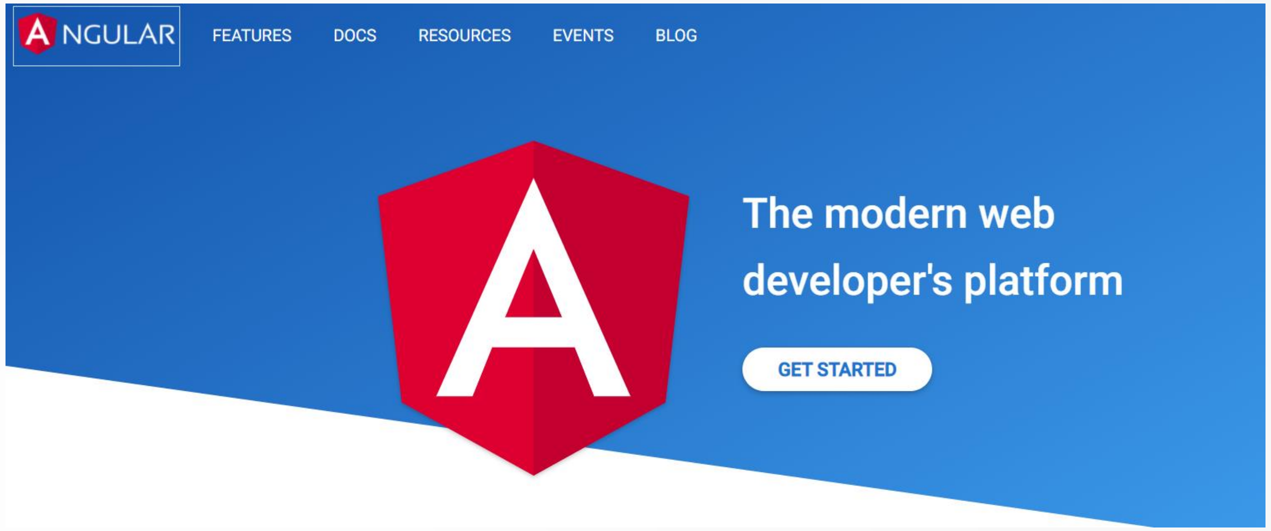
# MKJ IT Learnings

Corporate Training Company

www.mkj-it-learnings.com



# Angular

## Creating Front End

*(Always latest version)*

## About us

MKJ IT Learning is the corporate training firm, delivered enterprise level training across the globe.

The instructor of the training is
Ashish Bansal
Having 10+ years of experience in corporate trainings and project consultancies

https://www.mkj-it-learnings.com/online-training-team-profiles

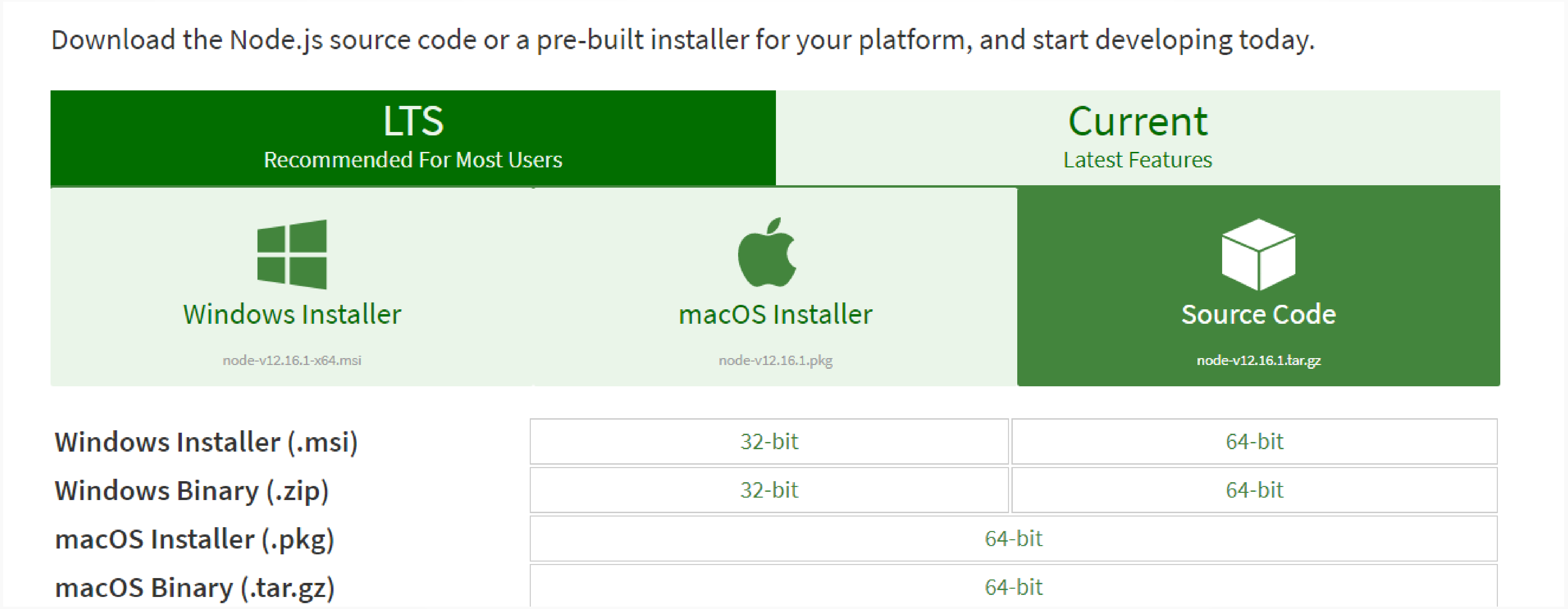
## ES6 & Typescript

https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

TypeScript (TS) is an open source programming language developed by Microsoft. It's a superset of JavaScript and adds optional static typing and classes.

# Setting up Development Environment

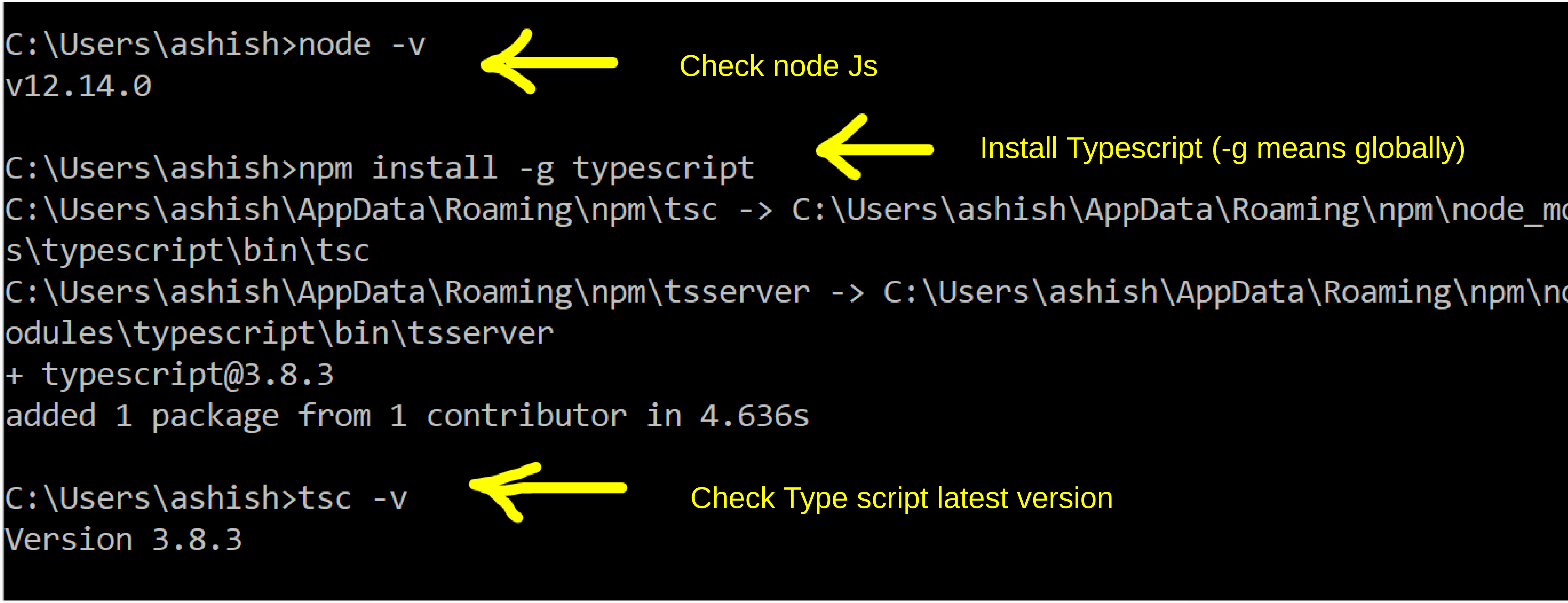
1) Download Node .js



Download the Node.js source code or a pre-built installer for your platform, and start developing today.

| | | |
|---|---|---|
| Windows Installer (.msi) | 32-bit | 64-bit |
| Windows Binary (.zip) | 32-bit | 64-bit |
| macOS Installer (.pkg) | 64-bit | |
| macOS Binary (.tar.gz) | 64-bit | |

# Continue…

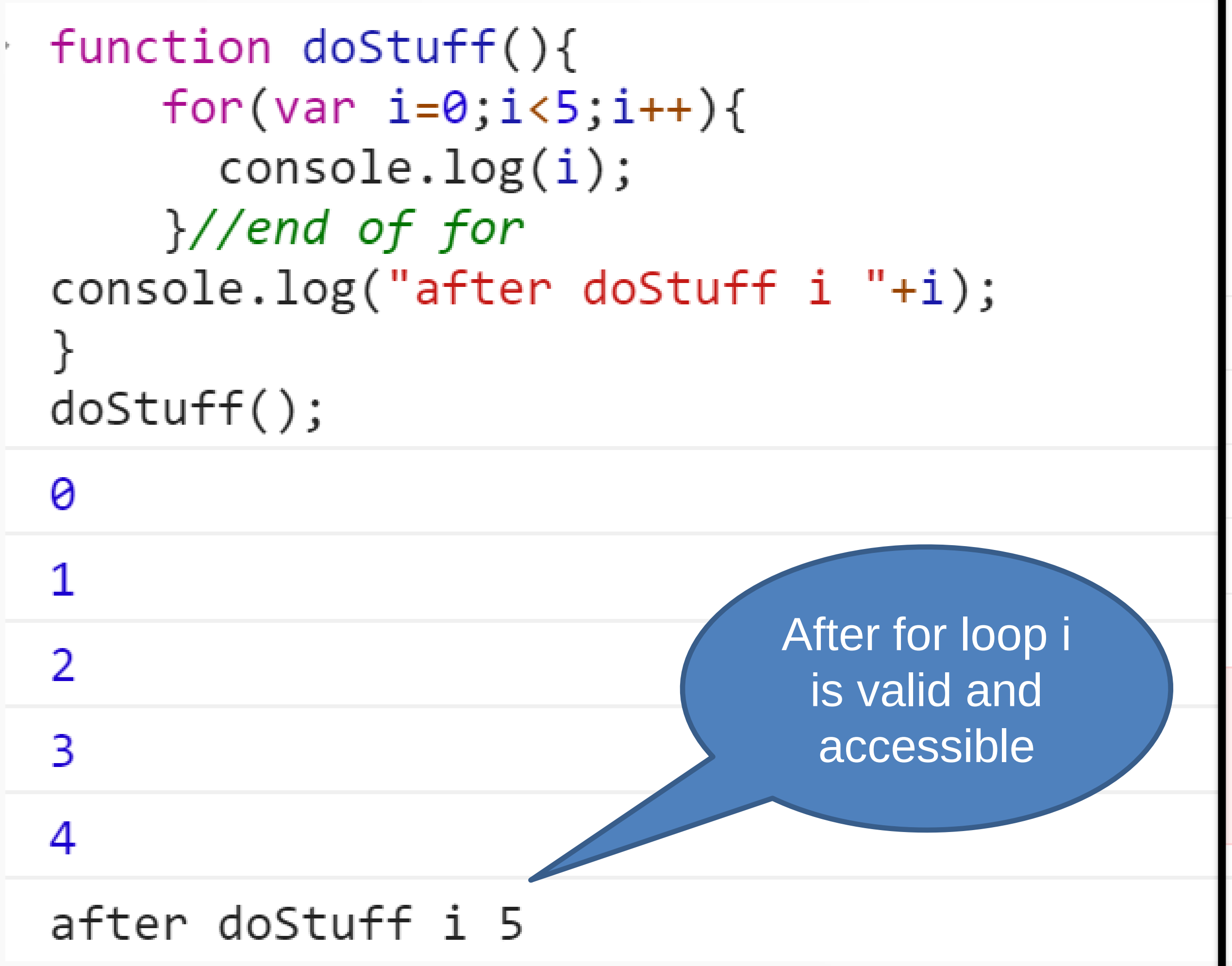


```
C:\Users\ashish>node -v
v12.14.0

C:\Users\ashish>npm install -g typescript
C:\Users\ashish\AppData\Roaming\npm\tsc -> C:\Users\ashish\AppData\Roaming\npm\node_mo
s\typescript\bin\tsc
C:\Users\ashish\AppData\Roaming\npm\tsserver -> C:\Users\ashish\AppData\Roaming\npm\no
odules\typescript\bin\tsserver
+ typescript@3.8.3
added 1 package from 1 contributor in 4.636s

C:\Users\ashish>tsc -v
Version 3.8.3
```

Check node Js
Install Typescript (-g means globally)
Check Type script latest version

# Advance Java Script ES6

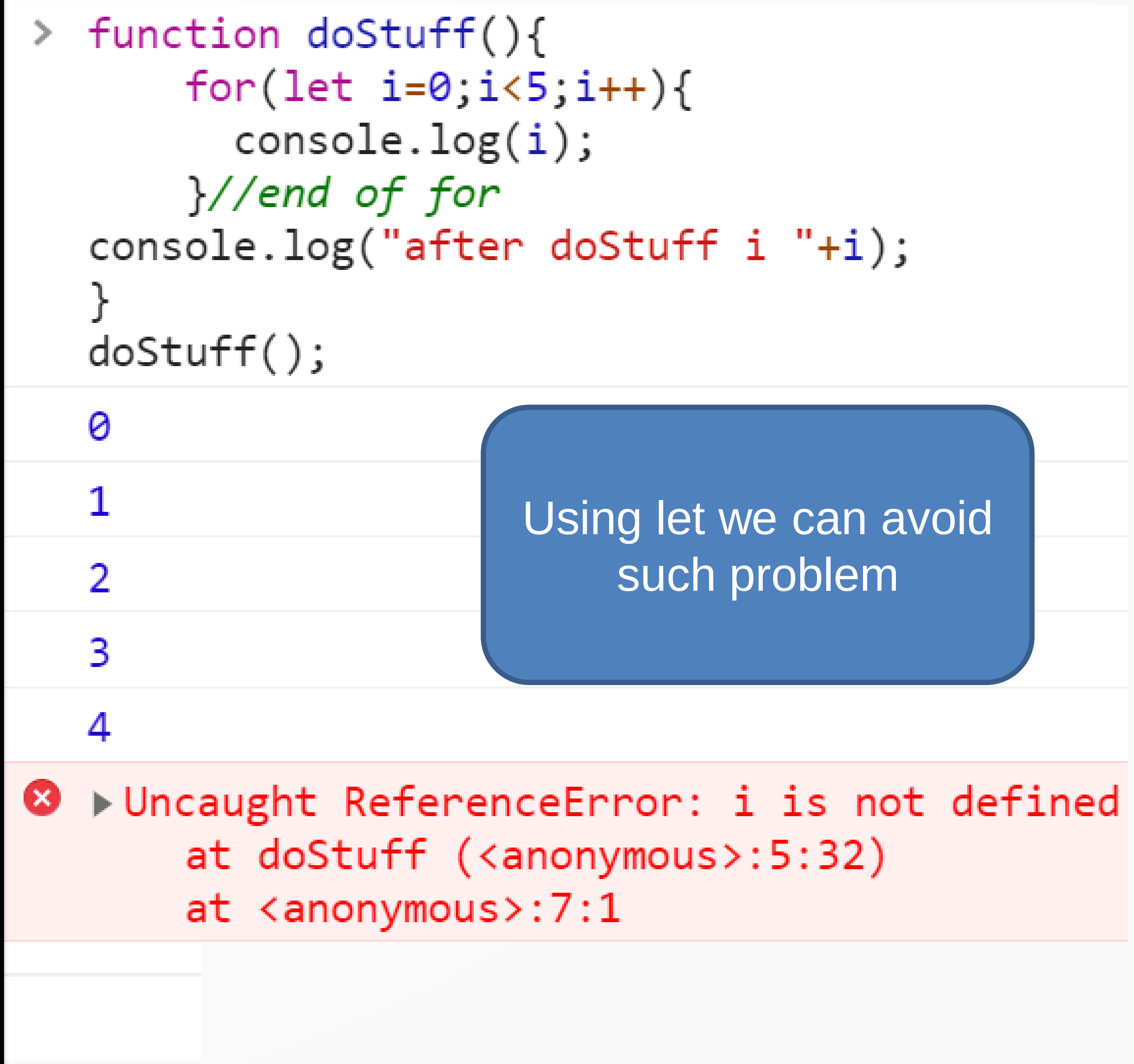
## Var vs let



```
function doStuff(){
    for(var i=0;i<5;i++){
      console.log(i);
    }//end of for
console.log("after doStuff i "+i);
}
doStuff();
```

```
0
1
2
3
4
after doStuff i 5
```

After for loop i is valid and accessible

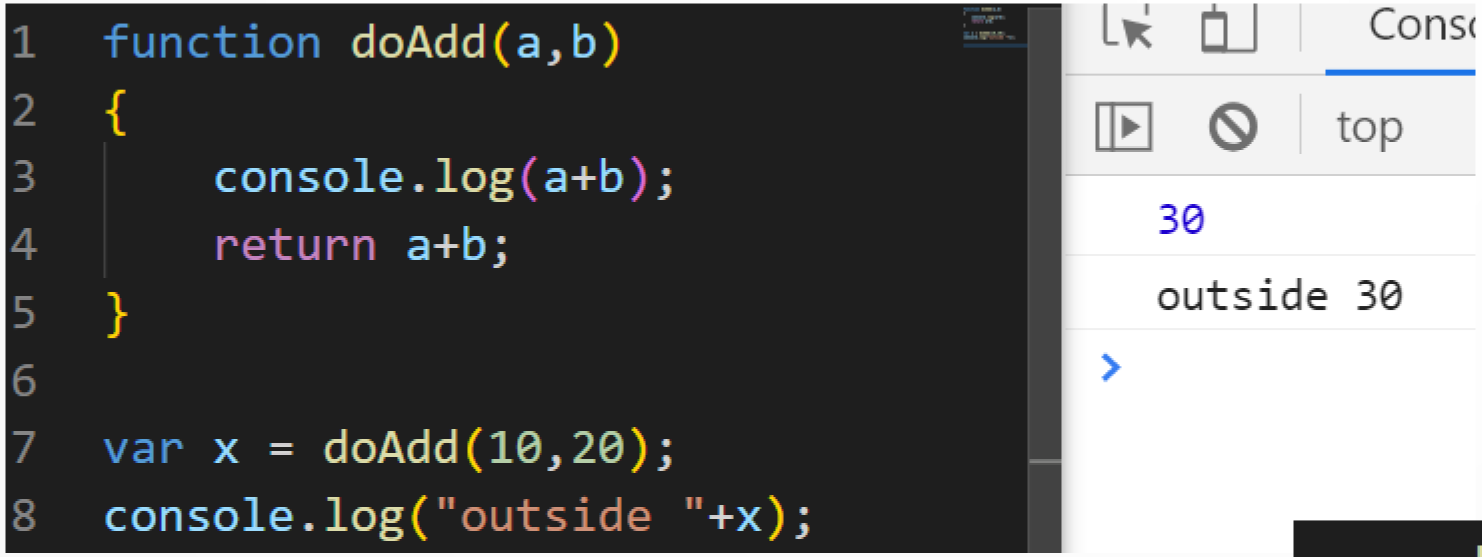


```
> function doStuff(){
      for(let i=0;i<5;i++){
        console.log(i);
      }//end of for
  console.log("after doStuff i "+i);
  }
  doStuff();
```

```
0
1
2
3
4
Uncaught ReferenceError: i is not defined
    at doStuff (<anonymous>:5:32)
    at <anonymous>:7:1
```

Using let we can avoid such problem

# Functions

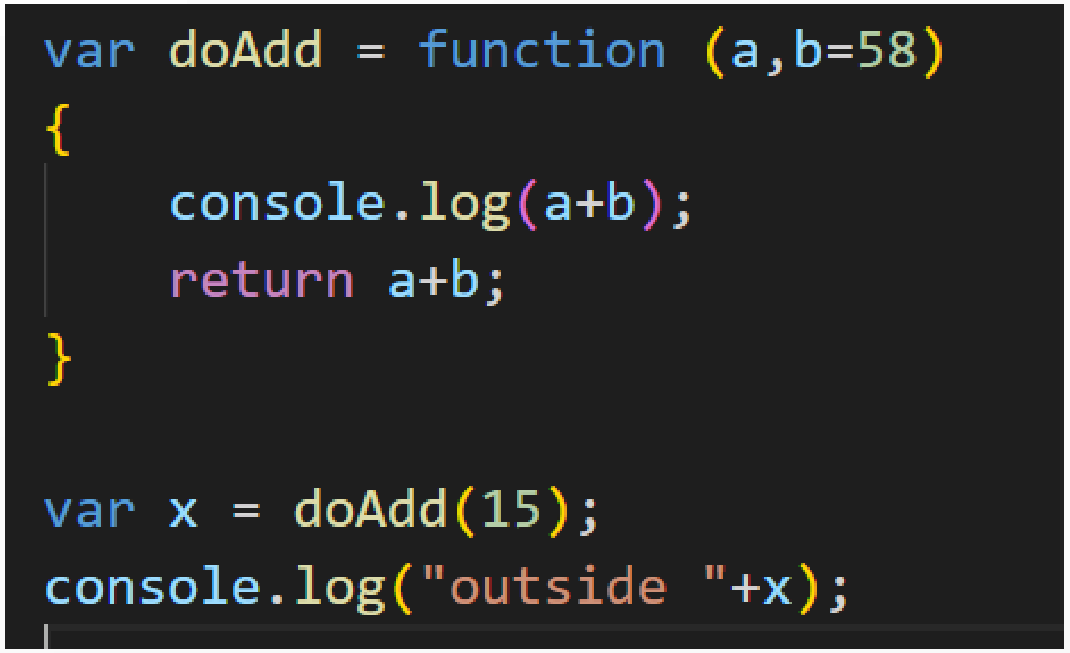


```
function doAdd(a,b)
{
    console.log(a+b);
    return a+b;
}

var x = doAdd(10,20);
console.log("outside "+x);
```

```
30
outside 30
>
```

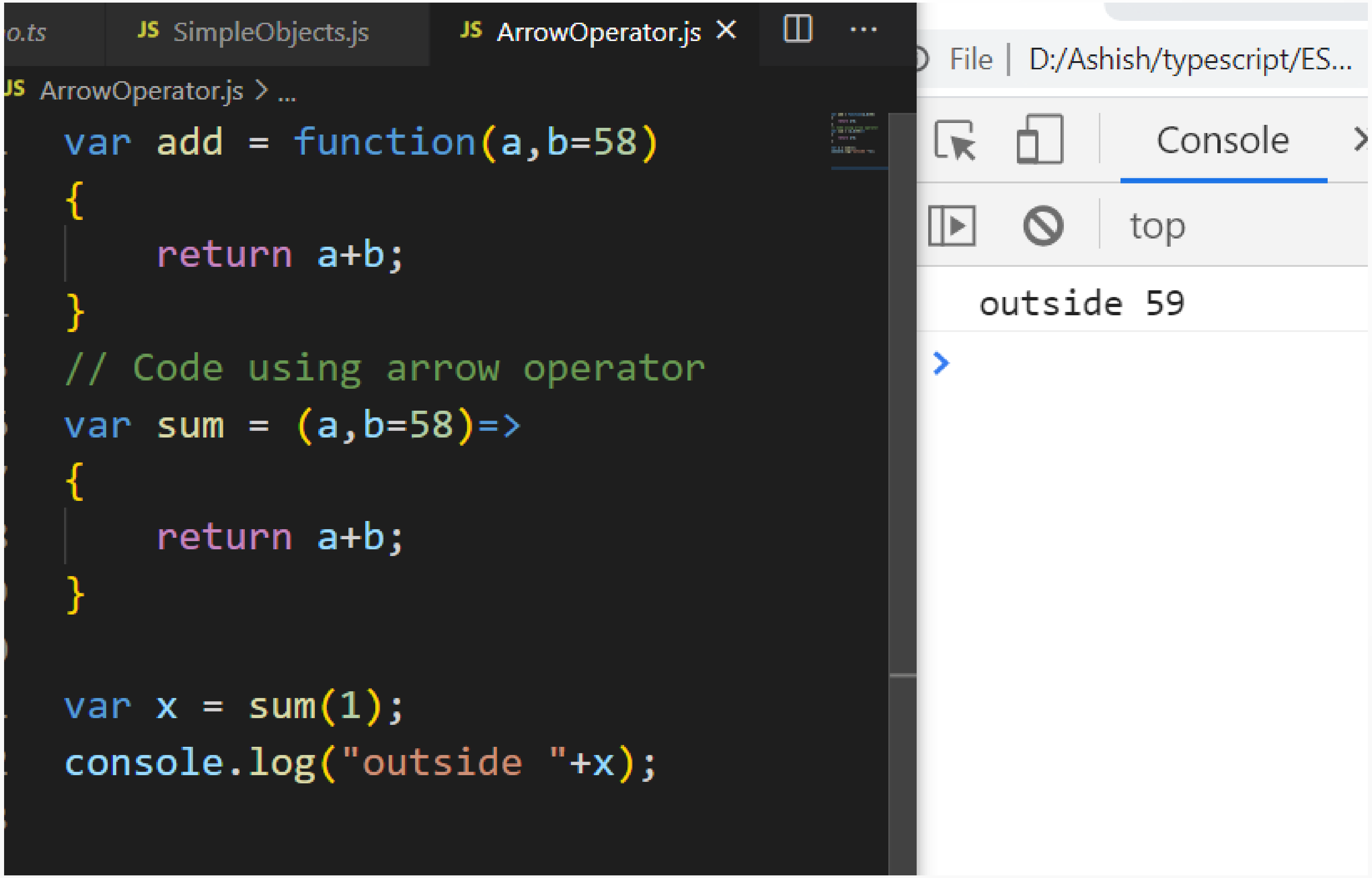
- Java Script allows to pass default arguments

- Function object and calling a function



```
var doAdd = function (a,b=58)
{
    console.log(a+b);
    return a+b;
}

var x = doAdd(15);
console.log("outside "+x);
```

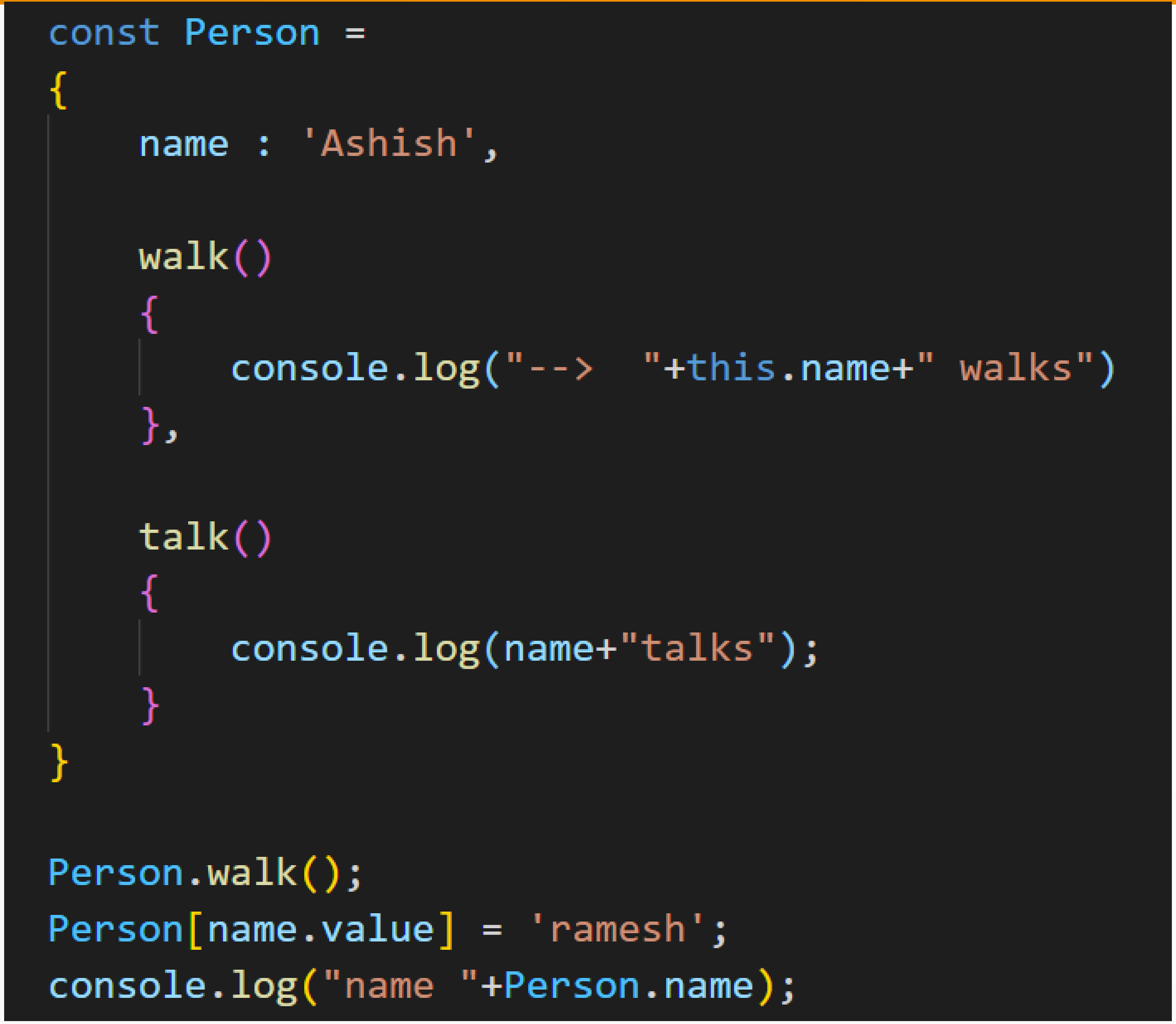
# Arrow ( =>) Operator



```
var add = function(a,b=58)
{
    return a+b;
}
// Code using arrow operator
var sum = (a,b=58)=>
{
    return a+b;
}

var x = sum(1);
console.log("outside "+x);
```

```
outside 59
```

# Creating Object in Java Script

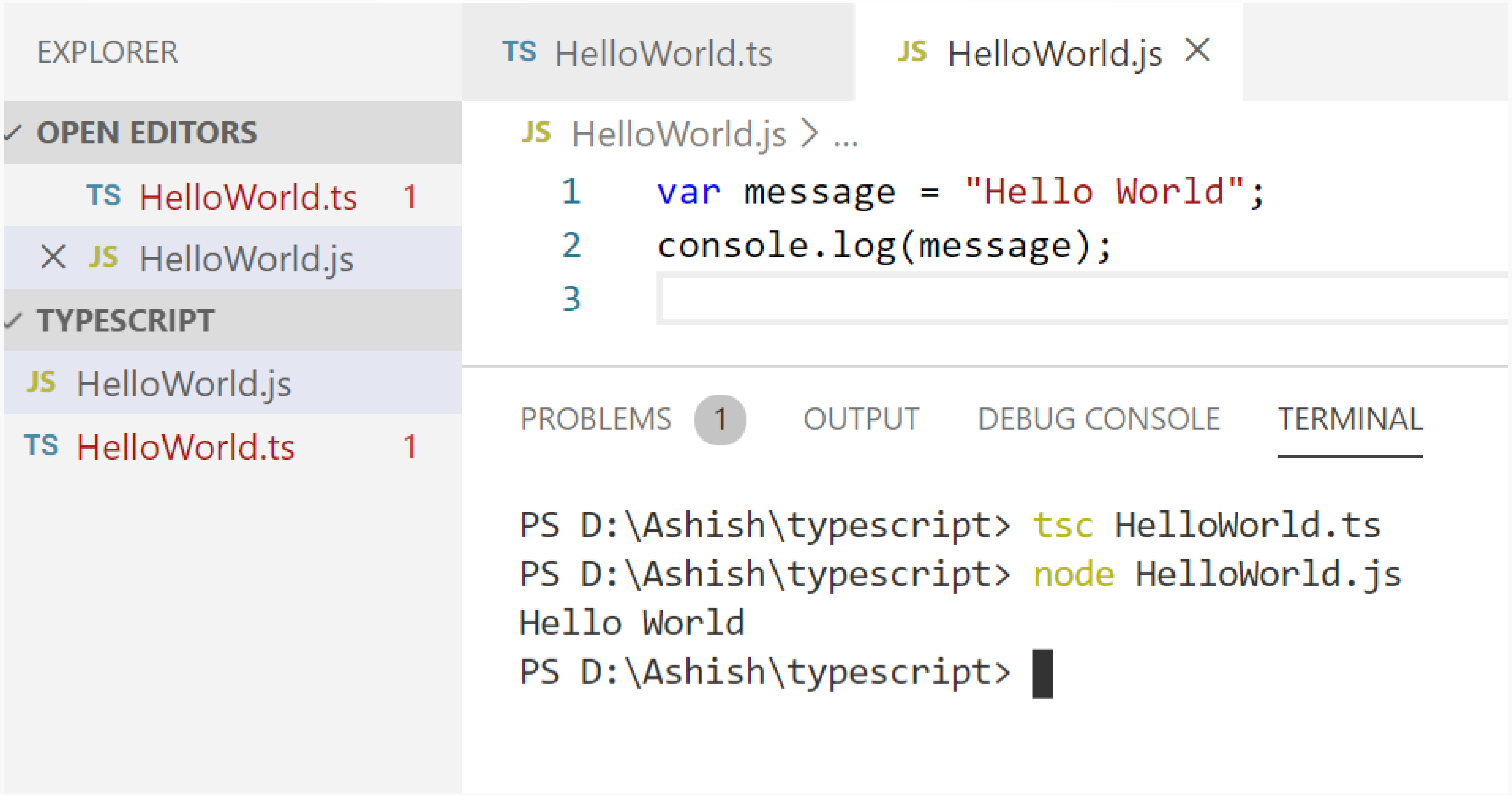


```
const Person =
{
    name : 'Ashish',

    walk()
    {
        console.log("-->  "+this.name+" walks")
    },

    talk()
    {
        console.log(name+"talks");
    }
}

Person.walk();
Person[name.value] = 'ramesh';
console.log("name "+Person.name);
```

**Discuss**

- What is Objects.
- What is Behaviour
- dot operator
- [ ] operator

# Hello World of Type Script

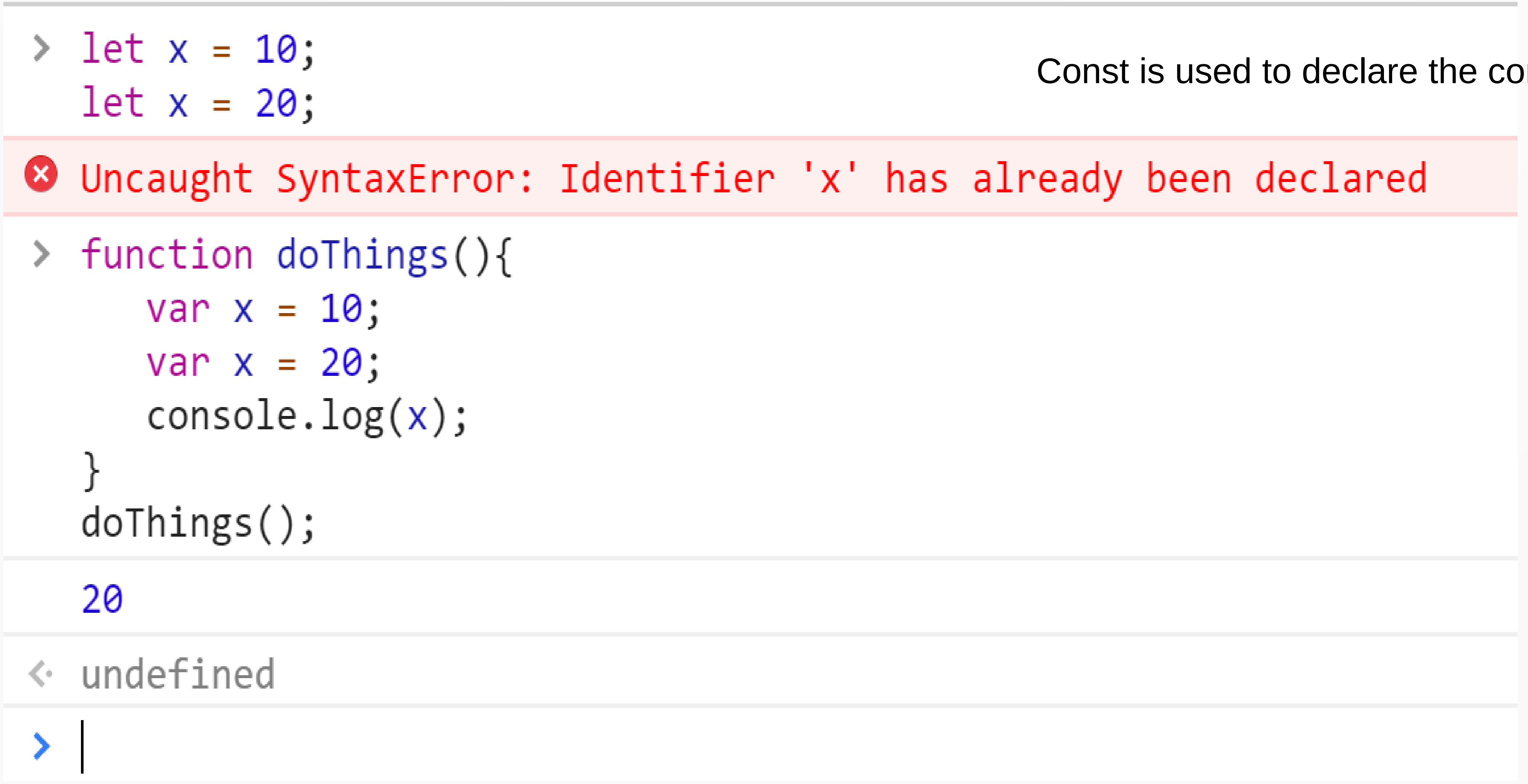


```
var message = "Hello World";
console.log(message);
```

```
PS D:\Ashish\typescript> tsc HelloWorld.ts
PS D:\Ashish\typescript> node HelloWorld.js
Hello World
PS D:\Ashish\typescript>
```

