

**1**

**a) Course Name: HTML5**

**-**

**The Language Module Name: Case**

**-**

**insensitivity, Platformindependency, DOCTYPE Declaration,**

**Types of Elements, HTML Elements**

**-**

**Attributes, Metadata**

**Element Include the Metadata element in Homepage.html for**

**providing description as "IE**

**Kart's is an online shopping website**

**that sells goods in retail. This company deals with various**

**categories like Electronics, Clothing, Accessories etc.**

<!

DOCTYPE HTML

>

<

head

>

<

title>ShopTime website</title

>

<

meta charset="UTF

-

8

"

>

<

meta name="description

" content="ShopTime is an online shopping

website that sells goods in retail. This company deals with various

categories like Electronics, Clothing, Accessories etc">

<

meta name="keywords" content="clothing,footwear,shopping"

>

<

meta name="author" content="

Myself">

<

meta name="viewport" content="width=device

-

width, initial

-

scale=1.0">

<

/Head

>

>

<

body bgcolor="cyan"

<

h1 align="center"><i>ShopTime<i></h

1>

<

h2 align="center"><i>One stop for all your needs<i></h

2>

<

p>Here we are using meta tags like

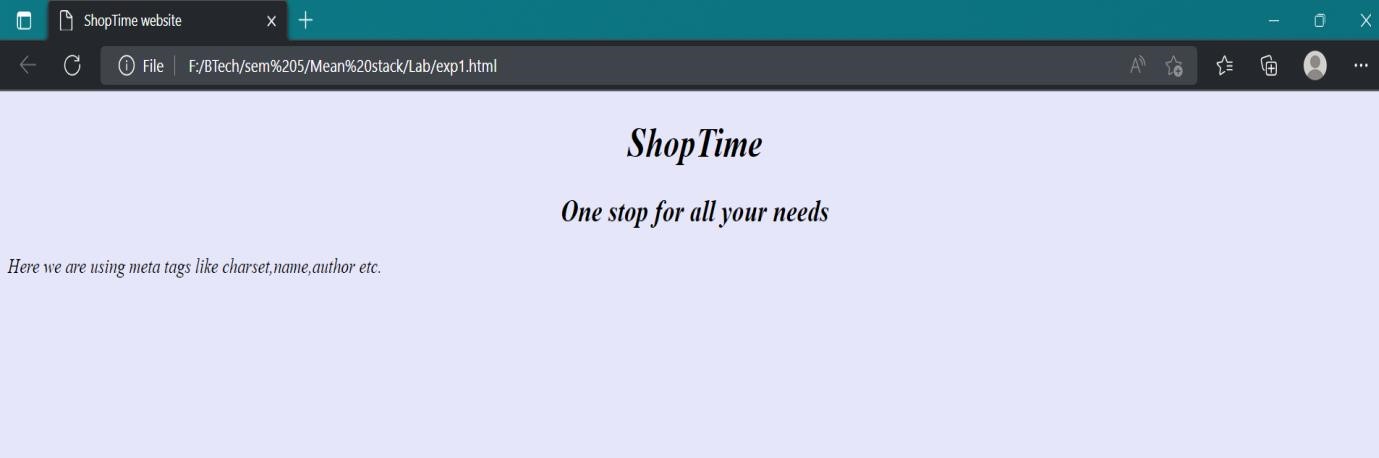
charset,name,author etc.</p>

<

/body></ht

ml>

**OUTPUT:**



**1b) Course Name: HTML5 The Language Module Name:**

**Sectioning Elements Enhance the Homepage.html of IEKart's Shopping Application by adding appropriate sectioning elements.**

<!DOCTYPE HTML>

<head>

<title>ShopTime website</title>

</Head>

<body bgcolor="cyan">

<h1 align="center"><i>ShopTime<i></h1>

<h2 align="center"><i>One stop for all your needs<i></h2> <nav align="center"><h3>

Home || Login || Register || Wishlist || My

Orders || Help</h3></nav>

<main>

<section>

<p>Clothing</p>

</section>

<section>

<p>Footwear</p>

</section>

<section>

<p>Electronics</p>

</section>

<section>

<p>Furniture</p>

</section>

<section>

<p>Cosmetics</p>

</section>

</main>

<article>

<h1>Special Offer</h1>

<aside>

<p>Download our app at PlayStore and win exciting prizes.</p></aside>

</arcicle><header>

</body>

<footer>

Copyright @ WayFar, 2020

</footer>

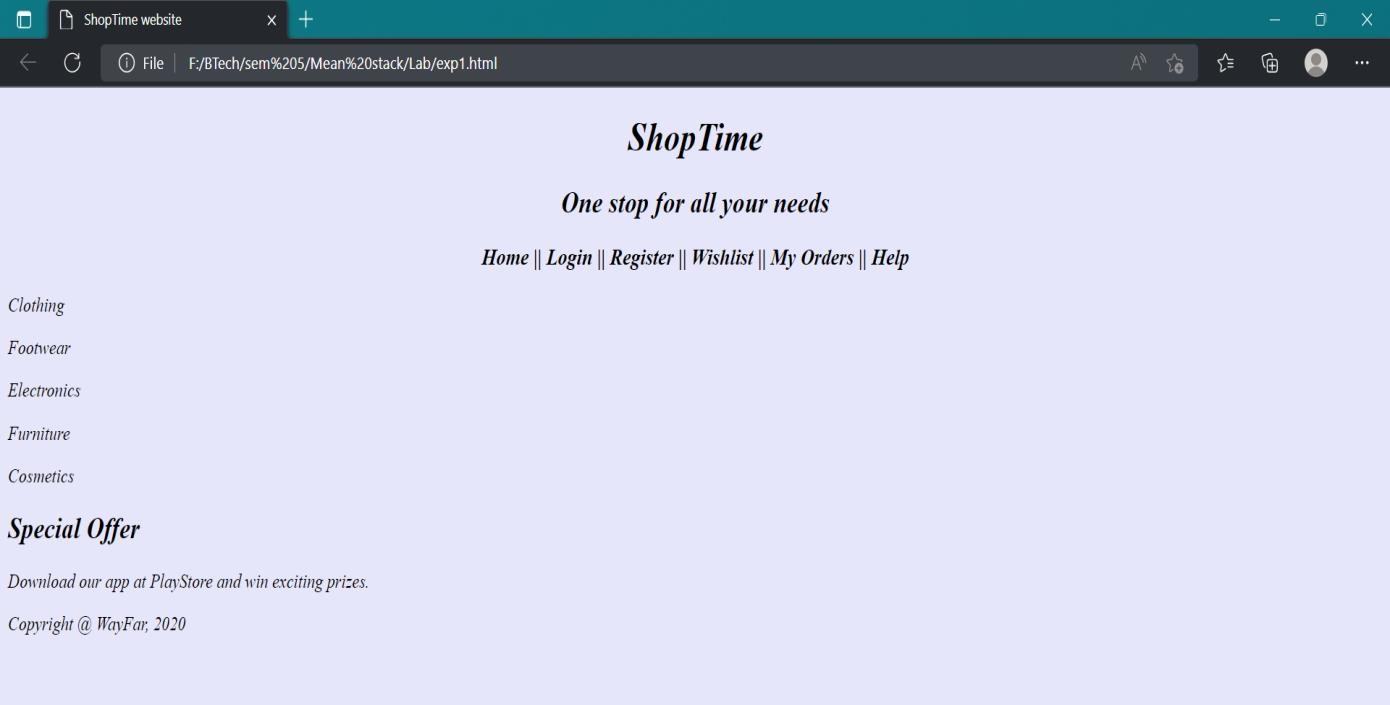


<

/html

>

**OUTPUT:**



**1**

**c) Course Name: HTML5**

**The Language Module Name: Paragraph**

**Element, Division and Span Elements, List Element Make use of**

**appropriate grouping elements such as list items to "About Us"**

**page of IEKart's Shopping Application.**

<!

DOCTYPE HTML

>

<

head

>

>

<

title>ShopTime website</title

<

/Head

>

<

body bgcolor="lavender"

>

<

h1 align="center"><i>ShopTime<i></h

1>

<

h2 align="center"><i>One stop for all your needs<i></h

2>

<

nav align="center"><h

3>

Home || Login || Track Order </h3>

<

/nav

>

>

<

center

<

p

>

<

img src="https://www.logomaker.com/wp

-

content/uploads/2018/01/FLS

-

Blog

-

Black

-

Logos\_Hero.jpg" alt="Top

shopping brands" width="950" height="300"</p>

<

/center

>

<

/body

>

>

<

footer

<

a href=aboutus.html">About Us</a

>

<

/footer

>

<

/html

>

**OUTPUT:**

**Aboutus.html**

<!

DOCTYPE HTML

>

<

html

>

<

head

>



<

title>ShopTime website</title

>

<

meta name="viewport" content="width=device

-

width, initial

-

scale=1">

>

<

/Head

<

body bgcolor="cyan"

>

<

div class="about

-

section">

<

h1>About Us Page</h

1>

<

p><span

style="color:blue;font

-

weight:bold"><i>ShopTime</i></span> is an indigenous e

-

commerce

website discovering new ways to satisfy customer's needs.</p></div>

2>

<

h2>Founders</h

<

ul

>

<

li>Jane Doe

-

Foumder&CEO</li>

<

li>John Doe

-

Art Designer</li>

<

/ul

>

>

<

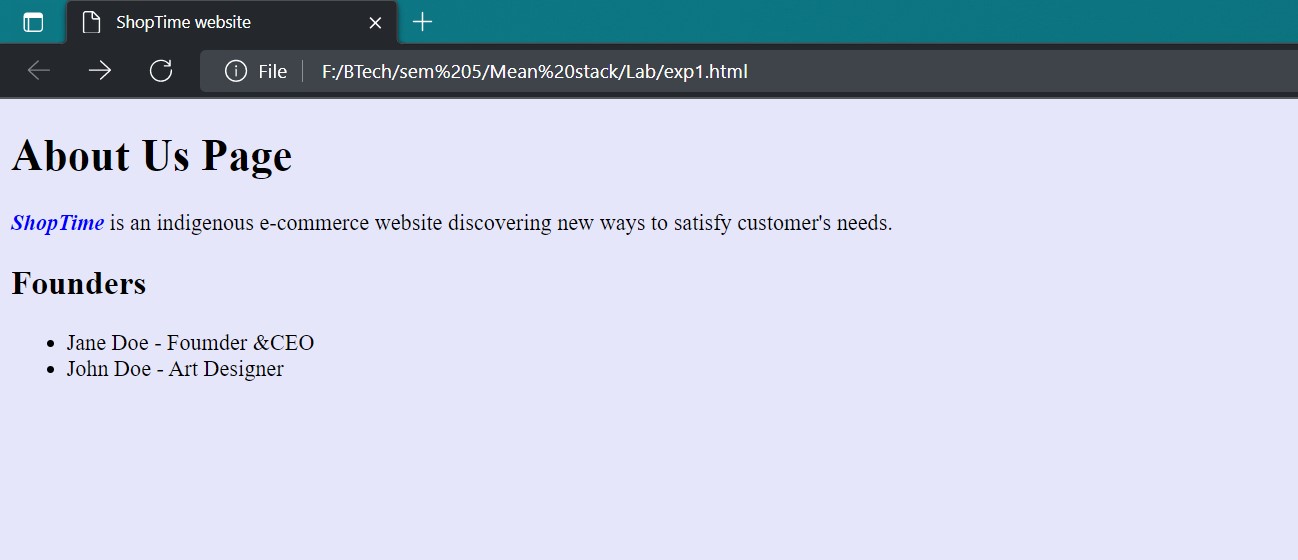
/body

<

/html

>

**OUTPUT:**



**1d) Course Name: HTML5 The Language Module Name: Link**

**Element Link "Login", "SignUp" and "Track order" to "Login.html", "SignUp.html" and "Track.html" page respectively. Bookmark each category to its details of IEKart's Shopping application**.

