# MASTER OF COMPUTER APPLICATIONS2021-23

CENTER FOR DEVELOPMENT OF ADVANCED COMPUTING

## FULL STACK DEVELOPMENT

## PRACTICAL FILE


**Submitted to:**

Mr. Saurabh Chhabra

**Submitted by:**

Pankaj

01711804421

# INDEX

| S.no | LAB EXERCISES | DATE |
|---|---|---|
| 1 | WAP to demonstrate the use of let, var & const variables in JS. | 26-4-22 |
| 2 | WAP to create react class component without creating react app. | 10-5-22 |
| 3 | WAP that has a function that accepts variable length argument and print their sum. | 10-5-22 |
| 4 | WAP to demonstrate anonymous function. | 17-5-22 |
| 5 | WAP to demonstrate the arrow function in JS. | 17-5-22 |
| 6 | WAP that executes the code in strict mode. | 17-5-22 |
| 7 | WAP that uses function created using function objects. | 17-5-22 |
| 8 | WAP to demonstrate the use of template literal. | 17-5-22 |
| 9 | WAP to demonstrate generator function in JS. | 17-5-22 |
| 10 | WAP to demonstrate object in JS having attributes fname, lname, age course &fullname, where fullname has function that Displays full name of the person. | 24-5-22 |
| 11 | Write a function in JS that creates person object same as in the above example. | 24-5-22 |
| 12 | WAP that has array of doctors, where each doctor is object with the fname, lname, salary. Also demonstrate the different loops for foreach to display the total salary of the doctors. | 24-5-22 |
| 13 | WAP that uses the map function to update the salary of the doctor by 40%. | 24-5-22 |
| 14 | WAP that uses reduce function that calculates the total salary of the doctors. | 24-5-22 |
| 15 | WAP that uses the filter function to return the details that have salary above 22 LPA. | 24-5-22 |
| 16 | WAP that has a function that uses setTimeOut to produce the employee id of the third employee and produces its details (fname ,lname, salary) after their increment the salary by 20% and prints it. | 31-5-22 |
| 17 | WAP to illustrate the concept of promise. | 31-5-22 |

| | | |
|---|---|---|
| 18 | WAP to illustrate the concept of async and await. | 31-5-22 |
| 19 | WAP to illustrate DOM Manipulation. | 7-6-22 |
| 20 | WAP to illustrate callbacks in JS. | 7-6-22 |
| 21 | WAP to illustrate the callback hell in JS. | 7-6-22 |
| 22 | WAP to illustrate Object and array destructuring in JS. | 7-6-22 |
| 23 | WAP to demonstrate state management using useState Hook in React. | 14-6-22 |
| 24 | WAP to demonstrate the use of useEffect Hook in React. | 14-6-22 |
| 25 | WAP to illustrate form in React. | 21-6-22 |
| 26 | WAP that displays student details and has a button that changes course name on click. | 21-6-22 |
| 27 | WAP to illustrate the use of useContext. | 21-6-22 |
| 28 | Write code to display employee details and updates the employee salary and display it in a higher component. | 21-6-22 |
| 29 | WAP to illustrate Redux concept. | 28-6-22 |
| 30 | WAP to fetch data from API. | 28-6-22 |
| 31 | WAP to illustrate the concept of routing in React. | 28-6-22 |
| 32 | WAP to display details of five students using react. Also, use CSS for making border, giving background color, text color, padding and margin. | 28-6-22 |
| 33 | Write code for reading and writing files (both synchronously and asynchronously) using node.js. | 5-7-22 |
| 34 | Write code for sending different html page as response based on the URL entered by client using node.js. | 5-7-22 |
| 35 | Write code for using path module in node.js. | 5-7-22 |
| 36 | Write code for creating a database, collection and inserting documents in mongoDB using node.js. | 12-7-22 |
| 37 | Write code for creating a database, collection and inserting documents in mongoDB using node.js and mongoose. | 12-7-22 |
| 38 | WAP to make a simple angular ‘Hello World’ app. | 12-7-22 |

# Lab Exercise – 1

WAP to demonstrate the use of let, var & const variables in JS.

Sol:

```
<html>
<title>
    JS6
</title>

<body>
    <script>
        // If 'a' is declared as var it can be redeclared
        var a = 10;
        const b = 60;
        document.write(a + ' ');
        document.write(b + ' ');
        {
            document.write(b + ' ');
            //If 'a' is declared using let it can't be redaclered and its scope is limited to block.
            let a = 20;
            document.write(a + ' ');
            a=30;
            document.write(a + ' ');
        }
        document.write(a + ' ');    // a is 10 again
        var a = 40;
        document.write(a + ' ');    //a is 40 now
        console.log(a);
    </script>
</body>

</html>
```

## Output

10 60 60 20 30 10 40

# Lab Exercise – 2

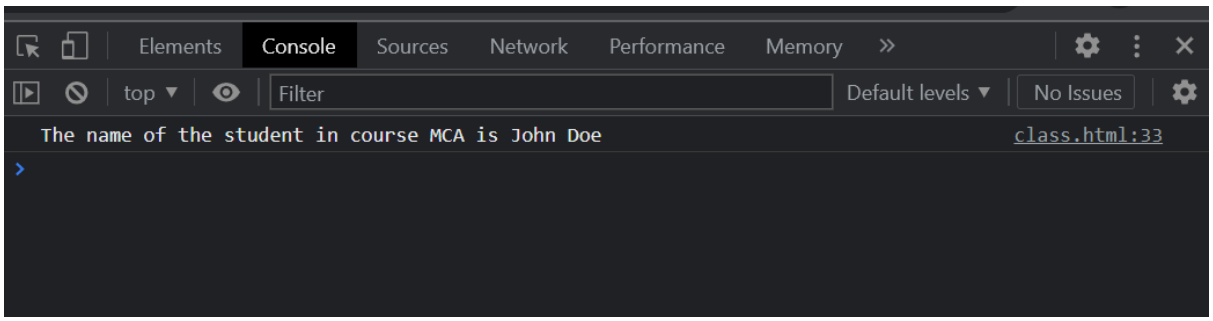
WAP to create react class component without creating react app.

