Closure:

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

Output:

main.js

```javascript
function createMultiplier(multiplier) {
    return function(number) {
        return number * multiplier;
    };
}

const double = createMultiplier(5);
console.log(double(9));
```

Output

```
45

=== Code Execution Successful ===
```

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

Output:

main.js

```javascript
function createCounter() {
    let count = 0;
    return {
        increment() {
            count++;
        },
        getCount() {
            return count;
        }
    };
}

const counter = createCounter();
counter.increment();
counter.increment();
counter.increment();
console.log(counter.getCount());
```

Output

```
3

=== Code Execution Successful ===
```

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

Output:

main.js

```javascript
function createCounterFactory() {
    return function() {
        let count = 0;
        return {
            increment() {
                count++;
            },
            getCount() {
                return count;
            }
        };
    };
}

const newCounter = createCounterFactory();
const counter1 = newCounter();
const counter2 = newCounter();
counter1.increment();
counter1.increment();
console.log(counter1.getCount());
console.log(counter2.getCount());
```

Output

```
2
0

=== Code Execution Successful ===
```

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

Output:

main.js

```javascript
function createPrivateCounter() {
    let privateCount = 0;
    return {
        increment() {
            privateCount++;
        },
        getCount() {
            return privateCount;
        }
    };
}

const privateCounter = createPrivateCounter();
privateCounter.increment();
privateCounter.increment();
console.log(privateCounter.getCount());
```

Output

```
2

=== Code Execution Successful ===
```

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

Output:

main.js

```javascript
function createFunctionFactory(operation) {
    return function(a, b) {
        switch (operation) {
            case "add":
                return a + b;
            case "subtract":
                return a - b;
            default:
                return null;
        }
    };
}

const adder = createFunctionFactory("add");
console.log(adder(10, 3));
```

Output

```
13

=== Code Execution Successful ===
```

Promise, Promises chaining:

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

Output:

main.js

```javascript
function delayed(seconds) {
    return new Promise((resolve) => {
        setTimeout(() => resolve("Hello, world!"), seconds * 1000);
    });
}

delayed(2).then(console.log);
```

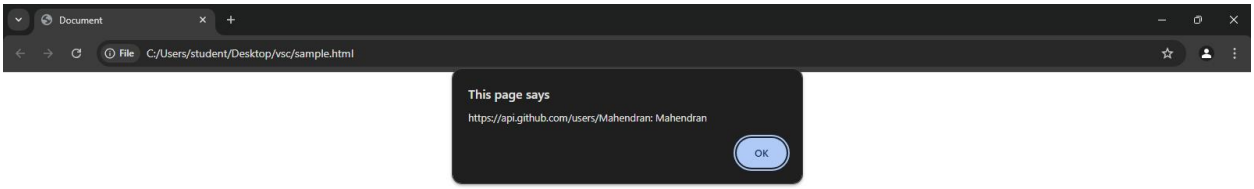
Output

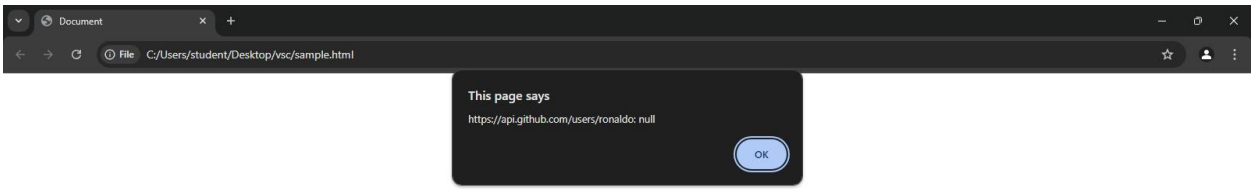
```
Hello, world!

=== Code Execution Successful ===
```

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>
let urls = [
'https://api.github.com/users/mahendran',
'https://api.github.com/users/ronaldo'
];
let requests = urls.map(url => fetch(url));
Promise.all(requests)
.then(responses =>
Promise.all(responses.map(response => response.json()))
)
.then(data => data.forEach(user =>

alert(`${user.url}: ${user.name}`)
));
console.log(typeof request);
</script>
</body>
</html>
```

Output:





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

Output:

main.js

```javascript
function randomPromise() {
    return new Promise((resolve, reject) => {
        const rand = Math.random();
        rand > 0.5 ? resolve("Success!") : reject("Failure!");
    });
}

randomPromise()
    .then(console.log)
    .catch(console.error);
```

Output

```
Failure!

