**Course content**

**part-1**

1. what is web application?
2. multi-page application
3. single page application
4. Angular introduction
5. ang Js vs ang f/m
6. ang fm version history
7. advantage
8. Ang architecture

**part-2**

1. typescript introdction
2. advantages
3. javascript vs typescript
4. installation
5. variables
6. datatypes
7. OOPS
8. class
9. objects
10. constructor
11. inheritance
12. access modifiers
13. Intefaces(Contracts)
14. Enumeratioins
15. modules
16. example with tyescript

**Part-3**

1. Revise ang intriduction
2. revise ang architecture
3. Environmen setup
4. First app creation using angular
5. Understanding ang application folder structure
6. Live server to run ng pps
7. Components
8. Modules
9. Data bindings
10. Login app using angular
11. reg App using angular
12. Directives
13. Pipes
14. form based app using angular
    * Template driven
    * Reacive forms
15. form validation
16. Routings

part 4

---------------

1. http client
2. consuming backeng services creted in Rest API

Now a days, in the industry companies are following micro services architecture to develop the applications.

Micro services is an architectural Design Pattern which is used to develop our applications with loosely coupling and easy maintanance.

as art of thi we will develop business logic as Rest APIs.

RESTAPIs are for B2B communication(Business to business)

Angular is a client side framework which is used to develop webapplication.

web pplicaion is the applicaiton hich runs in the server.

multiple user can acces it using internet. Ex. Fb, gmail

we can see 2 types of eb appication

1. multipage web applicatioon

2. single page web application using ngulr frmewok

**what is multipage application?**

for each request if our page(it may contains common thing or jsps also) is going to reload then that is multi-page application

we get web page as a repsosnse

what is single page application?

here for eachrequest we will get json data as a response instead of reloading the same page.

for first requet, we will get webpage as a response or html or web page will be loaded but after that we wil get json data as a data

ex. gmail.com,angular.io data will be rendered.

Advantages

=============

1. developement will be easy

2. mainatinance will be simple

3. good performance.

Note:- in single pge application, common content willbe loaded once.

angular history

angularverions

angular architecture

=====================

1. component

2. meta data

3. services

4. template

5. directives

6. pipes

there are 2 flavours of angular

1. Angular JS (angular 1.x)

2. Angular (Angular 2+ version)

Angular JS is developed using java script

Angular framework is developed by using Type script

TYPESCRIPT

* It is programming language which is developed by javascript.
* its a superset of javascript which adds data types + classes + interfaces etc
* it can be used for both client +server side programming.

Type script features

* it is general purpose programming language which is used anywhere.
* built on top of java script.
* it supports all java script features with some additional features.

NOTE: java script can be executed in browser but typescript cant, so we need to convert typescript into javascrpt first. This process is called as transpilation.(typescript compiler)

file.ts ->tsc -> fle.js -> browser -> output

typescript versions

-------------------

typescript 0.8 : 2012 (initial version)

typescript 4.x: 2020

## static typing & dynamic typing

whenever we can fix a data type for the variable while variable declaration and we can't change its data type throughout the program then it is called as ***static typing***

ex: int e=20; //valid

age="ankur";//invalid

C, C++, java ,.Net are static typing language

If we can't fix a data type for variable while variable declaration and the data type will be automatically taken by the runtime engine at the time of program execution, then it is called as ***dynamic typing***.

var age;

age=20;//valid

age="ankur";//valid

ex:. javascript and python are dynamic typing language

Typescript support "***Optional static typing***". It supports both static and dynamic typing.

Type script maintains type safety like Generics in java. If we specify data type while declaring the variable and if we assign wrong type of values into that variable then compiler shows error.

**Installation of typescript**

==========================

inssall node js

npm install -g typescript

tsc -v

First typescript example

=======================

Create Typescript filewith .ts extenstion and add below code

var s:string="Hello Anur";

console.log(s);

Open cmd and compile it with

tsc <filename>.ts

it will convert ts file into js (traanspilation)

Run js file using node with below command

node <filename>.js

## Variables in Typescript

Syntax : var variable\_name:data\_type=value;

## Datatypes in typescript

====================================

***number*** : for all type of numbers(int,float can be used)

***string*** : collections of charaacters in double or single quote

***boolean*** : true or false

***any*** : any type of value

## const, var,let keywords

what is 'const' keyowrd typescript?

used to declare constant varible where initialization in mandatory.

we can't modify that variable value

what is var keyword?

