

Sr.No.	Name of Practical	Date	CO	Sign
1	Create an application to demonstrate Node.js Modules	26-09-24	CO1	
2	Events <ul style="list-style-type: none"> Create an application to demonstrate various Node.js Events write a node js program to print table of Ten and Eleven using event emitter. 	1-10-24	CO1	
3	Functions <ul style="list-style-type: none"> Create an application/program to demonstrate Node.js user defined Functions Create an application/program to demonstrate Node.js system defined Functions Write a node js program to print table of 15 on console using function. 	3-10-24	CO1	
4	File Handling <ul style="list-style-type: none"> Using File Handling demonstrate all basic file operations (Create, write, append, read, delete) Write a node js program to Copy the contents of one file to another Write a node js program to Open a file and append some data in the same. write a node js program to program to print table of Five in a file and paste it on another file 	15-10-24	CO1	
5	Web server (HTTP Module) <ul style="list-style-type: none"> Create an HTTP Server and perform different operations on it Write a node js program to create a web server and print below given pattern on web Browser. <pre> ***** ***** ***** *** ** *</pre>	17-10-24	CO1	

	<ul style="list-style-type: none"> Write a node js program to create a web 8-server and print table of 12 on web Browser. 			
6	Database: <ul style="list-style-type: none"> Create an application to establish a connection with the MySQL database and perform below given operations on it. <ul style="list-style-type: none"> Create Employee Table with Primary Key Insert at least 10 records Read the Records Create a Library database with the table (Without Primary key) and perform below given operations on it <ul style="list-style-type: none"> Insert at least 10 records Read the records Update the employee location with Mumbai whose location is Delhi Delete 	8-11-24	CO2 CO2	
7	ReactJS: <ul style="list-style-type: none"> Write a programme to demonstrate Expressions , large block of HTML and conditional statement in JSX 	5-12-24	CO3	
8	Components: <ul style="list-style-type: none"> Create a react JS application which demonstrates Functional Components Create a react JS application which demonstrates Class Components Create a react JS application to implement Component Life Cycle Create an application in ReactJS to implement Rendering 	7-12-24	CO3	
9	Create an application in ReactJs to Import and Export the files (Components)	7-12-24	CO3 CO4	
10	Create an application to implement <ul style="list-style-type: none"> a. State b. Props 	9-12-24	CO3 CO4	
11	Create an application in reactJs to implement DOM events <ul style="list-style-type: none"> a. Click b. Change 	9-12-24	CO3 CO4	

	c. Blur d. Keypress and Keyup			
12	Create an application in ReactJs form and add client and server side validation	10-12-24	CO4	
13	Create an application to implement three types of Hooks a. useState b. Context c. Effect	10-12-24	CO4	

PRACTICAL - 1

a) Create an application to demonstrate Node.js Modules

Code:-

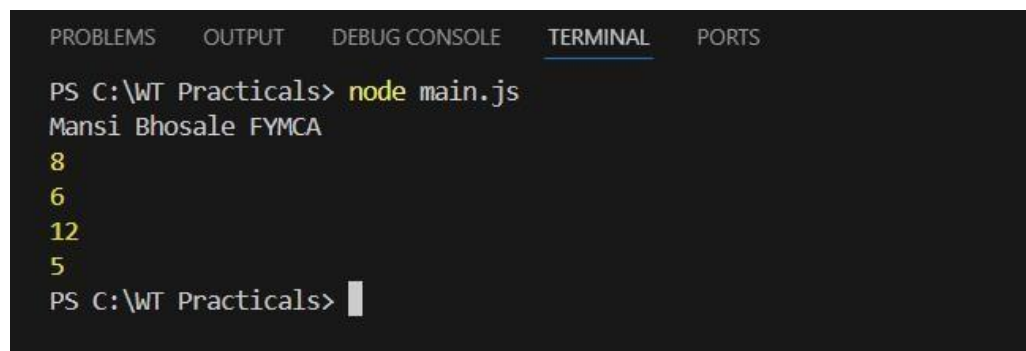
Calculator.js

```
module.exports = {  
  add: (a, b) => a + b,  
  subtract: (a, b) => a - b,  
  multiply: (a, b) => a * b,  
  divide: (a, b) => a / b  
};
```

Main.js

```
const calculator =  
require('./Calculator');  
console.log('Mansi Bhosale FYMCA ');  
console.log(calculator.add(5, 3));  
console.log(calculator.subtract(10, 4));  
console.log(calculator.multiply(2, 6));  
console.log(calculator.divide(15, 3));
```

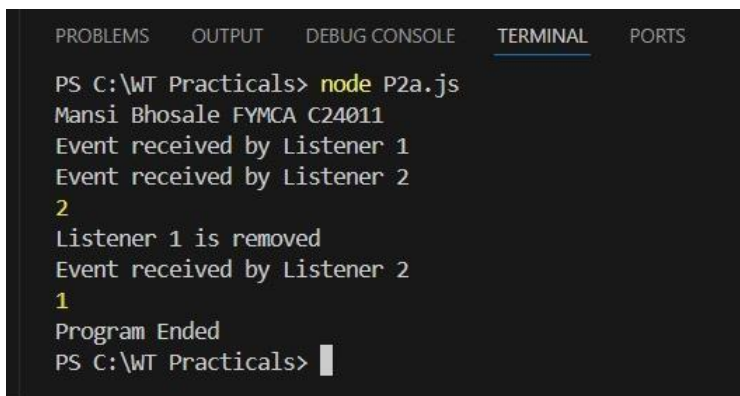
Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
PS C:\WT Practicals> node main.js  
Mansi Bhosale FYMCA  
8  
6  
12  
5  
PS C:\WT Practicals> 
```

PRACTICAL – 2**Events****1) Create an application to demonstrate various Node.js Events**

```
const events=require('events');
const eventEmitter=new
events.EventEmitter();function Listener1(){
console.log('Mansi Bhosale FYMCA C24011');
console.log('Event received by Listener 1');
}
function Listener2(){
console.log('Event received by Listener 2');
}
eventEmitter.addListener('write', Listener1);
eventEmitter.on('write', Listener2);
eventEmitter.emit('write');
console.log(eventEmitter.listenerCount('write'));
eventEmitter.removeListener('write',Listener1)
console.log('Listener 1 is removed');
eventEmitter.emit('write');
console.log(eventEmitter.listenerCount('write'));
console.log('Program Ended');
```

Output:

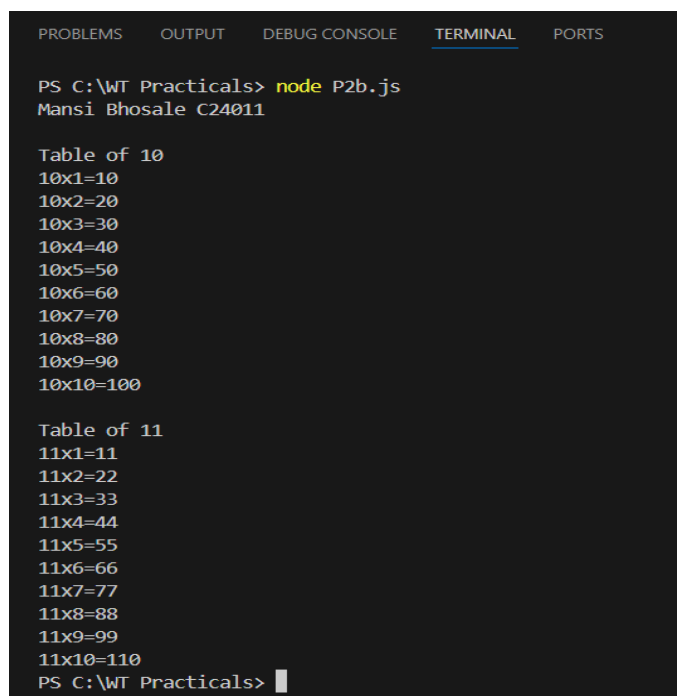
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\WT Practicals> node P2a.js
Mansi Bhosale FYMCA C24011
Event received by Listener 1
Event received by Listener 2
2
Listener 1 is removed
Event received by Listener 2
1
Program Ended
PS C:\WT Practicals>
```

2) Write a node js program to print table of Ten and Eleven using event emitter.

Code:-

```
const events=require('events');
const eventEmitter=new
events.EventEmitter();console.log("Mansi
Bhosale C24011"); function table(num){
  for (let i = 1; i <= 10; i++){
    console.log(num + "x" + i + "=" + num * i);
  }
  eventEmitter.on('tableOf10', () =>
table(10));eventEmitter.on('tableOf11', () =>
table(11));console.log("\nTable of 10");
eventEmitter.emit('tableOf10');
console.log("\nTable of 11");
eventEmitter.emit('tableOf11');
```

Output:



```
PS C:\WT Practicals> node P2b.js
Mansi Bhosale C24011

Table of 10
10x1=10
10x2=20
10x3=30
10x4=40
10x5=50
10x6=60
10x7=70
10x8=80
10x9=90
10x10=100

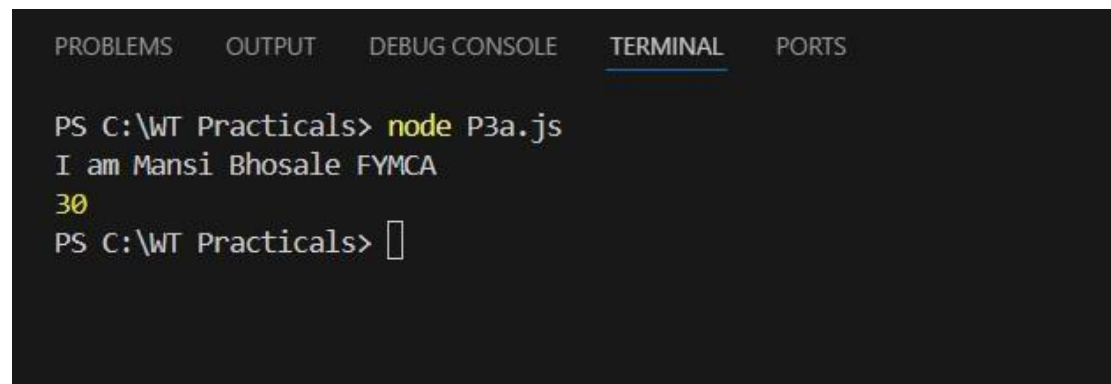
Table of 11
11x1=11
11x2=22
11x3=33
11x4=44
11x5=55
11x6=66
11x7=77
11x8=88
11x9=99
11x10=110
PS C:\WT Practicals>
```

PRACTICAL-3**Function****1) Create an application/program to demonstrate Node.js user defined Functions****Code:-**

```
function displayresult(para)
{
console.log(para);
}

function calculate(x,y,mycallback)
{
let sum=x+y;
mycallback(sum);
}

console.log ('I am Mansi Bhossale FYMCA');
calculate(20,10,displayresult);
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

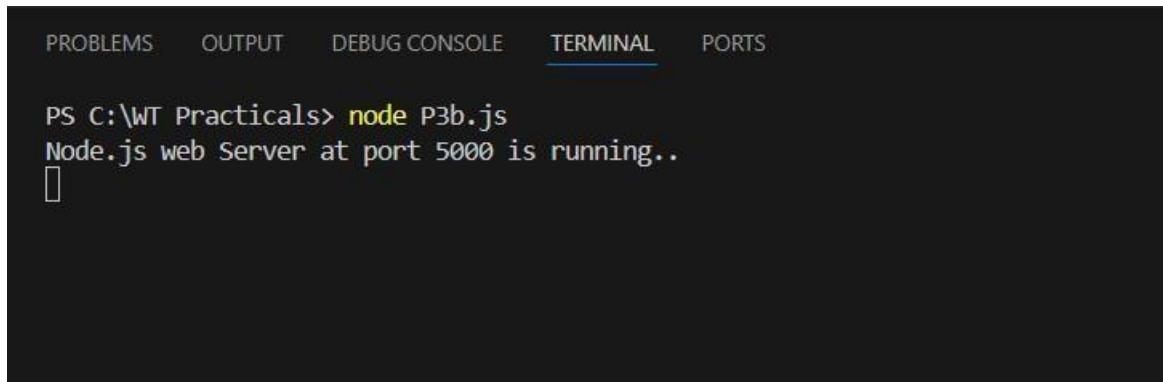
PS C:\WT Practicals> node P3a.js
I am Mansi Bhossale FYMCA
30
PS C:\WT Practicals> 
```

2) Create an application/program to demonstrate Node.js system defined Functions

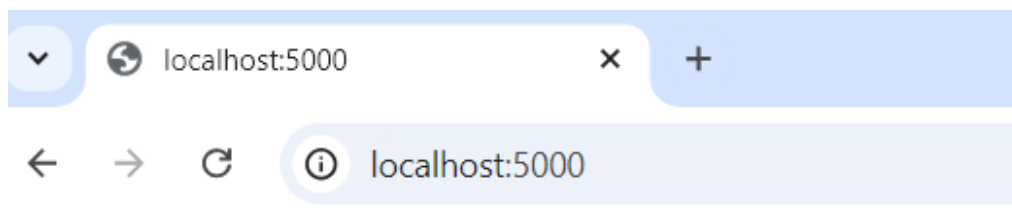
Code:-

```
var http = require('http');  
var server = http.createServer(function (req, res)  
{  
  res.write("I am Mansi Bhosale FYMCA\n");  
  
  res.write("Hello VS Studio this is WT Program");  
  res.end();  
});  
server.listen(5000);  
console.log('Node.js web Server at port 5000 is running..')
```

Output:



The screenshot shows the VS Code interface with the 'TERMINAL' tab selected. The terminal displays the command 'node P3b.js' and its output: 'Node.js web Server at port 5000 is running..' followed by a cursor.