=== Code Execution Successful
```

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

Output:

index.html

C: > Users > student.AT-30 > index.html > html > body > script > urls

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Filter (e.g. text, !exclude, \escape)

```
https://api.github.com/users/JEYSAN-V: 403
```

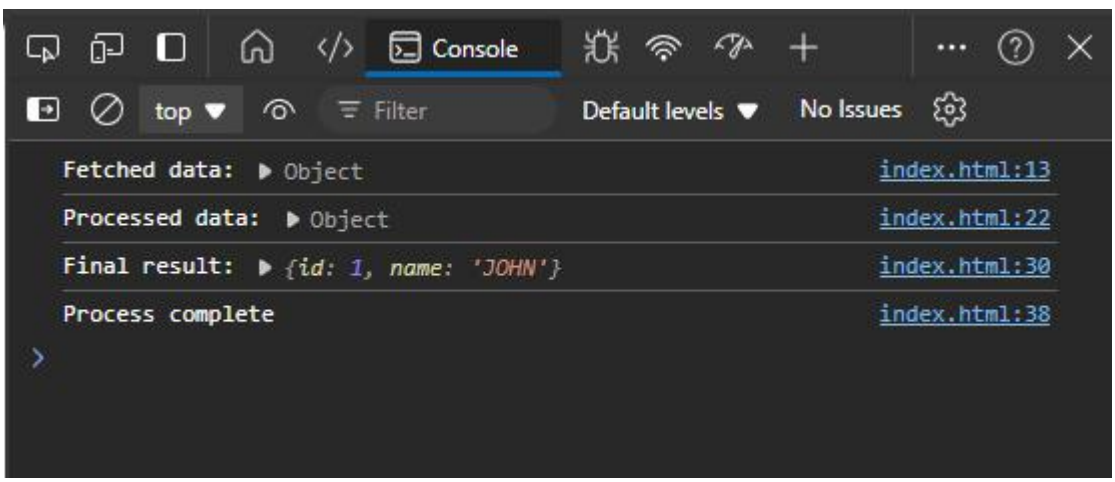
Task 5: Chain multiple promises to perform a series of asynchronous actions in sequence.

```html
<!DOCTYPE html>

<html lang="en">

<body>

 <script>

 function fetchData() {
```

```
return new Promise((resolve) => {
setTimeout(() => {
const data = { id: 1, name: "John" };
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:



Async/await:

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

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>Promise Chain Example</title>
</head>
<body>
 <script>
 data = async()=>{
 try{
 const url = await fetch('https://api.github.com/users/JEYSAN-V');
 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>
</body>
</html>
```

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>JavaScript</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:

```
PROBLEMS 3   OUTPUT   DEBUG CONSOLE   TERMINAL   ...
  true
```

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>Promise Chain Example</title>
</head>
<body>
  <script>
    async function getData() {
      let urls = [
        'https://api.github.com/users/JEYSAN-V'
      ];
      try {
        let requests = await Promise.all(urls.map(url => fetch(url)));

        let data = await Promise.all(requests.map(res => res.json()));

        return data;
      } catch (error) {
        console.error('Error:', error);
      }
    }
    getData().then(responses => {
      if (responses) {
        responses.forEach(response => {
          console.log(`${response.name}: ${response.login}`);
        });
      }
    }).catch(error => {
      console.error('Error:', error);
    });
```

```
  </script>

</body>

</html>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Filter (e.g. text, !exclude, \escape)

```
undefined: undefined
```

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

main.js

Share Run

```
1  async function waitForAll() {
2      const tasks = [
3          new Promise(resolve => setTimeout(() => resolve("Task 1 done"),
               1000)),
4          new Promise(resolve => setTimeout(() => resolve("Task 2 done"),
               2000)),
5      ];
6      const results = await Promise.all(tasks);
7      console.log(results);
8  }
9
10 waitForAll();
11
```

Output

```
[ 'Task 1 done', 'Task 2 done' ]

=== Code Execution Successful ===
```

6. Modules introduction, Export and Import:

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

```
export const variable = "Hello";
export function greet() {
    return "Hello, World!";
}
export class Greeter {
    sayHello() {
        return "Hi!";
    }
}
import { variable, greet, Greeter } from './zoro.js';
console.log(variable);
console.log(greet());
console.log(new Greeter().sayHello());
```

Output:

```
C:\Program Files\nodejs\node.exe .\..\vsc\luffy.js
Hello
Hello, World!
Hi!
```

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

```
import { variable, greet, Greeter } from './zoro.js';
console.log(variable);
console.log(greet());
console.log(new Greeter().sayHello());
```

Output:

```
C:\Program Files\nodejs\node.exe .\..\vsc\luffy.js
Hello
Hello, World!
Hi!
```

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

```
export const variable = "Hello";
export function greet() {
    return "Hello, World!";
}
export class Greeter {
    sayHello() {
        return "Hi!";
    }
}
```

Output:

```
C:\Program Files\nodejs\node.exe .\..\vsc\luffy.js
Hello
Hello, World!
Hi!
```

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

```
export const variable = "Hello";
export function greet() {
    return "Hello, World!";
}
export class Greeter {
    sayHello() {
        return "Hi!";
    }
}
import { variable, greet, Greeter } from './zoro.js';
console.log(variable);
console.log(greet());
console.log(new Greeter().sayHello());
```

Output:

```
C:\Program Files\nodejs\node.exe .\..\vsc\luffy.js
Hello
Hello, World!
Hi!
```

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

```
export const variable = "Hello";
export default function greet() {
    return "Hello, World!";
}

import greet from './zoro.js';
console.log(greet());
```

Output:

```
C:\Program Files\nodejs\node.exe .\..\vsc\luffy.js
Hello, World!
```

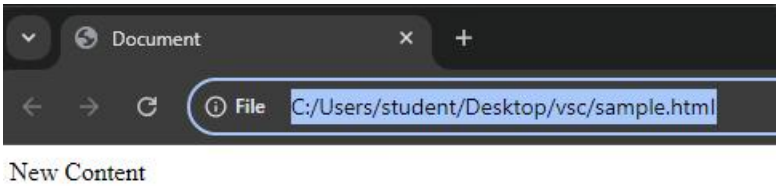
Browser: DOM Basics:

Task 1: Select an HTML element by its ID and change its content using JavaScript.

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

```html
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<div id ='a'>
</div>
<script>
document.getElementById('a').textContent = "New Content";
</script>
</body>
</html>
```

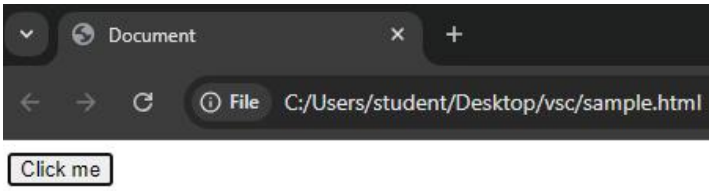
Output:

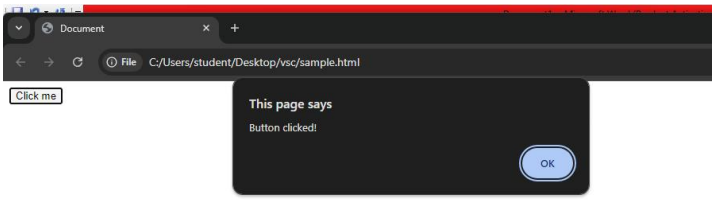


Task 2: Attach an event listener to a button, making it perform an action when clicked.

```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 id="buttonId">Click me</button>
<script>
document.getElementById('buttonId').addEventListener('click', () => {
    alert("Button clicked!");
});
</script>
</body>
</html>
```

Output:

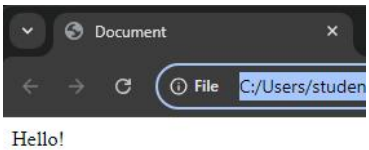




Task 3: Create a new HTML element and append it to the DOM.

```
<!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>
<script>
const newElement = document.createElement('div');
    newElement.textContent = "Hello!";
    document.body.appendChild(newElement);

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

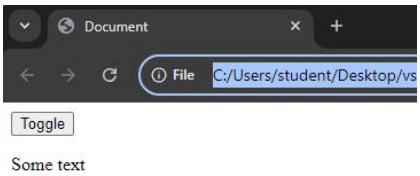
Output:

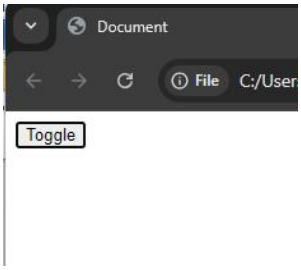


Task 4: Implement a function to toggle the visibility of an element.

```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>
    <div>
        <button onclick="toggleVisibility('myElement')">Toggle</button>

        <div id="myElement" style="display: none;">
            <p>Some text</p>
        </div>
<script>

function toggleVisibility(id) {
    const element = document.getElementById(id);
    element.style.display = element.style.display === "none" ? "block" : "none";
}
toggleVisibility("myElement");

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

Output:

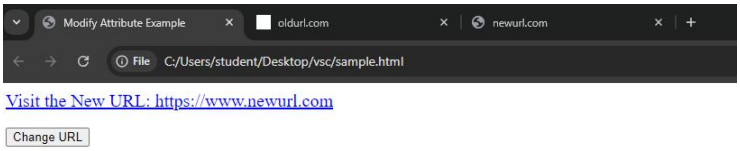




Task 5: Use the DOM API to retrieve and modify the attributes of an element.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Modify Attribute Example</title>
    <style>
        #link {
            font-size: 20px;
            color: blue;
        }
    </style>
</head>
<body>
    <a href="https://www.oldurl.com" id="link" target="_blank">Visit the Old
URL</a><br><br>

    <button id="changeUrlButton">Change URL</button>

    <script>
        const link = document.getElementById('link');
        const changeUrlButton = document.getElementById('changeUrlButton');

        console.log("Old URL:", link.getAttribute('href'));

        changeUrlButton.addEventListener('click', () => {
            const newUrl = "https://www.newurl.com";
            link.setAttribute('href', newUrl);
            console.log("New URL:", link.getAttribute('href'));
            link.textContent = `Visit the New URL: ${newUrl}`;
        });
    </script>
</body>
</html>
```

Output:


Modify Attribute Example
oldurl.com
newurl.com
File
C:/Users/student/Desktop/vsc/sample.html
Visit the New URL: https://www.newurl.com
Change URL