<!DOCTYPE HTML>

<head>

<title>ShopTime website</title>

<meta charset="UTF-8">

<meta name="description" content="ShopTime is an online shopping website that sells goods in retail. This company deals with various categories like Electronics, Clothing, Accessories etc">

<meta name="keywords" content="clothing,footwear,shopping">

<meta name="author" content="Myself">

<meta name="viewport" content="width=device-width, initialscale=1.0">

</Head>

<body bgcolor="lavender">

<h1 align="center"><i>ShopTime<i></h1>

<h2 align="center"><i>One stop for all your needs<i></h2>

<nav align="center"><h3>

<a href=”home.html”>Home</a> ||<a href=”login.html”> Login</a>

||<ahref=”trackorder.html”>Track Order</a></h3>

</nav>

<center>

<p><imgsrc="homeimg.png" alt="Top shopping brands" width="1350" height="300"</p>

</center>

</body>

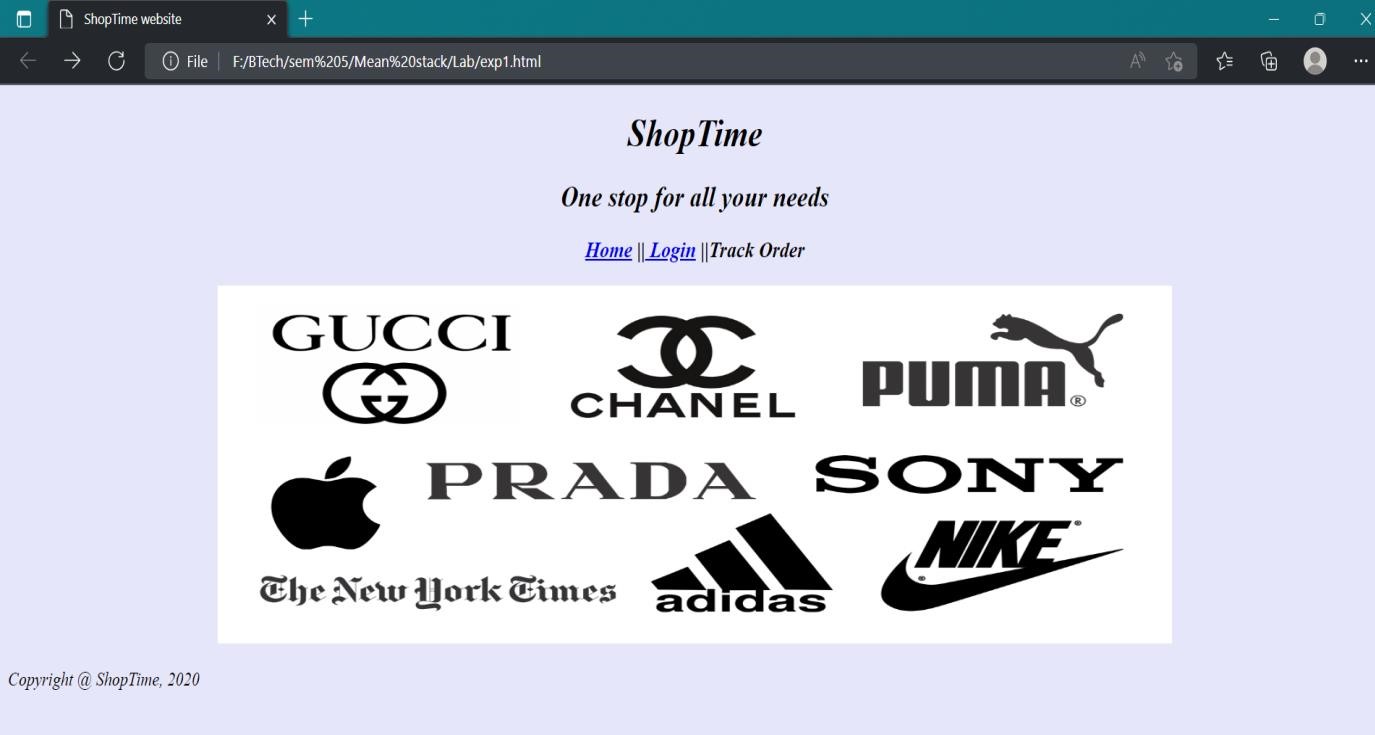
<footer>

Copyright @ ShopTime, 2020

</footer></h

tml>

**OUTPUT:**



**1e) Course Name: HTML5 The Language Module Name: Character**

**Entities Add the © symbol in the Home page footer of IEKart's Shopping application**

<!DOCTYPE HTML>

<head>

<title>ShopTime website</title>

<meta charset="UTF-8">

<meta name="description" content="ShopTime is an online shopping website that sells goods in retail. This company deals with various categories like Electronics, Clothing, Accessories etc">

<meta name="keywords" content="clothing,footwear,shopping">

<meta name="author" content="Myself">

<meta name="viewport" content="width=device-width, initialscale=1.0">

</Head>

<body bgcolor="lavender">

<h1 align="center"><i>ShopTime<i></h1>

<h2 align="center"><i>One stop for all your needs<i></h2>

<nav align="center"><h3>

<a href=”home.html”>Home</a> ||<a href=”login.html”> Login</a> ||

<a href=”trackorder.html”>Track Order</a></h3>

</nav>

<center>

<p><img src="https://www.logomaker.com/wp-

content/uploads/2018/01/FLS-Blog-Black-Logos\_Hero.jpg" alt="Top shopping brands" width="950" height="300"</p>

</center>

</body>

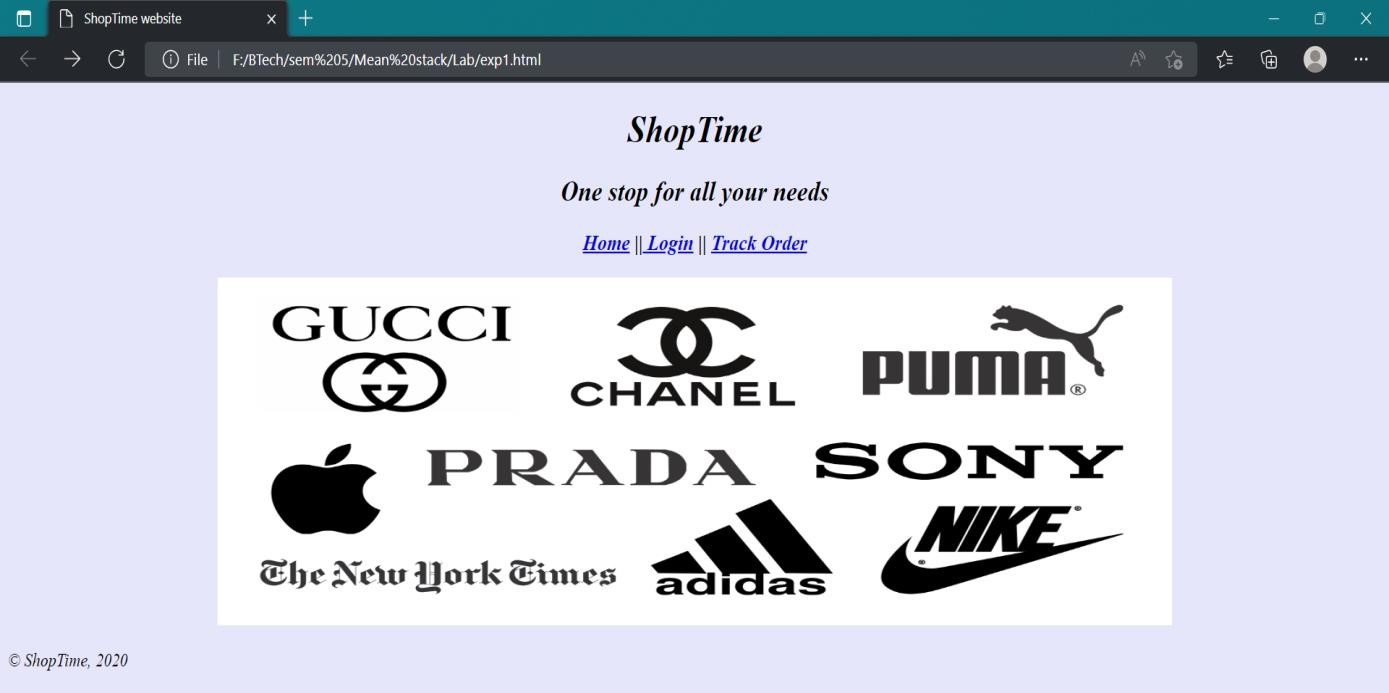
<footer>

&copy; ShopTime, 2020

</footer></html>



**OUTPUT:**



**1**

**f) Course Name: HTML5**

**The Language Module Name: HTML5**

**Global Attributes Add the global attributes such as**

**contenteditable, spellcheck, id etc. to enhance the Signup Page**

**functionality of IEKart's Shopping application.**

<

html

>

<

head><title>ShopTime</titl

e></head>

>

<

body bgcolor="cyan"

<

h1 align="center"><i>ShopTime</i></h

1>

<

form

>

<

center

>

<

h3>Sign up</h

3>

<

table

>

>

<

tr><td>Name :</td><td><input type="text"></td></tr

<

tr><td>Email:</td><td><input type="email" contenteditable="true"

spellcheck="true" ></td></tr>

<

tr><td>User Name :</td><td><input type="text"></td></tr

>

<

tr><td>Date of birth:</td><td><input type="date"></td></tr

>

<

tr><td>Password :</td><td><input type="password"></td></tr

>

<

tr><td>Confirm Password

/td><td><input

:<

type="password"></td></tr>

<

tr><td></td><tdalign="center"><input type="submit"></td></tr

>

<

/center

>

<

/form

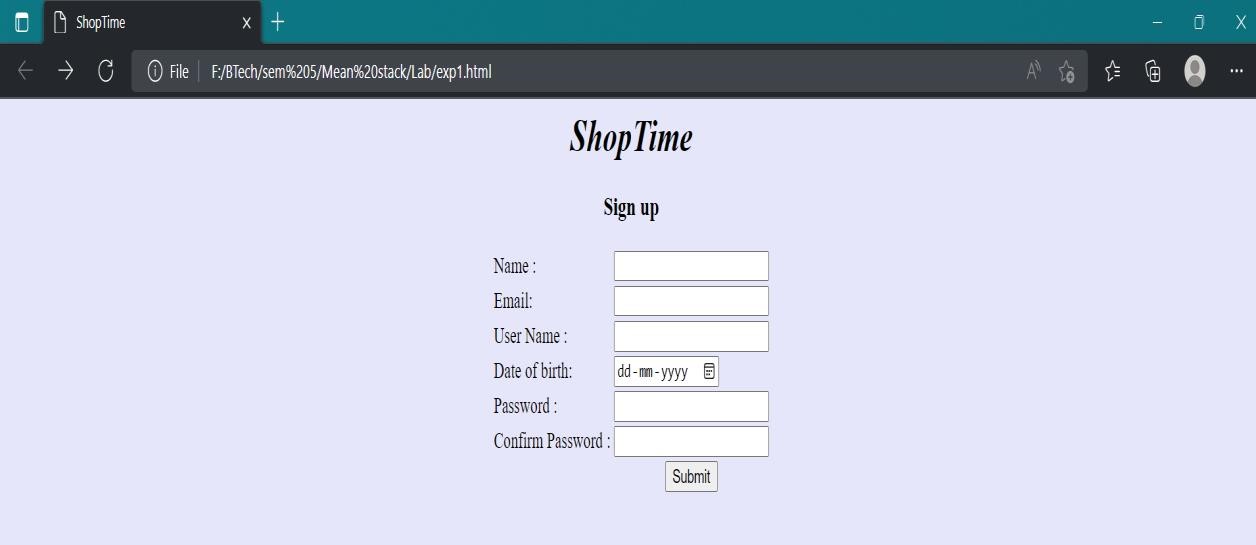
>

<

/body></

html>

**OUTPUT:**



**2a) Course Name: HTML5 - The Language Module Name: Creating Table Elements, Table Elements :Colspan/Rowspan Attributes, border, cellspacing, cellpadding attributes Enhance the details page of IEKart's Shopping application by adding a table element to display the available mobile/any inventories.**

<!DOCTYPE HTML>

<html>

<head>

<title>ShopTime Website</title>

</head>

<body bgcolor="lavender">

<table cellspacing="1" cellpadding="0" border="1" align="center">

<caption><h1>Electronics</h1></caption>

<b><tr bgcolor="white"><td>Smartwatches & Fitness trackers</td><td><img

src="https://consumer.huawei.com/content/dam/huawei-cbgsite/common/mkt/pdp/wearables/watch-fit/dynamic/watchfit/img/pc/huawei-watch-fit-personal-assistant-3.jpg" width = "120px" height = "120px" alt="Smartwatches"></td></tr>

<tr><td>Earbuds</td><td><img

src="https://i.pinimg.com/originals/db/96/6c/db966cbb958a6c398e

4e099f423ffb56.jpg" width = "130px" height = "130px" alt="Earbuds"></td></tr>

<tr bgcolor="white"><td>Speakers</td><td><img src="https://www.monitoraudio.com/site/assets/files/33185/silver10

0le-actionblock-2.jpg" width = "130px" height = "130px" alt="Speakers"></td></tr></b>

</table>

</body>

</html>



**2b) Course Name: HTML5 The Language Module Name: Creating**

**Form Elements, Color and Date Pickers, Select and**

**DatalistElements Using the form elements create Signup page for IEKart's Shopping application.**

<html>

<body bgcolor="lavender">

<form align="center">

<table align=center>

<caption><h1>Sign Up</h1></caption>

<tr><td><label>First Name:</label></td><td><input type="text"></td><br>

<tr><td><label>Email:</label></td><td><input type="email" ><br>

<tr><td><label>Date of birth:</label></td><td><input type="date"></td>

<tr><td><label>Gender: </label><td><input type="radio" name="gender" value="Male"> Male <input type="radio" name="gender" value="Female"> Female</td><br>

<tr><td><label>Mobile:</label></td><td><input type="number" ><br>

<tr><td><label>Username:</label></td><td><input type="text"><br>

<tr><td><label>Password:</label></td><td><input type="password" ><br>

<tr><td><label>Confirm Password:</label></td><td><input type="password" ><br>

<tr rowspan="3"><td><label>Address :<br></label></td><td><textarea rows="3" cols="30" ></textarea></td></tr><br><br>