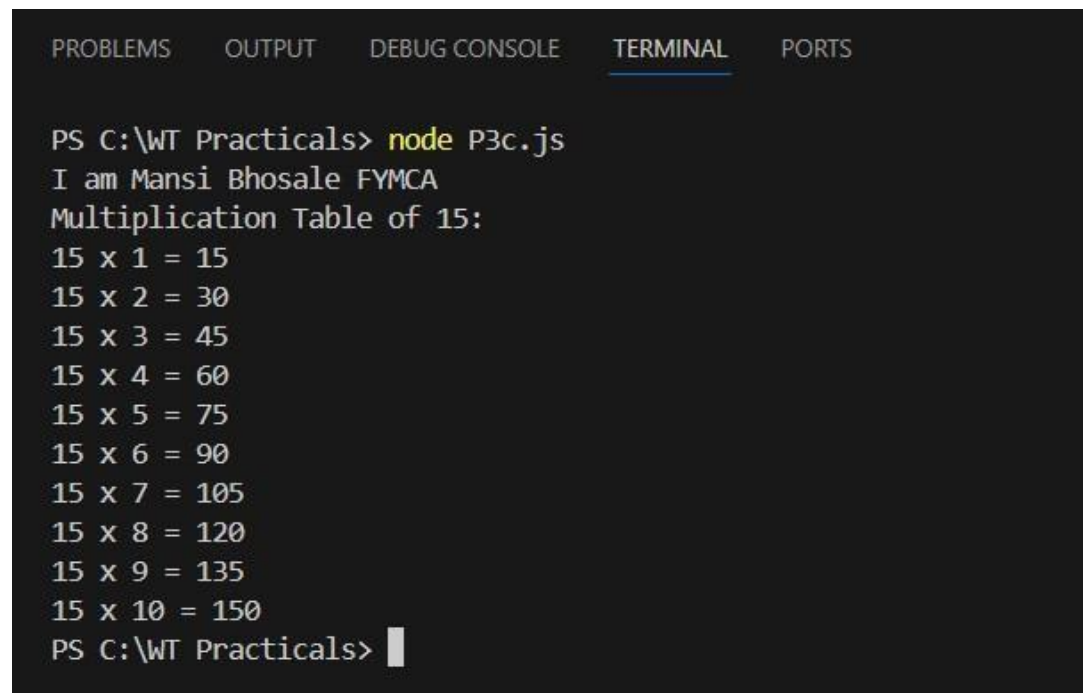
I am Mansi Bhosale FYMCA
Hello VS Studio this is WT Program

3) Write a node js program to print table of 15 on console using function.

Code:-

```
function printTableOf15() {  
    const number = 15;  
    console.log('I am Mansi Bhosale FYMCA');  
    console.log(`Multiplication Table of ${number}:`);  
    for (let i = 1; i <= 10; i++) {  
        console.log(`${number} x ${i} = ${number * i}`);  
    }  
}  
printTableOf15();
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
  
PS C:\WT Practicals> node P3c.js  
I am Mansi Bhosale FYMCA  
Multiplication Table of 15:  
15 x 1 = 15  
15 x 2 = 30  
15 x 3 = 45  
15 x 4 = 60  
15 x 5 = 75  
15 x 6 = 90  
15 x 7 = 105  
15 x 8 = 120  
15 x 9 = 135  
15 x 10 = 150  
PS C:\WT Practicals> |
```

PRACTICAL-4**File Handling****1) Using File Handling demonstrate all basic file operations (Create, write, append, read, delete)****Code:**

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

const filePath = 'example.txt';

console.log('I am Mansi Bhosale FYMCA');

function createAndWriteFile() {
  fs.writeFileSync(filePath, 'Hello, this is the initial content in the file!', 'utf8');
  console.log('File created and initial content written.');
```



```
}

function appendToFile() {
  fs.appendFileSync(filePath, '\nAppending more content to the file.',
    'utf8'); console.log('Content appended to the file.');
```



```
}

function readFile() {
  const data = fs.readFileSync(filePath, 'utf8');
  console.log('Reading from the file:\n' +
    data);
}
```



```
function deleteFile() {
  if (fs.existsSync(filePath)) {
    fs.unlinkSync(filePath);
```

```
console.log('File deleted successfully.');
```

```
} else {
```



```
console.log('File does not exist.');
```

```
}
```

```
}
```



```
function runp4a() {
```



```
createAndWriteFile()
```



```
appendToFile();
```



```
readFile();
```

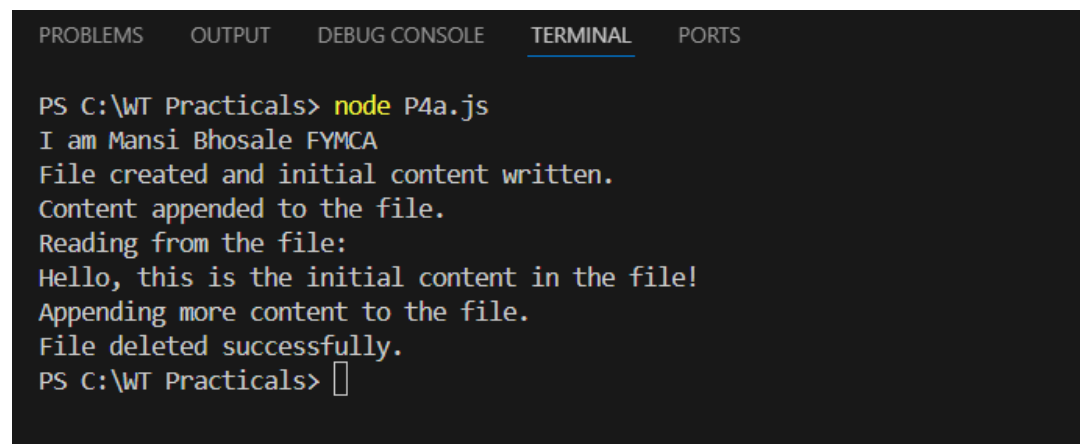


```
deleteFile();
```

```
}
```



```
runp4a();
```

Output:-

The screenshot shows a terminal window with a dark background and light-colored text. At the top, there are five tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. The terminal content shows a command prompt 'PS C:\WT Practicals>' followed by the command 'node P4a.js'. The output of the script is displayed line by line: 'I am Mansi Bhosale FYMCA', 'File created and initial content written.', 'Content appended to the file.', 'Reading from the file:', 'Hello, this is the initial content in the file!', 'Appending more content to the file.', 'File deleted successfully.', and finally the prompt 'PS C:\WT Practicals>' with a cursor.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
```

```
PS C:\WT Practicals> node P4a.js
```

```
I am Mansi Bhosale FYMCA
```

```
File created and initial content written.
```

```
Content appended to the file.
```

```
Reading from the file:
```

```
Hello, this is the initial content in the file!
```

```
Appending more content to the file.
```

```
File deleted successfully.
```

```
PS C:\WT Practicals> 
```

2) Write a node js program to Copy the contents of one file to another

Code:

```
const fs = require('fs'); const
sourceFile = 'file.txt';

const destinationFile = 'destination.txt';

fs.copyFile(sourceFile, destinationFile, (err) => {

  if (err) {

    console.error("Error copying file:", err);

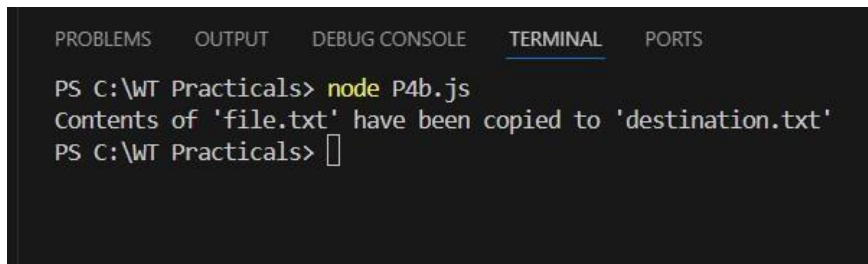
  } else {

    console.log(`Contents of '${sourceFile}' have been copied to '${destinationFile}'`);

  }

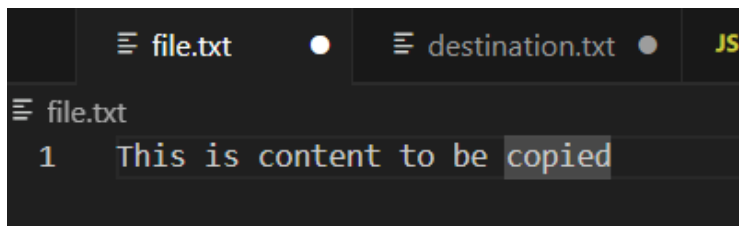
});
```

Output:-



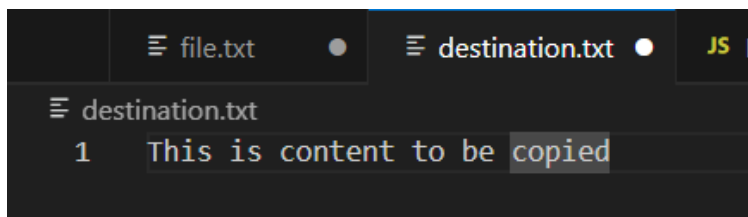
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\WT Practicals> node P4b.js
Contents of 'file.txt' have been copied to 'destination.txt'
PS C:\WT Practicals> 
```



```
file.txt  destination.txt  JS

file.txt
1 This is content to be copied
```



```
file.txt  destination.txt  JS P

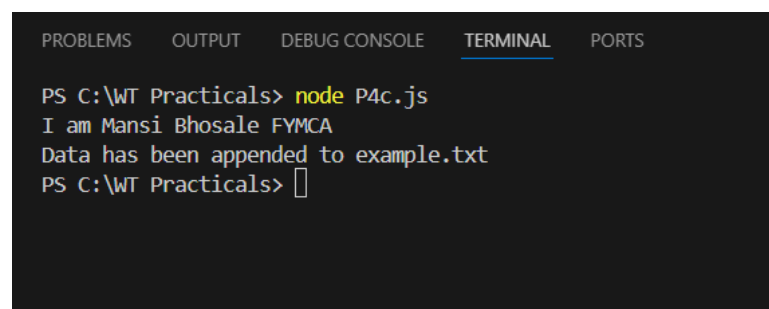
destination.txt
1 This is content to be copied
```

3) Write a node js program to Open a file and append some data in the same.

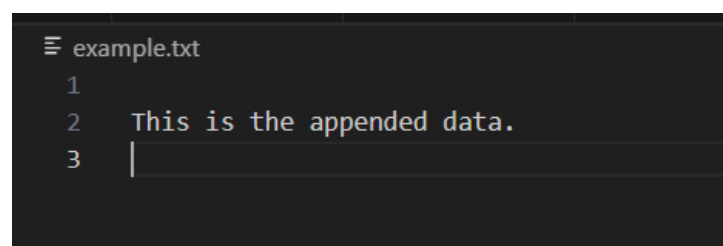
Code:-

