6.a Course Name: Node.js Module Name: How to use Node.js Verify how to execute different functions successfully in the Node.js platform.

Aim: Learning about use of Node.js and verifying how to execute different functions successfully in the Node.js platform.

Description:

Node.js is an open source server environment.Node.js allows you to run JavaScript on the server. Create a JavaScript file. Execute the JavaScript file using node.js. Let us create our first Node.js program.

Step 1: Create a folder NodeJS in C drive and create a new JavaScript file, 1.js inside the folder. Type the below code inside the JavaScript file.

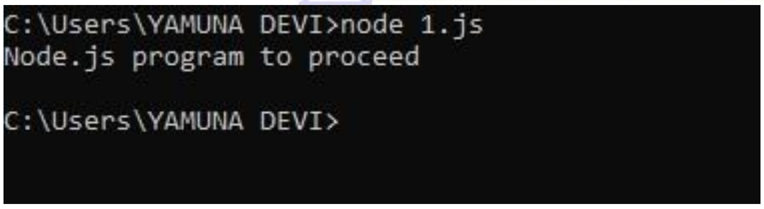
On notepad:

console.log("Node.js program to proceed");

Step 2: Navigate to the created NodeJS folder in the NodeJS command prompt and execute the JavaScript file, 1.js using the node command.

node 1.js

Step 3: After the successful interpretation of the code, we can see the output in the Node.js command prompt as shown below

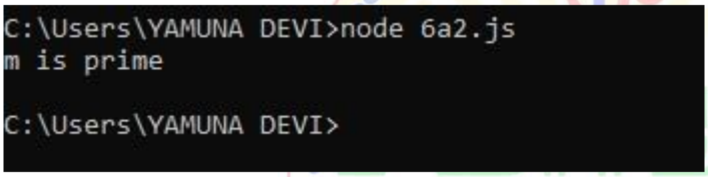


```
C:\Users\YAMUNA DEVI>node 1.js
Node.js program to proceed

C:\Users\YAMUNA DEVI>
```

Program-2:

```
function tester()
{
var m=10;
var message;
if (m%2==0)
{
message = "m is prime";
}
else
{
message = "m is not prime";
}
console.log(message);
}
tester();
```

OUTPUT:



```
C:\Users\YAMUNA DEVI>node 6a2.js
m is prime

C:\Users\YAMUNA DEVI>
```

6.b Course Name: Node.js Module Name: Create a web server in Node.js Write a program to show the workflow of JavaScript code executable by creating web server in Node.js.

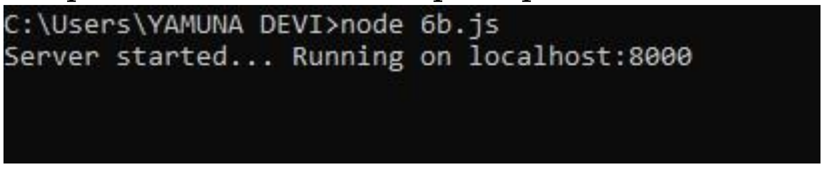
Aim: Creating a web server in Node.js and showing the workflow of JavaScript code executable by creating web server in Node.js.

Description:
Using require() and createServer() method Running a web server in Node.js
Step 1: Create a new JavaScript file httpserver.js and include the HTTP module.
Step 2: Use the createServer() method of the HTTP module to create a web server.
Step 3: Save the file and start the server using the node command. When the file executes successfully, we can observe the following output in the console.
Step 4: We will observe the output in the browser.

Program

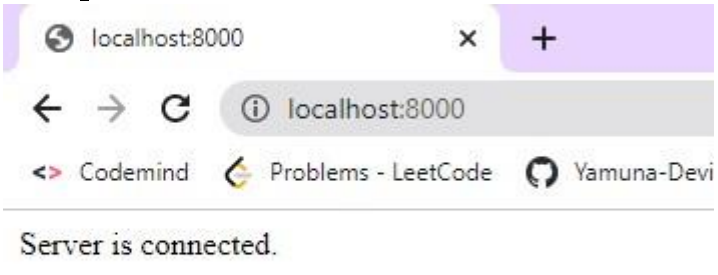
```
var http = require('http');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/html'});
res.end(‘Server is connected.');
}).listen(8000);
console.log("Server started... Running on localhost:8000");
```

Output:
Output on the command prompt –



```
C:\Users\YAMUNA DEVI>node 6b.js
Server started... Running on localhost:8000
```

Output on the browser:



Server is connected.

6.c Course Name: Node.js Module Name: Modular programming in Node.js Write a Node.js module to show the workflow of Modularization of Node application.

Aim: Write a Node.js module to show the workflow of Modularization of Node application

Description: Modularization is a software design technique in which the functionality of a program is separated into independent modules, such that each module contains the desired functionality.

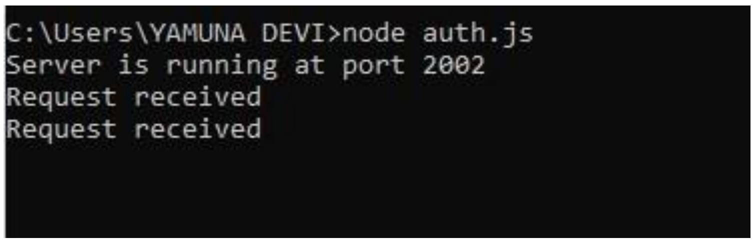
Readability: Modular code highly organizes the program based on its functionality. This allows the developers to understand what each piece of code does in the application.
Easier to debug: When debugging large programs, it is difficult to detect bugs. If a program is modular, then each module is discrete, so each module can be debugged easily by the programmer.
Reusable Code: Modular code allows programmers to easily reuse code to implement the same functionality in a different program. If the code is not organized modularly into discrete parts, then code reusability is not possible.
Reliability: Modular code will be easier to read. Hence it will be easier to debug and maintain the code which ensures smoother execution with minimum errors.