**functional scope**

what is let keyword?  **block scope**

ex:

var age:number=12;

function getAge(){

var age:number=25;

console.log('Inside function age:',age);

}

getAge();

console.log('Outside function age:',age);

var index=0;

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

console.log('inside for loop index:',index);

}

console.log('outside index:',index);

output

Inside function age: 25

Outside function age: 12

inside for loop index: 0

inside for loop index: 1

inside for loop index: 2

inside for loop index: 3

inside for loop index: 4

inside for loop index: 5

outside index: 6

let keyword example

console.log("--------------------let demo------------------------");

function disply() {

let msg: string = "nikhil";

console.log('outside block:',msg);

{

let msg: string = "ankur";

console.log('inside block:', msg);

}

console.log('outside block:',msg);

}

disply();

output:

--------------------let demo------------------------

outside block: nikhil

inside block: ankur

outside block: nikhil

## Arrays in typescript

=======================

used to store group of heterogonous values without any array size limit.

in typescript array we can store values of different types of values in single array

syntax:

let/var variablename:**datatype[]**;

let/var variablename:**Array<datatype>**;

let/var variablename:**Array<datatype1 | datatype2>**;

ex:

let names:string[];

names=['ankur','dhanu','manu'];

console.log(names);

let surnames:Array<string>;

surnames=['wadatkar','sabale'];

console.log(surnames);

let randomData:Array<string | number>;

randomData=['ssss',654,'fsdfs'];

console.log(randomData);

let anyData:Array<any>;

anyData=['ssss',654,'fsdfs',true,12.23];

console.log(anyData);

output[ 'ankur', 'dhanu', 'manu' ]

[ 'wadatkar', 'sabale' ]

[ 'ssss', 654, 'fsdfs' ]

[ 'ssss', 654, 'fsdfs', true, 12.23 ]

## FUNCTION IN TYPESCRIPT

=========================

* Functions ae primary building blocof any program
* as javascript is functional programming language so funcions are very
* important in typesscript.
* it ensure that program is maintaninable nd reusable
* typescript supports both oops and Functions
* in typescript, functions can be of two types : named and annonymous function
* it contains return type and parameters

Syntax

function function\_name(var\_name:data\_type) : returnvalue\_dataType{

  //body

  return statement ;

  }

ex:

function addNumbers(d:number,b:number):number{

return d+b;

}

console.log(addNumbers(5,10));

function tellName(d:string):string{

return d;

}

console.log(tellName('nkur'));

## ANNONYMOUS FUNCTION INTYPESCRIPT

==================================

* the function which doesn't contain a name is called as an Anonymous function.
* it is declared as an expression.

Syntax

let var\_name = function (parameters): return\_type {

  return statement;

}

ex:

let substract=function(a:number,b:number) : number{

return a-b;

}

console.log(substract(100,50));

## Optional Parameters in typescript

=======================================

In function we pass parameters as a input, but sometimes the parameters sent may not match due to mistake, so we have the parameters in typescript called 'Optional Parameters'

* this should be the last parameters/argument
* represented as ?

function wish(age: number, name?: string): string {

  retrurn name + ' ' + age;

}

Ex:

let wishMe=function(msg:string, names?:string):string{

return msg+' '+names;

}

console.log(wishMe('GM')); //GM undefined

## Default parameters

========================

We can use default parameters also.

refer below example:

let wishMe=function( msg:string,names:string="ankur"):string{

return msg+' '+names;

}

console.log(wishMe('GM','nikhil'));

console.log(wishMe('Hii'));

output:

GM nikhil

Hii ankur

## REST PARAMETERS in Typescript

=============================

if we don't know how mny argument we need to pass then we cn use

variable arguments in typescript.

Note: it should be the last paramters of the functons paramters

list.

ex:

function showNames(...z:string[]){

console.log(z);

}

showNames('aa','dd');

## OOPS IN Typescript

===========================================

In OOPS clsses re the main entity which is used to crete reusable components.

Class is a model or plan tocreate the objects.

Class contains following things:

Fields/propeties : varibles declrred in class.

Methods: to perform appication

Nested classes: class can contain another class

Constructor: used to initialization of objects

Object: physical item or collection of properties.

