SYNERGETICS
GET IT RIGHT

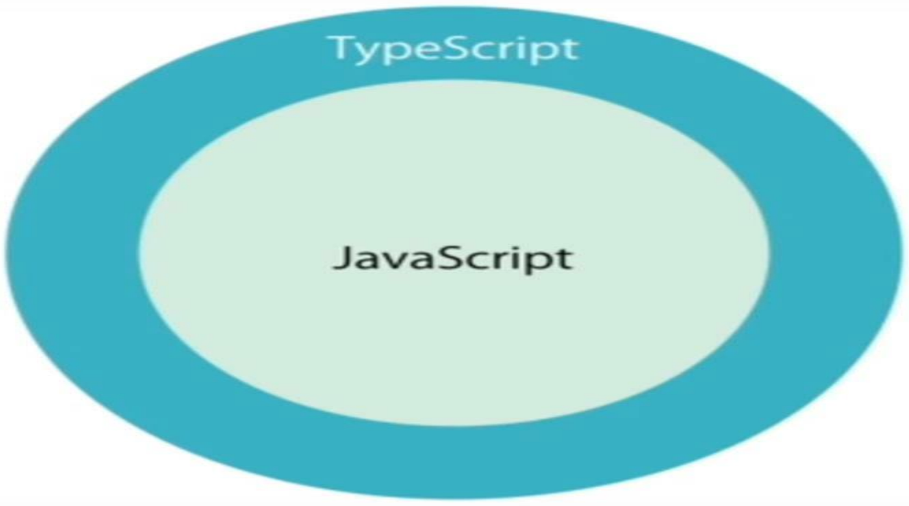
Type Script

Lesson Objectives

- Introduction to Typescript
- JavaScript & Typescript
- The type system-Variable, Array
- Defining class and interface
- Arrow Functions
- Template Strings
- Defining a module
- Importing a module
- Generics

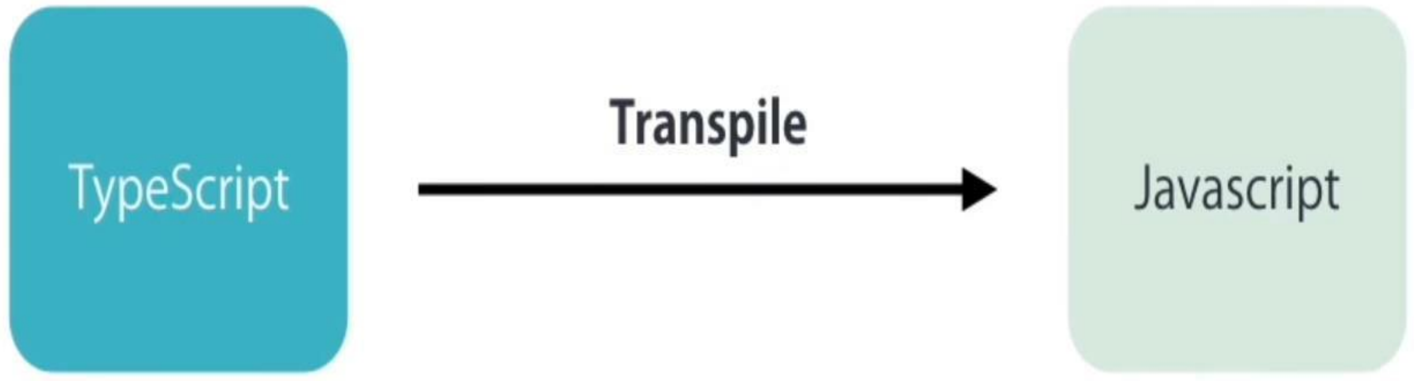
TypeScript

- **TypeScript** is an open-source programming language developed and maintained by Microsoft.
- It is superset of JavaScript.
- It is a strict syntactical superset of JavaScript, and adds optional static typing to the language.
- Anders Hejlsberg, lead architect of C# and creator of Delphi & Pascal, has worked on the development of TypeScript.
- TypeScript may be used to develop JavaScript applications for client-side or server-side -Node.js execution.



