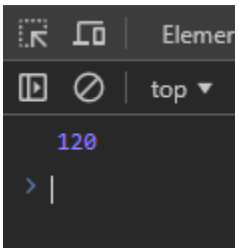
Recursion and stack:

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<=1) return 1;

    return num*factorial(num-1);
 }
 console.log(factorial(5));
    </script>
</body>
</html>
```

Output:

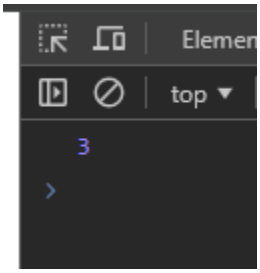


Task 2: Write a recursive function to find the nth Fibonacci number.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript</title>
</head>

<body>
    <script>
        function fibo(num) {
            if (num == 0) {
                return 0;
            } else if (num == 1) {
                return 1;
            } else
                return fibo(num - 1) + fibo(num - 2);
        }
        console.log(fibo(4));
    </script>
```

Output:

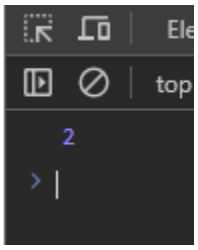


Task 3: Create a function to determine the total number of ways one can climb a staircase with 1, 2, or 3 steps at a time using recursion.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript</title>
</head>

<body>
    <script>
        function cou(num) {
            if (num < 0) {
```

```
                return 0;
            } else if (num == 0 || num == 1) {
                return 1;
            } else if (num == 2) {
                return 2;
            } else {
                return cou(num - 1) + cou(num - 2) + coU(num - 3);
            }
        }
        console.log(cou(2));
    </script>
</body>

</html>
```

OUTPUT:

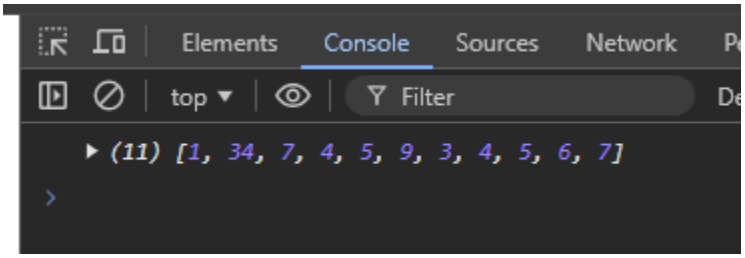


Task 4: Write a recursive function to flatten a nested array structure.

```
<!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 flattenArray(arr) {
            let result = [];
            for (let i = 0; i < arr.length; i++) {
                if (Array.isArray(arr[i])) {
                    result = result.concat(flattenArray(arr[i]));
                } else {
```

```
                    result.push(arr[i]);
                }
            }
            return result;
        }
        console.log(flattenArray([1, [[34, 7], [4, 5]], [9, 3], [4, [5, 6]],
7]));
    </script>
</body>
```

Output:

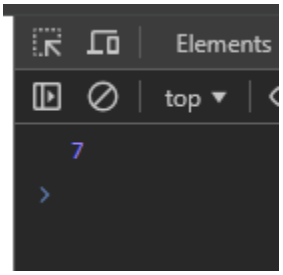


Task 5: Implement the recursive Tower of Hanoi solution.

```
<!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>
        var c = 0;
        function towerofhanoi(n, start, center, end) {
            if (n == 0) {
                return;
            }
            towerofhanoi(n - 1, start, center, end);
            c++;
            towerofhanoi(n - 1, center, end, start)
        }
        n = 3
        towerofhanoi(n, 'A', 'B', 'C')
        console.log(c);
```

```
    </script>
</body>

</html>
```

Output:

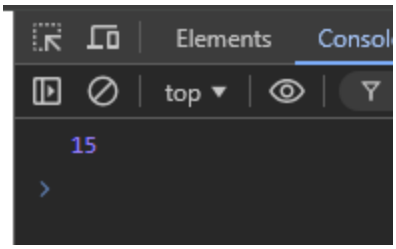


2. JSON and variable length arguments/spread syntax:

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 = (...arr)=>{
        return arr.reduce((a,b)=>a+b,0);
        }
        console.log(sum(1,2,3,4,5))
        </script>
</body>

</html>
```

