Software Engineering for WWW

Introduction to Angular and TypeScript

Dr. Vinod Dubey
SWE 642
George Mason University

Acknowledgement/References

https://angular.io/start
https://malcoded.com/posts/
https://javabrains.io/
https://www.tutorialspoint.com/angular6/
https://www.tutorialspoint.com/angular6/angular6_quick_guide.htm
https://www.tutorialspoint.com/angular6/angular6_overview.htm

Agenda

- Angular/Angular Project/Angular component
- Data binding
- Directives
- Styles for components
- Creating/Using Modules
- Services
- Dependency Injection
- HttpClient
- Making RESTful calls
- Routes
- Forms
- TypeScripts

What is angular?

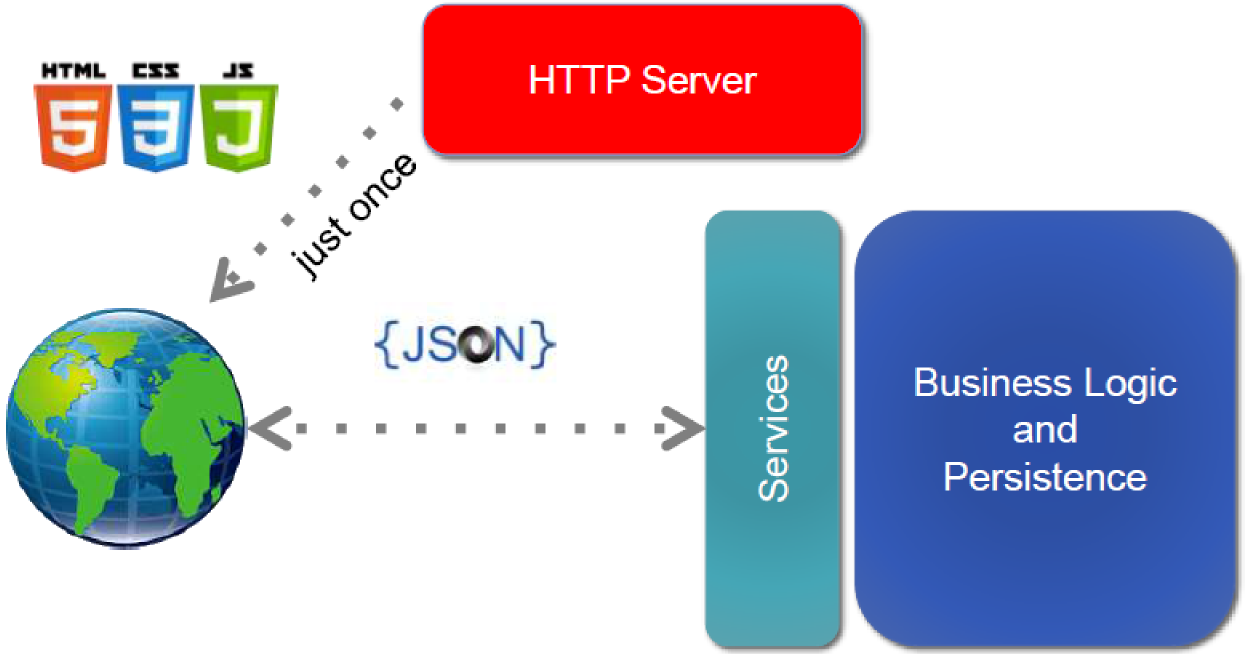
- **An opensource JavaScript-based Web application development framework developed by Google**
  - Angular is a complete rewrite of AngularJS .
  - Angular uses a hierarchy of components as its primary architectural characteristic
  - Core functionality via modules
  - Uses Microsoft's TypeScript language
    - Static Typing, including Generics
    - Annotations
    - TypeScript is a superset of JavaScript
  - Dynamic loading of backend data into view components
  - Asynchronous template compilation
  - Integration with RESTful Web Services using HttpClient

What is angular?

