717821T140

MERN Stack Training

Weekly Tasks-3

1. Recursion and stack:

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

```
<body>
    <script>
function factorial(n) {
        let fact = 1;
        if(n === 0)
            return 1;
        for (let i = 2; i <= n; i++)
            fact = fact * i;
        return fact;
    }
    console.log(factorial(3));
    </script>
</body>
```

PROBLEMS DEBUG CONSOLE ··· Filter (e.g. text, !exclude)

6

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

```
function fibonacci(n){
  if(n<2){
      return n;
  }else{
      return fibonacci(n-1)+fibonacci(n-2);
  }
}
console.log(fibonacci(10));
  </script>
</body>
</html>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

55

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

```
function stairs(n){
  if(n==0){
      return 1;
  }else if(n<0){
      return 0;
  }else{
      return stairs(n-3)+stairs(n-2)+stairs(n-1);
  }
}
console.log(stairs(4));
  </script>
```

PROBLEMS DEBUG CONSOLE ··· Filter (e.g. text, !exclude)

7

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

```
console.log(stairs(4));*/
const arr=[1,2,[3,4],5];
function flatten(arr){
  console.log(arr.flat(Infinity));
}
console.log(flatten(arr));
  </script>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
> (5) [1, 2, 3, 4, 5]
  undefined
```

o Task 5: Implement the recursive Tower of Hanoi solution

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

PROBLEMS DEBUG CONSOLE ··· Filter (e.g. text, !exclude)

```
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
```

2. JSON and variable length arguments/spread syntax:

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

```
function arbitrary(){
    console.log(arguments);
    let sum=0;
for(let i=0;i<arguments.length;i++){
sum+=arguments[i];
}
console.log(sum);
return sum;
}
arbitrary(1,2,3);
    </script>
</body>
</html>
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    Filter (e.g. text, !e
> Arguments(3) [1, 2, 3, callee: ƒ, Symbol(Symbol.iterator): ƒ]
  6
```

o Task 2: Modify a function to accept an array of numbers and return their sum using the spread syntax.

```
function sum(x, y) {
  return x + y ;
}

const numbers = [2,2];

console.log(sum(...numbers));
</script>
</body>
</html>
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMI
  4
```

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

```
let cust={
    "name":"arun",
    "id":107
};
let custobj=JSON.parse(JSON.stringify(cust));
console.log(cust);
    </script>
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
> {name: 'arun', id: 107}
```

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

```
let clg1={
name:'reema',
clg:'KCE'
};
let clg2={
 girlname:'divya',
clg:'KCE'
};
let clg3={
    ...clg1,
    ...clg2
};
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
> {name: 'reema', clg: 'KCE', girlname: 'divya'}
```

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

```
let cust='{"name":"arun","id":107}';
let custobj=JSON.stringify(JSON.parse(cust));
console.log(custobj);
    </script>
```

PROBLEMS DEBUG CONSOLE ··· Filter (e.g. text, !exclude)

```
{"name":"arun","id":107}
```

3. Closure:

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

```
function counter(){
let count=11;
return function(){
    count++;
    console.log(count);
}
}
let closures=counter();
closures();
    </script>
</body>
```

PROBLEMS OUTPUT DEBUG CONSOLE T

```
12
```

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

```
function counter(){
let count=11;
return {
increament:function(){
    count++;
    console.log(count);
},
decreament:function(){
    count--;
    console.log(count);
}
}
}
let closures=counter();
closures.increament();
closures.increament();
```

PROBLEMS DEBUG CONSOLE ··· Filter (e.g. t

```
12
13
```

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

```
console.log(custobj);*/
function counter(){
let count=11;
return {
increament:function(){
    count++;
    console.log(count);
},
decreament:function(){
    count--;
    console.log(count);
}
}
}
let closures=counter();
closures.increament();
closures.increament();
closures.increament();
closures.increament();
closures.increament();
closures.increament();
```

PROBLEMS DEBUG CONSOLE ··· Filter (

```
12
13
14
15
16
17
```

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

```
function counter(){
let count=11;
return function(){
    count++;
    console.log(count);
}
}
let closures=counter();
closures();
    </script>
</body>
```

PROBLEMS OUTPUT DEBUG CONSOLE