```
const fs = require('fs');  
console.log('I am Mansi Bhosale FYMCA');  
function openAndp4c(filePath, dataToAppend) {try  
{  
fs.appendFileSync(filePath, dataToAppend, 'utf8');  
console.log(`Data has been appended to ${filePath}`);  
} catch (error) {  
console.error("An error occurred while appending data:", error.message);  
}  
}  
const filePath = 'example.txt';  
const dataToAppend = '\nThis is the appended  
data.';openAndp4c(filePath, dataToAppend);
```

Output:-



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
  
PS C:\WT Practicals> node P4c.js  
I am Mansi Bhosale FYMCA  
Data has been appended to example.txt  
PS C:\WT Practicals> 
```



```
example.txt  
1  
2 This is the appended data.  
3 |
```

4) Write a node js program to program to print table of Five in a file and paste it on another file**Code:-**

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

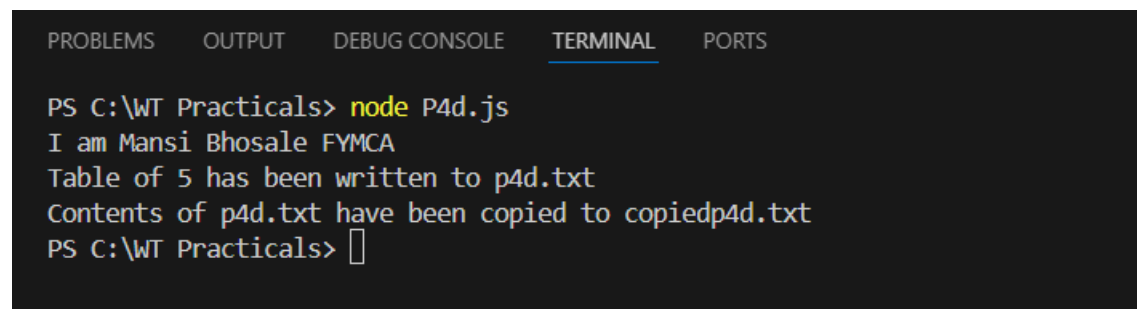
console.log('I am Mansi Bhosale FYMCA');

function createp4d() {
  let tableContent = "";
  for (let i = 1; i <= 10; i++) { tableContent
    += `5 x ${i} = ${5 * i}\n`;
  }
  return tableContent;
}

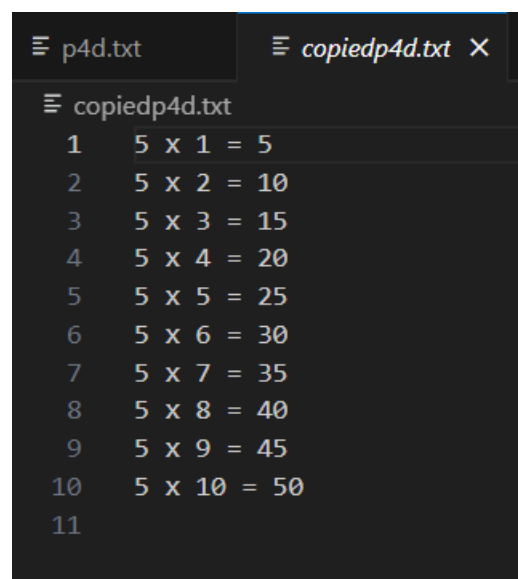
function writeTableToFile(filePath, content) {
  try {
    fs.writeFileSync(filePath, content, 'utf8');
    console.log(`Table of 5 has been written to ${filePath}`);
  } catch (error) {
    console.error("An error occurred while writing the table:", error.message);
  }
}

function copyFileContent(sourceFile, destinationFile) {
  try {
    const data = fs.readFileSync(sourceFile, 'utf8');
    fs.writeFileSync(destinationFile, data, 'utf8');
    console.log(`Contents of ${sourceFile} have been copied to ${destinationFile}`);
  } catch (error) {
    console.error("An error occurred while copying the file:", error.message);
  }
}
```

```
}  
}  
  
const tableFilePath = 'p4d.txt';  
const copyFilePath =  
'copiedp4d.txt';const tableContent =  
createp4d();  
writeTableToFile(tableFilePath, tableContent);  
copyFileContent(tableFilePath, copyFilePath);
```

Output:-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
  
PS C:\WT Practicals> node P4d.js  
I am Mansi Bhosale FYMCA  
Table of 5 has been written to p4d.txt  
Contents of p4d.txt have been copied to copiedp4d.txt  
PS C:\WT Practicals> 
```



```
p4d.txt  copiedp4d.txt X  
  
copiedp4d.txt  
1    5 x 1 = 5  
2    5 x 2 = 10  
3    5 x 3 = 15  
4    5 x 4 = 20  
5    5 x 5 = 25  
6    5 x 6 = 30  
7    5 x 7 = 35  
8    5 x 8 = 40  
9    5 x 9 = 45  
10   5 x 10 = 50  
11
```

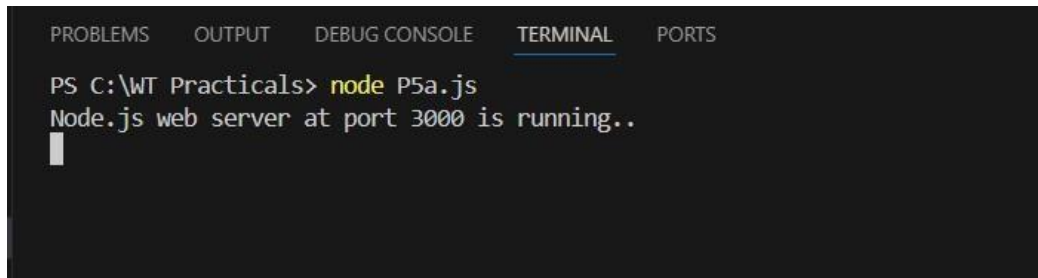
PRACTICAL-5**Web Server(HTTP Module)****1) Create an HTTP Server and perform different operations on it****Code:-**

```
var http = require('http');

var server = http.createServer(function (req, res)
{
  if (req.url == '/'){
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<html><body><p>This is home page of Mansi Bhosale.</p></body></html>');
    res.end();
  }
  else if (req.url == "/student")
  {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<html><body><p>This is student page of Mansi Bhosale.</p></body></html>');
    res.end();
  }
  else if (req.url == "/admin")
  {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<html><body><p>This is admin page of Mansi Bhosale.</p></body></html>');
    res.end();
  }
  else
    res.end('Invalid Request!');
});

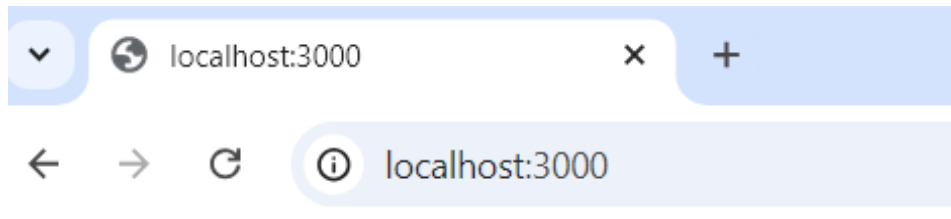
server.listen(3000);

console.log('Node.js web server at port 3000 is running..')
```

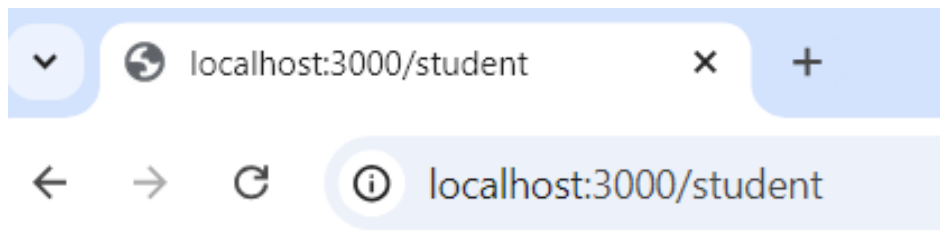

Output:-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

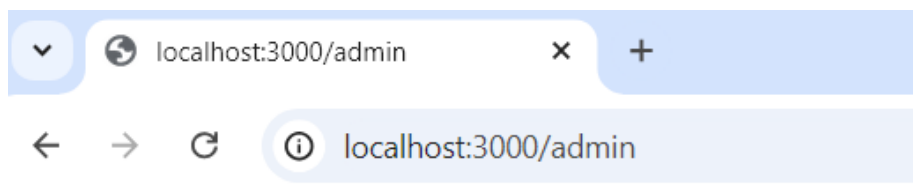
PS C:\WT Practicals> node P5a.js
Node.js web server at port 3000 is running..
|
```



This is home page of Mansi Bhosale.



This is student page of Mansi Bhosale.



This is admin page of Mansi Bhosale.

2) Write a node js program to create a web server and print below given pattern on web Browser

Code:-

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

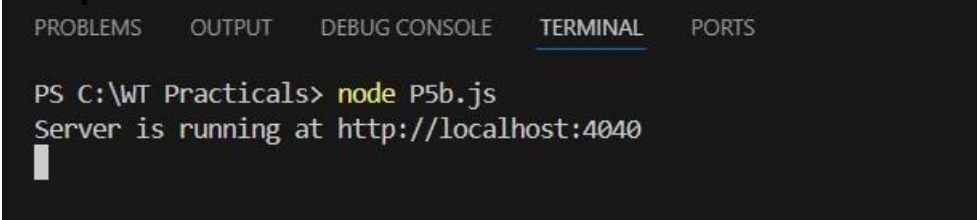
const pattern = `
*****<br>
*****<br>
*****<br>
***<br>
**<br>
*<br>
`;

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.write(`<h1>Pattern
Output</h1><pre>${pattern}</pre>`);res.end();
});

const port = 4040;

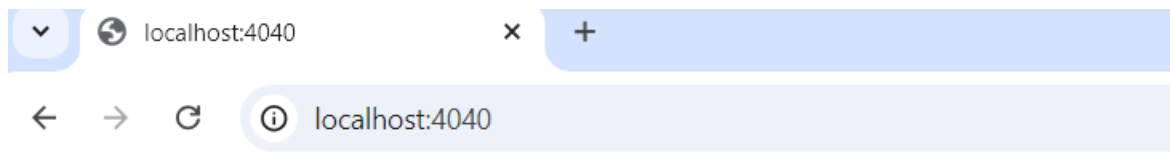
server.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\WT Practicals> node P5b.js
Server is running at http://localhost:4040
```



Pattern Output

**

3) Write a node js program to create a web server and print table of 12 on web Browser.

Code:-

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

let tableOf12 = "<h1>Multiplication Table of 12</h1><pre>";

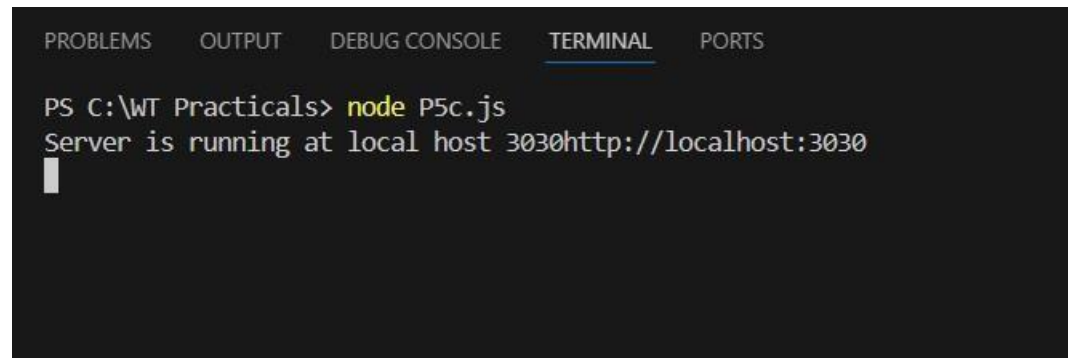
for (let i = 1; i <= 10; i++) {
  tableOf12 += `12 x ${i} = ${12 * i}<br>`;
}

tableOf12 += "</pre>";

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.write(tableOf12);
  res.end();
});

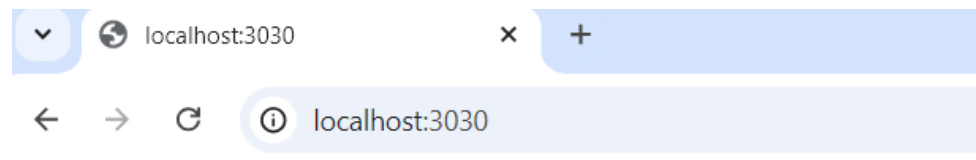
const port = 3030;

server.listen(port, () => {
  console.log(`Server is running at local host 3030http://localhost:${port}`);
});
```

Output:-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\WT Practicals> node P5c.js
Server is running at local host 3030http://localhost:3030
```



Multiplication Table of 12

12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
12 x 6 = 72
12 x 7 = 84
12 x 8 = 96
12 x 9 = 108
12 x 10 = 120

Steps to connect to database in Node.js

1. Download XAMPP CONTROL PANEL.
2. Once installed start Apache and MySQL in the control panel.
3. Type localhost in browser and then click on phpMyAdmin.
4. Create a new folder in VS code and write the code for database connectivity in the file.