Program:
module.js
```
exports.authenticateUser = (a, b) => {
return a+b;
 };
```

Auth.js
```
const http = require("http");
var dbmodule = require("./module");
var server = http.createServer((request, response) => {
result = dbmodule.authenticateUser(2000,2);
response.writeHead(200, { "Content-Type": "text/html" });
response.end("<html><body><h1>" + result + "- You have connected to the localhost 2002 </h1></body></html>");
console.log("Request received");
});
```

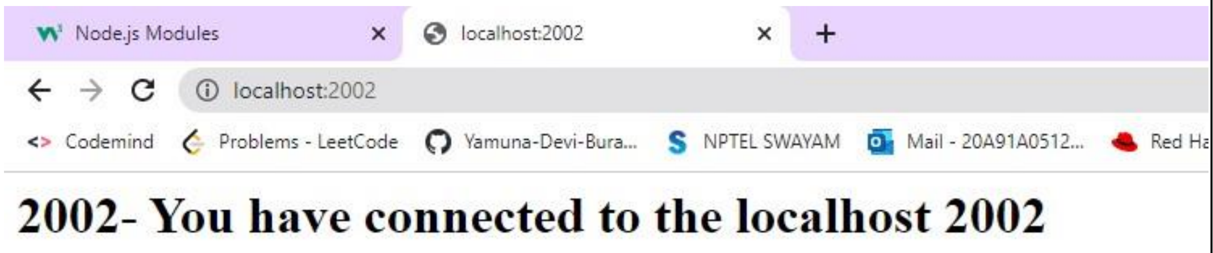
```
server.listen(2002);
console.log("Server is running at port 2002")
```

Output on the command prompt



```
C:\Users\YAMUNA DEVI>node auth.js
Server is running at port 2002
Request received
Request received
```

Output in the browser



2002- You have connected to the localhost 2002

6.d Course Name: Node.js Module Name: Restarting Node Application Write a program to show the workflow of restarting a Node application.

Aim: Program to show the workflow of restarting a node application.

Description:
Whenever we are working on a Node.js application and we do any change in code after the application is started, we will be required to restart the Node process for changes to reflect. In order to restart the server and to watch for any code changes automatically, we can use the Nodemon tool.

Nodemon

Nodemon is a command-line utility that can be executed from the terminal. It provides a different way to start a Node.js application. It watches the application and whenever any change is detected, it restarts the application. It is very easy to get started with this tool. To install it in the application, run the below command.

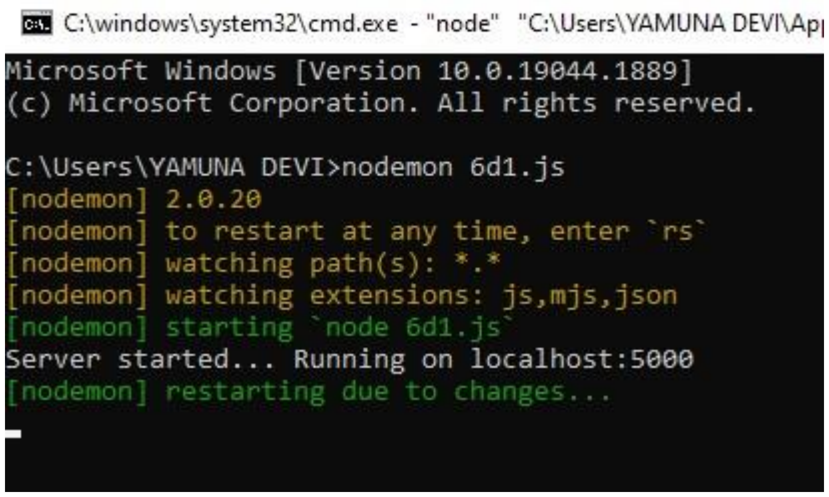
```
npm install
nodemon –g
```

Once the 'nodemon' is installed in the machine, the Node.js server code can be executed by replacing the command "node" with "nodemon".

Program:

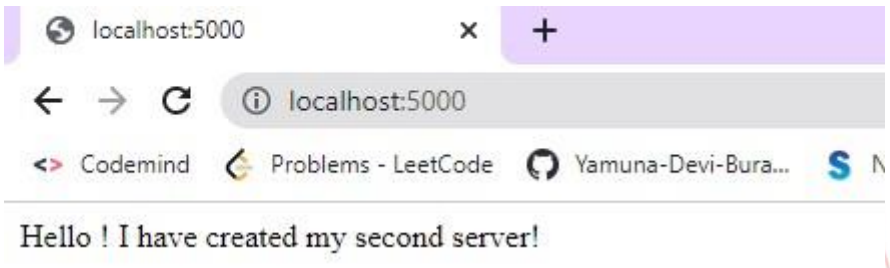
```
const http = require("http");
var server = http.createServer((req, res) => {
res.write("Hello ! I have created my second server!");
res.end();
});
server.listen(5000);
console.log("Server started... Running on localhost:5000");
```

Output in command prompt:



```
C:\windows\system32\cmd.exe - "node"  "C:\Users\YAMUNA DEVI\Ap
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\YAMUNA DEVI>nodemon 6d1.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node 6d1.js`
Server started... Running on localhost:5000
[nodemon] restarting due to changes...
```

Output in the browser:

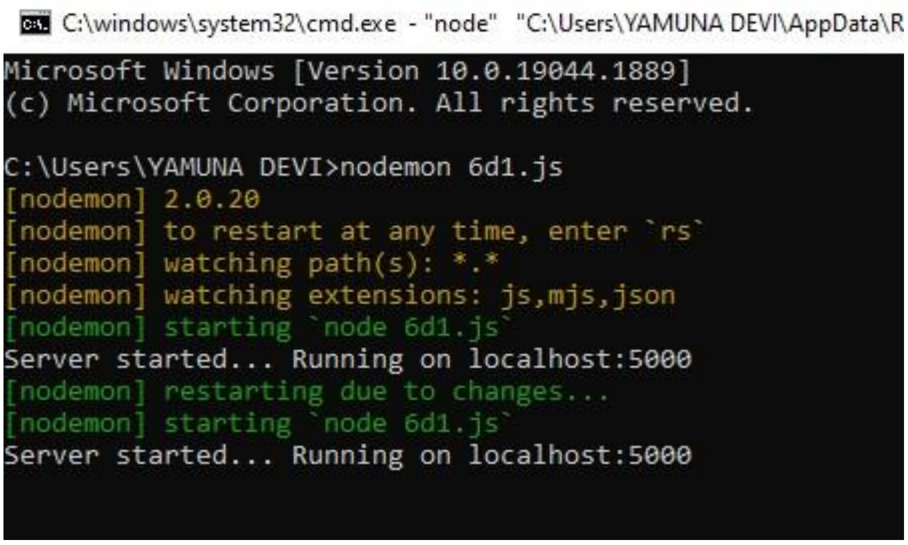


Hello ! I have created my second server!