Syntx:

class <class\_name>{

  //Fields

  //Methods

}

class Student{

let name:string;

let age:number;

getname():string{

//logic

return name;

}

}

how to create object in typescript?

using new keyword.

let vaiable\_name=new classname();

let student=new Student();

assigning values to object

student.name='ankur';

student.age=55;

class Student{

//constructor

constructor(name:string,age:number){

this.studentAge=age;

this.studentname=name;

}

//variable declaration

studentname:string;

studentAge:number;

//method definition

getWish():string{

if(this.studentAge==18)

return "18";

else if(this.studentAge==20)

return "20";

else return "don't know";

}

}

let student=new Student('NANA',55);

student.studentAge=15; //overriding vlues

student.studentname='nana';

console.log(student.getWish());

console.log(student.studentname);

CONSTRUCTOR IN TYPESCRIPT

==========================

\* same as java constructor.

\* constructor overlading is not here

\* use consructor keyword to define Constructor.

## ACCESS MODIFIERS IN TYPESCRIPT

=========================================

it will specify the access of the member of the class like field or function.

used to achieve security in OOPS.

we have following odifiers

public private protected

above all modifiers are same as in java.

syntax same as java.

## INTERFACES IN TYPESCRIPT

======================================

same as interface in java.

those are called as contracts.

syntax:

interface name{

prperty:datatype;

method(arg): returnType;

}

class implementationClass implements interacname{

prperty:datatype;

method(arg): returnType{

//body;

}

}

## Typescript - Modules

=========================

in large application, we will have multiple classes.

it is highly recommended to write each class in separate file.

To access the class of one file in another file we will use module concept in Typescript.

Module is file(ts file) which can export one or more classes to other files.

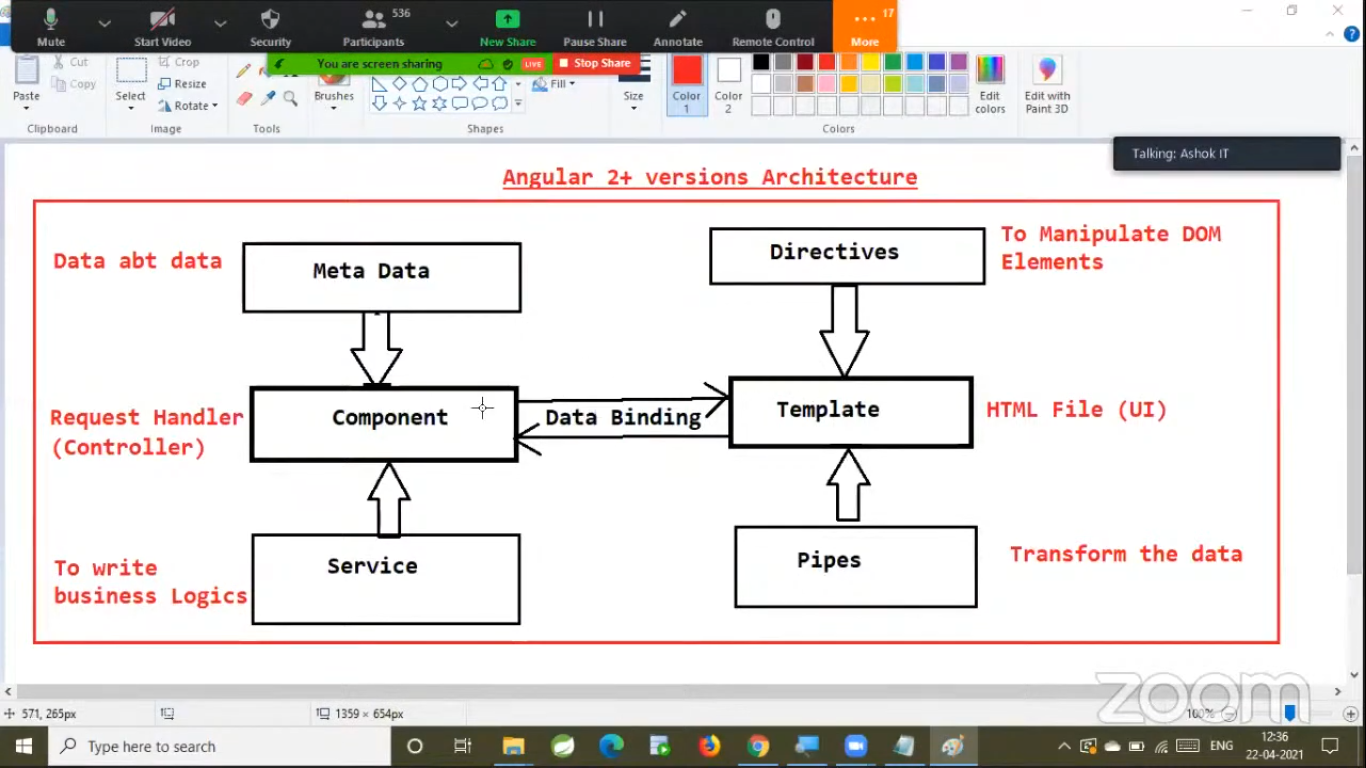
to export a class, we will use 'export' keyword.

