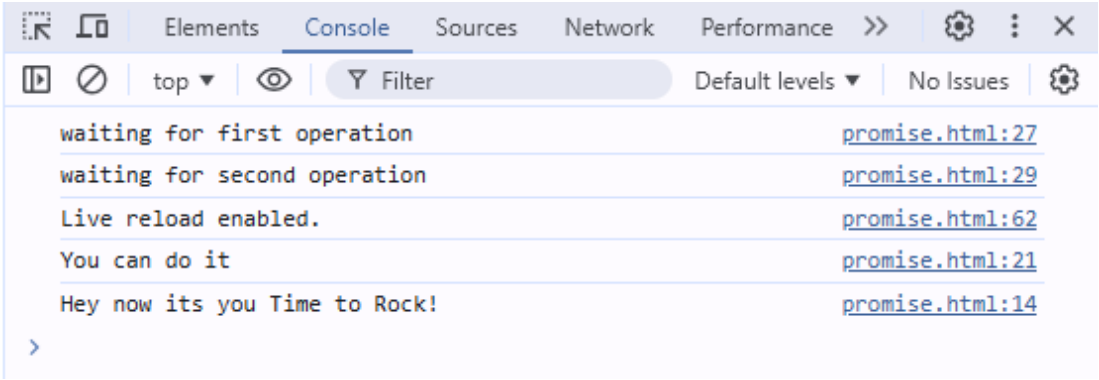


Task 5: Create an async function that waits for multiple asynchronous operations to complete before proceeding.

```
<!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>
      function myfirst()
      {
         return new Promise((resolve)=>{
            setTimeout(() => {
               console.log("Hey now its you Time to Rock! ");},3000)
        })
      }
      function micro()
      {
         return new Promise((resolve)=>{
            setTimeout(()=>{
               console.log("You can do it");},2000)
            })
         }
      async function first()
      {
       console.log("waiting for first operation");
       myfirst();
       console.log("waiting for second operation")
      micro();
      }
      first();
   </script>
</body>
</html>
```

Output



6. Modules introduction, Export and Import:

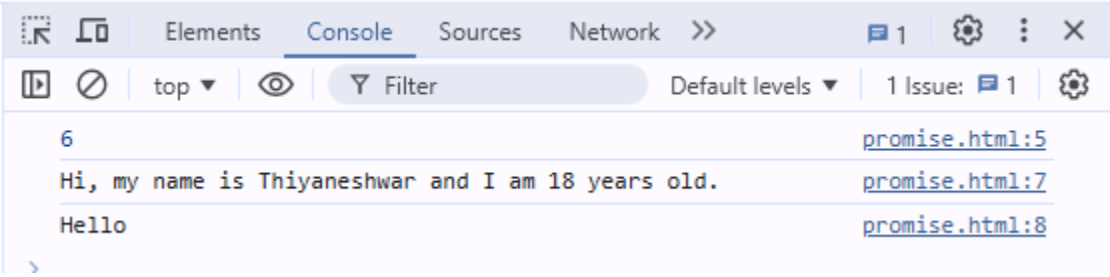
Task 1: Create a module that exports a function, a class, and a variable.

```
export default function add(a,b)
{
   return a+b;
}
export default class person
{
   name="Thiyaneshwar";
   age=18;
   display() {
      return `Hi, my name is ${this.name} and I am ${this.age} years old.`;
}}
export let f="Hello";
```

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

```
<html>
<body>
<script type="module">
import add,{person,f} from './exportfile.js'
console.log(add(2,4));
const a=new person();
console.log(a.display());
console.log(f);
</script>
</body>
</html>
```

Output



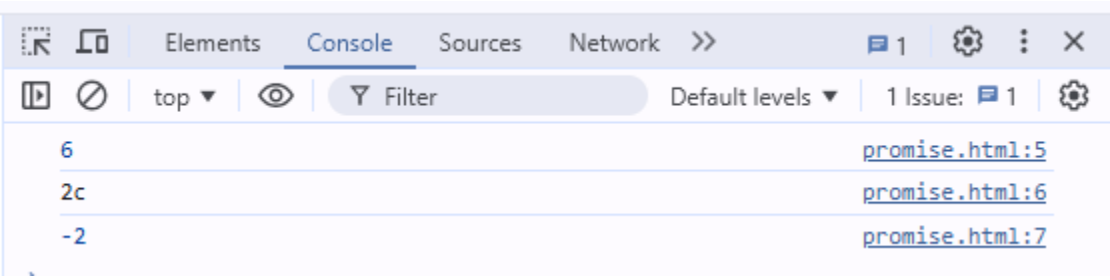
Task 3: Use named exports to export multiple functions from a module

```
export default function temperature(a)
{
   return a;
}
export default function add(a,b)
{
   return a+b;
}
export default function sub(a,b)
{
   return a-b;}
```

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

```
<html>
<body>
<script type="module">
import {add,temperature,sub} from './exportfile.js'
console.log(add(2,4));
console.log(temperature(2)+"c");
console.log(sub(2,4));
</script>
</body>
</html>
```