Modified code in the file nodemon1.js

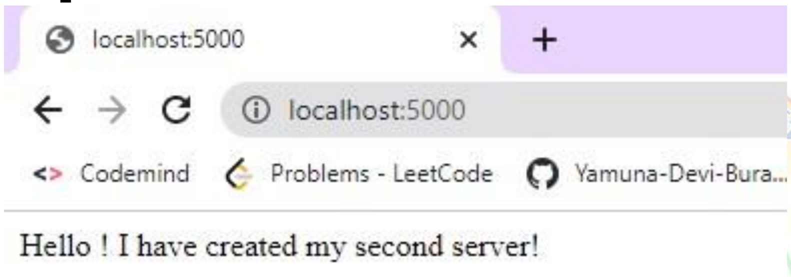
```
const http = require("http");
var server = http.createServer((req, res) => {
res.writeHead(200,{'Content-Type': 'text/html'});
res.write("Hello ! I have created my second server!");
res.end(); });
server.listen(5000);
console.log("Server started... Running on localhost:5000");
```

Output in command prompt:



```
C:\windows\system32\cmd.exe - "node"  "C:\Users\YAMUNA DEVI\AppData\R
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\YAMUNA DEVI>nodemon 6d1.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node 6d1.js`
Server started... Running on localhost:5000
[nodemon] restarting due to changes...
[nodemon] starting `node 6d1.js`
Server started... Running on localhost:5000
```

Output in the browser



Hello ! I have created my second server!

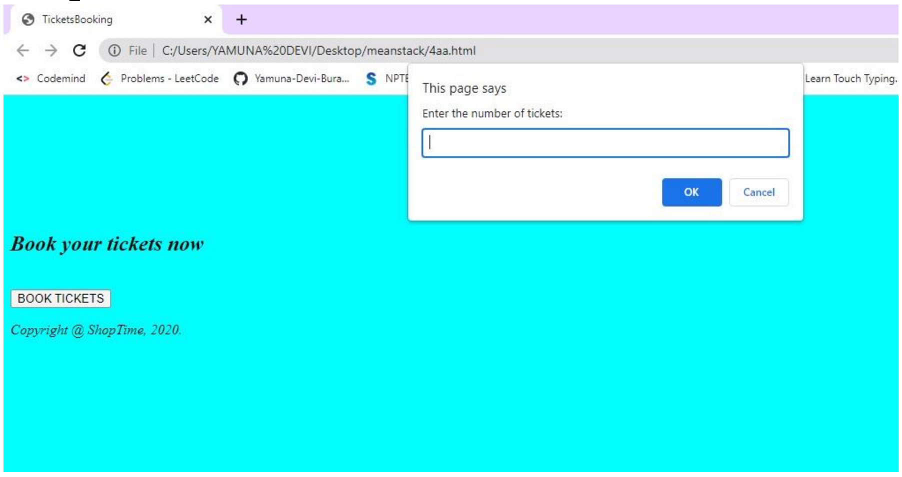
4.a Course Name: Javascript Module Name: Types of Functions, Declaring and Invoking Function, Arrow Function, Function Parameters, Nested Function, Built-in Functions, Variable Scope in Functions Write a JavaScript code to book movie tickets online and calculate the total price based on the 3 conditions:
(a) If seats to be booked are not more than 2, the cost per ticket remains Rs. 150.
(b) If seats are 6 or more, booking is not allowed.
(c) If seats to be booked are more than 2 but less than 6, based on the number of seats booked, do the following - Calculate total cost by applying a discount of 3, 5, 7, 9, 11 percent, and so on for customers till 5 respectively. Try the code with different values for the number of seats.
Write the following custom functions to implement given requirements:

i. calculateCost(seats): Calculate and display the total cost to be paid by the customer for the tickets they have bought.
ii. calculateDiscount(seats): Calculate discount on the tickets bought by the customer. Implement using arrow functions.

Aim: To write a Javascript code to book movie tickets online and calculate the total price based on the given 3 conditions using functions.

Description: A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses (). Function names can contain letters, digits, underscores, and dollar signs (same rules as variables). The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)
Syntax:
```
function name(parameter1, parameter2, parameter3)
 {
   // code to be executed
 }
```
Program:
```
<html>
<head>
<title>TicketsBooking</title>
        <script>
var x;
var y;
var z;
fun=()=>
     {
           var a=prompt("Enter the number of tickets:");
if(a<6)
{
document.getElementById("id").innerHTML="Total amount you need to
pay:";
document.getElementById("id1").innerHTML="Rs."+calculateCost(a);
 document.getElementById("id2").innerHTML="Discount
Amount is: Rs.”+calculateDiscount(a);
}
else
        {
           document.getElementById("id").innerHTML="Sorry! You can
book upto 5 tickets only in online!!";
           document.getElementById("id1").innerHTML="";
document.getElementById("id2").innerHTML="";
        }
}
const p=150;
calculateCost=(a)=>{
var i=1;
s=0;
j=0;
k=0.03
if(a>2 && a<6)
{
        do
        {
                j=p-(p*k);
                s+=j;
                j=0;
                k+=0.02;
                i+=1
        }
        while(i<=a);
}
else if(a<=2)
{
                s=p*a;
}
else
        s=0;
return s;
}
calculateDiscount=(a)=>
{
        var g=calculateCost(a);
        var z=a*p;
        return z-g;
}
</script>
    </head>
      <body bgcolor="cyan">
        <center><h1><i>ShopTime</i></h1></center>
        <h2 align="center"><i>One stop for all your needs<i></h2>
       <header>
             <nav align="center"><h3>
                     Home || Login || Register || Wishlist || My
Orders || Help</h3>
             </nav>
             <center>
        </header>
      <h2>Book your tickets now</h2>
        <br>
        <input type="button" value="BOOK TICKETS"
onclick="fun()">
<p id="id"> </p>
   <p id="id1"> </p>
    <p id="id2"> </p>
       </body>
<footer>
    Copyright @ ShopTime, 2020.
</footer>
</html>
```

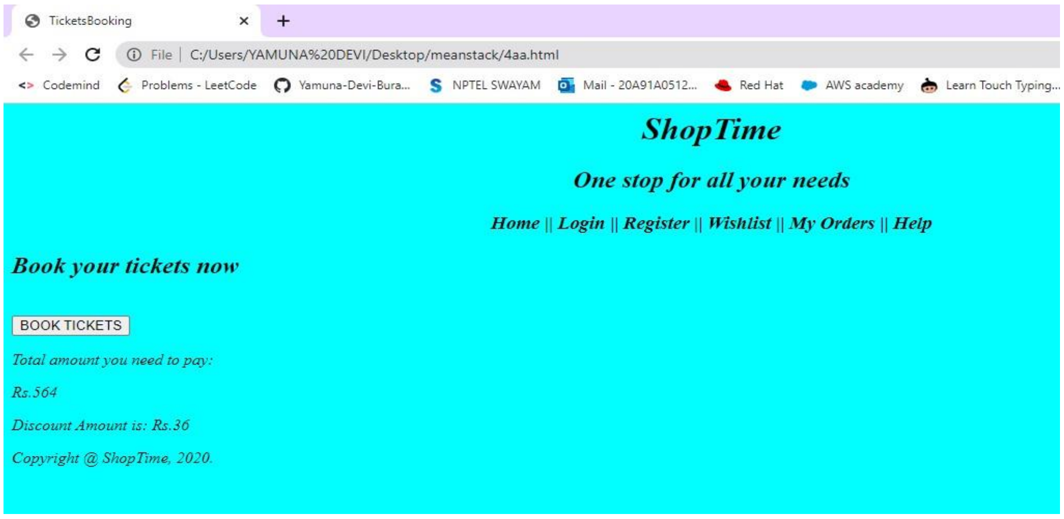
Output:


This page says
Enter the number of tickets:

Book your tickets now

BOOK TICKETS

Copyright @ ShopTime, 2020.