to import a class, we will use 'import' keyword. ex import {class\_name} from "package\_name";

ANGULAR

# Building blocks of angular

1. component : application state + application logic
2. metadata : details about component / module
3. template: view file (presentation logic)
4. data binding : connection between HTML elements and component properties
5. module : group of components, directives an pipes
6. services: reusable business logic
7. dependency injection: injecting service objects into components
8. directives : to manipulate DOM elements
9. pipes: transforming values before displaying



**Component :** It is a typescript class which is responsible to handle the request coming to template

**Template** is html file containing presentation logic.

**Data binding** is used to bind data between component and template.

**Service** is typescript class which contains reusable business logic.

**Dependency injection** is used for injecting service calls obj to component class obj.

**Metadata** : data about data called as metadata.. relation between component and templae will be maintained in metadata.

**Directives** used to manipulate DOM elements.

**Pipes** are used to transform data before displaying in template.

**Collection** of components called as Module

**Routings** are used to navigate from one to another page.

# First Angular Application

To create angular application, execute below command

***Ng new <app-name>***

Once application got created, navigate into application folder an execute belocommand to run app

***Ng serve***

Angular applications re deployed to live server which runs 4200 on port number.

<http://lolhost:4200/>

*when we create angular application by default ‘app-component’ folder will be created.*

*‘app- component’ will be called as a* ***parent component*** *in angular application.*

*Angular application execution will start from ‘app-component’ only.*

*App.module.ts*

*App.component.ts -🡪 contains logic to handle request*

*App.component.spec.ts*

*App.component.html*

*App.component.css*

# Angular folder structure

Package.json

Tsconfig.json

Angular.json4main.ts

Src/index.html

Src/styles.css

App Module is clleds ‘strtup module’ in angular application.

‘App Module’ will bootstrap angular application and will load ‘App component’

**App Module** is called startup module of our application.

It will bootstrap angular application and will load “app Component”

* Angular application execution flow starts from “App Module”. Also called a s bootstrapping module in angular
* App Module bootstraps App component.
* To access component, we will use Component selector, we can find that tag in component.ts file
* In angular application, index.file will act as welcome file. In this file we will write App Component selector to access that component when this file will be loaded.

# Component related points

It represents certain sections in web page. For ex. For login we will represent it by Login component.

We can create a component by

***Ng generate component <componentName>***

Or

***Ng g c <componentName>***

When a component is created, below files will be created automatically

<component\_Name>.component.ts properties and methods

<component\_Name>.component.css styles

<compone­nt\_Name>.component.html presentation

<component\_Name>.component.spec.ts unit testing

It is typescript class which is used for dealing with request.

We can invoke a component in web page by using selector tag which is present in component.ts file

We can represent a typescript class as a component using ***@Component*** decorator.

# Data binding

* Data binding is technique which is used to establish a relation between component and templates.
* By using data binding, we can send data from component to template and vice versa.
* If component is changed then template changes automatically. If template is changed then the component is changed automatically.

Data bindings are of 4 types:

1. Interpolation
2. Property binding
3. Event binding
4. Two way binding

## Interpolation

Used to display variable/property values in template.

If variable/property values is changed then immediately it will be updated in the template.

Syntax

{{propertyName}}

## Property binding

* It is used to send the data from component to template and assign the same into an attribute of tag.
* If the property is changed then immediately it will be updated in template.