PRACTICAL -6

1. Create an application to establish a connection with the MySQL database and perform below given operations on it.

- a. Create Employee Table with Primary Key
- b. Insert at least 10 records
- c. Read the Records

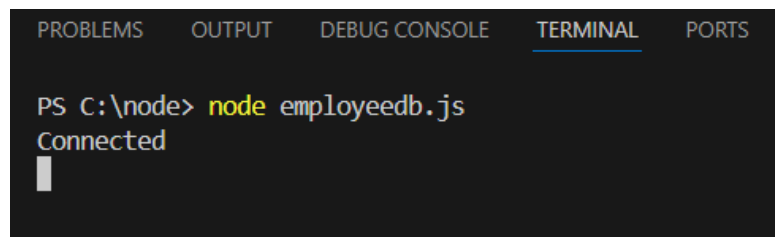
For database connection

Code:

```
var mysql=require('mysql');
var con = mysql.createConnection({
  host:"localhost",
  user:"root",
  password:""

});
con.connect(function(err){
  if(err) throw err;
  console.log("Connected");
})
```

Output:

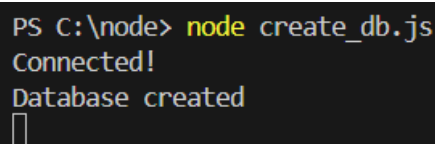


The screenshot shows a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. Below the tabs, the command prompt shows 'PS C:\node> node employeedb.js'. The output of the command is 'Connected', followed by a white cursor line.

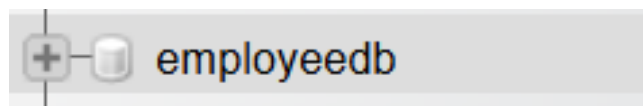
Creating database employeedb.**Code:**

```
var mysql=require('mysql');
var con = mysql.createConnection({
  host:"localhost",
  user:"root",
  password:""

});
con.connect(function(err){
  if(err) throw err;
  con.query("CREATE DATABASE employeedb",function(err,result){
    if(err) throw err;
    console.log("Database created");
  });
})
```

Output:

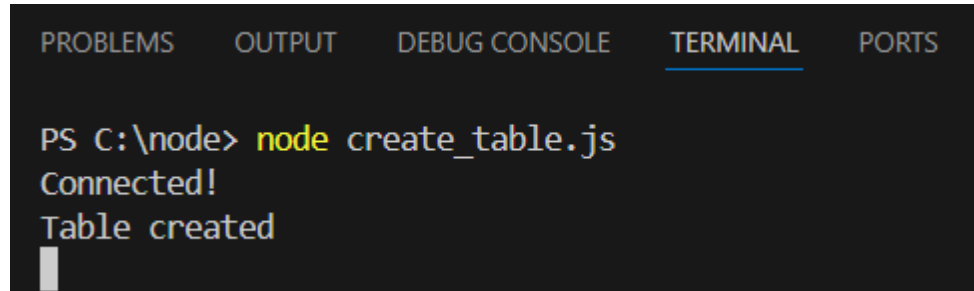
```
PS C:\node> node create_db.js
Connected!
Database created
█
```

**a) Create Employee Table with Primary Key.****Code:**

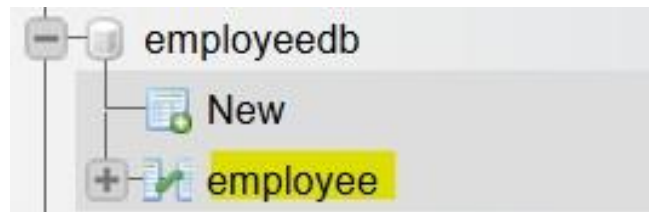
```
var mysql=require('mysql');
var con = mysql.createConnection({
  host:"localhost",
  user:"root",
  password:"",
  database : "employeedb"

});
con.connect(function(err){
  if(err) throw err;
  console.log("Connected!");
  var sql="CREATE TABLE Employee(Name VARCHAR(255),Department VARCHAR(255))";
  con.query(sql,function(err,result){
```

```
    if(err) throw err;  
    console.log("Table created");  
  });  
});
```

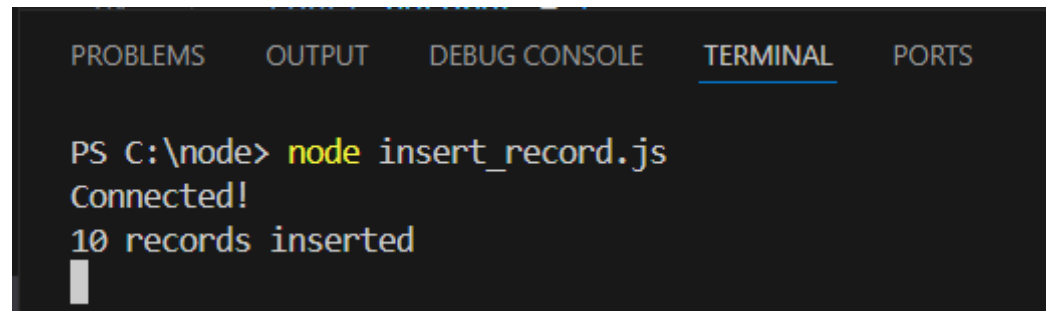
Output:

```
PS C:\node> node create_table.js  
Connected!  
Table created
```

**b) Insert at least 10 records.****CODE:**

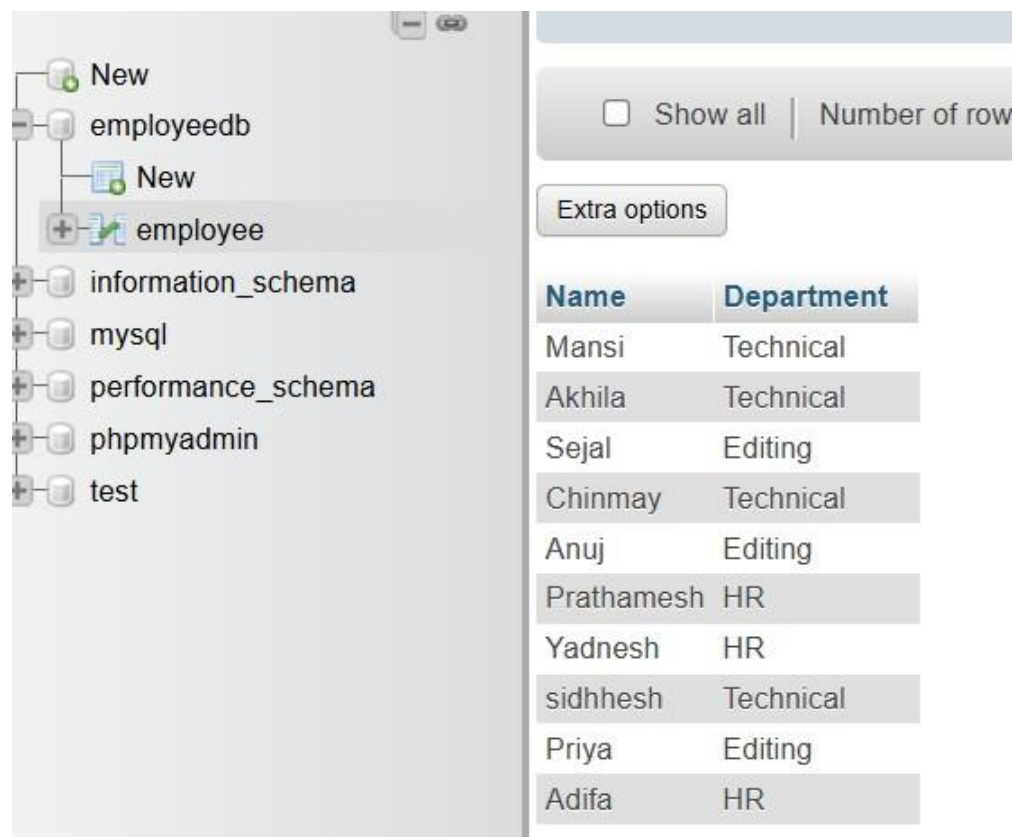
```
const mysql = require('mysql');  
var con = mysql.createConnection({  
  user: "root",  
  password: "",  
  database: "employeedb"  
});  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  const records = [  
    ['Mansi', 'Technical'],  
    ['Akhila', 'Technical'],  
    ['Sejal', 'Editing'],  
    ['Chinmay', 'Technical'],  
    ['Anuj', 'Editing'],  
    ['Prathamesh', 'HR'],  
    ['Yadnesh', 'HR'],  
    ['sidhmesh', 'Technical'],  
    ['Priya', 'Editing'],  
    ['Adifa', 'HR']  
  ];  
});
```

```
var sql = "INSERT INTO employee (name, department) VALUES ?";
con.query(sql, [records], function (err, result) {
  if (err) throw err;
  console.log("10 records inserted");
});
});
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\node> node insert_record.js
Connected!
10 records inserted
```




Name	Department
Mansi	Technical
Akhila	Technical
Sejal	Editing
Chinmay	Technical
Anuj	Editing
Prathamesh	HR
Yadnesh	HR
sidhmesh	Technical
Priya	Editing
Adifa	HR

c) Reading the Records.**Code:**

```
var mysql=require('mysql');
var con = mysql.createConnection({
  host:"localhost",
  user:"root",
  password:"",
  database:"employeeeb"

});
con.connect(function(err){
  if(err) throw err;

  con.query("SELECT* FROM employee",function(err,result,fields){
    if(err) throw err;
    console.log(result);
  });
});
```

Output:

```
PS C:\node> node reading_record.js
[
  RowDataPacket { Name: 'Mansi', Department: 'Technical' },
  RowDataPacket { Name: 'Akhila', Department: 'Technical' },
  RowDataPacket { Name: 'Sejal', Department: 'Editing' },
  RowDataPacket { Name: 'Chinmay', Department: 'Technical' },
  RowDataPacket { Name: 'Anuj', Department: 'Editing' },
  RowDataPacket { Name: 'Prathamesh', Department: 'HR' },
  RowDataPacket { Name: 'Yadnesh', Department: 'HR' },
  RowDataPacket { Name: 'sidhhesh', Department: 'Technical' },
  RowDataPacket { Name: 'Priya', Department: 'Editing' },
  RowDataPacket { Name: 'Adifa', Department: 'HR' }
]
```

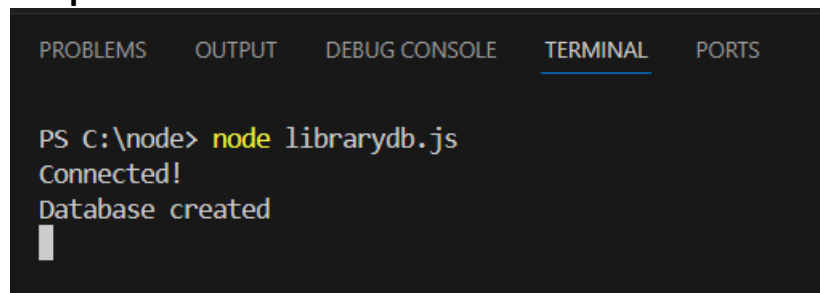
2. Create a Library database with the table (Without Primary key) and perform below given operations on it.

Database creation

code:

```
var mysql=require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: ""
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE librarydb", function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

Output:

A screenshot of a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. The terminal shows the command 'PS C:\node> node librarydb.js' being executed. Below the command, the output 'Connected!' and 'Database created' is displayed on two separate lines. A white cursor is visible at the end of the first line of output.