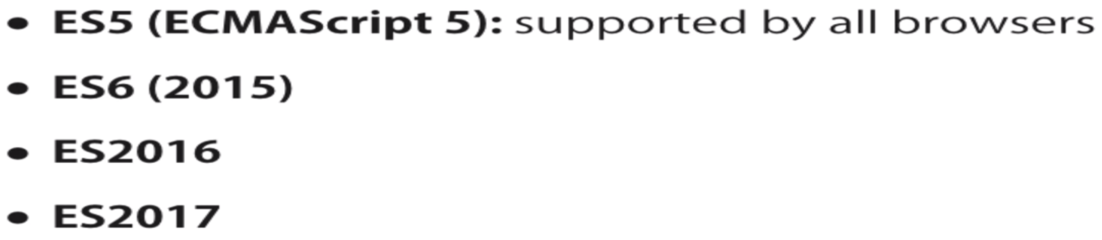
TypeScript

- Strong Typing
- Object Oriented features
- Compile time catching error
- Browser don’t understand typescript so we need to compile or trans pile into JavaScript. So Browser can understand it



Why TypeScript

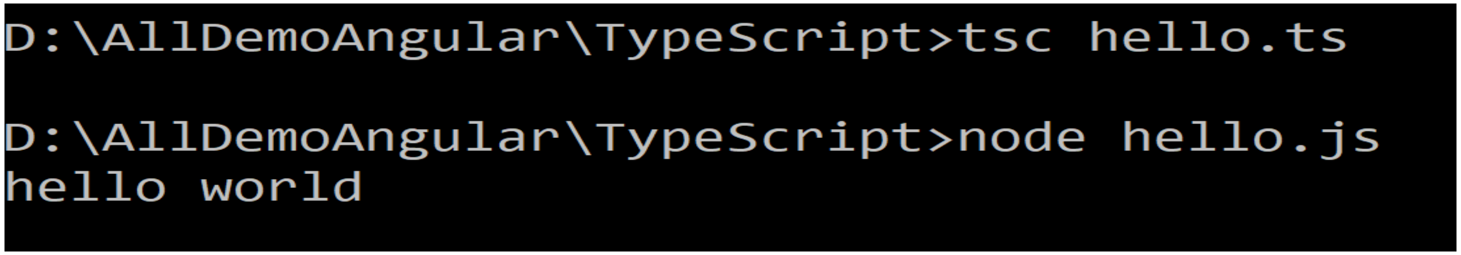
- TypeScript can be used for cross browser development and is an open source project.
- Using TypeScript developers can apply class-based approach, compile them down to JavaScript without waiting for the next version of JavaScript.
- With TypeScript existing JavaScript code can be easily incorporated with popular JavaScript libraries like jQuery, Backbone, Angular and so on.
- Enable scalable application development with optional Static types, classes and modules. Static types completely disappear at runtime.
- TypeScript converts JavaScript Programming from loosely typed to strongly typed.
- JavaScript Version:



- **ES5 (ECMAScript 5):** supported by all browsers
- **ES6 (2015)**
- **ES2016**
- **ES2017**

Installing TypeScript

- ➢ First Install Node
  - Via npm (the Node.js package manager)--npm install -g typescript
  - Or DownLoad typescript compiler –master & set in class path & work
  - https://www.typescriptlang.org/play/ ---- work online
- ➢ Open Eclipse
  - Create Typescript project name as ‘hello.ts’
  - Open command prompt & redirect to that eclipse folder
- ➢ For NPM users & use typescript without downloading
  - npm install -g typescript
  - At the command line, run the TypeScript compiler:
  - tsc hello.ts
  - When we write tsc hello.ts it will convert into js
  - Then write node hello.js

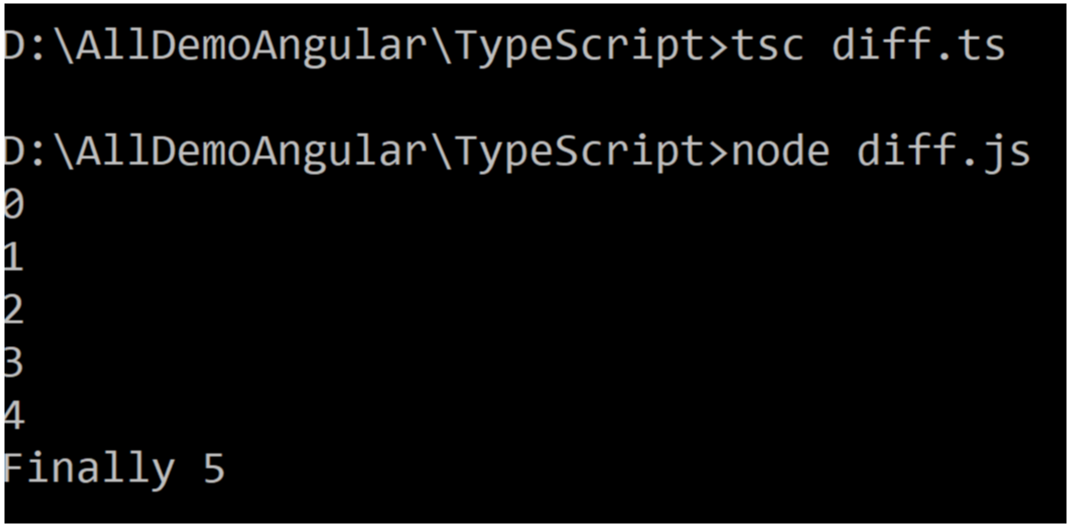


```
D:\AllDemoAngular\TypeScript>tsc hello.ts

D:\AllDemoAngular\TypeScript>node hello.js
hello world
```