- **Angular uses the concept of Single Page Application (SPA)**
  - A fully contained applications in the browser
    - that do not need to make requests for new pages on the server.
  - Usually makes request just of the data that will be rendered inside of the pages
    - Accesses backend via REST+JSON services
  - SPA Advantage: Faster application,
    - Eliminates the download of html, js, and css code in each request

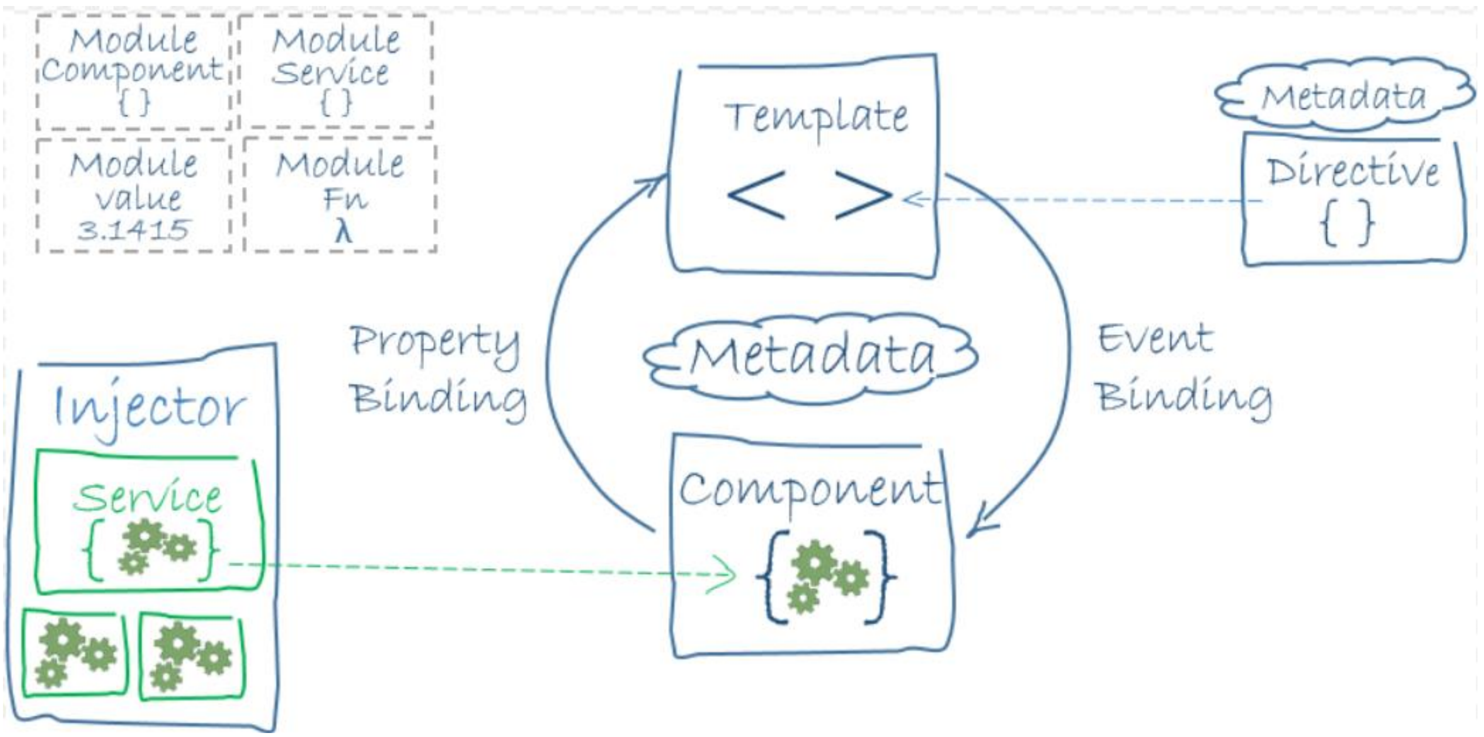
What is angular?