Syntax : [attribute]=\*property

## Event binding

It is used to pass event notification from template to component.

Syntax: <tag (event) = “method()”></tag>

## Two way binding

* Two way data binding is the combination of both property binding and event binding.
* To achieve two-way data binding have “ngModule” directive.
* Two way data binding is applicable only for <input> and<select> tags.
* To use two-way data binding we must import “Forms Module”.

# Directives

Directives are instructions in the DOM. They specify how to place your components and business logic in the Angular.

Directives are js class and declared as @directive. There are 3 directives in Angular.

* Component Directives
* Structural Directives
* Attribute Directives

**Component Directives:** Component directives are used in main class. They contain the detail of how the component should be processed, instantiated and used at runtime.

**Structural Directives:** Structural directives start with a \* sign. These directives are used to manipulate and change the structure of the DOM elements. For example, \*ngIf and \*ngFor.

**Attribute Directives:** Attribute directives are used to change the look and behavior of the DOM elements. For example: ngClass, ngStyle etc.

Difference between Attribute Directive and Structural Directive

|  |  |
| --- | --- |
| **Attribute Directives** | **Structural Directives** |
| Attribute directives look like a normal HTML Attribute and mainly used in databinding and event binding. | Structural Directives start with a \* symbol and look different. |
| Attribute Directives affect only the element they are added to. | Structural Directives affect the whole area in the DOM. |

Directives Examples:

Component.ts file

import { Component } from '@angular/core';

@Component({

  selector: 'app-directives-basics',

  templateUrl: './directives-basics.component.html',

  styleUrls: ['./directives-basics.component.css']

})

export class DirectivesBasicsComponent  {

  colorName:string="red";

  className:string="class1";

  firstName: string = 'Nikhil';

  lastName: string = 'Wadatkar';

  age: number = 25;

  receiveNewslettr: boolean = true;

  gender: string = 'male';

  country: string = 'India';

  address: string = 'Morshi';

  condiion:boolean=true;

  conditionTemplate:string="";

  condiionIfElse:boolean=true;

  switchcasevar:string="";

  changeInfo() {

   if(this.colorName=="red")

   this.colorName="pink";

  }

  changeClass() {

    if(this.className=="class1")

    this.className="class2";

   }

   CheckCondition(){

    if(this.conditionTemplate=="aa")

    this.condiionIfElse=true;

    else

    this.condiionIfElse=false;

   }

}

Component.html file

**ngStyle**

syntax: <tag [style.cssproperty]=”component.property name” </tag>

FirstName:

<input type="text" [(ngModel)]="firstName" [style.color]="colorName" /> Change

color:

<button type="submit" (click)="changeInfo()">Click here to check ngStyle</button><br>

**ngClass**

syntax: <tag [ngClass]=”component.property name” </tag>

LastName:

<input type="text" [(ngModel)]="lastName" [ngClass]="className" /> Change color:

<button type="submit" (click)="changeClass()">

  Click here to check ngClass</button><br>

**ngIf**

syntax: <tag \*ngIf=”condition”>

</tag>

<h2>\*ngIf Directive</h2>

<p \*ngIf="condiion">Congratulation</p>

<br>

<p \*ngIf="!condiion">try again</p>

<br>

**ngIfClass**

syntax:

<***tag \*ngIf=”condition; then template1; else template2”></***tag>

<ng-template # template1>

……….

</ ng- template >

<ng-template # template2>

……….

</ ng- template >

h2>\*ngIf else Directive Write here:</h2>

if(aa)

true else false<br>

<input type="text" [(ngModel)]="conditionTemplate" />

<button type="submit" (click)="CheckCondition()">Check Condition</button>

<p \*ngIf="condiionIfElse; then template1; else template2">Congratulation</p>

<br />

<ng-template #template1> Template 1 </ng-template>

<ng-template #template2> Template 2 </ng-template>

Ng-template is a container inside which we can place any number of tags.

