

8.a Course Name: Express.js**Module Name: CRUD Operations**

Write a program to perform various CRUD (Create-Read-Update-Delete) 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.

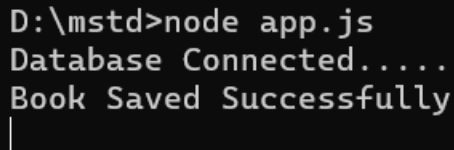
Program:**Create:**

```
const express=require('express')
const mongoose=require('mongoose')
const url="mongodb://0.0.0.0:27017/Hell";
mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).then(=>console.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;
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");  
});
```



```
D:\mstd>node app.js  
Database Connected.....  
Book Saved Successfully  
|
```

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}).then(()=>console.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;  
var queryBooks = function(){  
  
    Book.find( function(err, result){
```



```
if ( err )  
  
    throw err;  
  
console.log("Find Operations: " + result);  
  
});  
  
}  
  
queryBooks();
```

Output:

```
Database Connected....  
Find Operations: {  
  _id: new ObjectId("636cf1bb7a291be85ed8067d"),  
  name: 'Mongoose Demo 1',  
  isbn: 'MNG123',  
  author: 'Author1, Author2',  
  pages: 123,  
  __v: 0  
}, {  
  _id: new ObjectId("636cf36e1e0e614dbb5a0089"),  
  name: 'Mongoose Demo 1',  
  isbn: 'MNG123',  
  author: 'Author1, Author2',  
  pages: 123,  
  __v: 0  
}, {  
  _id: new ObjectId("636cf473e450b49c8bc26843"),  
  name: 'Mongoose Demo 1',  
  isbn: 'MNG123',  
  author: 'Author1, Author2',  
  pages: 123,  
  __v: 0  
}, {  
  _id: new ObjectId("636cf473e450b49c8bc26844"),  
  name: 'JAvA',  
  isbn: 'Hello',  
  author: 'kello',  
  pages: 123,  
  __v: 0  
}, {  
  _id: new ObjectId("636cf4a59b74e217409d7b0b"),  
  name: 'JAvA',  
  isbn: 'Hello',  
  author: 'kello',  
  pages: 123,  
  __v: 0  
}  
}
```

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}).then(()=>console.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;

var updateBook = function(){
    Book.updateOne({ $name: "JAvA" }, { $set: { name: "JAVA" } }, function(err, result){
        console.log("Updated successfully");
        console.log(result);
    });
}updateBook();
```

Output:

```
C:\WINDOWS\system32\cmd. x + v

D:\mstd>node app.js
Database Connected.....
Updated successfully
{
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
```

Delete

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

const mongoose=require('mongoose')

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

mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).then(=>console.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;

var deleteBook = function(){

Book.deleteMany({name:"JAvA"},function(err,result){if(err)

    console.log(err);

    else

    console.log("deleted"))}).exec(); }

deleteBook();
```

Afterdeletong records Database is:

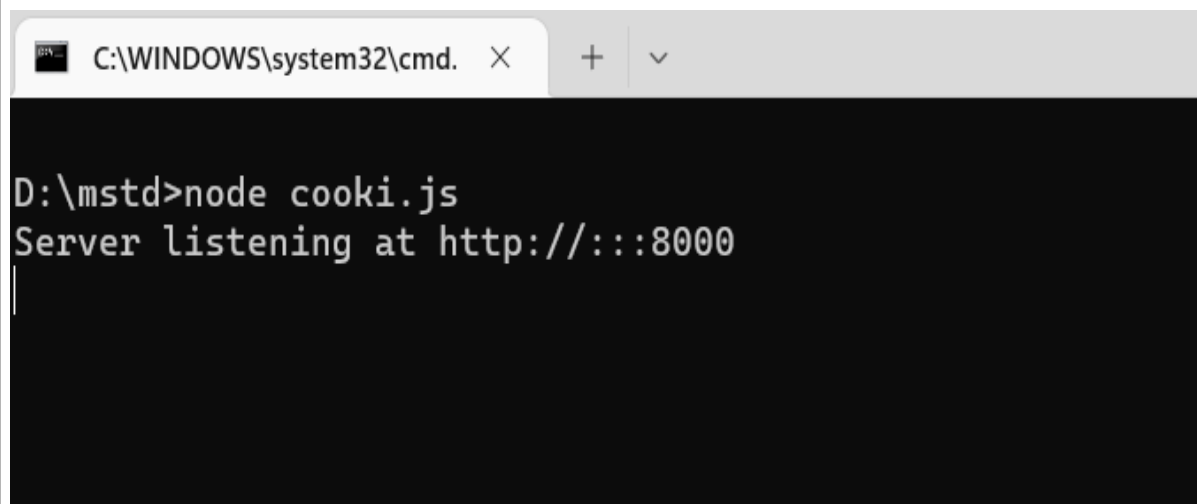
```
C:\WINDOWS\system32\cmd.  X  +  v

D:\mstd>node app.js
Database Connected.....
Find Operations: {
  _id: new ObjectId("636cf1bb7a291be85ed8067d"),
  name: 'JAVA',
  isbn: 'MNG123',
  author: 'Author1, Author2',
  pages: 123,
  __v: 0
}, {
  _id: new ObjectId("636cf36e1e0e614dbb5a0089"),
  name: 'JAVA',
  isbn: 'MNG3',
  author: 'Author1, Author2',
  pages: 123,
  __v: 0
}, {
  _id: new ObjectId("636cf473e450b49c8bc26843"),
  name: 'JAVA SCRIPT',
  isbn: 'MNG897',
  author: 'Author1, Author2',
  pages: 123,
  __v: 0
}
|
```

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.cookies);
});
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:

```
C:\WINDOWS\system32\cmd.  X  +  v
D:\mstd>node cooki.js
Server listening at http://:::8000
```

127.0.0.1:8000

x

+

←

→

↻

127.0.0.1:8000

Welcome to Aditya

127.0.0.1:8000/cookieset

x

+

←

→

↻

127.0.0.1:8000/cookieset

Cookie is set

127.0.0.1:8000/cookieget

x

+

←

→

↻

127.0.0.1:8000/cookieget

```
{"cookie_name": "cookie_value", "College": "Aditya", "Branch": "Cse"}
```

**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('express-session')

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)

})

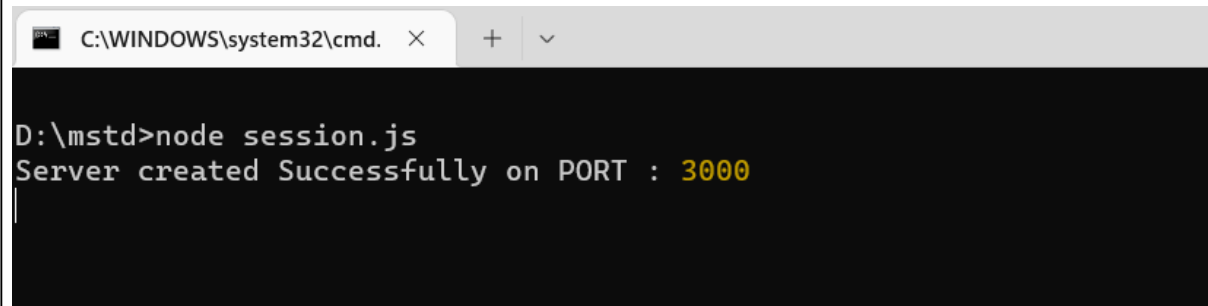
app.listen(PORT, function(error){

if(error) throw error

console.log("Server created Successfully on PORT :", PORT)

})
```


Output:



```
C:\WINDOWS\system32\cmd. x + v

D:\mstd>node session.js
Server created Successfully on PORT : 3000
```



```
127.0.0.1:3000/session x +

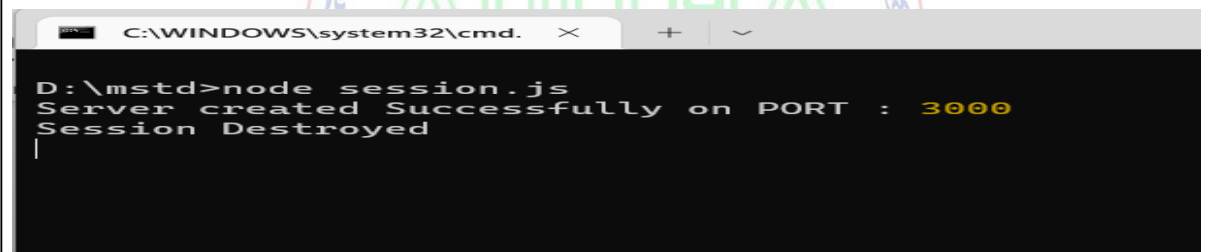
127.0.0.1:3000/session

Sessionname:alr

127.0.0.1:3000/session x +

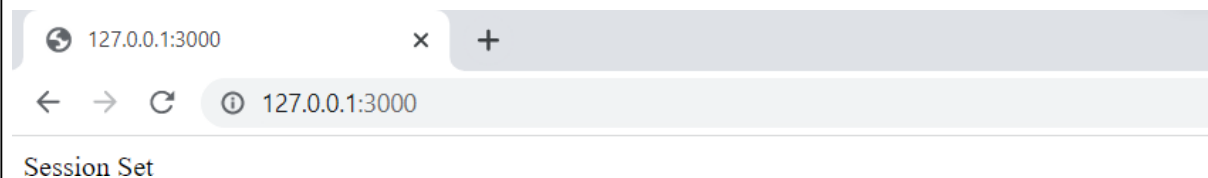
127.0.0.1:3000/session

{"cookie":{"originalMaxAge":null,"expires":null,"httpOnly":true,"path":"/"},"name":"Sessionname:alr"}
```



```
C:\WINDOWS\system32\cmd. x + v

D:\mstd>node session.js
Server created Successfully on PORT : 3000
Session Destroyed
```



```
127.0.0.1:3000 x +

127.0.0.1:3000

Session Set
```

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

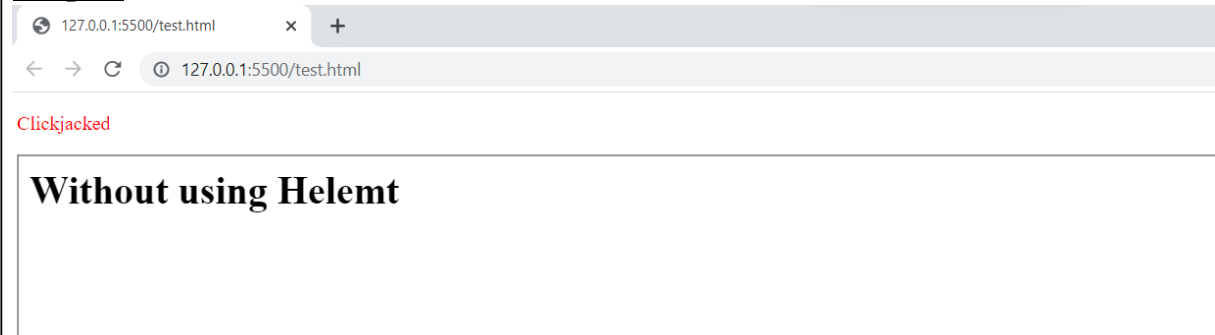
```
const express = require('express');
const router = express.Router();
router.get('/', function (req, res) {
  res.send('<h1>Express</h1>');
});
router.get('/about', function (req, res) {
  res.send('About Us Page');
});
module.exports = router;
```

test.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <style>
    p {
      color: red;
    }
    iframe {
      width: 100%;
      height: 90%
    }
  </style>
</head>
```

```
</style>
</head>
<body>
  <p>Clickjacked</p>
  <iframe src="http://localhost:3000"></iframe>
</body>
</html>
```

Output:



Without using Helmet

Implementing Helmet

App.js

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

```
const obj:{ GoldPlatinum: 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:

```
D:\mstd\typescrip>ts-node 9a.ts
```

```
Mobilecolor Price
```

```
GoldPlatinum:    $1000
```

```
PinkGold:        $900
```

```
SilverTitanium:  $1500
```



Exp No:

Date:

Page No:

9.b) Define an arrow function inside the event handler to filter the product array with the selected product object using the productId received by the function. Pass the selected product object to the next screen.

Program:

```
var getProductDetails=(productId : number):string=>{return "product Id:"+productId};  
console.log(getProductDetails(1234));
```

Output:

```
D:\mstd\typescrip>ts-node 9b.ts  
product Id: 1234  
  
D:\mstd\typescrip>
```



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:

```
D:\mstd\typescrip>ts-node 9c.ts  
The available Samsung mobile list: [Samsung Galaxy S6 Edge,Samsung Galaxy Note 7,Samsung Galaxy J7 SM-J700F]  
  
The available Iphone mobile list: [Apple iPhone 5s,Apple iPhone 6s ,Apple iPhone 7]  
D:\mstd\typescrip>
```

9d)Course Name: Typescript**Module Name: Arrow Function Consider that**

developer needs 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.

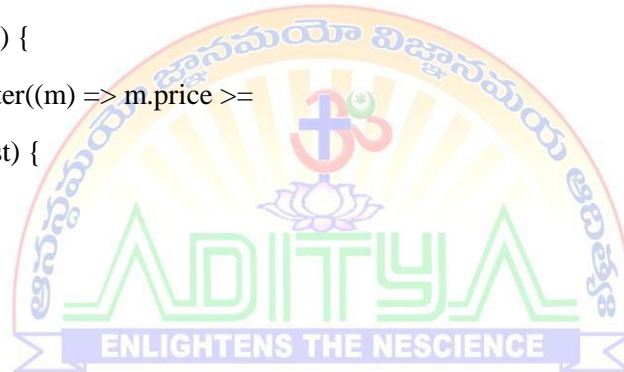
Program:

```
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() {
test = manufacturers.filter((m) => m.price >=
150);for (var item of test) {
console.log(item.id);
}
}

myFunction();
```

**output:**

```
D:\mstd\typescrip>ts-node 9d.ts
Details of Manufacturer array are :
Samsung
Microsoft
Apple

D:\mstd\typescrip>
```

9e)Course Name: Typescript**Module Name: Optional and Default Parameters Declarea**

function - getMobileByManufacturer with two parameters namely manufacturer and id, where manufacturer value should be 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 list 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 getMobileByManufacturer(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))
```

Output:

```
C:\Users\admin\Pictures>ts-node exp9e.ts
The available mobile list : Apple iPhone 5s, Apple iPhone 6s, Apple iPhone 7
The available mobile list : Moto G Play, 4th Gen,Moto Z Play with Style Mod
```

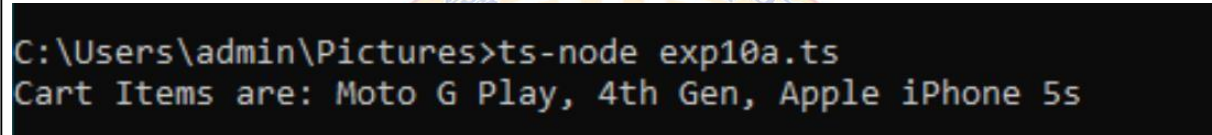

10.a Course Name: Typescript

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:



```
C:\Users\admin\Pictures>ts-node exp10a.ts  
Cart Items are: Moto G Play, 4th Gen, Apple iPhone 5s
```



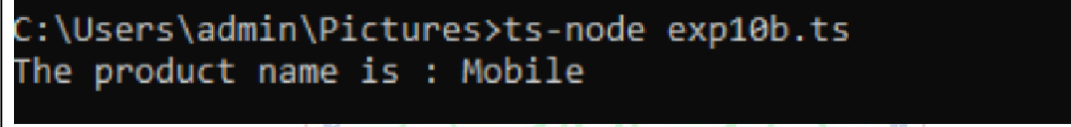
10.b Course Name: Typescript

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:



```
C:\Users\admin\Pictures>ts-node exp10b.ts  
The product name is : Mobile
```

ENLIGHTENS THE NESCIENCE

10.c Course Name: Typescript

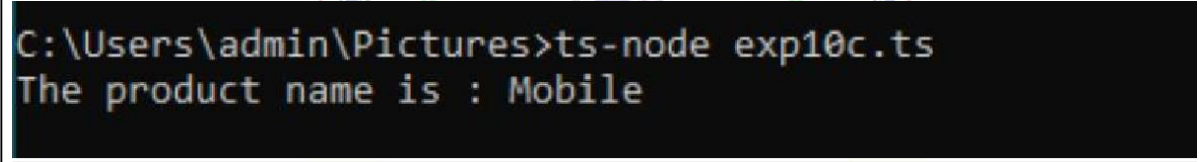
Module Name: Duck Typing Declare an interface named

- 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: number;  
  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:



```
C:\Users\admin\Pictures>ts-node exp10c.ts  
The product name is : Mobile
```

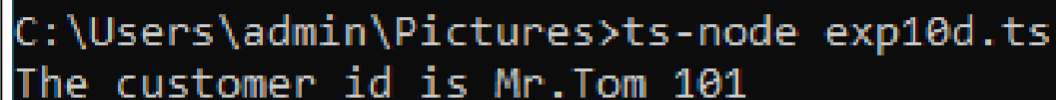
10.d Course Name: Typescript

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:



```
C:\Users\admin\Pictures>ts-node exp10d.ts  
The customer id is Mr.Tom 101
```

ENLIGHTENS THE NESCIENCE