**Angular**

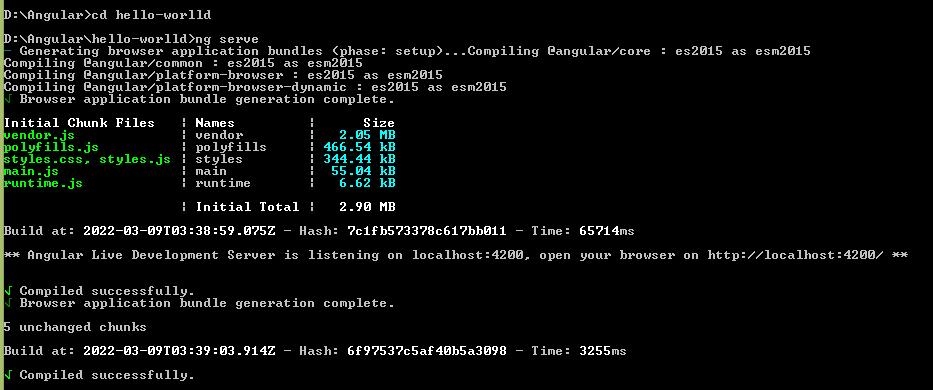
# Why Angular

* ***Frontend,*** (also called UI) is what we see on web page or website and validations to restrict or help for valid data. Built with HTML, CSS, java script (for validation and communicate with backend).
* ***Backend***, (what exactly business logic is and what it need to do) is what happened if a link is clicked or button is pressed or form is submitted, which is out of focus for website user. Several actions performed and respective intended operation completed like delete, create, update, fetch etc. Flow involves, backend validations, hitting data base (where all data resides).
* Any web application is combo of front end + backend. Many of existing old websites built using HTML + Java script or jQuery. If java script still existed what is need of building framework around it.
* Angular deals with front end, and just communicate with backend to pass request and get response from backend.
* Since websites are not stable; means, many functionalities or features added and removed as part of business requirement, maintenance is very difficult with plain vanilla java script.
* To make maintenance easy many java script design patterns (proven solution for particular kind of problem) used which makes life of developer’s worst ☺ and testing also tedious for large scale web applications with plain java script.
* So over past few years many java script frame works like Angular, React, Vue, expressjs etc. built and used to achieve above stated problems with large scale web applications.
* Framework gives advantage to application like structural, easy decoupling, and easy maintenance.

# Setting up development environment

* Mandatory is latest stable version of node JS(chose stable version not latest)
  + <https://nodejs.org/en/download/>
  + Why node required is, Node will make creating, running and deploying Angular apps super easy.
  + Once installed open command prompt and run
    - node --version
  + Node provides NPM (Node Package Manager) useful to install third party libraries.
    - Third party libraries like Angular, react etc.
  + We use Angular CLI (Command Line Interface) to create angular applications, and it reduce manual effort on creating so much of boiler plate code.
    - npm install -g @angular/cli
    - to make sure with installation ng --version

# Creating First Angular Project

* To create new project
  + Make sure the right folder selected to create code in CMD.
  + ng new hello-world
    - Takes more time to download required files. Around 15 to 20mins also based on Network speed ☹☹
  + So till now all skeleton code created and now need to add our own functionality on it.
  + We can start coding with note pad or word pad also.
  + But there exists more developer friendly code editors called IDEs.
  + One such IDE is Visual studio code.
    - <https://code.visualstudio.com/download>
  + Angular skeleton is always runnable without any issues. To run the application
    - ng serve
  + 
  + To see how web application looks like, go to browser type
    - localhost:4200/
    - If any change made to application in DEV those, the new changes will not reflect automatically and need to restart application every time.
    - Installing webpack will solve issue
      * **npm install -g @angular-devkit/build-web-pack**