# Variable Declarations

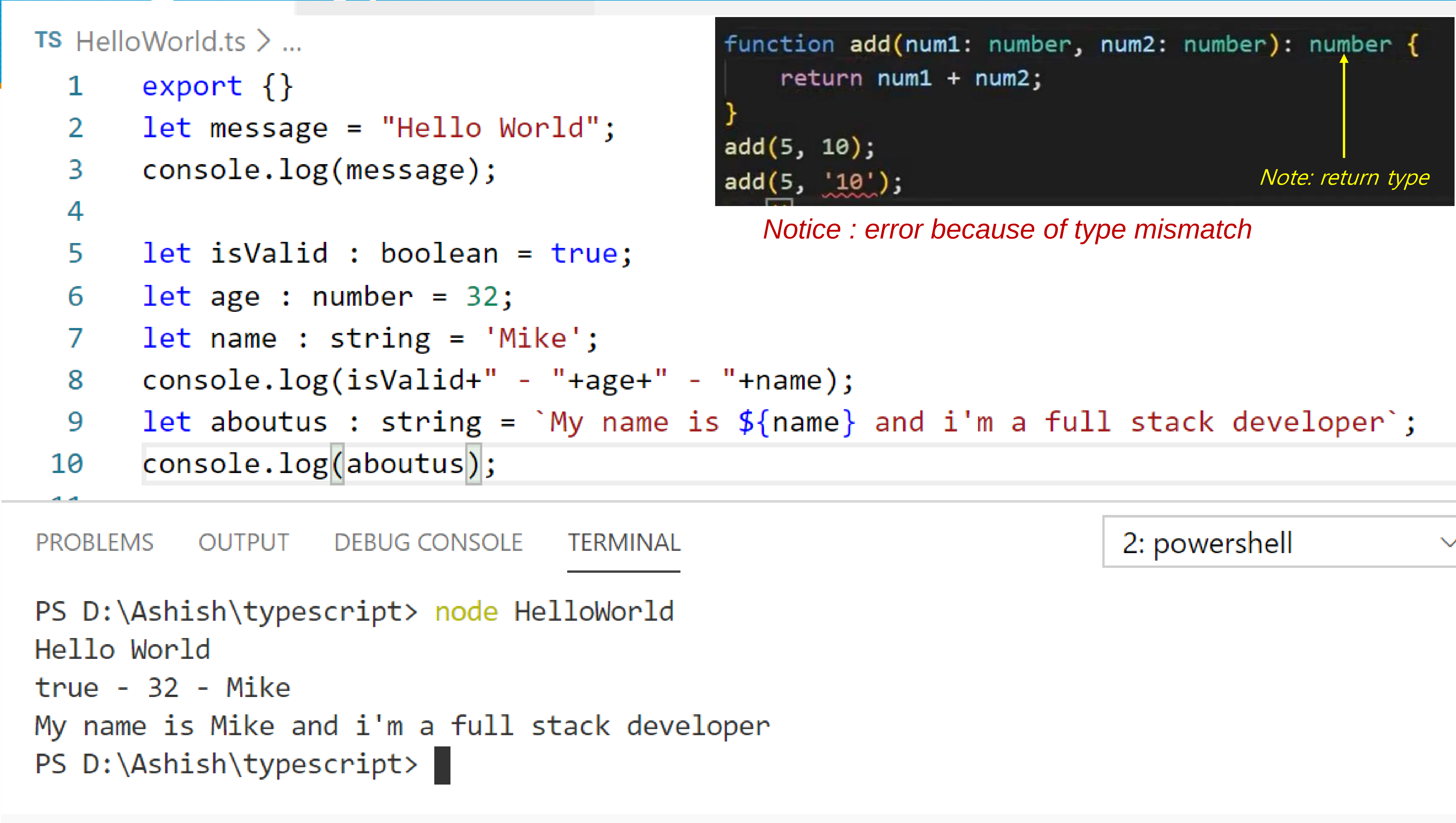
1) Typescript uses let & const keywork to declare variables.
2) One side JavaScript only has global scope and function scope , let & const solve this issue .
   let don't allow to re declare the variable and also provides block level support.



```
> let x = 10;
  let x = 20;
✖ Uncaught SyntaxError: Identifier 'x' has already been declared
> function doThings(){
      var x = 10;
      var x = 20;
      console.log(x);
  }
  doThings();
  20
<· undefined
>
```

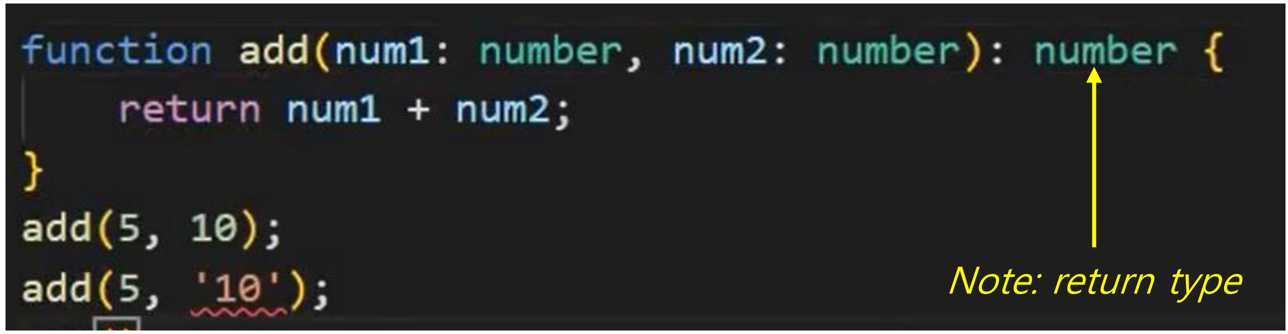
Const is used to declare the constants.

# Assign A Type



TS HelloWorld.ts > ...

```
export {}
let message = "Hello World";
console.log(message);

let isValid : boolean = true;
let age : number = 32;
let name : string = 'Mike';
console.log(isValid+" - "+age+" - "+name);
let aboutus : string = `My name is ${name} and i'm a full stack developer`;
console.log(aboutus);
```



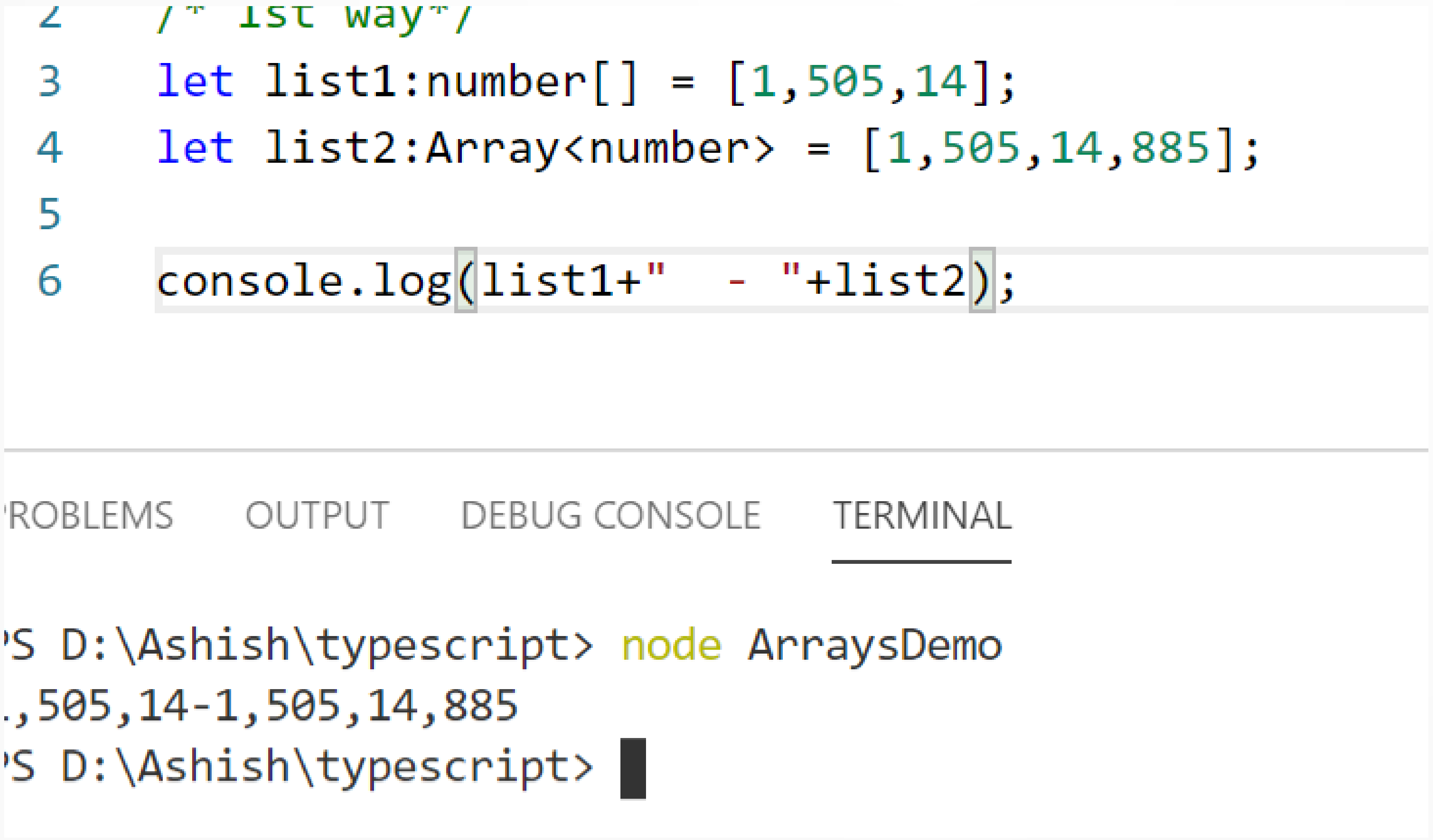
```
function add(num1: number, num2: number): number {
    return num1 + num2;
}
add(5, 10);
add(5, '10');
```

*Notice : error because of type mismatch*

```
PS D:\Ashish\typescript> node HelloWorld
Hello World
true - 32 - Mike
My name is Mike and i'm a full stack developer
PS D:\Ashish\typescript>
```

# Array Type

We have two different type of syntaxes for array declaration in type script

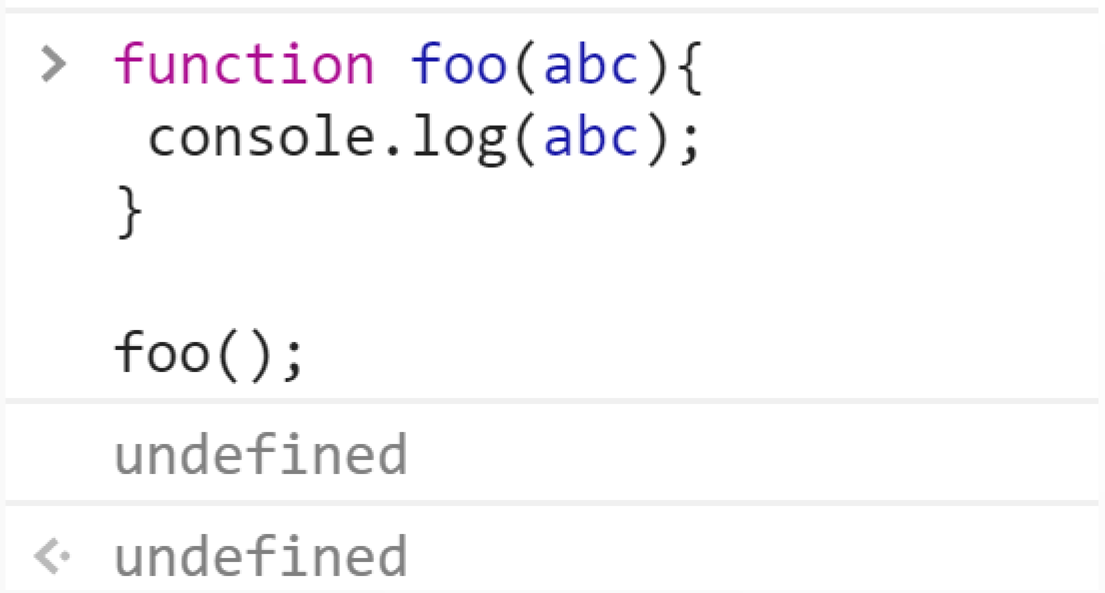


```
/* 1st way*/
let list1:number[] = [1,505,14];
let list2:Array<number> = [1,505,14,885];

console.log(list1+"  - "+list2);
```

```
'S D:\Ashish\typescript> node ArraysDemo
.,505,14-1,505,14,885
'S D:\Ashish\typescript>
```

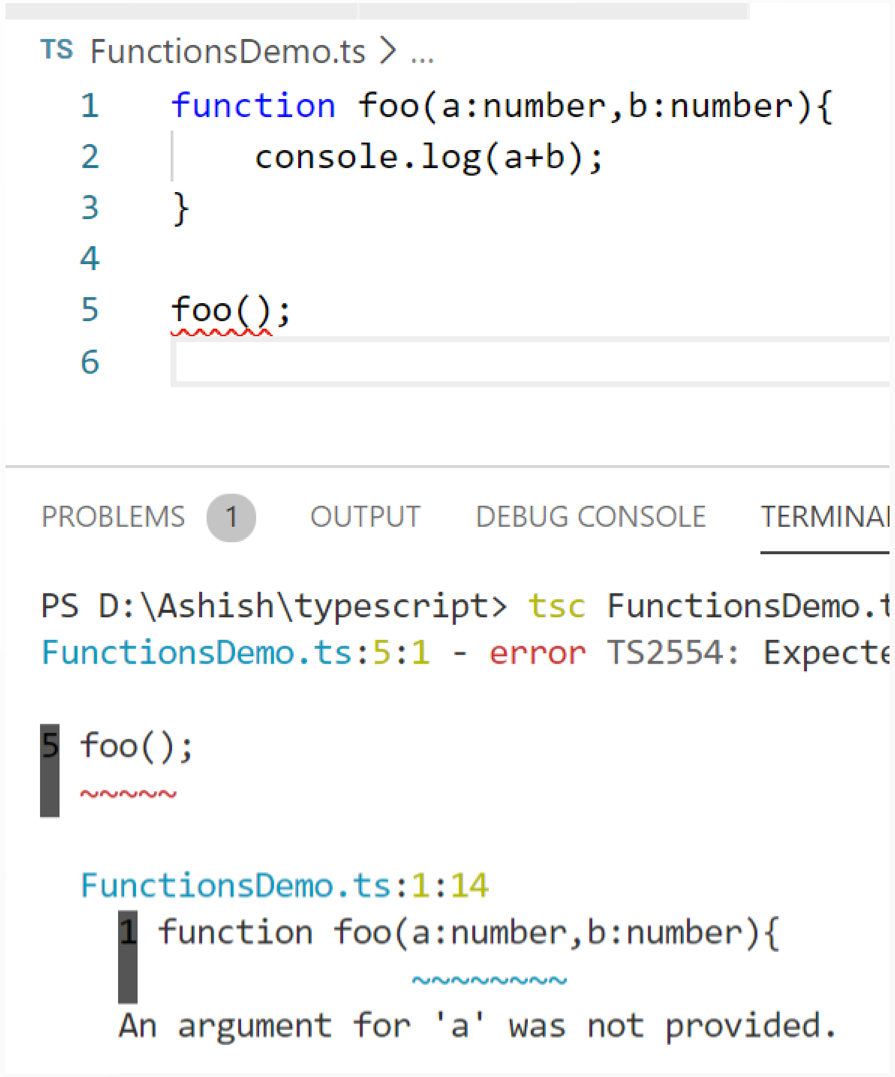
# Functions

Typescript supports optional parameters & default parameters



```
> function foo(abc){
   console.log(abc);
  }

  foo();
  undefined
< undefined
```

Notice : In JavaScript we can call method without parameter



TS FunctionsDemo.ts > ...

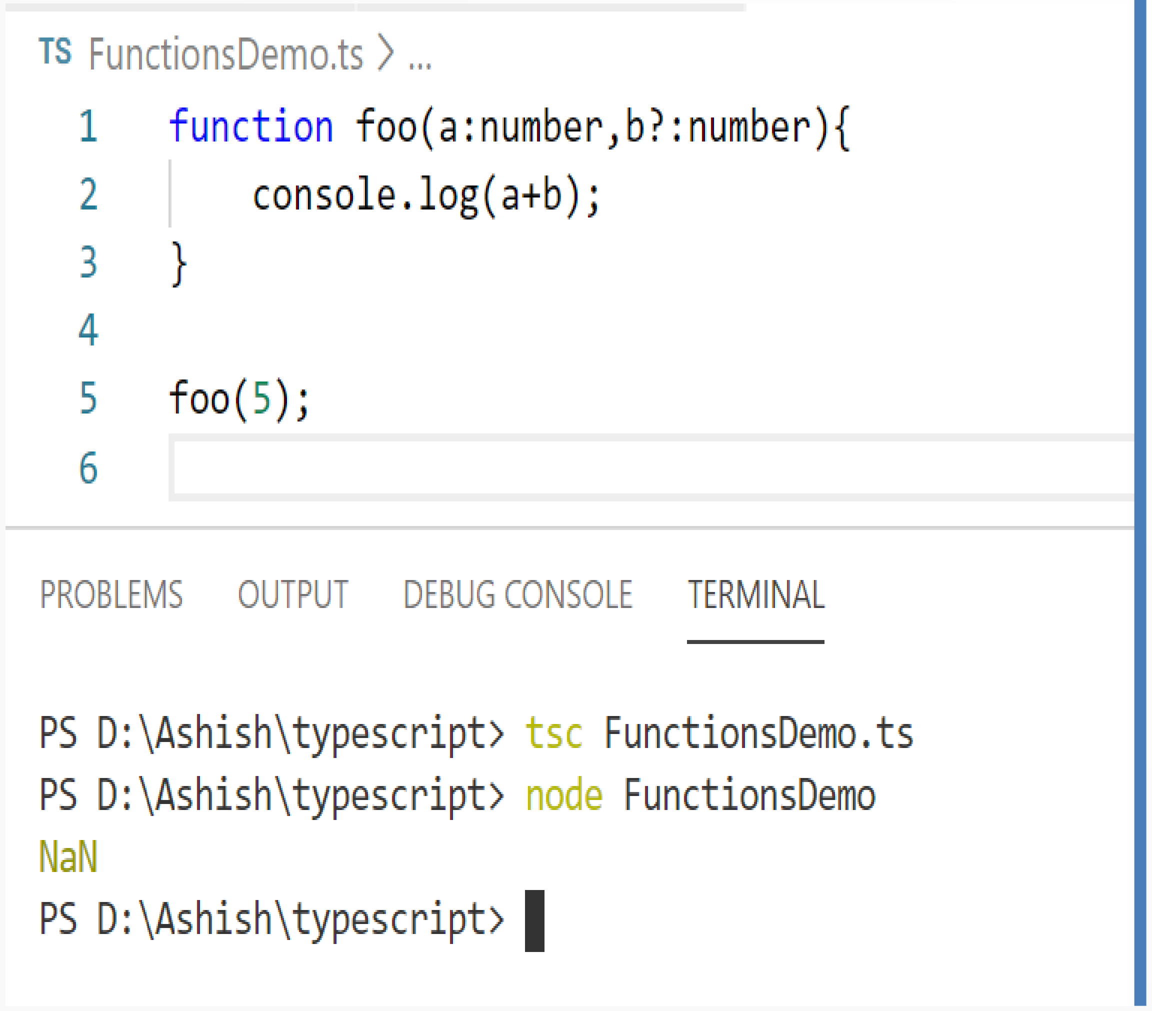
```
function foo(a:number,b:number){
    console.log(a+b);
}

foo();

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\Ashish\typescript> tsc FunctionsDemo.t
FunctionsDemo.ts:5:1 - error TS2554: Expecte

5 foo();
  ~~~~~

  FunctionsDemo.ts:1:14
    1 function foo(a:number,b:number){
                   ~~~~~~~~
    An argument for 'a' was not provided.
```

# Continue…

? Used to make parameter optional

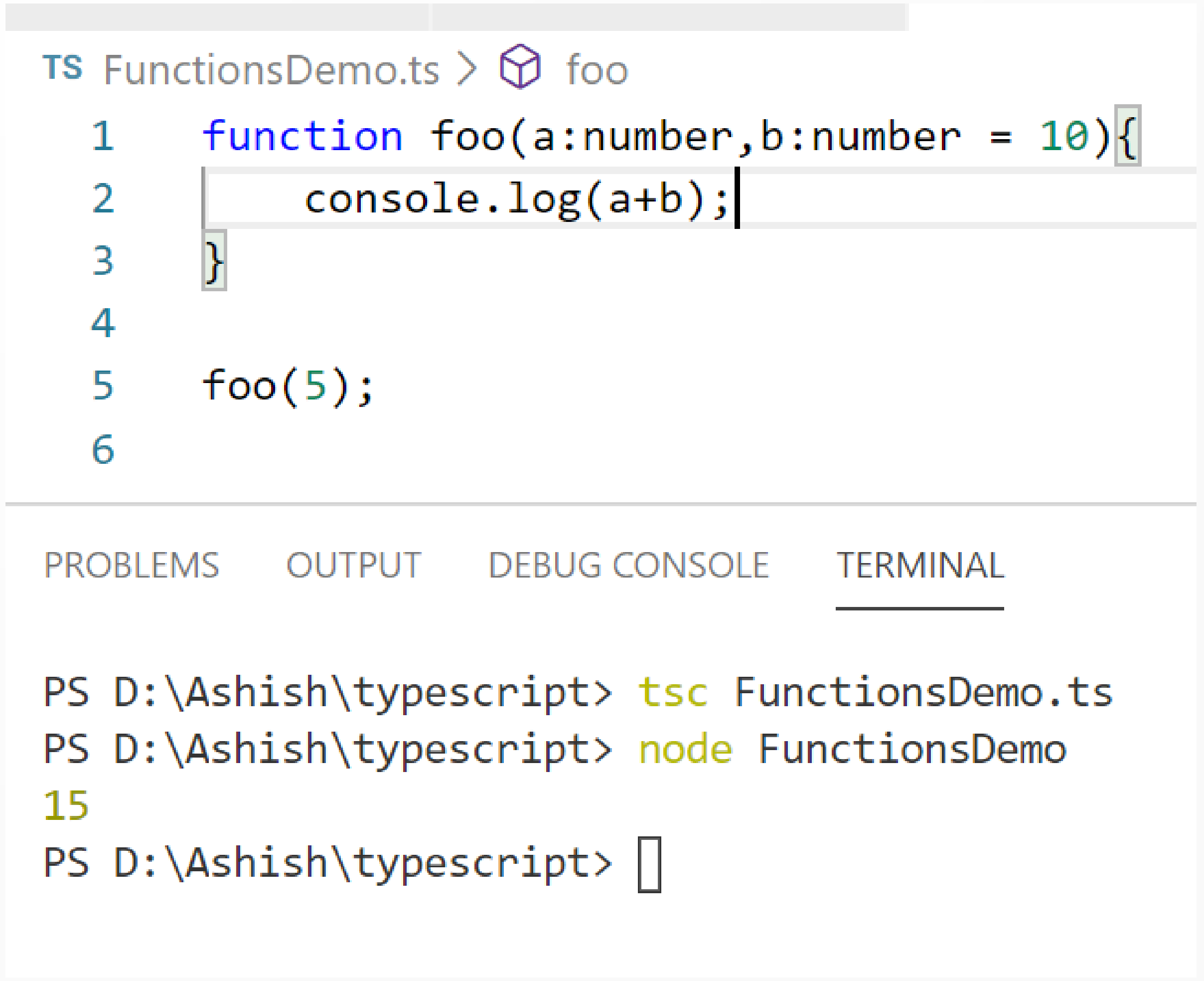


TS FunctionsDemo.ts > ...

```
function foo(a:number,b?:number){
    console.log(a+b);
}

foo(5);

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```
PS D:\Ashish\typescript> tsc FunctionsDemo.ts
PS D:\Ashish\typescript> node FunctionsDemo
NaN
PS D:\Ashish\typescript>
```



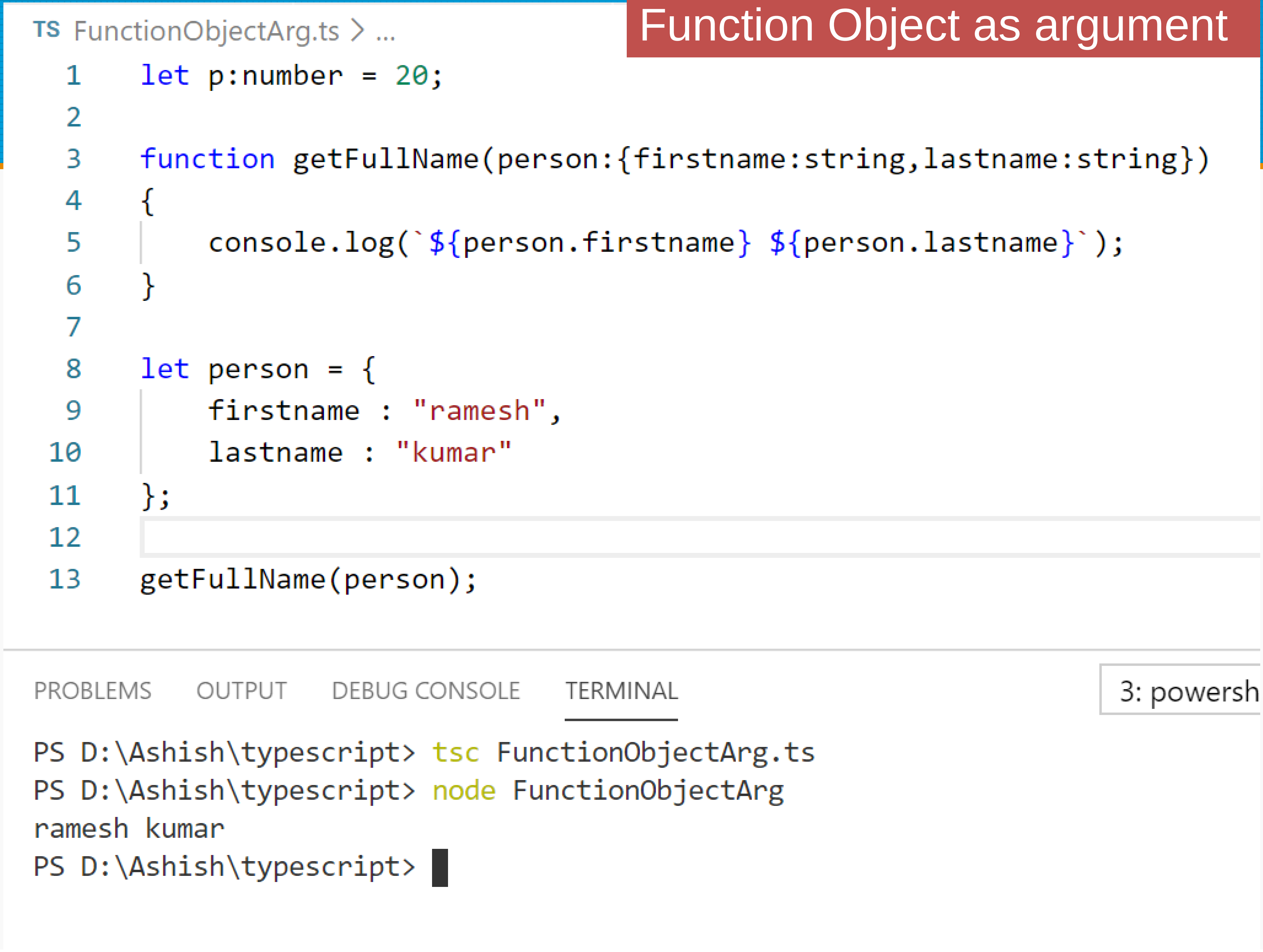
TS FunctionsDemo.ts > foo

```
function foo(a:number,b:number = 10){
    console.log(a+b);
}

foo(5);

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

```
PS D:\Ashish\typescript> tsc FunctionsDemo.ts
PS D:\Ashish\typescript> node FunctionsDemo
15
PS D:\Ashish\typescript>
```

# Function Object as argument



```
let p:number = 20;

function getFullName(person:{firstname:string,lastname:string})
{
    console.log(`${person.firstname} ${person.lastname}`);
}

let person = {
    firstname : "ramesh",
    lastname : "kumar"
};

getFullName(person);
```

```
PS D:\Ashish\typescript> tsc FunctionObjectArg.ts
PS D:\Ashish\typescript> node FunctionObjectArg
ramesh kumar
PS D:\Ashish\typescript>
```

## Class

1) Creation of class
2) Adding a method
3) Extending the relationship

Self Learning

1) Modules
2) Decorators (more we discuss during angular.js)

# Introduction of Angular What & Why?

1) What is Angular

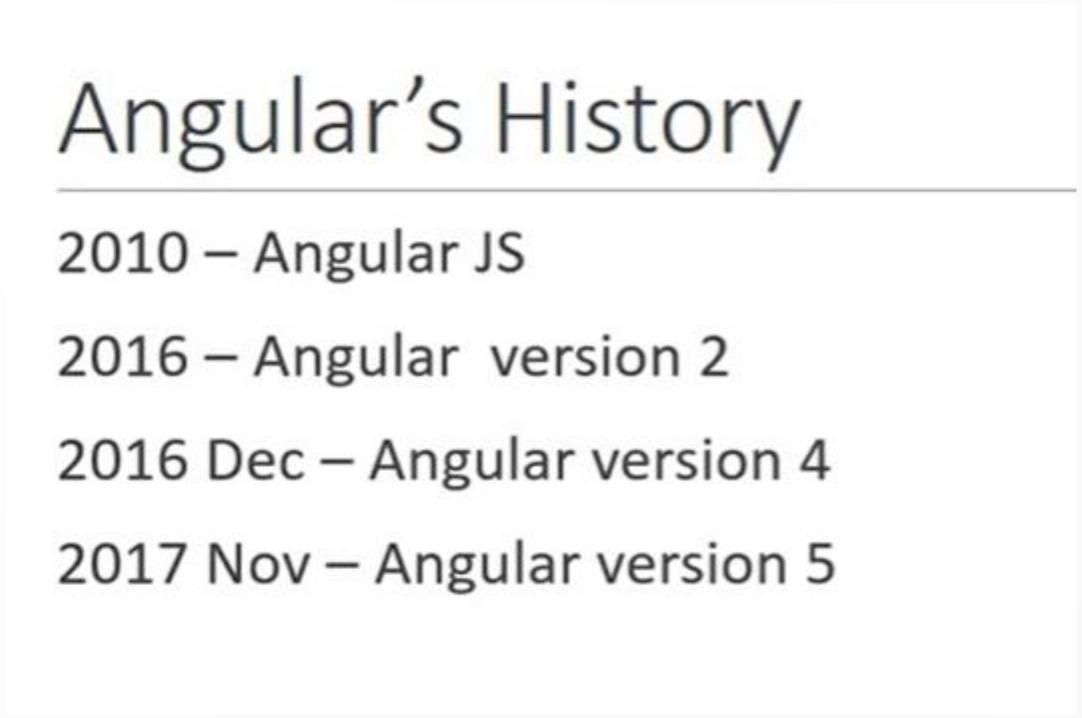
2) Modular Approach

3) Re-Usable Code.

4) Inbuild Validation , Routing functionality

5) Unit testable

6) Using TypeScript from Microsoft.



## Angular's History

2010 – Angular JS

2016 – Angular  version 2

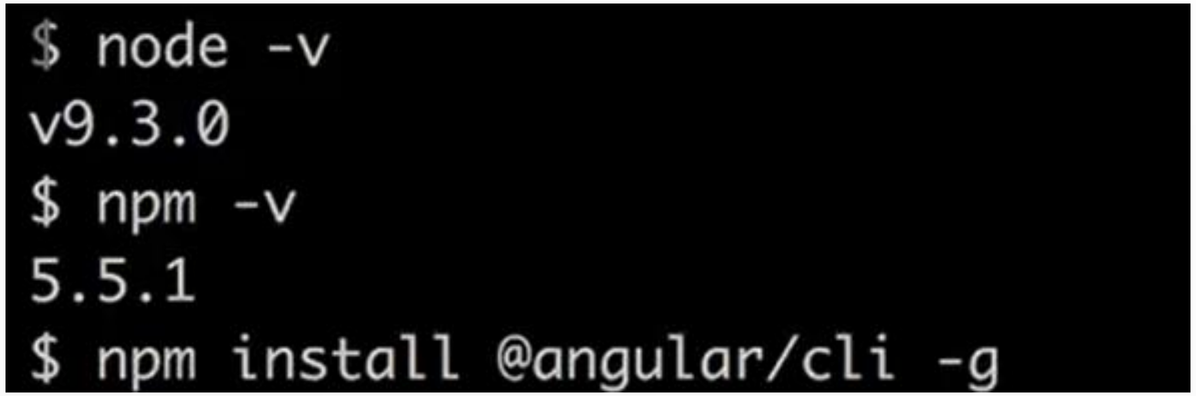
2016 Dec – Angular version 4

2017 Nov – Angular version 5

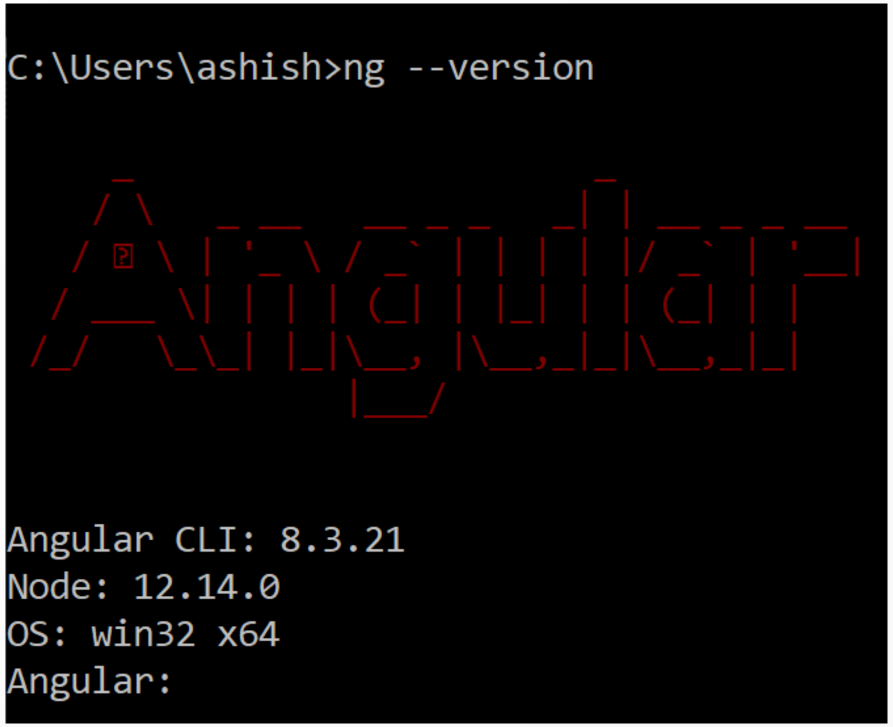
# Setting up Development Environment

For Angular Js We need

1) node
2) Npm (node package manager)
3) Angluar cli



```
$ node -v
v9.3.0
$ npm -v
5.5.1
$ npm install @angular/cli -g
```



```
C:\Users\ashish>ng --version

     _                      _
    / \   _ __   __ _ _   _| | __ _ _ __
   / △ \ | '_ \ / _` | | | | |/ _` | '__|
  / ___ \| | | | (_| | |_| | | (_| | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|
                |___/


Angular CLI: 8.3.21
Node: 12.14.0
OS: win32 x64
Angular:
```

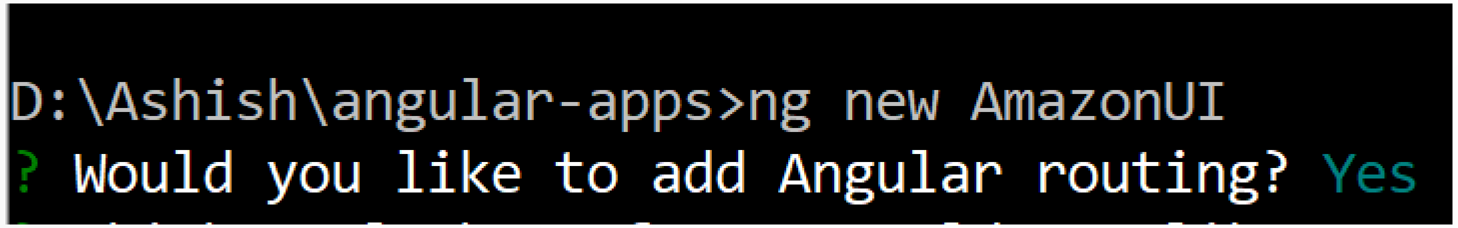
# Angular CLI Commands

| Purpose | Command | Shortcut Command |
|---|---|---|
| New Application | ng new DemoApp | |
| New Component | ng generate component User-View | ng g c User-View |
| New Class | ng generate class User | ng g cl User |
| New Class or Component within folder | ng generate class classes/User | ng g cl classes/User |
| New Service | ng generate service UserOperations | ng g s UserOperations |
| New Service with Module registration | ng generate service UserOperations --module | ng g s UserOperations --module |
| New Service with Module registration with specific module | ng generate service UserOperations -m=app.module | ng g s UserOperations  -m=app.module |
| New Interface | ng generate interface Role | ng g i Role |
| New Module | ng generate module UserModule | ng g m UserModule |

Check for –flat option

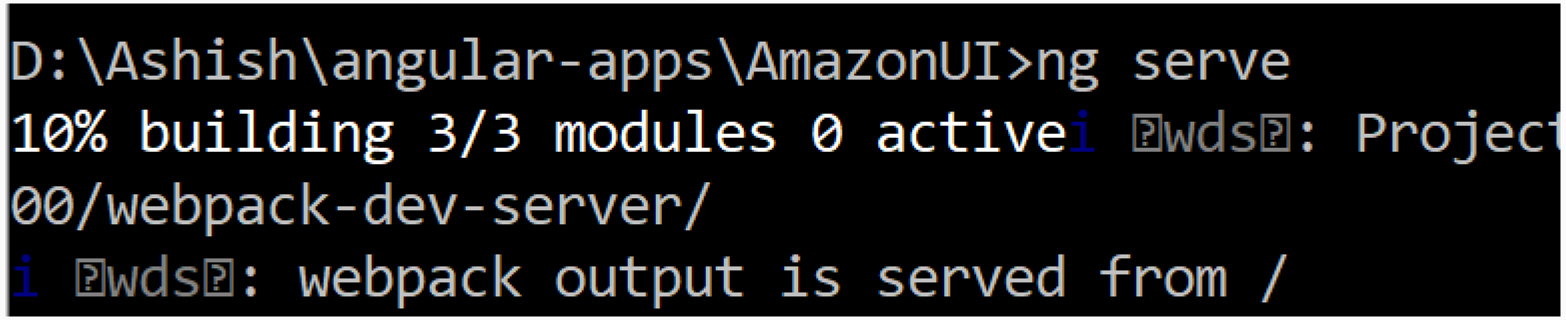
# Creating the First Project

Creating the Application



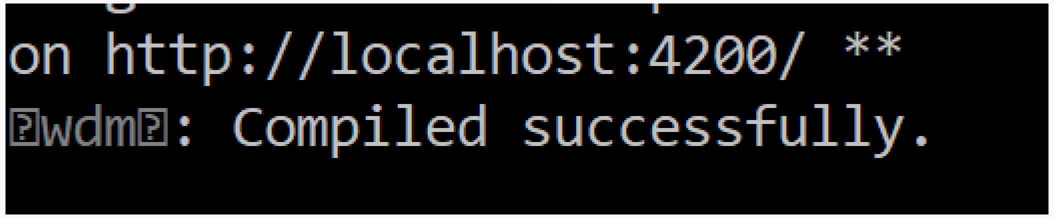
```
D:\Ashish\angular-apps>ng new AmazonUI
? Would you like to add Angular routing? Yes
```

Compile the Application



```
D:\Ashish\angular-apps\AmazonUI>ng serve
10% building 3/3 modules 0 activei ｢wds｣: Project
00/webpack-dev-server/
i ｢wds｣: webpack output is served from /
```

Application live on



```
on http://localhost:4200/ **
｢wdm｣: Compiled successfully.
```

# Understanding main Building blocks of Angular application

## Components

- @Component Decorator is used to make a typescript class as component
- Angular app can have one or more components.