Sol:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        class Person
        {
            constructor(name)
            {
                this.name = name
            }

        }
```

```
        class Student extends Person
        {
            constructor(name, course)
            {
                super(name)
                this.course = course
            }

        }

        const student1 = new Student('John Doe','MCA');
        console.log(`The name of the student in course ${student1.course} is ${student1.name}`)


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

## Output-



# Lab Exercise – 3

WAP that has a function that accepts variable lengthargument and print their sum.

Sol:

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

    <title>Document</title>
</head>
<body>
    <script>

function f1(...args){
    return args.reduce(function (acc, cur) {
        return acc + cur;
    })
}
document.write('Sum of 2,1 and 3 is '+ f1(2,3, 5));

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

Output-

Sum of 2,1 and 3 is 10

# Lab Exercise – 4

WAP to demonstrate anonymous function.

Sol:

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

<head>

    <title>Anonymous Functions</title>
</head>

<body>
    <script>
        f1 = function () {
            document.write("Anonymous function")
        }

        f1()
    </script>
</body>
</html>
```

Output-

Anonymous function

# Lab Exercise – 5

WAP to demonstrate the arrow function in JS.

Sol:

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

<head>

    <title>Arrow Functions</title>
</head>

<body>
    <script>
        double1 = (x) => {
            return 2 * x;
        }
        double2 = (x) => 3 * x;
        document.write(double1(5) + ' ');
        document.write(double2(6));

    </script>
</body>

</html>
```

Output-

10 18

# Lab Exercise – 6

WAP that executes the code in strict mode.

Sol:

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

<head>

    <title>Document</title>
</head>

<body>
    <script>
        "use strict"
        var product = (x, y) => x * y;

        var z = product(6, 5)

        document.write('Product of 6 and 5 is ' + z)

    </script>
</body>

</html>
```

Output-

Product of 6 and 5 is 30

# Lab Exercise - 7

WAP that uses function created using function objects.

Sol:

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

<head>

    <title>Document</title>
</head>

<body>
    <script>
        var sum = new Function("a", "b", "return a + b")
        a = 2
        b = 4
        x = sum(a, b)
        document.write('sum of ' + a + ' & ' + b + ' is ' + x)
    </script>
</body>

</html>
```

Output-

sum of 2 & 4 is 6

# Lab Exercise– 8

WAP to demonstrate the use of template literal.

Sol:

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

<head>

    <title>Document</title>
</head>

<body>
    <script>
        var sum = new Function("a", "b", "return a + b")
        a = 2
        b = 4
        x = sum(a, b)
        document.write(`sum of ${a} & ${b} is ${x}`)
    </script>
</body>

</html>
```

Output-

sum of 2 & 4 is 6

# Lab Exercise– 9

WAP to demonstrate generator function in JS.

Sol:

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

<head>

    <title>Document</title>
</head>

<body>
    <script>
        function* weekdays() {

            yield 'Monday'
            yield 'Tuesday'
            yield 'Wednesday'
            yield 'Thursday'
            yield 'Friday'
        }
        for (let day of weekdays()) {

            document.write(day, "</br>")

        }
    </script>
</body>

</html>
```

Output-

Monday
Tuesday
Wednesday
Thursday
Friday

# Lab Exercise – 10

WAP to demonstrate object in JS having attributes fname, lname, age course &fullname, where fullname has function that Displays full name of the person.

Sol:

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

<head>

    <title>Use of JS Objects</title>
</head>

<body>
    <script>
        var person = {

            fname: 'John',
            lname: 'Doe',
            age: 22,
            course: 'MCA',
            fullname: function () { return this.fname + " " + this.lname }

        }
        document.write(`Name of the student is ${person.fullname()}`)
    </script>
</body>

</html>
```

Output-

Name of the student is John Doe

# Lab Exercise– 11

Write a function in JS that creates person object same as in the above example.

Sol:

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

<head>

    <title>Use of JS function</title>
</head>

<body>
    <script>
        function person(fn, ln, age, course) {
            this.fname = fn
            this.lname = ln
            this.age = age
            this.course = course
            this.fullname = () =>
                document.write(`The name of student is ${this.fname} ${this.lname} and age is ${this.age}`)

        }

        var person1 = new person('John', 'Doe', 23, 'B.tech')
        person1.fullname()
    </script>
</body>

</html>
```

Output-

The name of student is John Doe and age is 23

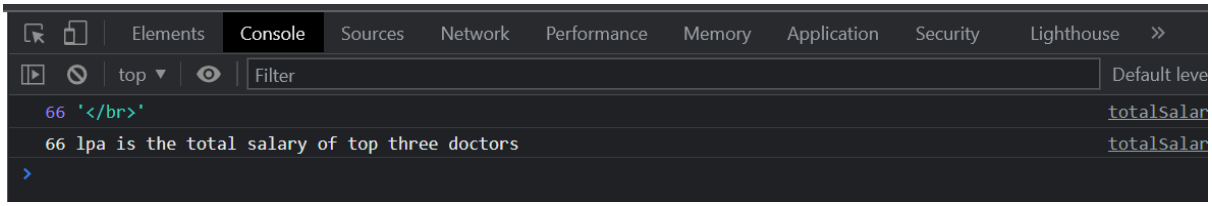
# Lab Exercise– 12

WAP that has array of doctors, where each doctor is object with the fname, lname, salary. Also demonstrate the different loops for foreach to display the total salary of the doctors.

