**The Basics**

* Decorators are typescript feature which allows you to enhance your classes, enhance elements used in code. Its always added with @ in starting.

**Role of AppModule and Component Declarations:**

* Module is a bundle of functionalities of our app in a packages.
* **Bootstrap** in AppModule basically tells angular that which component should be aware of when whole application starts up(which component should be recognize in index.html file and i.e, appComponent)
* **Declarations array** is used to register new components and **Imports** **array** allow us to insert new modules in our app.
* **Components are directives with selector and template**

**String Interpolation:** The only condition for it is :- Any expression which can be resolved into string at the end is used in string interpolation. {{ ‘any string’ }}. Cannot add **if , for** in it interpolation although **ternary operator** can be used. Also number and Boolean value can be used to display in interpolation.

**Important**: For events, you don't bind to onclick but only to click (=> (click)).

**Event Binding**: While event binding using $event will emit data on that event (give access to event data).

* **Directives:** are instructions in the DOM.
  + **\*ngIf** is a (\* indicates it is a)structural directive which changes the structure of the DOM

**Example**  for \*ngIf and else:-   
<p \*ngIf=”some condition”; else localReference> Some other text </p>

<ng-template # localReference > // local reference is a marker

<p> some text</p>

</ng-template>

*Unlike Structural directives, Attribute directives don’t add or remove DOM elements. They only change the element they were placed on.*

* + **ngStyle:**  is a *Attribute directives* where we define key value pairs while property binding it style name as key and value of style as value.

Based on this function the style (backgroundColor) will change

[ngStyle]= {

backgroundColor: getColor()

}

getColor(){

return this.serverStatus === ‘online’ ? ‘green’ : ‘red’

* + **ngClass :** is also used as a property binding with key value pairs in javascript object.
    - where **key** is a Class and
    - **value** is the condition based on which that class will be selected.

**Parent to Child and Child to Parent** data transfer example in code implementation.

**Local References(is of type element)**: can be placed on any Html element. Its written in the form of #yourChoiceofName.

This reference holds reference not only to its value but to whole html element with all its properties.

We can use these references anywhere in the template and not in the typescript

* If we want to get access to the local references in typescript before calling it in method, we can do that by @ViewChild()

In this type the local reference is of type ElementRef.

Do not set value of nativeElement value in above method(ViewChild) in typescript.Simple by using @viewChild #yourChoiceofName won’t work we need to give selector in @viewChild. So we can pass @viewChild(‘yourChoiceofName’) #yourChoiceofName like this

With this we will get access to local reference(#yourChoiceofName -> refer this with this.yourchoicename in typescript) in typescript file and then we can use it in our method.

* **Ng-content is a Directive:-** which lets you pass some(complex) code into the other component via <ng-content></ng-content> selector

**Eg:-**  <parent-component>

<div class="panel-body">{{serElement.content}}</div>

</parent-component>

<app-server-element

            \*ngFor="let serElement of serverElements"

            [element]="serElement">

            <div class="panel-body">{{serElement.content}}</div>

        </app-server-element>

<child-component>

<ng-content> </ng-content> :- this will show the data which we pass in parent component in div into child component.

</child-component>

  <div class="panel panel-default">

                <div class="panel-heading">

                    <div [ngClass]="{

                        'red-text': element.type === 'server',

                        'blue-text': element.type === 'blueprint'

                    }">{{element.name}}</div>

                </div>

                <ng-content></ng-content>

            </div>

**Component Lifeycle Hooks**

* **ngOnChanges:**  called after bound input property changes.
* **ngOnInit:** called once the component is initialized.
* **ngDoCheck:** called during on every change detection run(on every check / on any event also triggered to check whether something is changed or not).
* **ngAfterContentInit:** called after content (ng-content)has been projected into the view(parent component).
* **ngAfterContentChecked:** Called everytime the projected content has been checked.
* **ngAfterViewInit:** called after the components view (and child’s view ) has been initialized
* **ngAfterViewChecked:** called everytime the view has been checked.
* **ngOnDestroy:** called once the component is about to destroyed

**How to acces @viewChild in Lifcycle hook:**

**It can only be used in afterViewInit hook.**

**afterViewInit** gives you access to template element, which can be accessed then and used in code.

Until this hook has reached you **can’t check the value of template element in DOM** bcoz it hasn’t been rendered yet.

For **ContentChild**, the same adjustments as for **ViewChild** apply: <https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/14865241>

**Getting access to ng-content via @ContentChild**

Whatever code is passed to ng-content ,if I place a local ref in it and want to access it in child component or want to access inside the component where it is passed on ng-content selector use **@ContentChild**

If I want to access local reference in child component can use @ContentChild just like @ViewChild

And this value can be accessed once we reach the ngAfterContentInit lifecycle hook.

To get access to content which is present in Component A and passed it to Component B via ng-content we use @ContentChild

**Directives**

* **Attribute directives:** looks like a normal html attribute (possibility with property binding or event binding).

**Eg: ngClass / ngStyle**

Only affect or change the element they are added to.

* **Structural directives :** have a leading \* . Affects the whole DOM (elements get added or removed like \*ngIf)

**Eg:** **\*ngIf / \*ngFor**

* **The \* symbol on structural directive is converted by Angular(**behind the scene**) into <ng-template [ngIf]=”condition”> some random code </ng-template>**

**Creating Attribute Directive:** [Angular - Attribute directives](https://angular.io/guide/attribute-directives#building-an-attribute-directive)

we used the Angular Renderer class to change the style of a HTML element. As explained in that lecture, you should use the Renderer for any DOM manipulations.

For more info visit this link : [Angular - Renderer2](https://angular.io/api/core/Renderer2)

With the @[HostListener](https://angular.io/api/core/HostListener)() decorator, you can subscribe to events of the DOM element that hosts an attribute directive

@HostBinding allows us to bind to the property on which directive is placed on. For more info google 😛

**Services and Dependency Injection**