**ngSwitch**

syntax:

***<tag \*ngSwitch=”propertyname”>***

***<tag \*ngSwitchCase=”’value1’”></tag>***

***<tag \*ngSwitchCase=”’value3’”></tag>***

***<tag \*ngSwitchCase=”’value2’”></tag>***

***</tag>***

Here he value should be inside “**’ value ’**”

h2>Switch case exxample Enter option:</h2>

<select [(ngModel)]="switchcasevar" >

  <option>aaa</option>

  <option>bbb</option>

  <option>ccc</option>

  <option>ddd</option>

</select>

<div [ngSwitch]="switchcasevar">

  <p \*ngSwitchCase="'aaa'">aaa</p>

  <p \*ngSwitchCase="'bbb'">bbb</p>

  <p \*ngSwitchCase="'ccc'">ccc</p>

  <p \*ngSwitchCase="'ddd'">ddd</p>

  <p \*ngSwitchDefault>default</p>

</div>

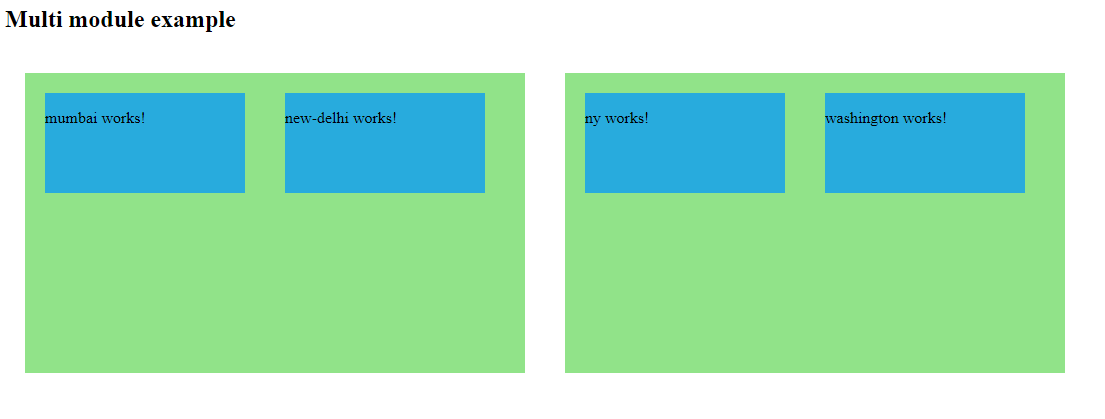
**Multi Module Example**

We are going to create a two modules(India, USA) which contain two components(India Component, USA Component) each and at last we will add two modules to App Module

India Component have two child components New Delhi and Mumbai whereas

USA Component has two components NY and Washington

1. Create two modules India and USA
2. Create 4 components
3. Add two modules to Main/App module
4. Declare two components in India module i.e. Delhi and Mumbai
5. Declare two components in USA module i.e. NY and Washington
6. Import above two modules to App Module
7. Add India component selector and USA component selector in app.component.html file



**Component communication**

Passing data from child component to parent component

## Pass data from Child to parent component

There are three ways in which the parent component can interact with the child component

1. Listens to Child Event
2. Uses [Local Variable](https://www.tektutorialshub.com/angular/template-reference-variable-in-angular/) to access the child
3. Uses a [@ViewChild](https://www.tektutorialshub.com/angular/understanding-viewchild-viewchildren-querylist-in-angular/) to get the reference to the child component

Let us look at each of those scenarios in detail

## Parent listens for child event

The Child Component exposes an [EventEmitter](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) Property. This Property is adorned with the [@Output](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) decorator. When Child Component needs to communicate with the parent it raises the event. The Parent Component listens to that event and reacts to it.

### EventEmitter Property

To Raise an event, the component must declare an EventEmmitter Property. The Event can be emitted by calling the .emit() method

For Example

|  |  |
| --- | --- |
| 1  2  3 | countChanged: EventEmitter<number> = new EventEmitter() |

And then call emit method passing the whatever the data you want to send as shown below

|  |  |
| --- | --- |
| 1  2  3 | this.countChanged.emit(this.count); |

### @Output Decorator

Using the EventEmitter Property gives the components ability to raise an event. But to make that event accessible from parent component, you must decorate the property with @Output decorator

### How to Pass data to parent component using @Output

In the child component

1. Declare a property of type EventEmitter and instantiate it
2. Mark it with a @Output Decorator
3. Raise the event passing it with the desired data

In the Parent Component

1. Bind to the Child Component using [Event Binding](https://www.tektutorialshub.com/angular/angular-data-binding/#Event-Binding) and listen to the child events
2. Define the event handler function

Reference link: <https://www.tektutorialshub.com/angular-tutorial/>