Sol:

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

<head>
    <title>Use of JS Objects</title>
</head>

<body>
    <script>
        var doctors = [
            {
                fname: 'Sherry',
                lname: 'Verma',
                salaryinlpa: 21
            },
            {
                fname: 'Joy',
                lname: 'Mukherji',
                salaryinlpa: 22
            },
            {
                fname: 'Tsetan',
                lname: 'Nmagyal',
                salaryinlpa: 23
            }]

        total = 0
        for (let i = 0; i < doctors.length; i++) {
            total += doctors[i].salaryinlpa



        }
        console.log(total, "</br>")
        var total = 0
        doctors.forEach((doctor) => total += doctor.salaryinlpa)
        console.log(total + " lpa is the total salary of top three doctors")

    </script>
</body>

</html>
```

Output-


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
top
Filter
Default leve
66 '</br>'
totalSalar
66 lpa is the total salary of top three doctors
totalSalar

# Lab Exercise – 13

WAP that uses the map function to update the salary of the doctor by 40%.

Sol:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Use of JS Objects</title>
</head>
<body>
    <script>
        var doctors = [
            {
                fname: 'Sherry',
                lname: 'Verma',
                salaryinlpa: 21
            },
            {
                fname: 'Joy',
                lname: 'Mukherji',
                salaryinlpa: 22
            },
            {
                fname: 'Tsetan',
                lname: 'Nmagyal',
                salaryinlpa: 23
            }]
        var salary = doctors.map((doctor) => doctor.salaryinlpa * 1.4)

        document.write('Total salary of the doctor is ' + salary + 'lpa')
    </script>
</body>
</html>
```

Output-

Total salary of the doctor is 29.4,30.799999999999997,32.199999999999996lpa

# Lab Exercise – 14

WAP that uses reduce function that calculates the total salary of the doctors.

Sol:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Use of JS Objects</title>
</head>
<body>
    <script>
        var doctors = [
            {
                fname: 'Sherry',
                lname: 'Verma',
                salaryinlpa: 21
            },
            {
                fname: 'Joy',
                lname: 'Mukherji',
                salaryinlpa: 22
            },
            {
                fname: 'Tsetan',
                lname: 'Nmagyal',
                salaryinlpa: 23
            }]
```

```
        var total = doctors.reduce((salary, doctor) => {

            salary += doctor.salaryinlpa
            return salary
        }, 0)
        document.write(`The total salary is ${total} lpa `)
    </script>
</body>
</html>
```

Output-


The total salary is 66 lpa

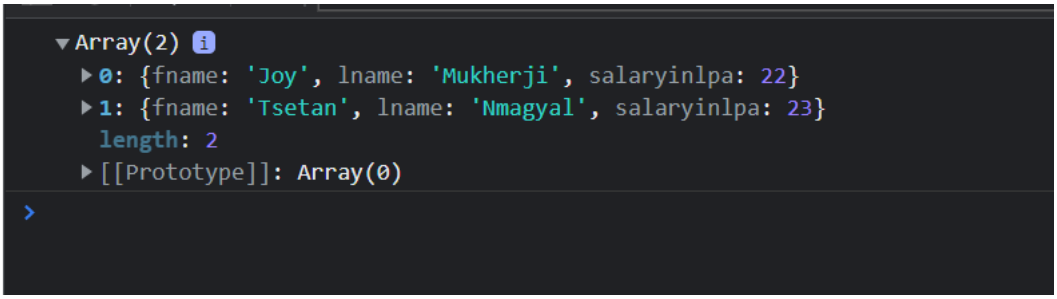
# Lab Exercise– 15

WAP that uses the filter function to return the details that have salary above 22 LPA.

Sol:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Use of filter function in array</title>
</head>
<body>
    <script>
        var doctors = [
            {
                fname: 'Sherry',
                lname: 'Verma',
                salaryinlpa: 21
            },
            {
                fname: 'Joy',
                lname: 'Mukherji',
                salaryinlpa: 22
            },
            {
                fname: 'Tsetan',
                lname: 'Nmagyal',
                salaryinlpa: 23
            }]
            var aspergrade = doctors.filter((doctor) => {
                if (doctor.salaryinlpa >= 22) {
                    return doctor
                }
            })
            console.log(aspergrade)
        </script>
    </body>
    </html>
```

Output-


Array(2)
0: {fname: 'Joy', lname: 'Mukherji', salaryinlpa: 22}
1: {fname: 'Tsetan', lname: 'Nmagyal', salaryinlpa: 23}
length: 2
[[Prototype]]: Array(0)

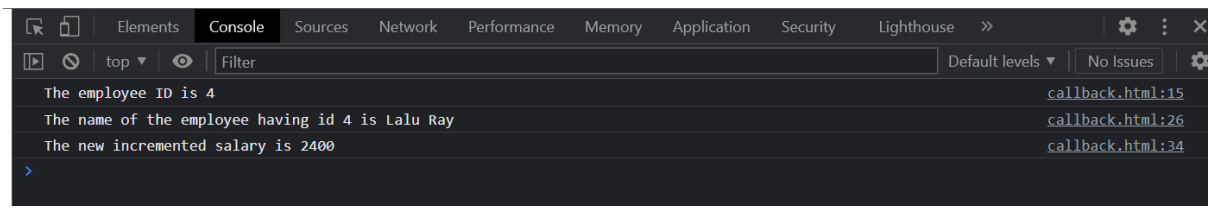
# Lab Exercise – 16

WAP that has a function that uses setTimeOut to produce the employee id of the third employee and produces its details (fname ,lname, salary) after their increment the salary by 20% and prints it.