## Templates

- It’s a HTML along with angular expressions

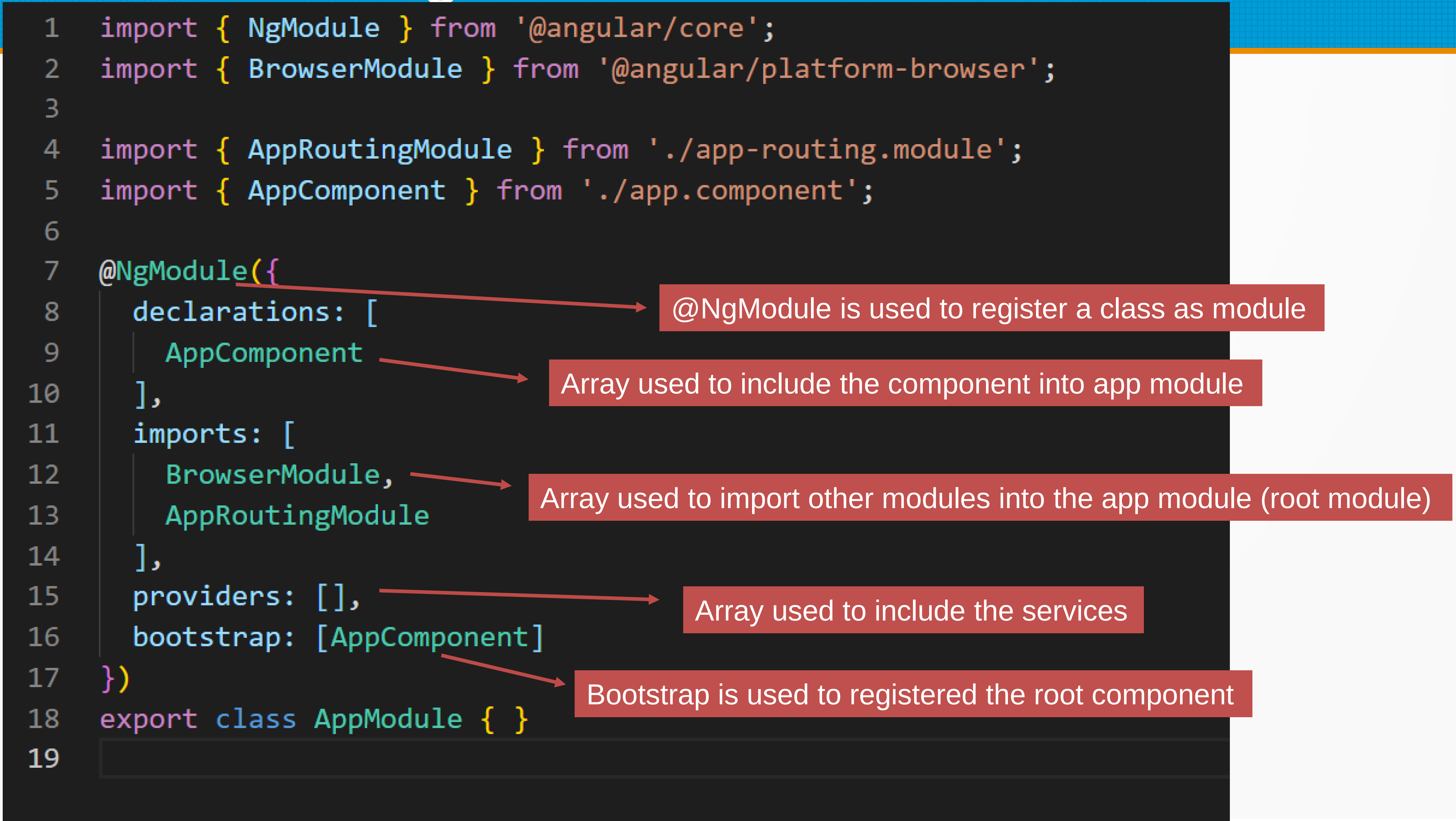
## Modules

- Modules has collections of Components & Services

## Service

- Services is a typescript class used for business logic and separation of concerns

# Understanding Modules



```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

@NgModule is used to register a class as module

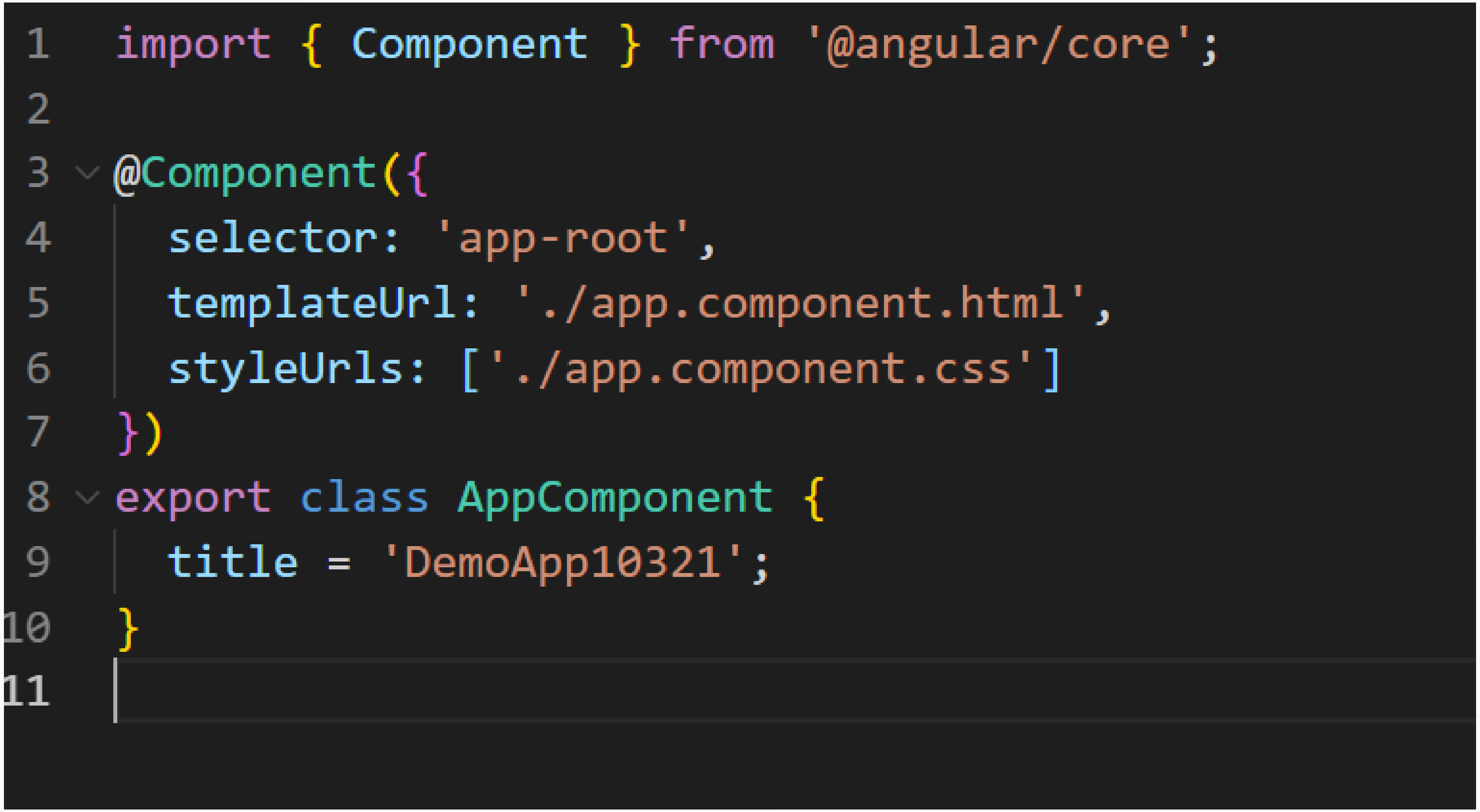
Array used to include the component into app module

Array used to import other modules into the app module (root module)

Array used to include the services

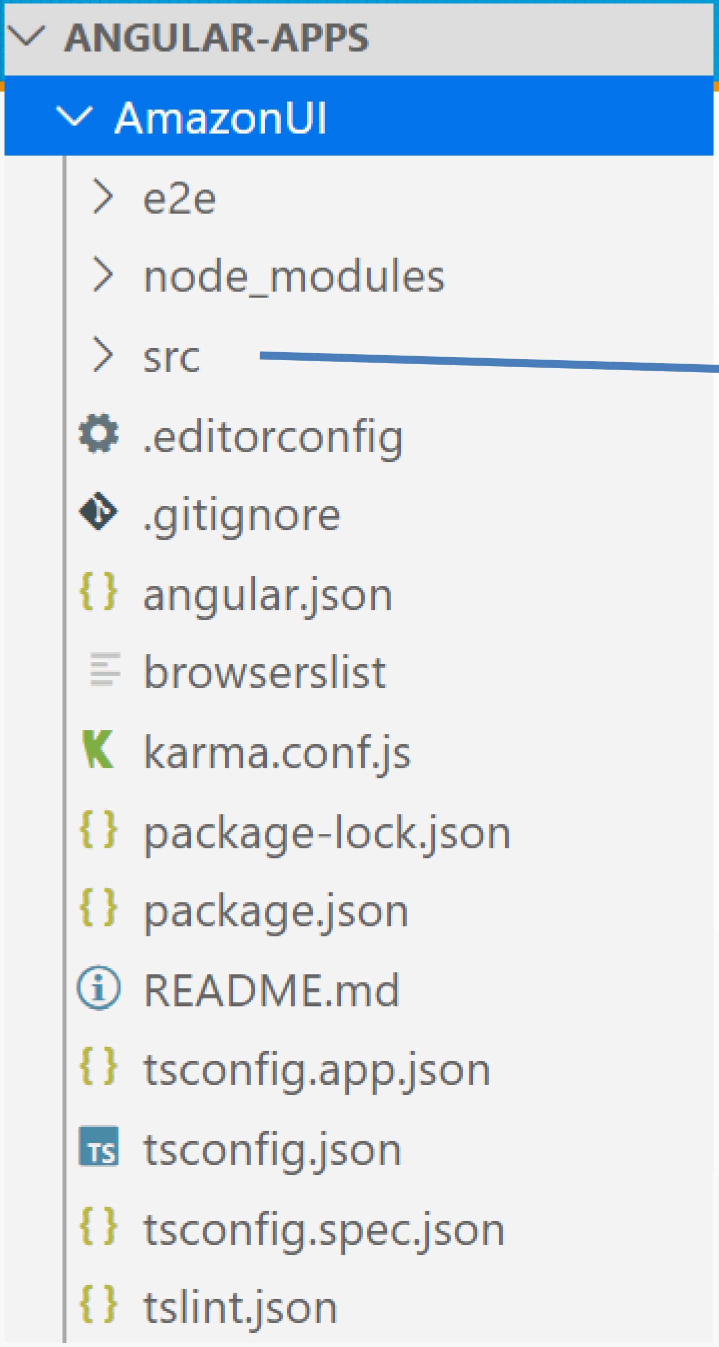
Bootstrap is used to registered the root component

# Understanding the component

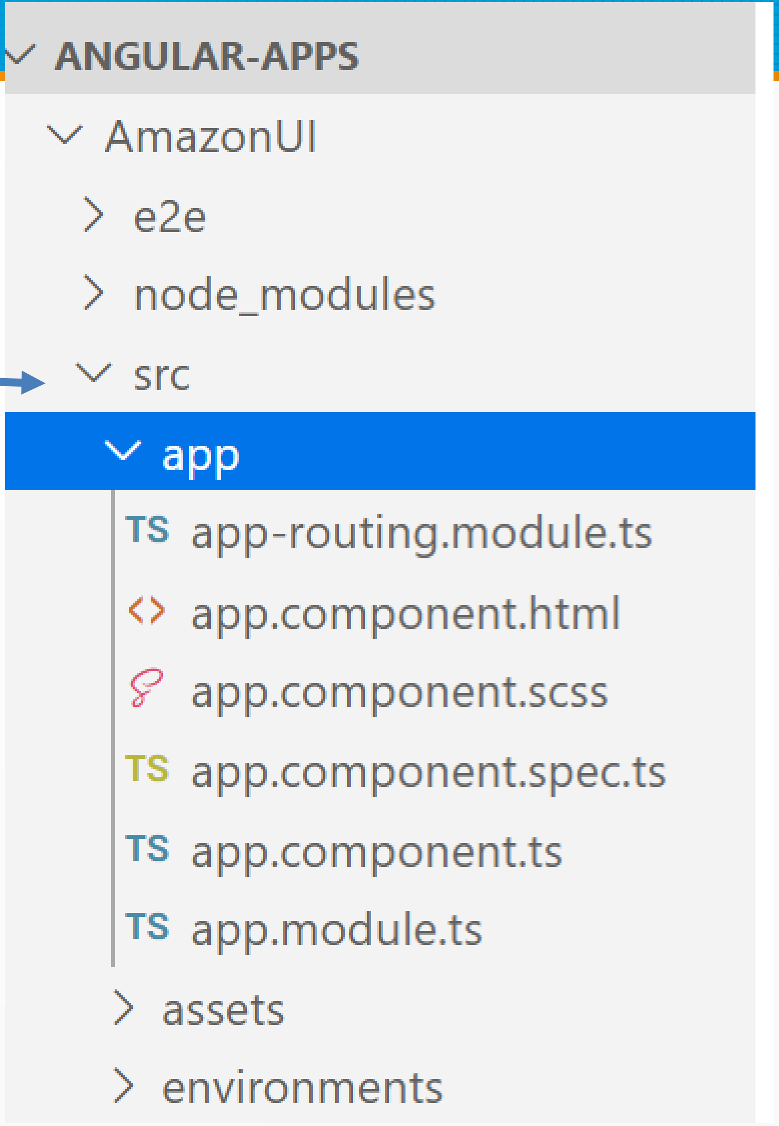


```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'DemoApp10321';
}

```

# Project Structure



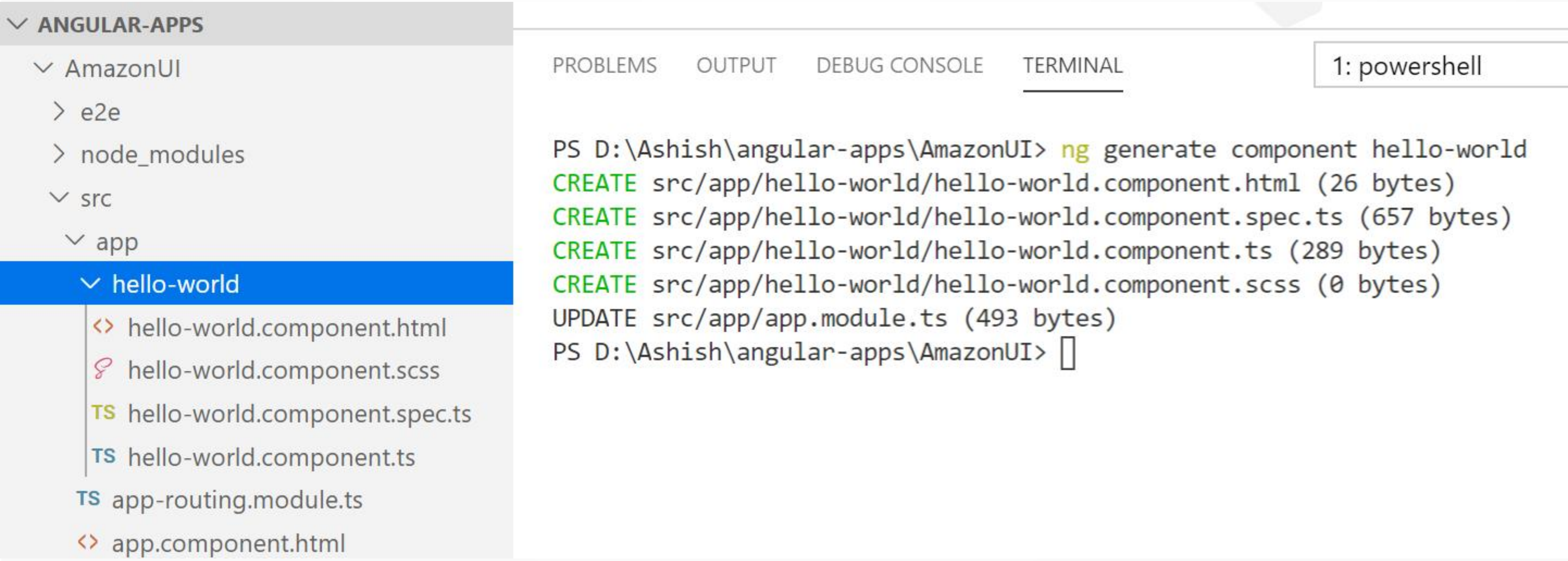
```
ANGULAR-APPS
  AmazonUI
    e2e
    node_modules
    src
    .editorconfig
    .gitignore
    angular.json
    browserslist
    karma.conf.js
    package-lock.json
    package.json
    README.md
    tsconfig.app.json
    tsconfig.json
    tsconfig.spec.json
    tslint.json
```



```
ANGULAR-APPS
  AmazonUI
    e2e
    node_modules
    src
      app
        app-routing.module.ts
        app.component.html
        app.component.scss
        app.component.spec.ts
        app.component.ts
        app.module.ts
      assets
      environments
```

# Creating Angular Component in Angular 8

:> ng generate component <component-name>

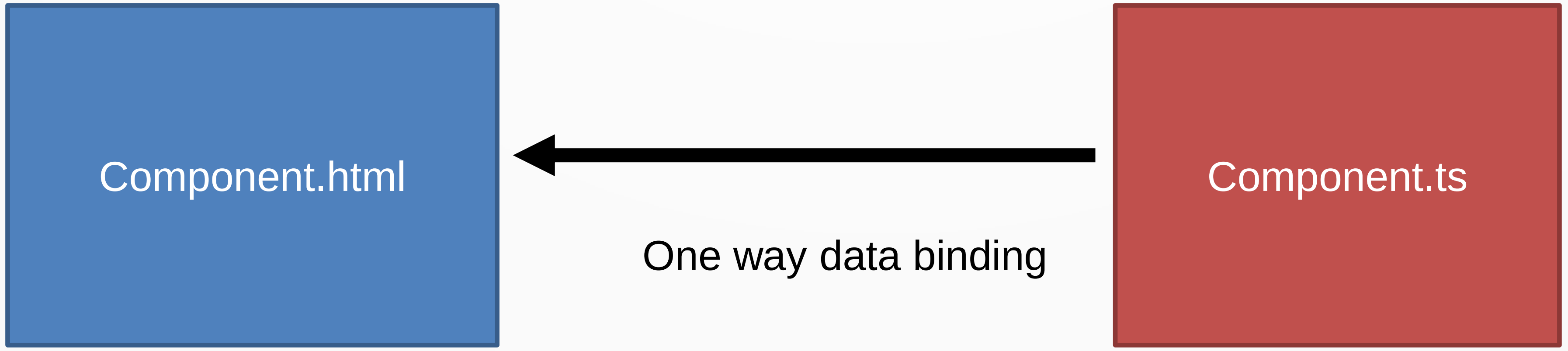


```
PS D:\Ashish\angular-apps\AmazonUI> ng generate component hello-world
CREATE src/app/hello-world/hello-world.component.html (26 bytes)
CREATE src/app/hello-world/hello-world.component.spec.ts (657 bytes)
CREATE src/app/hello-world/hello-world.component.ts (289 bytes)
CREATE src/app/hello-world/hello-world.component.scss (0 bytes)
UPDATE src/app/app.module.ts (493 bytes)
PS D:\Ashish\angular-apps\AmazonUI>
```

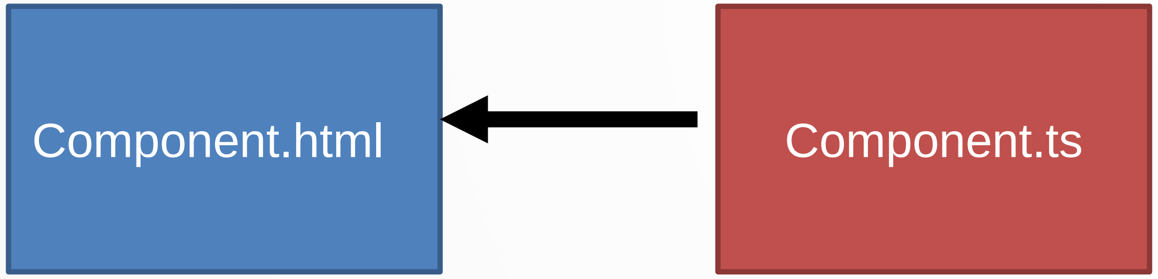
## Interpolation

{{ expression }}

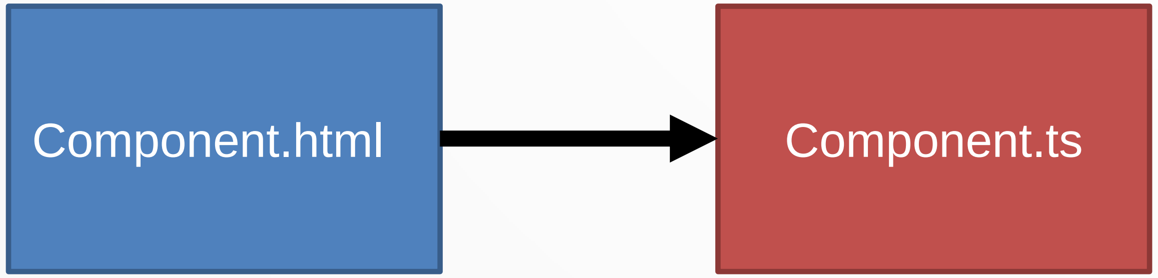
Used to retrieve values from component to html page (*.ts) to (*.html)



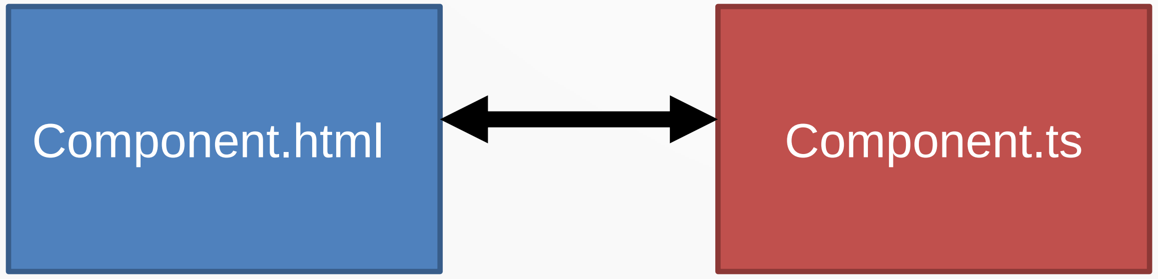
# Other Data Binding approaches



**Property binding**
[property]  = “value”
```
 <input [disabled]="status" type="text"/>
```



Event binding
(event)  = “Handler”
```
<button (click)="getTitle"> My Button
 </button>
```

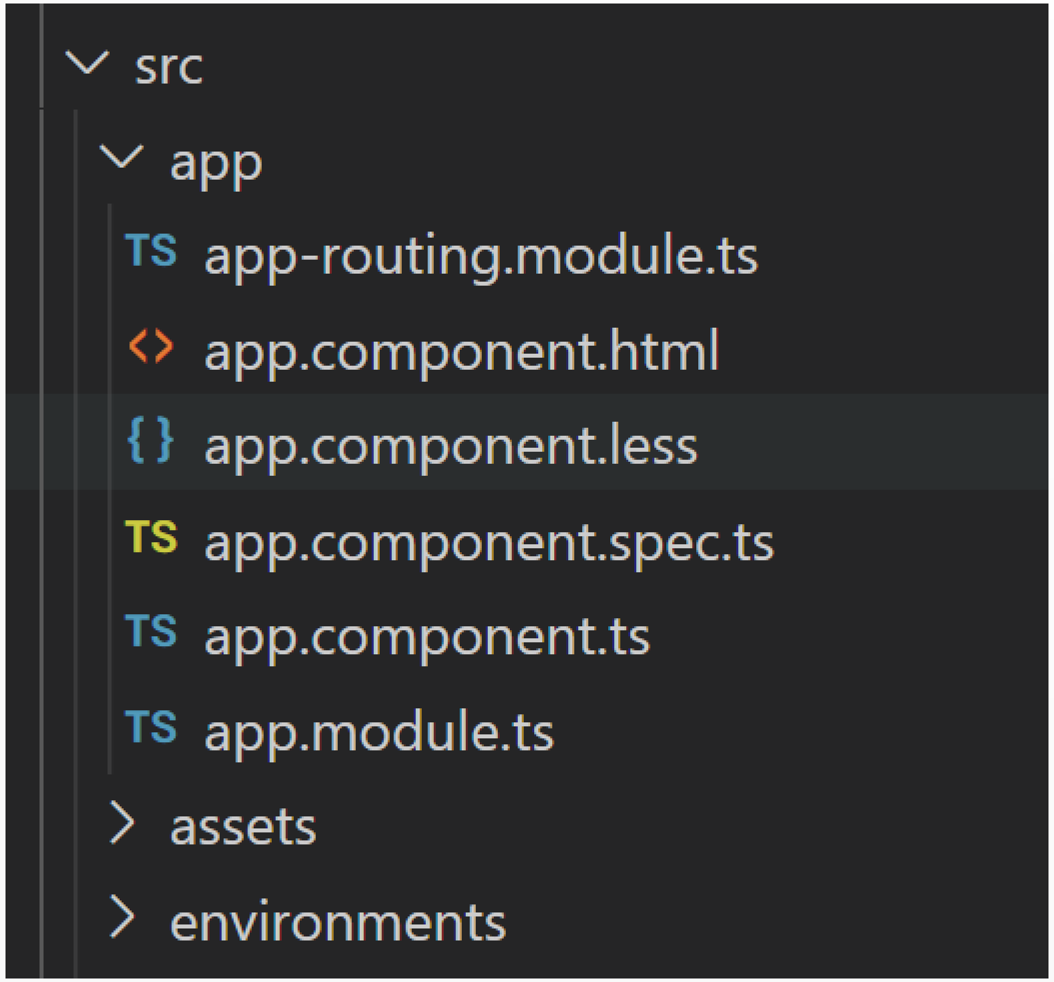


Two ways Data Binding
 using ngModel
 [(ngModel)] = “XYZModel.propertyName”

https://github.com/mkjitlearnings/AllAngular8/blob/main/TwoWayBinding

## Component

component is the main building block of the angular application.
App component is the root component.



```
src
  app
    app-routing.module.ts
    app.component.html
    app.component.less
    app.component.spec.ts
    app.component.ts
    app.module.ts
  assets
  environments
```

App.component.html is a view template

App.component.ts is the model .

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.less']
})
export class AppComponent {
  ….. .
}
```

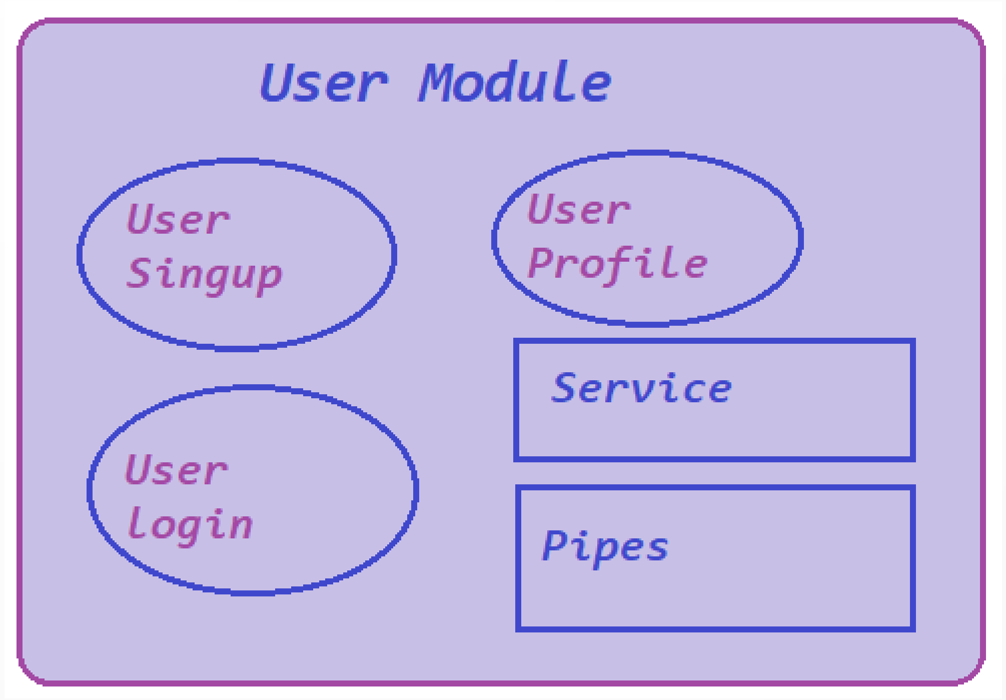
# Modules

So a module is a collection of related components.

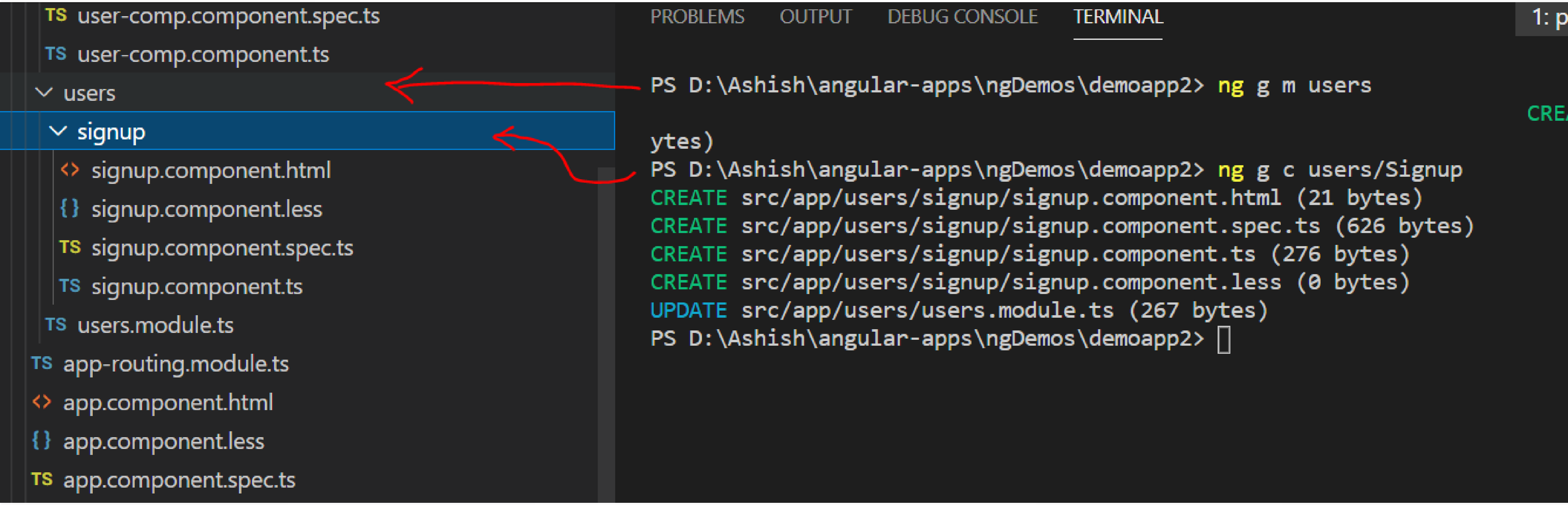
**ng g m mymodule**

Create component inside module

**ng g c mymodule/mycomponent**



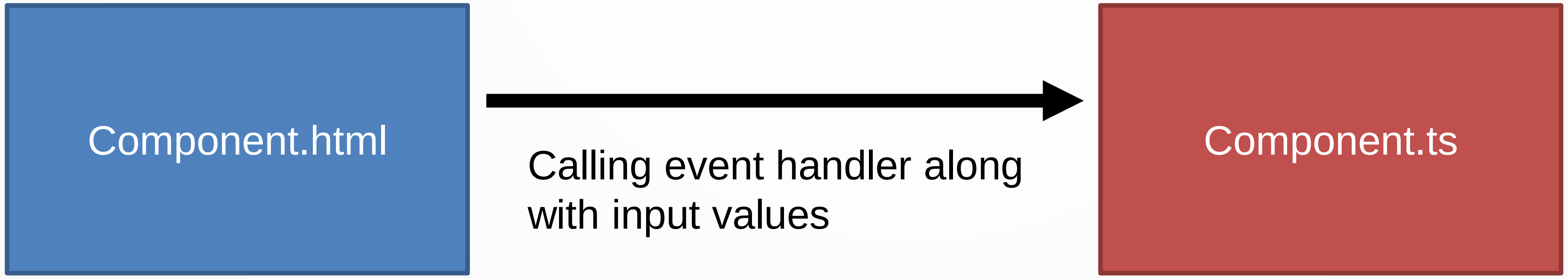
# Creation of Module and Sub Components



```
PS D:\Ashish\angular-apps\ngDemos\demoapp2> ng g m users
                                                                  CRE

ytes)
PS D:\Ashish\angular-apps\ngDemos\demoapp2> ng g c users/Signup
CREATE src/app/users/signup/signup.component.html (21 bytes)
CREATE src/app/users/signup/signup.component.spec.ts (626 bytes)
CREATE src/app/users/signup/signup.component.ts (276 bytes)
CREATE src/app/users/signup/signup.component.less (0 bytes)
UPDATE src/app/users/users.module.ts (267 bytes)
PS D:\Ashish\angular-apps\ngDemos\demoapp2> 
```

## Template Reference Variable

Template reference variable is used to bind DOM component to the model property
Mostly used with event binding



- Template reference variable is assigned using # followed by variable name.

https://github.com/mkjitlearnings/AllAngular8

## Angular – 8 Directives

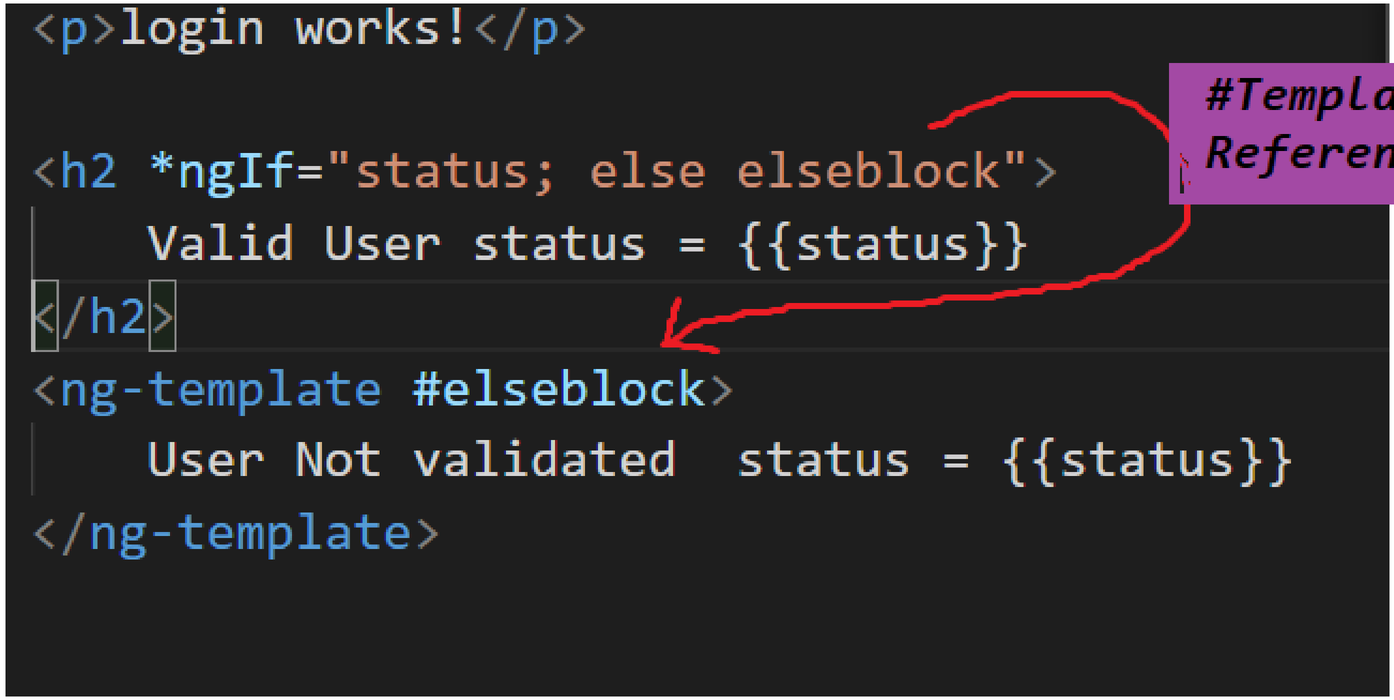
https://angular.io/guide/attribute-directives

There are three kinds of directives in Angular

1. Components—directives with a template.
2. Structural directives—change the DOM layout by adding and removing DOM elements.
3. Attribute directives—change the appearance or behaviour of an element, component, or another directive.