ShopTime

One stop for all your needs

Home || Login || Register || Wishlist || My Orders || Help

Book your tickets now

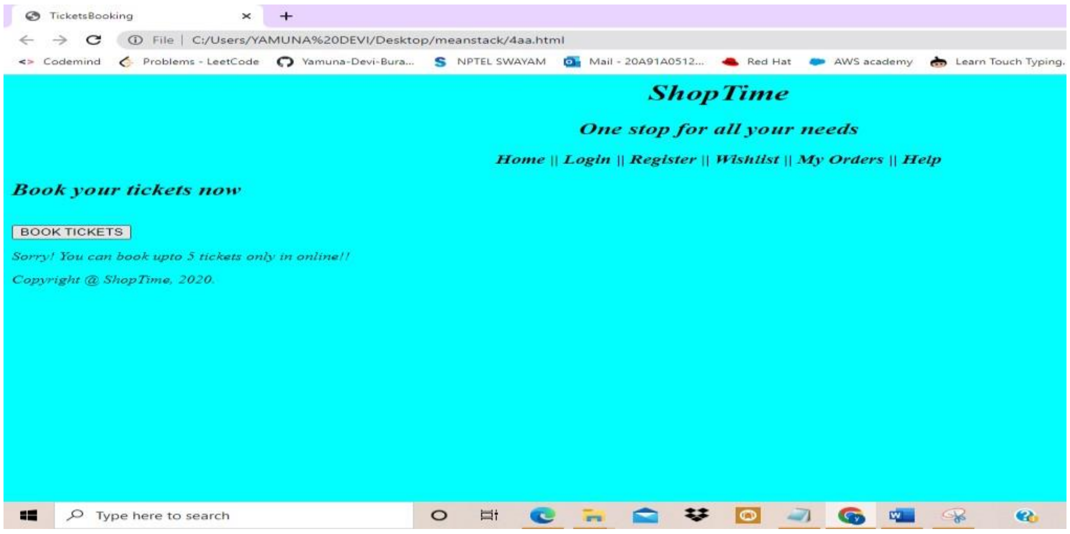
BOOK TICKETS

Total amount you need to pay:

Rs.564

Discount Amount is: Rs.36

Copyright @ ShopTime, 2020.


ShopTime

One stop for all your needs

Home || Login || Register || Wishlist || My Orders || Help

Book your tickets now

BOOK TICKETS

Sorry! You can book upto 5 tickets only in online!!

Copyright @ ShopTime, 2020.

4.b Course Name: Javascript Module Name: Working With Classes, Creating and Inheriting Classes Create an Employee class extending from a base class Person. Hints:
(i) Create a class Person with name and age as attributes.
(ii) Add a constructor to initialize the values
(iii) Create a class Employee extending Person with additional attributes role and contact
(iv) The constructor of the Employee to accept the name, age, role and contact where name and age are initialized through a call to super to invoke the base class constructor
(v)Add a method getDetails() to display all the details of Employee

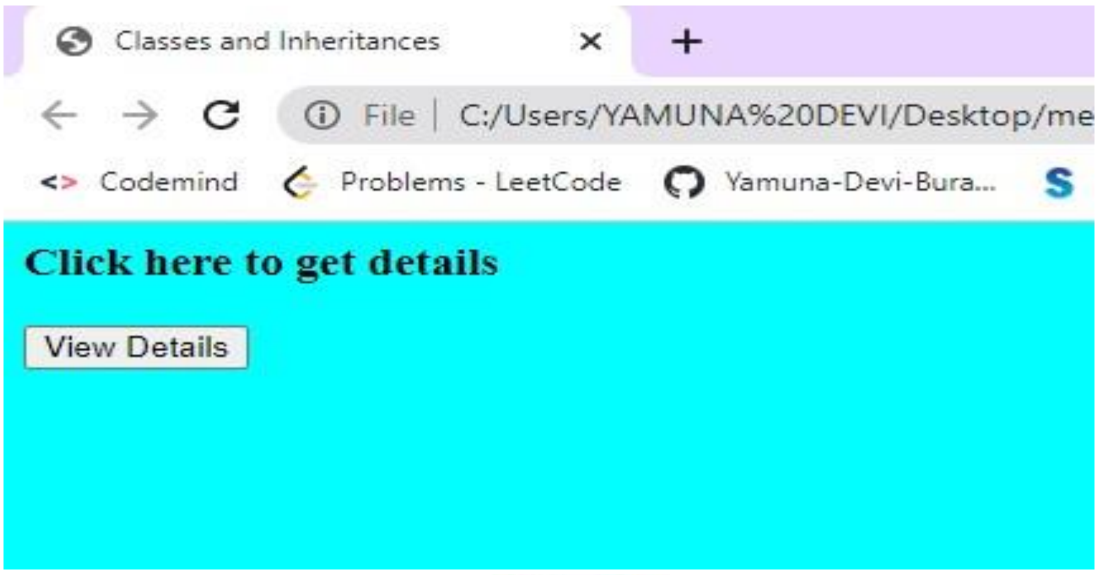
Aim: To write a Javascript with classes,creating and inheriting classes.

Description:
To create a class inheritance, use the extends keyword. The super() method refers to the parent class. By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods. Classes also allows you to use getters and setters. It can be smart to use getters and setters for your properties, especially if you want to do something special with the value before returning them, or before you set them. To add getters and setters in the class, use the get and set keywords.

