==========
Angular
==========

-> Angular is a client side framework which is used to create web applications.

-> Angular can be used in combination with any backend platform such as Java, Node JS, Asp.Net, PHP, Python etc.

Angular Frontend app =====> Java Backend app

Angular frontend app =====> Asp.net backend app

Angular frontend app =====> Python backend app

Angular frontend app =====> Node JS backend app

-> Angular is developed using TypeScript language which is superset of Java Script.

-> Angular was developed by Google

-> Angular is free to use & Open Source

-> Angular is cross platform that means it works in all operating systems.

-> Angular is cross browser compatible that means it works in all browsers.

-> Angular is mainley used to create Single Page Applications (SPA).

==========
Multi Page web app vs Single Page web app
==========

-> In Multi Page web application complete web page will be reloaded for every request.

Ex : www.zoom.us

-> In Single page web applications, only content will be reloaded for every request

Ex: angular.io, www.gmail.com

==========

Angular JS vs Angular
==========

-> Angular JS is called as Angular 1 which is developed based on Java Script.

-> Angular 2 is not extension for Angular 1. Angular 2 framework is completley re-written using TypeScript.

-> Angular 2+ versions are almost same with minor changes

-> Angular JS having performance drawbacks because of 'digest loop'
-> Angular is faster in performance because no digest loops and no repetions

-> Angular JS is based on JavaScript which is prototype based oop language
-> Angular is based on TypeScript which is class based oop language.

-> Angular JS is based on MVC architecture. The code will be divided into 3 parts Model, View and Controller.
-> Angular is based on Components. Component contains logic to manipulate the data in view.

==========
TypeScript installation & First Example
==========

-> TypeScript is a general purpose programming language

-> TypeScript is superset of Java Script

-> TypeScript can be used at client side development & at server side development also

-> Browsers can't understand TypeScript directley

-> TypeScript files should be converted to Java Script and Java Script files will be executed in browser.

-> The process of converting TS file into JS file is called as 'Transpilation'

-> TypeScript compiler (tsc) we will use to perform Transpilation

-> TypeScript supports for OOPS

-> Type Script developed by Microsoft in 2012

==========
TypeScript Installation

==========

1) Install Node JS

2) Install TypeScript

-> Download Node JS installer from -> https://nodejs.org/en/download/

-> After installing Node, we can verify installation process by executing below command in cmd

```
node -v
```

-> Install TypeScript using below command

```
npm install -g typescript
```

-> After installing TypeScript, we can verify installation using below command

```
tsc -v
```

==========
TypeScript First Example
==========

-> Create TypeScript file with .ts extension and add below code

----------HelloWorld.ts----------

```
var s:string = "Hello World";
console.log(s);
```

-> Open command prompt and compile ts file

```
tsc <filename>.ts
```

Note: tsc compiler will convert ts file into js file (Transpilation)

-> Run JS file using below command

```
node <filename>.js
```

-> It should print HelloWorld message in console.

==========
Executing Java Script In browser
==========

-> By linking JS file with HTML file we can execute JS file in browser

---

```
<html>
   <head>
  <script type="text/javascript" src="helloworld.js"></script>
   </head>
   <body>
     <h2> This is my first html page</h2>
   </body>
</html>
```

-> Save the above file with .html extension

-> Right click on the file and open with browser

-> Open Developer Tools and see Java Script output in developer console.
(Fn + F12)

-> In Realtime we don't develop applications using notepad / notepad ++

-> We will use IDE to develop our applications.

Ex : VS Code IDE (Microsoft)

-> Download VS Code IDE from below URL & Install it

https://code.visualstudio.com/download

==========

Variables

==========

-> Variable is a named memory which is used to store the value

Syntax : var variableName:datatype = value;

Ex : var a:number = 100;
var name:string = "Dollop";

==========

DataTypes

==========

-> Data Type specifies what type of value that can be stored into variable

==========
List of TypeScript DataTypes
==========

number : All types of numbers (integers / floating-numbers) Ex: 101, 101.30

string : collection of characters in double quotes or single quotes. Ex : "hello"

boolean : true or false

any : any type of value

```
var pname:string="Dollop";
var age:number = 20;
var gender:string="Male";
var isEmployed:boolean = false;

console.log(`Name = ${pname}, Age = ${age}, Gender=${gender}, Is Employed=${isEmployed}`);
```

