Angular Fundamentals:

__/__/__

→ Building Blocks of Angular App :-

* Components
  ↳ Data
  ↳ HTML Template
  ↳ Logic

→ An application consists of basic components which are reusable or static.
→ These components combined together under some logic forms an application.

* Modules :-
  ↳ containes of group of related components
  ↳ main is app module we get in start of application.

→ Components :-

* These are created inside src/App.
* Components are of type class → (exported class)
* We have to add a decorator to define it as component.

```
@ Component ({
Keyword to select Component. ← selector : " "
HTML Code ← template :
})
```

* Now this component must be added in app.module.ts inside declarations.

* To create a component using Angular CLI :-

```
npx ng g c <component name>

Or

npx ng g c <"path/"comp. name">
```

g ↙ generate, c ↘ component

→ Templates :-

It is a property of component decorator & can be hardcoded or binded with components data or fuctions

Example

```
({
    Selector : " ",
    Template : '<h2>{{ }}</h2>'
})
```

Data

↓

this can be defined in class.

→ Directives :-

↳ to maniplulcate the dom < html code > as code

Eg *ngFor

<Li *ngFor=" Lex Vor of List "></li>

* this Li will be rendered n times where n is length of the list.

→ Services :-

* When we are defining business Logic within the component, it creates a tightly coupled component.

* To solve this issue we can create a service and use that is a component . Eg HTTP Service

Naming convention → "name". service. ts

* To define a Servive :-

```
export class "name"Service {

        //.... Logic .
}
```

→ Dependency injection

* For using the service we have to create its object & then fetch its dependencies. This will tightly couple the code.

* For this we can use dependcy injection of a service by declarg it inside constructor only.

Eg
```
constructor (service : < your servce class>)
{
    //...  }
```

* Now we need to register this service class in providers. (app.module.ts)

* To generate service using CLI :-

```
npx ng g s <sevice name>
```

* @injectable :- Its a decorator for service class only if it has services has dependencies in constructor.

—/—/—

→ Data & Event Handling :-

* Property Binding :-

→ we bind property of dom element with values defined in .ts

Eg → <img src={{ vas }} />

* Attribute Binding :-

→ As most html attribute are not in DOM, so we need to use attribute of HTML to bind data.

Eg → [attr.{property} = {{ value }}]

* Class Binding :-

<tag class = " [class.<<class name>>] = "<<value>>" "

this will be Defined in .ts

* Style Binding :

< tag [style.<<property>>] = " data / value " />

Same defined in .ts

__/__/__

imp *** Event Binding :-

Using this we can bind the event of a DOM Element accordingly

Eg → when a button is clicked, we can capture and can trigger a func̄tion

$$\underbrace{\text{< button (click)}}_{\text{event}} = \underbrace{\text{"func()"}}_{\text{\$event can be passed}} \text{ />}$$

→ $event ?
↳ it will represent standard / custom event of the DOM

→ event bubbling :-
when 2 elements are triggred by ~~only~~ either's trigger is known as event bubbling.

$event. stop Propogation()
↳ to stop bubbling

→ Key up Event :-

$ event. Keycode can be used.
↙
filtering on event property.

_/_/_

→ Template Variable :-
To not use event.target.value each time, we can define a template variable and can pass it's value

Eg <input #email (keyup.enter) = " func(var.value)/>
$\downarrow$
variable

* 2-way Binding :-

<input [(ngModel)] = "email" />
↳ banana in the box

→ now this email variable is 2-way binded & can be changed from both DOM & typescript.

* Import Forms module in app.module.ts

* PIPES :-
↳ uppercase, lowercase, Decimal, Currency, Percent.

{{ course.title | <pipeline>> }}
↳ data will be shown according to pipeline we use.