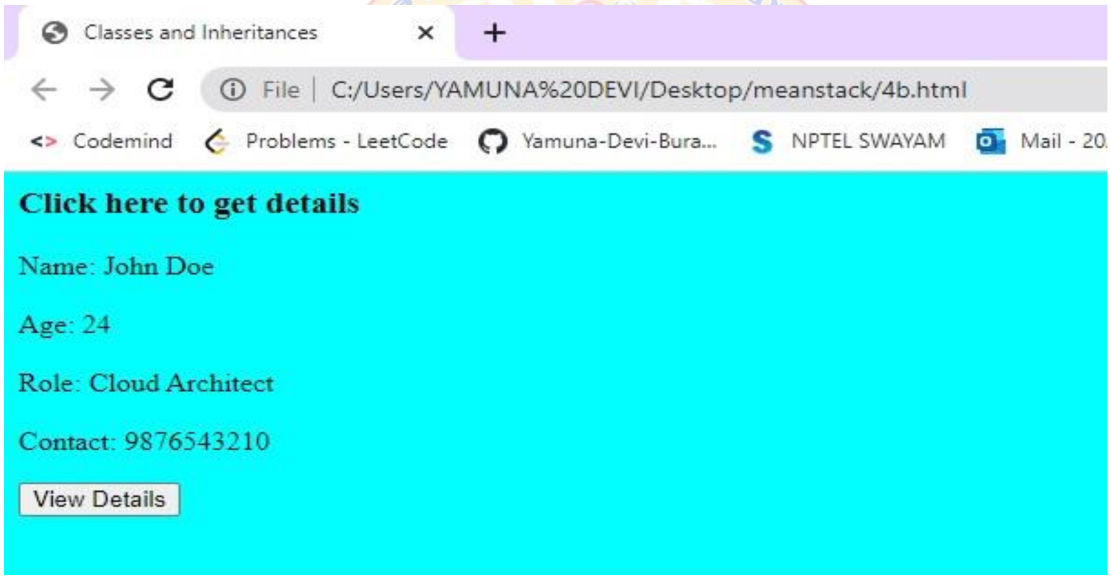
Program:
```
<html>
    <head>
        <title>Classes and Inheritances</title>
          <script>
         class Person
            {
                constructor(name,age)
                {
                  this.name=name;
                  this.age=age;
                }
                  det()
                  {
                  return "Name: "+this.name+"<br>"+"<br>"+"Age:
"+this.age;
```

```
                }
        }
        class Employee extends Person
           {
               constructor(name,age,role,contact)
               {
                 super(name,age);
                 this.roll=role;
                 this.contact=contact;
               }
               getDetails()
               {
                         return this.det()+"<br>"+"<br>"+"Role:
"+this.roll+"<br>"+ "<br>"+"Contact: "+this.contact;
                 }
           }
        function fun()
        {
                 let v=new Employee("John Doe",24,"Cloud
Architect","9876543210");
                 document.getElementById("id1").innerHTML=v.getDetails();
        }
        </script>
    </head>
    <body bgcolor="cyan">
        <h1 style="background-color:white"><center></center></h1>
        <h3>Click here to get details </h3>
        <p id="id1">
         </p></center>
        <input type="button"value="View Details" onclick="fun()">
    </body>
</html>
```

Output:



Click here to get details



Click here to get details

Name: John Doe

Age: 24

Role: Cloud Architect

Contact: 9876543210

4.c Course Name: Javascript Module Name: In-built Events and Handlers Write a JavaScript code to book movie tickets online and calculate the total price based on the 3 conditions:
(a) If seats to be booked are not more than 2, the cost per ticket remains Rs. 150.
(b) If seats are 6 or more, booking is not allowed.
(c) If seats to be booked are more than 2 but less than 6, based on the number of seats booked, do the following - Calculate total cost by applying discounts of 3, 5, 7, 9, 11 percent, and so on for customer 1,2,3,4 and 5. Try the code with different values for the number of seats. Write the following custom functions to implement given requirements:
(i) calculate Cost(seats): Calculate and display the total cost to be paid by the customer for the tickets he has bought.
(ii) calculate Discount(seats): Calculate discount on the tickets bought by the customer. Invoke this function only when the user clicks on a link/button.

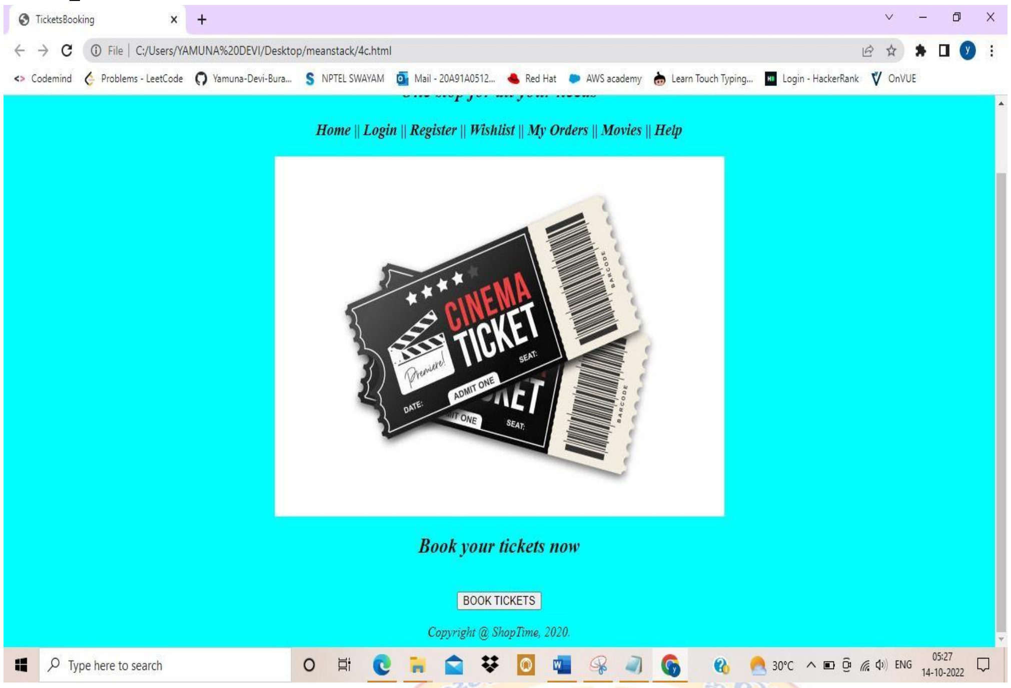
Aim: To write a Javascript code to book movie tickets online and calculate the total price.