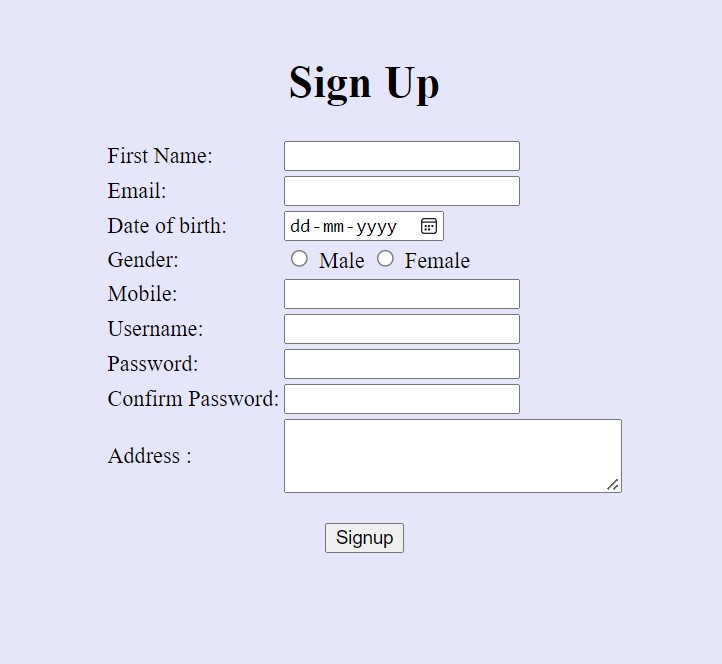
</table>

<br><button type="submit">Signup</button>

</form>

</body>

</html>



**2c) Course Name: HTML5 The Language Module Name: Input**

**Elements - Attributes Enhance Signup page functionality of IEKart's Shopping application by adding attributes to input elements.**

<html>

<body bgcolor="lavender">

<form align="center">

<table align=center>

<h1 align="center"><i>ShopTime</i></h1>

<caption><h3>Sign Up</h3></caption>

<tr><td><label>Name:</label></td><td><input type="text"></td><br>

<tr><td><label>Mobile:</label></td><td><input type="number" autocomplete="on"></td><br>

<tr><td><label>Date of birth:</label></td><td><input type="date"></td>

<tr><td><label>Gender: </label><td><input type="radio" name="gender" value="Male"> Male <input type="radio" name="gender" value="Female"> Female</td><br>

<tr><td><label>Email:</label></td><td><input type="email"><br>

<tr><td><label>Username:</label></td><td><input type="text" pattern="[A-Zaz]" maxlength="20" minlength="9"><br>

<tr><td><label>Password:</label></td><td><input type="password" placeholder="\*\*\*\*\*\*\*\*"><br>

<tr><td><label>Confirm Password:</label></td><td><input type="password" placeholder="\*\*\*\*\*\*\*\*"><br>

<tr rowspan="3"><td><label>Address :<br></label></td><td><textarea rows="3" cols="30" spellcheck="true" ></textarea></td></tr><br><br>

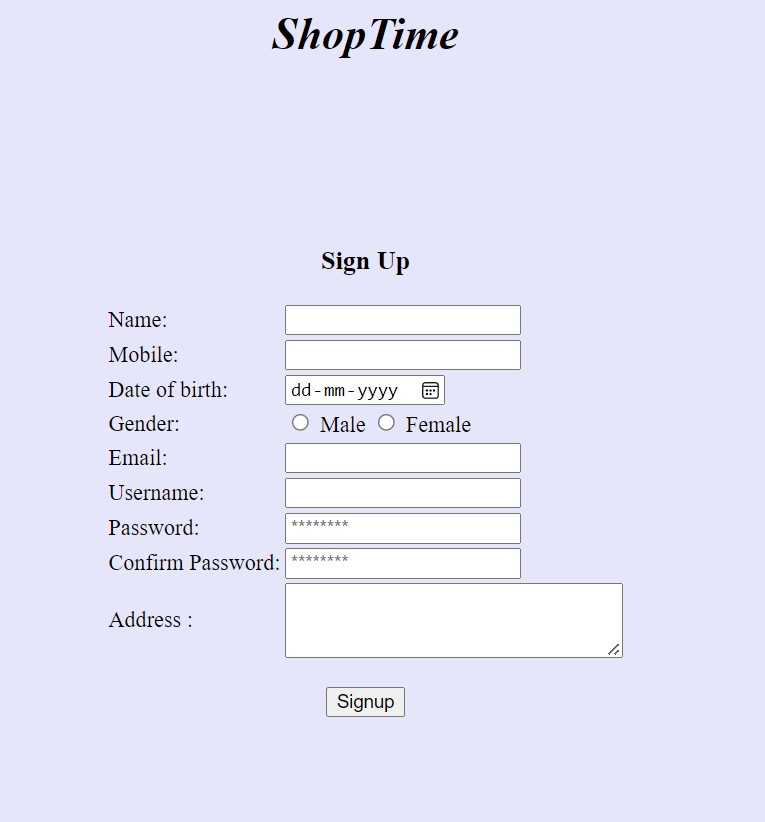
</table>

<br><button type="submit">Signup</button>

</form>

</body>

</html>



**2d) Course Name: HTML5 The Language Module Name: Media, Iframe Add media content in a frame using audio, video, iframe elements to the Home page of IEKart's Shopping application**.

<!DOCTYPE html>

<html>

<body bgcolor="lavender">

<h1 align="center"><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 || Help</h3>

</nav>

<center>

</header>

<p>

<iframesrc="homeimg1.png" name="iframe\_1 height="300" width="600" title="Iframe Example"></iframe>

<iframesrc="video1.Mp4" name="iframe\_2" height="300" width="600" title="Iframe Example"></iframe>

<iframesrc="audio.Mp3" name="iframe\_3" height="100" width="1200" title="Iframe Example"></iframe></p>

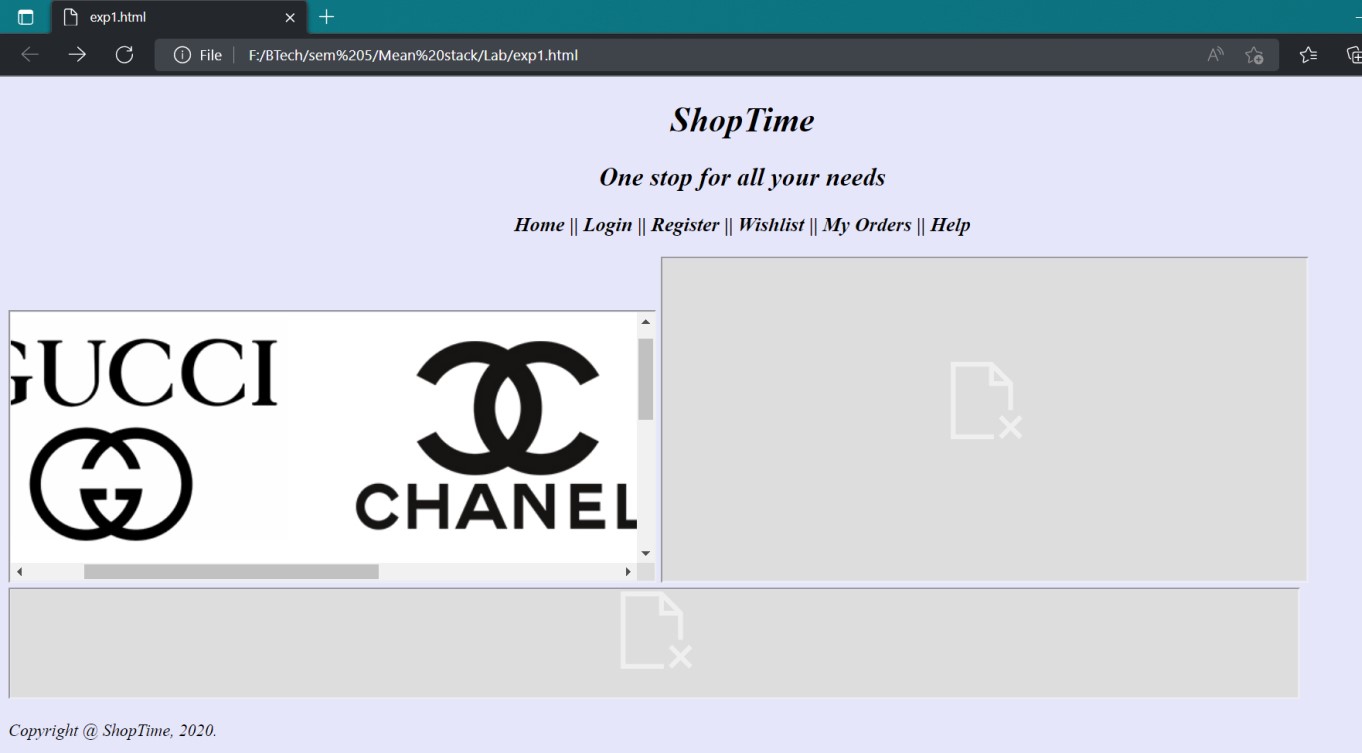
</body>

<footer>

Copyright @ ShopTime, 2020.

</footer>

</html>



|  |  |
| --- | --- |
| **3.a Course Name: Javascript Module Name: Type of Identifiers Write a JavaScript program to find the area of a circle using radius (var and let - reassign and observe the difference with var and let) and PI (const).**  <html>  <body>  <h2>Area of a circle with var and let</h2>  <script>  var rad=10;  const pi=3.14; area=pi\*rad\*rad;  document.write("Area of circle using var="+area+"<br>"); var rad=14; area=pi\*rad\*rad;  document.write("Area of circle after re-assigning var=="+area+"<br>"); let radius=10; area=pi\*radius\*radius;  document.write("Area of circle using let="+area+"<br>");    </script>  </body>  </html>  **OUTPUT:** | |
|  |  |
| **LET:**  <html>  <body> |



<

h2>Area of a circle with var and let</h

2>

<

script

>

var rad=10;

const pi=3.14;

area=pi\*rad\*rad;

document.write("Area of circle using var ="+area+"<br>");

var rad=14;

area=pi\*rad\*rad;

document.write("Area of circle after re

-

assigning var=="+area+"<br>");

let radius=10;

area=pi\*radius\*radius;

document.write("Area of circle using let="+area+"<br>");

let radius=14;

area=pi\*radius\*radius;

document.write("Area of circle after re

-

assigning let="+area+"<br>");

<

/script

>

<

/body

>

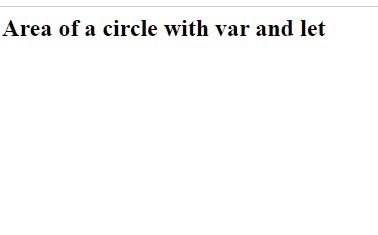
<

/html

>

**OU**

**TPUT:**



**3b) Course Name: Javascript Module Name: Primitive and Non Primitive Data Types Write JavaScript code to display the movie details such as movie name, starring, language, and ratings. Initialize the variables with values of appropriate types. Use template literals wherever necessary.**

<html>

<head>

<title>Movie Details</title>

<style> div#maincontent{ height:100px; width:500px; border:1px solid #CEE2FA; text-align:left; color:#08438E; font-family:calibri; font-size:20;

padding:5px;

}

div#heading{ text-decoration:bold; text-align:center; margin-top:80px; width:500px; border:1 px solid #CEE2FA;

text-align:center; color:#08438E; background-color:#CEE2FA; font-family:calibri; font-size:20;

padding:5px;

}

</style>

</head>

<body>

<center>

<div id="heading">Movie Details</div>

<div id="maincontent">

<script>

let movie="Interstellar"; let starring="Matthew McConaughey";



let lang="English";

const rating=4.5;

document.write(movie+"<br>"+starring+"<br>"+lang+"<br>"+rating);

<

/script

>

<

/div

>

/center

>

<

<

/body

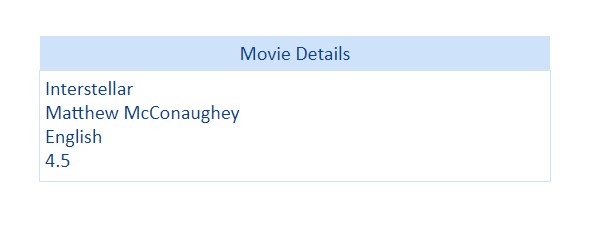
>

<

/html

>

**OUTPUT:**



**3c) Course Name: Javascript Module Name: Operators and Types of Operators Write JavaScript code to book movie tickets online and calculate the total price, considering the number of tickets and price per ticket as Rs. 150. Also, apply a festive season discount of 10% and calculate the discounted amount**

<html>

<head>

<title>Movie Details</title>

<style> div#maincontent{ height:100px; width:500px; border:1px solid #CEE2FA; text-align:left; color:#08438E; font-family:calibri; font-size:20;

padding:5px;

}

div#heading{ text-decoration:bold; text-align:center; margin-top:80px; width:500px; border:1 px solid #CEE2FA;

text-align:center; color:#08438E; background-color:#CEE2FA; font-family:calibri; font-size:20;

padding:5px;

}

</style>

</head>

<body>

<center>

<div id="heading">Movie Details</div>

<div id="maincontent">

<script>

let number\_tickets=prompt("Enter the number of tickets:"); var ticket\_amt=number\_tickets\*150; var dis=10/100\*ticket\_amt;



total\_cost=ticket\_amt

-

dis;

document.write

(

"Total number of tickets : ",number\_tickets+"<br>");

document.write("Total cost of tickets : ",ticket\_amt+"<br>");

document.write("Festive season discount is : 10%"+"<br>");

document.write("Total cost of tickets after discount :

",total\_cost+"<br>");

<

/sc

ript>

/div

>

<

<

/center

>

<

/body

>

<

/html

>

**OUTPUT:**



**3d) Course Name: Javascript Module Name: Types of Statements,**

**Non - Conditional Statements, Types of Conditional Statements, if Statements, switch Statements Write a JavaScript code to book movie tickets online and calculate the total price based on the 3**

**conditions:**

1. **If seats to be booked are not more than 2, the cost per ticket remains Rs. 150.**
2. **If seats are 6 or more, booking is not allowed.**
3. **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**