- **Service Oriented Front-End Architecture (SOFEA) –** can be considered as a **synonym** of **Single Page Application**



Architecture of an Angular application

- **The main building blocks: modules, components, templates, metadata, data binding, directives, services, and dependency injection**

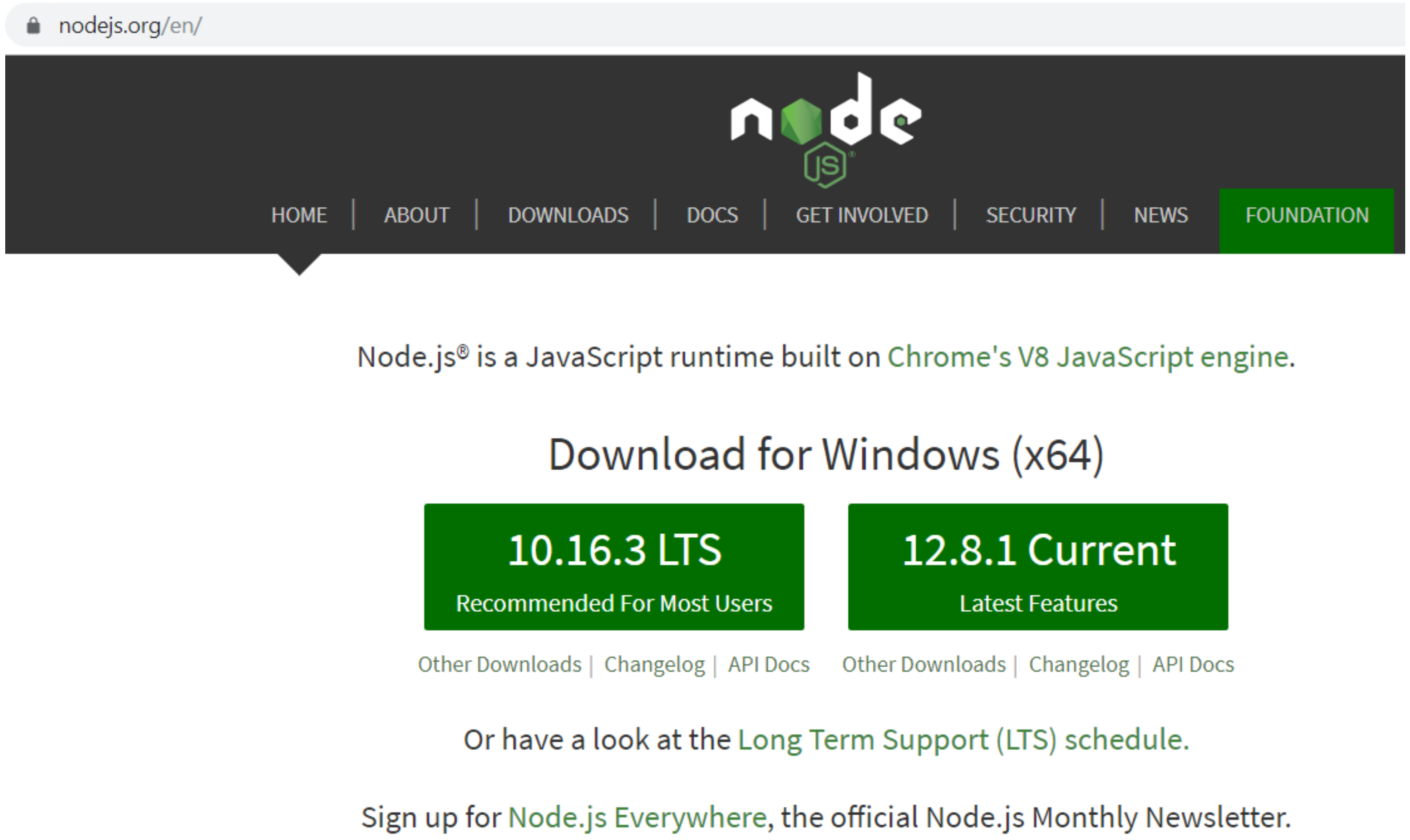


Angular Installation

- **To use Angular, need to install three things**
  - Node.js – Runtime environment for executing JavaScript code outside the browser.
    - Provides tools needed to build angular projects
  - Integrated Development Environment (IDE) – Needed to edit your source code
    - Visual Studio Code from ***code.visualstudio.com – has great support for TypeScript***
    - Or, any other IDE, such as Atom, Sublime Text
  - Angular Command Line Interface (CLI)
    - Command line tool to create new angular projects or generate some boiler plate code