Output



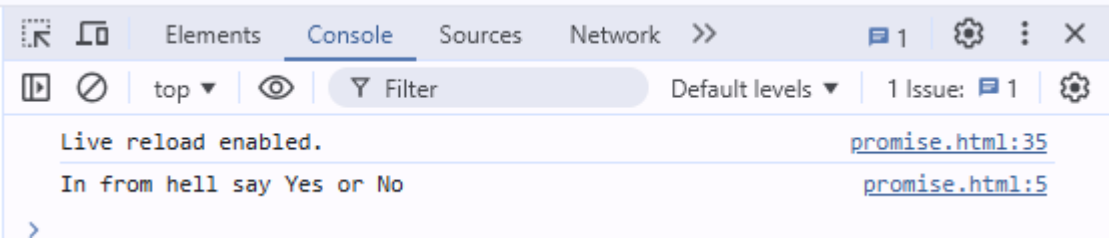
Task 5: Use default export and import for a primary function of a module.

```
<html>
<body>
<script type="module">
import greeting from './exportfile.js'
console.log(greeting());
</script>
</body>
</html>
```

exportfile

```
export default function greeting(){
   return "In from hell say Yes or No";
}
```

Output

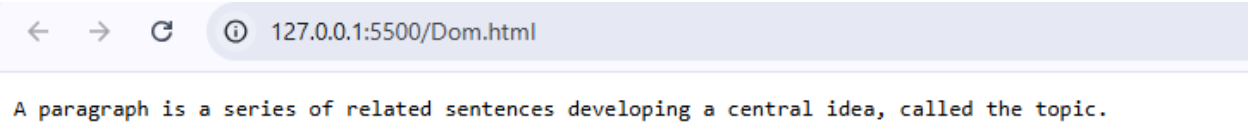


7. Browser: DOM Basics:

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>
```

Output



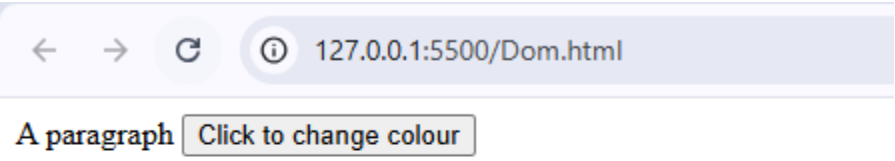
Task 2: Attach an event listener to a button, making it perform an action when clicked.

```
<!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>
<per id="parah">A paragraph </per>
<script src="Dom.js">
</script>
</body>
</html>
```

Dom.js

```
const myb=document.createElement("button");
myb.textContent="Click to change colour";
myb.onclick=()=>{
document.body.style.background='yellow';
}
document.body.append(myb);
```

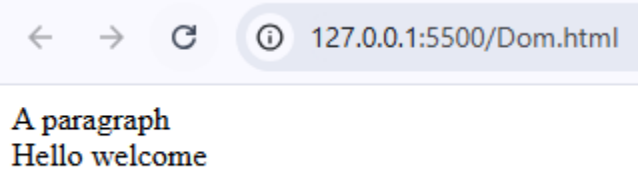
Output



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>
<per id="parah">A paragraph </per>
<script >
   const help=document.createElement("div");
   help.textContent="Hello welcome";
   document.body.append(help)
</script>
</body>
</html>
```

Output



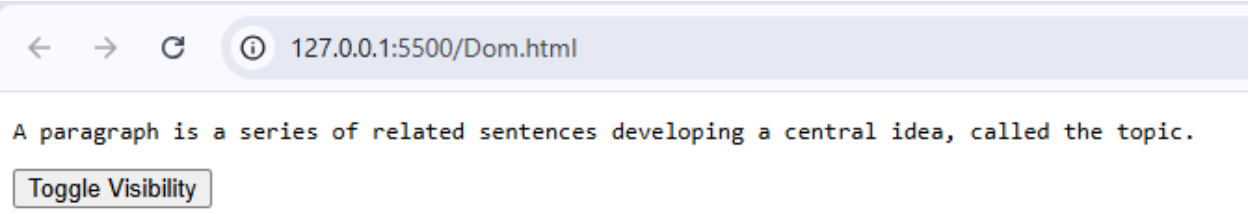
Task 4: Implement a function to toggle the visibility of an element

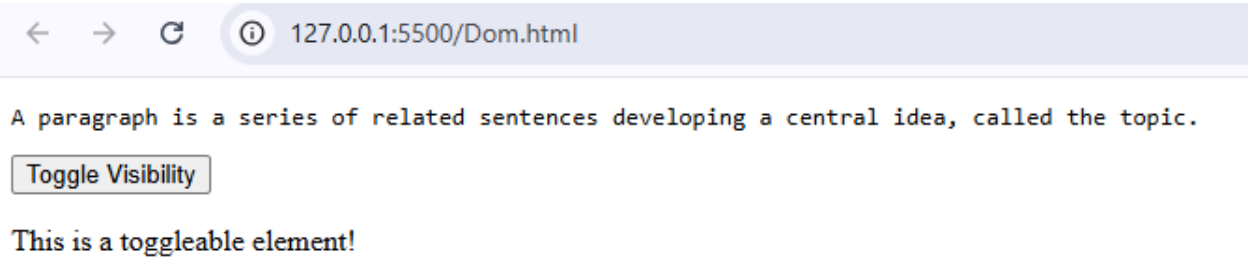
```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<pre id="parah">
A paragraph is a series of related sentences developing a central idea, called the topic.
</pre>
<button onclick="toggleVisibility('myDiv')">Toggle Visibility</button>
<div id="myDiv">
<p>This is a toggleable element!</p>
</div>
<script src="Dom.js">
</script>
</body>
</html>
```

Dom,js

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

Output





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>
```

Output