<html>

<head>

<title>Movie Details</title>

<style> div#maincontent{ height:100px; width:500px; border:1px solid #CEE2FA;

text-align:left; color:#08438E; font-family:calibri; font-size:20; padding:5px;

}

div#heading{ text-decoration:bold; text-align:center; margin-top:80px; width:500px; border:1 px solid #CEE2FA; text-align:center; color:#08438E; background-color:#CEE2FA;



n=window.prompt("Enter a number:");

document.write("For n tickets,you need to pay :",total\_cost);

font-family:calibri; font-size:20; padding:5px;

}

</style>

</head> <body>

<center>

<div id="heading">Movie Details</div>

<div id="maincontent"> <script>

if(n<=2)

{

total\_cost=n\*150;

}

else if(n>=6)

{

document.write("Bookings are not Allowed");

} else

{

if(n==3)

{

 t1=150-(150\*(3/100)); t2=150-(150\*(5/100)); t3=150-(150\*(7/100)); total\_cost=t1+t2+t3; document.write("For 3 tickets : ",total\_cost);

}

else if (n==4)

{

t1=150-(150\*(3/100)); t2=150-(150\*(5/100)); t3=150-(150\*(7/100)); t4=150-(150\*(9/100)); total\_cost=t1+t2+t3+t4; document.write("For 4 tickets : ",total\_cost);

}

else

{

t1=150-(150\*(3/100)); t2=150-(150\*(5/100)); t3=150-(150\*(7/100)); t4=150-(150\*(9/100)); t5=150-(150\*(11/100)); total\_cost=t1+t2+t3+t4+t5; document.write("For 5 tickets : ",total\_cost);

}

}

</script>

</div>

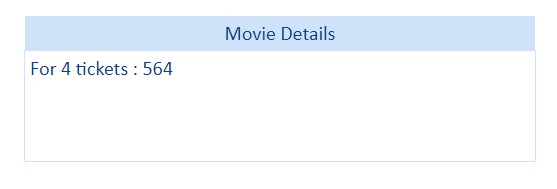
</center>

</body>

</html>



**OUTPUT:**



**3e. Course Name: Javascript Module Name: Types of Loops Write a JavaScript code to book movie tickets online and calculate the total price based on the 3 conditions:**

1. **If seats to be booked are not more than 2, the cost per ticket remains Rs. 150.**
2. **If seats are 6 or more, booking is not allowed.**
3. **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. Implement the problem statement using 'for' loop, 'while' loop and 'do-while' loop.**

<html>

<head>

<title>Movie</title>

</head>

<body>

<h1 align="center">Shop</h1>

<h2>Online Bookings</h2>

<script> for(let j=0;j<5;j++)

{

n=window.prompt("Enter a number"); if(n<=2)

{

document.write("For "+n+" tickets you need to pay "+(150\*n)+"<br>");

}

else if(n>=6)

{

document.write("Bookings are not Allowed"+"<br>");

}

else

{

if (n==3)

{

t1=150-(150\*(3/100)); t2=150-(150\*(5/100)); t3=150-(150\*(7/100)); tcost=t1+t2+t3; document.write("For 3 tickets,you need to pay :



document.write("For 4 tickets,you need to pay :

"+tcost+"<br>");

}

else if (n==4)

{ t1=150-(150\*(3/100)); t2=150-(150\*(5/100)); t3=150-(150\*(7/100)); t4=150-(150\*(9/100)); tcost=t1+t2+t3+t4;

"+tcost+"<br>");

} else { t1=150-(150\*(3/100)); t2=150-(150\*(5/100)); t3=150-(150\*(7/100)); t4=150-(150\*(9/100)); t5=150-(150\*(11/100)); tcost=t1+t2+t3+t4+t5; document.write("For 5 tickets,you need to pay :

"+tcost+"<br>");

}

}

}

</script>

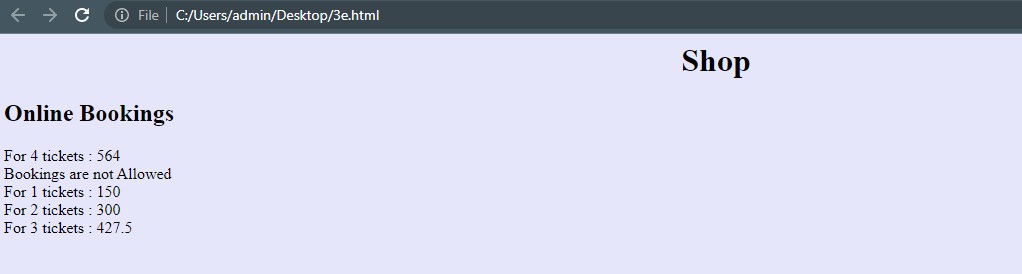
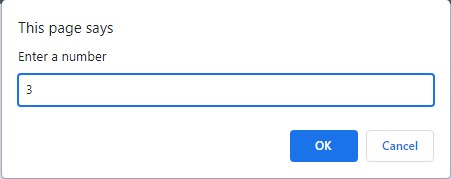
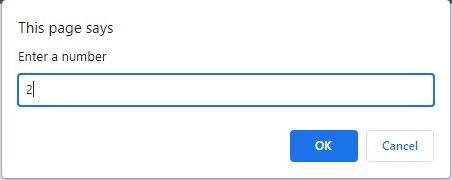
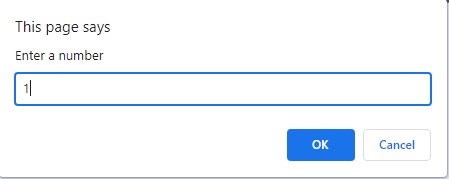
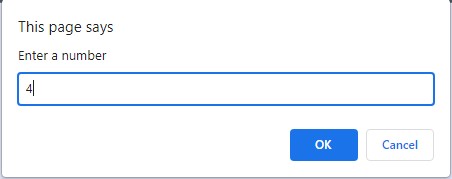
</body>

</html>



Output:

**OUTPUT:**



**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.**

1. **If seats are 6 or more, booking is not allowed.**
2. **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:**

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

<html>

<body>

<h2>Online Bookings</h2>

<script>

n=window.prompt("Enter a number:"); if(n<=2)

{

cal\_Costs(n);

}

else if(n>2 && n<6)

{

dis\_Cost(n);

}

else

{

document.write("Bookings are not Allowed");

}

function cal\_Costs(n)

{

total\_cost=n\*150;

document.write("For n tickets,you need to pay :",total\_cost);

}

function dis\_Cost(n)

{ if(n==3)

{

t1=150-(150\*(3/100)); t2=150-(150\*(5/100)); t3=150-(150\*(7/100)); total\_cost=t1+t2+t3;

document.write("For 3 tickets : ",total\_cost);

}

else if (n==4) {

t1=150-(150\*(3/100)); t2=150-(150\*(5/100)); t3=150-(150\*(7/100)); t4=150-(150\*(9/100)); total\_cost=t1+t2+t3+t4;

document.write("For 4 tickets : ",total\_cost);

}

Else{

t1=150-(150\*(3/100)); t2=150-(150\*(5/100)); t3=150-(150\*(7/100)); t4=150-(150\*(9/100)); t5=150-(150\*(11/100)); total\_cost=t1+t2+t3+t4+t5;

document.write("For 5 tickets : ",total\_cost); }

}

</script>

</body>

<

/html

>

**Output:**



**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.**

1. **Add a constructor to initialize the values**
2. **Create a class Employee extending Person with additional attributes role and contact**
3. **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.** class Person

{

constructor(name,age)

{

this.name = name;

this.age = age;

}

}

class Employee extends Person

{

constructor(name,age,role,contact)

{

super(name,age); this.role = role;

this.contact = contact;

}

getDetails()

{

console.log(`Name : ${this.name}`); console.log(`Age : ${this.age}`); console.log(`Role : ${this.role}`);

console.log(`Contact : ${this.contact}`);

}

}

let e1 = new Employee('Jack',21,'developer',1646731); e1.getDetails();

**Output:**



**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.**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script> var movie = [];

movie[0] = prompt('Enter movie name ' ); movie[1] = prompt('Enter Starring '); movie[2] = prompt('Enter Language '); movie[3] = prompt('Enter Rating ');

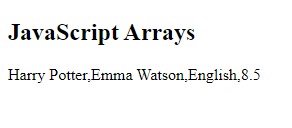
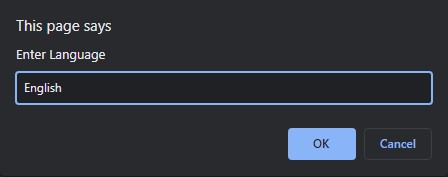
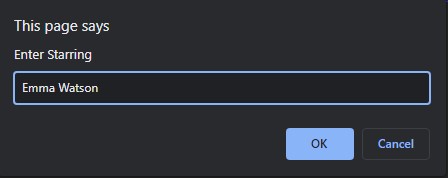
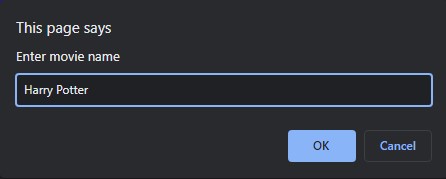
document.write(movie);

</script>

</body>

</html>

**Output:**



Exp:

Date: Page no:

|  |
| --- |
| 5.c Course Name: Javascript  Module Name: Creating Modules, Consuming Modules Validate the user by creating a login module. Hints: (i) Create a file login.js with a User class. (ii) Create a validate method with username and password as arguments. (iii) If the username and password are equal it will return "Login Successful" else will return "Unauthorized access". (iv) Create an validateUser.html file with textboxes username and password and a submit button. (v) Add a script tag in HTML to include validateUser.js file. (vi) Create an validateUser.js file which imports login module and invokes validate method of User class. (vii) On submit of the button in HTML the validate method of the User class should be invoked. (viii) Implement the validate method to send the username and password details entered by the user and capture the return value to display inthe alert.  Program:  validateUser.html  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge" />  <meta name="viewport" content="width=device-width, initial-scale=1.0" /> <title>Document</title>  </head>  <body>  <input type="text" name="name" id="name"  placeholder="Enter your user name here"  />  <input type="password" name="pass" id="password"  placeholder="Enter your password"  />  <button type="submit" id="btn">LOGIN</button>  <script src="validateUser.js" type="module"></script>  <script src="login.js" type="module"></script> </body> </html>  validateUser.js  import { User } from './login.js';  document.getElementById('btn').addEventListener('click', () => { let username = document.getElementById('name').value; let password = document.getElementById('password').value; let user1 = new User("abc", '123'); |

ADITYA ENGINEERING COLLEGE(A) Roll no:20A91A05B1



document.writeln(user1.validateUser(username, password));

})

;

Login.js

class User {

constructor(name, pass) {

this.username = name;

this.password = pass;

}

validateUser(name, pass) {

return (name == this.name && pass == this.password) ? "Login Succesfull" :

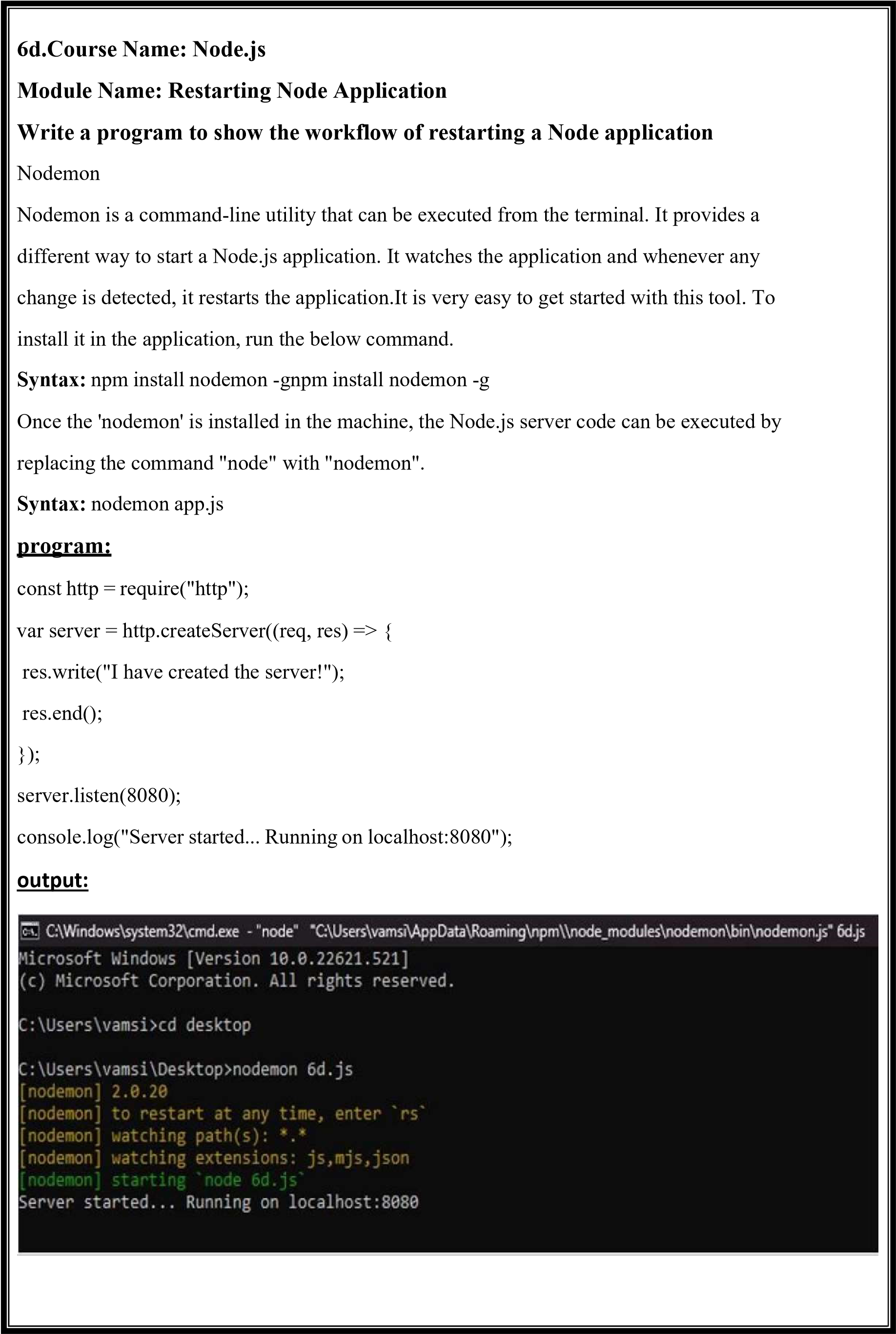
"Unauthorized access";