Difference between let & var

- Using var

```
function doGet()
{
   for(var i = 0; i < 5; i++)
   {
      console.log(i);
   }
   console.log(“Finally ” + i);
}

doGet();
```



```
D:\AllDemoAngular\TypeScript>tsc diff.ts

D:\AllDemoAngular\TypeScript>node diff.js
0
1
2
3
4
Finally 5
```

- Now if you use ‘let’ instead of ‘var’ it will give compilation error, because scope is limited
- So in typescript we have to use ‘let’ instead of ‘var’

Type Annotations

- Type annotations in TypeScript are lightweight ways to record the intended contract of the function or variable.

```
let empId: number;                                   --- number
let empName: string;                                 --- string
let empFeedback: boolean;                            --- Boolean
let anyType: any;                                    --- any
let myArray: number[] = [1, 2, 3];                   --- number
let anyArrayType: any[] = [1, ‘Zara’, false, true];  --- any array
```

Type Annotations

- Enum & Constant

```
const colored = 0;
const colorBlue = 1;
const colorGreen = 2;

enum Color { Red = 0, Green = 1, Blue = 2 };

let backgroundColor = Color.Red;

console.log(backgroundColor);
```

Type Assertion in TypeScript

- TypeScript allows changing a variable from one type to another.
- TypeScript refers to this process as *Type Assertion*.
- The syntax is to put the target type between < > symbols and place it in front of the variable or expression.

```
let str: string;

str.substring(2,3);


let str2;

(<string>str2).length;
(str2 as string).length;
```

Assertion in TypeScript

Function:

➢ Creating Functions

```
//2 parameter with number as return type
function getsum(numOne: number, numTwo: number): number{
    return numOne + numTwo;
}

let add = getsum(10,6);
console.log("Sum is " + add );
```

```
//any number of data--know as rest parameter
function sumAll(...num: number[]){
    let sum: number = 0;
    for (let data of num) {
        sum = sum + data;
        console.log("Addition of number " + data);
}
    console.log("Sum is " + sum);
}

sumAll(6, 7, 8, 9);
```

Optional, Default

- ? Is know as optional parameter

```
//Optional parameter----? for optional & Default parameter
function doGet(one: number, two = 5, three?: number): void{
    //alert("hii");
    console.log(one.toString());
    console.log(two.toString());
    console.log(three.toString());
}

//doGet(10);
doGet(10);
```

Arrow Functions

- => is a and also called a Arrow function

```
let log = function(message)
{
    console.log('Welcome to Arrow');
}

//Arrow function equivalent to above function
let doLog = (message) => console.log(message);

//Arrow function equivalent to no parameter  function
let withoutparameter = () => console.log();
```

Interfaces

- TypeScript is object oriented JavaScript. TypeScript supports object-oriented programming features like classes, interfaces, etc.
- An interface is a syntactical contract that an entity should conform to.
- In other words, an interface defines the syntax that any entity must adhere to.
- Interfaces define properties, methods, and events, which are the members of the interface.
- Interfaces contain only the declaration of the members.
- It is the responsibility of the deriving class to define the members.
- It often helps in providing a standard structure that the deriving classes would follow.