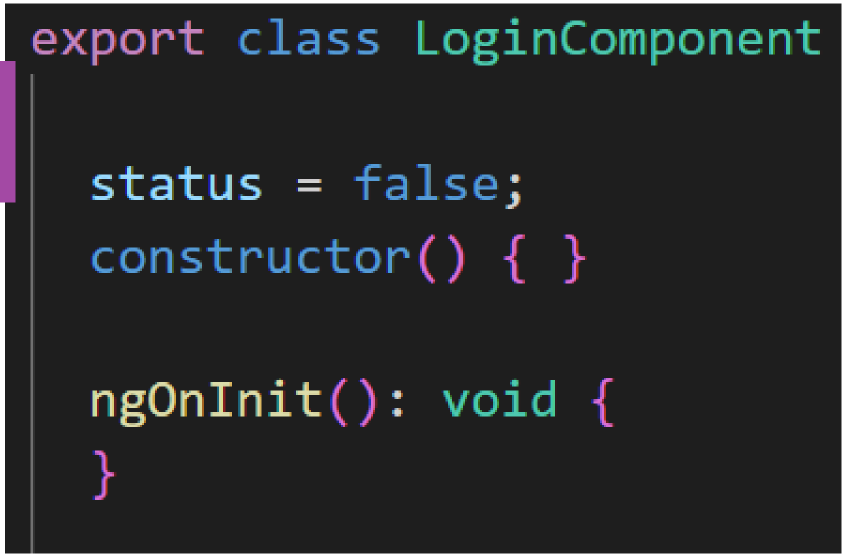
# Structural Directive *ngIf & ngSwitch

https://angular.io/api/common/NgIf

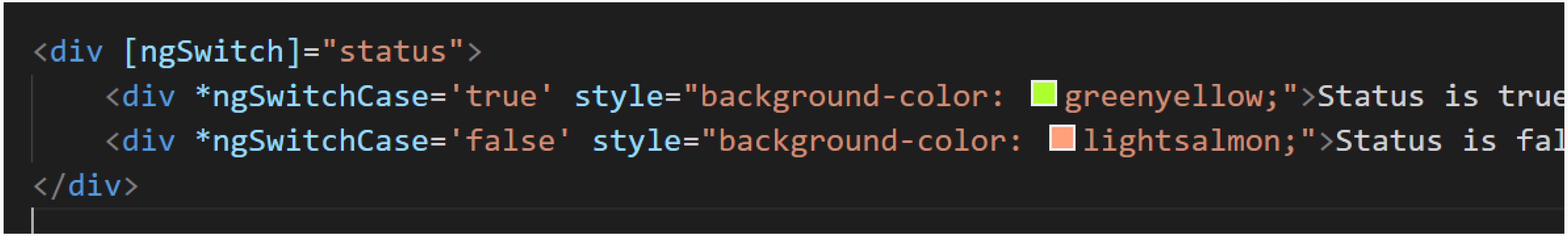


```
<p>login works!</p>

<h2 *ngIf="status; else elseblock">
    Valid User status = {{status}}
</h2>
<ng-template #elseblock>
    User Not validated  status = {{status}}
</ng-template>
```

#Template Reference

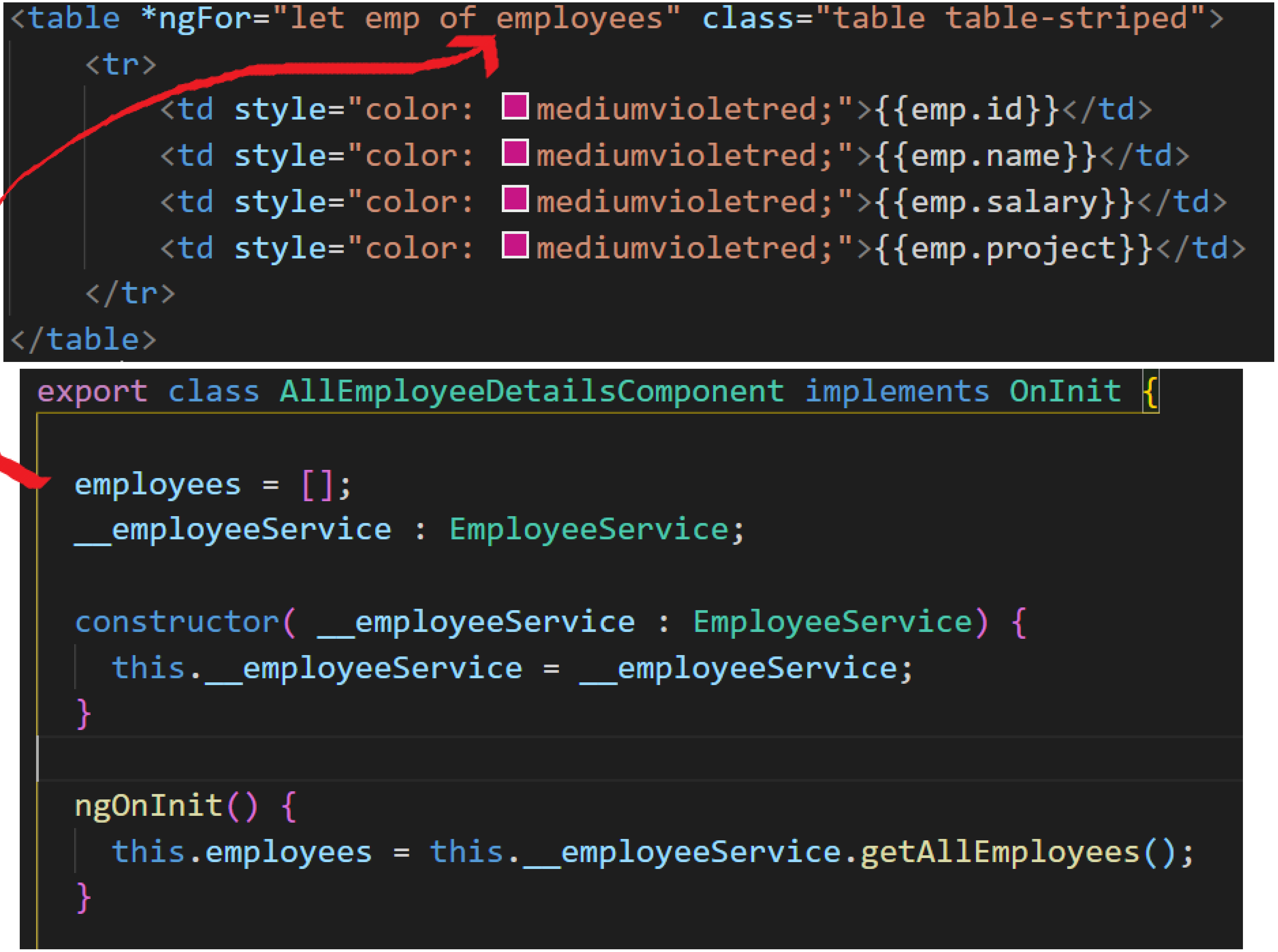


```
export class LoginComponent

  status = false;
  constructor() { }

  ngOnInit(): void {
  }
```



```
<div [ngSwitch]="status">
    <div *ngSwitchCase='true' style="background-color: 🟩greenyellow;">Status is true
    <div *ngSwitchCase='false' style="background-color: 🟧lightsalmon;">Status is fal
</div>
```

# Ng for



```
<table *ngFor="let emp of employees" class="table table-striped">
    <tr>
        <td style="color: mediumvioletred;">{{emp.id}}</td>
        <td style="color: mediumvioletred;">{{emp.name}}</td>
        <td style="color: mediumvioletred;">{{emp.salary}}</td>
        <td style="color: mediumvioletred;">{{emp.project}}</td>
    </tr>
</table>
```

```
export class AllEmployeeDetailsComponent implements OnInit {

  employees = [];
  __employeeService : EmployeeService;

  constructor( __employeeService : EmployeeService) {
    this.__employeeService = __employeeService;
  }

  ngOnInit() {
    this.employees = this.__employeeService.getAllEmployees();
  }
```

# Angular Lifecycle hooks

## Introduction

https://angular.io/guide/lifecycle-hooks

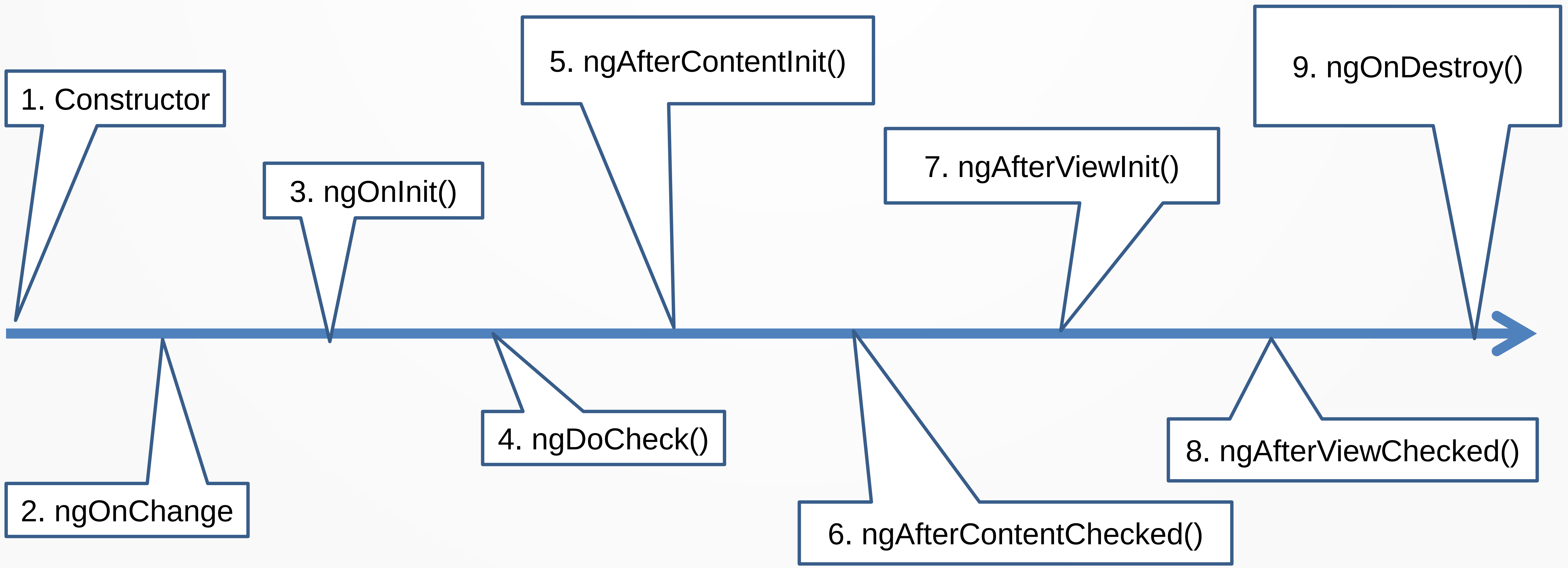
Angular allows us to trigger the actions at the specific point in the lifecycle of components.
Such as

a) Whenever property of component changes. or
b) Any view render or
c) Any component created or destroyed.

the lifecycle hooks (special lifecycle method) will be called.*

** No directive or component will implement all of the lifecycle hooks*

## Sequence of hooks

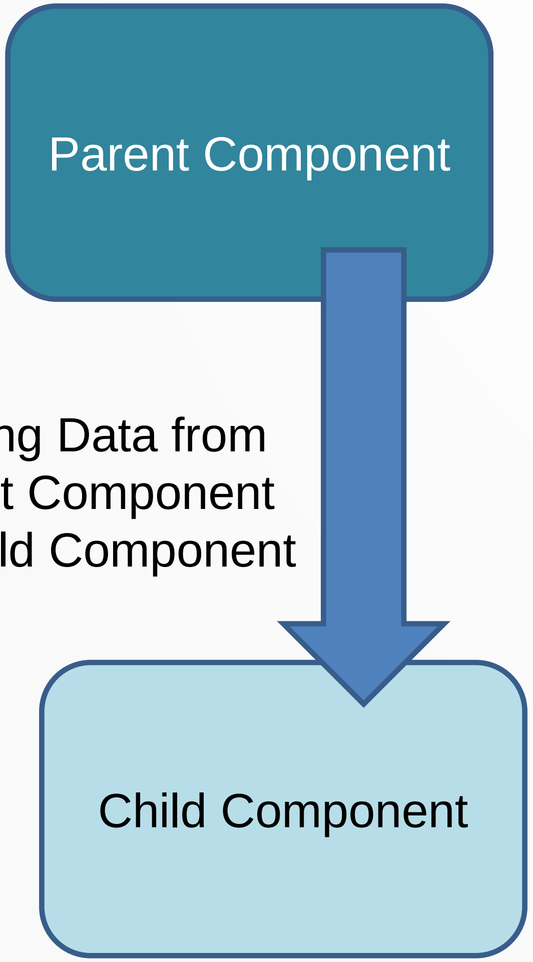


## Decorators

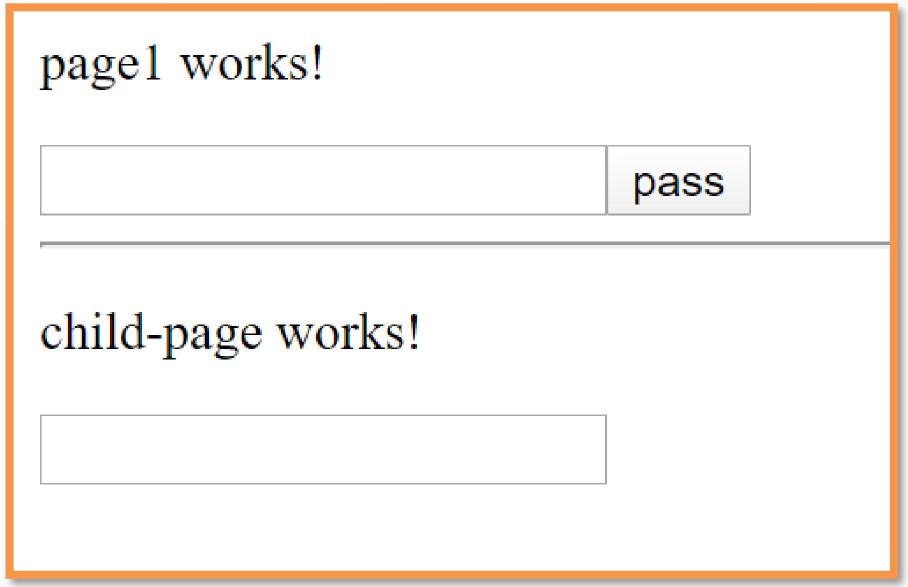
Decorators are a design pattern that is used to separate modification or *decoration* of a class without modifying the original source code. In AngularJS, decorators are functions that allow a service, directive or filter to be modified prior to its usage.

1) @NgModule
2) @Component
3) @Injectable
4) @Pipe
5) @Input & @Output

# @Input() Decorator



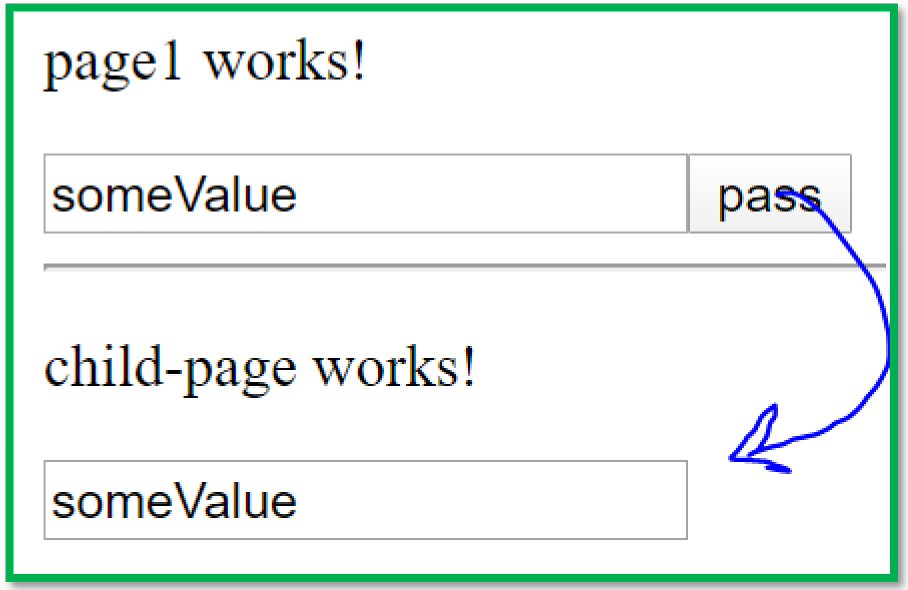
Passing Data from Parent Component to Child Component



page1 works!

pass

child-page works!



page1 works!

someValue

pass

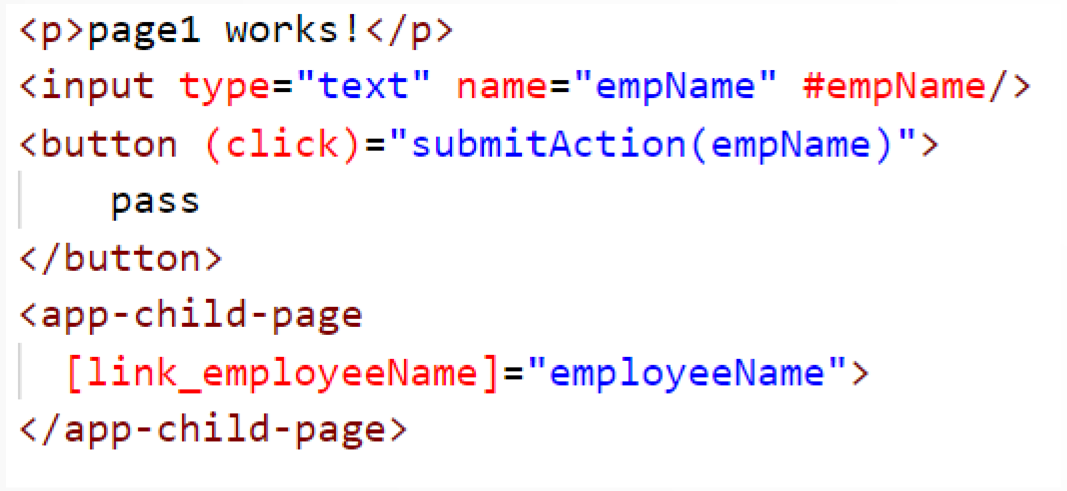
child-page works!

someValue

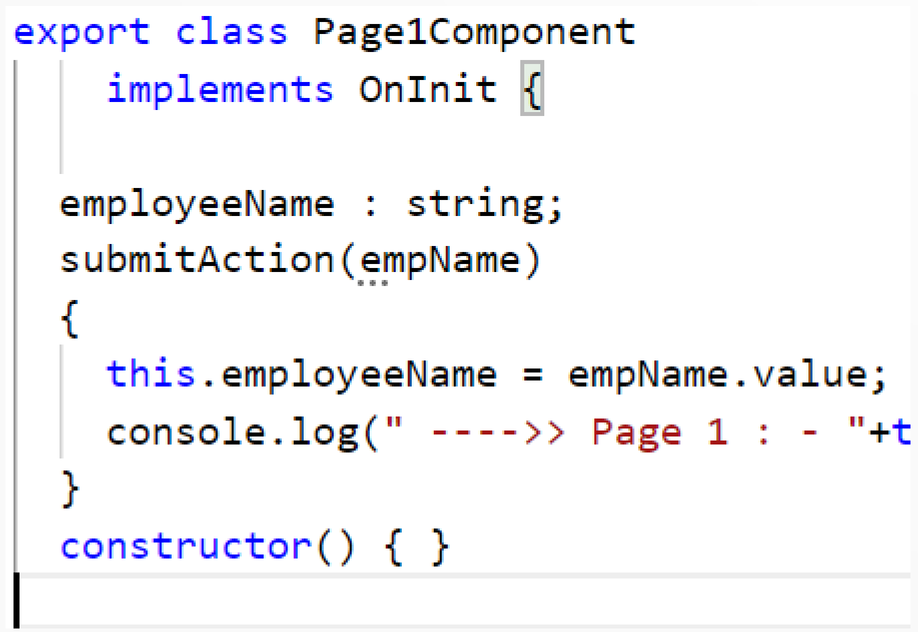
On click event data is passing from parent component to child component

# Continue..

## Parent Component

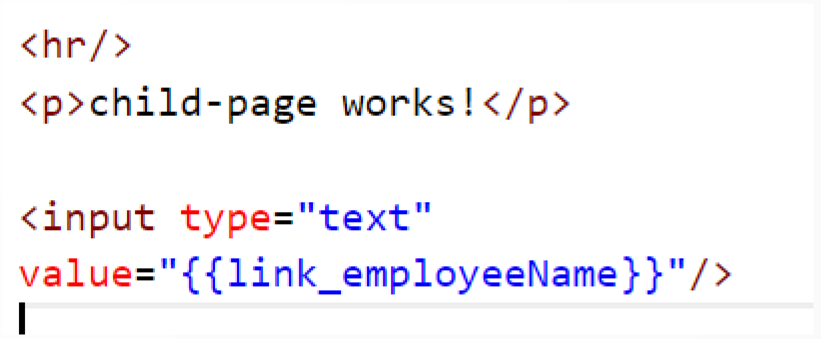


```
<p>page1 works!</p>
<input type="text" name="empName" #empName/>
<button (click)="submitAction(empName)">
    pass
</button>
<app-child-page
  [link_employeeName]="employeeName">
</app-child-page>
```

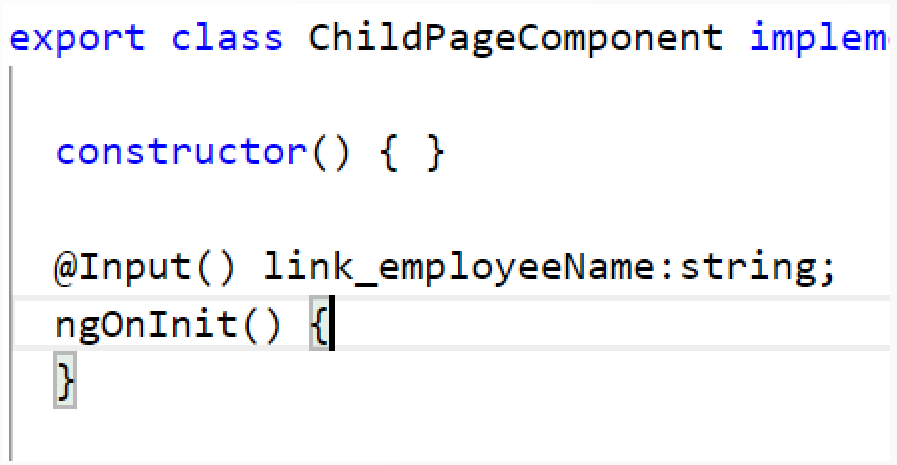


```
export class Page1Component
    implements OnInit {

  employeeName : string;
  submitAction(empName)
  {
    this.employeeName = empName.value;
    console.log(" ---->> Page 1 : - "+t
  }
  constructor() { }
```

## Child Component

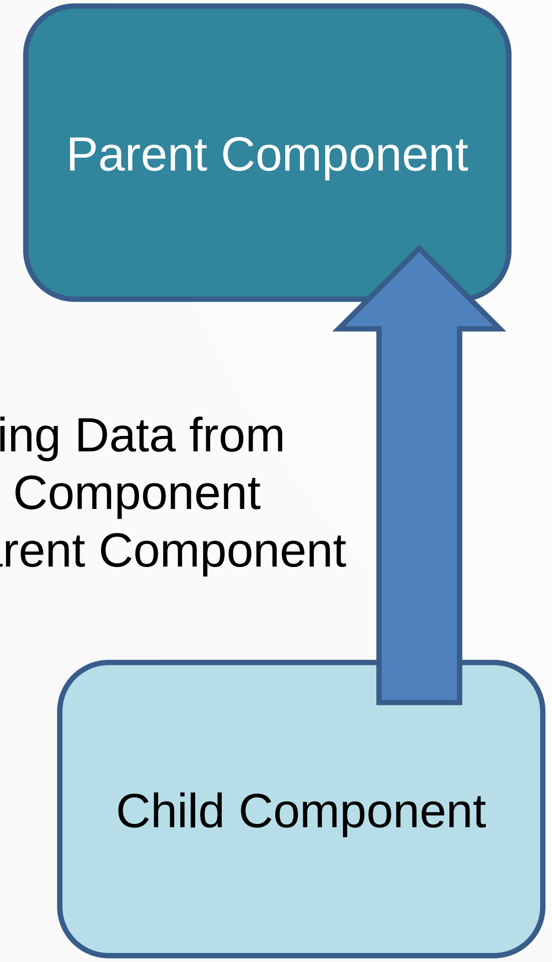


```
<hr/>
<p>child-page works!</p>

<input type="text"
value="{{link_employeeName}}"/>
```

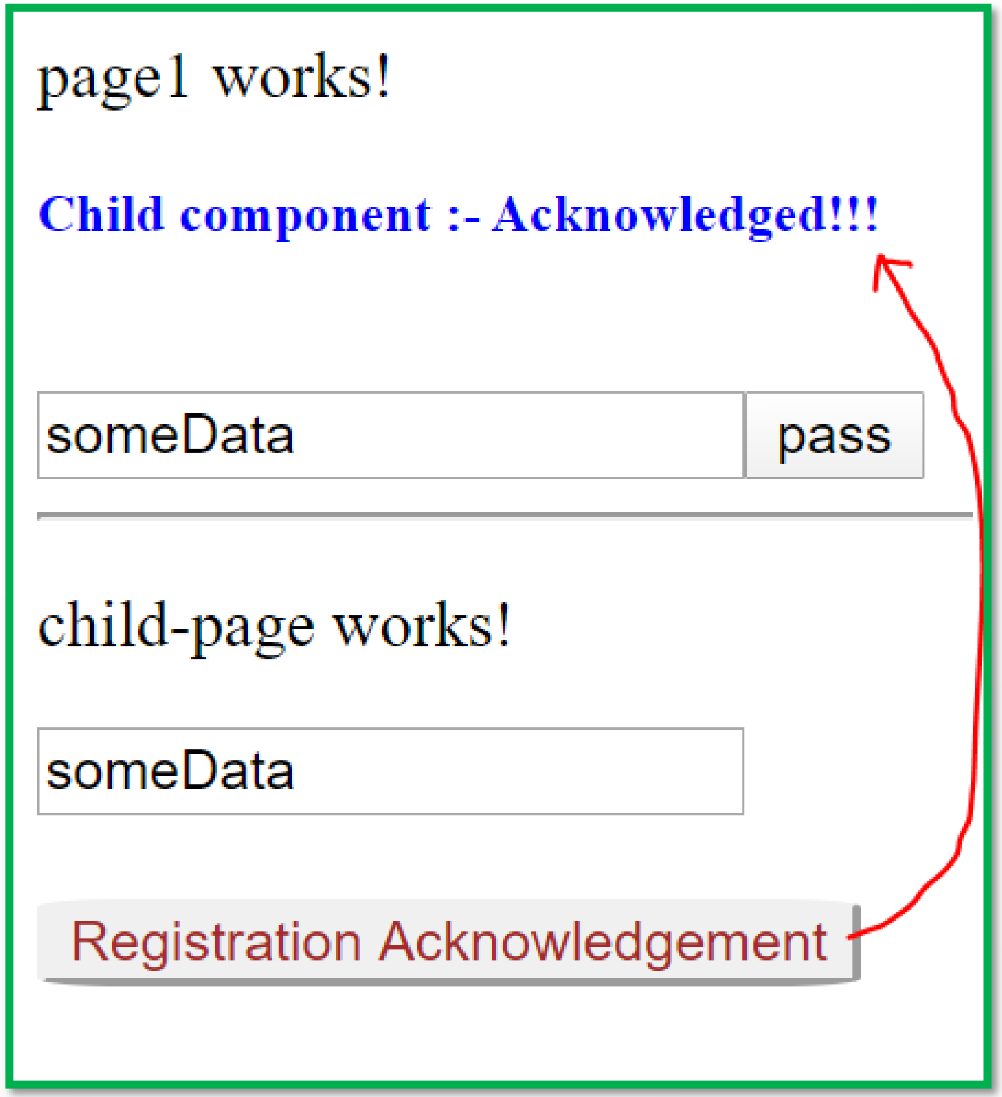


```
export class ChildPageComponent implem

  constructor() { }

  @Input() link_employeeName:string;
  ngOnInit() {
  }
```

# @Output() Decorator



Passing Data from
Child Component
to Parent Component



page1 works!

**Child component :- Acknowledged!!!**

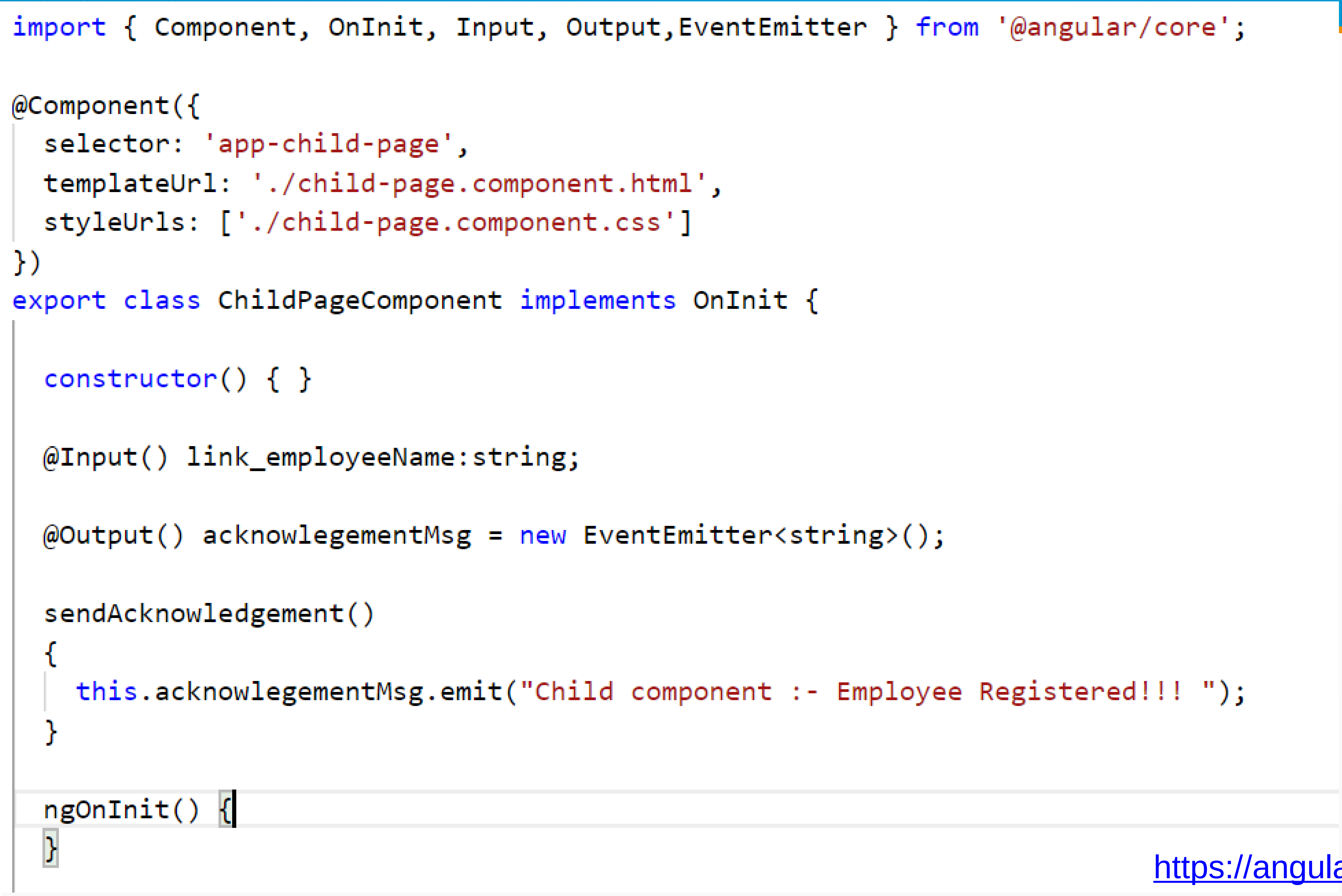
someData

child-page works!

someData

Onclick of button
Parent component get data
from Child Component

# Event Emitter

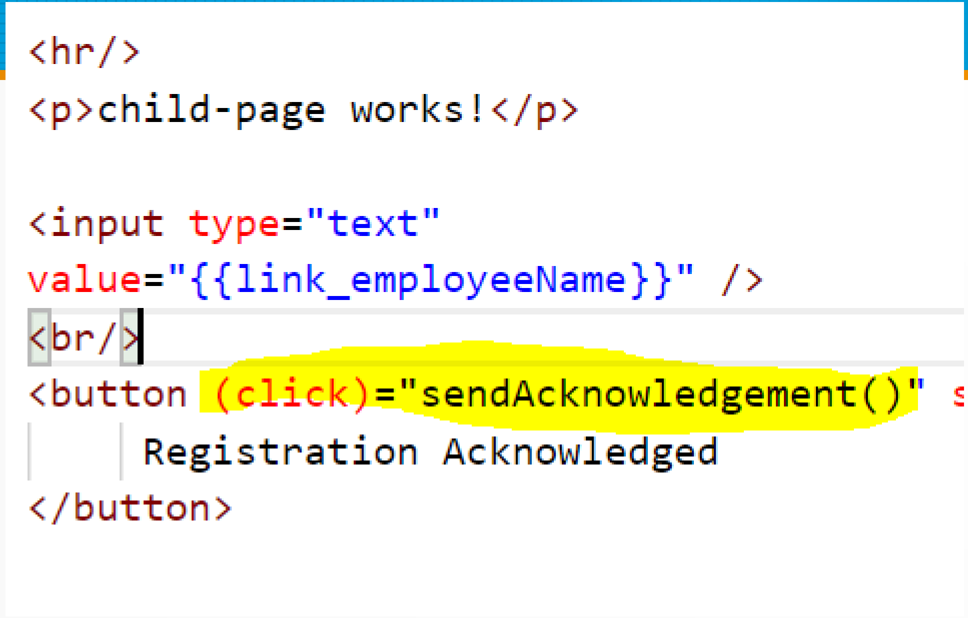


```
import { Component, OnInit, Input, Output,EventEmitter } from '@angular/core';

@Component({
  selector: 'app-child-page',
  templateUrl: './child-page.component.html',
  styleUrls: ['./child-page.component.css']
})
export class ChildPageComponent implements OnInit {

  constructor() { }

  @Input() link_employeeName:string;

  @Output() acknowlegementMsg = new EventEmitter<string>();

  sendAcknowledgement()
  {
    this.acknowlegementMsg.emit("Child component :- Employee Registered!!! ");
  }

  ngOnInit() {
  }
```