==========
what is var in javaScirpt?
==========
-> var keyword is used to declare variables in java script.

syntax : var variableName;

ex : var carName = "Benz";

-> var keyword is functional scope
----------variables.ts----------

```
var petName:string = "dog";

function setPetName(){
   var petName:string = "cat";
   console.log("Inside Function :: " + petName);
}
setPetName();
console.log("Outside Function :: "+ petName);

var index = 0;
for(var index=0; index<5;index++){
```

```
   console.log("Index Inside For Loop :: "+index);
}
console.log("Index Outside For Loop :: "+index);
```

-> In the above example it is chaning the value of index variable outside the loop also. (It is loop hole in var keyword)

==========
what is let keyword?
==========

-> To overcome the problems of 'var' keyword 'let' keyword got introduced

-> let keyword is block scope

-> let keyword variable needs to be declared before being used

syntax : let variableName = value;

ex : let msg:string = "Welcome to Angular...";

----------variables.ts----------

```
function display(){
   let msg:string = 'Welcome to Dollop IT..!!';
   {
      let msg:string = 'Welcome to Angular..!!';
      console.log("Inside Block :: "+ msg);
   }
   console.log("Outside Block ::"+ msg);
}
display();

let index = 0;
for(let index=0; index<5;index++){
   console.log("Index Inside For Loop :: "+index);
}
console.log("Index Outside For Loop :: "+index);
```

-> In the above program, outside loop index value will be printed as '0'

==========
Arrays
==========

-> Arrays are used to store group of values
-> In TypeScript, Arrays size is not fixed. We can store as many values as we want
-> In Type Script Array we can store Hetereogenious values also

```
--------------------------------variables.ts----------------------------------
let fruits:string[];
fruits = ['mango','apple','banana'];
console.log(fruits);

let animals:Array<string>;
animals = ['Tiger','Lion'];
console.log(animals);

let genricArray:Array< string | number | boolean >;
genricArray = ['IBM',10,true,102,'TCS'];
console.log(genricArray);
------------------------------------------------------------------------
-> We can access arrays vales based on index also. Index starts from zero (0).
   ex :  console.log(genericArray[0]);  ---> it will print IBM

-----------------------------------------------------------------------------
<html>
   <head>
      <script src="variables.js"></script>
   </head>
   <body>
      <h2>Working with Variables</h2>
   </body>
</html>
-----------------------------------------------------------------------------

let fruits:string[];
fruits = ['mango','apple','banana'];
console.log(fruits);


let animals:Array<string>;
animals = ['Tiger','Lion'];
console.log(animals);

let genricArray:Array<string | boolean | number >;
genricArray = ['IBM',10,true,102,'TCS'];
console.log(genricArray[0]);
================
Loops
================

-> If we want to execute same logic multiple times then we will go for loops concept.

In programming language we can use 2 types of loops
```

1) Range based loops / Definite Loop

If we know no.of iterations for a loop then we will use range based loops

Ex : for loop

Scenario : Print Numbers from 1 to 10

2) Conditional based loops/Indefinite Loops

If we want to execute logic repeatedly until unless condition becomes false.

Ex : while & do-while

Scenario : Print Numbers till evng 4 pm.

What is break statement?

If we want to come out from the loop we will use break statement.

What is continue statement?

If we want to skip current iteration of the loop, we will go for continue statement.

----------loops.ts----------

```
for (let i=1; i<=10 ; i++){
    for(let j = 1; j<=5; j++){
      if(j==3){
         break;
      }
      console.log(" j value : "+j);
    }
    console.log("i value : "+i);
}
```

-> When we use break stmt inside inner loop only that loop execution will be stopped.

----------loops.ts----------

```
for (let i=1; i<=10 ; i++){

   if(i == 5){
      continue;
   }
    console.log("i value : "+i);
}
```

==========
Functions
==========

-> Functions are primary building blocks of any program

-> In Java Script, functions are very important part bcz java script is a functional programming language.