# Project and folder structure

* Once project opened in Visual studio code, it organised in different folders
  + **e2e** End to End test folder
    - **but when I created application e2e folder not created ☹**
    - End to End testing is set of test cases which simulates actual user and test cases like
      * Opening code in browser
      * Clicking on a button
      * Fillout some form etc.
    - This folder contains 4 files
      * app.e2e-spec.ts
      * app.po.ts
      * tsconfig.e2e.json
  + **node\_modules**
    - node\_modules contains all the third party libraries on which the application depends on.
    - Remember, while deploying application node\_modules folder not going to be taken with deployment.
    - All the required dependencies and their versions bundled into package.json file and while installing application on server, the package.json file passed to some script and the script will read the file download required dependencies.
  + **Src**
    - Contains actual source code of angular application.
    - This folder contains other sub folers.
      * **App**
        + Contains 5 files

App.component.css

App.component.html

App.component.spec.ts

App.component.ts

App.module.ts

* + - * + UI of application is no single chunk, it is combination of different parts. Each logical part is called component. Also each functionality is also defined as components.
        + **Component.html** contains, logic that display data to user
        + **Component.ts** contains validation logic, backend connectivity logic etc.
        + **Component.spec.ts,** is for testing.
        + **Module.ts, will update ☺**
      * **Assets**
        + Assets needed for application like, image files, text files or icons will go into assets folder.
      * **Environments**
        + configuration settings for different environments

example, test environment will not use PROD DB server, not to be deployed to PROD application server, vice versa with PROD.

These setting s managed with configuration files

Environement.prod.ts 🡪 prod configurations go here

Environment.ts 🡪 DEV configurations go here

* + - * **Favicon.ico** 🡪 Icon to be displayed browser
      * **Index.html 🡪** contains angular application and displayed to browser
      * **Main.ts 🡪** starting point of application, likes boot strapping starting module of application.
      * **Polyfills.ts:**
        + Which contains some scripts that are required for running angular application
        + **Polyfills** fills gap between java script features required by angular application and the java script that supported by browser.
      * **Style.css** 🡪 styles required at application level.
      * **Test.ts 🡪** setting up test environement
      * **Angular.json 🡪** standard angular CLI configurations
      * **Editor.config 🡪**  for entire team, similar editor configurations to be use, those settings will be uploaded here.
      * **Gitignore🡪**
        + list of files to be ignored to push to git repository
      * **karma.conf.js 🡪**
        + configuration file for testing java script code
      * **package.json 🡪** determines list of files that application requires basically maintains dependencies list.
      * **Tsconfig.json 🡪**
        + contains settings for type script compiler.
        + Based on the settings typescript code compiled into java script code
      * **Tslint.json 🡪** static analysis tool for type script code. Checks readability, functionality and maintainability errors of typescript code based on this file.