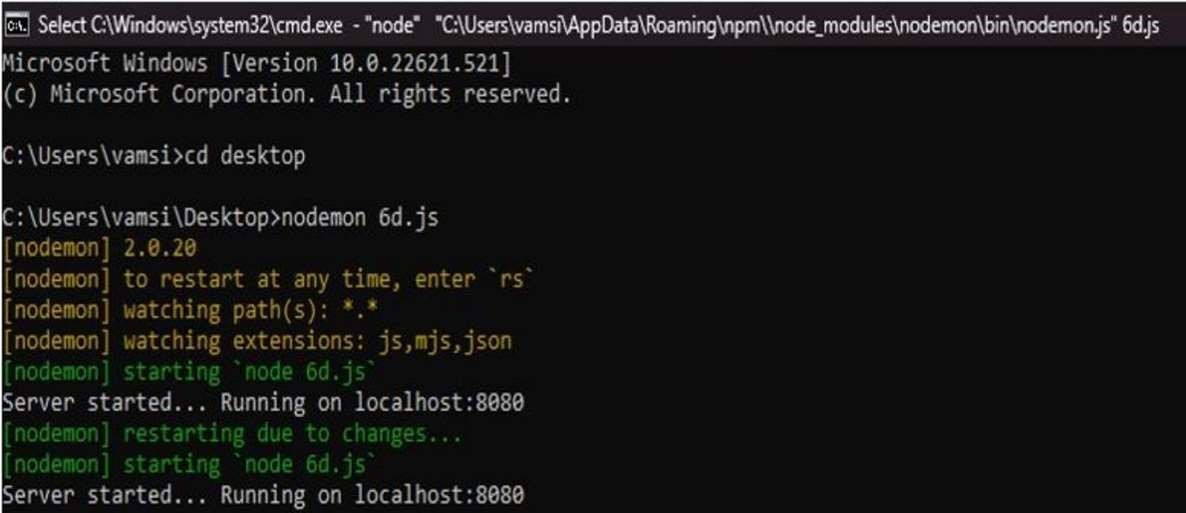
}

}

export { User };

Output:





Modified program:

const http = require("http");

var server = http.createServer((req, res) => {

res.write("I have modified the server!");

res.end();

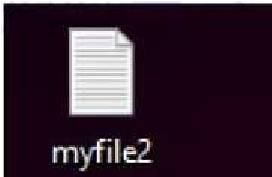
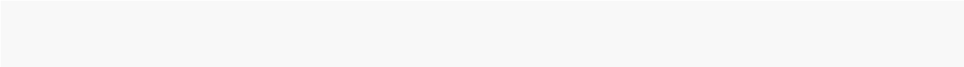
})

;

server.listen(8080);

console.log("Server started... Running on localhost:8080");

output:



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

Create Files:

The File System module has methods for creating new files:

1

.fs.appendFile()

2.

fs.open()

3.

fs.writeFile()

•

The fs.appendFile() method appends specified content to a file. If the file does not

exist, the file will be created.

•

The fs.writeFile() method replaces the specified file and content if it exists. If the file

does not exist, a new file, containing the specified content, will be created.

•

The fs.open() method takes a "flag" as the second argument, if the flag is "w" for

"writing", the specified file is opened for writing. If the file does not exist, an empty

file is created.

Program:

var fs = require('fs');

fs.appendFile('myfile2.txt', 'Mongo,Express,Angular,Node', function (err) {

if (err) throw err;

console.log('Saved!');

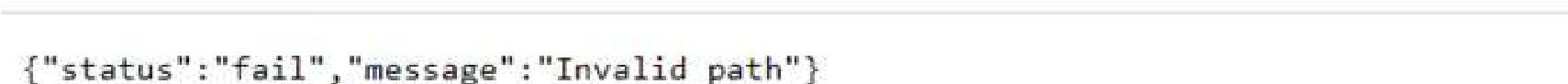
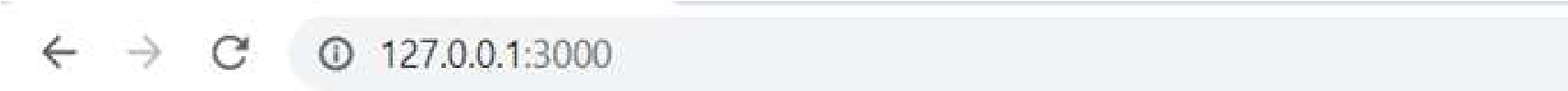
})

;

Output:

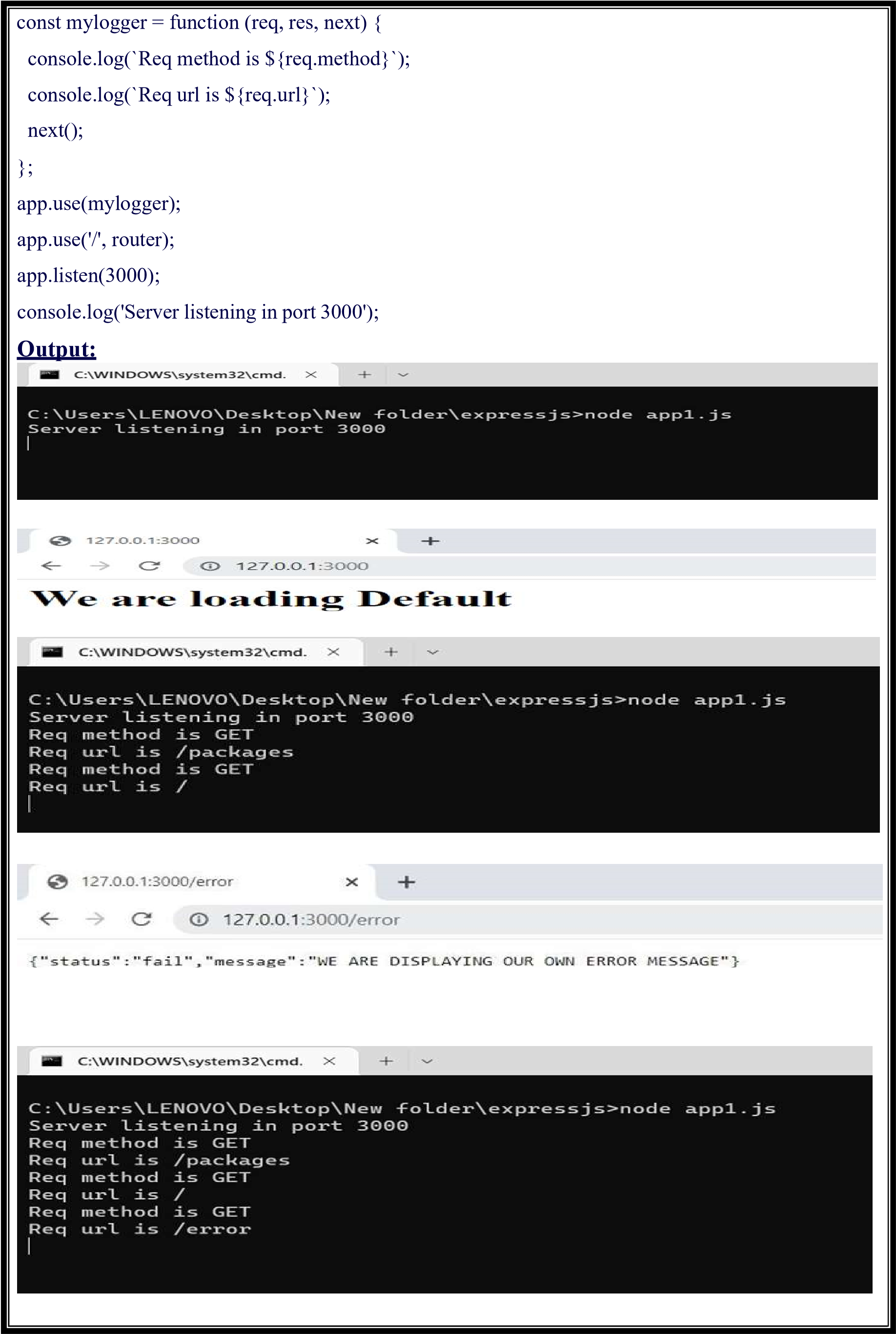
|  |
| --- |
| 7.a Course Name: Express.js  Module Name: Defining a route, Handling Routes, Route Parameters, Query  Parameters Implement routing for the AdventureTrails application by embedding the necessary code in the routes/route.js file.  Description:  Routing: The application object has different methods corresponding to each of the HTTP verbs (GET, POST, PUT, DELETE). These methods are used to receive HTTP requests. Syntax: router.method(path,handler) router: express instance or router instance method: one of the HTTP verbs path: is the route where request runs  handler: is the callback function that gets triggered whenever a request comes to a particular path for a matching request type .  Program:  //myNotes.js File  exports.packages = async (req, res) => {  try {  res.status(200).json({  message: 'You can now get the requested notes for your request ', });  } catch (err) { res.status(404).json({ status: 'fail', message: err,  });  } };  exports.bookpackage = async (req, res) => {  try {  res.status(201).json({  data: 'New booking added for the POST request', });  } catch (err) { res.status(404).json({ status: 'fail',  message: err.errmsg,  });  }  }; |

|  |
| --- |
| exports.invalid = async (req, res) => {  res.status(404).json({ status: 'fail', message: 'Invalid path',  });  };  Routing/route.js File  const express = require('express');  const routing = express.Router(); const notesController = require('../Controller/myNotes'); routing.get('/packages', notesController.packages); routing.post('/bookpackage', notesController.bookpackage); routing.all('\*', notesController.invalid); module.exports = routing;  //App.js File  const express = require('express'); const route = require('./routes/route');  const app = express(); app.use('/', route);  const port = process.env.PORT || 3000; app.listen(port, () => {  console.log(`App running on port ${port}...`); }); |



Output:

|  |
| --- |
| 7.b Course Name: Express.js  Module Name: How Middleware works, Chaining of Middlewares, Types of Middlewares In myNotes application: (i) we want to handle POST submissions. (ii) display customized error messages. (iii) perform logging.  Description:  A middleware can be defined as a function for implementing different cros-cutting concerns such as authentication, logging, etc.  Themain arguments of a middleware function are the request object, response object, and also the next middleware function defined in the application.  A function defined as a middleware can execute any task mentioned below:   * Any code execution. * Modification of objects - request and response. * Call the next middleware function. * End the cycle of request and response.   Example to Define a MiddleWare.  const mylogger = async (req, res, next) => {  console.log(new Date(), req.method, req.url); next();  };  Program:  //Route1.js file  const express = require('express'); const router = express.Router();  const myController = require('../Controller/myNotes1'); router.get('/', myController.myMethod); router.get('/about', myController.aboutMethod); module.exports = router; //myNotes.js File  exports.myMethod = async (req, res, next) => { res.send('<h1>Welcome</h1>');  };  exports.aboutMethod = async (req, res, next) => {  res.send('<h1>About Us Page</h1>');  };  //app1.js File  const express = require('express'); const router = require('./Routes/route1'); const app = express(); |



**7.a Course Name: Express.js**

**Module Name: Defining a route, Handling Routes, Route**

**Parameters, Query**

**Parameters Implement routing for the AdventureTrails application by embedding the necessary code in the routes/route.js file.**

**Description:**

**Routing:** The application object has different methods corresponding to each of the HTTP verbs (GET, POST, PUT, DELETE). These methods are used to receive HTTP requests. **Syntax: router.method(path,handler) router:** express instance or router instance **method:** one of the HTTP verbs **path:** is the route where request runs **handler:** is the callback function that gets triggered whenever a request comes to a particular path for a matching request type .