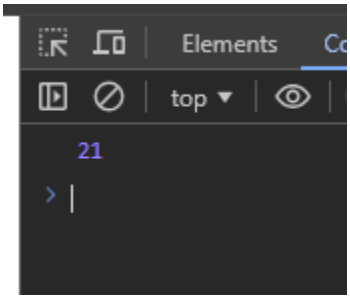
Output:



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>
        let sum = (...arr)=>{
        return arr.reduce((a,b)=>a+b,0);
        }
        arr = [2,2,4,5,8]
        console.log(sum(...arr))
        </script>
</body>

</html>
```

Output:

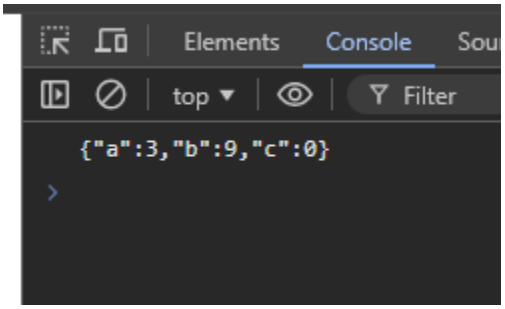


Task 3: Create a deep clone of an object using JSON methods

```
<!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>
    <body>
        <script>
        let sum = {a:3,b:9,c:0}
        console.log(JSON.stringify(sum));
        </script>
       </body>
</body>

</html>
```

Output:

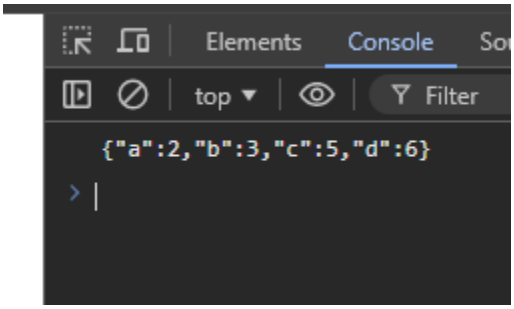


Task 4: Write a function that returns a new object, merging two provided objects using the spread syntax

```
<!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>
            let sum = { a: 2, b: 3, c: 4 }
            let count = { c: 5, d: 6 }
            let newObj = { ...sum, ...count }
```

```
            console.log(JSON.stringify(newObj));
        </script>
</body>

</html>
```

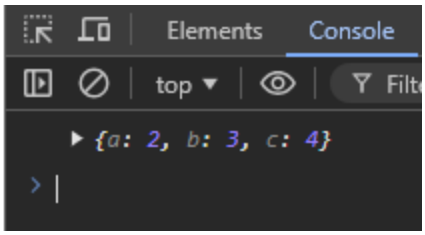
Output:



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 sum = { a: 2, b: 3, c: 4 }
        let newOb = JSON.stringify(sum)
        console.log(JSON.parse(newOb));
    </script>
</body>

</html>
```

Output:

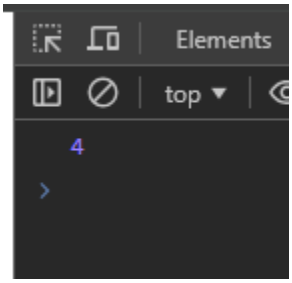


3. Closure:

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>
        function out(a){
        return (b)=>a+b
        }
        let outer = out(2)
        console.log(outer(2));
        </script>
</body>

</html>
```

Output:



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

```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>
        function counter() {
            let c = 0;
            return () => ++c
        }
        c1 = counter();
        console.log(c1())
        console.log(c1())
    </script>
</body>