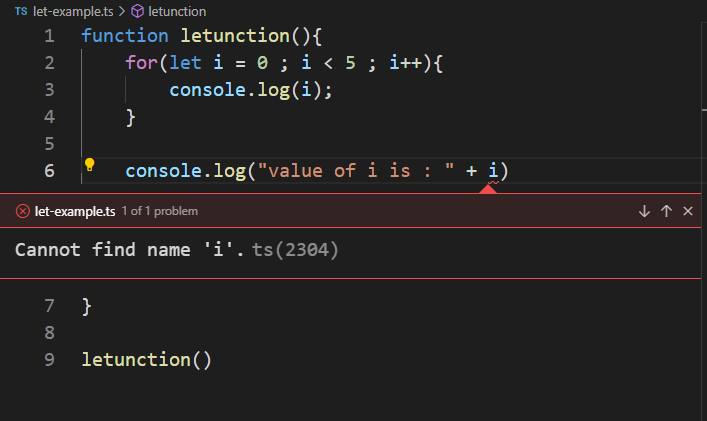
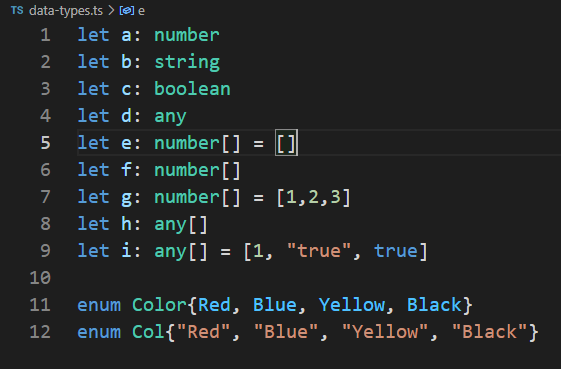
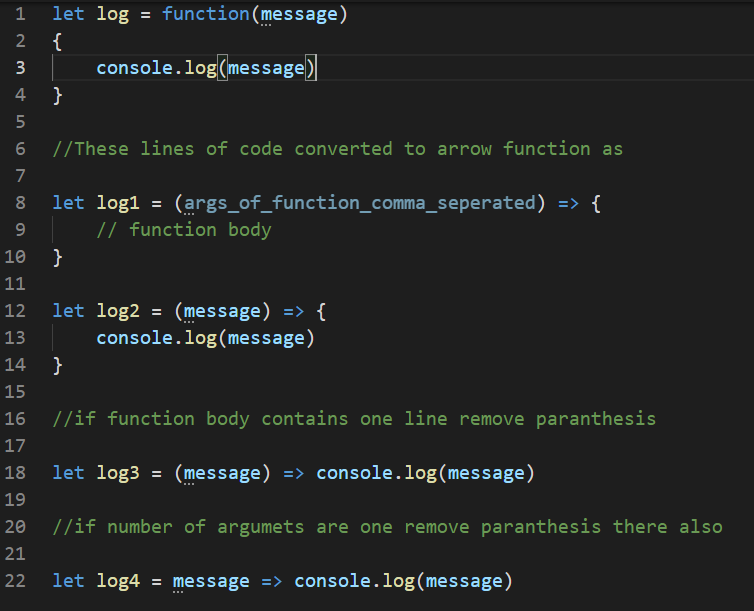
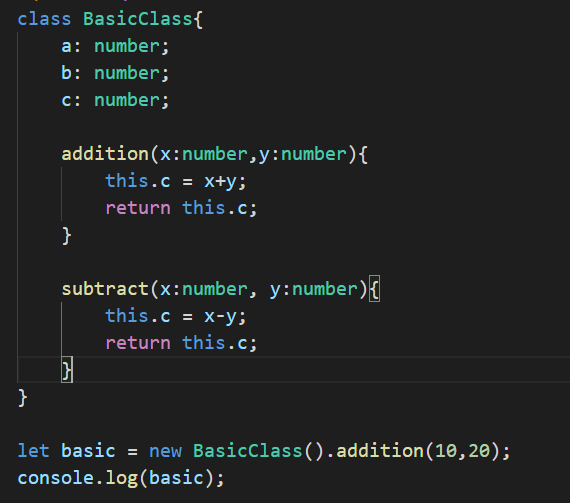
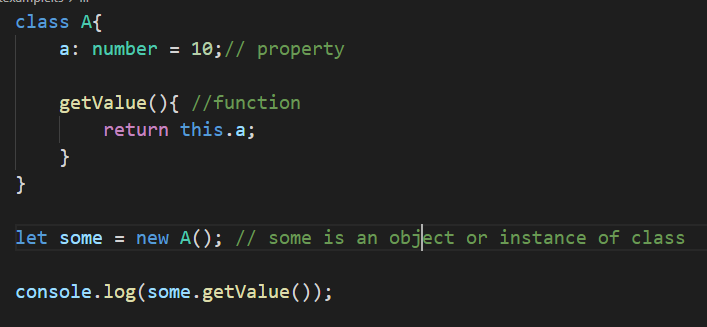