**Program:**



message: 'You can now get the requested notes for your request ',

**//myNotes.js File**

exports.packages = async (req, res)

=> { try { res.status(200).json({

});

} catch (err) { res.status(404).json({ status: 'fail',

message: err,

});

}

};

exports.bookpackage = async (req, res)

=> { try { res.status(201).json({ data: 'New booking added for the POST request',

});

} catch (err) { res.status(404).json(

{ status: 'fail',

message: err.errmsg,

});

}

};

exports.invalid = async (req, res) => { res.status(404).json({ status: 'fail', message:

'Invalid path',

});

};

**Routing/route.js File**  const express = require('express');



const routing = express.Router();

const notesController = require('../Controller/myNotes');

routing.get('/packages', notesController.packages);

routing.post('/bookpackage', notesController.bookpackage);

routing.all('\*', notesController.invalid);

module.exports = routing;

**//App.js File** const express = require('express'); const route = require('./routes/route');

const app = express(); app.use('/', route);

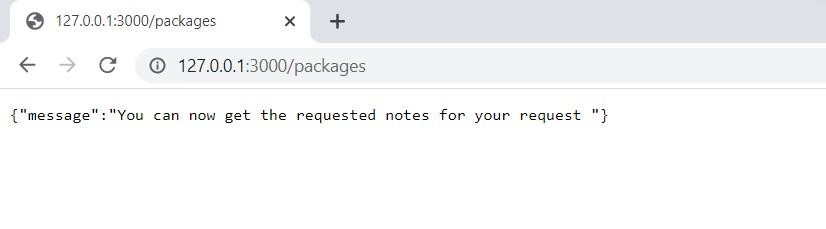
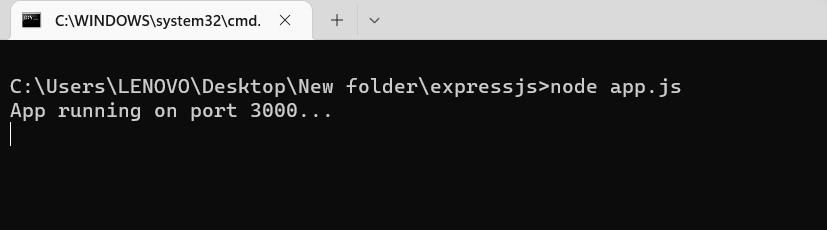
const port = process.env.PORT || 3000; app.listen(port, () => { console.log(`App running on port

${port}...`);

});



**Output:**



**7.b Course Name: Express.js**

**Module Name: How Middleware works, Chaining of Middlewares, Types of Middlewares In myNotes application: (i) we want to handle POST submissions. (ii) display customized error messages. (iii) perform logging.**

**Description:**

A middleware can be defined as a function for implementing different croscutting concerns such as authentication, logging, etc.

Themain arguments of a middleware function are the **request** object, **response** object, and also the **next** middleware function defined in the application.

A function defined as a middleware can execute any task mentioned below:

* Any code execution.
* Modification of objects - request and response.
* Call the next middleware function.
* End the cycle of request and response.

**Example to Define a MiddleWare.**

**const mylogger = async (req, res, next)**

**=> { console.log(new Date(), req.method, req.url); next();**

**};**

**Program:**

**//Route1.js file** const express = require('express'); const router = express.Router(); const myController = require('../Controller/myNotes1'); router.get('/', myController.myMethod); router.get('/about', myController.aboutMethod); module.exports = router; **//myNotes.js File**  exports.myMethod = async (req, res, next) => {

res.send('<h1>Welcome</h1>');

};

exports.aboutMethod = async (req, res, next) => { res.send('<h1>About Us Page</h1>');

};

**//app1.js File**  const express =require('express'); const router

=require('./Routes/route1'); const app = express(); const mylogger = function (req, res, next) { console.log(`Req method is ${req.method}`); console.log(`Req url is ${req.url}`); next();

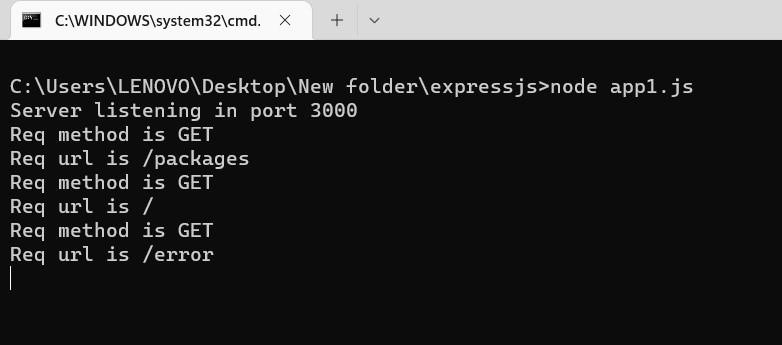
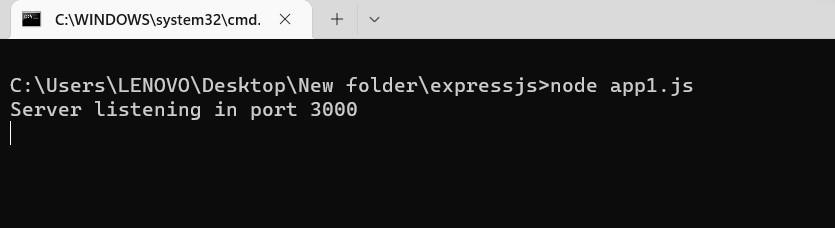
};

app.use(mylogger); app.use('/', router); app.listen(3000);

console.log('Server listening in port 3000');



**Output:**



**7.c Course Name: Express.js**

**Module Name: Connecting to MongoDB with Mongoose,**

**Validation Types and Defaults**

**Write a Mongoose schema to connect with MongoDB**

**Description:**

Before we get into the specifics of validation syntax, please keep the following rules in mind:



•You can manua

lly run validation using doc.validate(callback) or

•You can manually mark a field as invalid (causing validation to fail) by using

•Validators are not run on undefined values. The only exception is the

•Validation is defined in the SchemaType

•Validation is middleware. Mongoose registers validation as a pre('save') hook on every schema by default.

•You can disable automatic validation before save by setting the validateBeforeSave option

doc.validateSync()

doc.invalidate(...)

required validator.

**Program:**

const express=require('express') const mongoose=require('mongoos e') const app=express()

app.listen(3000,()=>console.log(" server

running......")) const

url="mongodb+srv://mstdatabase:mstdatabase@cluster0.xx7bb4u.mongodb .net/?retryWrites=true&w

=majority";

mongoose.connect(url).then(()=>console.lo g("Database

Connected.....")).catch(err=>console.log(er r));



**Output:**

**Creating schema**

const

express=require('express')

cons

t

mongoose=require('mongoos

e')

const app=express()

app.listen(3000,()=>console.log(" Server

running......")) const

url="mongodb+srv://mstdatabase:mstdatabase@cluster0.xx7bb4u.mongodb

.net/?retryWrites=true&w

=

majority";

mongoose.connect(url).then(()=>conso

le.lo

g("Database

Connected.....")).catch(err=>console.log(er

r)); var bookSchema =

mongoose.Schema({

name: String,

isbn: {type: String,

index: true},

author:

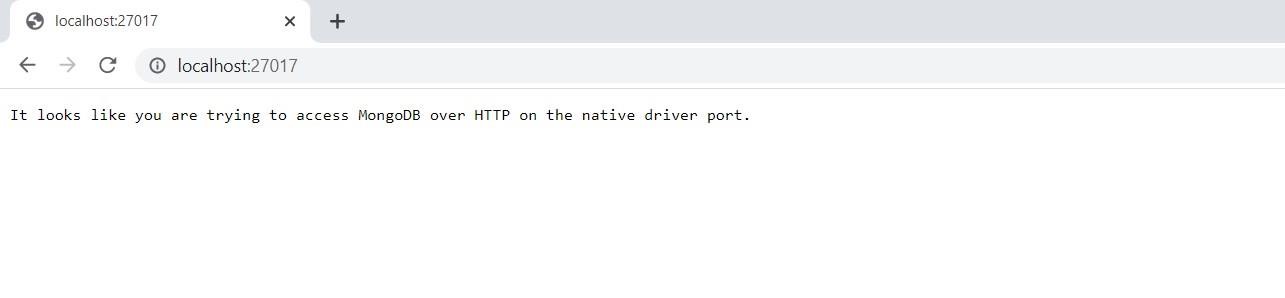
String,

pages: Number

})

;

**Output:**





**7**

**.d Course Name: Express.js**

**Module Name: Models**

**Write a program to wrap the Schema into a Model object**

**Description**

:

Schema wrapping (@graphql

-

tools/wrap) creates a modified

version of a schema that proxies, or "wraps", the original unmodified

schema. This technique is particularly useful when the original schema

cannot be changed, such as with remote schemas.

Schema wrapping works by creating a

new "gateway" schema that simply

delegates all operations to the original subschema. A series of transforms

are applied that may modify the shape of the gateway schema and all

proxied operations; these operational transforms may modify an operation

prior

to delegation, or modify the subschema result prior to its return.

**Program:**

const express=require('express')

const mongoose=require('mongoose')

const url="mongodb://0.0.0.0:27017/Hell";

mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true

.the

})

n(()=>consol e.log("Database Connected.....")).catch(err=>console.log(err));

var bookSchema = mongoose.Schema({

name: String,

isbn: {type: String, index:

true},

author: String,

pages: Number

;

})

var Book = mongoose.model("Book", bookSchema

)

;

var db = mongoose.connection; db.on("error",

console.error.bind(console, "connection error:"));

db.once("open", function(){ console.log("Connected

to DB");

})

;

**Output:**



**8.a Course Name: Express.js**

**Module Name: CRUD Operations**

**Write a program to perform various CRUD (Create-Read-UpdateDelete) operations using Mongoose library functions**

**Description:** CRUD OPERATIONS

**Create:** We’ll be setting up a post request to ‘/save’ and we’ll create a new student object with our model and pass with it the request data from Postman.Once this is done, we will use .save() to save it to the database.

**Retrieve:** To retrieve records from a database collection we make use of the .find() function.

**Update:** Just like with the delete request, we’ll be using the \_id to target the correct item.

.findByIdAndUpdate() takes the target’s id, and the request data you want to replace it with.

**Program:**

**Create:**

const express=require('express') const

mongoose=require('mongoose') const url="mongodb://0.0.0.0:27017

/Hell"; mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).the

n(()=>console.log("D atabase Connected.....")).catch(err=>console.log(err)); var bookSchema = mongoose.Schema({ name: String,

isbn: {type: String,

index: true}, author: String, pages: Number });

var Book = mongoose.model("Book", bookSchema);

var db = mongoose.connection; var

book1 = new Book({

name:"Mongoose Demo

1", isbn: "MNG123", author: "Author1,

Author2", pages: 123

});



book1.save(function(

err){

if ( err )

throw err;

console.log("Book Saved Successfully");

})

;

**Out**

**put:**

**Read**

const express=require('express')

const

mongoose=require('mongoose')

const

url="mongodb://0.0.0.0:27017

/Hell";

mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).the

n(()=>console.log("D atabase Connected.....")).catch(err=>console.log(err));

var bookSchema = mongoose.S

chema({

name: String,

isbn: {type: String,

index: true},

author:

String,

pages: Number

})

;

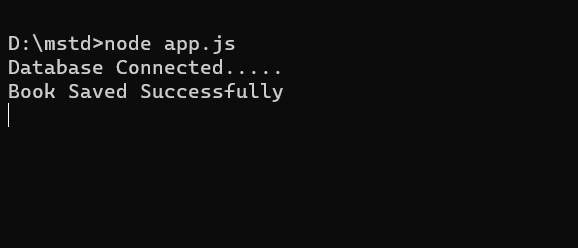
var Book = mongoose.model("Book", bookSchema);

var db =

mongoose.connection; var

queryBooks = function(){

Book.find( function(err, result){



if ( err )

throw err;

console.log("Find Operations: " + result);

})

;

}

queryBooks();

**Output:**

**Update**

const express=require('express')

const

mongoose=require('mongoose')

const

url="mongodb://0.0.0.0:27017

/Hell";

mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).the

n(()=>console.log("D atabase Connected.....")).catch(err=>console.log(err));

var bookSchema = mongoose.Schema({

name: String,

isbn: {t

ype: String,

index: true},

author:

String,

pages: Number

})

;

var Book = mongoose.model("Book",

bookSchema); var db =



mongoose.connection; var updateBook

=

function(){

Book.updateOne({$name: "JAva"}, {$set: {name: "JAVA"}},

function(err, result){ console.log("Updated successfully");

console.log(result);

})

;

}

updateBook();

**Output:**

**Delete**

const express=require('express')

const

mongoose=require('mon

goose')

const

url="mongodb://0.0.0.0:27017

/Hell";

mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).the

n(()=>console.log("D atabase Connected.....")).catch(err=>console.log(err));

var bookSchema = mongoose.Schema({

name: String,

isbn: {type: String,

index: true},

author:

String,

pages: Number});

var Book = mongoose.model("Book", bookSchema);

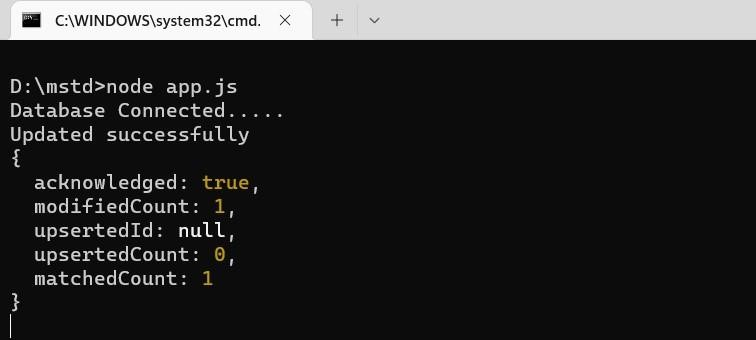
var db =

mongoose.connection; var

deleteBook = function(){

Book.deleteMany({name:"JAva"},function(err,r

esult){if(err) console.log(err); else



|  |  |  |  |
| --- | --- | --- | --- |
| Exp No:  Date: |  |  | Page No: |
| console.log("deleted")}).exec(); } deleteBook();  **After deleting records Database is:** | | | |