</html>
```

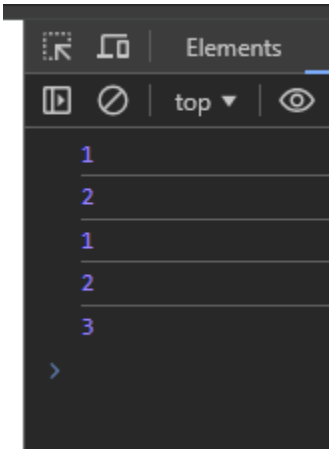
Output:

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

```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>
        function counter() {
            let c = 0;
            return () => ++c
        }
        c1 = counter();
        console.log(c1())
        console.log(c1())
        c2 = counter();
        console.log(c2())
        console.log(c2())
        console.log(c2())
    </script>
</body>

</html>
```

Output:

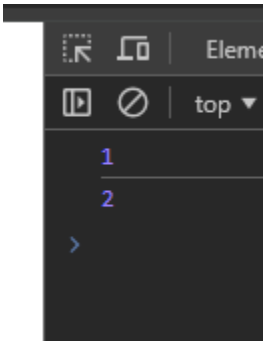


Task 4: Use closures to create private variables within a function

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <script>
```

```
        function counter() {
            let c = 0;
            return () => ++c
        }
        c1 = counter();
        console.log(c1())
        console.log(c1())
    </script>

</body>

</html>
```

Output:

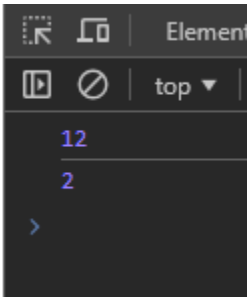


Task 5: Build a function factory that generates functions based on some input using closures.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <script>
        function cal(val) {
            if (val === "+")
                return (a, b) => a + b;
            else if (val === "-")
                return (a, b) => a - b;
        }
        n = cal("+");
```

```
        m = cal("-");
        console.log(n(5, 7))
        console.log(m(7, 5))
    </script>

</body>

</html>
```

Output:



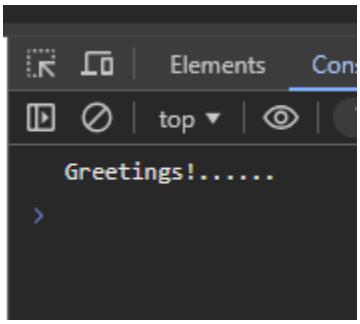
4. Promise, Promises chaining:

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>
        let promise = new Promise((resolve, reject) => {
            setTimeout(() => resolve("Greetings!......"), 1000)
        })
        promise.then(
            result => console.log(result),
            error => console.log(error)
        )
    </script>

</body>
```

```
</html>
```

Output:

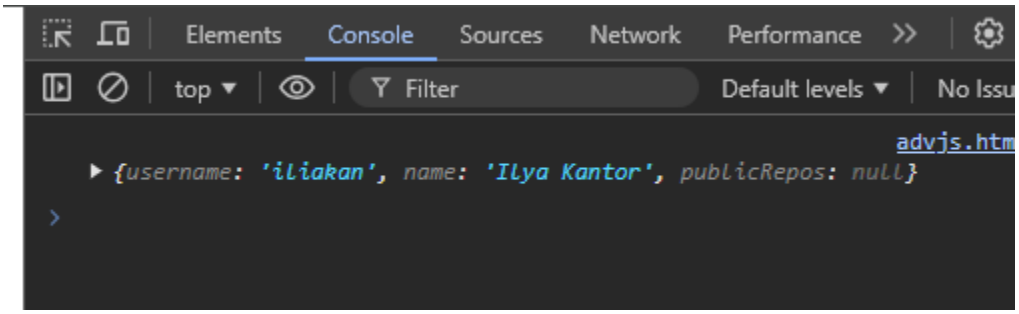


Task 2: Fetch data from an API using promises, and then chain another promise to process this data

```
<!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>
        fetch('https://api.github.com/users/iliakan')
            .then(response => {
                if (!response.ok) return Promise.reject('Failed to fetch data');
                return response.json();
            })
            .then(data => {
                const processedData = {
                    username: data.login,
                    name: data.name,
                    publicRepos: data.bio
                };
                console.log(processedData);
            })
            .catch(error => console.error('Error:', error));
    </script>
```

```
</body>

</html>
```