Install Node.js from nodejs.org

- **Download the installer from nodejs.org, the one associated to left green box below – "Recommended for Most Users"**


nodejs.org/en/

HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | NEWS | FOUNDATION

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

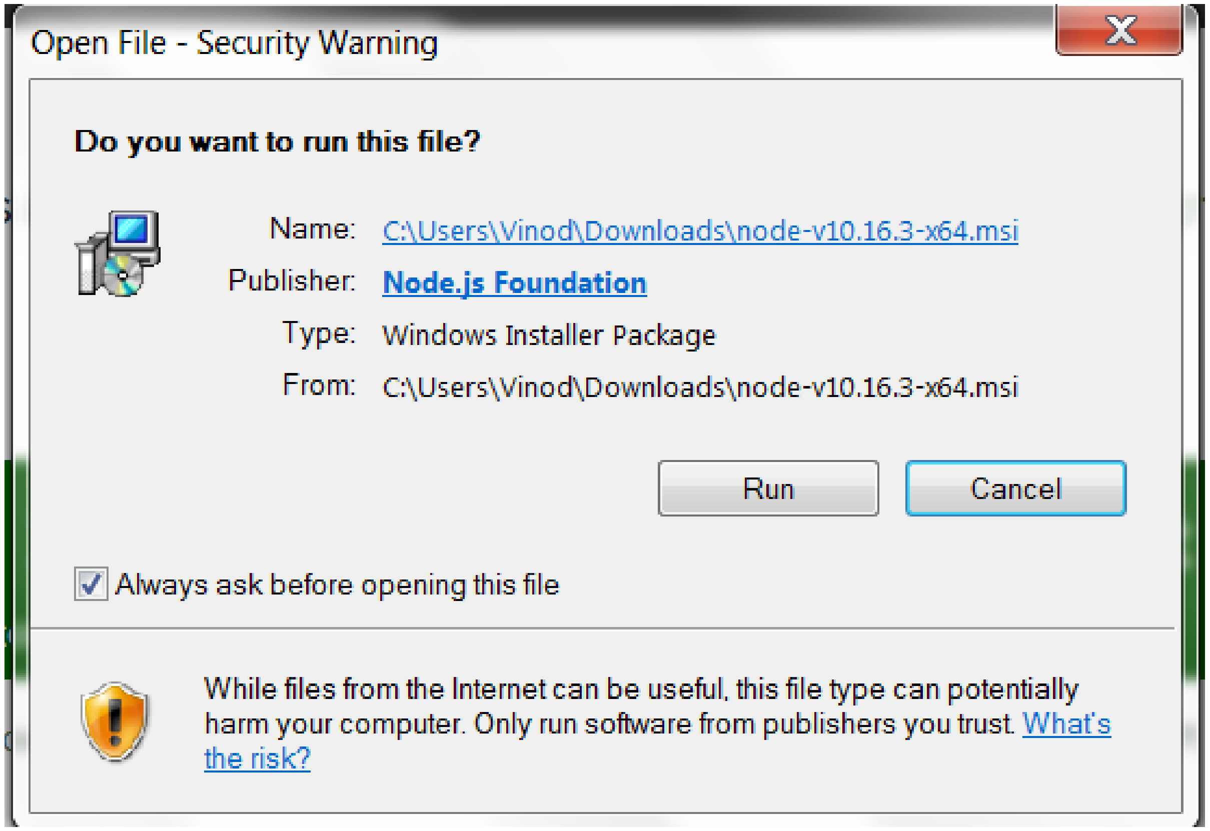
10.16.3 LTS
Recommended For Most Users
12.8.1 Current
Latest Features
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs

Or have a look at the Long Term Support (LTS) schedule.

Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.

Install Node.js from nodejs.org

- **Double click on the downloaded installer, and press Run and follow the default steps all the way to *Finished* to install node**



Open File - Security Warning

Do you want to run this file?