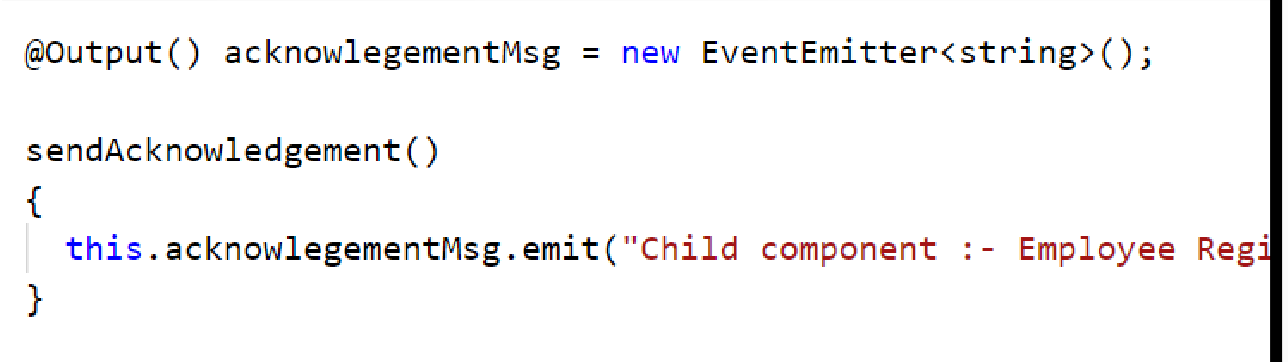
https://angular.io/api/core/EventEmitter

# Code

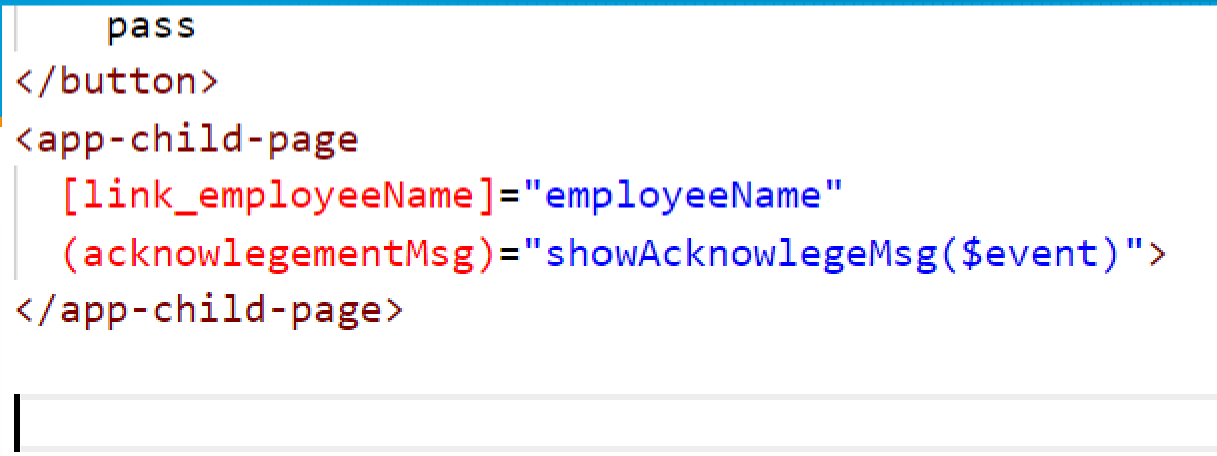
https://github.com/mkjitsolution/angular_ComponentInteraction



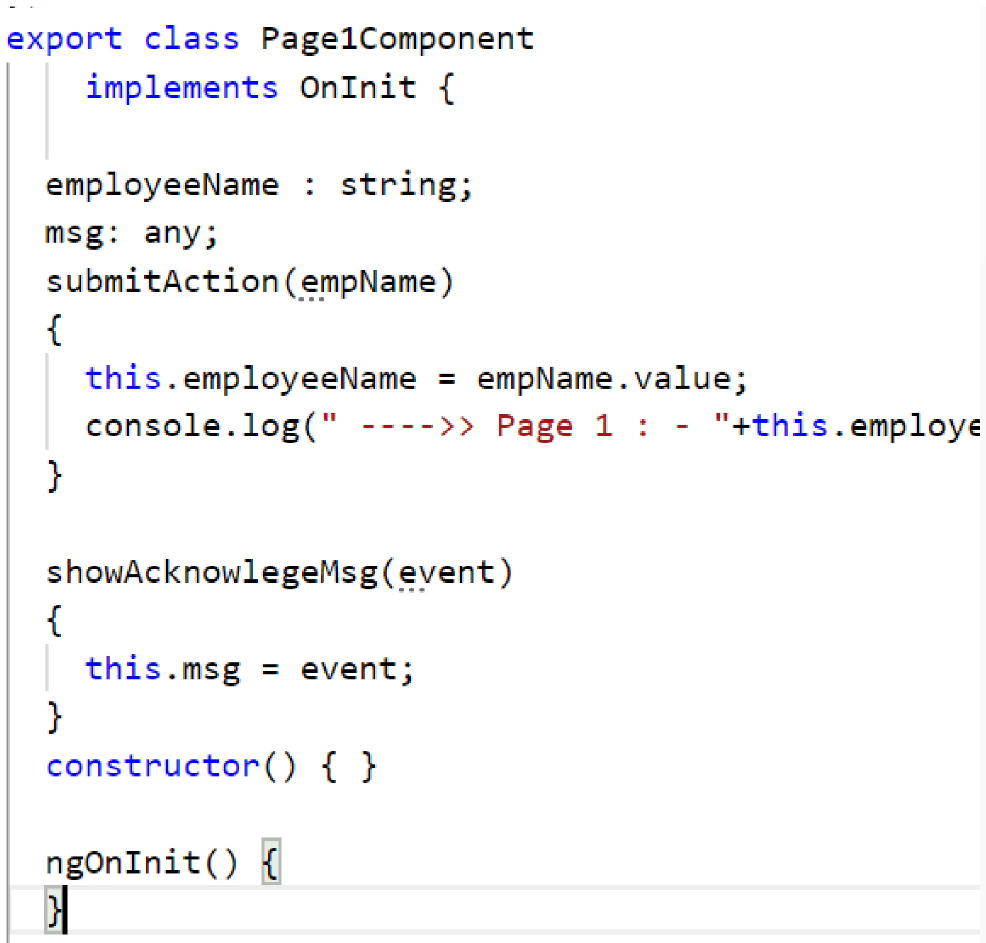
```
<hr/>
<p>child-page works!</p>

<input type="text"
value="{{link_employeeName}}" />
<br/>
<button (click)="sendAcknowledgement()" s
    Registration Acknowledged
</button>
```



```
@Output() acknowlegementMsg = new EventEmitter<string>();

sendAcknowledgement()
{
  this.acknowlegementMsg.emit("Child component :- Employee Regi
}
```



```
    pass
</button>
<app-child-page
  [link_employeeName]="employeeName"
  (acknowlegementMsg)="showAcknowlegeMsg($event)">
</app-child-page>
```



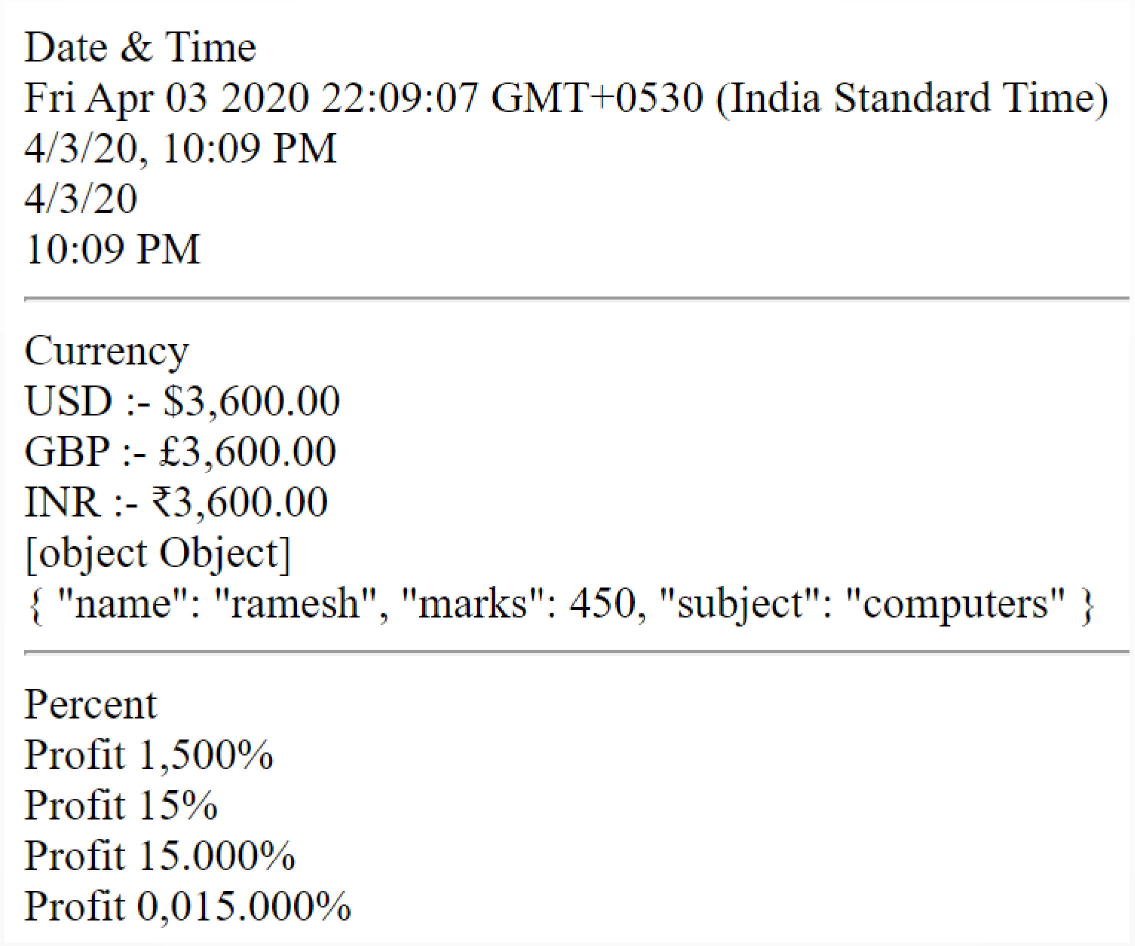
```
export class Page1Component
  implements OnInit {

  employeeName : string;
  msg: any;
  submitAction(empName)
  {
    this.employeeName = empName.value;
    console.log(" ---->> Page 1 : - "+this.employe
  }

  showAcknowlegeMsg(event)
  {
    this.msg = event;
  }
  constructor() { }

  ngOnInit() {
  }
```

Notice : add EventEmitter in import statement from angular/core

# Pipes

Pipes allowing to transform data before rendering them in view.

1) Lowercase
2) Uppercase
3) Titlecase
4) Slice
5) Json
6) Number
7) Percent
8) Currency
9) Date



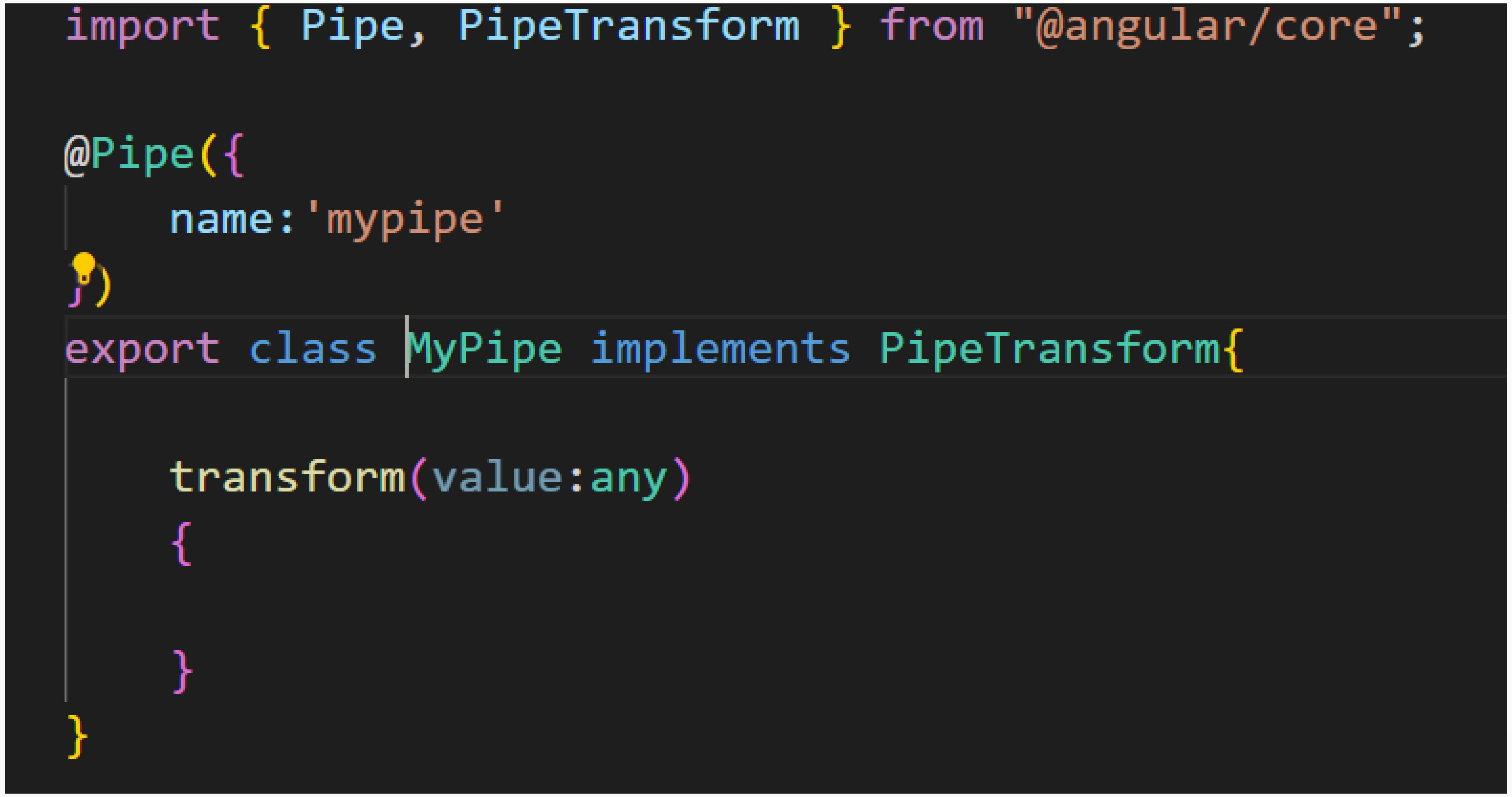
```
Date & Time
Fri Apr 03 2020 22:09:07 GMT+0530 (India Standard Time)
4/3/20, 10:09 PM
4/3/20
10:09 PM

Currency
USD :- $3,600.00
GBP :- £3,600.00
INR :- ₹3,600.00
[object Object]
{ "name": "ramesh", "marks": 450, "subject": "computers" }

Percent
Profit 1,500%
Profit 15%
Profit 15.000%
Profit 0,015.000%
```

https://www.concretepage.com/angular-2/angular-2-decimal-pipe-percent-pipe-and-currency-pipe-example

# Creation of Custom Pipes

In order to create Custom pipes.

1) Create a Separate class which implements PipeTrasform interface
2) Implement method **“transform”.**
3) Decorate the class with @Pipe decorator



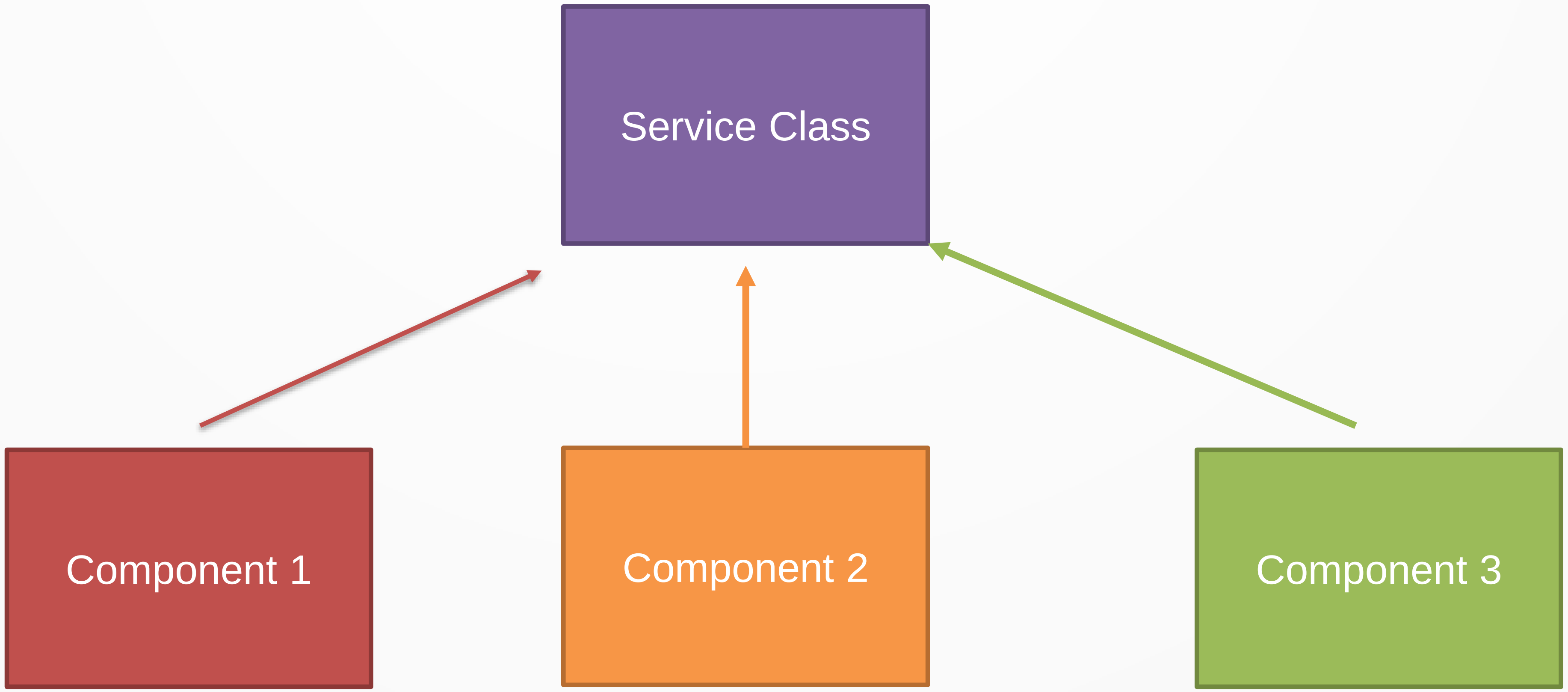
```
import { Pipe, PipeTransform } from "@angular/core";

@Pipe({
    name:'mypipe'
})
export class MyPipe implements PipeTransform{

    transform(value:any)
    {

    }
}
```

## Services

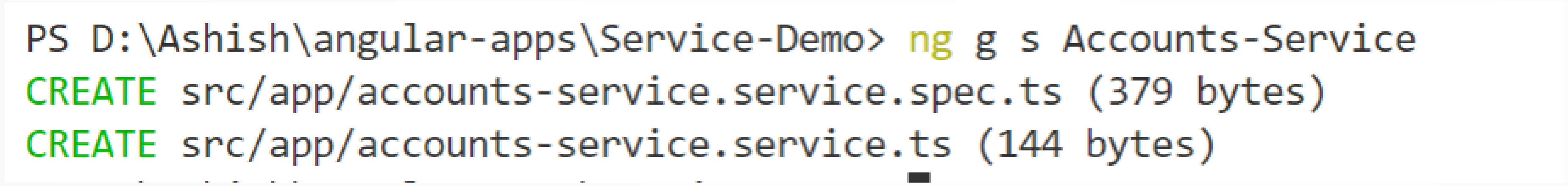
Services is the way of Separation of concerns

1) @Injectable

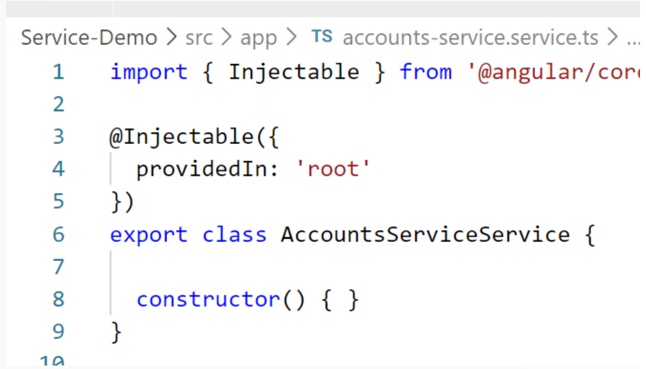
It is a separate class with Dependency Injection design pattern



# Services – Development Process

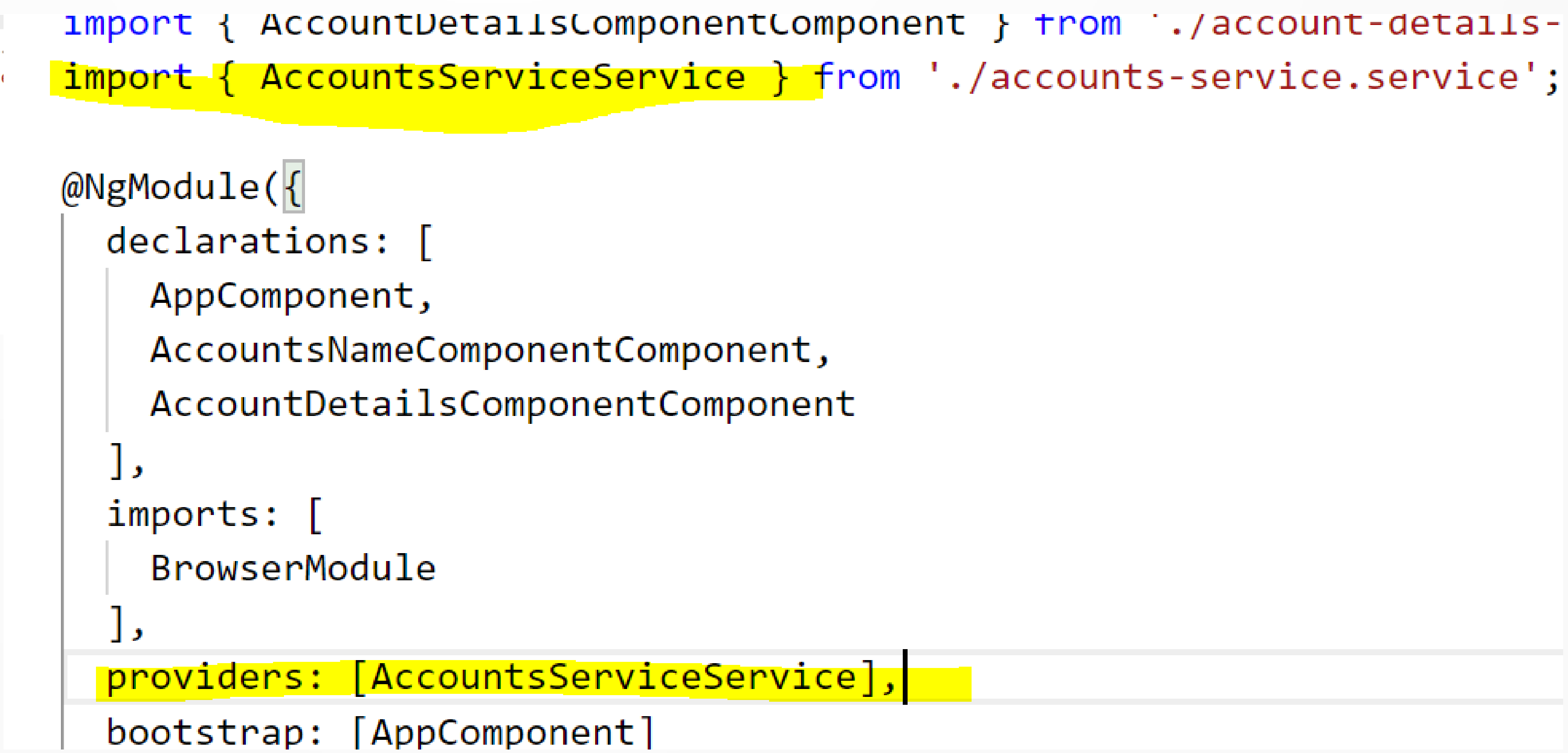


```
PS D:\Ashish\angular-apps\Service-Demo> ng g s Accounts-Service
CREATE src/app/accounts-service.service.spec.ts (379 bytes)
CREATE src/app/accounts-service.service.ts (144 bytes)
```



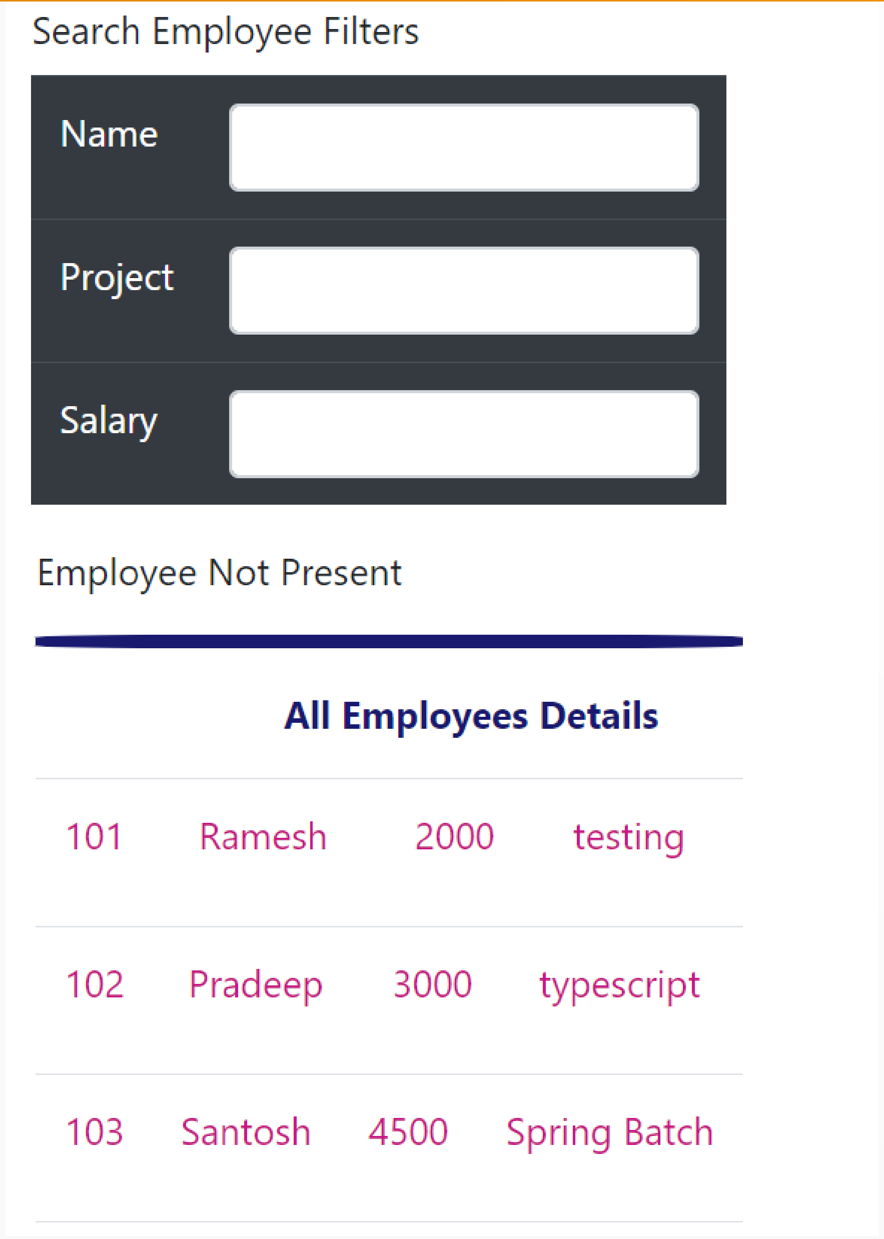
Service-Demo > src > app > TS accounts-service.service.ts > ...

```
import { Injectable } from '@angular/core

@Injectable({
  providedIn: 'root'
})
export class AccountsServiceService {

  constructor() { }
}
```



```
import { AccountDetailsComponentComponent } from './account-details-
import { AccountsServiceService } from './accounts-service.service';

@NgModule({
  declarations: [
    AppComponent,
    AccountsNameComponentComponent,
    AccountDetailsComponentComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [AccountsServiceService],
  bootstrap: [AppComponent]
```

# Assignment


Search Employee Filters

| | |
|---|---|
| Name | |
| Project | |
| Salary | |

Employee Not Present

**All Employees Details**

| | | | |
|---|---|---|---|
| 101 | Ramesh | 2000 | testing |
| 102 | Pradeep | 3000 | typescript |
| 103 | Santosh | 4500 | Spring Batch |

Create an Application to Filter out Employees based on following Criteria

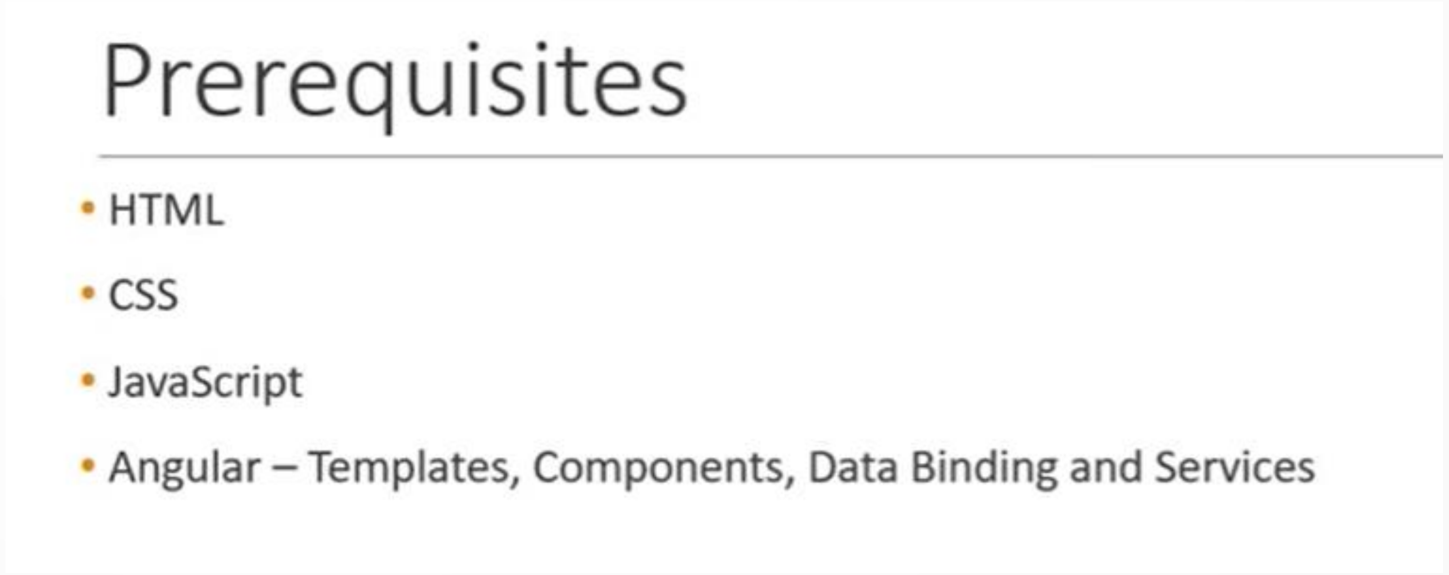
1) By Name or
2) Based on Project or
3) Based on salary (1500-3500, return two employees record)

# Angular Forms

Forms are the vital part of business application

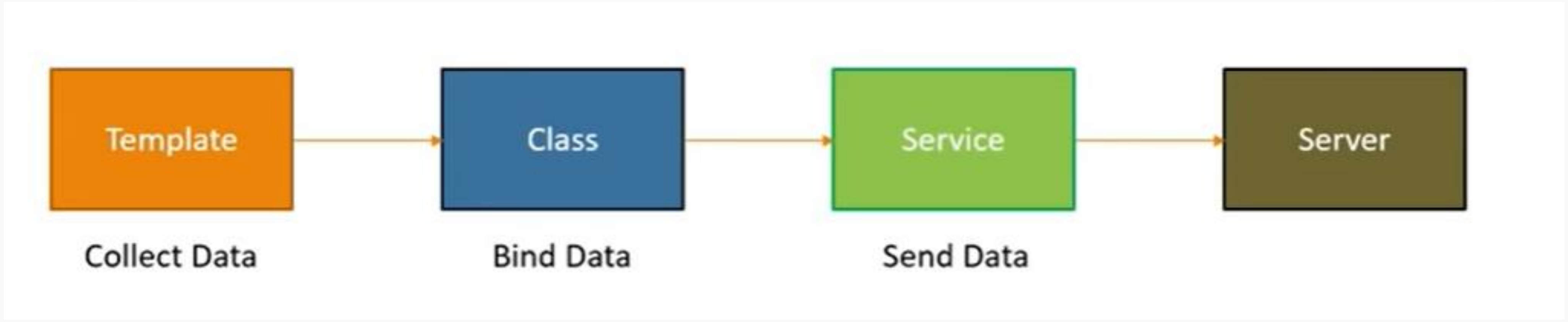
**Developers Task**

1) Data Binding
2) Change Tracking
3) Validation
4) Visual Feedback
5) Error Messages
6) Form submission

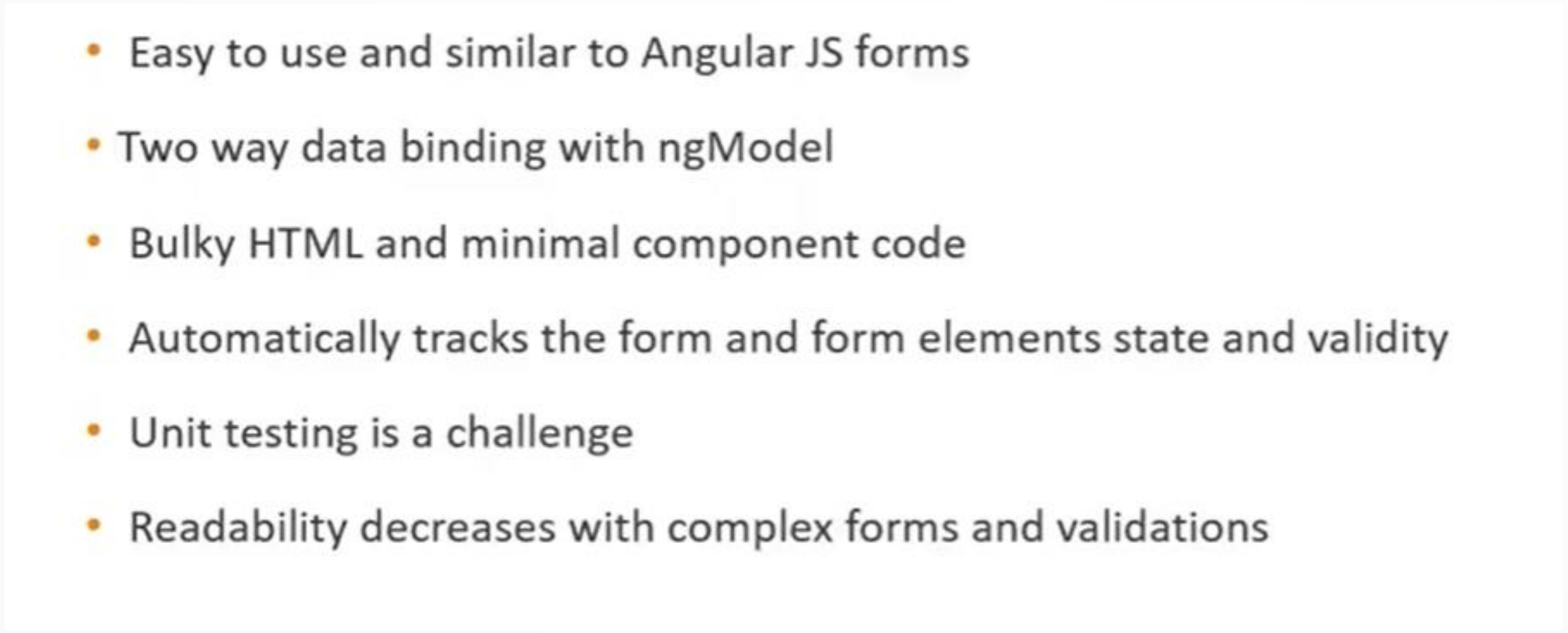


## Prerequisites

- HTML
- CSS
- JavaScript
- Angular – Templates, Components, Data Binding and Services



