

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.

Aim: To consider the Mobile Cart application, Create objects of the Product class and place them into the productlist array.

Description: TypeScript is object oriented JavaScript. TypeScript supports object-oriented programming features like classes, interfaces, etc. A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object. Typescript gives built in support for this concept called class. JavaScript ES5 or earlier didn't support classes. Typescript gets this feature from ES6. Use the class keyword to declare a class in TypeScript. The syntax for the same is given below -

```
class class_name {  
    //class scope  
}
```

Program:

```
class Product {  
    static productPrice: string;  
    productId: number;  
    constructor()  
    { this.productId =1234;  
    }  
    getProductId(): string {  
        return 'Product id is : ' + this.productId;  
    }  
}  
const product: Product = new Product();  
const p={  
    producti :product.getProductId(), };  
console.log(p.producti);
```

Output:

```
C:\Users\91879\Pictures>ts-node exp11b.ts  
Product id is : 1234
```

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 <>"

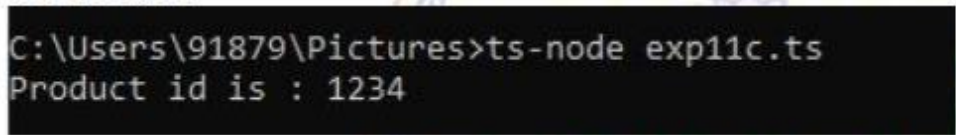
Aim: To declare a class named - Product with the below-mentioned declarations: (i) productId as number property (ii) Constructor to initialize this value (iii) getProductId method to return the message "Product id is <>" .

Description: A constructor is a special function of the class that is automatically invoked when we create an instance of the class in Typescript. We use it to initialize the properties of the current instance of the class. Using the constructor parameter properties or Parameter shorthand syntax, we can add new properties to the class. We can also create multiple constructors using the technique of constructor method overload. The constructor method in a class must have the name constructor. A class can have only one implementation of the constructor method. The constructor method is invoked every time we create an instance from the class using the new operator. It always returns the newly created object.

Program:

```
class Product {  
  static productPrice: string;  
  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:



```
C:\Users\91879\Pictures>ts-node exp11c.ts  
Product id is : 1234
```

Date:

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.

Aim: To 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.

Description: Like other programming languages, Typescript allows us to use access modifiers at the class level. It gives direct access control to the class member. These class members are functions and properties. We can use class members inside its own class, anywhere outside the class, or within its child or derived class. The access modifier increases the security of the class members and prevents them from invalid use. We can also use it to control the visibility of data members of a class. If the class does not have to be set any access modifier, TypeScript automatically sets public access modifier to all class members.

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:

```
C:\Users\91879\Pictures>ts-node exp11d.ts
Product category is : SmartPhone
The Product id is : 1234
Product name is : Mobile
Product price is : $150
```

**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()**.

Aim: To create a Product class with 4 properties namely productId and methods to setProductId() and getProductId().

Description: In typescript, the method is a piece of code that has been declared within the class and it can be carried out when it is called. Method property in it can split a huge task into little sections and then execute the particular operation of that program so that code can be reusable which can improve the module from the program.

Program:

```
// declaring a Product class
class Product {
  static productPrice: string;
  productId: number;
  // constructor declaration
  constructor(productId: number) {
    this.productId = productId;
  }
  getProductId(): string {
    return 'Product id is : ' + this.productId;
  }
}
// creation of Product class object
const product: Product = new Product(2345);
// line to populate the product id details
console.log(product.getProductId());
```

Output:

```
C:\Users\91879\Pictures>ts-node exp12a.ts
Product id is : 2345
```

**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.**

Aim: To create a namespace called **ProductUtility** and place the **Product** class definition in it. **Import the Product class inside productlist file and use it.**

Description: In typescript, the method is a piece of code that has been declared within the class and it can be carried out when it is called. Method property in it can split a huge task into little sections and then execute the particular operation of that program so that code can be reusable which can improve the module from the program. The classes or interfaces which should be accessed outside the namespace should be marked with keyword **export**. To access the class or interface in another namespace, the syntax will be `namespaceName.className`

Program:

```
namespace_one12b.ts:
import util = Utility.Payment;
let paymentAmount = util.CalculateAmount(1800, 6);
console.log(`Amount to be paid: ${paymentAmount}`);
namespace_two12b.ts:
namespace Utility {
export namespace Payment {
export function CalculateAmount(price: number, quantity:
number): number
{return price * quantity;
}
}
}
```

Output:

```
C:\Users\91879\Pictures>tsc --outFile result.js namespace_two12b.ts namespace_one12b.ts
C:\Users\91879\Pictures>node result.js
Amount to be paid: 10800
```




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.

Aim: To creating and using Modules Consider the Mobile Cart application.

Description: A module refers to a set of standardized parts or independent units that can be used to construct a more complex structure. TypeScript modules provides a way to organize the code for better reuse.