Sol:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Use of callback</title>
</head>
<body>
    <script>
        func1 = (cb) => {
            setTimeout(() => {
                employeeId = [1, 2, 3, 4, 5]
                a = employeeId[3]
                console.log(`The employee ID is ${employeeId[3]}`)
                cb();
            }, 2000)
        }
        func2 = (cb) => {
            setTimeout(() => {
                empfour = {
                    fname: 'Lalu',
                    lname: 'Ray',
                    salary: 2000
                }
                console.log(`The name of the employee having id ${a} is ${empfour.fname} ${empfour.lname}`)
                cb()
            }, 3000, a)
        }

        func3 = () => {
            setTimeout(() => {
                incrementedSalary = empfour.salary * 1.2
                console.log(`The new incremented salary is ${incrementedSalary}`)
            }, 4000, empfour.salary)
        }

        func1(function () { func2(function () { func3() }) })
    </script>
</body>

</html>
```

Output-



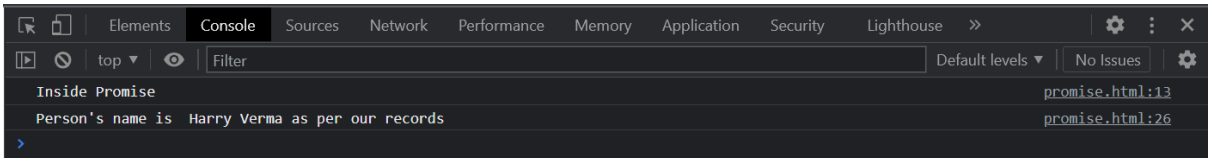
# Lab Exercise –17

WAP to illustrate the concept of promise.

Sol:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        promise1 = new Promise((resolve,reject)=>{

            console.log('Inside Promise' )
            const person ={
                fname: 'Harry',
                lname: 'Verma'
            }
            a =1
            if(a==1){
                resolve(` ${person.fname} ${person.lname}`)
            }
            else
            reject('No data received')

        })
        promise1.then((xval)=>{console.log(`Person's name is ${xval} as per our records`)}).catch((val)=>{console.
    </script>
</body>
</html>
```

**Output-**



# Lab Exercise– 18

WAP to illustrate the concept of async and await.

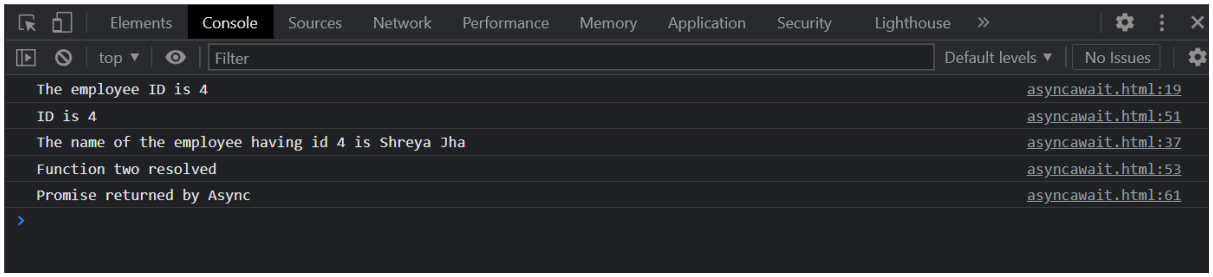
Sol:

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Use of callback</title>
</head>

<body>
    <script>
        func1 = () => {
            return new Promise((resolve, reject) => {
                setTimeout(() => {
                    x = 2
                    employeeId = [1, 2, 3, 4, 5]
                    a = employeeId[3]
                    console.log(`The employee ID is ${employeeId[3]}`);
                    if (x == 2)
                        resolve(a)
                    else
                        reject()

                }, 2000)
            })
        }
```

```
        func2 = () => {
            return new Promise((resolve, reject) => {
                setTimeout(() => {
                    x = 2;
                    empfour = {
                        fname: 'Shreya',
                        lname: 'Jha',
                        salary: 20000
                    }
                    console.log(`The name of the employee having id ${a} is ${empfour.fname} ${empfour.lname}`)
                    if (x == 2) {
                        resolve(empfour)

                    }
                    else
                        reject('Error occured in function 2')

                }, 3000)
            })
        }
```

```
48          async function asynAwaitEx() {
49              try {
50                  const value1 = await func1();
51                  console.log(`ID is ${value1}`);
52                  const value2 = await func2();
53                  console.log(`Function two resolved`)
54
55              }
56              catch (err) {
57                  console.log(`Error occured is ${err}`)
58
59              }
60          }
61          asynAwaitEx().then(() => console.log('Promise returned by Async'));
62      </script>
63  </body>
64
65  </html>
```

Output-



# Lab Exercise– 19

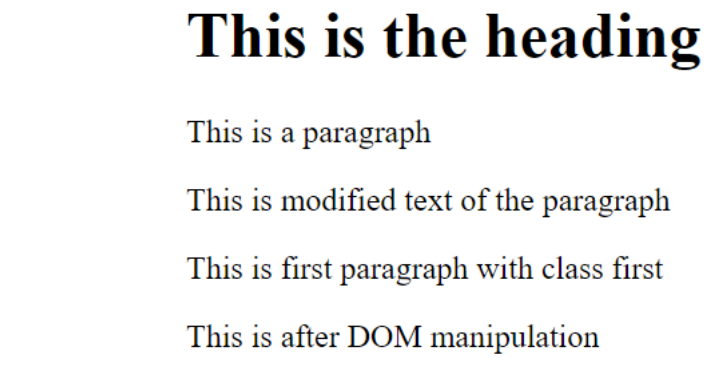
WAP to illustrate DOM Manipulation.

Sol:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

<h1>This is the heading</h1>

<div>
    <p>This is a paragraph</p>
    <p id ="one">This is paragraph with id one</p>
    <p class="first">This is first paragraph with class first</p>
    <p class="first">This is second paragraph with class first</p>
</div>
    <script src = "domManipulation.js"></script>
</body>
</html>
```

```
elementOne = document.getElementById('one')

elementOne.innerHTML = 'This is modified text of the paragraph'


classFirst = document.getElementsByClassName('first')

classFirst[1].innerHTML = 'This is after DOM manipulation'

console.log(classFirst)

parentElement = classFirst[0].parentNode

console.log(parentElement)


pargraphCreated =document.createElement("p")

pargraphCreated.innerHTML = 'This is new created paragraph'

document.getElementById('One').nextSibling
document.body.appendChild(pargraphCreated)
```

Output-



# Lab Exercise– 20

WAP to illustrate callbacks in JS.