**8.c Course Name: Express.js**

**Module Name: Why Session management, Cookies**

**Write a program to explain session management using cookies.**

**Program:**

var express = require('express');

var cookieParser = require('cookie-parser');

var app = express(); app.use(cookieParser()); app.get('/cookieset',function(req, res){

res.cookie('cookie\_name',

'cookie\_value'); res.cookie('College', 'Aditya'); res.cookie('Branch', 'Cse'); res.status(200).send('Cookie is set');

}); app.get('/cookieget', function(req, res) { res.status(200).send(req.cook

ies);

}); app.get('/', function (req, res)

{

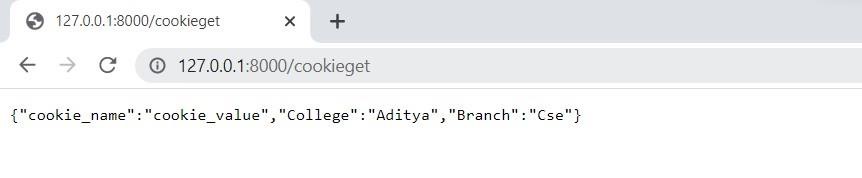
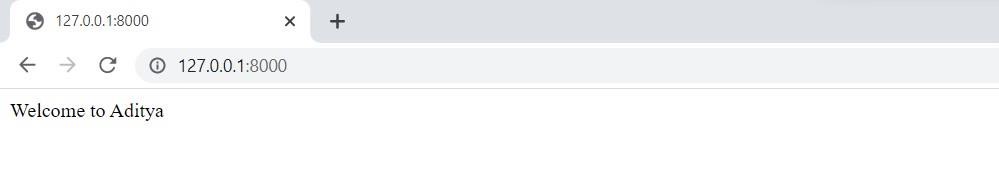
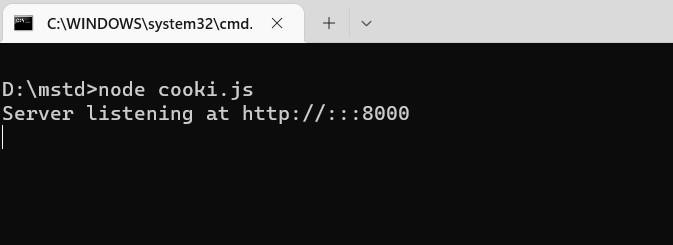
res.status(200).send('Welcome to Aditya'); }); var server = app.listen(8000, function () { var host = server.address().address; var port = server.address().port;

console.log('Server listening at http://%s:%s', host, port);

});



**Output:**



**8.d Course Name: Express.js**

**Module Name: Sessions**

**Write a program to explain session management using sessions**

**Program:**

const express = require("express") const session = require('expresssession') const app = express() var PORT = process.env.port || 3000 app.use(session({ secret: 'Your\_Secret\_Key', resave: true, saveUninitialized: true

})) app.get("/", function(req, res){ req.session.name = 'Sessionname:alr' return res.send("Session Set") }) app.get("/session", function(req, res){ var name = req.session.name return res.send(name)



})

app.listen(PORT

, function(error){

if(error) throw

console.log("Server created Successfully

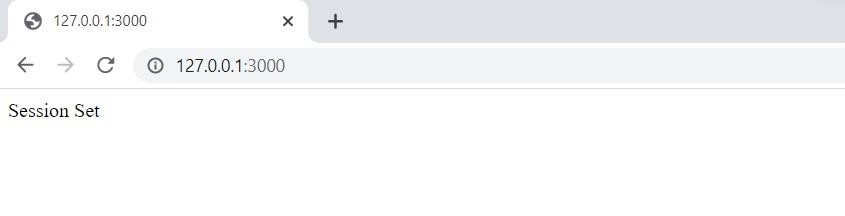
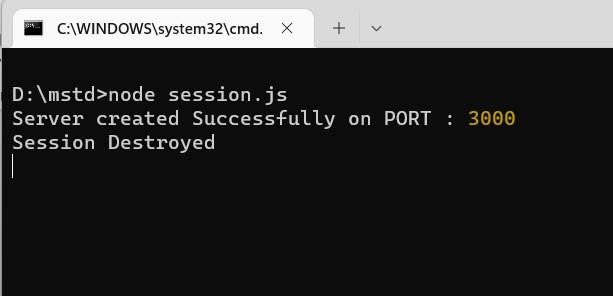
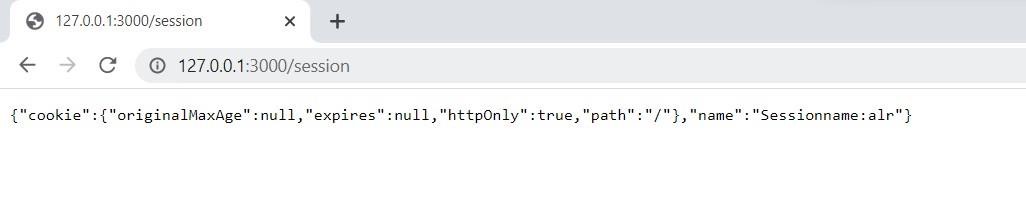
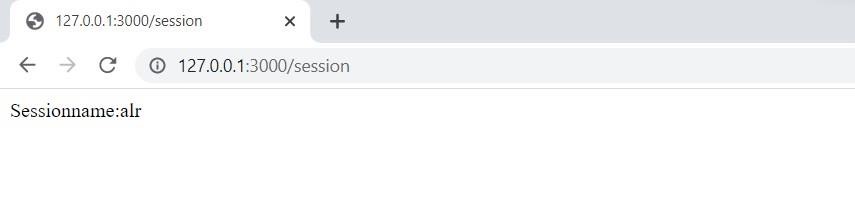
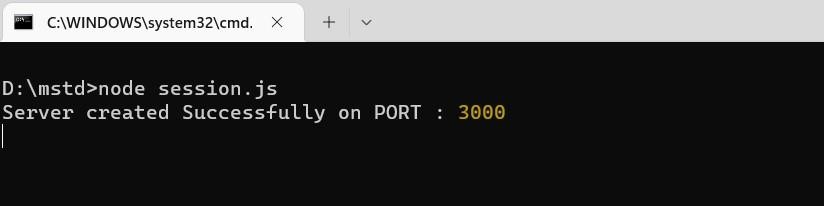
error

on PORT :", PORT)

})



**Output:**



**8.e Course Name: Express.js Module Name:**

**Why and What Security, Helmet Middleware Implement security features in myNotes application.**

**Program: App.js**  const express =

require('express'); const routing = require('./route'); const app = express(); app.use('/', routing); app.listen(3000);

console.log('Server listening in port 3000');

**route.js**  const express =

require('express'); const router = express.Router(); router.get('/', function (req, res) {

res.send('<h1>Express</h1>')

;

}); router.get('/about', function (req, res) {

res.send('About Us Page');

});

module.exports = router;

**test.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<style> p { color: red; } iframe { width: 100%; height: 90%

}

</style>



<

/head>

<

body>

<

p>Clickjacked</p>

<

iframe src="http://localhost:3000"></iframe>

<

/body>

<

/html>

**Output:**

**Implementing Helmet**

**App.js**

const express =

require('express'); const helmet

=

requi

re('helmet'); const

routing = require('./route');

const app = express();

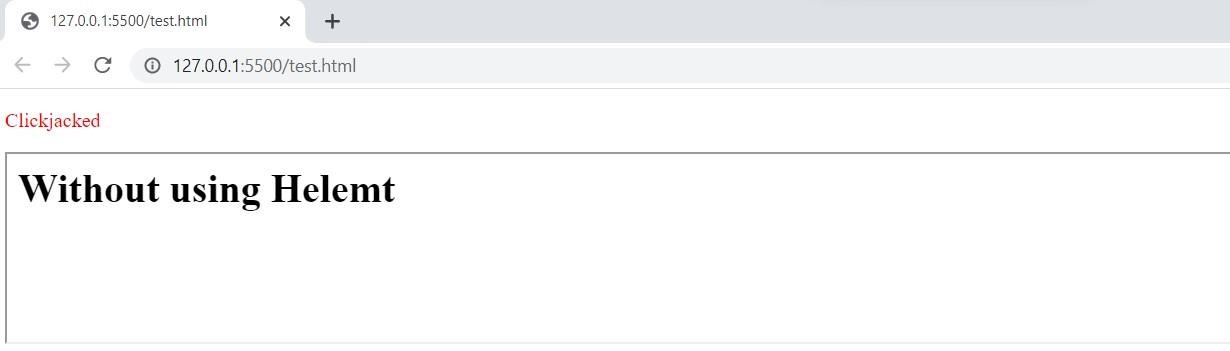
app.use(helmet()); app.use('/',

routing); app.listen(3000);

console.log('Server listening in

port 3000');

**Output:**



**9**

**.a Course Name: Typescript**

**Module Name: Basics of TypeScript**

**On the page, display the price of the mobile**

**-**

**based in three**

**different colors. Instead of using the number in our code,**

**represent them by string values like GoldPlatinum, PinkGold,**

**SilverTitanium**

**Program:**

c

onst

obj:{Gol

dPlatinum:

string}={GoldPlatinum:"$1000"}

const ob1:{PinkGold: string,}={PinkGold:"$900"}

const

ob2:{SilverTitanium:

string}={SilverTitanium:"$1500"}

console.log("

\

nMobilecolor

Price

\

n")

console.log("

\

nGoldPlatinum:

\

t

"+obj.GoldPlatinum+"

\

n")

console.log(

"PinkGold:

\

t "+ob1.PinkGold+"

\

n")

console.log("

SilverTitanium:

\

t"+ob2.SilverTitanium+"

\

n")

**Output:**



**9**

**.b) Define an arrow function inside the event handler**

**to filter the product array withthe selected product**

**object using the productId received by the function.**

**Pass the selected product object to the next screen.**

**Program:**

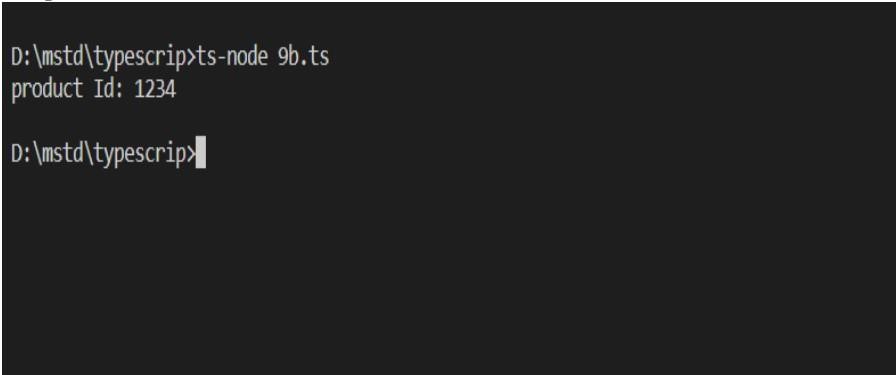
var getproductdetails=(pr

oductId : number):string=>{return "product

Id:"+productId};

console.log(getproductdetails(1234));

**Output:**



**9**

**.c Course Name: Typescript**

**Module Name: Parameter Types and Return Types**

**Consider that developer needs to declare a function**

**-**

**getMobileByVendor which accepts string as input parameter and**

**returns the list of mobiles.**

**Program:**

function getMobileByManufacturer(manufacturer:

string): string[] { let mobileList: string[]; if

(

manufacturer === 'Samsung') {

mobileList = ['Samsung Galaxy S6 Edge', 'Samsung Galaxy Note 7',

'Samsung Galaxy J7 SM

-

J700F']; return

mobileList;

}

else if (manufacturer === 'Apple') { mobileList =

[

'Apple iPhone 5s', 'Apple iPhone 6s ', 'Apple iPhone 7'];

return mobileList;

}

else { mobileList = ['Nokia 105',

'Nokia 230 Dual Sim']; return

mobileList;

}

}

console.log('The available Samsung mobile list: [' +

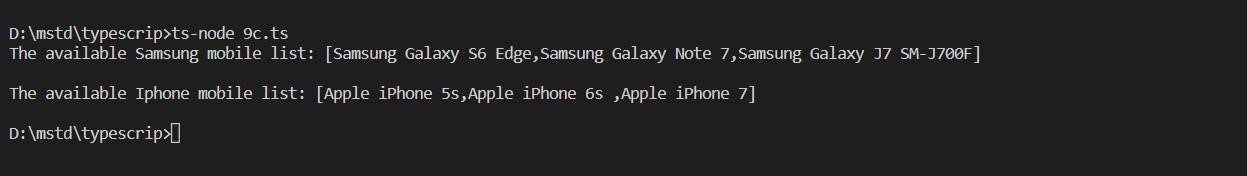
getMobileByManufacturer('Samsung')+']'); console.log('

\

nThe available

Iphone mobile list: [' + getMobileByManufacturer('Apple')+"]");

**Output:**



|  |
| --- |
| **9d)Course Name: Typescript**  **Module Name: Arrow Function Consider that developerneeds to declare a manufacturer's array holding 4 objects with id and price as a parameter and needs to implement an arrow function - myfunction to populate the id parameter of manufacturers array whose price is greater than or equal to 150 dollars then below mentioned code snippet would fit into this requirement. Program:**  var manufacturers = [{ id: 'Samsung', price: 150 }, id: 'Microsoft', price: 200 }, id: 'Apple', price: 400 }, id: 'Micromax', price: 100 }  ]; var test; console.log('Details of Manufacturer array are : ');function myFunction() {  console.log(item.id);  test = manufacturers.filter((m) => m.price >=    150)  ;for (var item of test) {      myFunction  ();  **Outpu t:** |
|  |
|  |