Description:
HTML events are "things" that happen to HTML elements. An HTML event can be something the browser does, or something a user does. JavaScript lets you execute code when events are detected. HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

Program:
```
<html>
<head>
<title>TicketsBooking</title>
        <script>
var x;
var y;
var z;
fun=()=>
    {
        var a=prompt("Enter the number of tickets:");
        if(a<6)
        {
                document.getElementById("id").innerHTML="Total amount
        you need to pay:";
                document.getElementById("id1").innerHTML="Rs."+calculat
        eCost(a);
                document.getElementById("id2").innerHTML="Discount
        Amount is: Rs."+calculateDiscount(a);
        }
        else
        {
            document.getElementById("id").innerHTML="Sorry! You can
book upto 5 tickets only in online!!";
            document.getElementById("id1").innerHTML="";
document.getElementById("id2").innerHTML="";
        }
}
const p=150;
calculateCost=(a)=>{
        var i=1;
        s=0;
        j=0;
        k=0.03;
        if(a>2 && a<6)
        {
            do
            {
                j=p-(p*k);
                s+=j;
                j=0;
                k+=0.02;
                i+=1;
            }
            while(i<=a);
          }
        else if(a<=2)
        {
          s=p*a;
        }
        else
            s=0;
```

```
          return s;
}
calculateDiscount=(a)=>
{
        var g=calculateCost(a);
        var z=a*p;
        return z-g;
}
   </script>
    </head>
    <body bgcolor="cyan">
        <center><h1><i>ShopTime</i></h1></center>
        <h2 align="center"><i>One stop for all your needs<i></h2>
       <header>
             <nav align="center"><h3>
                    Home || Login || Register || Wishlist || My
Orders || Movies || Help</h3>
             </nav>
             <center>
        </header>
      <center><img src="tickets.jpg" alt="Tickets"></img>
      <h2>Book your tickets now</h2>
        <br>
        <input type="button" value="BOOK TICKETS"
onclick="fun()">       <p id="id"> </p>
        <p id="id1"> </p>
    <p id="id2"> </p>
       </body>
<footer>
    Copyright @ ShopTime, 2020.
</footer></center>
</html>
```

Output:

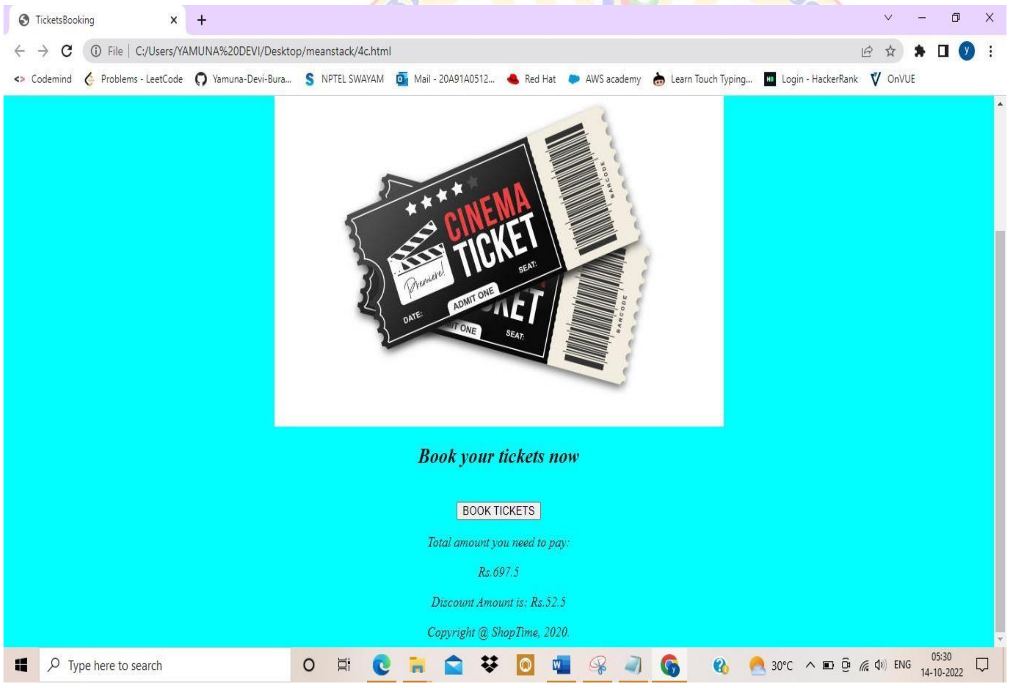


Home || Login || Register || Wishlist || My Orders || Movies || Help

Book your tickets now

BOOK TICKETS

Copyright @ ShopTime, 2020.



Book your tickets now

BOOK TICKETS

Total amount you need to pay:

Rs.697.5

Discount Amount is: Rs.52.5

Copyright @ ShopTime, 2020.

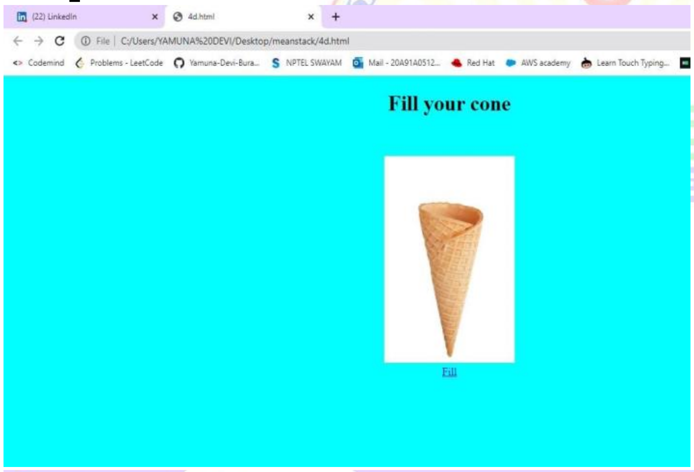
**4.d Course Name: Javascript Module Name: Working with Objects, Types of Objects,
Creating Objects, Combining and cloning Objects using Spread operator,
Destructuring Objects, Browser Object Model, Document Object Model If a user clicks on the given link, they should see an empty cone, a different heading, and a different message and a different background color.
If user clicks again, they should see a refilled cone, a different heading, a different message, and a different color in the background.**

Aim: To write a Javascript with Objects, Creating Objects, Combining and cloning Objects using Spread operator, Destructuring Objects, Browser Object Model, Document Object Model.

Description:
A JavaScript object has properties associated with it.
A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dotnotation.
objectName.propertyName

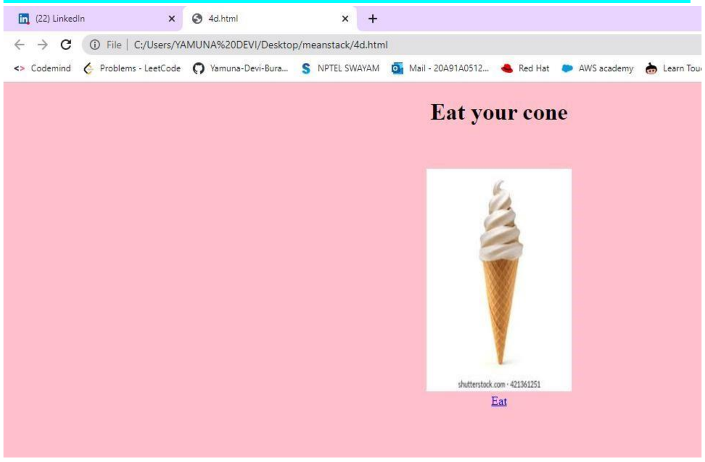
Program:
```
<!DOCTYPE html>
<html>    <script>
var c=0;
function fun()
    {
if(c==0)
 {
       document.body.style.backgroundColor = "cyan";
       document.getElementById("id1").innerHTML="Fill your cone";
       document.getElementById("imag").src="cone.jpg";
       document.getElementById("link").innerHTML="Fill";
       c=1;
}
else
{
       document.body.style.backgroundColor = "pink";
       document.getElementById("id1").innerHTML="Eat your cone";
       document.getElementById("imag").src="cone1.jpg";
       document.getElementById("link").innerHTML="Eat";
       c=0;
}
}
  </script>
   <center>
    <h1 id="id1">Eat your cone</h1>
    <br><br>
    <img src="cone1.jpg" alt="Reload"
height="300px" width="200px" id="imag">
         <br>
    <a href="javascript:fun()" id="link">Eat</a>
   </center>
 </body>
</html>
```

Output:



Fill your cone

Fill



Eat your cone

Eat

6.e Course Name: Node.js Module Name: File Operations Create a text file src.txt and add the following data to it. Mongo, Express, Angular, Node.