```
export interface InterfaceName {  
    //Block of statements
```

```
}
```

Program:

```
module_one12c.ts:
```

```
export function MaxDiscountAllowed(noOfProduct: number):  
number
```

```
{if (noOfProduct > 5) {  
return 30;
```

```
} else {
```

```
return 10;
```

```
}
```

```
}
```

```
class Utility {
```

```
CalculateAmount(price: number, quantity: number): number
```

```
{return price * quantity;
```

```
}
```

```
}
```

```
interface Category
```

```
{ getCategory(productId: number):
```

```
string;
```

```
}
```

```
export const productName = 'Mobile';
```

```
export {Utility, Category};
```

```
module_two12c.ts:
```

```
import { Utility as mainUtility, Category, productName,  
MaxDiscountAllowed } from
```

```
"./module_one12c";
```

```
const util = new mainUtility();
```

```
const price = util.CalculateAmount(1350, 4);
```

```
const discount = MaxDiscountAllowed(2);
```

```
console.log(`Maximum discount allowed is: ${discount}`);
```

```
console.log(`Amount to be paid: ${price}`);
```

```
console.log(`ProductName is: ${productName}`);
```

Output:

```
C:\Users\91879\Pictures>tsc module_one12c.ts module_two12c.ts
```

```
C:\Users\91879\Pictures>node module_two12c.js
```

```
Maximum discount allowed is: 10
```

```
Amount to be paid: 5400
```

```
ProductName is: Mobile
```



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.

Aim: To create a generic array and function to sort numbers as well as string values.

Description: Whenever any program or code is written or executed, one major thing one always takes care of which is nothing but making **reusable components** which further ensures the scalability and flexibility of the program or the code for a long time.

Generics, thus here comes into the picture as it provides a user to flexibly write the code of any particular data type (or return type) and that the time of calling that user could pass on the data type or the return type specifically.

Generics provides a way to make the components work with any of the data types (or return types) at the time of calling it for a certain number of parameters (or arguments).

In generics, we pass a parameter called **type parameter** which is put in between the lesser sign (<) and the greater sign (>), for example, it should be like **<type_parameter_name>**.

Program:

```
// declaring a Generic Array named orderDetails
function orderDetails<T>(arg: Array<T>): Array<T> {
  console.log(arg.length);
  return arg;
}
// creating a variable to hold a number array
const orderid: Array<number> = [201, 202, 203, 204];
// creating a variable to hold a string array
const ordername: Array<string> = ['Dresses', 'Toys',
  'Footwear', 'cds'];
// creating a variable to hold result of orderDetails function
with a number array as
parameter
const idList = orderDetails(orderid);
// line to populate the result of line no 14
console.log(idList);
// creating a variable to hold result of orderDetails function
with a string array as parameter
const nameList = orderDetails(ordername);
// line to populate the result of line no 20
console.log(nameList);
```

Output:

```
C:\Users\91879\Pictures>ts-node exp12d.ts
4
[ 201, 202, 203, 204 ]
4
[ 'Dresses', 'Toys', 'Footwear', 'cds' ]
```

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.

Aim: To declare a productList interface which extends properties from two other declared interfaces like Category

Description:

An interface can be extended from an already existing one using the extends keyword.

In the code below, extend the productList interface from both the Category interface and Product interface.

Example:

```
interface Category{
    categoryName:string;
}
interface Product{
    productName:string;
    productid:number;
}
interface productList extends Category,Product{
    list:['Samsung','Motorola','LG']
}
```

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:

```
C:\Users\durga\OneDrive\Desktop\New folder\5th sem\typescript>ts-node 11a.ts
Product Name is Mobile
Product List is Samsung,Motorola,LG
```