Sol:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Use of callback</title>
</head>
<body>
    <script>
        func1 = (cb) => {
            setTimeout(() => {
                employeeId = [1, 2, 3, 4, 5]
                a = employeeId[3]
                console.log(`The employee ID is ${employeeId[3]}`)
                cb();
            }, 2000)
        }
        func2 = (cb) => {
            setTimeout(() => {
                empfour = {
                    fname: 'Sherry',
                    lname: 'Verma',
                    salary: 20000
                }
                console.log(`The name of the employee having id ${a} is ${empfour.fname} ${empfour.lname}`)
                cb()
            }, 3000, a)
        }

        func3 = () => {
            setTimeout(() => {
                incrementedSalary = empfour.salary * 1.2
                console.log(`The new incremented salary is ${incrementedSalary}`)
            }, 4000, empfour.salary)
        }

        func1(function () { func2(function () { func3() }) })
    </script>
</body>

</html>
```

Output-

```
The employee ID is 4
The name of the employee having id 4 is Sherry Verma
The new incremented salary is 24000
>
```

# Lab Exercise – 21

WAP to illustrate the callback hell in JS.

Sol:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
func1 = ()=>{
    setTimeout(()=>{
        employeeId = [1,2,3,4,5]
        a= employeeId[3]
        setTimeout(()=>{
            empfour = {
                fname: 'John',
                lname: 'Deep',
                salary:20000
            }
            console.log(`The name of the employee having id ${a} is ${empfour.fname} ${empfour.lname}`)
            setTimeout(()=>{
                incrementedSalary = empfour.salary*1.2
                console.log(`The new incremented salary is ${incrementedSalary}`)
            },empfour.salary)
            },2000,a)
},2000)
```

```
}
func1()
</script>
</body>
</html>
```

Output-

```
The name of the employee having id 4 is John Deep
The new incremented salary is 24000
>
```

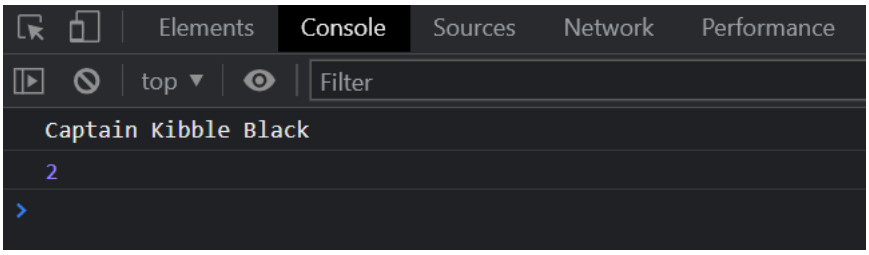
# Lab Exercise - 22

WAP to illustrate Object and array destructuring in JS.

Sol:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>

        function func1() {

            const pet= {
                name: 'Captain',
                food: 'Kibble',
                color: 'Black'
            };
            const { name, food, color } = pet;
            console.log(name, food, color)
            a = [1, 2, 3, 4, 5]
            const [x1, x2, ...rest] = a
            console.log(x2)
        }
     func1()
    </script>
</body>

</html>
```

Output-



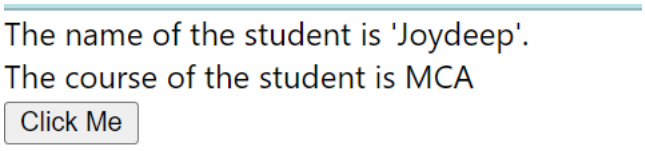
# Lab Exercise– 23

WAP to demonstrate state management using useState Hook in React.

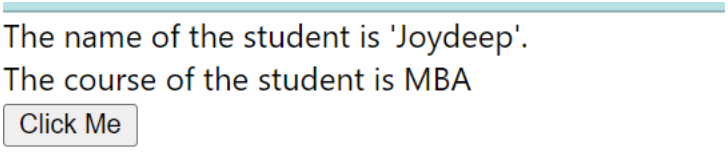
Sol:

```jsx
import React,{useState} from 'react';

function App(){
  const [course,setCourse] = useState('MCA');
  const changeCourse = ()=>{
    setCourse('MBA');
  }
  return(
  <>
    <div> The name of the student is 'Joydeep'.</div>
    <div>The course of the student is {course} </div>
    <button type='button' onClick={changeCourse}> Click Me</button>
  </>);
}

export default App;
```

Output-




**After clicking on Click Me**



# Lab Exercise– 24

WAP to demonstrate the use of useEffect Hook in React.

Sol:

```jsx
import React,{useState,useEffect} from 'react';
function App(){
  const [course,setCourse] = useState('MCA');
  const changeCourse = ()=>{
    setCourse('MBA');
  }
  useEffect( ()=>{
    console.log('hello');
    setTimeout( ()=>{ alert('hello'); }, 3000);
 });
  return(
  <>
    <div> The name of the student is 'Joydeep'.</div>
    <div>The course of the student is {course} </div>
    <button type='button' onClick={changeCourse}> Click Me</button>
  </>);
}
export default App;
```

Output-

# Lab Exercise– 25

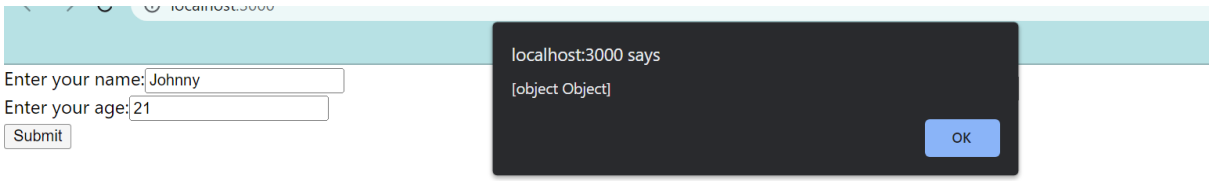
WAP to illustrate form in React.

Sol:

```jsx
import { useState } from 'react';

function App() {
  const [inputs, setInputs] = useState({});

  const handleChange = (event) => {
    const name = event.target.name;
    const value = event.target.value;
    setInputs(values => ({ ...values, [name]: value }))
  }