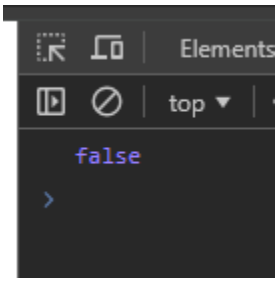
Output:



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>
        randomnumber = (num) => {
            return new Promise((response, reject) => {
                if (num % 2 == 0) response(true);
                else reject(false)
            })
        }
        let i1 = randomnumber(10)
        i1.then(
            result => console.log(result),
            error => console.log(error)
        )
    </script>

</body>

</html>
```

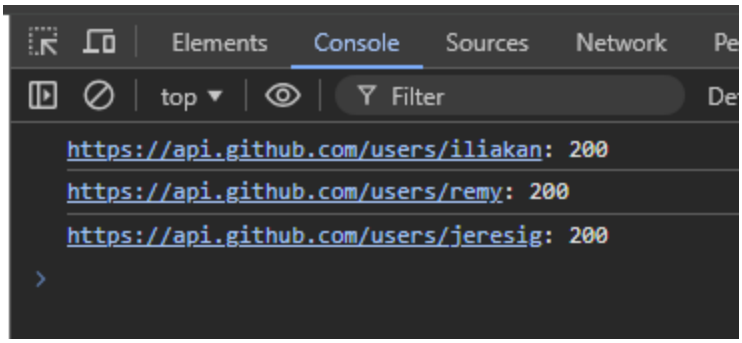
Output:



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>
        let urls = [
            'https://api.github.com/users/iliakan',
            'https://api.github.com/users/remy',
            'https://api.github.com/users/jeresig'
        ];
        let requests = urls.map(url => fetch(url));
        Promise.all(requests)
            .then(responses => responses.forEach(
                response => console.log(`${response.url}: ${response.status}`)
            ));
    </script>
</body>
</html>
```

Output:;

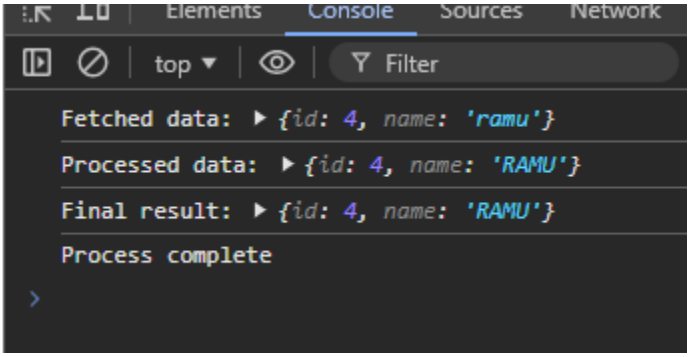


Task 5: Chain multiple promises to perform a series of asynchronous actions in sequence.

```
<!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>
        function fetchData() {
            return new Promise((resolve) => {
                setTimeout(() => {
                    const data = { id: 4, name: "ramu" };
                    console.log('Fetched data:', data);
                    resolve(data);
                }, 1000);
            });
        }
        function processData(data) {
            return new Promise((resolve) => {
                setTimeout(() => {
                    data.name = data.name.toUpperCase();
                    console.log('Processed data:', data);
                    resolve(data);
                }, 1000);
            });
        }
        function logResult(data) {
            return new Promise((resolve) => {
                setTimeout(() => {
                    console.log('Final result:', data);
```

```
                    resolve('Process complete');
                }, 1000);
            });
        }
        fetchData()
            .then(data => processData(data))
            .then(processedData => logResult(processedData))
            .then(result => console.log(result))
            .catch(error => console.error('Error:', error));
    </script>
</body>

</html>
```

Output:



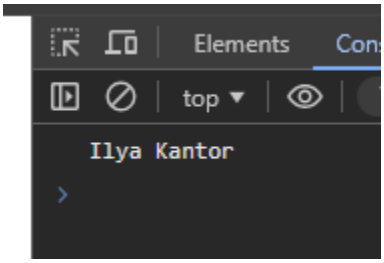
5. Async/await:

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>
        async function fetchData() {
            try {
                const response = await
fetch('https://api.github.com/users/iliakan');
                if (!response.ok) {
```

```
                    throw new Error('Failed to fetch data');
                }
                const data = await response.json();
                console.log(data.name);
                return data;
            } catch (error) {
                console.error('Error:', error);
            }
        }
        fetchData();
    </script>
</body>
</html>
```

Output :

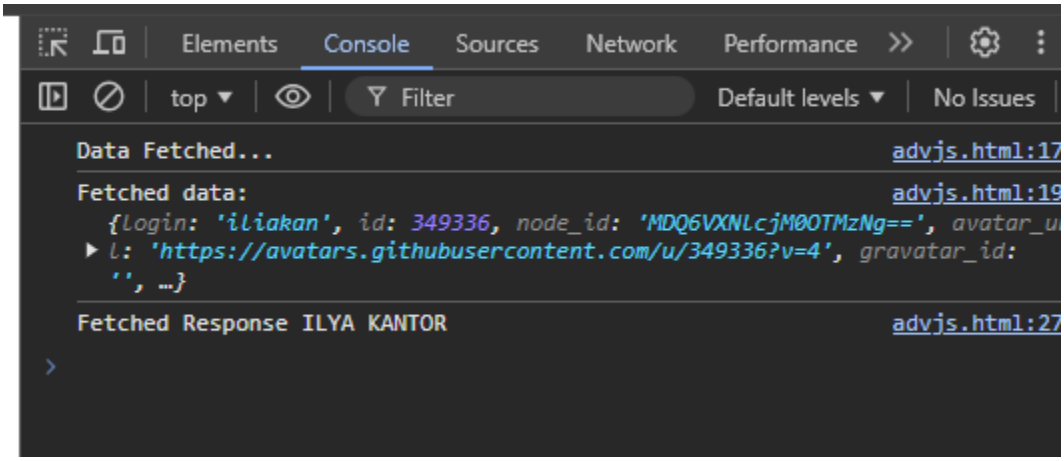


Task 2: Create an async function that fetches data from an API and processes it.

```
<!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>
        data = async () => {
            try {
                const url = await fetch('https://api.github.com/users/iliakan')
                if (!url.ok)
                    throw new Error("Can't able to fetch the data");
                console.log("Data Fetched...");
                const pro = await url.json()
                console.log("Fetched data: ", pro);
```

```
                const pdata = await pro.name.toUpperCase()
                return pdata;
            } catch (error) {
                console.error(error)
            }
        }
        data().then((res) => {
            console.log("Fetched Response", res);
        })
    </script>
    </script>
</body>

</html>
```

Output:

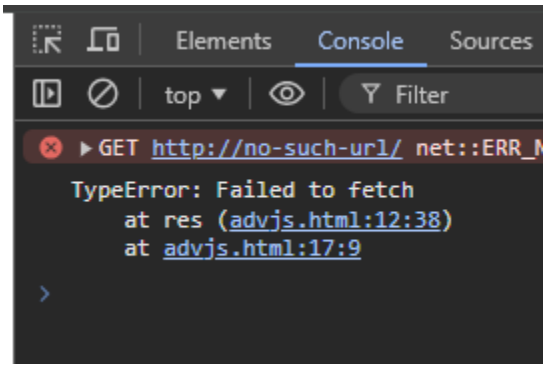


Task 3: Implement error handling in an async function using try/catch.

```
<!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>
        async function res() {
            try {
                let response = await fetch('http://no-such-url');
            } catch (err) {
```

```
                console.log(err);
            }
        }
        res();
    </script>
</body>

</html>
```

Output:

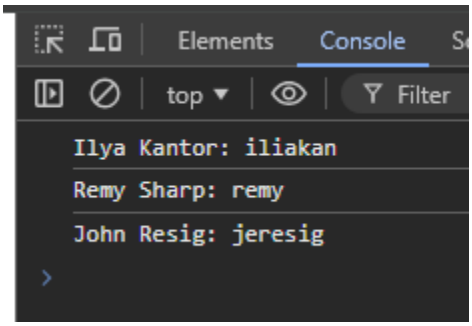


Task 4: Use async/await in combination with Promise.all.