```
PS C:\node> node librarydb.js
Connected!
Database created
```

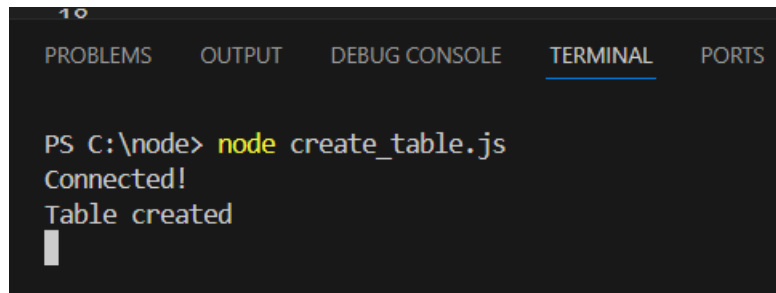
Table creation

code:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "librarydb"
});

var sql=con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql="CREATE TABLE library (name VARCHAR(10),location VARCHAR(15))"
  con.query(sql, function (err, result) {
```

```
if (err) throw err;  
console.log("Table created");  
});  
});
```

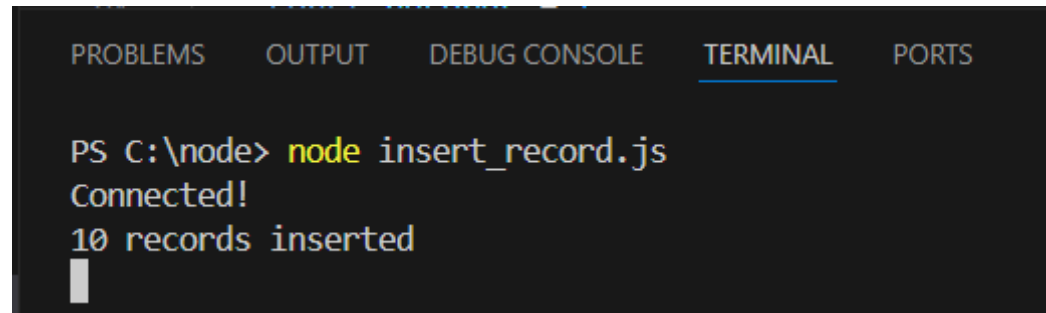
Output:

```
PS C:\node> node create_table.js  
Connected!  
Table created
```

**a) Insert at least 10 records****Code:**

```
const mysql = require('mysql');  
var con = mysql.createConnection({  
  user: "root",  
  password: "",  
  database: "librarydb"  
});  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  const records = [  
    ['Mansi', 'Bangalore'],  
    ['Ram', 'Bangalore'],  
    ['Omkar', 'Pune'],  
    ['Chinmay', 'Mumbai'],  
    ['Akhila', 'Delhi'],  
    ['Yadnesh', 'Delhi'],  
    ['Anuj', 'Pune'],  
    ['Prathmesh', 'Mumbai'],  
    ['Sidhhesh', 'Bangalore'],  
    ['Kapil', 'Pune']  
  ];  
  var sql = "INSERT INTO library (name, location) VALUES ?";  
  con.query(sql, [records], function (err, result) {  
    if (err) throw err;
```

```
console.log(`10 records inserted.`);  
});  
});
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
  
PS C:\node> node insert_record.js  
Connected!  
10 records inserted
```

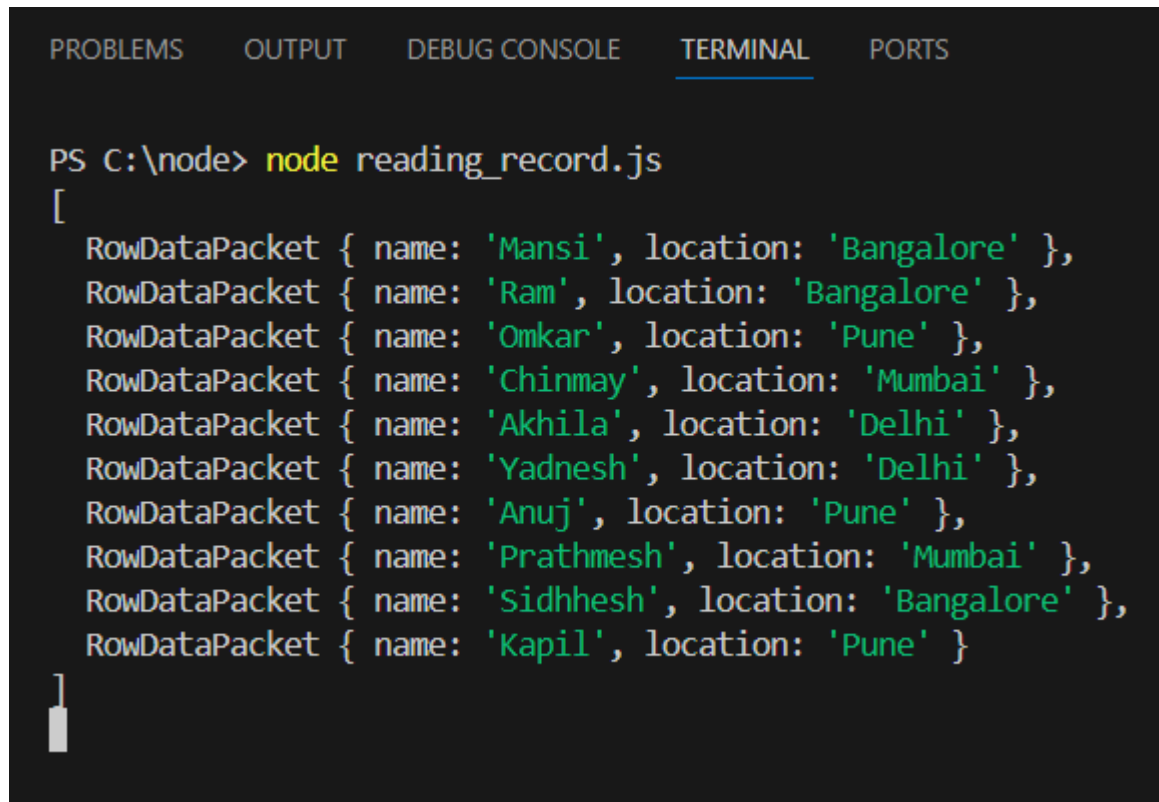
name	location
Mansi	Bangalore
Ram	Bangalore
Omkar	Pune
Chinmay	Mumbai
Akhila	Delhi
Yadnesh	Delhi
Anuj	Pune
Prathmesh	Mumbai
Sidhhesh	Bangalore
Kapil	Pune

☐ Show all | Number

b) Read the records**Code:**

```
var mysql=require('mysql');  
var con = mysql.createConnection({  
  host:"localhost",  
  user:"root",  
  password:"",  
  database:"librarydb"
```

```
});  
con.connect(function(err){  
    if(err) throw err;  
  
    con.query("SELECT* FROM library",function(err,result,fields){  
        if(err) throw err;  
        console.log(result);  
    });  
});
```

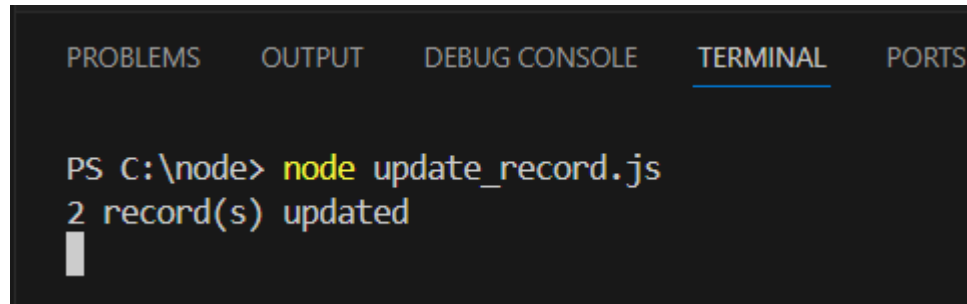
Output:

```
PS C:\node> node reading_record.js  
[  
  RowDataPacket { name: 'Mansi', location: 'Bangalore' },  
  RowDataPacket { name: 'Ram', location: 'Bangalore' },  
  RowDataPacket { name: 'Omkar', location: 'Pune' },  
  RowDataPacket { name: 'Chinmay', location: 'Mumbai' },  
  RowDataPacket { name: 'Akhila', location: 'Delhi' },  
  RowDataPacket { name: 'Yadnesh', location: 'Delhi' },  
  RowDataPacket { name: 'Anuj', location: 'Pune' },  
  RowDataPacket { name: 'Prathmesh', location: 'Mumbai' },  
  RowDataPacket { name: 'Sidhresh', location: 'Bangalore' },  
  RowDataPacket { name: 'Kapil', location: 'Pune' }  
]
```

c) Update the employee location with Mumbai whose location is Delhi.**Code:**

```
var mysql = require('mysql');  
var con = mysql.createConnection({  
    host: "localhost",  
    user: "root",  
    password:"",  
    database: "librarydb"  
});  
con.connect(function(err) {  
    if (err) throw err;  
    var sql = "UPDATE library SET location='Mumbai' WHERE location='Delhi'";
```

```
con.query(sql, function (err, result) {  
  if (err) throw err;  
  console.log(result.affectedRows + " record(s) updated");  
});  
});
```

Output:

```
PS C:\node> node update_record.js  
2 record(s) updated
```

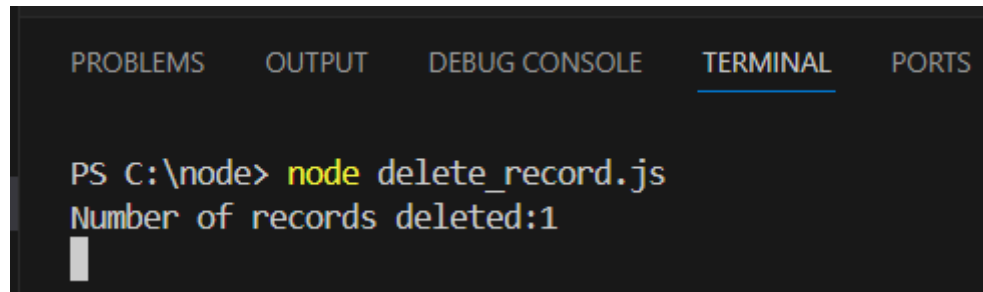
name	location
Mansi	Bangalore
Ram	Bangalore
Omkar	Pune
Chinmay	Mumbai
Akhila	Mumbai
Yadnesh	Mumbai
Anuj	Pune
Prathmesh	Mumbai
Sidhhesh	Bangalore
Kapil	Pune

☐ Show all | Nun

d) Delete**Code:**

```
var mysql = require('mysql');  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "root",  
  password: "",  
  database: "librarydb"  
});
```

```
con.connect(function(err) {  
  if (err) throw err;  
  var sql = "DELETE FROM library WHERE name = 'Kapil'";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("Number of records deleted:" + result.affectedRows);  
  });  
});
```

Output:

```
PS C:\node> node delete_record.js  
Number of records deleted:1
```

name	location
Mansi	Bangalore
Ram	Bangalore
Omkar	Pune
Chinmay	Mumbai
Akhila	Mumbai
Yadnesh	Mumbai
Anuj	Pune
Prathmesh	Mumbai
Sidhresh	Bangalore