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(inputs);
  }
```

```jsx
  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>Enter your name:
          <input
            type="text"
            name="username"
            value={inputs.username || ""}
            onChange={handleChange}
          />
        </label>
      </div>
      <div>
        <label>Enter your age:
          <input
            type="number"
            name="age"
            value={inputs.age || ""}
            onChange={handleChange}
          />
        </label>
      </div>
      <input type="submit" />
    </form>
  )
}

export default App;
```

# Output-


Enter your name: Johnny
Enter your age: 21
Submit
localhost:3000 says
[object Object]
OK

# Lab Exercise– 26

WAP that displays student details and has a button that changes course name on click.

Sol:

```
import React, { Component } from 'react';

class Student extends React.Component {
    constructor(props) {
        super(props)
            this.state = {
            course: this.props.course
        }
    }
    changeCourse = () => {

        if (this.state.course != 'MCA')
            this.setState({ course: 'MCA' })
        else
            this.setState({ course: 'MBA' })

    }
    render() {
        return (<div>
            Student Name is: {this.props.name}<br></br>
            Age is: {this.props.age}<br></br>
            Course is:{this.state.course}
            <button onClick={this.changeCourse}>Click ME!</button>

        </div>);

    }
}
export default Student;
```

Output-

Student Name is: John Doe
Age is: 22
Course is:MCA Click ME!

**After clicking-**

# Lab Exercise– 27

WAP to illustrate the use of useContext.

Sol:

```jsx
import { useState, createContext, useContext } from "react";

const UserContext = createContext();
function App() {
  const [user, setUser] = useState("John Doe");

  return (
    <UserContext.Provider value={user}>
      <h1>{`Hello ${user}!`}</h1>
      <Component1 user={user} />
    </UserContext.Provider>
  );
}
```

```jsx
function Component1() {
  return (
    <>
      <h1>Component 1</h1>
      <Component2 />
    </>
  );
}
function Component2() {
  return (
    <>
      <h1>Component 2</h1>
      <Component3 />
    </>
  );
}
function Component3() {
  const user = useContext(UserContext);
  return (
    <>
      <h1>Component 3</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}
export default App;
```

Output-

# Hello John Doe!

# Component 1

# Component 2

# Component 3

## Hello John Doe again!

# Lab Exercise– 28

Write code to display employee details and updates the employee salary and display it in a higher component.

Sol:

App.js

```
import Employee from "./Employee";

function App() {

    return (<>
        <Employee name = 'Rajat' ID = '100' age = '23' totalSalary ='30000' basicSalary = '15000' HRA = '8000'
        miscAllowance = '7000'  />

        </>);

    }

export default App;
```

Employee.js

```
import React, { Component } from 'react';
import Salary from './Salary';

class Employee extends React.Component {
    constructor(props) {
        super(props)

        this.state = {
            newSalary: ''
        }
    }
    updatedSalary = (salary) => {

        this.setState({ newSalary: salary })

    }
```

```
    render() {
        return (<div>
            <p><h1>Employee Component</h1></p>
            Employee Name:{this.props.name}
            <p>
                Employee ID:{this.props.ID}
            </p>
            <p>
                Employee age:{this.props.age}
            </p>
            <p>
                Employee totalSalary:{this.props.totalSalary}
            </p>
            <p>
                Updated Salary:{this.state.newSalary}
            </p>
            <Salary basicSalary={this.props.basicSalary} HRA={this.props.HRA}
            miscAllowance={this.props.miscAllowance} setEmployeeSalary={this.updatedSalary} />
        </div>);
    }

}

export default Employee;
```

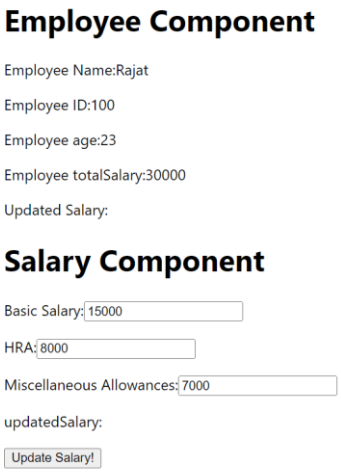
Salary.js

```
import React, {Component} from 'react';

class Salary extends React.Component
{
constructor(props)
{
    super(props)

    this.state = {
        basicSalary: this.props.basicSalary,
        HRA: this.props.HRA,
        miscAllowance: this.props.miscAllowance,
        salary:''

    }
}
changeSalary = () =>{
let updatedSalary = parseInt(this.refs.basicSalary.value) + parseInt(this.refs.HRA.value) +
parseInt(this.refs.miscAllowance.value)
this.setState({salary:updatedSalary})
this.props.setEmployeeSalary(updatedSalary)
}
```

```
render()
{
    return(<div>


        <p><h1>Salary Component</h1></p>
        Basic Salary:<input type = 'text' defaultValue = {this.state.basicSalary} ref = 'basicSalary'/>
        <p>
        HRA:<input type = 'text' defaultValue = {this.state.HRA} ref = 'HRA'/>
        </p>
        <p>
        Miscellaneous Allowances:<input type = 'text' defaultValue = {this.state.miscAllowance}
        ref = 'miscAllowance' />
        </p>
        <p>
            updatedSalary:{this.state.salary}
        </p>
        <button onClick={this.changeSalary}>Update Salary! </button>


    </div>);
}

}
export default Salary;
```

Output-

Before updating the salary-




After updating the salary-

# Employee Component

Employee Name:Rajat

Employee ID:100

Employee age:23

Employee totalSalary:30000

Updated Salary:30000

# Salary Component

Basic Salary: 15000

HRA: 8000

Miscellaneous Allowances: 7000

updatedSalary:30000

Update Salary!

# Lab Exercise– 29

WAP to illustrate Redux concept.

Sol:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import {createStore} from 'redux';
```

Creating actions-

```
const increment = () => {
  return {
    type: 'INCREMENT'
  }

}

const decrement = () => {
  return {
    type: 'DECREMENT'
  }

}
```

Creating reducers-

```
const counter = (state = 0, action) => {

  switch (action.type) {

    case 'INCREMENT':
      return state + 1;

    case 'DECREMENT':
      return state - 1;

  }
}
```

Creating store-

```
//store- globalizestate

let store = createStore(counter);
```

Subscribing to store and dispatching the actions–

```
//subscribe
store.subscribe(() =>console.log(store.getState()))

//dispatch
store.dispatch(increment());
store.dispatch(decrement());
store.dispatch(decrement());
store.dispatch(decrement());
store.dispatch(decrement());
```

Output-

```
1
0
-1
-2
-3
```

# Lab Exercise– 30

WAP to fetch data from API.