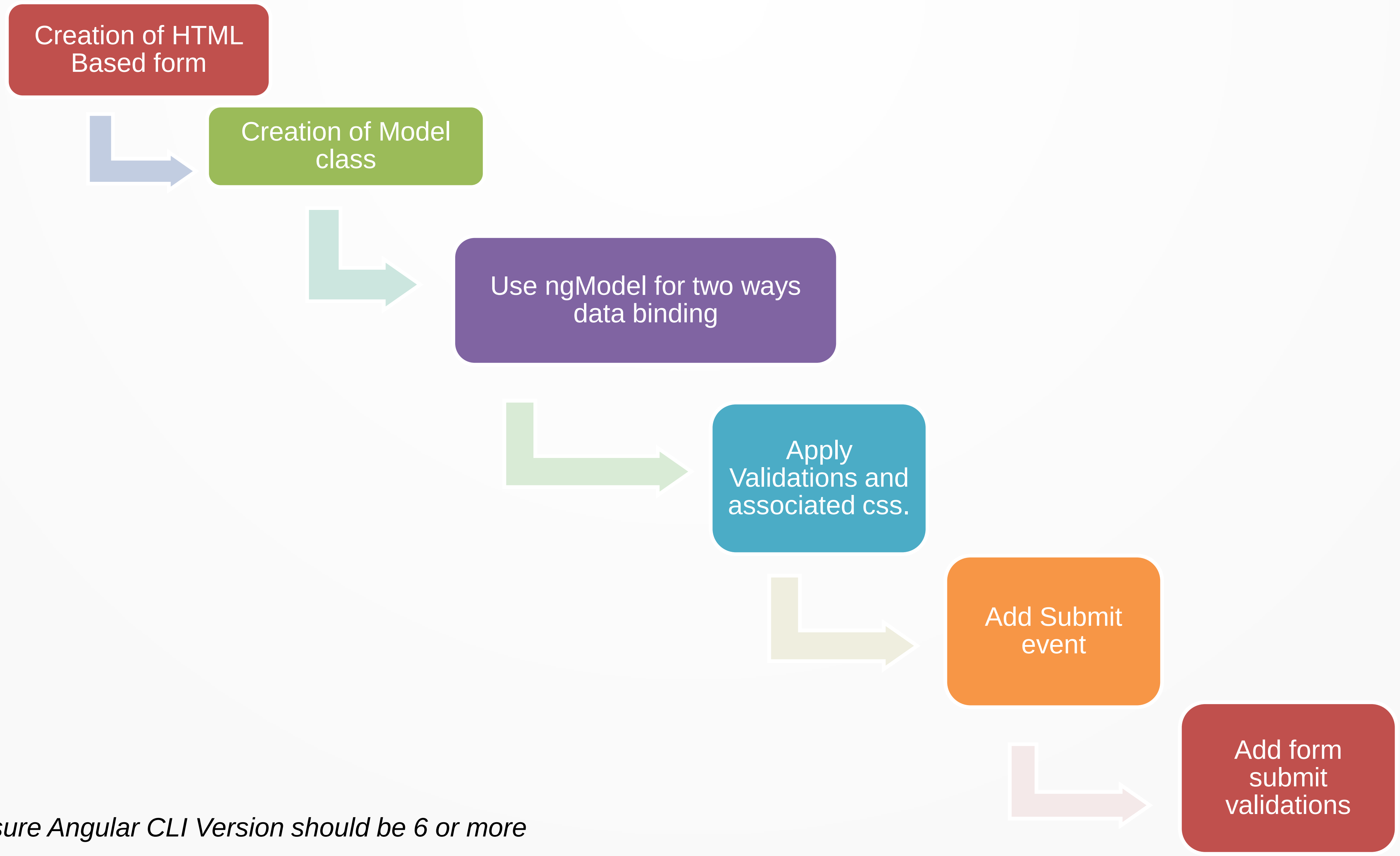
# Template Driven Form (TDF)



- Easy to use and similar to Angular JS forms
- Two way data binding with ngModel
- Bulky HTML and minimal component code
- Automatically tracks the form and form elements state and validity
- Unit testing is a challenge
- Readability decreases with complex forms and validations

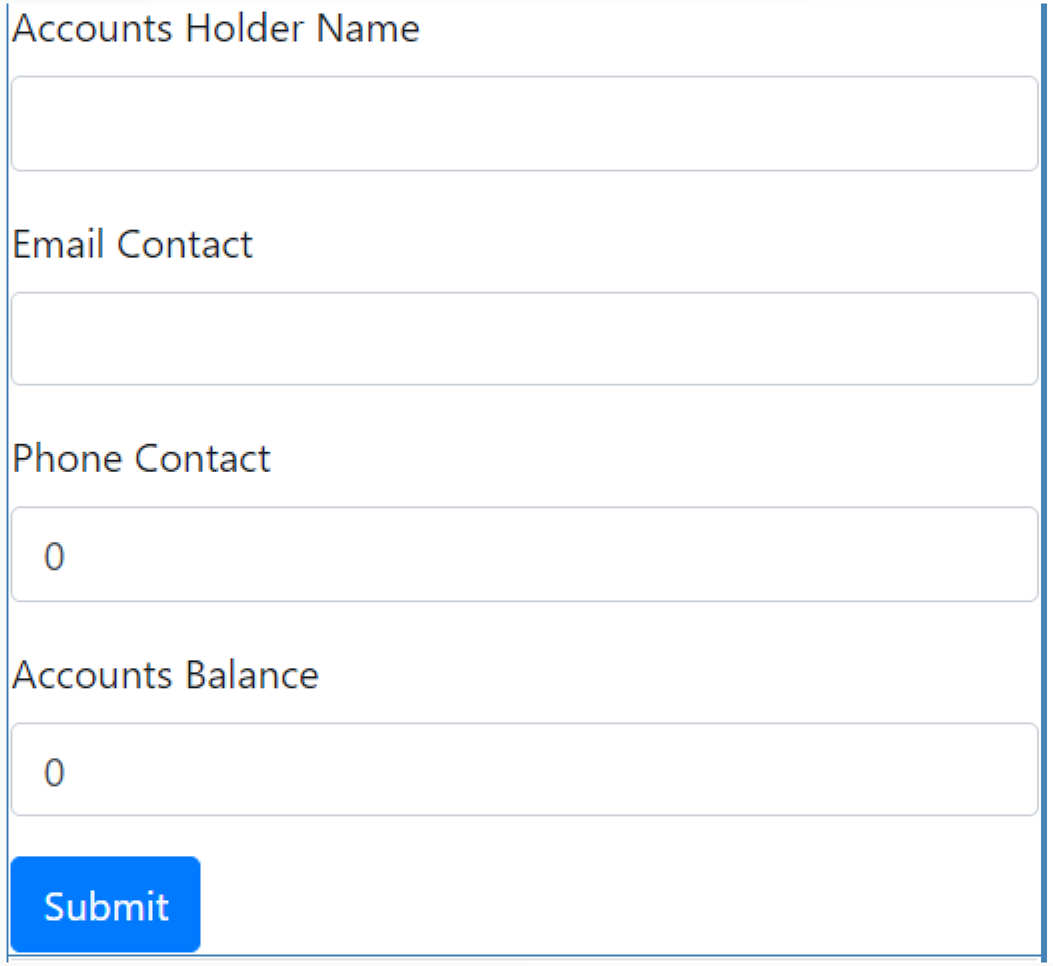
Git resource :- https://github.com/mkjitsolution/template_driven_forms

# Development Process


Creation of HTML Based form
Creation of Model class
Use ngModel for two ways data binding
Apply Validations and associated css.
Add Submit event
Add form submit validations

*Note : Make sure Angular CLI Version should be 6 or more*

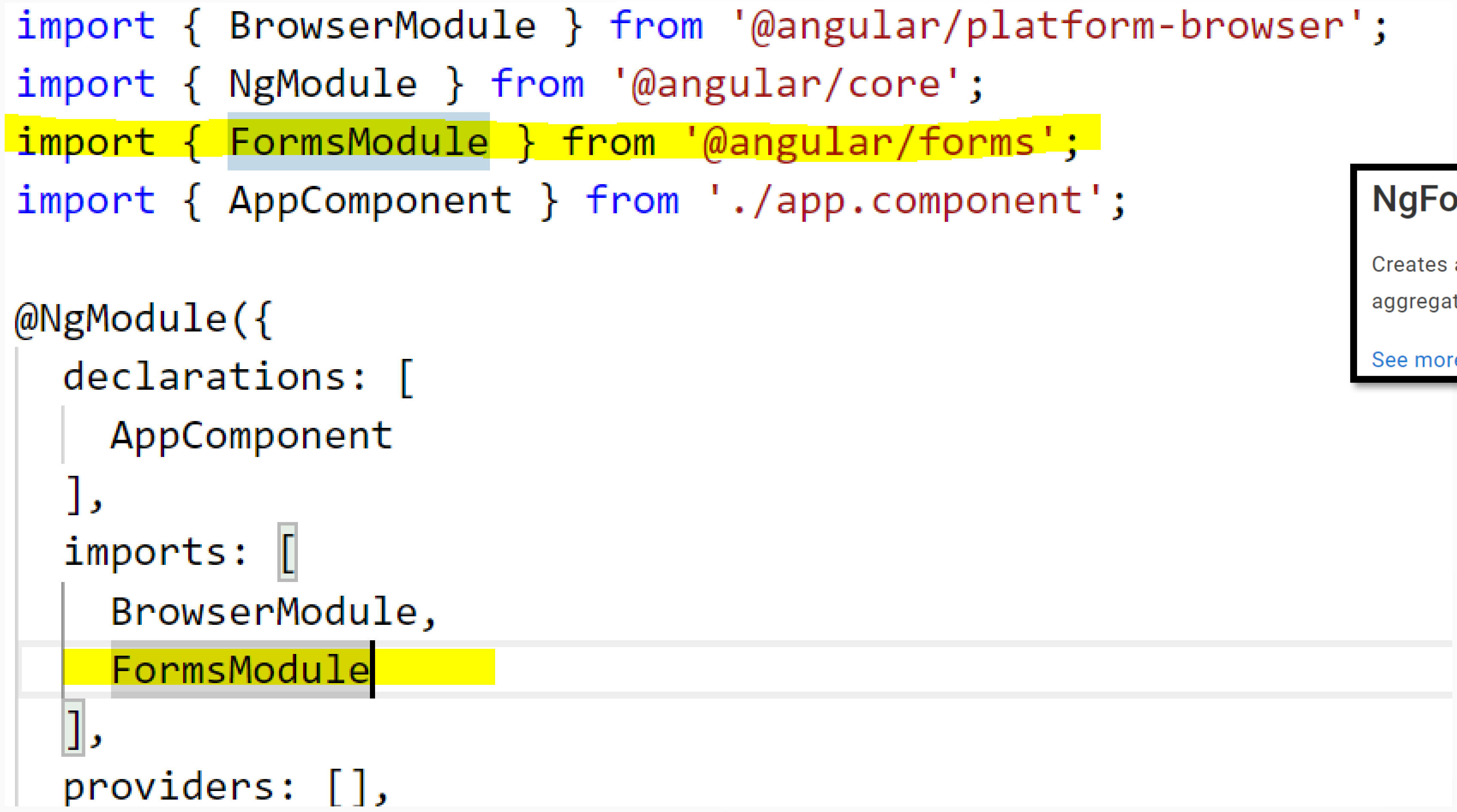
# Step 1 : Create Form



|  |
|---|
| Accounts Holder Name |
|  |
| Email Contact |
|  |
| Phone Contact |
| 0 |
| Accounts Balance |
| 0 |

# Step 1-B : Binding Form With Component

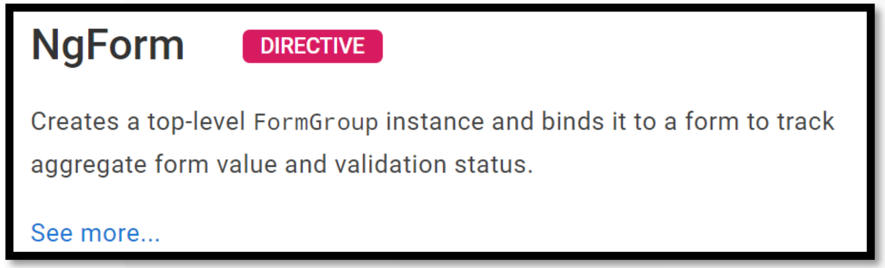
**Step 1 : Add form modules in app.module.ts**



```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
```

Discuss about :

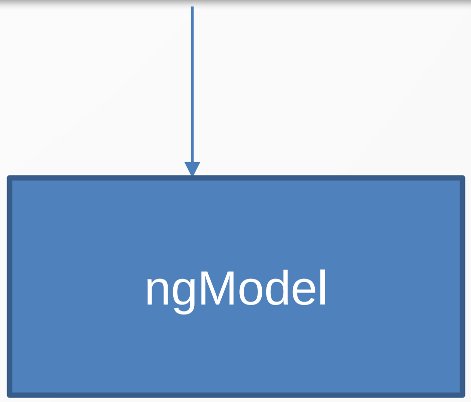
ngForm



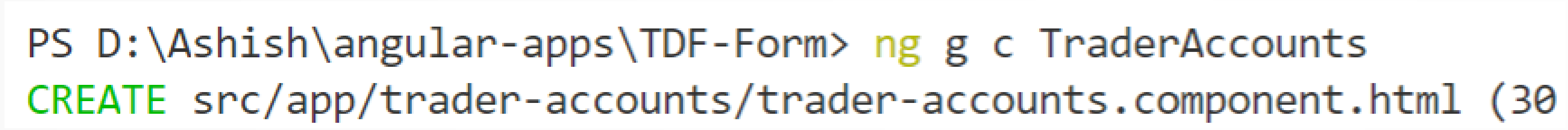
NgForm DIRECTIVE

Creates a top-level FormGroup instance and binds it to a form to track aggregate form value and validation status.

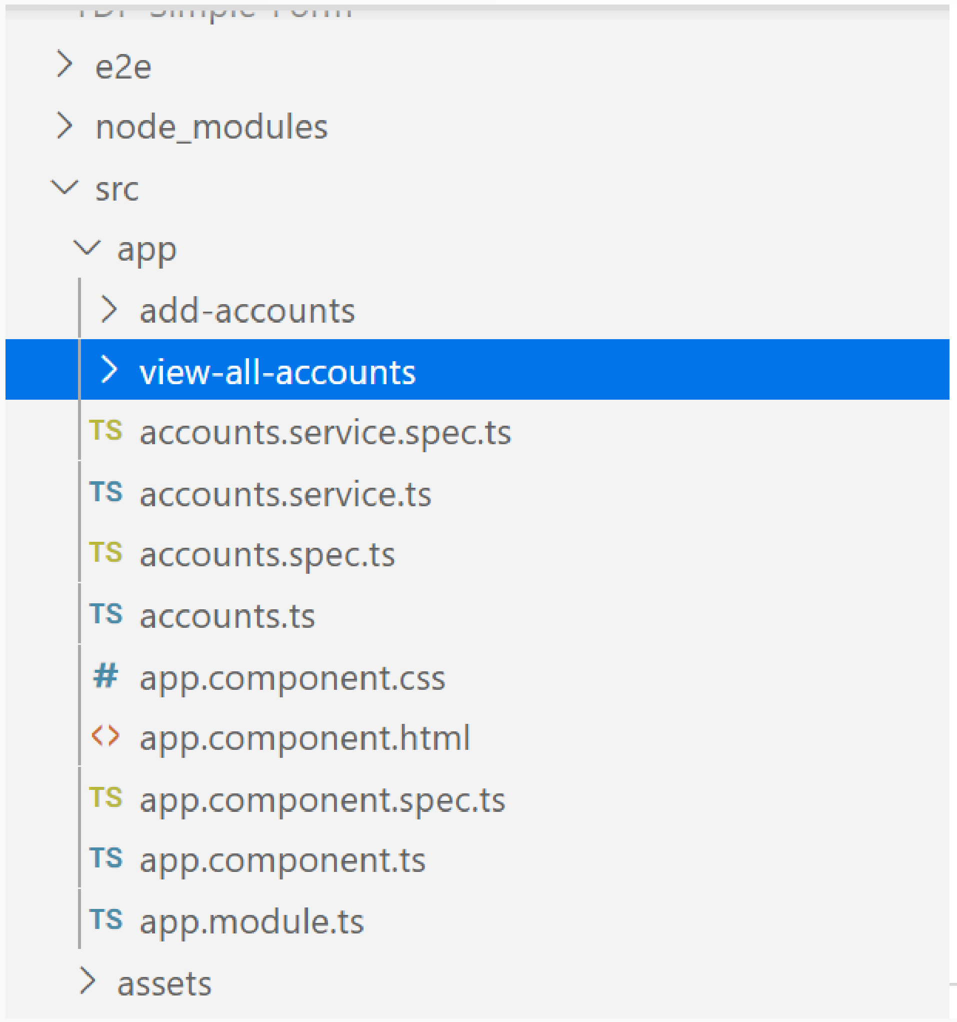
See more...



# Step 2: Generate Model Class



```
PS D:\Ashish\angular-apps\TDF-Form> ng g c TraderAccounts
CREATE src/app/trader-accounts/trader-accounts.component.html (30
```

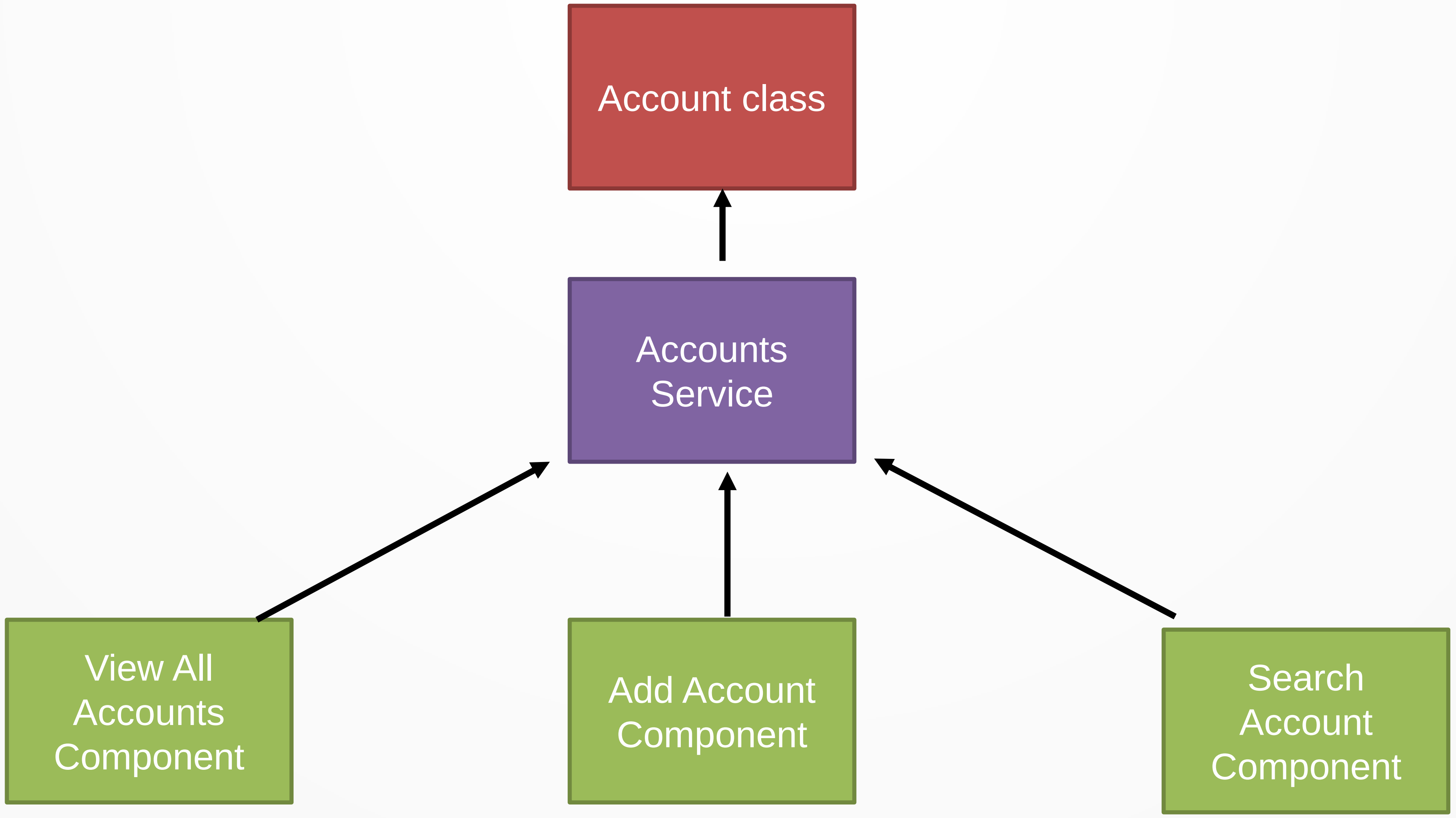


```
  > e2e
  > node_modules
  v src
    v app
      > add-accounts
      > view-all-accounts
      TS accounts.service.spec.ts
      TS accounts.service.ts
      TS accounts.spec.ts
      TS accounts.ts
      # app.component.css
      <> app.component.html
      TS app.component.spec.ts
      TS app.component.ts
      TS app.module.ts
    > assets
```

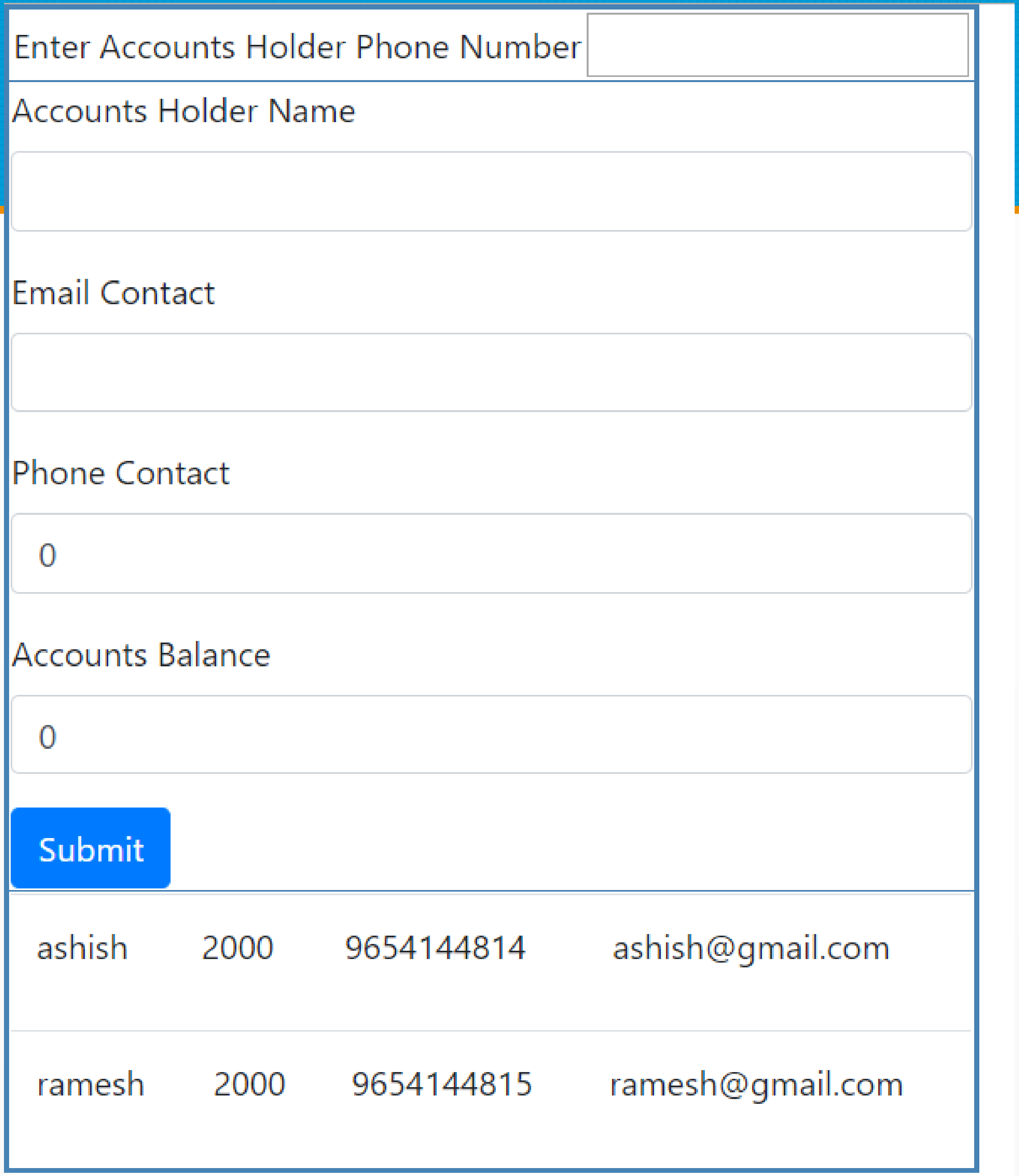
1) Model class will be responsible to bind form values with class properties
   using ngModel attribute of input controls

2) Model class at least consist a parameterized constructor.

# Application Architecture



# Application



Enter Accounts Holder Phone Number

Accounts Holder Name

Email Contact

Phone Contact

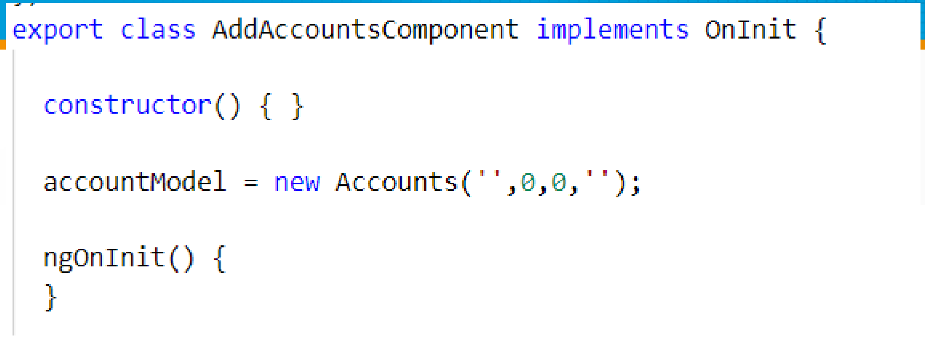
0

Accounts Balance

0

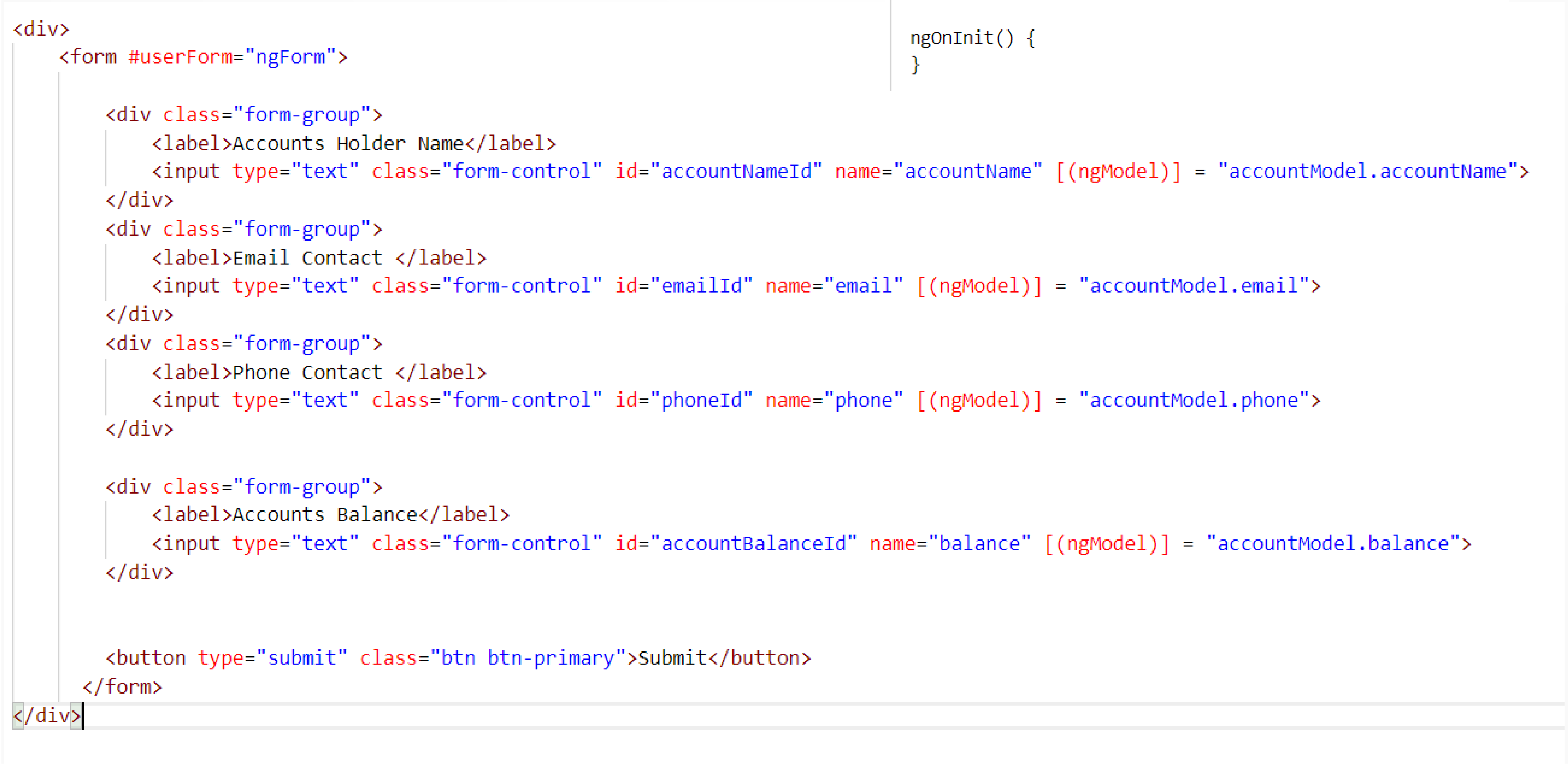
| | | | |
|---|---|---|---|
| ashish | 2000 | 9654144814 | ashish@gmail.com |
| ramesh | 2000 | 9654144815 | ramesh@gmail.com |

# Step 3 : Initialize Model class for two ways data binding



```
export class AddAccountsComponent implements OnInit {

  constructor() { }

  accountModel = new Accounts('',0,0,'');

  ngOnInit() {
  }
```

Create Model Object in Component
And associate model properties with controls



```
<div>
    <form #userForm="ngForm">

        <div class="form-group">
            <label>Accounts Holder Name</label>
            <input type="text" class="form-control" id="accountNameId" name="accountName" [(ngModel)] = "accountModel.accountName">
        </div>
        <div class="form-group">
            <label>Email Contact </label>
            <input type="text" class="form-control" id="emailId" name="email" [(ngModel)] = "accountModel.email">
        </div>
        <div class="form-group">
            <label>Phone Contact </label>
            <input type="text" class="form-control" id="phoneId" name="phone" [(ngModel)] = "accountModel.phone">
        </div>

        <div class="form-group">
            <label>Accounts Balance</label>
            <input type="text" class="form-control" id="accountBalanceId" name="balance" [(ngModel)] = "accountModel.balance">
        </div>


        <button type="submit" class="btn btn-primary">Submit</button>
      </form>
</div>
```

## Angular Built In Validations

Angular Provides few built in validations,

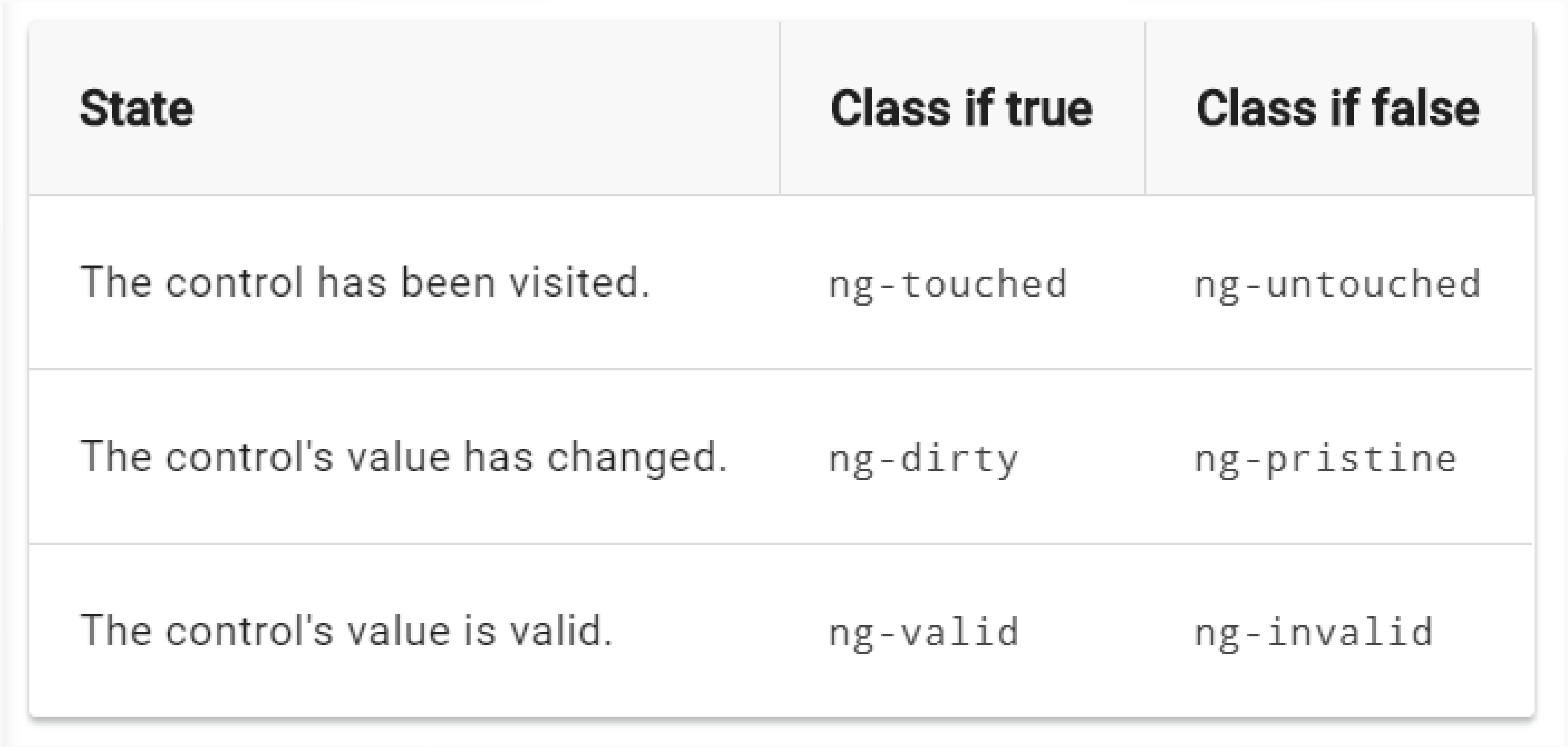
- required.
- minlength.
- maxlength.
- Pattern

These required, minlength, maxlength and pattern attributes are *already* in the official HTML specification.

They are a core part of HTML and we don’t actually need Angular in order to use them.

If they are present in a form then the browser will perform some default validation itself.

## Understanding ngModel Validation properties



| State | Class if true | Class if false |
|---|---|---|
| The control has been visited. | `ng-touched` | `ng-untouched` |
| The control's value has changed. | `ng-dirty` | `ng-pristine` |
| The control's value is valid. | `ng-valid` | `ng-invalid` |

*Note : form.valid : property is used to identify form valid or invalid state.*

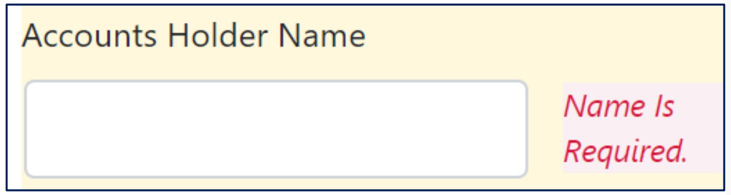
[Source : https://angular.io/guide/forms](https://angular.io/guide/forms)

# Continue…

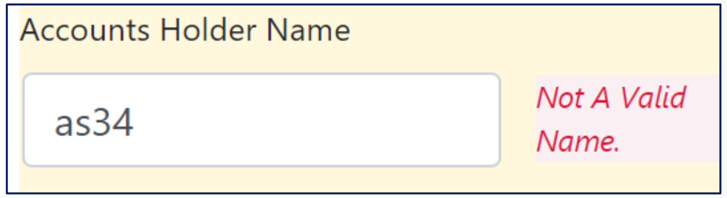
| Property | Description |
|---|---|
| error | error object contains all the validation attributes applied to the specified element. |
| pristine | Returns true if the user has not interacted with control yet else returns false. |
| valid | Returns true if the model is valid |
| invalid | Returns true if the model is invalid |
| dirty | Returns true if user changed the value of model at least once |
| touched | Returns true if the user has tabbed out from the control. |
| untouched | Returns true if the user has not tabbed out from the control. |

# Step 4 : Applying Validations

```
<div class="form-group">
    <label>Accounts Holder Name</label>
    <table>
        <tr>
            <td> <input type="text" class="form-control" id="accountNameId" name="accountName"
                [(ngModel)] = "accountModel.accountName"
                required pattern="[a-zA-Z][a-zA-Z ]+"
                #name="ngModel">
            </td>
            <td>
                <div *ngIf="name.invalid && (name.dirty || name.touched)" class="myerror">
                    <div *ngIf="name.invalid == true">
                        <div *ngIf="name.errors.required">
                            Name is required.
                        </div>
                        <div *ngIf="name.errors.pattern">
                            not a valid name.
                        </div>
                    </div>
                </div>
             </td>
        </tr>
    </table>
```


Accounts Holder Name

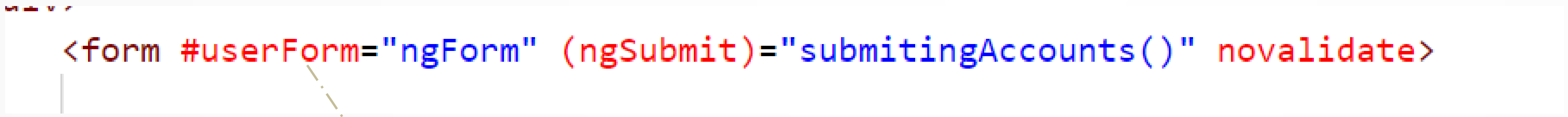
Name Is Required.