☐ Show all | Nun

Installation to React JS

You can now create a new React application by typing:

npx create-react-app my-app

where my-app is the name of the folder for your application. This may take a few minutes to create the React application and install its dependencies.

```
PS C:\react> npx create-react-app my-app

Creating a new React app in C:\react\my-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

added 1315 packages in 2m

261 packages are looking for funding
  run `npm fund` for details
Git repo not initialized Error: Command failed: git --version
  at genericNodeError (node:internal/errors:983:15)
  at wrappedFn (node:internal/errors:537:14)
  at checkExecSyncError (node:child_process:888:11)
  at execSync (node:child_process:960:15)
  at tryGitInit (C:\react\my-app\node_modules\react-scripts\scripts\init.js:46:5)
  at module.exports (C:\react\my-app\node_modules\react-scripts\scripts\init.js:276:7)
  at [eval]:3:14
  at runScriptInThisContext (node:internal/vm:209:10)
  at node:internal/process/execution:118:14
  at [eval]-wrapper:6:24 {
  status: 1,
  signal: null,
  output: [ null, null, null ],
  pid: 13064,
  stdout: null,
  stderr: null
}
```

```
removed 1 package, and audited 1361 packages in 3s

265 packages are looking for funding
  run `npm fund` for details

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

Success! Created my-app at C:\Users\manas\OneDrive\Documents\Web Development\my-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd my-app
  npm start

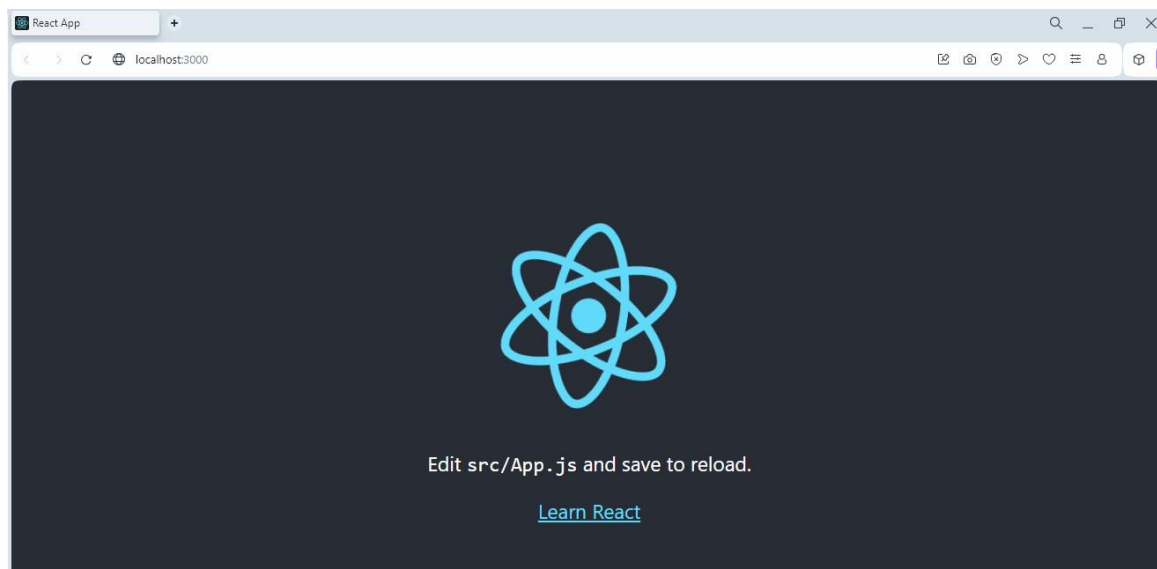
Happy hacking!
```


Let's quickly run our React application by navigating to the new folder and typing `npm start` to start the web server and open the application in a browser:

```
cd my-app  
npm start
```

```
Compiled successfully!  
  
You can now view my-app in the browser.  
  
Local:      http://localhost:3000  
On Your Network:  http://192.168.205.203:3000  
  
Note that the development build is not optimized.  
To create a production build, use npm run build.  
  
webpack compiled successfully
```

You should see the React logo and a link to "Learn React" on <http://localhost:3000> in your browser. We'll leave the web server running while we look at the application with VS Code.



PRACTICAL - 7

Expressions in JSX are snippets of JavaScript code that are enclosed within curly braces {}. They are used to dynamically render values, call functions, or compute data.

Write a program to demonstrate Expression, large block of HTML and a Conditional statement in JSX.

Code:

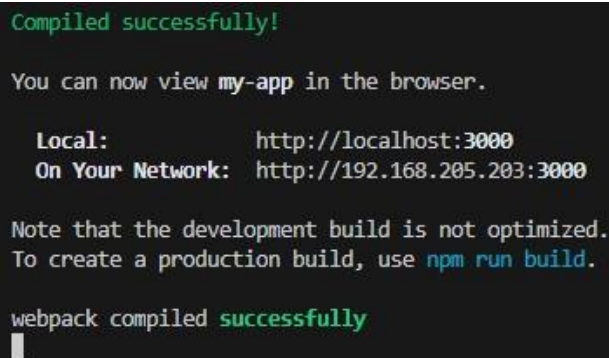
Expression:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = <h1>subtraction of 8-2 is {8-2}. This is with JSX</h1>

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Output:



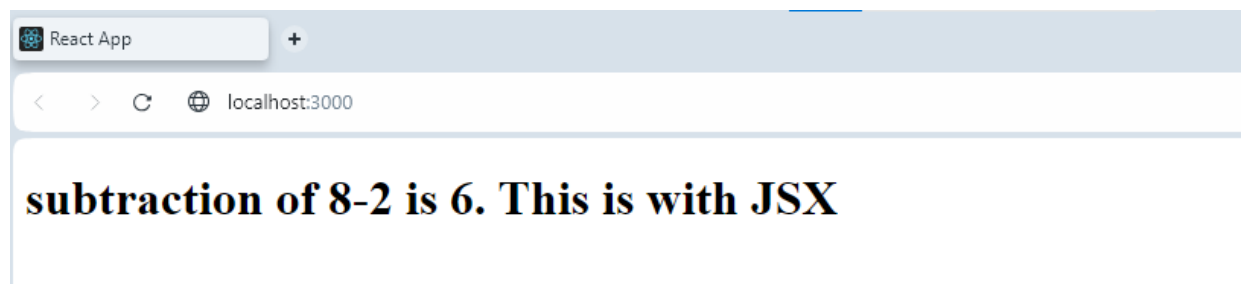
```
Compiled successfully!

You can now view my-app in the browser.

Local:      http://localhost:3000
On Your Network:  http://192.168.205.203:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



Large block of HTML

In React, large blocks of code, such as loops or complex computations, are usually handled outside JSX and their results are passed into JSX.

Code:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const myElement = (
  <div>
    <h1>Hi, My Name is Mansi</h1>
    <h1>I am currently a student at NMITD.</h1>

  </div>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Output:



Conditional Statement

Conditional rendering allows you to decide what to display based on conditions.

Code:

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';

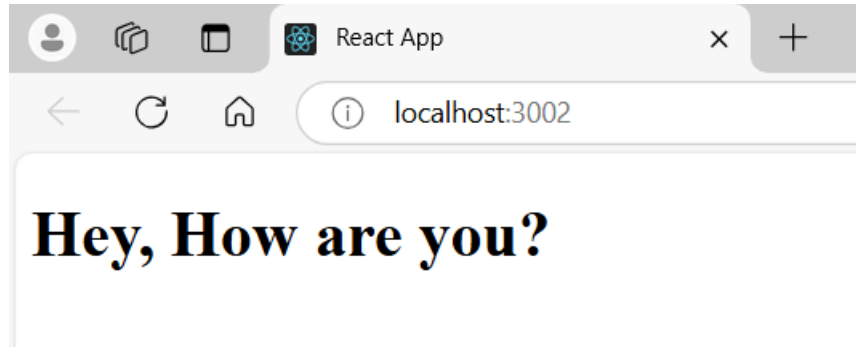
const x = 13;
let text = "Hey, How are you?";
if (x<12){
  text = "Good Morning , Have a Nice day!";
}

const myElement = <h1>{text}</h1>
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(myElement);
```

Output:

If $x > 12$ (Example. 13)

**index.js**

```
import React from 'react';  
import ReactDOM from 'react-dom/client';
```

```
const x = 4;  
let text = "Hey, How are you?";  
if (x < 12) {  
  text = "Good Morning, Have a Nice day! ";  
}
```

```
const myElement = <h1>{text}</h1>
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(myElement);
```

Output:

If $x < 12$ (Example. 4)



PRACTICAL- 8

Functional components are simple JavaScript functions that accept props as an argument and return JSX. They are widely used in modern React development.

a) Create a react JS application which demonstrates Functional Components.

Code:

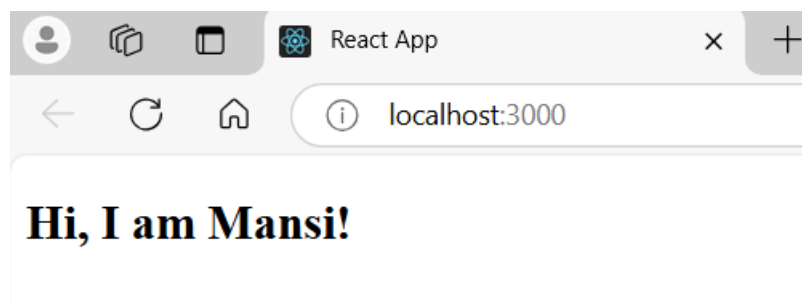
index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Intro(){
  return <h2>Hi, I am Mansi!</h2>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Intro/>);
```

Output:



Functional Component using props.

Props are like function arguments, and you send them into the component as attributes.

Functional component using props in React is a simple JavaScript function that accepts an argument called props (short for properties). This argument is an object containing data passed from a parent component to the child component.

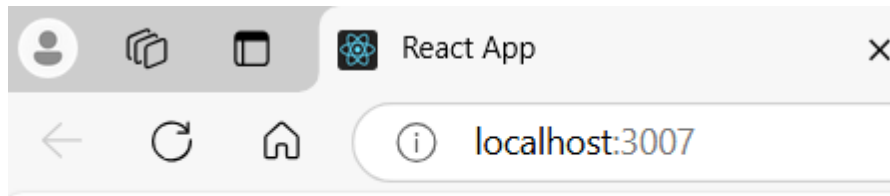
Code:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function College(props){
  return<h3>Hi Myself Mansi!!I am from {props.location} </h3>;
}

const root=ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(<College location="Borivali"/>);
```

Output:

Hi Myself Mansi!!I am from Borivali

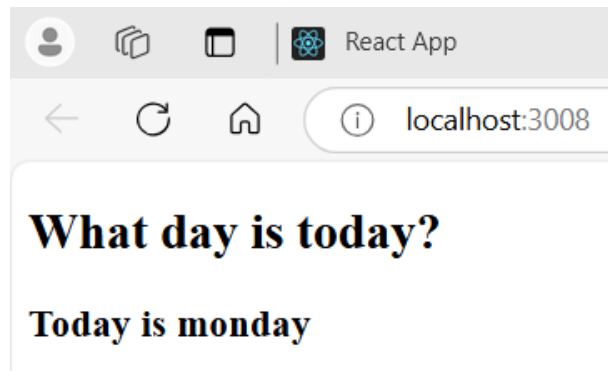
Component in component

A component within a component (also called nested components) in React allows one component to use another component inside it. This practice helps in organizing code and reusing components efficiently.

Code:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
function College(props){
  return<h3>Today is monday</h3>;
}
function Management(){
  return(
    <>
    <h2>
    What day is today?
    </h2>
    <College/>
    </>
  );
}
const root=ReactDOM.createRoot(document.getElementById('root'));

root.render(<Management/>);
```

Output:**b) Create a ReactJS application which demonstrates Class Component.**

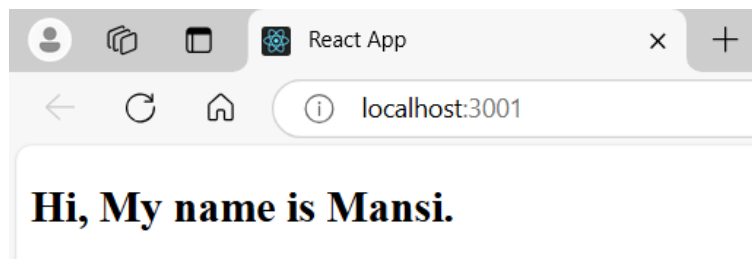
A class component in React is a JavaScript ES6 class that extends `React.Component` and is used to define reusable, stateful components in React applications. It includes a `render()` method that returns JSX.

Code:**index.js**

```
import React from 'react';
import ReactDOM from 'react-dom/client';

class Intro extends React.Component{
  render(){
    return <h2>Hi, My name is Mansi.</h2>;
  }
}

const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(<Intro/>);
```

Output:

Component Constructor

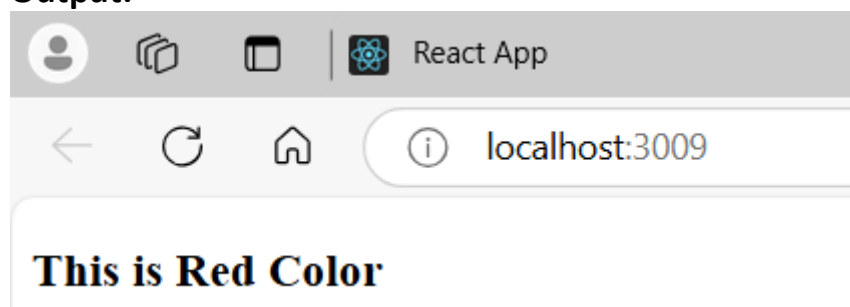
The constructor in a React component is called before the component is mounted. When you implement the constructor for a React component, you need to call `super(props)` method before any other statement.

If you do not call `super(props)` method, `this.props` will be undefined in the constructor and can lead to bugs.

Code:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class College extends React.Component{
  constructor(){
    super();
    this.state={Name:"Red"};
  }
  render(){
    return<h3>This is {this.state.Name} Color</h3>
  }
}
const container=document.getElementById("root");
const root=ReactDOM.createRoot(container);
root.render(<College/>)
```

Output:



Props with Constructor

Props are like function arguments, and you send them into the component as attributes.