-> With the help of functions we can mimic/implement concepts of OOPS like classes, objects, polymorphishm and abstraction.

-> Functions ensure that the program is maintainable and reusable and organized into readable blocks.

-> TypeScript supports OOPS but functions are still part of Type Script language.

-> In TypeScript, functions can be of two types.

  a) Named Functions
  b) Anonynous Functions

==========
Named Functions
==========
-> The function which contains the name is called as 'Named Function'.


```
function welcome(){
    console.log("Welcome to Angular");
}

welcome( );
```

-> A function can have parameters & return type also.

```
function add(x: number, y:number) : number {
  return x+y;
}
----------functions.ts----------
function welcome(){
   console.log("Welcome to Angular");
}

function sum(x:number, y:number): number{
   return x + y;
}
```

```
welcome();
let result = sum(2,3);
console.log(result);
```

==========
Anonyous functions
==========

-> Anonymous function is the one which is declared as an expression.

-> This expression is stored in a variable and function itself doens't contain name.

-> Anonymous functions will be invoked using the variable name what function is store in.

-> Anonymous functions also can have parameters & return types

```
let result = function (x:number, y:number): number {
  return x + y;
}

result(10, 20);
```

```
function doWork(x:number, y:number) : number {
        let result:number = x + y;
    return result;
}

let result = doWork(20,30);
console.log(result);
```

```
let result = function(x:number, y:number) : number {
  return x + y;
}
let value = result(10,20);
console.log(value);
```

==========
Function Parameters
==========
-> Parameters are the values or arguments what we will pass to a function

-> In Typescript, the compiler excutes a function to recieve the exact number and type of arguments as defined in the function signature.

-> If function expects 3 arguments, the compiler checks that user has passed values for all three parameters (It will check exact matches).

```
function fullName(fname:string, lname:string) : string{
  return fname + lname;
}

fullName("Dollop"); // Compiler Error : Expected 2 args, but got 1

fullName("Dollop", "Infotech"); //Ok, return "Dollop Infotech"

fullName("Dollop", "Infotech", "ABC"); //Compiler Error : Expected 2, but got 3
```

==========
Optional Parameters
==========

-> TypeScript has an Optional Parameter Functionality

-> Optional Parameters Should be presented at the end

-> Optional Parameter will be represented with ? symbol

```
function greet(msg:string, name?:string) : string {

    return msg +' '+ name + ' !';
}
```

```
greet('Hello', 'Dollop'); //Ok it return Hello Dollop!
greet('Hello'); //Ok, return Hello undefined
greet('Hello', 'Dollop', 'Hi'); // CE - Expected 2 but got 3
```

==========
Default Parameters
==========

-> TypeScript provided option to add default values to parameters

-> If user doesn't pass the value for parameter, TypeScript will initialize the parameter with default value.

```
function(name:string, msg:string="Hi") : string {

    return msg + name ;
```

```
}

console.log(greet('Dollop', 'Good Morning'));
console.log(greet('Dollop'));
```

==========
Rest parameters
==========

-> If we don't how many parameters we need to take then we can use Rest Parameters.

-> We can specify Rest Parameters using '...' (ellipses)

-> When function having rest parameters, we can pass zero or more arguments.

-> The Rest parameter should be last parameter in the function

---

```
function greet(msg:string, ...names:string[]) : string {

  return msg + " " + names.join(",") ;
}

console.log(greet('Hi', 'Dollop' , 'Infotech')); //OK
console.log(greet('Hello')); //Ok
```


==========
Arrow Functions
==========
-> Fat arrow functions are used for anonymous functions

-> Anonymous functions are the functions with expression

-> Anonymous functions are called as Lambda Functions in Java
synax :

(param1, param2, param3..... paramN) => expression

-> Using fat arrow (=>) we are removing function keyword

Example

```
 let sum = (x:number, y:number) : number => {
        return x + y;
 }
```

sum(20,20);

-> In the above example sum() is an arrow function

-> (x:number, y:number) denotes function parameters

-> :number represents function return type

-> fatarrow (=>) seperates function parameters and function body

-> Rigtside of the fat arrow part is called function expression. It can contain one or more statements

```
================================
Arrow Function Without Parameters
================================
let msg = () => console.log ("Welcome to Dollop IT");

msg();
```



```
================================
Function Overloading
================================
```

-> TypeScript supports for function Overloading

-> Function Overloading means we can write multiple functions with same name but different parameter types and return type.