Name: C:\Users\Vinod\Downloads\node-v10.16.3-x64.msi
Publisher: **Node.js Foundation**
Type: Windows Installer Package
From: C:\Users\Vinod\Downloads\node-v10.16.3-x64.msi

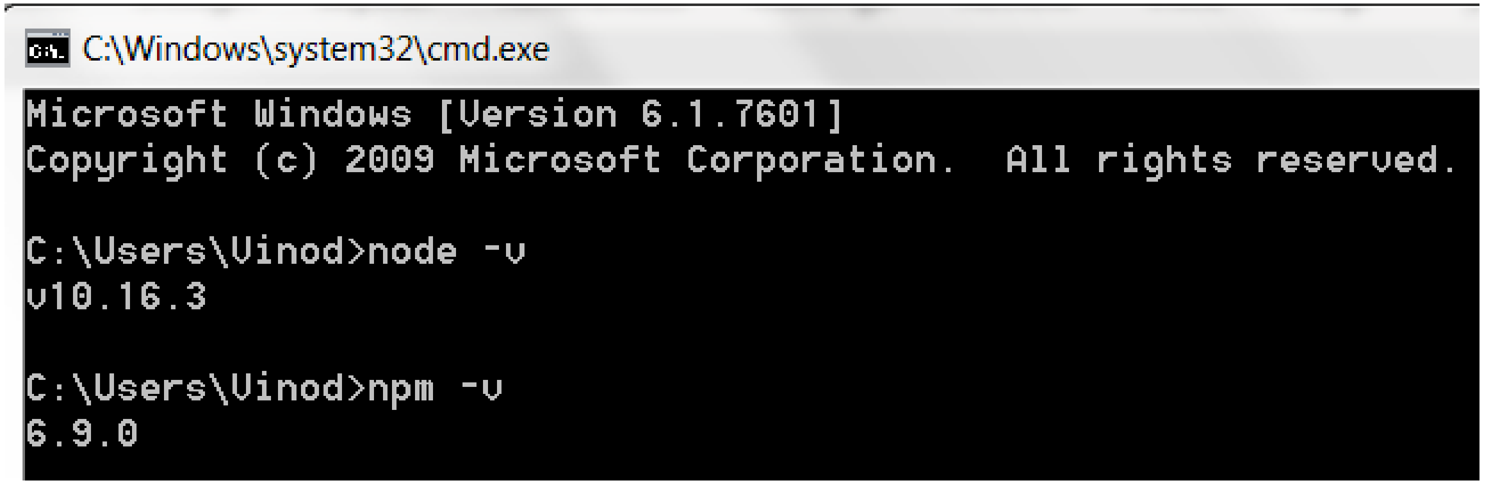
[✓] Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Install Node.js from nodejs.org

- **To verify that Node is installed, run $*node –v* in the command prompt**
  - Tells the version of the node installed

- **You can also run $*npm –v* to see the version of the node package manager installed**