Aim:

To create a text file src.txt and add the following data to it.

Description:

Some of the file operations that we will be discussing are:

1. Writing data to a file
2. Reading data from a file
3. Updating content in a file

The File System module has the following methods for creating a new file and writing data to that file:

- writeFile()
- appendFile()

WriteFile:

The fs.writeFile() method will overwrite the content if the content already exists.

If the file does not exist, then a new file will be created with the specified name and content.

Syntax:

fs.writeFile(file, data, callback);

1. file: Placeholder to give the file name in which you are going to write the data.
2. data: The data/content must be written to the file.
3. callback: The callback method, that will be executed, when 'writeFile()' function is executed. This callback will be executed in both success as well as failure scenarios.
4.

AppendFile:

The appendFile() method first checks if the file exists or not. If the file does not exist, then it
creates a new file with the content, else it appends the given content to the existing file.

Syntax:

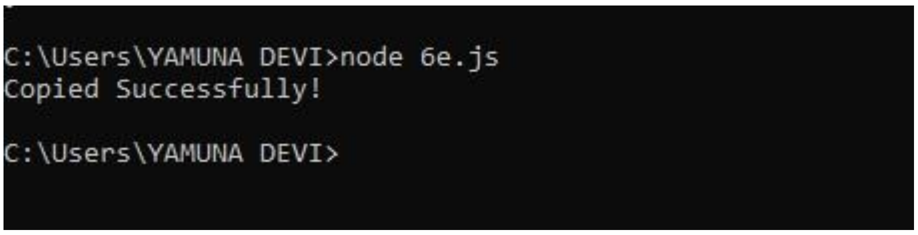
fs.appendFile(path, data, callback)

1. path: Placeholder to give the file name in which you are going to append the data.
2. data: The data/content which must be appended to the file.
3. callback: The callback method, that will be executed, when 'appendFile()' function is executed. This callback will be executed in both success as well as failure scenarios.

Program

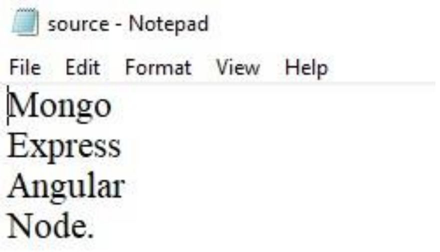
```
const fs = require('fs');
const src = "source.txt";
const dest = "destination.txt";
fs.copyFile(src, dest, (error) => {
 // incase of any error
if (error)
{
console.error(error);
return;
 }
console.log("Copied
Successfully!");
});
```

Output in command prompt:



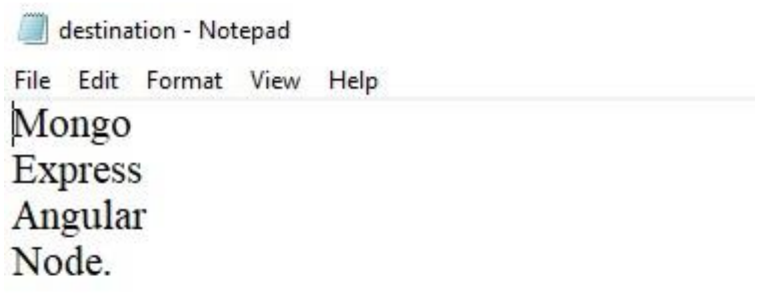
```
C:\Users\YAMUNA DEVI>node 6e.js
Copied Successfully!

C:\Users\YAMUNA DEVI>
```

Source file:source.txt



```
Mongo
Express
Angular
Node.
```

Destination.txt



```
Mongo
Express
Angular
Node.
```

5.a Course Name: Javascript
Module Name: Creating Arrays, Destructuring Arrays, Accessing Arrays, Array Methods
Create an array of objects having movie details. The object should include the movie name, starring, language, and ratings. Render the details of movies on the page using the array.

Aim: To create an array of object having movie details.

Description:
Creating an array:- Using an array literal is the easiest way to create a JavaScript Array.
Syntax:
const *array_name* = [*item1*, *item2*, ...];

Destructuring an Array:-When destructuring in javascript, a syntax you would want to keep in mind would be placing the element you want to destructure on the right side of the assignment operator and placing the variable you want to access on the left side of the assignment operator. The variables should be in {} when destructuring objects and [ ] when destructuring arrays.
const [var1, var2, ...] = arrayName;

Accessing Arrays:-The items of an array are called elements. To access an array element, you have to write the array name, followed by square brackets and pass the index value of the element you want to access to the square brackets.

Array Methods
In JavaScript, there are various array methods available that makes it easier to perform useful calculations.
Some of the commonly used JavaScript array methods are: concat():-joins two or more arrays and returns a result indexOf():-searches an element of an array and returns its position find():-returns the first value of an array element that passes a test
findIndex():-returns the first index of an array element that passes a test
forEach():-calls a function for each element
includes():-checks if an array contains a specified element push():-aads a new element to the end of an array and returns the new length of an array