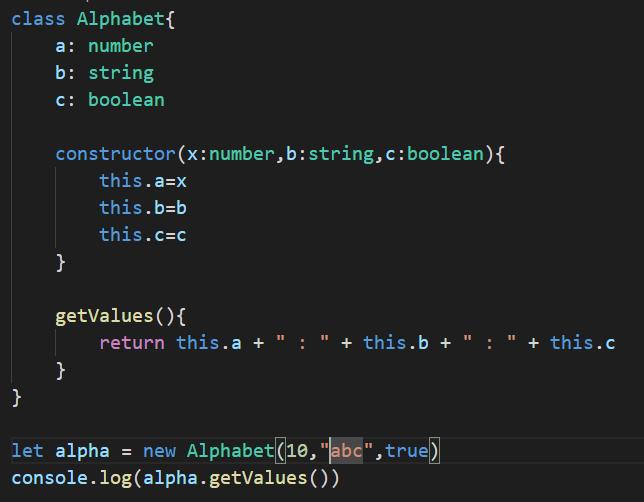
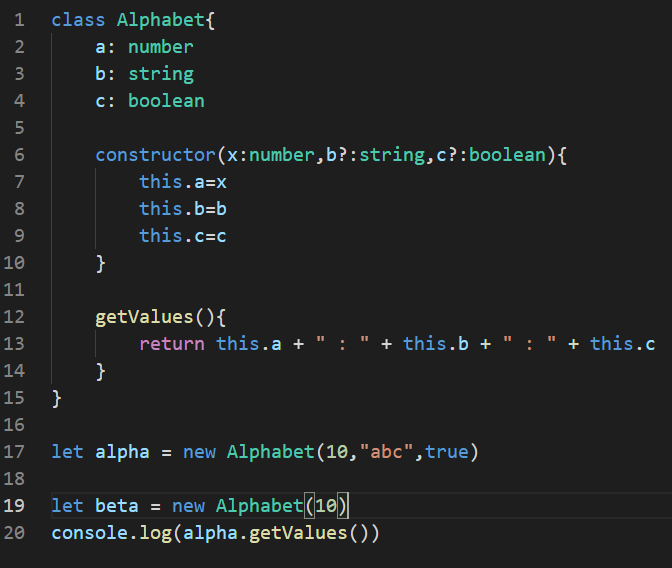
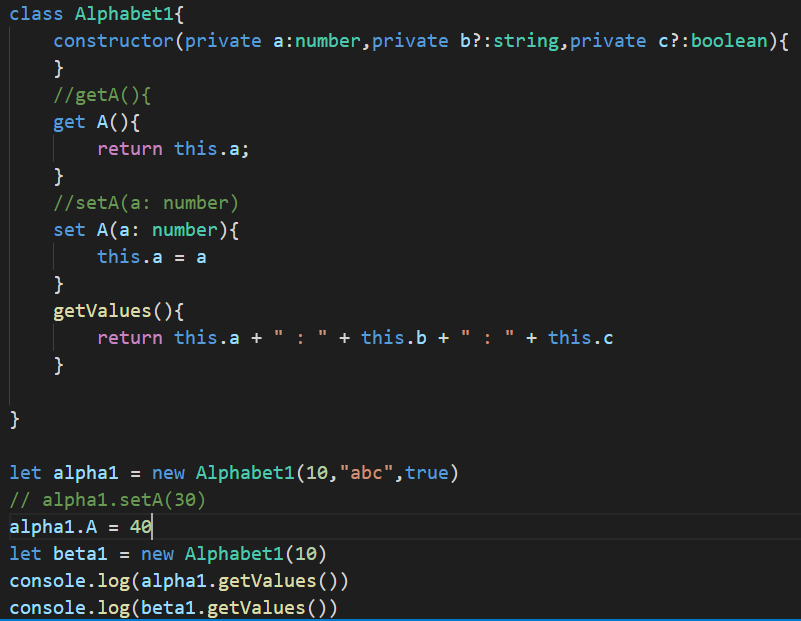
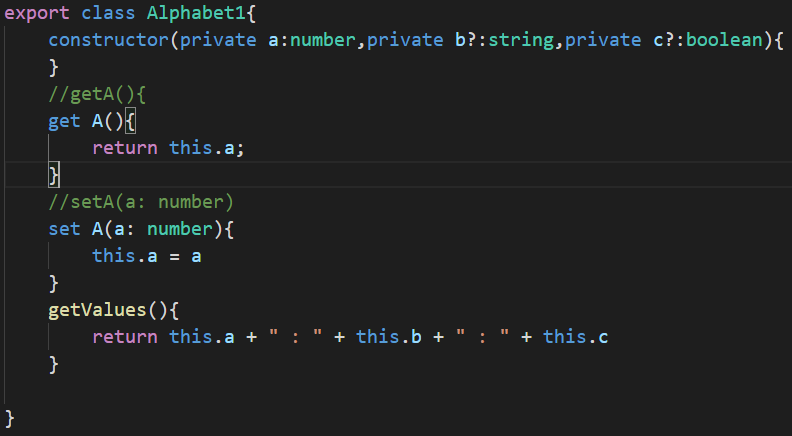
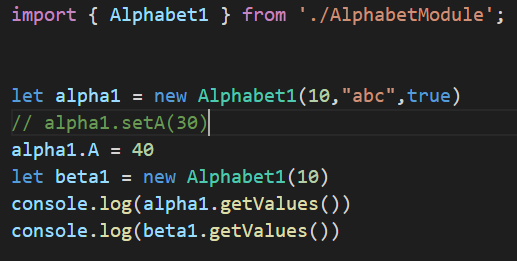
# Typescript fundamentals