Note: In function overloading no.of parameters should be same.

```
function add (a:number, b:number) : number;
function add (a:string, b:string) : string;

function add (a:any, b:any) : any {
  return a + b;
}

add(10,20);
add("Dollop", "Infotech");

================
OOPS in TypeScript
================
```

-> OOPS stands for Object Oriented Programming System

-> In Object Oriented Programming Languages Classes are fundamental entities which are used to create re-usable components.

-> Interms of Oops Class is a template or blueprint for creating objects

Class definition contains following things
---

1) Fields / Properties : Variables declared in a class

2) Methods: Represents action

3) Constructors : Responsibe for initializing object

4) Nested Class : A class can contain another class

5) Object : A physical item or collection of properties

-> TypeScript is an Object - Oriented Java Script Language so it supports Object Oriented Programming features like classes, interfaces, polymorphism, data-binding etc.

-> JavaScript ES5 version doesn't support for classes.

-> TypeScript supported this feature from ES6 version

-> Typescript has built-in support for classes because it is based on ES6 version.

Syntax to declare class In TypeScript

```
class <class-name> }

    // fields

    // methods
}
```

-> 'class' is a keyword which is used to declared class in TypeScript

Type Script Class Example

```
class Student {
```

```
  let studentName : string;

  let studentRank : number;

   getStudentGrade() : string {
   return "A+";
   }
}
```

-> Once we transpile Student class it looks like below in Java Scirpt

```
var Student = /** @class */ (function () {
    function Student() {
    }
    Student.prototype.getStudentGrade = function () {
        return "A+";
    };
    return Student;
}());
```

Typescript class with Properties & Function

```
class Student{

    studentName : string;
    studentRank : number;
    studentMarks: number;

     getStudentGrade() : string {
        if(this.studentMarks >= 75){
            return "A";
        }else if(this.studentMarks >=65 && this.studentMarks <60){
            return "B";
        }else{
            return "C";
        }
     }
  }

     let s1 = new Student(); //obj creation
     s1.studentName="John";
     s1.studentRank=100;
     s1.studentMarks=80;

     console.log(s1.studentName);
```

```
    console.log(s1.studentRank);
    console.log(s1.studentMarks);
    console.log(s1.getStudentGrade());

    let s2 = new Student(); //obj creation
    s2.studentName="Smith";
    s2.studentRank=200;
    s2.studentMarks=50;

    console.log(s2.studentName);
    console.log(s2.studentRank);
    console.log(s2.studentMarks);
    console.log(s2.getStudentGrade());
```

-> For one class we can create multiple objects.

-> All the objects of class will share same properties & functions

-> Using object we can access properties & functions

==========
Constructor
==========

-> Constructor is a special function which is part of class

-> Constructor is used to initialize the object

-> Constructor name will be same as class name

-> In TypeScript, constructor always will be defined with 'constructor' keyword

-> In constructor, we can access members of class using 'this' keyword

-> When we create Object for a class, constructor will be called automatically.

-> If we create multiple objects for a class, for each object creation constructor will be executed.

-> Constructor can arguments but it can't return any value

-> TypeScript doesn't support for Constructor Loading

-> Constructor is a special function which is used to initialize current class variables

-> In TypeScript constructor will be represented using 'constructor' keyword.

-> Constructor can take arguments but no return type

-> Constructor Overloading not supported in TypeScript.

-> Constructor will be called when we create object for the class

---

```
class Student {
    studentId: number;
    studentName: string;

    constructor(studentId: number, studentName: string){
    this.studentId = studentId;
        this.studentName = studentName;
    }
}

let student = new Student(101, "John");
```

```
class Student {
   studentId: number;
   studentName: string;

   constructor(studentId: number=10, studentName: string="Roy"){
       this.studentId = studentId;
       this.studentName = studentName;
   }
}

let student = new Student();
console.log(student.studentId);
console.log(student.studentName);
```

-> In the above code we have declared default values for constructor arguments.