```
12
```

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

```
    <script>
        function createPerson(firstName, lastName) {
  return {
    firstName: firstName,
    lastName: lastName,
    getFullName() {
      return firstName + ' ' + lastName;
    },
  };
}
let person1 = createPerson('John', 'Doe');
console.log(person1.getFullName());
    </script>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Filter (e.g. text, !exclude)

```
John Doe                                    index
```

4. Promise, Promises chaining:

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

```
function resolveAfter2Seconds() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}
const result =resolveAfter2Seconds();
  console.log(result);
```

```
▼ Promise i
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "resolved"
```

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

```
<script>
    let urls=[
    ('https://jsonplaceholder.typicode.com/posts/4'),
    ('https://jsonplaceholder.typicode.com/posts/5')
];
let reques=urls.map(url=>fetch(url));

Promise.all(reques)
.then(response=>response.forEach(
    response=>console.log(response)
));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
> Response {type: 'cors', url: 'https://jsonplaceholder.typicode.com/posts/4', redirected: false, status: 200, ok: true, …}
> Response {type: 'cors', url: 'https://jsonplaceholder.typicode.com/posts/5', redirected: false, status: 200, ok: true, …}
```

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

```
function myfunc(num){
    return new Promise((resolve,reject)=>{
        if(num>0){
            resolve("number is greater than 0");
        }else{
            reject("number is less than 0");
        }
    });
}
console.log(myfunc(4));
    </script>
```

PROBLEMS DEBUG CONSOLE ··· Filter (e.g. text, !exclude)

```
> Promise {[[PromiseState]]: 'fulfilled', [[PromiseResult]]: 'number is greater
  than 0'}
```

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

```
let urls=[
    ('https://jsonplaceholder.typicode.com/posts/4'),
    ('https://jsonplaceholder.typicode.com/posts/5')
];
let reques=urls.map(url=>fetch(url));

Promise.all(reques)
.then(response=>response.forEach(
    response=>console.log(response)
));
```

```
PROBLEMS    DEBUG CONSOLE    ...    Filter (e.g. text, !exclude)
> Response {type: 'cors', url: 'https://jsonplaceholder.typicode.com/posts/4', redi
  rected: false, status: 200, ok: true, …}
> Response {type: 'cors', url: 'https://jsonplaceholder.typicode.com/posts/5', redi
  rected: false, status: 200, ok: true, …}
```

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

```
function myfunc(num){
    return new Promise((resolve,reject)=>{
        resolve(num*num);
    });
}
console.log(myfunc(4).then(result=>myfunc(result)));
    </script>
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
v Promise {[[PromiseState]]: 'pending', [[PromiseResult]]: undefined}
  [[PromiseResult]]: 256
  [[PromiseState]]: 'fulfilled'
  > [[Prototype]]: Promise
```

5. Async/await:

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

```
function resolveAfter2Seconds() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}
async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result);
  // Expected output: "resolved"
}
asyncCall();
    </script>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Filter (e.g. text, !exclude)

```
calling
resolved
```

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

```
}
console.log(myfunc(4).then(result=>myfunc(result)));*/
async function fetchrequest(){
    let result=await fetch('https://jsonplaceholder.typicode.com/posts/4')
   let output=await result.json();
   console.log(output.title);
}
fetchrequest();
    </script>
</body>
</html>
```

PROBLEMS OUTPUT DEBUG CONSOLE ··· Filter (e.g. text, !exclude)

```
eum et est occaecati
```

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

```
async function fetchrequest(){
    try{
    let result=await fetch('https://jsonplaceholder.typicode.com/posts/4')
   let output=await result.jSon();
   console.log(output.title);
}
catch(error){
console.log("code has Error");
}
}
fetchrequest();
    </script>
</body>
</html>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, !exclud

```
code has Error
```

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

```
let myfunc1=()=>{
    return new Promise((resolve,reject)=>{
        resolve("I am Divya");
    });
}
let myfunc2=()=>{
    return new Promise((resolve,reject)=>{
        resolve("I am EEE student");
    });
}
let promise=async ()=>{
    let promise1=await Promise.all([myfunc1(),myfunc2()]);
    console.log(promise1);
}
promise();
    </script>
```

PROBLEMS OUTPUT DEBUG CONSOLE ··· Filter (e.g. text, !exclude)

```
> (2) ['I am Divya', 'I am EEE student']
```