* What is Typescript?
  + Superset of java script, which contains all JS features including additional features.
  + **Strong typing:**
    - Variable must tagged with data type.
    - This feature is optional in type script also but better to use
    - Object oriented features: Generics, access specifiers, constructors etc.
    - compile time errors
    - great tooling
  + Browsers can’t be able to understand type script so type script code need to transpiled (like compile) into java script, (it will happen automatically) while building application.
* **Installing type script**
  + npm install -g typescript
  + check version tsc --version
  + to compile
    - tsc file\_name.ts
  + new file will be generated as file\_name.js
    - function hello(name){

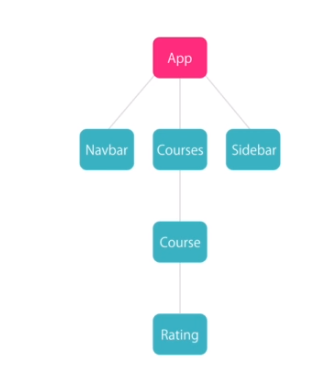
console.log(“Hello “ + name)

}

var some = hello(“Sunil”)

* to run generated JS file
  + node file\_name.js
* **var and let variables:**
  + the scope of ***var*** variable is scoped to nearest function
  + scope of ***let*** is limited to nearest block
  + 
  + here I is limited to for block and not accessible outside for block
* **data types in type script**
  + 
  + In enum, each value holds index value starting from 0, if any new value inserted between two values, index values get incremented
* **Type Assertions in type script**
  + Same as type casting, it will not change value of variable in memory; just tell compiler what type of data in variable.
  + 
* **Arrow functions:**
  + let something = (message) => console.log(message);
    - 
* **Interfaces**
  + In some places, there is need to create method with many parameters.
  + So passing those many parameters might cause wrong ordering of values, missing values or wrong types etc.
  + Create an interface with all those values and pass that particular interface as an argument to method
  + 
* **Classes:**
  + Class is a template or container which will group properties (variables) and methods (functions) together.
  + Need of grouping is, cohesion should be high, properties of class should not be accessed outside class directly, they must accessed thru methods.
  + 
  + **For default datatypes, memory allocated automatically.**
  + **For classes or interfaces memory need to be allocated using new operator**
* **Object:**
  + Object is instance of class
  + Example is easy ☺, Human is class, different persons are objects of Human class
  + Generally class will contain, properties and function :
    - For human properties are eyes, nose
    - Functionalities include, watching and breathing
  + So each object will have same properties as class, since those are instances.
  + 
* **Constructors:**
  + To assign value to properties of class, either we can directly assign them in class, disadvantage is properties always take same value which is not correct.
  + Another way is using object, value can be assigned to property of class
  + If 100 properties there, 100 lines of code to be added.
  + there comes concept of constructor.
    - Constructor is same as function but it won’t return any value and its only functionality is to assign values to properties of class
  + Using constructor values to properties, can be assigned at time of object creation
  + 
  + Properties of class must be referred with this key word.
  + ***this*** keyword represents current instance of variable.
  + If try to create object without passing properties, error will be thrown because passing properties is mandatory for object creation in above scenario.
  + To make properties for constructor as option follow below notation
  + 
  + observe **question mark next to b and c in constructor**
  + **Mandate rule for optional parameter is, if b made as optional parameter, c should also be made as optional (i.e.) if parameter passed as optional all the right side parameters should be passed as optional.**
* **Access Modifiers:**
  + Used to avoid data leakage with properties of object.
  + Also used to restrict access of properties directly from outside class.
    - **Public 🡪 properties can be accessed outside of class**
    - **Private 🡪 properties can’t be accessed outside of class**
    - **Protected**
* **Properties:**
  + If constructors used in program, no need to declare properties explicitly.
  + Declaration of properties can be done as constructor arguments.
  + 
  + If variables declared as private they can’t be referred outside, we know it ☺
  + In any OOP language, along with constructor, getter and setter methods used to access and pas values to properties of class. Type script will also allow the same.
  + 
  + two ways getters and setters can be declared
    - **getA() and accessed as obj\_name.getA()**
    - **get A() and accessed as obj\_name.A**
* **Modules:**
  + All the code can’t be able to written to single same file.
  + Each file can be viewed as module
  + Main purpose of any module is to reuse or refer in any other program or module or file.
  + To use file in other place **make class as export and use import keyword in program where module needs to be used.**
  + 
  + 