Accounts Holder Name

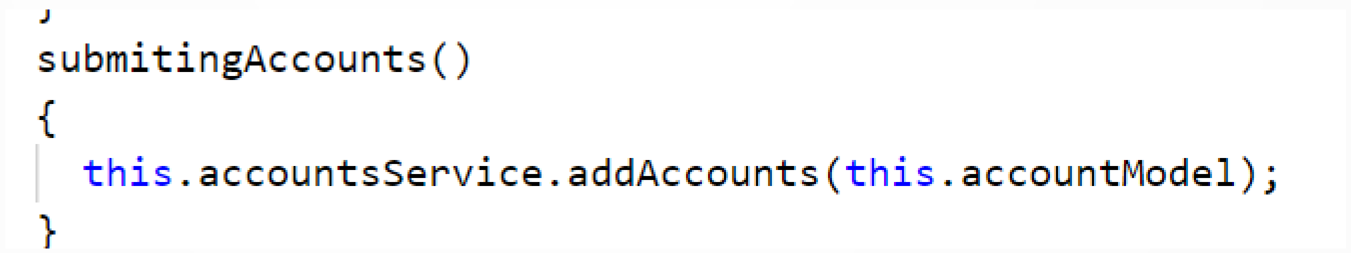
as34

Not A Valid Name.

## Step 5 : Submitting the Form

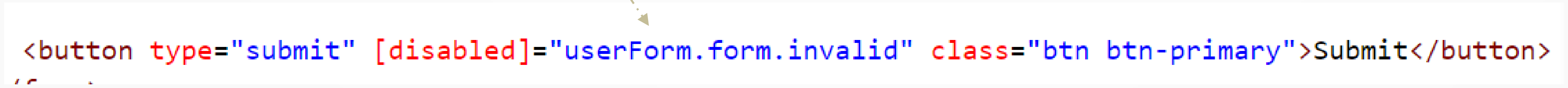


```
<form #userForm="ngForm" (ngSubmit)="submitingAccounts()" novalidate>
```



```
submitingAccounts()
{
  this.accountsService.addAccounts(this.accountModel);
}
```

## Step 6 : Adding form validation



```
<button type="submit" [disabled]="userForm.form.invalid" class="btn btn-primary">Submit</button>
```

## Reactive Forms

- ✓ Reactive forms are more explicit as they manage from component class.
- ✓ It has Structured data model and form validations are handled through functions.

---

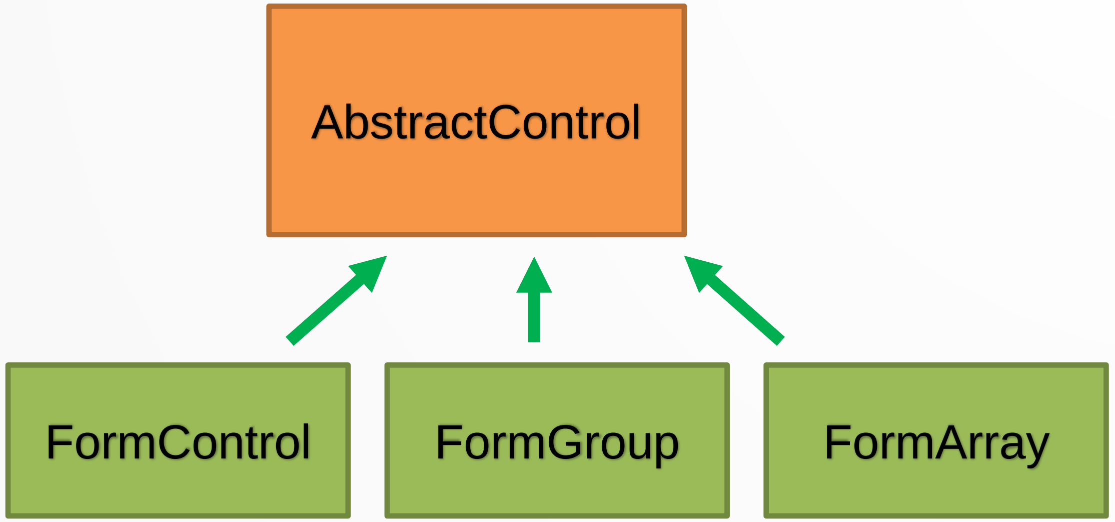
## FormControl Class

https://angular.io/api/forms/FormControl

- Angular FormControl is an inbuilt class that is used to get and set values and validation of the form control fields like <input> or <select>.
- The FormControl tracks the value and validation status of an individual form control.
- It can be used standalone as well as with a parent form.

## Understanding FormControl

Form control is one of the fundamental building block of Angular Forms, along with FormGroup and FormArray



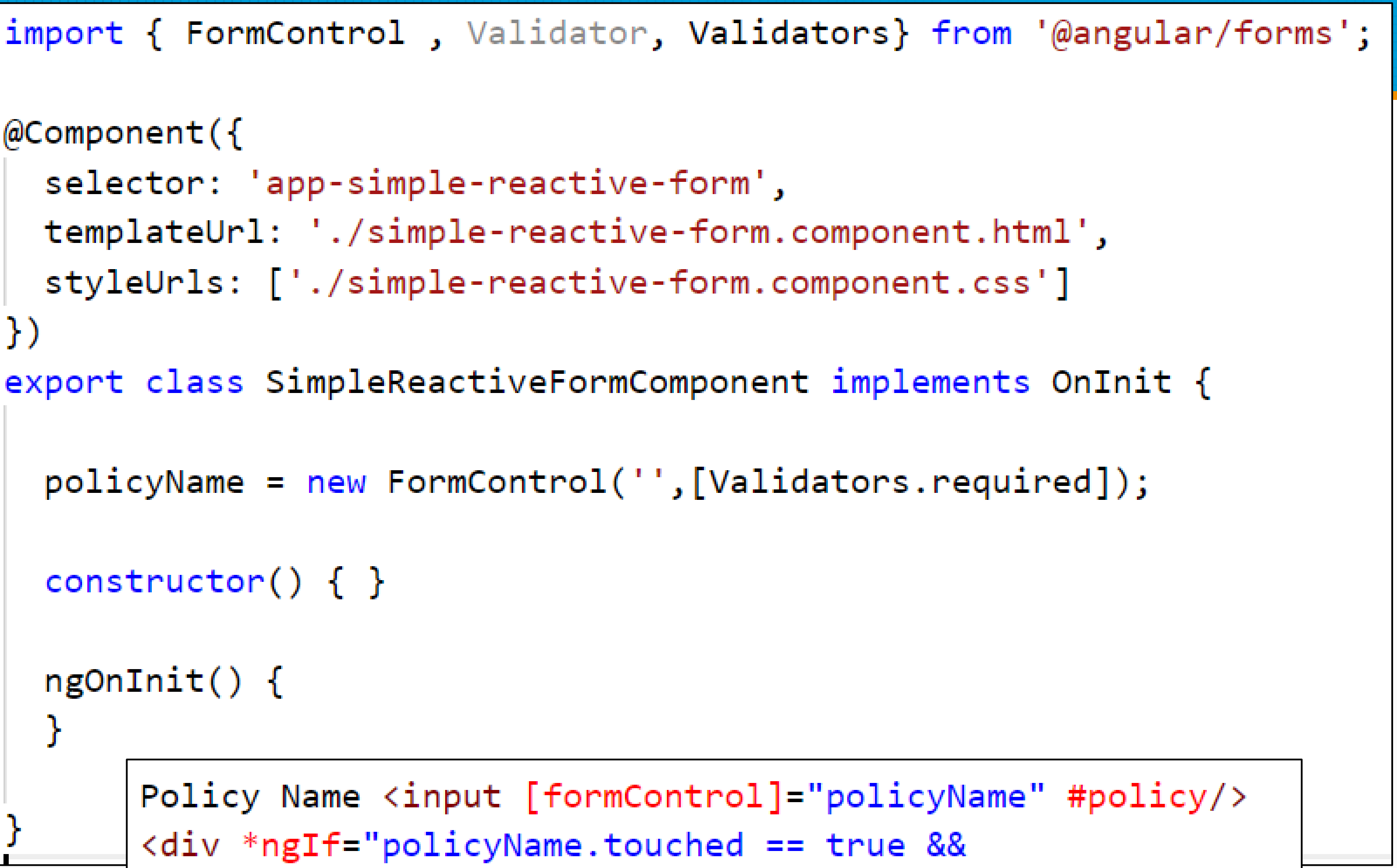
AbstractControl interface provides some of the shared behaviour like

1) Running validators
2) Calculating status and resetting state.

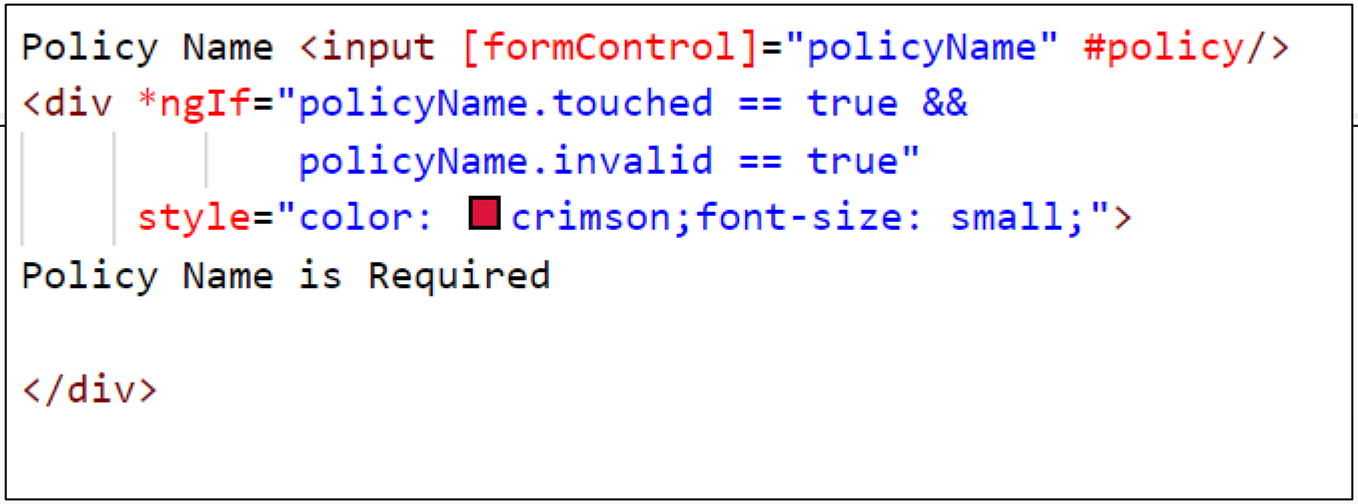
It also defines the properties that are shared between all sub-classes, like value, valid, and dirty. It shouldn't be instantiated directly.

For every form control such as text, checkbox, radio button, we need to create the instance of **FormControl** in our component.

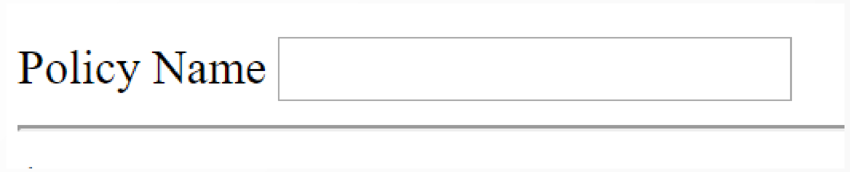
# Component Class & HTML-Template



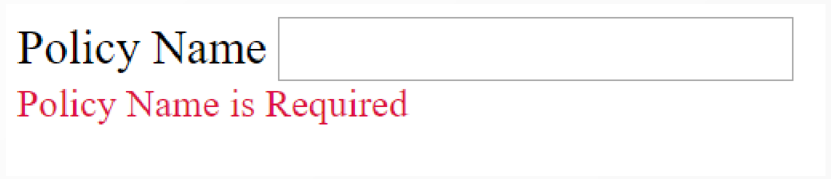
```
import { FormControl , Validator, Validators} from '@angular/forms';

@Component({
  selector: 'app-simple-reactive-form',
  templateUrl: './simple-reactive-form.component.html',
  styleUrls: ['./simple-reactive-form.component.css']
})
export class SimpleReactiveFormComponent implements OnInit {

  policyName = new FormControl('',[Validators.required]);

  constructor() { }

  ngOnInit() {
  }

}
```



```
Policy Name <input [formControl]="policyName" #policy/>
<div *ngIf="policyName.touched == true &&
            policyName.invalid == true"
     style="color: 🟥crimson;font-size: small;">
Policy Name is Required

</div>
```



Policy Name



Policy Name

Policy Name is Required

Git Resource

https://github.com/mkjitsolution/Reactive_Forms

## FormGroup

FormGroup is one of the three fundamental building blocks used to define the forms in Angular, along with FormControl and FormArray.

The FormGroup aggregates the values of each child FormControl into one object, with each control name as a key.

It calculates its status by reducing the status values of its children.

https://angular.io/api/forms/FormGroup#description

Git Resource

https://github.com/mkjitsolution/Reactive_Forms

## Self Learning

Form Builder & Custom Validation

The FormBuilder is the helper API to build forms in Angular.  It provides shortcuts to create the instance of the FormControl, FormGroup or FormArray. It reduces the code required to write the complex forms.

https://github.com/mkjitsolution/Reactive_Forms

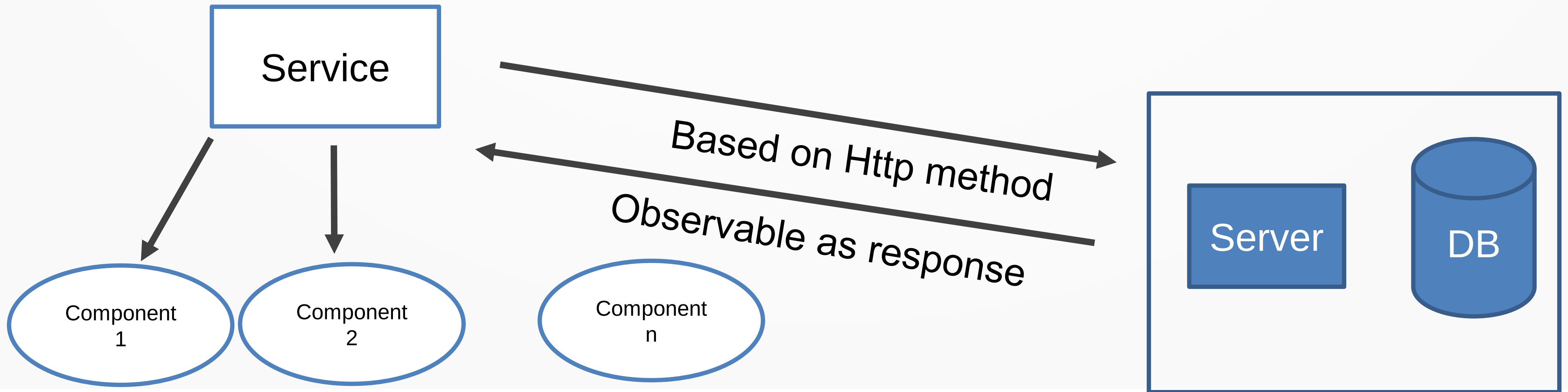
# Http Observable

Http Observable use to fetch data from Rest endpoints

The **HttpClient** in @angular/common/Http offers the simplified client HTTP API for Angular applications that rests on an **XMLHttpRequest** interface exposed by browsers.

The Observable isn't an Angular specific feature, but a new standard for managing async data that will be included in the ES7 release.

**1.Observables are lazy**
You could think of lazy observables as newsletters. For each subscriber a new newsletter is created. They are then only send to those people, and not to anyone else.
**2.Observables can have multiple values over time**
Now if you keep that subscription to the newsletter open, you will get a new one every once and a while. The sender decides when you get it but all you have to do is just wait until it comes straight into your inbox.



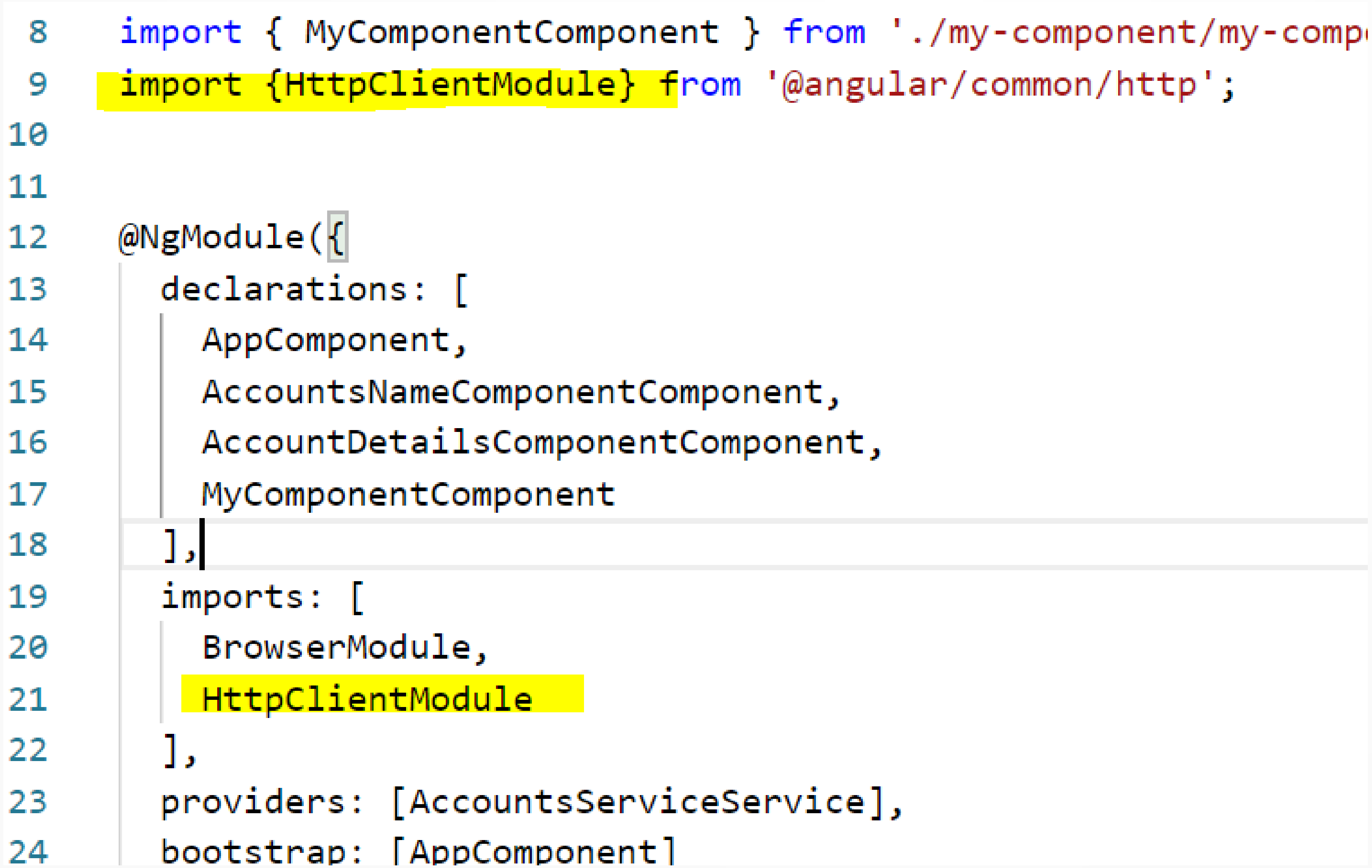
# Managing Subscription

We subscribe to the observable ourselves using the actual subscribe() method.

```
componentProperty: ModelClass;

ngOnInit() {
    //we've to  manually subscribe to this method and take the data
    // in our callback
    this. __studentService.callServiceMethod()
      .subscribe((feed)=>
      {
        this.componentProperty = feed ;
      })
      console.log(" ---- inside http observable component "+this.students.length);
  }
```

# Code-Step 1

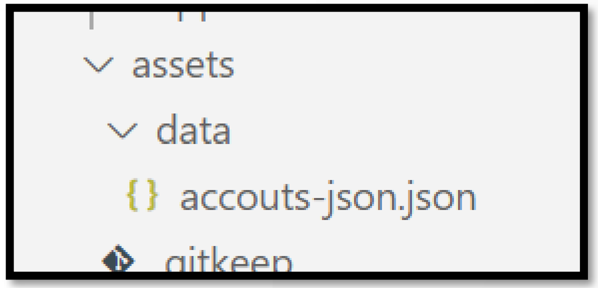
## Adding HttpClientModule in App module



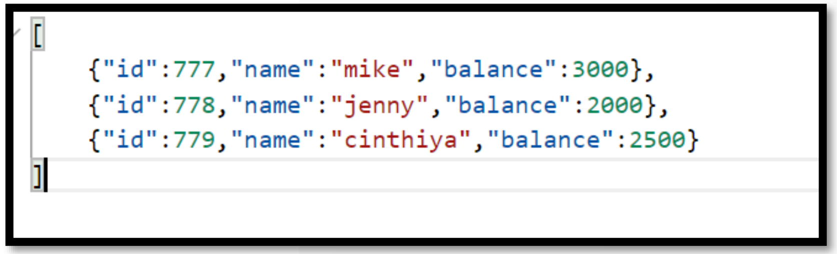
```
import { MyComponentComponent } from './my-component/my-comp
import {HttpClientModule} from '@angular/common/http';


@NgModule({
  declarations: [
    AppComponent,
    AccountsNameComponentComponent,
    AccountDetailsComponentComponent,
    MyComponentComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [AccountsServiceService],
  bootstrap: [AppComponent]
```

# Step 2 – Updating Service

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class AccountsServiceService {

  private endpoint = "/assets/data/accounts-json.json";

  constructor(private http:HttpClient) { }

  getAllAccountsFromServer():Observable<AccountsServiceService[]>
  { // this.http.get(endpoint);
    return this.http.get<AccountsServiceService[]>(this.endpoint);

  }
```

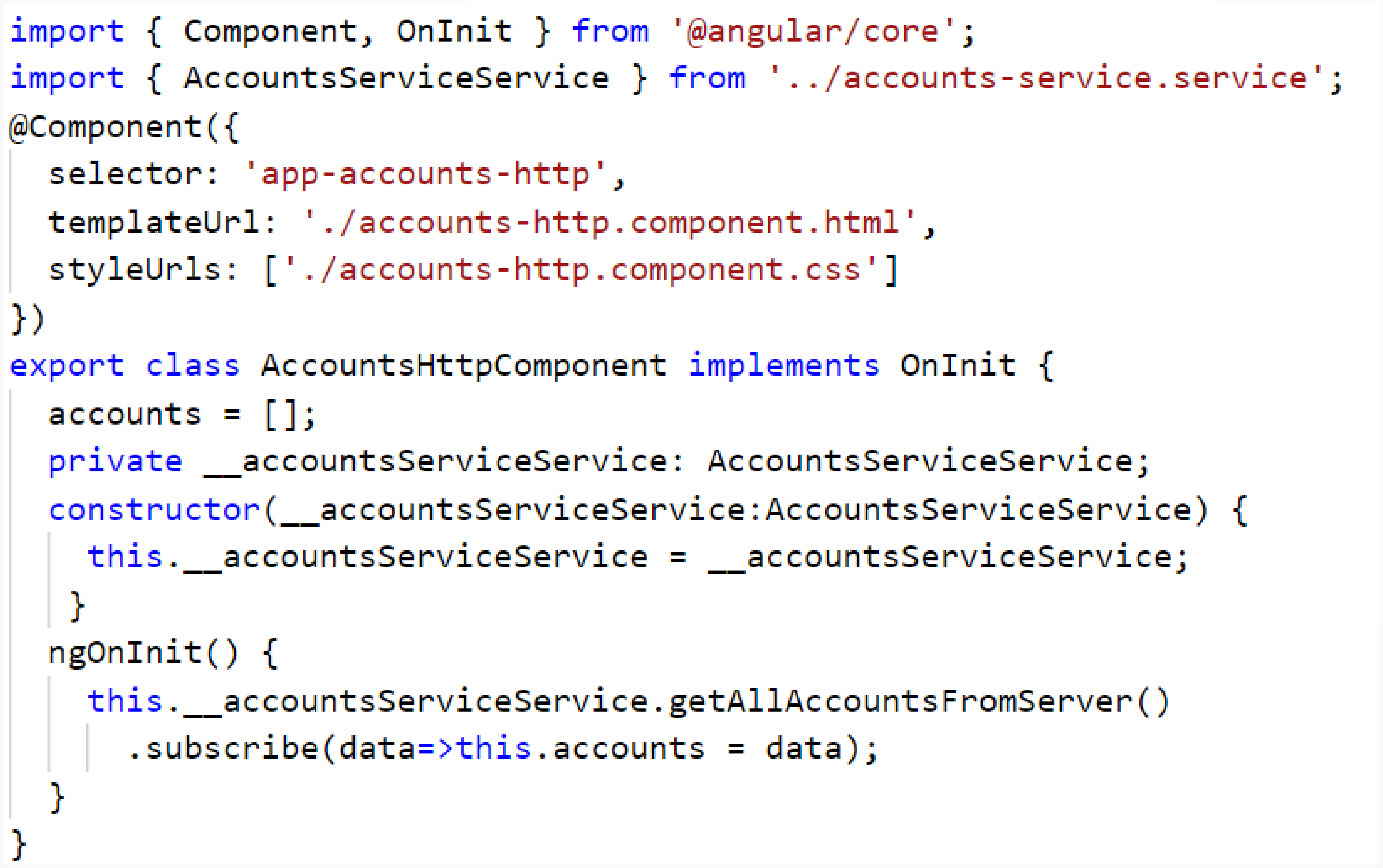


```
∨ assets
  ∨ data
    {} accouts-json.json
  gitkeep
```



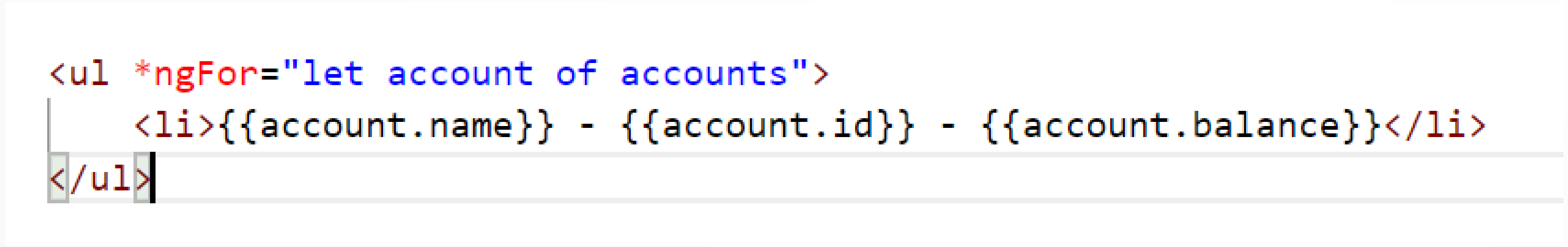
```
[
    {"id":777,"name":"mike","balance":3000},
    {"id":778,"name":"jenny","balance":2000},
    {"id":779,"name":"cinthiya","balance":2500}
]
```

# Step 3 : Updating Component & HTML template

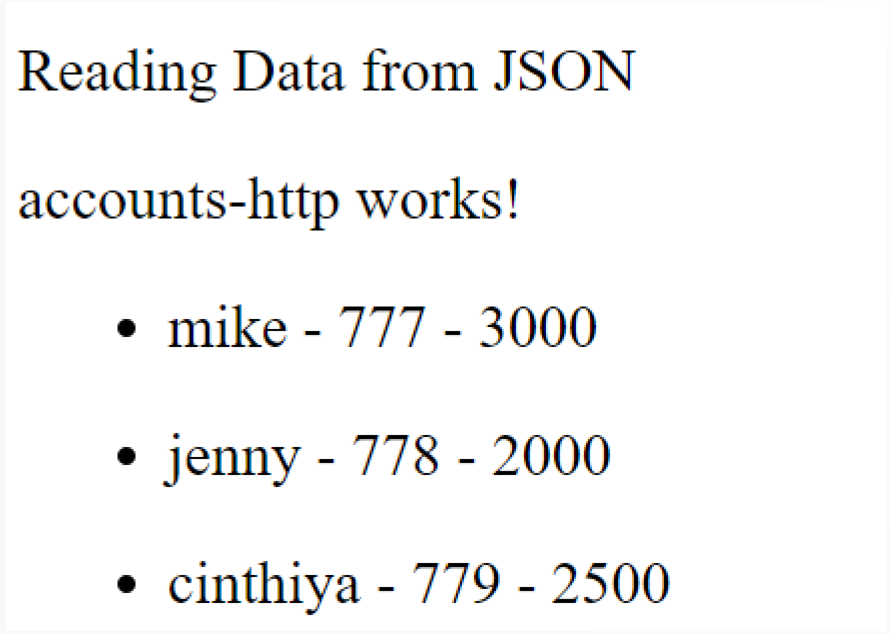


```
import { Component, OnInit } from '@angular/core';
import { AccountsServiceService } from '../accounts-service.service';
@Component({
  selector: 'app-accounts-http',
  templateUrl: './accounts-http.component.html',
  styleUrls: ['./accounts-http.component.css']
})
export class AccountsHttpComponent implements OnInit {
  accounts = [];
  private __accountsServiceService: AccountsServiceService;
  constructor(__accountsServiceService:AccountsServiceService) {
    this.__accountsServiceService = __accountsServiceService;
   }
  ngOnInit() {
    this.__accountsServiceService.getAllAccountsFromServer()
      .subscribe(data=>this.accounts = data);
  }
}
```

## Http Template



```
<ul *ngFor="let account of accounts">
    <li>{{account.name}} - {{account.id}} - {{account.balance}}</li>
</ul>
```



```
Reading Data from JSON

accounts-http works!

    • mike - 777 - 3000

    • jenny - 778 - 2000

    • cinthiya - 779 - 2500
```

# Assignment

Implement error handling messaging while any error occurs during server call.
Such as , Server 404, 500 errors , bad json error etc.

So in case we change file name

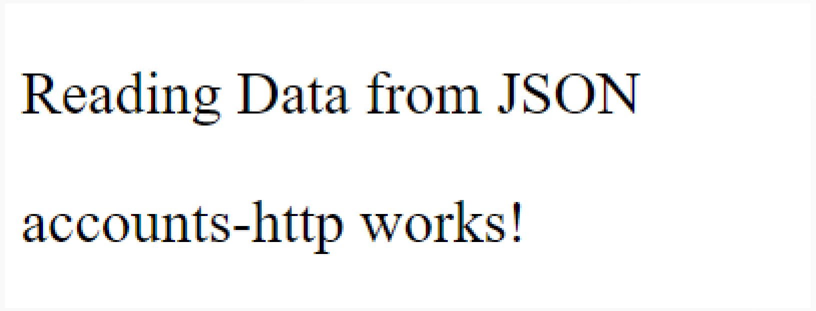
from

```
private endpoint = "/assets/data/accounts-json.json";
```

to

```
private endpoint = "/assets/data/accounts-json1.json";
```

We are not getting any output , instead we should get proper error message like “bad file name”



```
Reading Data from JSON

accounts-http works!
```

*Hint : Read rxjs/add/operators/catchError & throwError*

# Solution of the problem

Git Resource

https://github.com/mkjitsolution/HttpObservable

https://blog.angular-university.io/rxjs-error-handling/

https://www.concretepage.com/angular/angular-catcherror

## Routing & Navigation

https://angular.io/guide/router https://angular.io/api/router/Route

The Angular router is an essential element of the Angular platform.

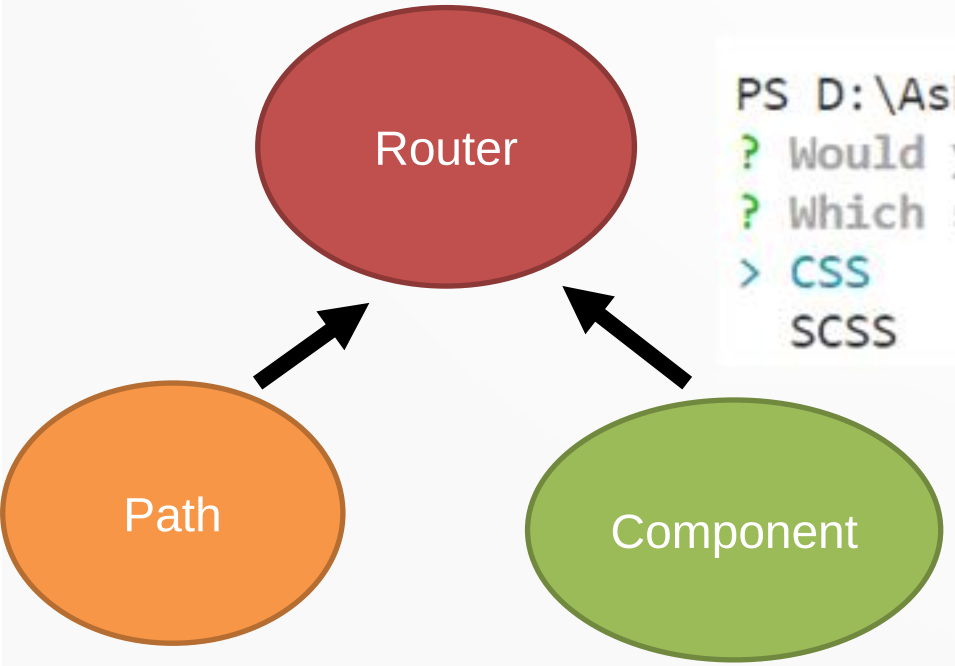
The Angular Router enables navigation from one view to the next as users perform application tasks.

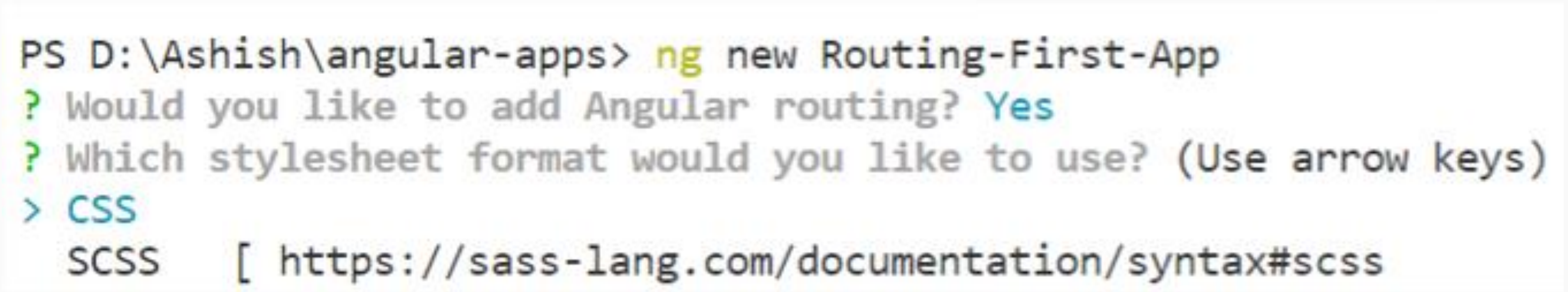
So ,
In Angular we have components, each component can have a different task,
for example in Accounts Application there can be a Profile, Balance, Policies , Investment , Insurance views.

- Routing enables a user to visit these pages or components with each one having a specified URL path.
- These URL's or RouterLinks can be accessed by a user through hyperlinks in HTML templates, javascript's navigate methods or by simply pasting in browser's address bar.
- It's contained in the @angular/router package.
- Through routing we can use the browser's URL to navigate between Angular components in the same way you can use the usual server side navigation.

# Configuration of Router

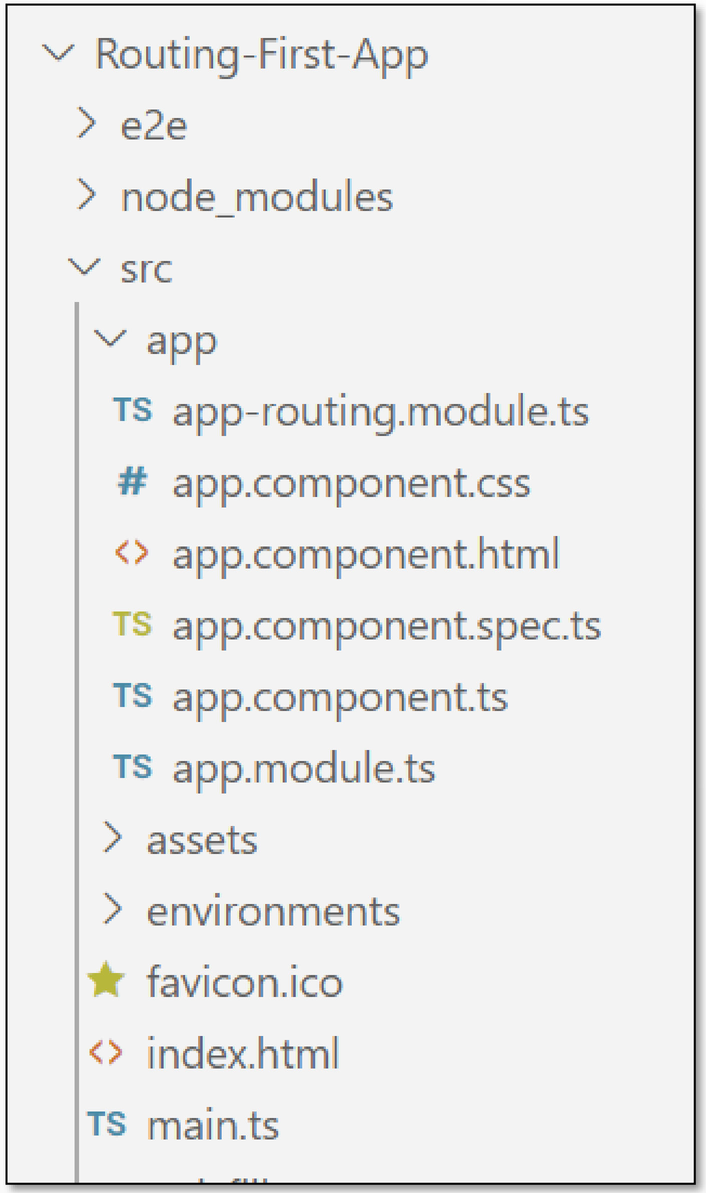
- A routed Angular application has one singleton instance of the *Router* service.
- When the browser's URL changes, that router looks for a corresponding Route from which it can determine the component to display
- A router has no routes until you configure it.