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

```
promise();*/
async function fetchAsync()
{
    let result=await fetch('https://jsonplaceholder.typicode.com/posts/5');
    let output=await result.json();
    return output;
}
fetchAsync()
.then((output)=>console.log(output))
.catch((reason)=>console.log("Message:"+reason.message));


    </script>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, !exclude)

```
> {userId: 1, id: 5, title: 'nesciunt quas odio', body: 'repudiandae veniam quaerat sun
  t sed
  alias aut …voluptatibus quis
  est aut tenetur dolor neque'}
```

6. Modules introduction, Export and Import:

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

index.html

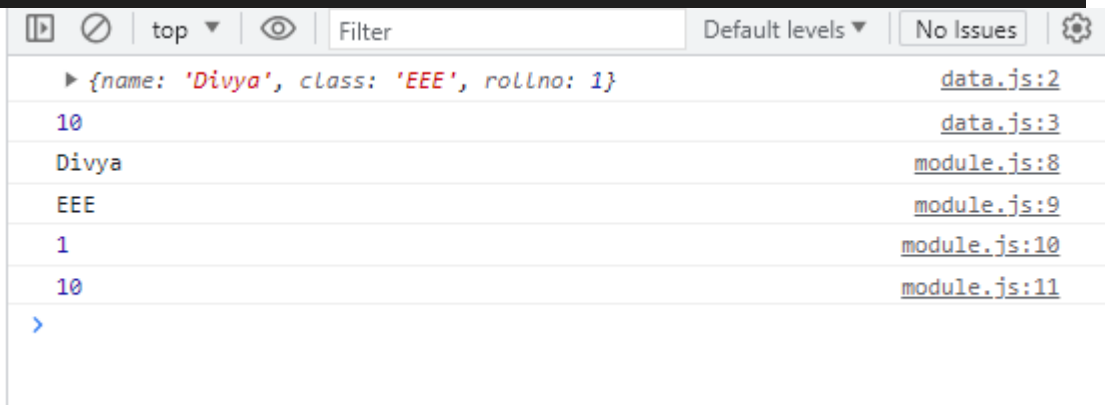
```
<script src="data.js" type="module">
```

Data.js

```
import { employee,a,a1 } from "./module.js";
console.log(employee);
console.log(a);
a1();
```

Module.js

```
export const employee={
    name:"Divya",
    class:"EEE",
    rollno:1,
};
export let a=10;
 export function a1(){
    console.log(employee.name);
    console.log(employee.class);
    console.log(employee.rollno);
    console.log(a);
}
```



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

index.html

```
<script src="data.js" type="module">
```

Data.js

```
import { employee,a,a1 } from "./module.js";
console.log(employee);
console.log(a);
a1();
```

Module.js

```
export const employee={
    name:"Divya",
    class:"EEE",
    rollno:1,
};
export let a=10;
 export function a1(){
    console.log(employee.name);
    console.log(employee.class);
    console.log(employee.rollno);
    console.log(a);
}
```

```
▶ {name: 'Divya', class: 'EEE', rollno: 1}        data.js:2
10                                                data.js:3
Divya                                             module.js:8
EEE                                               module.js:9
1                                                 module.js:10
10                                                module.js:11
>
```

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

```
export const employee={
    name:"Divya",
    class:"EEE",
    rollno:1,
};
export let a=10;
export function sayhello(){
    console.log("Hello!!!")
}
 export function a1(){
    console.log(employee.name);
    console.log(employee.class);
    console.log(employee.rollno);
    console.log(a);
}
```

```
Live reload enabled.                              index.html:207
▶ Object                                          data.js:2
10                                                data.js:3
Divya                                             module.js:11
EEE                                               module.js:12
1                                                 module.js:13
10                                                module.js:14
Hello!!!                                          module.js:8
>
```

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

```
import { employee,a,a1, sayhello } from "./module.js";
console.log(employee);
console.log(a);
a1();
sayhello();
```

```
Live reload enabled.                                   index.html:207
 ▶ Object                                                  data.js:2
10                                                         data.js:3
Divya                                                   module.js:11
EEE                                                     module.js:12
1                                                       module.js:13
10                                                      module.js:14
Hello!!!                                                 module.js:8
>
```

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