# Angular Fundamentals

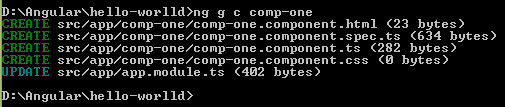
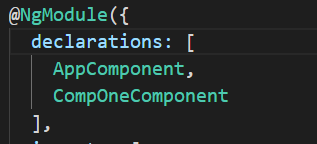
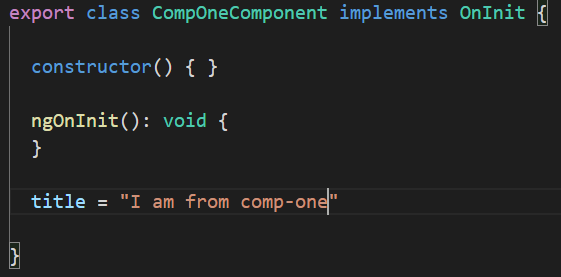
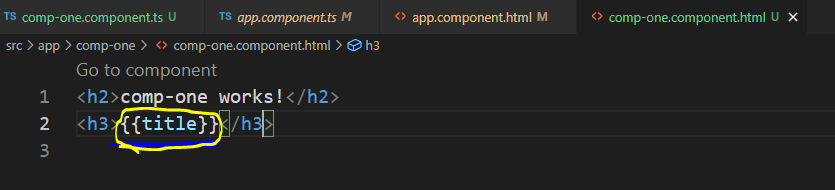
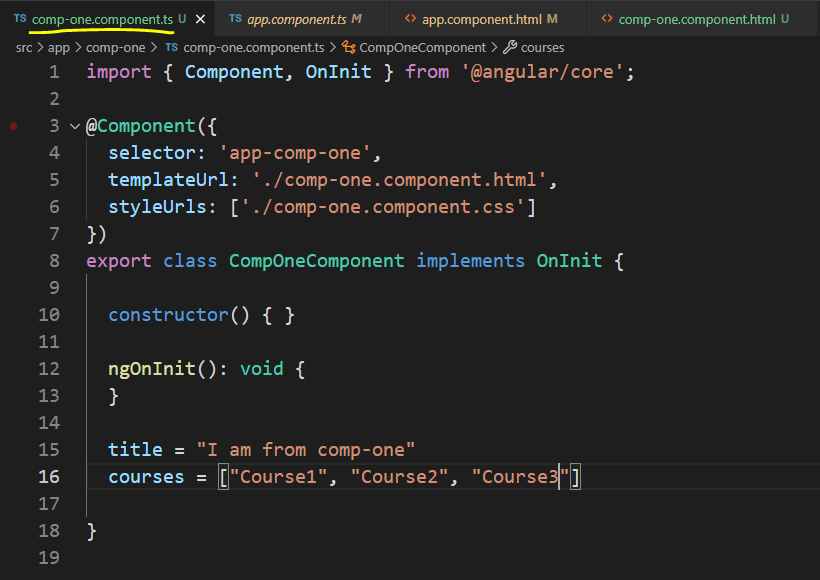
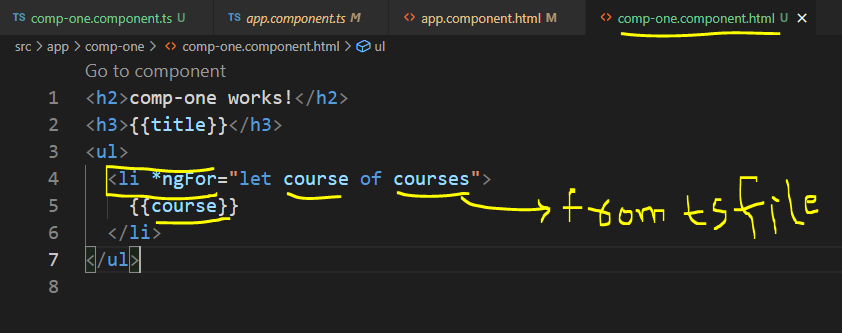
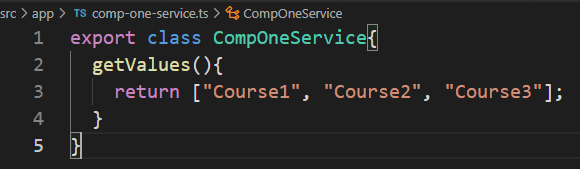
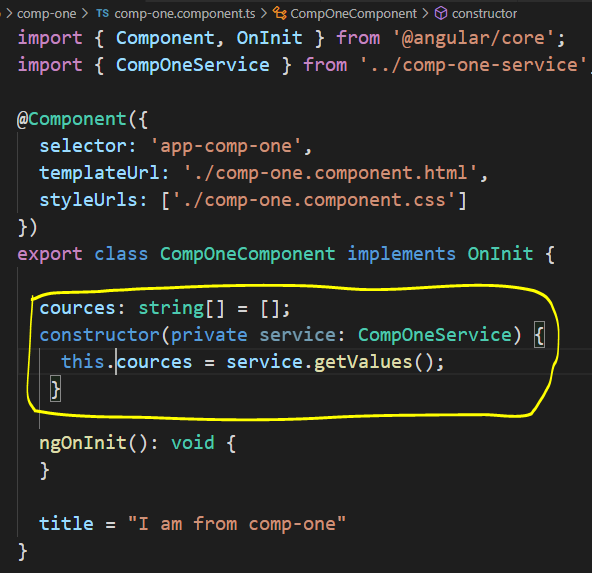
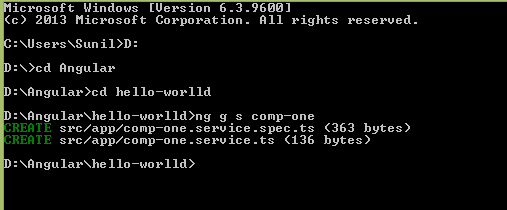
* **Basic building blocks:**
  + **Components**
    - Component = Data + HTML Template + Logic for screen which user sees
    - Every application has at least one component called APP component or root component
    - Generic angular application looks like
    - 
  + **Module:**
    - Module is container with group of related components
    - Example of analogy is super market
      * Super market as complete application
      * Different sections : Vegetable section, groceries section, beverage section as modules
      * Each item in section as component
* **Components:**
  + Create component 🡪 register component with module 🡪 add component into HTML mark-up
  + To make any class as component need to have decorator (annotations in java) called @Component on top of class and export the class. @Comnponent will imported from “@angular/core”. **Import {‘Component’} from ‘@angular/core’**
  + **Creating component:**
    - **@Component** will take an object argument. The object contains one or more properties which will tell how the component will work.
    - @Component({