```
PS D:\Ashish\angular-apps> ng new Routing-First-App
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss
```

# What’s new by choosing Routing



Routing-First-App
- e2e
- node_modules
- src
  - app
    - app-routing.module.ts
    - app.component.css
    - app.component.html
    - app.component.spec.ts
    - app.component.ts
    - app.module.ts
  - assets
  - environments
  - favicon.ico
  - index.html
  - main.ts

By opting Routing option we will get

1) app-routing.module.ts
2) Updated app-module.ts with AppRoutingModule
3) And updated index.html with <base href="/">
4) Updated app.component.html with <router-outlet/>

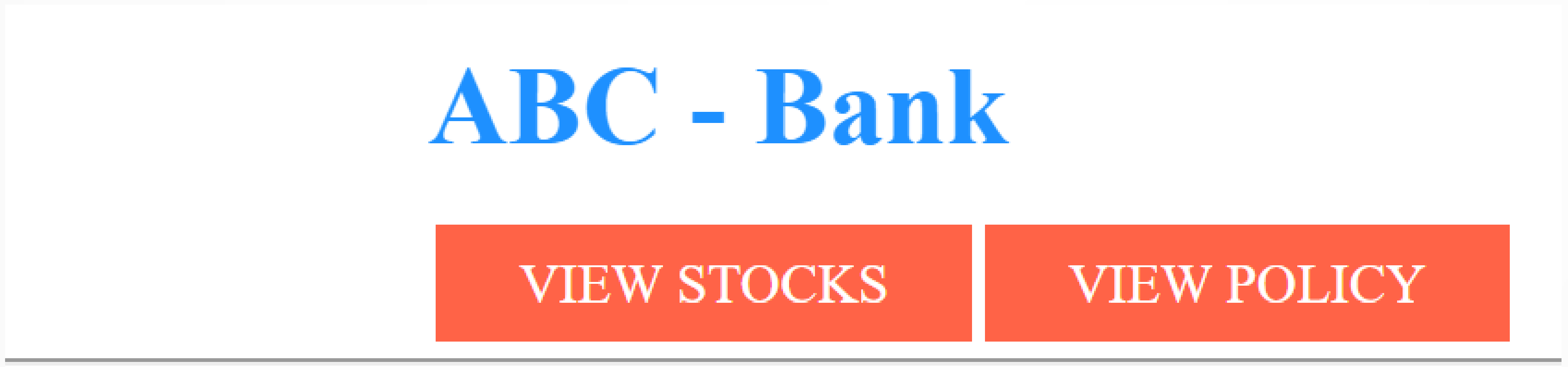
# Understanding

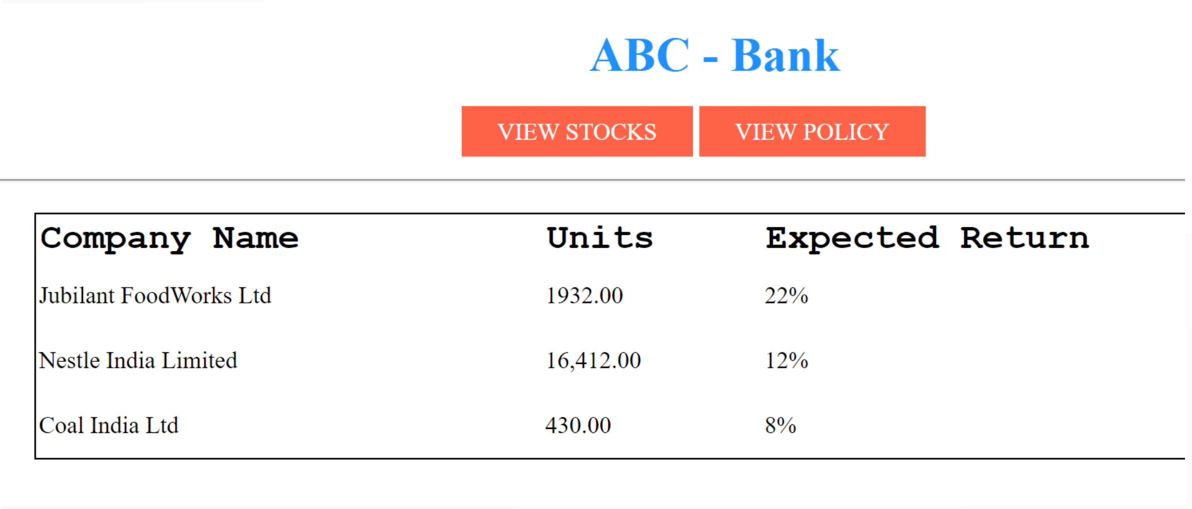
| | |
|---|---|
| Router-Outlet | The &lt;router-outlet&gt; is a directive that's available from the router library where the Router inserts the component that gets matched based on the current browser's URL.<br>Available at the end of app.component.html<br><br>https://angular.io/api/router/RouterOutlet |
| routerLink attribute | The routerLink attribute can be used on any element. It makes that element clickable in order to activate the specified route. It can be used in a similar way to the href attribute on links.<br><br>https://angular.io/api/router/RouterLink#description<br><br>For example<br><br>&lt;div routerLink="/component1"&gt; Click Here &lt;/div&gt;<br><br>&lt;a routerLink="/component 2 "&gt; View This &lt;/a&gt; |
| routerLinkActive | In order to style router links to the currently active route, the routerLinkActive attribute is provided. It accepts one or more class names, that will be toggled on the element when its routerLink points to the active route. |

## Development Process

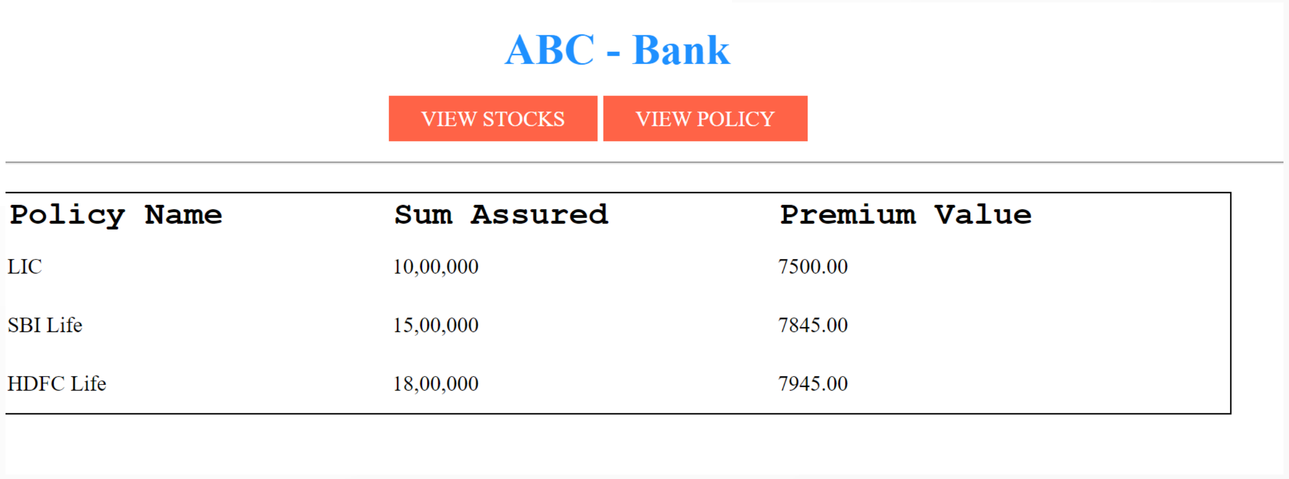
1. While Creating the project choose Routing.
2. Configure Routes/Links with Components
3. Provide navigation controls

# Application


ABC - Bank


ABC - Bank

| Company Name | Units | Expected Return |
|---|---|---|
| Jubilant FoodWorks Ltd | 1932.00 | 22% |
| Nestle India Limited | 16,412.00 | 12% |
| Coal India Ltd | 430.00 | 8% |

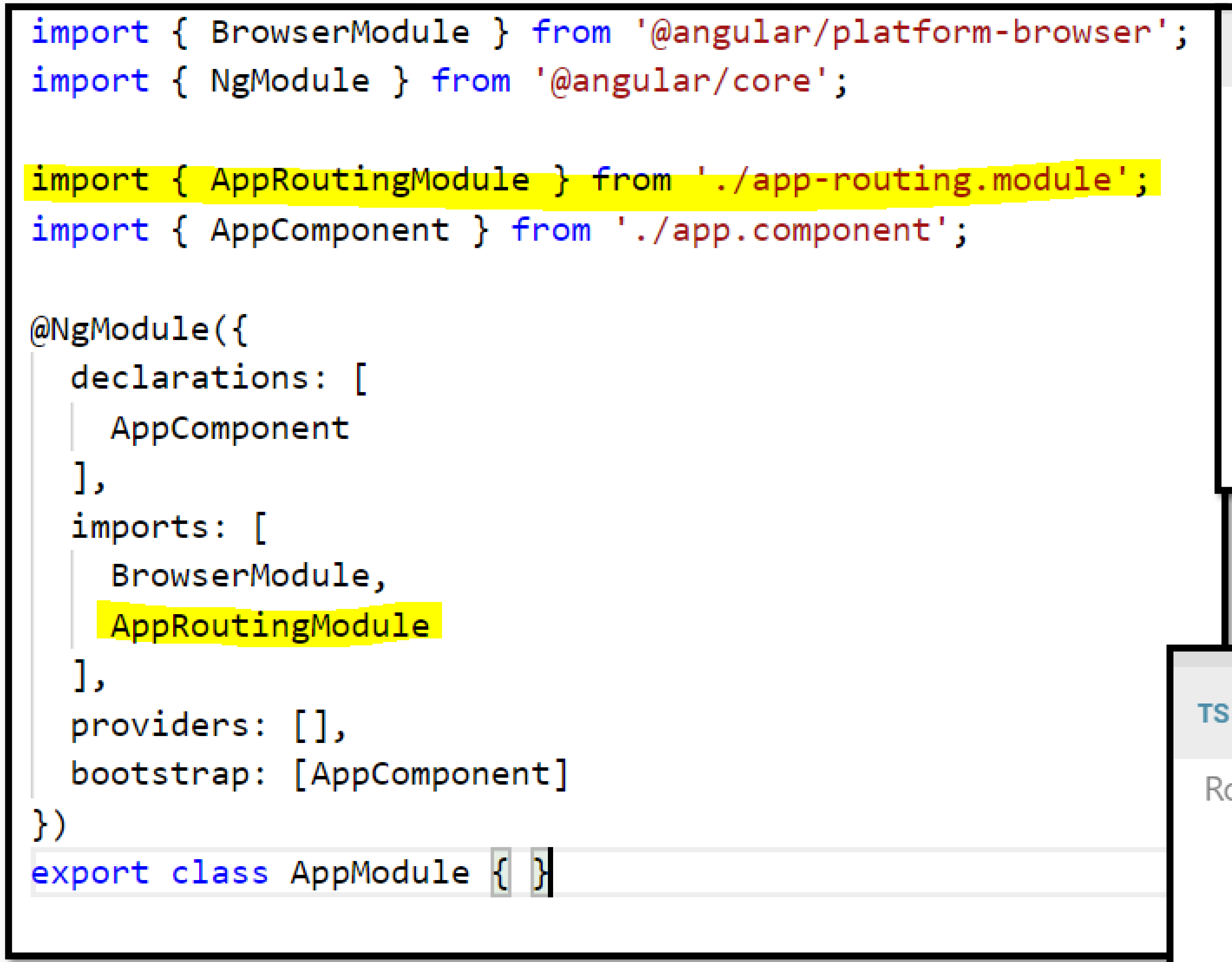

ABC - Bank

| Policy Name | Sum Assured | Premium Value |
|---|---|---|
| LIC | 10,00,000 | 7500.00 |
| SBI Life | 15,00,000 | 7845.00 |
| HDFC Life | 18,00,000 | 7945.00 |

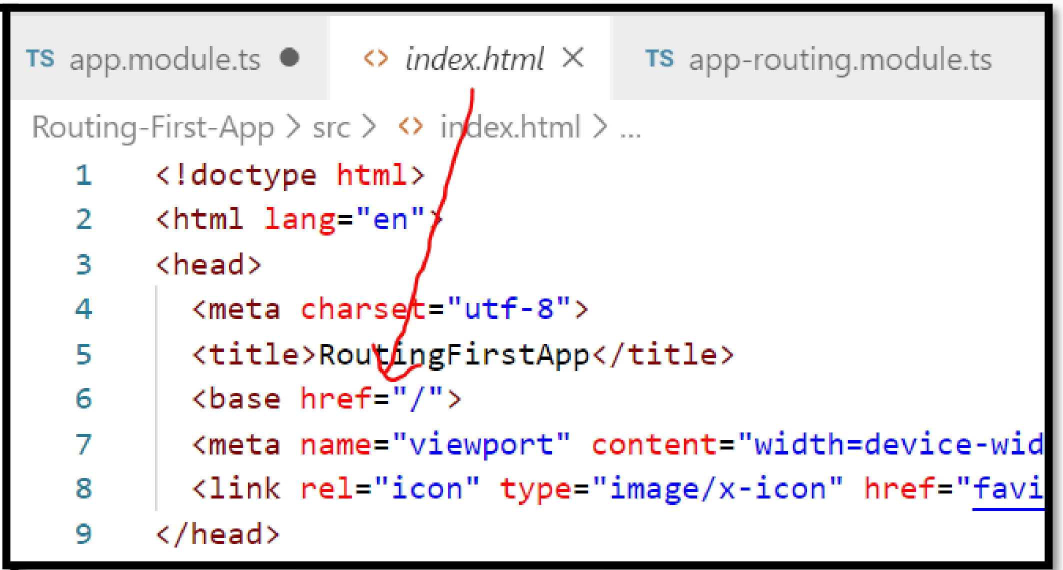
*For css of links*

*[https://css-tricks.com/](https://css-tricks.com/)*

# Step 1 : Basic Requirements



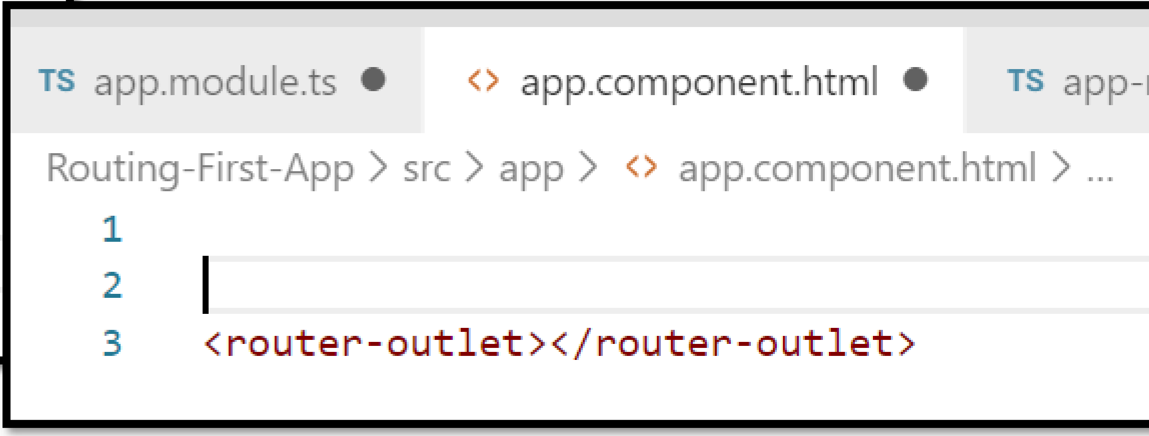
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



TS app.module.ts ● | <> index.html × | TS app-routing.module.ts

Routing-First-App > src > <> index.html > ...

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>RoutingFirstApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-wid
  <link rel="icon" type="image/x-icon" href="favi
</head>
```

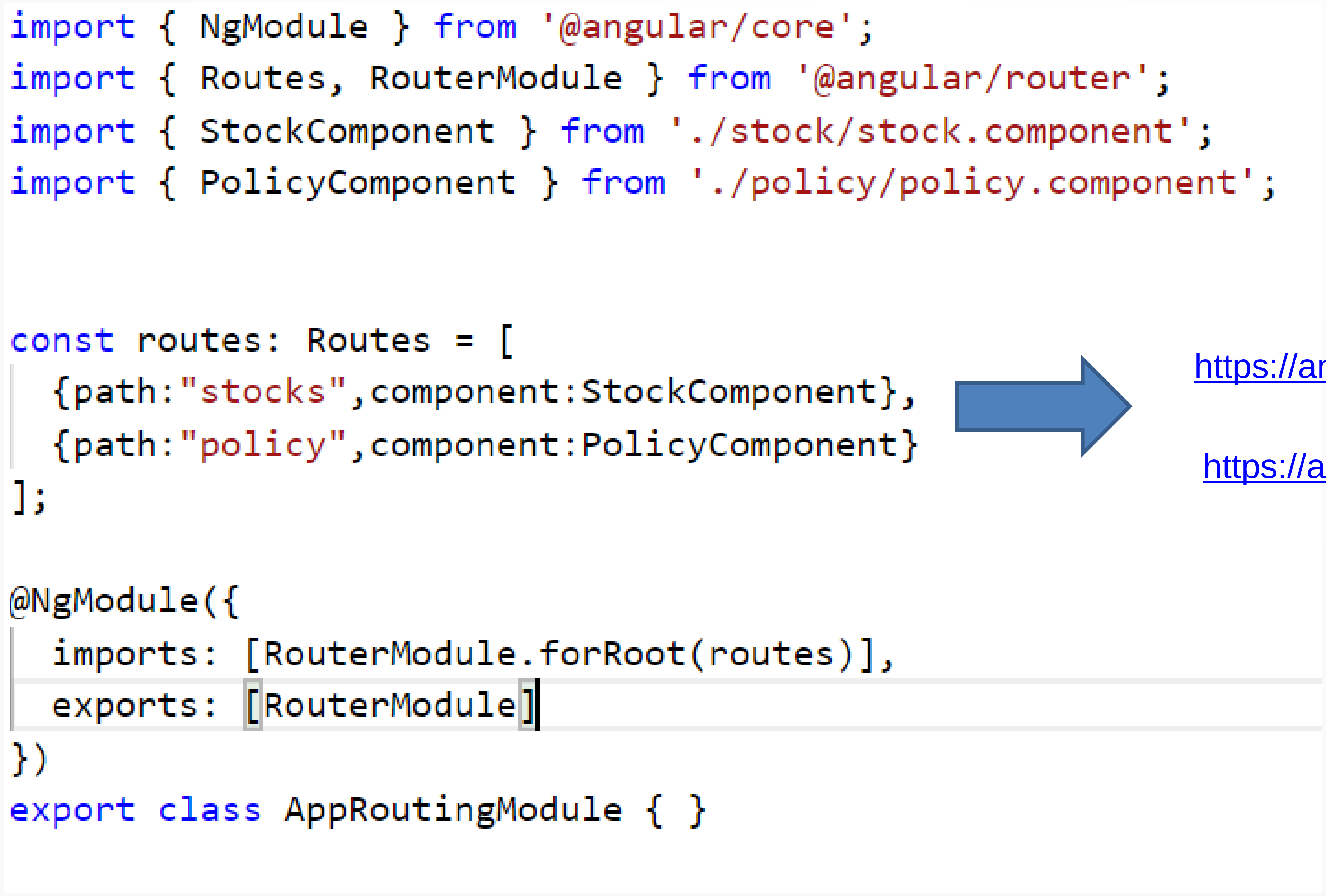


TS app.module.ts ● | <> app.component.html ● | TS app-

Routing-First-App > src > app > <> app.component.html > ...

```


<router-outlet></router-outlet>
```

## Step 2 : Configure Routes

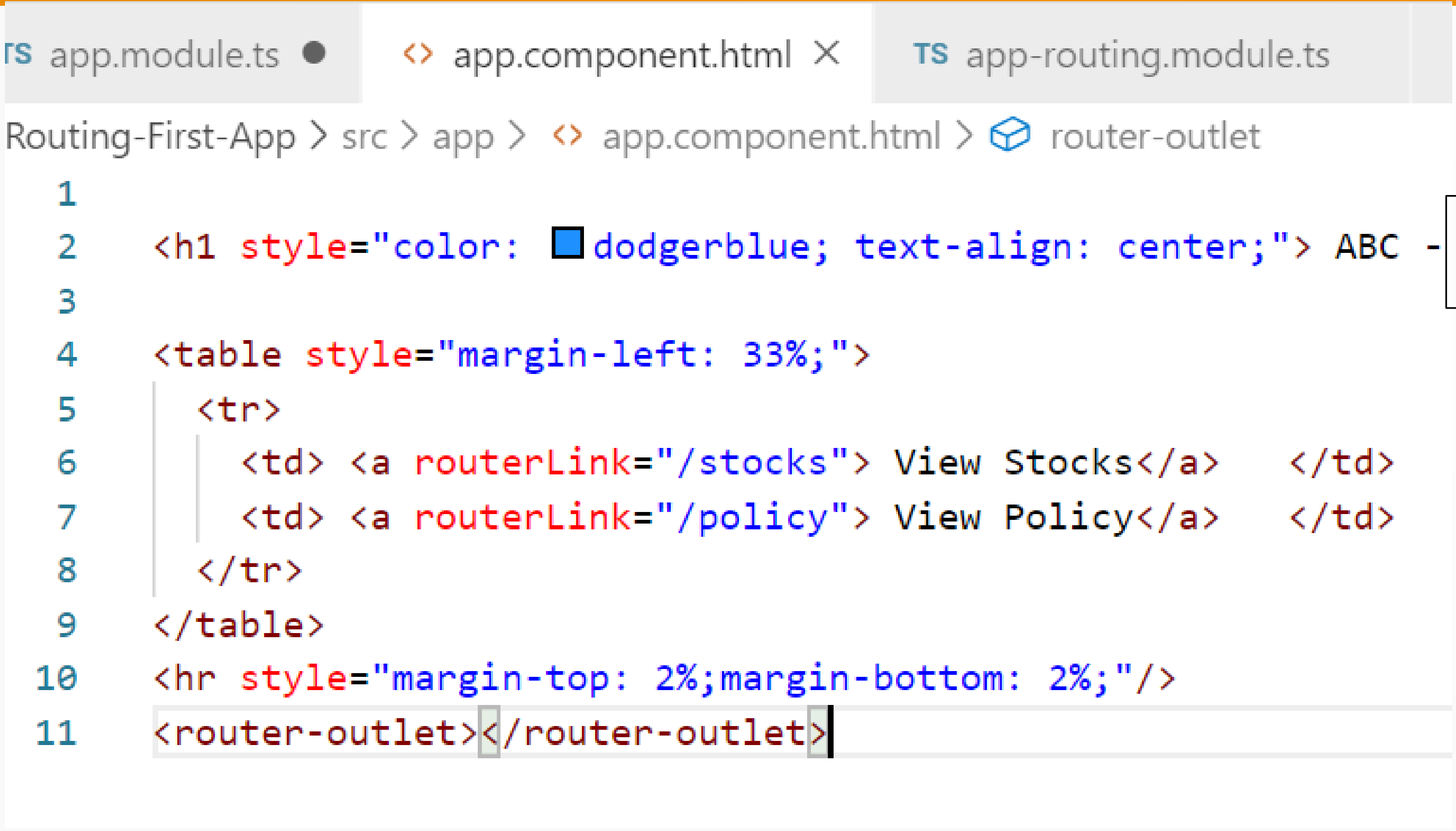


```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { StockComponent } from './stock/stock.component';
import { PolicyComponent } from './policy/policy.component';


const routes: Routes = [
  {path:"stocks",component:StockComponent},
  {path:"policy",component:PolicyComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

https://angular.io/api/router/Routes

https://angular.io/api/router/Route

# Step 3 : Navigation Controls



app.module.ts
app.component.html
app-routing.module.ts
Routing-First-App > src > app > app.component.html > router-outlet

```
<h1 style="color: dodgerblue; text-align: center;"> ABC -

<table style="margin-left: 33%;">
  <tr>
    <td> <a routerLink="/stocks"> View Stocks</a>    </td>
    <td> <a routerLink="/policy"> View Policy</a>    </td>
  </tr>
</table>
<hr style="margin-top: 2%;margin-bottom: 2%;"/>
<router-outlet></router-outlet>
```

Git Resource

https://github.com/mkjitsolution/Angular_Routing

Assignment : add component in case of wrong endpoint

# Route With Parameter

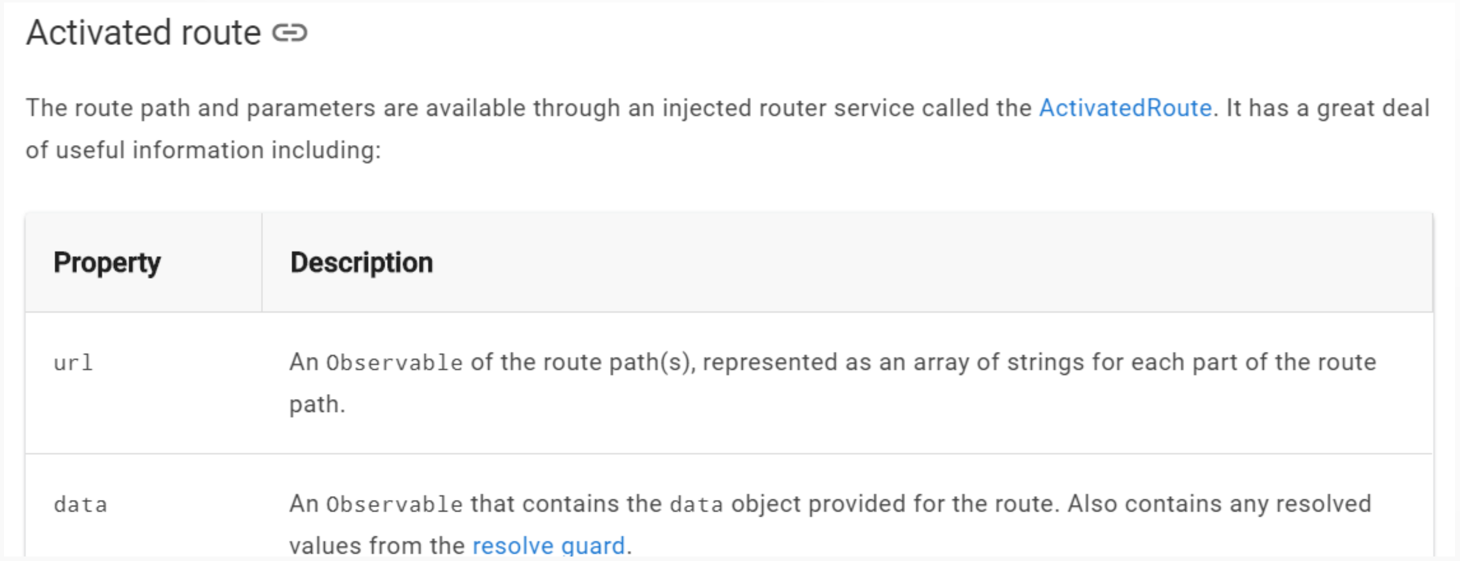
Business applications usually contains parameters endpoints.
For exp:

www.abc-bank/accounts/101

www.amazon.com/watch/man/analogwatch/101

In Order to read parameters we should relay on an interface ActivatedRoue

https://angular.io/api/router/ActivatedRoute   &   https://angular.io/guide/router#activated-route



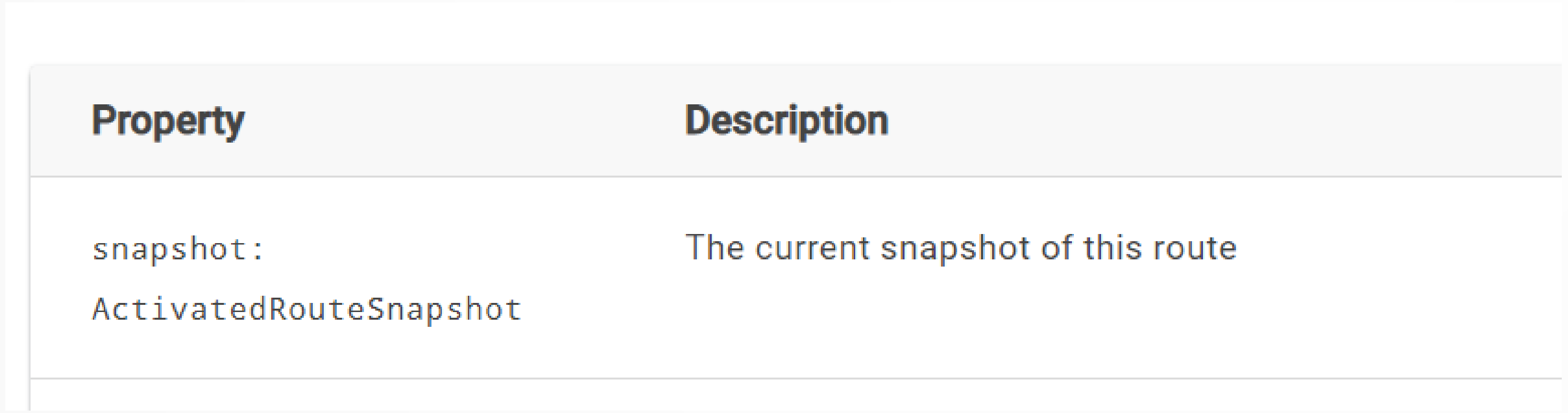
## Activated route

The route path and parameters are available through an injected router service called the ActivatedRoute. It has a great deal of useful information including:

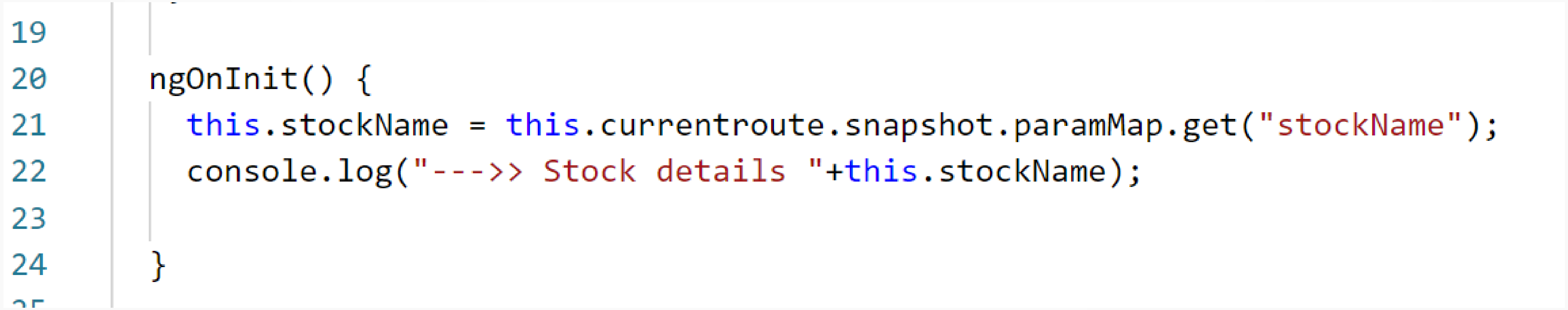
| Property | Description |
|---|---|
| `url` | An `Observable` of the route path(s), represented as an array of strings for each part of the route path. |
| `data` | An `Observable` that contains the `data` object provided for the route. Also contains any resolved values from the resolve guard. |

# Reading Parameter Values

https://angular.io/api/router/ActivatedRoute#snapshot



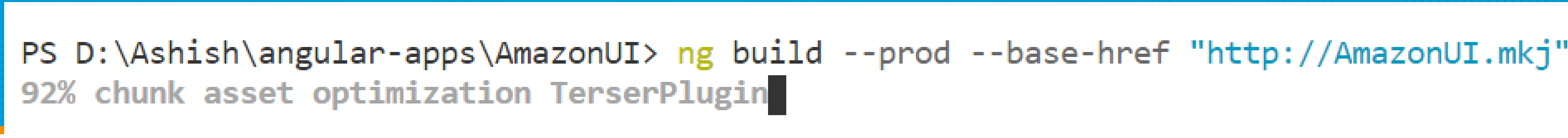
| Property | Description |
|---|---|
| `snapshot: ActivatedRouteSnapshot` | The current snapshot of this route |



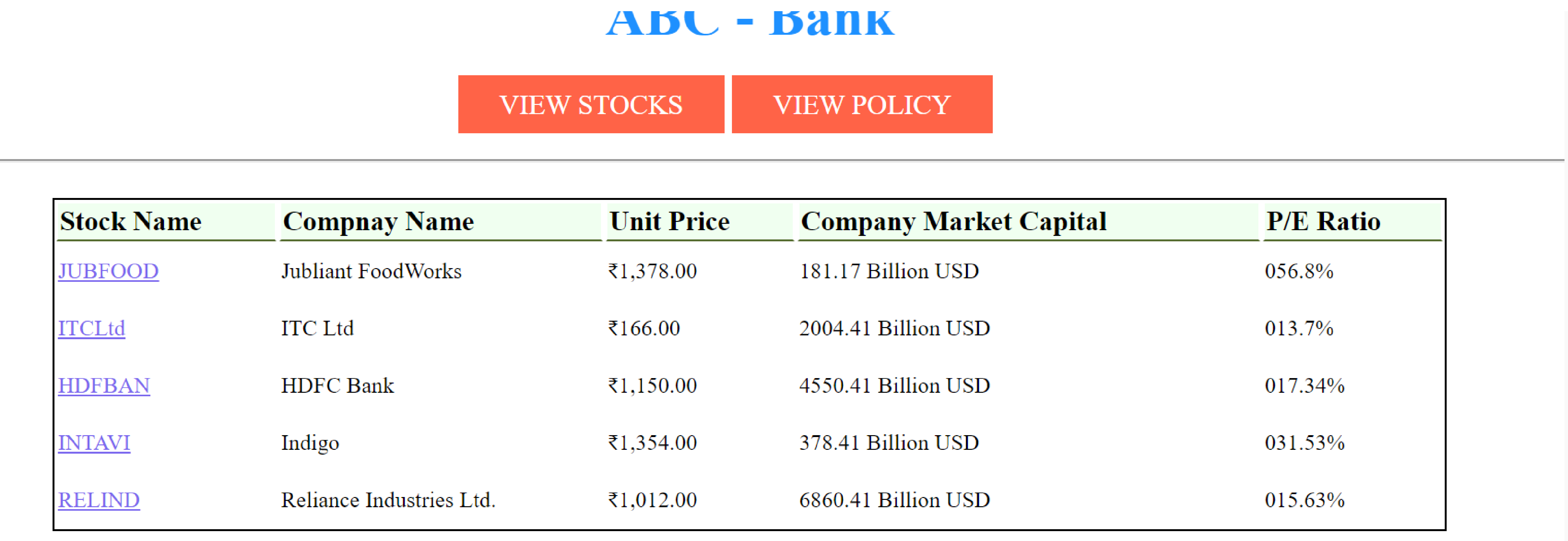
```
ngOnInit() {
  this.stockName = this.currentroute.snapshot.paramMap.get("stockName");
  console.log("--->> Stock details "+this.stockName);

}
```

*Note : The currentRoute is the instance of ActivatedRoute*

# Application



```
PS D:\Ashish\angular-apps\AmazonUI> ng build --prod --base-href "http://AmazonUI.mkj"
92% chunk asset optimization TerserPlugin
```



ABC - Bank

| Stock Name | Compnay Name | Unit Price | Company Market Capital | P/E Ratio |
|---|---|---|---|---|
| JUBFOOD | Jubliant FoodWorks | ₹1,378.00 | 181.17 Billion USD | 056.8% |
| ITCLtd | ITC Ltd | ₹166.00 | 2004.41 Billion USD | 013.7% |
| HDFBAN | HDFC Bank | ₹1,150.00 | 4550.41 Billion USD | 017.34% |
| INTAVI | Indigo | ₹1,354.00 | 378.41 Billion USD | 031.53% |
| RELIND | Reliance Industries Ltd. | ₹1,012.00 | 6860.41 Billion USD | 015.63% |

Git Resource https://github.com/mkjitsolution/Angular_Routing

## *Thanks for Choosing us*

Corporate Instructor's Panel

Ashish Bansal : Global Java Full Stack and Salesforce Instructor
Clients such as GOSI – Saudi Arabia , Noor Bank , UAE , Citi Bank-USA , Oracle University

Kirti Sharma : Global Machine Learning and Data Science Instructor
Clients such as HSBC Bank- London, Voya Financials- USA

Prabhjeet Saini : Global CCNP Trainers .
Clients such as CNN-Canada, ONGC- India , Vodafone- UK

Jitin Guglani , Jai Sapra , Chinthiya Somes , Robert