Interfaces

- The interface keyword is used to declare an interface.
- Interfaces are not to be converted to JavaScript. It's just part of TypeScript.
- There is no java script emitted when you declare an interface unlike a class. So interfaces have zero runtime JavaScript impact.

```
interface  IPerson{
    firstName: string;
    lastName: string;
    age: number;
    email: string;
}

let employee: IPerson={
        firstName: “Zara”,
        lastName: “Khan”,
        age: 32,
        email: ‘zara@gmail.com’
}

console.log(“Employee Name is ” + this.employee.firstName + “ ” +
this.employee.lastName + “ Age is ” + this.employee.age + “<br />”);
```

Interface with array

```
interface IPerson {
    firstName: string;
    lastName: string;
    age: number;
    email: string;
}

//with array concept
let custArray: IPerson[] = [];

custArray.push({
    firstName: “Ram”,
    lastName: “Kapoor”,
    age: 21,
    email: ‘ram@gmail.com’
});
    console.log(“With array Customer name is ” + this. custArray[0].firstName + “ ”
    this. custArray[0].lastName + “ Age is ” + this. custArray[0].age);
```

Classes in TypeScript

- Traditional JavaScript focuses on functions and prototype-based inheritance, it is very difficult to built application using object-oriented approach.
- Starting with ECMAScript 6 (the next version of JavaScript), JavaScript programmers can build their applications using this object-oriented class-based approach.
- 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.
- TypeScript supports public , private and protected access modifiers. Members of a class are public by default.
  Use the class keyword to declare a class in TypeScript.

Classes in TypeScript

- A class definition can include the following –
  1. Fields – A field is any variable declared in a class. Fields represent data pertaining to objects
  2. Constructors – Responsible for allocating memory for the objects of the class
  3. Functions – Functions represent actions an object can take. They are also at times referred to as methods

  These components put together are termed as the data members of the class.

- The **new** keyword is responsible for instantiation.
- The right-hand side of the expression invokes the constructor. The constructor should be passed values if it is parameterized.

Classes in TypeScript (Contd…)

```
class Employee {
    empId: number;
    empName: string;
    empsalary: number;

    static emppf: number = 12;
    static company: string = ‘My Dreams Company’;
}

let emp = new Employee();
emp.empId = 1001;
emp.empName = “Zara”;
emp.empsalary = 1111;
console.log(“ID is ” + emp.empId + “ Name is ” + emp.empName + “
company ” + Employee.company);
```

Constructor -Typescript

```
class EmployeeOne {
    empId: number;
    empName: string;

    constructor(id: number, name: string) {
        this.empId = id;
        this.empName = name;
    }

    getEmp(): void{
        console.log("Employee Info : "+this.empId + " " + this.empName);
    }
}

let empOne = new EmployeeOne(1001, "Zara");
empOne.getEmp();
```

Static Property

- In TypeScript we can also create static members of a class, those that are visible on the class itself rather than on the instances.

Static Property (Contd…)

```
class EmployeeTwo {
    empId: number;
    empName: string;
    static numberOfEmployee: number = 0;

    constructor(id: number, name: string) {
        this.empId=id;
        this.empName=name;
        EmployeeTwo.numberOfEmployee++;
    }

    getEmp(): void{
        console.log("Employee Info: "+this.empId+" "+this.empName);
    }
    static getNumber(): number{
        return EmployeeTwo.numberOfEmployee;
    }
}

let emp2=new EmployeeTwo(1001, "Zara");
emp2.getEmp();
console.log("No Of Employees created : "+EmployeeTwo.getNumber());
```

Inheritance

- TypeScript allows us to extend existing classes to create new ones using inheritance.
- 'extends' keyword is used to create a subclass.
- 'super()' method is used to call the base constructor inside the sub class constructor.

Inheritance (Contd…)

```
class Animal {
    constructor(public name: string) { }
    move(distanceInMeters:  number = 0) {
        console.log(`${this.name} moved ${distanceInMeters}m.`);
    }
}
class Snake extends Animal {
    constructor(name:  string) { super(name); }
move(distanceInMeters  = 5) {
    console.log("Slithering...");
        super.move(distanceInMeters);
    }
}
class Horse extends Animal {
    constructor(name:  string) { super(name); }
move(distanceInMeters  = 45) {
    console.log("Galloping...");
        super.move(distanceInMeters);
    }
```