```
<!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>
        async function name() {
            let urls = [
                'https://api.github.com/users/iliakan',
                'https://api.github.com/users/remy',
                'https://api.github.com/users/jeresig'
            ];
            let requests = await Promise.all(urls.map(url => fetch(url)))
            let obj = await Promise.all(requests.map((res) => res.json()))
            return obj;
        }
```

```
        name().then(responses => responses.forEach(
            response => console.log(`${response.name}: ${response.login}`)
        )).catch(error => {
            console.error('Error:', error);
        });
    </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>
        orderfood = (order) => {
            document.write(`Getting Order!........`)
            document.write("<br>")
            return new Promise((resolve) => {
                setTimeout(() => {
                    document.write(`Order is placed for ${order}`)
                    document.write("<br>")
                    resolve(order)
                }, 1000);
```

```
            })
        }
        preparefood = (order) => {
            document.write(`Your Food ${order} is Preparing!........`)
            document.write("<br>")
            return new Promise((resolve) => {
                setTimeout(() => {
                    document.write(`Your Food ${order} is Prepared!.....`)
                    document.write("<br>")
                    resolve(order)
                }, 1000);
            })
        }
        deliverfood = (order) => {
            document.write(`Getting Order to delivery!........`)
            document.write("<br>")
            return new Promise((resolve) => {
                setTimeout(() => {
                    document.write(`${order} is delivered`)
                    document.write("<br>")
                    resolve(order)
                }, 1500);
            })
        }
        food = async (value) => {
            const order = await orderfood(value)
            const Prepare = await preparefood(order)
            const deliver = await deliverfood(Prepare)
            if (deliver == order) {
                document.write("Thank You!......");
            } else {
                document.write("Please Wait!...")
            }
        }
        value = prompt("Enter the Dish you desire: ")
        food(value)
    </script>
</body>

</html>
```

Output:

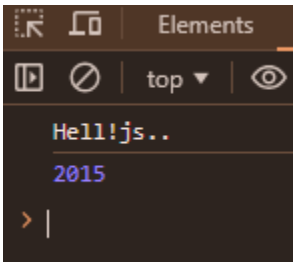
```
Order is placed for idli
Your Food idli is Preparing!.....
Your Food idli is Prepared!.....
Getting Order to delivery!.....
idli is delivered
Thank You!.....
```

6. Modules introduction, Export and Import:

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

```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>
        export let name = () => { console.log("Hell!js..") };
        export const val = 2015;
        export class User {
            constructor(name) {
                this.name = name;
            }
            sayhi = () => { console.log("kce");
             }
        }
    </script>
</body>
</html>
```

Output:



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

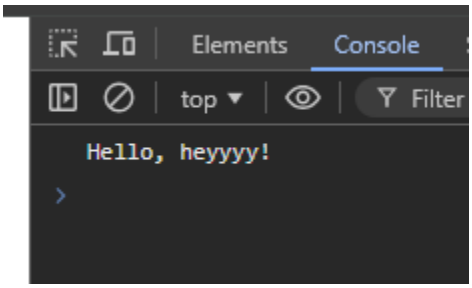
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script type="module">
        import { sayHi } from './task.js';
        sayHi('John');
    </script>
</body>
</html>
```

Output:

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

```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 type="module">
        export function sayHi(user) {
            console.log(`Hello, ${user}!`);
        }

        export function sayBye(user) {
            console.log(`Bye, ${user}!`);
        }
        sayHi("heyyyy")
    </script>
</body>

</html>
```

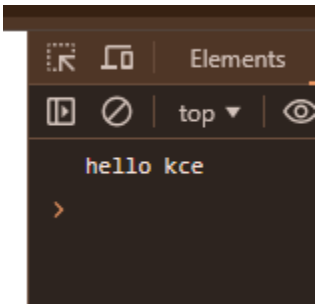
Output:



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 {sayhi} from './one.js';
        sayhi("hello kce");
    </script>
</body>
</html>
```

Output:



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

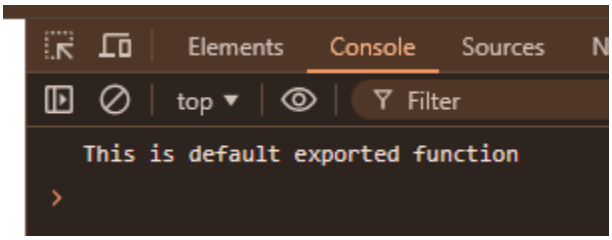
```
<!DOCTYPE html>
<html lang="en">
```

```
<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 sayhi from './one.js';
        sayhi("This is default exported function");
    </script>
</body>
</html>
```

One.js:

```
export default function sayhi(name){
    console.log(name);
}
```

Output:



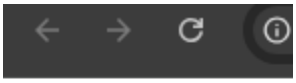
7. Browser: DOM Basics:

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>
    <button id="but" onclick="fun()">click me</button>
    <p id="p">hello</p>
    <script>
    function fun(){
        document.getElementById("p").innerHTML="hello,heyyyy";
    }
    </script>
</body>

</html>
```

Output:



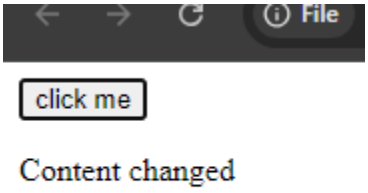
click me

hello,heyyyy

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>
    <button id="but">click me</button>
    <p id="p">hello,click the above button to change the content</p>
    <script>
     document.getElementById("but").addEventListener("click",()=>{
        document.getElementById("p").innerHTML="Content changed";
     })
    </script>
</body>
</html>
```

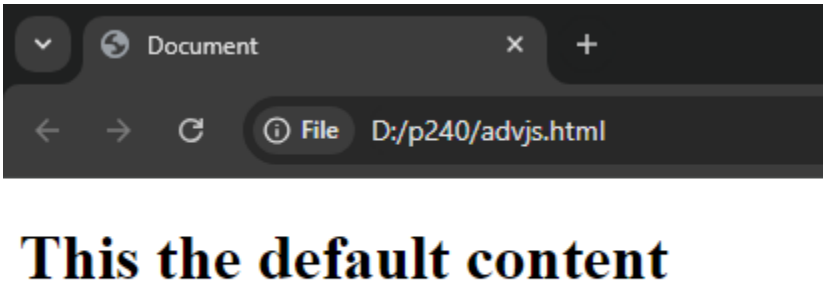
Output:



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>
    <h1 id="h">This the content</h1>
    <script>
        const change = () => {
            let ele = document.createElement("h").innerHTML = "Hai!Javascript"
            document.body.appendChild(ele);
        }
        change()
    </script>
</body>
</html>
```

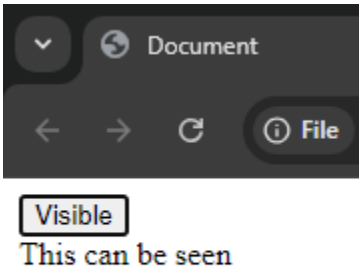
Output:



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>
    <button onclick="vis()">Visible</button>
    <div id="e" style="display: none;">
        This can be seen
    </div>
    <script>
        function vis() {
            const element = document.getElementById('e');
            if (element.style.display === 'none') {
                element.style.display = 'block';
            } else {
                element.style.display = 'none';
            }
        }
    </script>
</body>
</html>
```

Output:



Task 5: Use the DOM API to retrieve and modify the attributes 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>
    <button class="btn" id="btn" title="click me">Hello</button>
    <script>
        let a=document.getElementById("btn");
        console.log(a.getAttribute("class"));
        a.setAttribute("class","newclass");
        console.log(a.getAttribute("class"));
        console.log(a.getAttribute("title"));
    </script>
</body>
</html>
```

Output: