**TypeScript Basics:**

1) Create a Customer class. Add following members to it:

a) customerId (numeric)

b) firstName (alphabetic)

c) lastName (alphabetic)

d) contactNo (numeric)

e) email (alphanumeric)

f) isPrevilaged (boolean)

2) Create a CustomerOperations class. Create an array of customers in CustomerOperations class. Add following functions to CustomerOperations class:

a) Function to add a customer to an array

b) Function to retrieve all the customers from an array

Execute the above functions and display the output.

3) Create a Tuple named contactDetails. This tuple should contain contactNo and emailId. Create a variable of type contactDetails, assign values to contactNo and emailId and display those values.

4) Create an array of type any. Add few numbers, strings and booleans to this array. Display the array members.

5) Create an Enum as follows:

enum PrintMedia {

Newspaper = 1,

Newsletter,

Magazine,

Book

}

Write a function which accepts a parameter of type String and return the value of type PrintMedia.

E.g. – If ReadersDigest is passed as a parameter to the function, then function should return Magazine.

**Arrow Functions:**

1) Add an arrow function to the Customer class. This arrow function should display firstName, lastName and contactNo of the customer. Create an object of the Customer class and call the arrow function.

**Default Function Params:**

1) Create a function that accepts price and discount. The function should calculate the price after deducting the discount and return that discounted price. Discount should be 5 percent if user does not specify the discount.

**REST and SPREAD Operators:**

1) Create a function named multiply. You should be able to call multiply function using variable number of arguments. Call the multiply function once passing 3 numbers and once passing 4 numbers.

**DESTRUCTURING:**

1) Create an object as follows:

const note = {

    id: 1,

    title: 'My first note',

    date: '01/01/1970',

}

Using destructuring, assign the note object to another object. Display id, title and date.

2) Create an array as follows:

const doj = ['2005', '12', '01'];

Use destructuring to assign the doj array to another array. Display year, month and date.

**OOPS:**

**Assignment 1:**

1) Create an Employee class with following members:

a) empCode (numeric)

b) firstName (alphabetic)

c) lastName (alphabetic)

d) salary (numeric)

2) Write a constructor in Employee class. The constructor should accept firstName, lastName and salary from the user.

3) Write a getter and setter for empCode in Employee class. The setter should throw an error if the empCode is negative or greater than 1000.

4) Write a function in Employee class that returns salary of an employee.

5) Create an object an Employee class. Assign the value to the empCode using setter.

6) Display the values assigned to the members of the Employee class.

7) Call the function that returns salary of an Employee. Display the salary of an employee.

**Assignment 2:**

1) Create a Car class with following members:

a) name (alphanumeric)

b) a method named run. This method should accept a parameter named speed. Default value of speed should be 0. The method should display the car name and speed.

2) Inherit two classes from Car class. These classes should inherit the car name from the base class Car. Change the implementation of run method in these two classes. First class should assign the value of 150 to the speed. Second class should assign the value of 100 to the speed. Call the run method of Car class from the run method in both these classes.

**Assignment 3:**

1) Create an abstract class Person with following members:

 name: string;

    constructor(name: string) {

        this.name = name;

    }

    display(): void{

        console.log(this.name);

    }

2) Add an abstract method named find in the Person class. find method should accept personName as string and return the Person object.

3) Inherit Employee class from Person class. Employee class should have following members: with following members

a) empCode (numeric)

b) empName (alphabetic)

Inherit empName from name in the Person class.

4) Employee class should implement find method.

5) Create an object of Employee class and call the find method.

**Assignment 4:**

1) Create an interface named IPerson with following members:

a) name (alphabetic)

b) method named display which return void.

2) Create an interface named IEmployee with following members:

a) empCode (numeric)

3) Add an Employee class. Employee class should implement IPerson and IEmployee interfaces. display method should display empCode and name.

4) Call the display method.

**Assignment 5:**

1) Create two functions named add as follows:

The first function should accept two numeric parameters and return their addition.

The second function should accept two string parameters, concatenate those two parameters and return the concatenated value.

2) Call both the add functions and display the output.

**Generics:**

**Assignment 1:**

1) Create a function. This function should accept an array of type T. Function should concatenate the items in an array and return it.

2) Call getArray function by passing it a numeric array of 3 integers. Add one more integer to this array. Print the numeric array.

3) Call getArray function by passing it a numeric array of 2 strings. Add one more string to this array. Print the string array.

**Modules and Namespaces:**

**Assignment 1:**

1) Create an HRMS module.

2) Add an Employee class with following members to this HRMS module:

a) empCode (numeric)

b) firstName (alphabetic)

c) lastName (alphabetic)

d) a function named displayEmployee which prints details of an employee.

3) Create an EmployeeProcessor class. Create an object of Employee class in EmployeeProcessor class and call displayEmployee function with this object.

**Decorators:**

**Assignment 1:**

1) Create a custom decorator. This decorator should seal both the constructor and its prototype.

2) Apply this custom decorator on a class.

**INTERFACES**

1. Create an interface named **IFly** which has a **Fly()** method. The method should not accept any arguments, but must return a string.
2. Create 3 classes named **Sparrow, Superman** & **Missile** respectively. Each of these classes must implement the **IFly** interface. The implementation of the **Fly()** method must return a string from each class saying **xxx can fly** *(xxx here means the entity).*
3. Create a function named **FlySimulator()**. The method must internally call the **Fly()** method of any of the 3 classes depending on the argument passed to it.
4. Create 4 functions named **Add(), Subtract(), Multiply()** & **Divide()** respectively.
5. Each function must take 2 arguments and return the result of the appropriate mathematical calculation.
6. Create a function named **Compute()** which takes any of these functions as an argument and invokes the appropriate function passed to it. Use an interface to strongly type the function’s signature.

**OOP**

1. Create a class named **Account** which has a constructor which accepts the following arguments:
   1. Account id 🡪 number
   2. Name 🡪 string
   3. Initial balance 🡪 number
2. All these must be optional arguments and must have a default value if not passed.
3. Create the following public methods inside the class:
   1. Withdraw() 🡪 accept a number & perform a debit operation and return the

updated balance

* 1. Deposit() 🡪 accept a number & perform a credit operation and return the

updated balance

1. Create a private method named **Log()** which accepts a string and displays the string on the console. The **Withdraw()** & **Deposit()** functions must call the this method with a string representing the operation before the balance is returned.
2. Create an array of 3 **Account** objects. For one of the objects, do not pass any arguments to the constructor.
3. Iterate the array and call the **Withdraw()** & **Deposit()** methods to check if it displays the balance correctly or not for each object.
4. Add a private static member named **account\_count** of type number.
5. Create a static method named **CountAccounts()** which returns the number of **Account** objects in the memory.
6. Create public properties for the data members **account id, name** & **initial balance** respectively inside the **Account** class.
7. Create an abstract class named **Sorter** which has an abstract method named **Sort()** inside it. The method returns a string, but does not accept any argument.
8. Create 2 concrete classes named **Linear** & **Bubble** respectively which inherit the abstract class & provide an implementation to the **Sort()** method. Each implementation must return a string which says **xxx sort implemented.** *(xxx means the type of sorting implemented)*.
9. Create an array of 2 **Sorter** objects. Iterate the array and invoke the **Sort()** method.

**MODULES**

1. Put the interface named **IFly** created earlier in a module named **InterfaceModule** and export it.
2. Put the classes named **Sparrow, Superman** & **Missile** created earlier inside a module named **ClassesModule** and export them.
3. Consume them from another module named **AppModule**.
4. Change the type of export in step1 to a default export and refactor the code accordingly.