Sol:

App.js

```
import React from 'react';
import './App.css';
import './FetchData';
import FetchData from './FetchData';

function App() {
  return (
    <div className="App">
      {`Fetching dog images from the API`}
      <FetchData />
    </div>
  );
}

export default App;
```

FetchData.js

```
import React, {useState, useEffect} from 'react';
import './App.css';

function FetchData() {
    // 2. Create our *dogImage* variable as well as the *setDogImage* function via useState
    // We're setting the default value of dogImage to null, so that while we wait for the
    // fetch to complete, we dont attempt to render the image
  let [dogImage, setDogImage] = useState(null)

    // 3. Create out useEffect function
  useEffect(() => {
    fetch("https://dog.ceo/api/breeds/image/random/3")
    .then(response => response.json())
        // 4. Setting *dogImage* to the image url that we received from the response above
    .then(data => setDogImage(data.message))
  },[])

  return (
    <div className="App">
        {/* 5. Returning an img element for each url, again with the value of our src set to the image url */}
    {dogImage && dogImage.map((dog) => <img width={"200px"} height={"200px"} src={dog}></img>)}
    </div>
  );
}

export default FetchData;
```

Output-

Fetching dog images from the API

# Lab Exercise – 31

WAP to illustrate the concept of routing in React.

Sol:

App.js

```jsx
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
```

Layout.js

```jsx
import { Outlet, Link } from "react-router-dom";

const Layout = () => {
  return (
    <>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/blogs">Blogs</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Outlet />
    </>
  )
};

export default Layout;
```

Home.js

```
const Home = () => {
  return <h1>Home</h1>;
};

export default Home;
```

Blogs.js

```
const Blogs = () => {
    return <h1>Blog Articles</h1>;
};

export default Blogs;
```

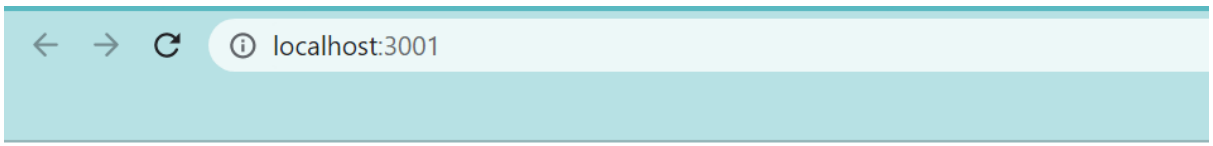
Contact.js

```
const Contact = () => {
    return <h1>Contact Me</h1>;
};

export default Contact;
```

NoPage.js

```
const NoPage = () => {
    return <h1>404</h1>;
};

export default NoPage;
```

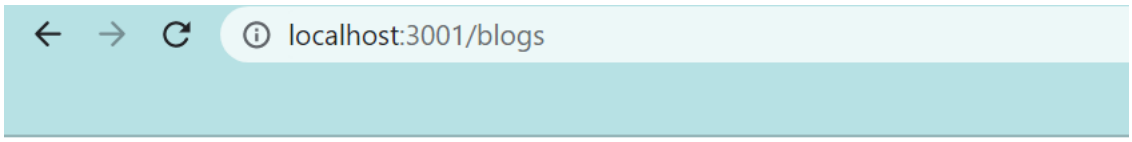
Output-




- Home
- Blogs
- Contact

# Home

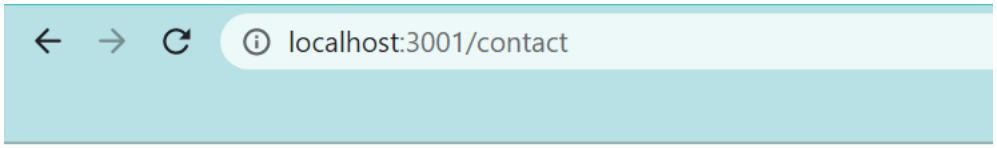
/blogs




- Home
- Blogs
- Contact

# Blog Articles

/contact




- Home
- Blogs
- Contact

# Contact Me

# Lab Exercise– 32

WAP to display details of five students using react. Also, use CSS for making border, giving background color, text color, padding and margin.

Sol:

Student.js

```
import React from 'react';

function Student(props){
    return(
        <div style ={{border:"1px solid black"}}>
            <p>
                <label>Student ID:<b>{props.data.ID}</b></label>
            </p>
            <p>
                <label>Student Name:<b>{props.data.Name}</b></label>
            </p>
            <p>
                <label>Course:<b>{props.data.Course}</b></label>
            </p>
        </div>
    )
}
export default Student;
```

App.js

```
import React from 'react';
import Student from './Student.js';

const students = [
  {ID:'1234',Name:'John',Course:'MCA'},
  {ID:'1235',Name:'Jenny',Course:'MCA'},
  {ID:'1236',Name:'Sherry',Course:'MBA'},
  {ID:'1237',Name:'Dev',Course:'B.tech'},
  {ID:'1238',Name:'Mic',Course:'MCA'},
]
function App(){
  const student =students.map((student)=><Student key={student.ID} data={student} />)
  return(
    <div>
      {student}
    </div>
  );
}
export default App;
```

Output-

Student ID:**1234**

Student Name:**John**

Course:**MCA**

---

Student ID:**1235**

Student Name:**Jenny**

Course:**MCA**

---

Student ID:**1236**

Student Name:**Sherry**

Course:**MBA**

---

Student ID:**1237**

Student Name:**Dev**

Course:**B.tech**

# Lab Exercise– 33

Write code for reading and writing files (both synchronously and asynchronously) using node.js.