**selector:’courses’, 🡪 How to include component into another HTML, tag name**

**template:’<h2>Courses</h2>’ 🡪 actual html used for the component**

**styleUrls 🡪 CSS files to be used for styling component**

})

* In general html will never be a single line of code on any given day ☺ so need to put all HTML code into separate file and include it in template.
* **Registering Component with module:**
  + In any module, there is section called declaration. Every component created in module need to added to declaration section
* **Add component (selector) to HTML:**
  + <selector-value></selector-value>
  + The template related to the selector value get rendered in HTML file where selector-value included.
* The manual approach is tedious if the application involves more components : involves more boiler plate code (creating class, add selectors, template URL, style URL, adding to module etc)
* Using angular CLI will remove all this boiler plate code.
* Command to create component
  + Ng generate component component-name
  + Ng g c component-name
* 
* **After successful execution of ng command for creating component, 4 files will be generated**
  + .html, .ts, .css and spec.tss
  + .html is HTML content rendered for the component
  + .ts is class component
  + .css is styles
  + Spec.ts id test file
  + App.module file also updated with component without manual touch
  + 
* **Template:**
  + For visualising data from component
  + **Data binding:**
    - **Interpolation**
    - {{ var\_name }} 🡪 this variable will come from component.ts file.
    - 
    - 
* **Directives:**
  + many directives available, covered in later part ☺. For example take NgFor
  + To display list of items of component to respective template. Need to loop over list and display items in template.
  + 
  + 
* **Services:**
  + Many times data for component come thru API calls or querying database.
  + Two places the API call logic or DB query logic can be place:
    - Inside same component
      * Problems with this approach are,
        + The component is tightly couple with backed call logic which is tedious for unit testing
        + If logic kept inside component, it can’t be reused
    - Created logic inside separate class and use it in target component
  + Like Component there is **no need for any decorator in service.**
  + **Dependency Injection:**
    - Injecting necessary dependencies of class into constructor
    - In case of services, service creates as separate class and get used in component.
    - Plain scenario is using new operator objet create for class and use in component.
    - But problem is tight cohesion with, outside class inside component. If constructor of service class changed all components using the service get impacted.
    - **Dependency Injection** will solve these problems.
    - 
    - 
    - Dependency injection will not work if the service is not listed in module.ts file.
    - Module.ts file contains, **Providers** section, there all services need to be registered.
    - Instead of having service file created, and adding to module file manually angular will generate service with ease ☺
      * Ng g s comp-one
    - 

# Displaying data and handling events

* **Property binding:**
  + {{ }} 🡪 binding property of DOM element to the field of source component
    - Eg
      * Template: `<img src = {{src}} />`
      * Inside component
        + src = <http://someimageurl>
      * when template rendered into HTML, actual image will be displayed
  + property binding comes with []
    - whenever value in component changed automatically update in template
    - <img [src] = “src” />
    - So when to use string interpolation and when to use property binding
      * When dealing with displaying string components, or adding dynamic values for headings go with interpolation ( {{ }} )
      * For binding properties of elements or headings then property binding
        + <h2 [textContent]=”title”></h2>
* **Attribute binding:**
  + **DOM vs HTML**
    - **DOM** is model of objects that represent document. It’s essentially tree of objects in memory
    - **HTML,** is markup language that used to represent DOM in text.
    - **99% of times,** HTML elements have one-to-one mapping with properties of DOM objects
    - **In property binding,** component fields mapped to properties of DOM objects not HTML elements
    - **If try to map properties of components to HTML properties and if the property is don’t have mapping with DOM element, throws error.**
    - For example, “**colspan**” colspan don’t have mapping with DOM
      * **<td [attr.colspan]=”colSpan”></td>**
* **Adding Bootstrap:**