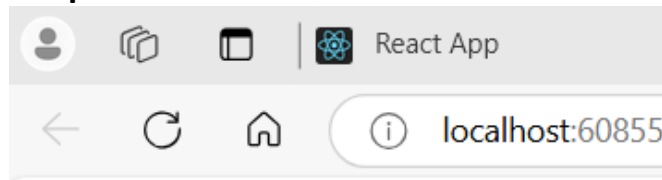
If your component has a constructor function, the props should always be passed to the constructor and also to the `React.Component` via the `super()` method.

Code:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class College extends React.Component{
  constructor(props){
    super(props);
```



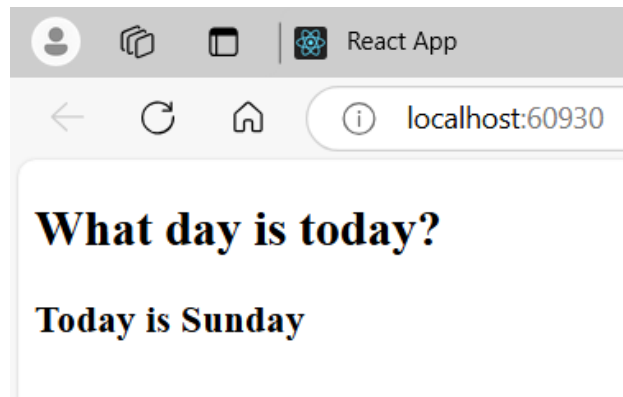
```
}  
  render(){  
    return<h3>Today is {this.props.Name} </h3>  
  }  
}  
  
//const container=document.getElementById("root");  
const root=ReactDOM.createRoot(document.getElementById("root"));  
root.render(<College Name="Sunday"/>)
```

Output:

Today is Sunday

Component in component**Code:**

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
class College extends React.Component{  
  
  render(){  
    return<h3>Today is Sunday</h3>  
  }  
}  
  
class Management extends React.Component{  
  render(){  
    return(  
      <div><h2>What day is today?</h2>  
      <College/></div>);  
  }  
}  
  
//const container=document.getElementById("root");  
const root=ReactDOM.createRoot(document.getElementById("root"));  
root.render(<Management/>)
```

Output:**c) Create a react JS application to implement Component Life Cycle**

React class components provide a way to implement the Component Lifecycle through lifecycle methods. These methods allow developers to hook into specific stages of a component's lifecycle, such as when it's mounted, updated, or unmounted

Code:**index.js**

```
import React from "react";
import ReactDOM from "react-dom/client";

class Test extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hello: "Node.js!" };
  }

  componentDidMount() {
    console.log("componentDidMount()");
  }

  changeState() {
    this.setState({ hello: "React.js!" });
  }

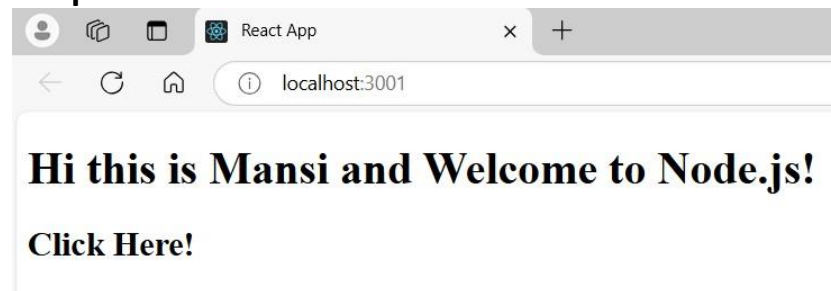
  render() {
    return (
      <div>
        <h1>Hi this is Mansi and Welcome to {this.state.hello}</h1>
        <h2>
          <a onClick={this.changeState.bind(
            this
          )}>
            >
              Click Here!
            </a>
          </h2>
        </div>
      </div>
    );
  }
}
```

```
        </a>
      </h2>
    </div>
  );
}

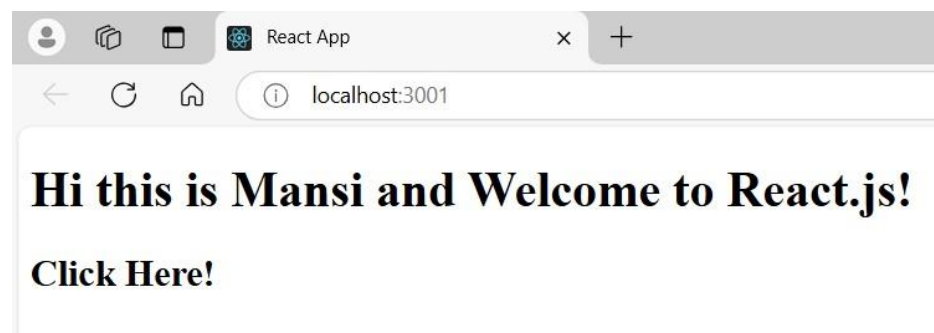
shouldComponentUpdate(nextProps, nextState) {
  console.log("shouldComponentUpdate()");
  return true;
}

componentDidUpdate() {
  console.log("componentDidUpdate()");
}
}

const root = ReactDOM.createRoot(
  document.getElementById("root")
);
root.render(<Test />);
```

Output:

After clicking on Click Here!



d) Create an Application in Reactjs to implement rendering

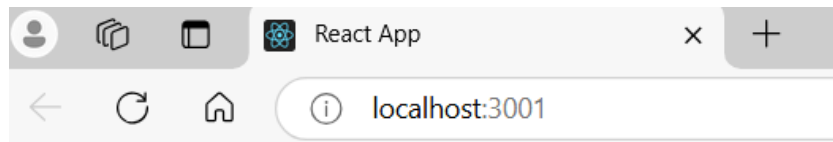
The render() method is required in class components and is used to return the JSX that will be rendered to the DOM.

Code:**index.js**

```
import React from 'react';
import ReactDOM from 'react-dom/client';

class Intro extends React.Component{
  render(){
    return <h2>Hi, I am Mansi and Welcome to React.</h2>;
  }
}

const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(<Intro/>);
```

Output:

Hi, I am Mansi and Welcome to React.

PRACTICAL – 9

Create an application in ReactJS to import and export files.

React is all about re-using code, and it can be smart to insert some of your components in separate files.

To do that, create a new file with a .js file extension and put the code inside it

Note that the file must start by importing React (as before), and it has to end with the statement `export default College;`

Here a file with `College.js` is created.

For functional Component

Code:

intro.js

```
function Intro (){  
  return <h2>Hi, My name is Mansi!</h2>;  
}
```

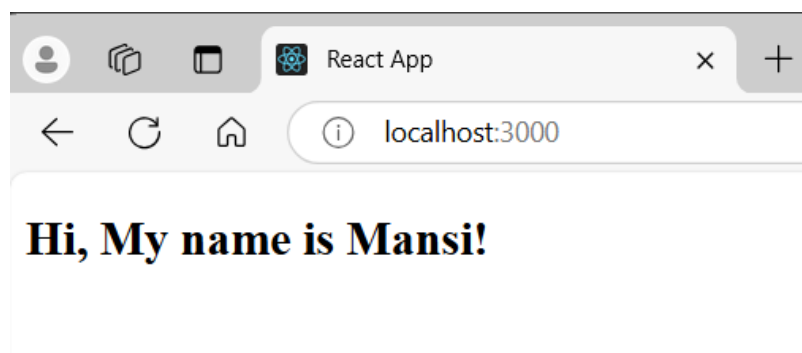
```
export default Intro;
```

index.js

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
import Intro from './intro.js';
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(<Intro/>);
```

Output:

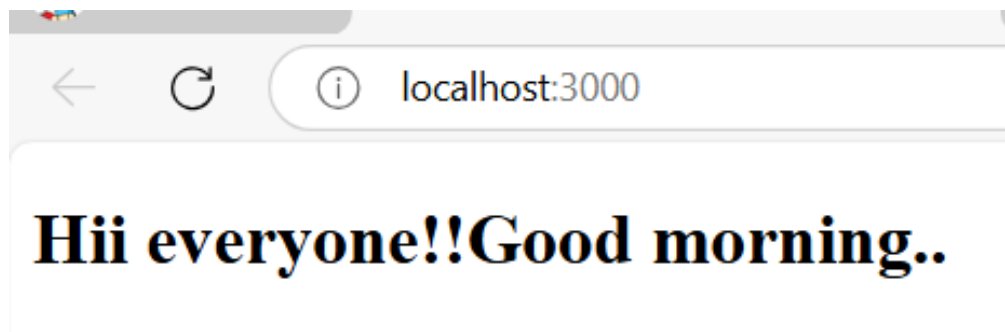


For Class Component**Code:****College.js**

```
import React from 'react';  
class College extends React.Component {  
  render() {  
    return <h2>Hii everyone!!Good morning..</h2>  
  }  
}  
export default College;
```

Index.js

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
import College from './College.js';  
const container = document.getElementById("root");  
const root = ReactDOM.createRoot(container);  
root.render(<College/>)
```

Output:

Practical 10

Create an application to implement

1.State

State is a local data storage in a React component that can only be updated within that component. It allows components to react dynamically to user input or other changes

Code:

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

```
class Counter extends Component {  
  constructor(props) {  
    super(props);  
    this.state = { count: 0 };  
  }  

```

```
  increment = () => {  
    this.setState({ count: this.state.count + 1 });  
  };  

```

```
  render() {  
    return (  
      <div style={{ textAlign: "center", marginTop: "50px" }}>  
        <p>Count: {this.state.count}</p>  
        <button  
          onClick={this.increment}  
  
          >  
            Increment  
          </button>  
      </div>  
    );  
  }  
}
```

```
function App() {  
  return (  
    <div>  
      <h1 style={{ textAlign: "center" }}>Counter Example</h1>  
      <Counter />  
    </div>  
  );  
}
```

export default App;

Output:

Counter Example

Count: 2

Increment

2.Props

React Props are like function arguments in JavaScript *and* attributes in HTML. To send props into a component, use the same syntax as HTML attributes.

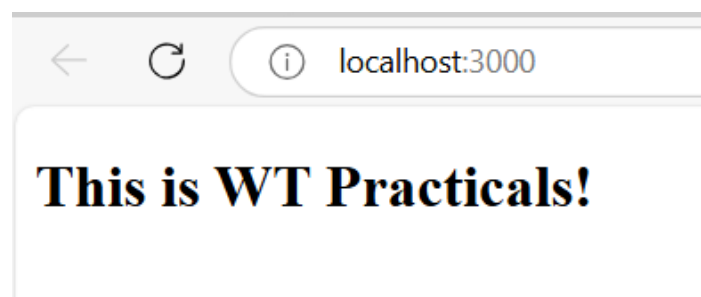
Code:

```
import React from 'react';  
import ReactDOM from 'react-dom/client';
```

```
function College(props) {  
  return <h2>This is { props.Name }!</h2>;  
}
```

```
const myElement = <College Name="WT Practicals" />;  
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(myElement);
```

Output:



Practical 11

Create an application in reactJs to implement DOM events

An event is an action that could be triggered as a result of the user action or system generated event.

React events are written in camelCase syntax:

onClick instead of onclick. React event handlers are written inside curly braces.

Click:

Description: Triggered when a user clicks an element, such as a button.

Usage: The onClick event is attached to the element.