-> If we don't specify the values for constructor at the time of obj creation then it will take default values.

==========

Inheritence

==========

-> The process of extending the properties from one class to another class is called as Inheritence.

-> Jus like object oriented languages such as java and c#, TypeScript classes can be extended to create new classes with inheritence.

-> To extend the properties from one class to another class we will use 'extends' keyword.

----------Person.ts----------

```
class Person {
      name: string;

      constructor(name:string){
      this.name = name;
      }
}
```

----------Employee.ts----------

```
class Employee extends Person {

       empId : number;

       constructor (empId : number, name: string){
     super(name);
         this.empId = empId;
       }
}
```

```
let p = new Person("John");
console.log(p.name);

let emp = new Employee(101, "Smith");
console.log(emp.empId);
console.log(emp.empName);
```


----------inhertience.ts----------

```
class Person {
   name: string;

   constructor(name:string){
    this.name = name;
   }
}

class Employee extends Person{

   empId : number;

   constructor (empId : number, name: string){
   super(name);
     this.empId = empId;
   }
}
```

```
let p = new Person("John");
console.log(p.name);

let emp = new Employee(101, "Smith");
console.log(emp.empId);
console.log(emp.name);
```
---

compile : tsc inheritence.ts

execution : node inheritence.js

Note: A class can extend the properties from only one class at a time. Multiple inheritence not supported by TypeScript.

==========
Access Modifiers
==========

-> Access Modifiers specifiy weather the members of the class can be accessible.

-> To specify accesibility for class members outside of the class we will use Access Modifiers.

-> These Access Modifiers are used for Security in OOP

-> For each member in class we can specify access modifier seperatley

==========
Type Script supports 3 Access Modifiers
==========

1) public

2) private

3) protected

-> public members of the class are accessible anywhere in the program. In the same class and also outside the class also we can access.

-> private members are accessible only with in the same class. If you try to access them outside the class it will throw compile time error.

-> protected members are accessible with in the same class and also in corresponding child classes.

Syntax for creating class members with access modifiers

```
-----------------------------------------------------------------------
class <class-name> {

  accessmodifier  propertyname : dataType;

      accessmodifier functionName : returnType {

  }
}
------------------------accessmodifiers.ts----------------------------
class Student{
   public studentId:number = 101;
   private studentName:string = "Dollop";
   protected marks:number = 80;

   public display1():void{
      console.log("------------Student Details: Parent-----------");
      console.log(this.studentId);
      console.log(this.studentName);
      console.log(this.marks);
   }
}

class EngStudent extends Student{

   public display2():void{
      console.log("------------Student Details: Child-----------");
      console.log(this.studentId); //accessible
      //console.log(this.studentName); // not accessible
      console.log(this.marks); //accessible
   }
}

class Test{

   sampleMethod(){
      var s1 = new Student();
      s1.display1();

      var s2 = new EngStudent();
      s2.display2();

      console.log(s1.studentId); //accessible
      //console.log(s1.studentName); // Not accessible
      //console.log(s1.marks); // Not accessible
   }
```

```
}

var t:Test = new Test();
t.sampleMethod();
```

==========
Interfaces
==========

-> Interface is the model of the class, which describes the list of properties and methods of a class.

-> Interface doesn't contain actual code, it contains only list of properties and methods.

-> In Interface we will write only method declarations. Interface doesn't contain method implementation.

-> When a class is implementing interface, its mandatory that all methods of interface should be implemented in that class.

-> To implement a interface class will use 'implements' keyword.

-> All the methods of interface by default public

-> One interface can be implemented by multiple classes and one class can implement multiple interfaces also.

-> Interfaces acts as mediator between two or more developers. Interfaces are called as Contracts.