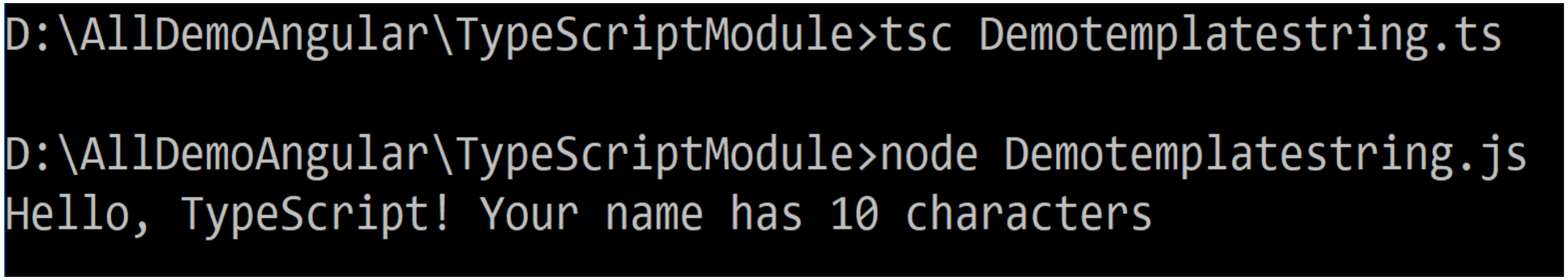
Template Strings

- In ES6 new template strings were introduced.
- The two salient features of template strings are
  - Variables within strings (without being forced to concatenate with +)
  - Multi-line strings (using backticks `)
  - TypeScript now supports ES6 template strings. These are an easy way to embed arbitrary expressions in strings:

```
var strName = "TypeScript";
console.log(`Hello, ${strName}! Your name has ${strName.length} characters`);
```

- When compiling to pre-ES6 targets, the string is decomposed:

```
var strName = “TypeScript!”;
console.log(“Hello, ”+strName +“ ! Your name has ” +strName.length +“ characters”);
```



```
D:\AllDemoAngular\TypeScriptModule>tsc Demotemplatestring.ts

D:\AllDemoAngular\TypeScriptModule>node Demotemplatestring.js
Hello, TypeScript! Your name has 10 characters
```

Generics

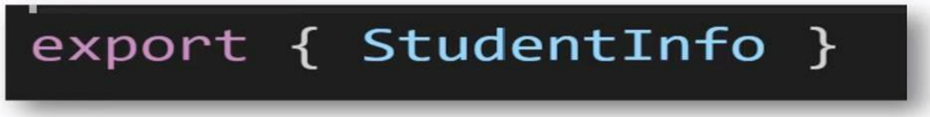
- Generics plays a vital role in creating reusable components.
- Component can be created to work over a variety of types rather than a single one.

```
function GetType<T>(val: T): string{
    return typeof(val);
}

let ename = “Abcd”;
let one = 10;
console.log(“Call Generics” + GetType(ename) + “ ” + GetType(one));

//class -generics
class GetNumber<T>{
    add:(one: T, two: T) => T;
}
var result = new GetNumber<number>();
result.add = function(x, y){
    return x+y;
}
console.log(“Addition of 5+2” + result.add(5,2));
```

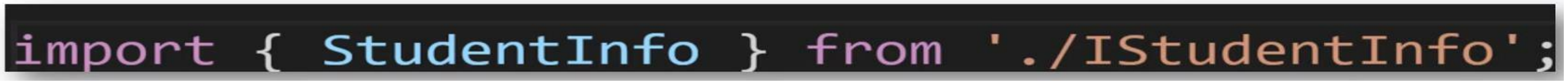
Modules

- Starting with the ECMAscript 2015,javascript has a concept of modules Typescript shares this concept.
- Modules are executed within their own scope, not in the global scope; this means that variables, functions, classes etc. declared un a module are not visible outside the module unless they are explicitly exported using one of the export forms.
- To consume a variable function class interface etc. exported from a different module.

Exporting and importing from modules are doing with these below syntax



```
export { StudentInfo }
```



```
import { StudentInfo } from './IStudentInfo';
```

Modules (Contd…)