Code:

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

function App() {
  const [inputValue, setInputValue] = useState("");
  const [clickMessage, setClickMessage] = useState("");

  // Handle input change
  const handleInputChange = (event) => {
    setInputValue(event.target.value);
  };

  // Handle button click
  const handleButtonClick = () => {
    setClickMessage(`Hello , ${inputValue}!`);
  };

  return (
    <div style={{ padding: '20px', fontFamily: 'Arial' }}>
      <h1>React Event Handling Example</h1>
```

```
<div>

  <label htmlFor="nameInput">Enter your name: </label>

  <input
    id="nameInput"
    type="text"
    value={inputValue}
    onChange={handleInputChange}
    placeholder="Type here..."
    style={{ marginRight: '10px', padding: '5px' }}
  />

  <button onClick={handleButtonClick} style={{ padding: '5px 10px' }}>
    Click Me
  </button>
</div>

{clickMessage && <p style={{ marginTop: '20px', color: 'green' }}>{clickMessage}</p>}
</div>

);
}
```

export default App;

Output:

React Event Handling Example

Enter your name:

Hello , Mansi!

Change

The onChange event in React is used to handle changes in the value of input elements like text fields, checkboxes, radio buttons, and select boxes. It is commonly used to update the component's state when a user interacts with a form.

Code:

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

function App() {

  // Define a state variable to store the input value
  const [inputValue, setInputValue] = useState("");

  // Event handler for input change
  const handleInputChange = (event) => {
    setInputValue(event.target.value); // Update the state with the input value
  };

  return (
    <div style={{ padding: '20px', fontFamily: 'Arial' }}>
      <h1>React onChange Event Example</h1>

      <div>
        <label htmlFor="nameInput">Enter your name: </label>
        <input
          id="nameInput"
          type="text"
          value={inputValue} // Bind the input value to state
          onChange={handleInputChange} // Attach the change handler
          placeholder="Type your name"
          style={{ padding: '5px' }}
        />
```

```
</div>

{/* Display the input value below the input field */}
{inputValue && (
  <p style={{ marginTop: '20px', color: 'blue' }}>
    You typed: {inputValue}
  </p>
)}
</div>

);
}

export default App;
```

Output:**React onChange Event Example**

Enter your name:

You typed: Mansi

Blur

The onBlur event in React is triggered when an input field loses focus. It is often used for validation or to trigger actions when a user finishes interacting with an input field.

Code:

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

function App() {
  // State to track if the input field is blurred
  const [isBlurred, setIsBlurred] = useState(false);
  const [inputValue, setInputValue] = useState("");
```

```
// Handle the blur event
const handleBlur = () => {
  setIsBlurred(true);
};

// Handle input change
const handleInputChange = (event) => {
  setInputValue(event.target.value); // Update the input value in state
};

return (
  <div style={{ padding: '20px', fontFamily: 'Arial' }}>
    <h1>React Blur Event Example</h1>

    <div>
      <label htmlFor="nameInput">Enter your name: </label>
      <input
        id="nameInput"
        type="text"
        value={inputValue}
        onChange={handleInputChange} // Handle input changes
        onBlur={handleBlur} // Handle the blur event
        placeholder="Type your name"
        style={{ padding: '5px' }}
      />
    </div>

    {}
  </div>
);
```

```
{isBlurred && !inputValue && (  
  <p style={{ color: 'red' }}>Input field lost focus, but no value entered!</p>  
)}  
  
{/* Show entered input value */}  
{isBlurred && inputValue && (  
  <p style={{ color: 'green' }}>You entered: {inputValue}</p>  
)}  
</div>  
);  
}
```

export default App;

Output:

React Blur Event Example

Enter your name:

You entered: Mansi

Keypress and Keyup

onKeyPress: Triggered when a key is pressed down. However, onKeyPress is deprecated in newer versions of React and browsers for most use cases (except for characters).

onKeyUp: Triggered when the key is released.

Code:

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

```
function App() {  
  // State to store the current input value  
  const [inputValue, setInputValue] = useState("");  
  const [lastKeyPressed, setLastKeyPressed] = useState("");  
  
  // Handle key press event  
  const handleKeyPress = (event) => {  
    // Update state with the character pressed by the user  
    setLastKeyPressed(event.key);  
    setInputValue(event.target.value); // Update the input value state  
  };  
  
  return (  
    <div style={{ padding: '20px', fontFamily: 'Arial' }}>  
      <h1>React KeyPress Event Example</h1>  
  
      <div>  
        <label htmlFor="nameInput">Type something: </label>  
        <input  
          id="nameInput"  
          type="text"  
          value={inputValue}  
          onKeyPress={handleKeyPress} // Attach the onKeyPress event handler  
          placeholder="Start typing..."  
          style={{ padding: '5px' }}  
        />  
      </div>  
  
      { /* Display the key pressed and the input value */ }
```

```
<p style={{ marginTop: '20px', color: 'blue' }}>
  Last Key Pressed: {lastKeyPressed}
</p>

</div>

);
}

export default App;
```

Output:

React KeyPress Event Example

Type something:

Last Key Pressed: a

Practical 12

Create an application in ReactJs form and add client and server side validation

Validation in React is an important part of building forms and ensuring that the data entered by users is correct. You can perform validation in various ways, such as checking if fields are empty, matching patterns (e.g., email), or even ensuring that the values fall within specific ranges. React provides ways to handle validation using both controlled and uncontrolled components.

Code:

```
import React, { useState } from "react";
```

```
function FormValidation() {
```

```
  const [formData, setFormData] = useState({
```

```
    name: "",
```

```
    email: "",
```

```
    password: "",
```

```
    confirmPassword: "",
```

```
  });
```

```
  const [errors, setErrors] = useState({
```

```
    name: "",
```

```
    email: "",
```

```
    password: "",
```

```
    confirmPassword: "",
```

```
  });
```

```
  const handleChange = (event) => {
```

```
    const { name, value } = event.target;
```

```
    setFormData({
```

```
      ...formData,
```

```
      [name]: value,
```

```
});  
};  
  
const validateForm = () => {  
  let valid = true;  
  let errors = {};  
  
  // Validate name  
  if (!formData.name) {  
    errors.name = "Name is required!";  
    valid = false;  
  }  
  
  // Validate email with a basic regex  
  const emailPattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;  
  if (!formData.email) {  
    errors.email = "Email is required!";  
    valid = false;  
  } else if (!emailPattern.test(formData.email)) {  
    errors.email = "Please enter a valid email address.";  
    valid = false;  
  }  
  
  // Validate password  
  if (!formData.password) {  
    errors.password = "Password is required!";  
    valid = false;  
  } else if (formData.password.length < 6) {  
    errors.password = "Password must be at least 6 characters long.";
```

```
    valid = false;
  }
  setErrors(errors);
  return valid;
};

const handleSubmit = (event) => {
  event.preventDefault();
  if (validateForm()) {
    console.log("Form submitted successfully!", formData);
  } else {
    console.log("Form validation failed.");
  }
};

return (
  <div style={{ textAlign: "center", marginTop: "50px" }}>
    <h1>Form Validation</h1>
    <form onSubmit={handleSubmit}>
      { /* Name Input */ }
      <div style={{ marginBottom: "10px" }}>
        <label>Name:</label>
        <input
          type="text"
          name="name"
          value={formData.name}
          onChange={handleChange}
          style={{
            padding: "8px",
```

```
        fontSize: "16px",
        border: "1px solid #ccc",
        borderRadius: "5px",
    }}
/>
{errors.name && <p style={{ color: "red" }}>{errors.name}</p>}
</div>
```

```
{/* Email Input */}
<div style={{ marginBottom: "10px" }}>
    <label>Email:</label>
    <input
        type="email"
        name="email"
        value={formData.email}
        onChange={handleChange}
        style={{
            padding: "8px",
            fontSize: "16px",
            border: "1px solid #ccc",
            borderRadius: "5px",
        }}
    />
    {errors.email && <p style={{ color: "red" }}>{errors.email}</p>}
</div>
```

```
{/* Password Input */}
<div style={{ marginBottom: "10px" }}>
    <label>Password:</label>
```

```
<input
  type="password"
  name="password"
  value={formData.password}
  onChange={handleChange}
  style={{
    padding: "8px",
    fontSize: "16px",
    border: "1px solid #ccc",
    borderRadius: "5px",
  }}
/>

{errors.password && <p style={{ color: "red" }}>{errors.password}</p>}

</div>

{/* Submit Button */}

<button
  type="submit"
  style={{
    padding: "10px 20px",
    fontSize: "16px",
    backgroundColor: "#4CAF50",
    color: "white",
    border: "none",
    borderRadius: "5px",
  }}
>
  Submit
</button>

</form>
```

```
</div>  
);  
}  
export default FormValidation;
```

Output:

Form Validation

Name:

Email:

Email is required!

Password:

Password must be at least 6 characters long.

Submit

Practical 13

Create an application to implement three types of Hooks

Hooks allow function components to have access to state and other React features.

useState

The React useState Hook allows us to track state in a function component.

State generally refers to data or properties that need to be tracking in an application.

Code:

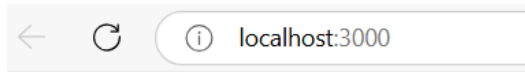
```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function College() {
  const [college, setCollege] = useState({
    management: "DES",
    name: "NMITD",
    year: "2008",
    location: "Dadar"
  });

  return (
    <>
      <h1>My {college.management}</h1>
      <p>
        It is a {college.name} at {college.location} from {college.year}.
      </p>
    </>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(<College />);
```

Output:

My DES

It is a NMITD at Dadar from 2008.

Context

React Context is a way to manage state globally.

It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

Code:

```
import { useState } from "react";
```

```
import ReactDOM from "react-dom/client";
```

```
function College() {
```

```
  const [college, setCollege] = useState({
```

```
    management: "DES",
```

```
    name: "NMITD",
```

```
    year: "2008",
```

```
    location: "Dadar"
```

```
  });
```

```
  return (
```

```
    <>
```

```
    <h1>My {college.management}</h1>
```

```
    <p>
```

```
      It is a {college.name} at {college.location} from {college.year}.
```

```
    </p>
```



```
</>
)
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<College />);
```

Output:

My DES

It is a NMITD at Dadar from 2008.

Effect

The useEffect Hook allows you to perform side effects in your components.

Code:

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom/client";

function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });

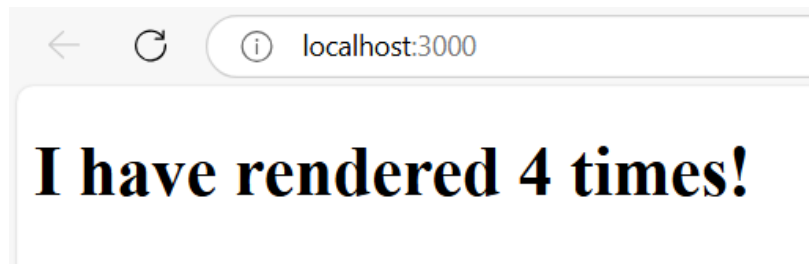
  return <h1>I have rendered {count} times!</h1>;
```

```
}
```

```
// The setTimeout() method calls a function after a number of milliseconds. //1 second = 1000 milliseconds.
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(<Timer />);
```

Output:

Practical 14

Create SPA using React Router

React Router is a JavaScript library that helps developers create single-page applications (SPAs) with dynamic routing

Routing is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.

Steps

1. Add React Router

To add React Router in your application, run this in the terminal from the root directory of the application:

```
npm i -D react-router-dom
```

If you are upgrading from v5, you will need to use the @latest flag:

```
npm i -D react-router-dom@latest
```

2. In our project, we will create two more components along with **App.js**, which is already present.

About.js

```
import React from 'react'

class About extends React.Component {

  render() {

    return <h1>About</h1>

  }

}

export default About
```

Contact.js

```
import React from 'react'

class Contact extends React.Component {

  render() {

    return <h1>Contact</h1>
```

```
}  
  
}  
  
export default Contact
```

App.js

```
import React from 'react'  
  
class App extends React.Component {  
  
  render() {  
  
    return (  
  
      <div>  
  
        <h1>Home</h1>  
  
      </div>  
  
    )  
  
  }  
  
}  
  
export default App
```

Index.js

```
import React from 'react';  
  
import ReactDOM from 'react-dom';  
  
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'  
  
import './index.css';  
  
import App from './App';  
  
import About from './about'  
  
import Contact from './contact'  
  
  
const routing = (  
  
  <Router>  
  
    <div>
```

```
<h1>React Router Example</h1>

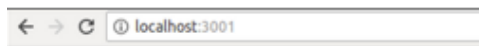
<Route exact path="/" component={App} />
<Route path="/About" component={About} />
<Route path="/Contact" component={Contact} />

</div>

</Router>

)

ReactDOM.render(routing, document.getElementById('root'));
```

Output:**React Router Example****Home****React Router Example****Home****About**