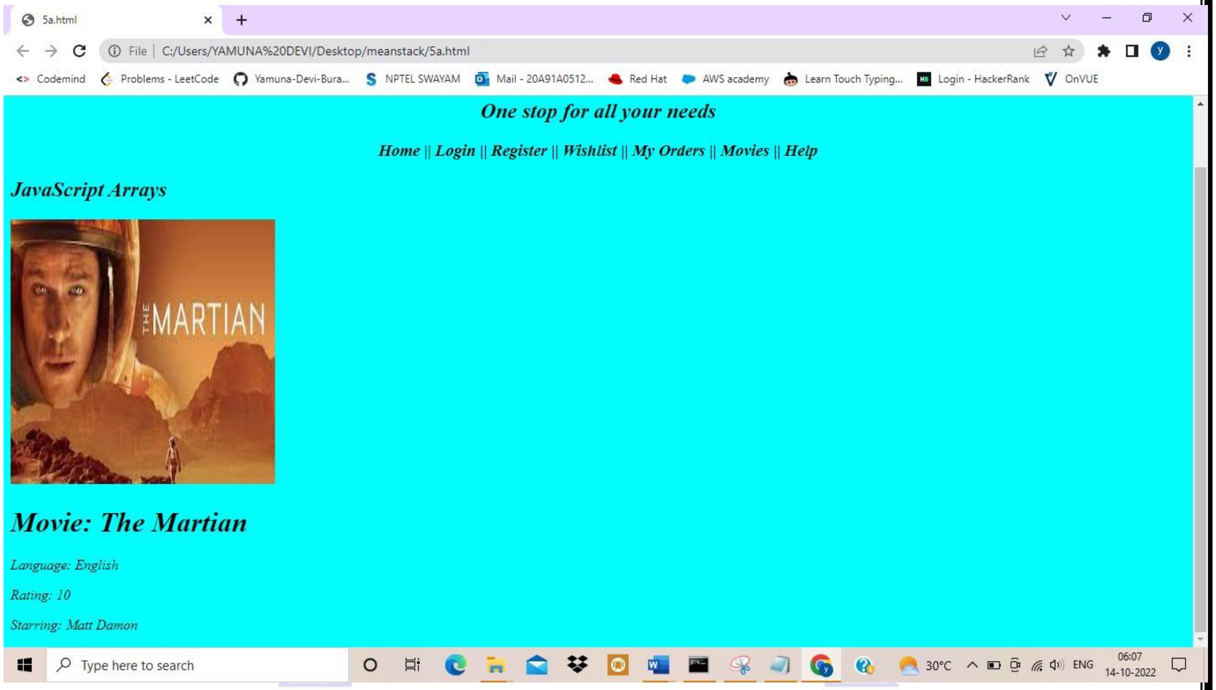
unshift():-adds a new element to the beginning of an array and returns the new length of an array
pop():-removes the last element of an array and returns the removed element
shift():-removes the first element of an array and returns the removed element
sort():-sorts the elements alphabetically in strings and in ascending order
slice():-selects the part of an array and returns the new array splice():-removes or replaces existing elements and/or adds new elements.

Program:

```
<!DOCTYPE html>
<html>
<body bgcolor="cyan">
<center><h1><i>ShopTime</i></h1>
<h2 align="center"><i>One stop for all your needs<i></h2>
       <header>
             <nav align="center"><h3>
                     Home || Login || Register || Wishlist || My
Orders || Movies || Help</h3>
             </nav>
        </header></center>
<I><h2>JavaScript Arrays</h2></I>
<img src="Martian.jpg" width="300px" height="300px"></img>
<B><h1 id="demo1"></h1></B>
<p id="demo2"></p>
<p id="demo3"></p>
<p id="demo4"></p>
<script> const
Movie = [ "The
Martian",
  "English",
  "10",
"Matt Damon",
 ];
document.getElementById("demo1").innerHTML = "Movie:  "+Movie[0];
document.getElementById("demo2").innerHTML = "Language:
"+Movie[1]; document.getElementById("demo3").innerHTML = "Rating:
```

```
"+Movie[2]; document.getElementById("demo4").innerHTML =
"Starring:  "+Movie[3];
</script>
</body>
</html>
```

Output:



One stop for all your needs

Home || Login || Register || Wishlist || My Orders || Movies || Help

JavaScript Arrays

Movie: The Martian

Language: English

Rating: 10

Starring: Matt Damon

5.b Course Name: Javascript Module Name:Introduction to Asynchronous Programming, Callbacks, Promises, Async and Await, Executing Network Requests using Fetch API Simulate a periodic stock price change and display on the console. Hints:

(i) Create a method which returns a random number - use Math.random, floor and other methods to return a rounded value.

(ii) Invoke the method for every three seconds and stop when the count is 5 – use the setInterval method.

(iii) Since setInterval is an async method, enclose the code in a Promise and handle the response generated in a success callback.

(iv) The random value returned from the method every time can be used as a stock price and displayed on the console.

Aim: To stimulate a periodic stock price change and display on the console.

Description:

To use the random function

Syntax: Math.random()

To use the setInterval function

Syntax:
myInterval = setInterval(function, milliseconds);
To stop the execution of setInterval function

Syntax: clearInterval(myInterval);

To create a Promise we have to use to following Syntax

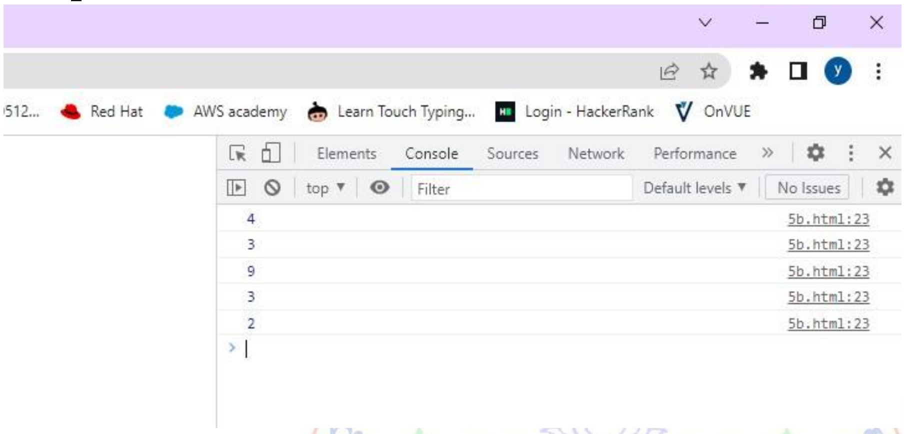
Syntax:

```
let myPromise = new Promise(function(Resolve, Reject) {

Resolve(); // when successful
Reject();  // when error
});
```

Program:

```
<!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,
initialscale=1.0">
<title>Exp__5b</title>
</head>
<body>
<script>
let c=0;
const
stock=setInterval(stokc,
3000);
function stokc(){
var myPromise = new Promise(function (resolve, reject)
{
         setTimeout(function ()
         {
                  var a=Math.floor(Math.random() * 10);
                   resolve(a);
         },
         3000);
});
myPromise.then(
         function (data)
         {
                           console.log(data);
         },
         function (error) {
                      console.log(error);
         }
);
c+=1;
if(c==5)
{
         Stop();
}
}
function Stop() {
            clearInterval(stock);
}
          </script>
</body>
</html>
```

Output:



```
4                                                        5b.html:23
3                                                        5b.html:23
9                                                        5b.html:23
3                                                        5b.html:23
2                                                        5b.html:23
```