```
<script src="data.js" type="module">
```

```
import  hi, { employee,a,a1, sayhello } from "./module.js";
console.log(employee);
console.log(a);
a1();
sayhello();
hi(5);
```

```
export const employee={
    name:"Divya",
    class:"EEE",
    rollno:1,
};
export let a=10;
export function sayhello(){
    console.log("Hello!!!")
}
 export function a1(){
    console.log(employee.name);
    console.log(employee.class);
    console.log(employee.rollno);
    console.log(a);
}
export default function(num){
console.log(num+2);
}
```

7. Browser: DOM Basics:

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

```
   <p id="hi">hello all</p>
    <script src="data.js">
```

Data.js

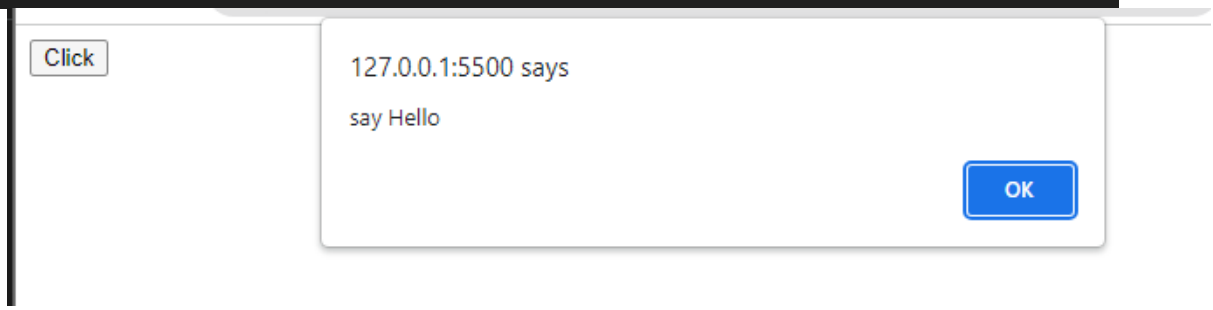
```
const a=document.gEEElementById("hi");
a.innerHTML="Everything is Fine";
```

Everything is Fine

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

```
<body>
    <button type="button">Click</button>
    <script src="data.js">
```

```
document.addEventListener('click',function(){
    alert("say Hello");
});
```



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

```
</head>
<body>
    <div id="name">
    <p id="hi"> </p>
</div>
    <script src="data.js">
```

```
const a=document.gEEElementById("name");
let num=document.gEEElementById("hi");
let h1=document.createElement('h1');
h1.textContent="HelloWorld";
a.appendChild(h1);
console.log(a);
```

HelloWorld

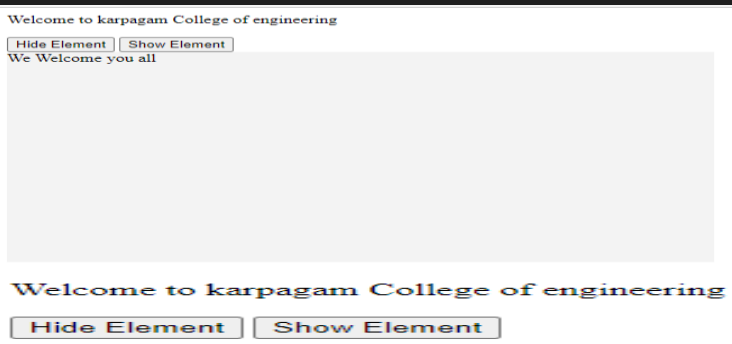
o Task 4: Implement a function to toggle the visibility of an element.

```
<html>
<style>
   #dip {
```

```
      width: 630px;
      height: 300px;
      background-color: #F3F3F3;
   }
</style>
<body>
   <p> Welcome to karpagam College of engineering </p>
   <button onclick="hide()"> Hide Element </button>
   <button onclick="show()"> Show Element </button>
   <div id="dip"> We Welcome you all</div>
   <script src="index.js">

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

```
function hide() {
    document.gEEElementById('dip').style.visibility = 'hidden';
 }
 function show() {
    document.gEEElementById('dip').style.visibility = 'visible';
 }
```



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

```
   <p id="hi">Hello World</p>
    <script src="data.js">
```

```
const a=document.gEEElementById("hi");
a.setAttribute("style","color:red");
```

Hello World