{

{

{

}

}



**9**

**e)Course Name: Typescript**

**Module Name: Optional and Default Parameters Declarea function**

**-**

**getMobileByManufacturer with two parameters namely**

**manufacturer andid, where manufacturer value should passed as**

**Samsung and id parameter should be optional while invoking the**

**function, if id is passed as 101 then this function should return**

**Moto mobile l**

**ist and if manufacturer parameter is either**

**Samsung/Apple then this function should return respective**

**mobile list and similar to make Samsung as default Manufacturer.**

**Below mentioned code**

**-**

**snippet would fit into this requirement**

**Program:**

function getMobile

ByManufacturer(manufacturer: string = 'Samsung',

id?: number): string[]{ let mobileList: string[]; if (id) { if (id === 101) {

mobileList = ['Moto G Play, 4th Gen', 'Moto Z Play with

Style Mod']; return mobileList;

}}

if (manufacturer === 'Samsung') {

mobileList = [' Samsung Galaxy S6 Edge', ' Samsung Galaxy Note 7',

' Samsung Galaxy J7 SM

-

J700F']; return mobileList;

else if (manufacturer === 'Apple') {

}

mobileList = [' Apple iPhone 5s', ' Apple iPhone 6s', ' Apple

iPhone 7']; return mobileList;

}

else {

mobileList = [' Nokia 105', ' Nokia 230 Dual

Sim']; return mobileList;}}

console.log('The available mobile list : ' +

getMobileByManufacturer('Apple'))

; console.log('The available mobile list

:

' + getMobileByManufacturer(undefined, 101))



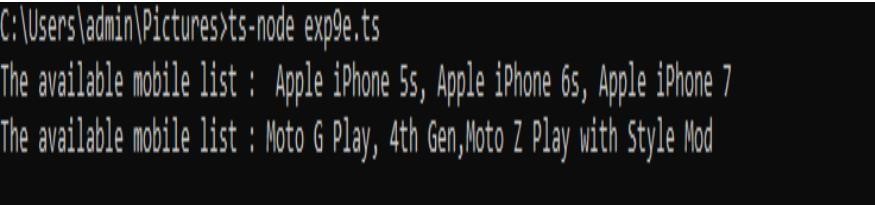
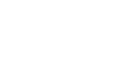
**Out**



**p**



**ut:**



**Module Name: Rest Parameter Implement business logic for**

**adding multiple Product values into a cart variable which is type**

**of string array.**

**Program:**

const cart: string[] = [];

const pushtoCart = (item: string) => {

cart.push(item); }; function

addtoCart(...productName: string[]):

string[] { for (const item of productName) {

pushtoCart(item);

}

return

cart;

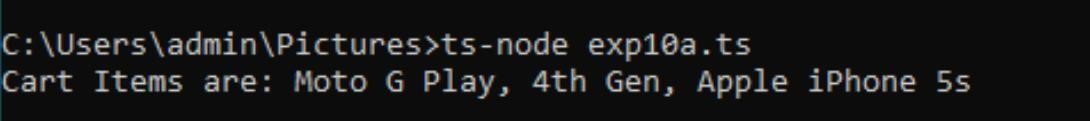
}

console.log('Cart Items are:' + addtoCart(' Moto G Play, 4th Gen', '

Apple

iPhone 5s'));

**Output:**



**Module Name: Creating an Interface Declare an interface named**

**-**

**Product with two properties like productId and productName with**

**a**

**number and string datatype and need to implement logic to**

**populate the Product details.**

**Program:**

interface Product {

productId: number ;

productName: string ;

}

function getProductDetails(productobj:

Product): string { return 'The product name is : '

+

productobj.productName;

}

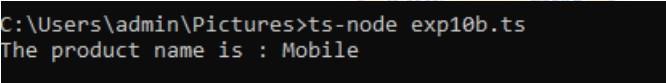
const prodObject = {productId: 1001,

productName: 'Mobile'}; const productDetails:

string = getProductDetails(prodObject);

console.log(productDetails);

**Output:**



**Module Name: Duck Typing Declare an interface named**

**-**

**Product with two properties like productId and productName**

**with the number and string datatype and need to implement logic**

**to populate the Product details.**

**Program:**

interface Product {

productId: num

ber;

productName:string;

}

function getProductDetails(productobj:

Product): string { return 'The product name is : '

productobj.productName;

+

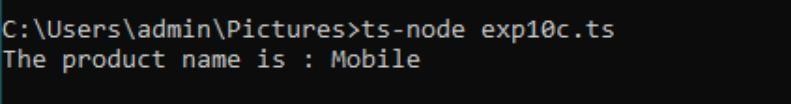
}

const prodObject = {productId: 1001, productName: 'Mobile',

productCategory: 'Gadget'}; const productDetails: string =

getProductDetails(prodObject); console.log(productDetails);

**Output:**



|  |
| --- |
| **Module Name: Function Types Declare an interface with function type and access its value**  **Program:**  function CreateCustomerID(name: string, id: number): string { return 'The customer id is ' + name + ' ' + id;    interface StringGenerator {  (chars: string, nums: number): string;    let idGenerator: StringGenerator;  idGenerator = CreateCustomerID;  const customerId: string = idGenerator('Mr.Tom', 101); console.log(customerId); **Output:** |
|  |
|  |

}

}

**11.a Course Name: Typescript**

**Module Name: Extending Interfaces**

**Declare a productList interface which extends properties from two other declared interfaces like Category,Product as well as implementation to create a variable of this interface type.**

**Program:**

interface Category { categoryName: string;

}

interface Product { productName: string; productId: number;

}

interface ProductList extends Category, Product { list: Array<string>;} const productDetails: ProductList = { categoryName: 'Gadget', productName: 'Mobile', productId: 1234,

list: ['Samsung', 'Motorola', 'LG']}; const listProduct = productDetails.list; const pname: string = productDetails.productName; console.log('Product Name is ' + pname); console.log('Product List is ' + listProduct);

**Output:**



**11.c Course Name: Typescript**

**Module Name: Constructor**

**Declare a class named - Product with the below-mentioned declarations: (i) productId as number property (ii)**

**Constructor to initialize this value (iii) getProductId method to return the message "Product id is <id>".**

**Program:**

class Product

{

productId: number;

constructor(productId: number)

{

this.productId = productId;

}

getProductId(): string {

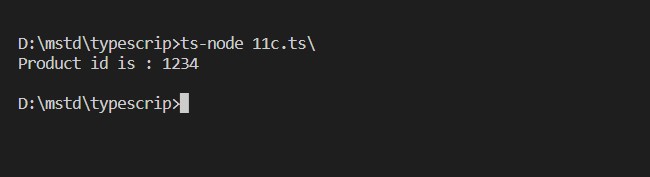
return 'Product id is : ' + this.productId;

}

}

const product: Product = new Product(1234); console.log(product.getProductId());

**Output:**



**11.d Course Name: Typescript**

**Module Name: Access Modifiers**

**Create a Product class with 4 properties namely productId, productName, productPrice, productCategory with private, public, static, and protected access modifiers and accessing them through Gadget class and its methods.**

**Program:**

class Product { static productPrice = 150; private productId: number; public productName: string; protected productCategory: string;

constructor(productId: number, productName:string , productCategory:string) { this.productId = productId; this.productName = productName; this.productCategory = productCategory;

}

getProductId() { console.log('The Product id is : ' + this.productId);

}

}

class Gadget extends Product { getProduct(): void {

console.log('Product category is : ' + this.productCategory);

}

}

const g: Gadget = new Gadget(1234, 'Mobile', 'SmartPhone'); g.getProduct();

g.getProductId(); console.log('Product name is : ' + g.productName); console.log('Product price is : $' + Product.productPrice); **Output:**



**12.a Course Name: Typescript**

**Module Name: Properties and Methods**

**Create a Product class with 4 properties namely productid and methods to setProductId() and getProductId().**

**Program:** class Product {

public productPrice = 150; private productId: number=0; public productName: string="Iphone"; public productCategory: string="Mobile";

set setProductId(productId:number){ this.productId=productId;

}

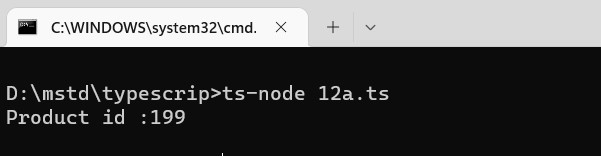
get getProductId():string{

return 'Product id :'+this.productId;

}

}

const ProductDetails:Product=new Product(); ProductDetails.setProductId=199; console.log(ProductDetails.getProductId); **Output:**



**12.b Course Name: Typescript**

**Module Name: Creating and using Namespaces**

**Create a namespace called ProductUtility and place the Product class definition in it. Import the Product class inside productlist file and use it.**

**Program:**

**Utility.ts** namespace ProductUtility{ export class Product1 { static productPrice: string; productId: number;

constructor(productId: number) { this.productId = productId;

}

getProductId(): string {

return 'Product id is : ' + this.productId;

}}}

# 12b.ts

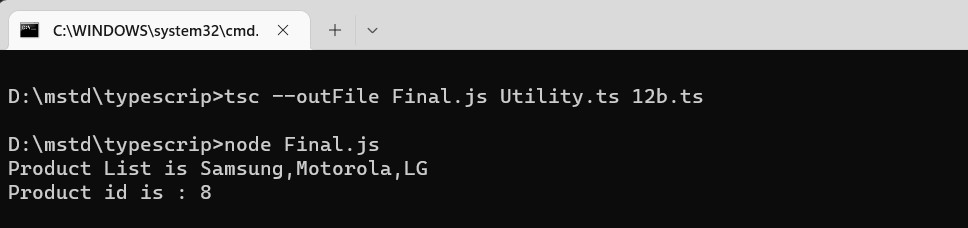
///<reference path="./Utility.ts"/> import prod=ProductUtility; let pe=new prod.Product1(8); interface ProductList { list: Array<string>;

}

const productDetails: ProductList= { list: ['Samsung', 'Motorola', 'LG']

};

const listProduct = productDetails.list; console.log('Product List is ' + listProduct); console.log(pe.getProductId()) **Output:**



**12.c Course Name: Typescript**

**Module Name: Creating and using Modules**

**Consider the Mobile Cart application which is designed as part of the functions in a module to calculate the total price of the product using the quantity and price values and assign it to a totalPrice variable.**

**Program:**

**Module.ts**

class Utility {

CalculateAmount(price: number, quantity: number): number { return price \* quantity;

}

}

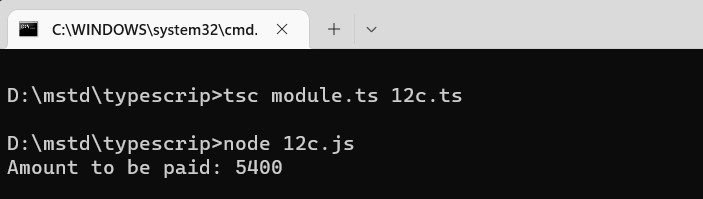
export{Utility};

# 12c.ts

import { Utility as mainUtility} from "./module"; const util = new mainUtility();

const price = util.CalculateAmount(1350, 4); console.log(`Amount to be paid: ${price}`);

**Output:**



**12.d Course Name: Typescript**

**Module Name: What is Generics, What are Type Parameters, Generic Functions, Generic Constraints Create a generic array and function to sort numbers as well as string values**

**Program:**

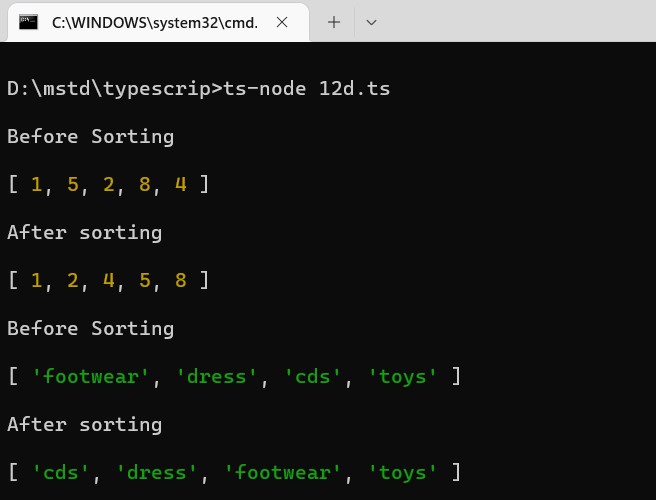
function sortArray<T>(arg: Array<T>): Array<T> { return arg.sort();

}

const numbers: Array<number> = [1,5,2,8,4];

const strings: Array<string> = ['footwear', 'dress', 'cds', 'toys']; console.log('\nBefore Sorting\n'); console.log(numbers); const idList = sortArray(numbers); console.log('\nAfter sorting\n'); console.log(idList);

console.log('\nBefore Sorting\n'); console.log(strings); console.log('\nAfter sorting\n'); const nameList = sortArray(strings); console.log(nameList); **Output:**



**11.b Course Name: Typescript**

**Module Name: Classes**

**Consider the Mobile Cart application, Create objects of the Product class and place them into the productlist array.**

**Program:**

class Product { productId: number; productName:string;

constructor(productId: number,productName:string) { this.productId = productId; this.productName=productName;

}

getDetails(): string {

return 'Product id is : ' + this.productId+'\n Product name is :'+this.productName;

}

}

const ar:Product[]=[]; let i:number=1; for(i=1;i<=4;i++)

{

ar.push(new Product(i,"str"+i));

}

console.log(ar);

**Output:**