Sol:

1. Reading from file asynchronously

```
const fs = require('fs');

fs.readFile('./hello.txt',(err,data)=>{
    if(err){
        console.log(error);
    }
    console.log(data.toString());
})
```

Output-

```
PS C:\Users\sinha\OneDrive\Desktop\fileSystemNode> node script.js
Helloooo there!!This is so cool
```

2. Reading from file synchronously-

```
const fs = require('fs');

const file = fs.readFileSync('hello.txt');
console.log(file.toString());
```

Output-

```
PS C:\Users\sinha\OneDrive\Desktop\fileSystemNode> node script.js
Helloooo there!!This is so cool
```

3. Writing to file asynchronously-

```
const fs = require('fs');

const file = 'hello_world.txt';
const data = 'Hello, World!';

fs.writeFile(file, data, error => {
  if (error) {
    throw error;
  } else {
    console.log(`Successfully wrote '${data}' to ${file}.`);
  }
});
```

Output-

```
PS C:\Users\sinha\OneDrive\Desktop\fileSystemNode> node script.js
Successfully wrote 'Hello, World!' to hello_world.txt.
```

hello_world.txt

```
1    Hello, World!
```

4. Write to file synchronously-

```
const fs = require('fs');

const file = 'hello_world.txt';
const data = 'Hello, World!';
fs.writeFileSync(file,'Sad to see you go',err=>{
    if(err){
        console.log(err);
    }
})
```

Output-

hello_world.txt

```
1    Sad to see you go
```

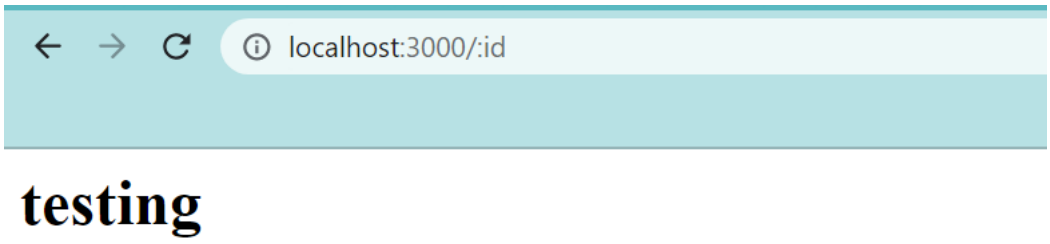
# Lab Exercise – 34

Write code for sending different html page as response based on the URL entered by client using node.js.

Sol:

```
const express = require('express');

const app = express();

app.get('/:id',(req,res)=>{
    console.log(req.params);
    res.send('<h1>testing</h1>');
})

app.listen(3000);
```

Output-




```
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
{ id: ':id' }
{ id: 'favicon.ico' }
```

# Lab Exercise– 35

Write code for using path module in node.js.

Sol:

```
var path = require("path");

// Normalization
console.log('normalization : ' + path.normalize('/test/test1//2slashes/1slash/tab/..'));

// Join
console.log('joint path : ' + path.join('/test', 'test1', '2slashes/1slash', 'tab', '..'));

// Resolve
console.log('resolve : ' + path.resolve('path.js'));

// extName
console.log('ext name : ' + path.extname('path.js'));
```

Output-

```
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
normalization : \test\test1\2slashes\1slash
joint path : \test\test1\2slashes\1slash
resolve : C:\Users\sinha\OneDrive\Desktop\node\path.js
ext name : .js
```

# Lab Exercise - 36

Write code for creating a database, collection and inserting documents in mongoDB using node.js.

Sol:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url,(err, client) =>{
  if (err)
    throw err;
  var dbo = client.db("tempDb");
    var firstobj = {name: "John Doe",email: "johndoe@gmail.com"};
    dbo.collection("Customer").insertOne(firstobj,(err,res)=>{
        if(err)
            console.log(err);
    console.log("Document inserted!");
    client.close();
  });
});
```

Output-

```
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Document inserted!
[nodemon] clean exit - waiting for changes before restart
```

```
> use tempDb
switched to db tempDb
> show collections
Customer
> db.Customer.find().pretty()
{
        "_id" : ObjectId("62d3be279039bc6638f62fa8"),
        "name" : "John Doe",
        "email" : "johndoe@gmail.com"
}
```

# Lab Exercise– 37

Write code for creating a database, collection and inserting documents in mongoDB using node.js and mongoose.

Sol:

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/randomdb',{useNewUrlParser:true})
.then(()=>console.log("Connection successfull..."))
.catch((err)=>console.log(err));

const details = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: String
})

const Detail = new mongoose.model("Detail",details);

const det = new Detail({
    name: 'John Doe',
    email: 'johndoe@gmail.com'
})
det.save();
```

Output-

```
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Connection successfull...
```

```
> use randomdb
switched to db randomdb
> show collections
details
> db.details.find()
{ "_id" : ObjectId("62d3c221382f03acf7bdb4f6"), "name" : "John Doe", "email" : "johndoe@gmail.com", "__v" : 0 }
```

# Lab Exercise - 38

WAP to make a simple angular ‘Hello World’ app.

Sol:

app.component.html

```
<!--The content below is only a placeholder and can be replaced.-->
<div>
  <h1>
    {{ title }}!
  </h1>
</div>
```

app.component.css

```
div {
  text-align: center;
}
```

app.component.ts

```
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styles: [
    `div{
      color:red;
      text-align:center;
    }`
  ]
})
export class AppComponent {
  title = "Hello World";
}
```

app.module.ts

```ts
import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";

import { AppComponent } from "./app.component";

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

index.html

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

<head>
  <meta charset="utf-8">
  <title>Angular</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>

<body>
  <app-root></app-root>
</body>
</html>
```

Output-