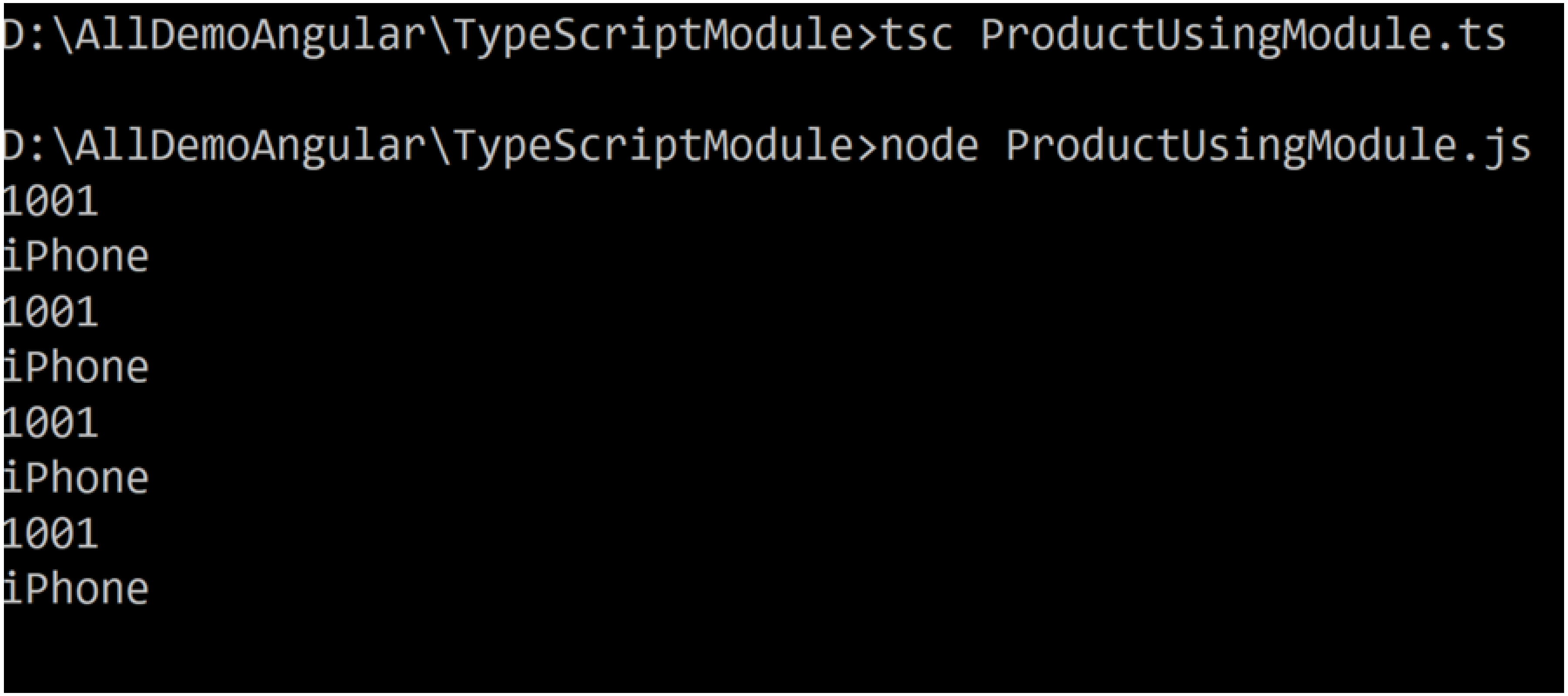
- Product.ts

```
export class IProduct {
    productId: number;
    productName: string;
}

export const company: string = “MyDreamCompany”;
```

Modules (Contd…)

```
import {IProduct} from "./Product";
import {company} from "./Product";
//Declare Product
let prod: IProduct={
        productId:1001,
       productName:"iPhone"
}
let productArray: IProduct[]=[
              {productId: 1002, productName: “LG”},
              {productId: 1003, productName: “CoolPad”},
              {productId: 1004, productName: “Mi”} ];
console.log(prod.productId);
console.log(prod.productName);

for (let pro of productArray) {
    console.log(prod.productId);
    console.log(prod.productName);
}

console.log(company);
```

Modules (Contd…)



```
D:\AllDemoAngular\TypeScriptModule>tsc ProductUsingModule.ts

D:\AllDemoAngular\TypeScriptModule>node ProductUsingModule.js
1001
iPhone
1001
iPhone
1001
iPhone
1001
iPhone
```

Summary

- TypeScript is an open source project maintained by Microsoft.
- TypeScript generates plain JavaScript code which can be used with any browser.
- TypeScript offers many features of object oriented programming languages such as classes, interfaces, inheritance, overloading and modules, some of which are proposed features of ECMA Script 6.
- TypeScript is a promising language that can certainly help in writing neat code and organize JavaScript code making it more maintainable and extensible.
- Angular 2 is built in typescript

Demo

- TypeScript Demo
- Typescript Module Demo

Lab

- Lab 1.1

Summary