Syntax For Interface Creation

```
interface InterfaceName {

  property : dataType;

  method(args) : returnType ;

}
```

Implementation class for interface

```
class ClassName implements InterfaceName {

    propertyName : dataType ;

    method(args) : returnType {

       //body goes here

   }
```

```
}


----------------------------------interfaces.ts-----------------------------------------
interface Student{
   fname : string;
   lname : string;
   getFullName() : string;
}

class StudentImpl implements Student{
   fname : string;
   lname : string;
   getFullName():string{
      return this.fname+" "+this.lname;
   }
}

//interface ref variable can hold its impl class obj
var s:Student = new StudentImpl();
s.fname = "Dollop";
s.lname = "Chakravarthy"
console.log(s.getFullName());
-------------------------------------------------------------------------------------
Enumerations
------------
-> Enumeration is a collection of constants

-> Enumeration acts as datatype

Syntax for creation Enumeration
-------------------------------
enum EnumName{

     const1, const2, const3..... constN
}


Create a property of Enumeration Type
-------------------------------------
class ClassName {

    property : enumerationName;
}
```

```
--------------------------------Enumerations.ts-------------------------------------------
enum CourseNames{
   Java, DotNet, Testing
}

class Student{
   name : string;
   age : number;
   course : CourseNames;
}

var s:Student = new Student();
s.name = "Dollop";
s.age = 26;
s.course = CourseNames.Java;

console.log(s.name);
console.log(s.age);
console.log(CourseNames[s.course]);
```



```
-------------------------------------------------------------------------------
Abstract Class
-------------------------------------------------------------------------------
```

-> The class which contains both concrete and abstract methods is called as abstract class.

-> To define abstract class in Typescript we will use 'abstract' keyword.

-> We will create sub class for abstract class. The sub class of abstract class must define all abstract methods.

```
-------------------------------------------------------------------------------
abstract class Person{

  name : string;

  constructor(name:string){
     this.name = name;
  }

  display(): void {
     console.log(this.name);
  }
```

```
  abstract find(string) : Person;
}

class Employee extends Person {

  empCode: number;

  constructor(name:string, code: number){
    super(name);
      this.empCode = code;
  }

  find(name:string) : Person {
          return new Employee(name, 1);
  }
}

let emp:Person = new Employee("Steve", 100);
emp.display();

let emp2:Person = emp.find('James');
```

==========

Type Script - Modules

==========

-> In Large applications it is recommended to write each class in seperate file

-> To access the class of one file in another file we will use Modules concept in TypeScript

-> Module is a file (ts file) which can export one or more classes (or interfaces or enums) to other files. The other files can import classes (or interfaces or enums) that are exported by the specific file.

-> To export class we will use 'export' keyword in source file

-> To import class we will use 'import' keyword in destination file

-> To import from current folder we will use './'

-> To import from sub folder in current folder, we will use ./subfolder

-> To import from parent we will use '../'

Syntax for developing Modules

```
----------------------------------------file1.ts----------------------------------------
export class ClassName1 {

    //properties & functions
}

----------------------------------------file2.ts----------------------------------------
import {ClassName1} from "./file1.ts"

class ClassName2 {
}
-----------------------------------------------------------------------------------------
--------------------------------Student.ts-----------------------------------------------
export class Student{

   studentId : number;
   studentName: string;

   constructor(id:number, name:string){
      this.studentId = id;
      this.studentName = name;
   }
}
--------------------------------School.ts-----------------------------------------------
import {Student} from "./Student";

class School {

   students:Student[] = [
      new Student(101, 'John'),
      new Student(102,  'Steve'),
      new Student(103, 'Nick')
   ];

  display() : void {
      for(var i in this.students){
         console.log(this.students[i]);
      }
  }
}

let school: School = new School();
school.display();
-----------------------------------------------------------------------------------------
```

Last session: Modules in TypeScript

-> If we want to use the class/interface/enum of one file in another file then we will go for modules.

-> To use class/interface/enum in another file we will use export and import concepts.

-> If we export then only we can import.

----------Student.ts----------

```
export class Student{
    //properties & functions
}
```

----------School.ts----------

```
import {Student} from './Student'

class School{
   //properties & functions
}
```

==========