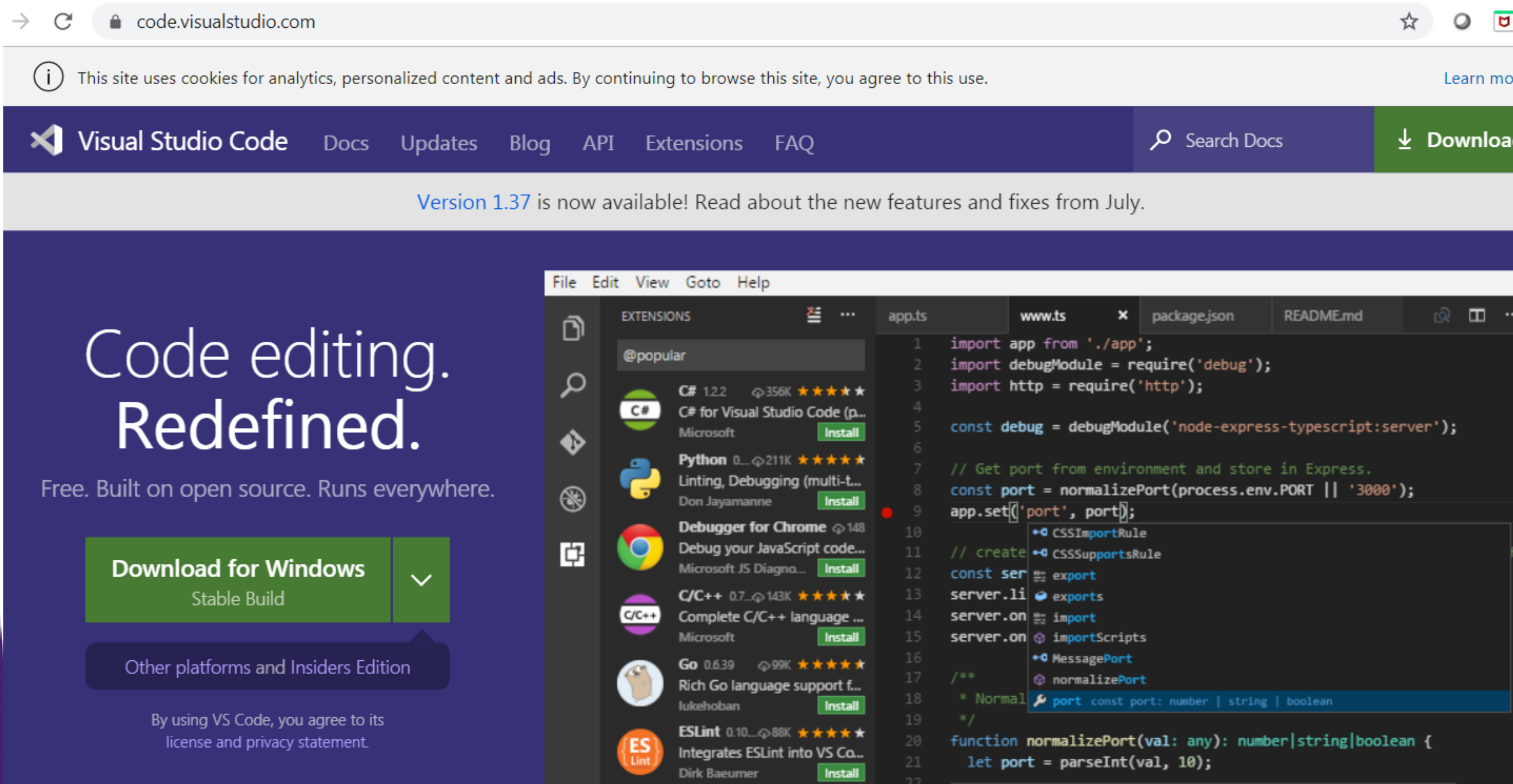


C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Vinod>node -v
v10.16.3

C:\Users\Vinod>npm -v
6.9.0
```

Install IDE Visual Studio Code

- **Install Visual Studio Code IDE from** *code.visualstudio.com*



This site uses cookies for analytics, personalized content and ads. By continuing to browse this site, you agree to this use.

Version 1.37 is now available! Read about the new features and fixes from July.

Code editing.
Redefined.

Free. Built on open source. Runs everywhere.

Download for Windows
Stable Build

Other platforms and Insiders Edition

By using VS Code, you agree to its
license and privacy statement.

C# 1.2.2 356K ★★★★★
C# for Visual Studio Code (p...
Microsoft

Python 0... 211K ★★★★★
Linting, Debugging (multi-t...
Don Jayamanne

Debugger for Chrome 148
Debug your JavaScript code...
Microsoft JS Diagno...

C/C++ 0.7... 143K ★★★★★
Complete C/C++ language ...
Microsoft

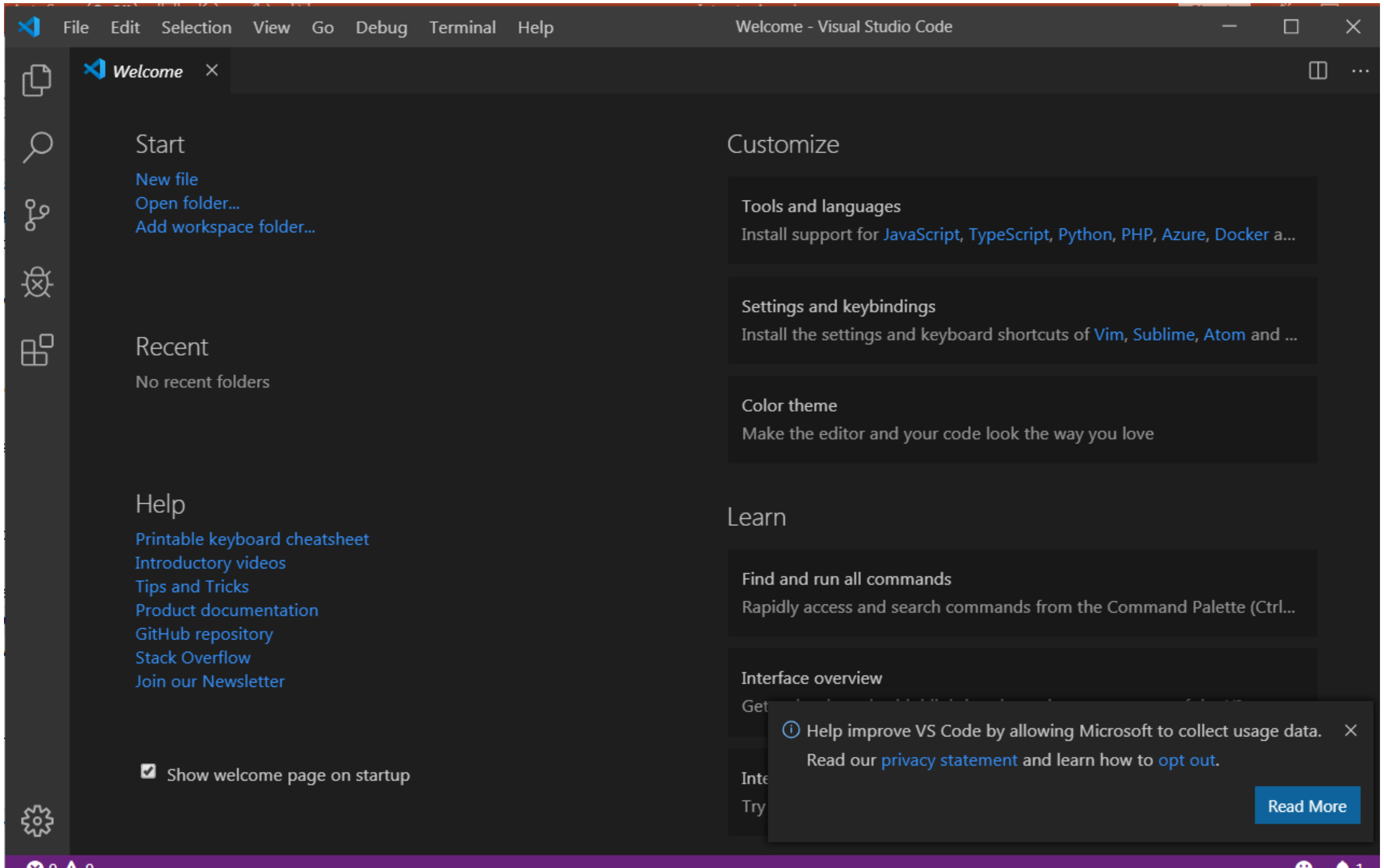
Go 0.6.39 99K ★★★★★
Rich Go language support f...
lukehoban

ESLint 0.10... 88K ★★★★★
Integrates ESLint into VS Co...
Dirk Baeumer

```
import app from './app';
import debugModule = require('debug');
import http = require('http');

const debug = debugModule('node-express-typescript:server');

// Get port from environment and store in Express.
const port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

// create
const ser
server.li
server.on
server.on

/**
 * Normal
 */
function normalizePort(val: any): number|string|boolean {
  let port = parseInt(val, 10);
```

Install IDE Visual Studio Code

- **Once installed it may show a welcome page that you can close**


Welcome - Visual Studio Code

Start

New file
Open folder...
Add workspace folder...

Recent

No recent folders

Help

Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Join our Newsletter

Show welcome page on startup

Customize

Tools and languages
Install support for JavaScript, TypeScript, Python, PHP, Azure, Docker a...

Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and ...

Color theme
Make the editor and your code look the way you love

Learn

Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl...

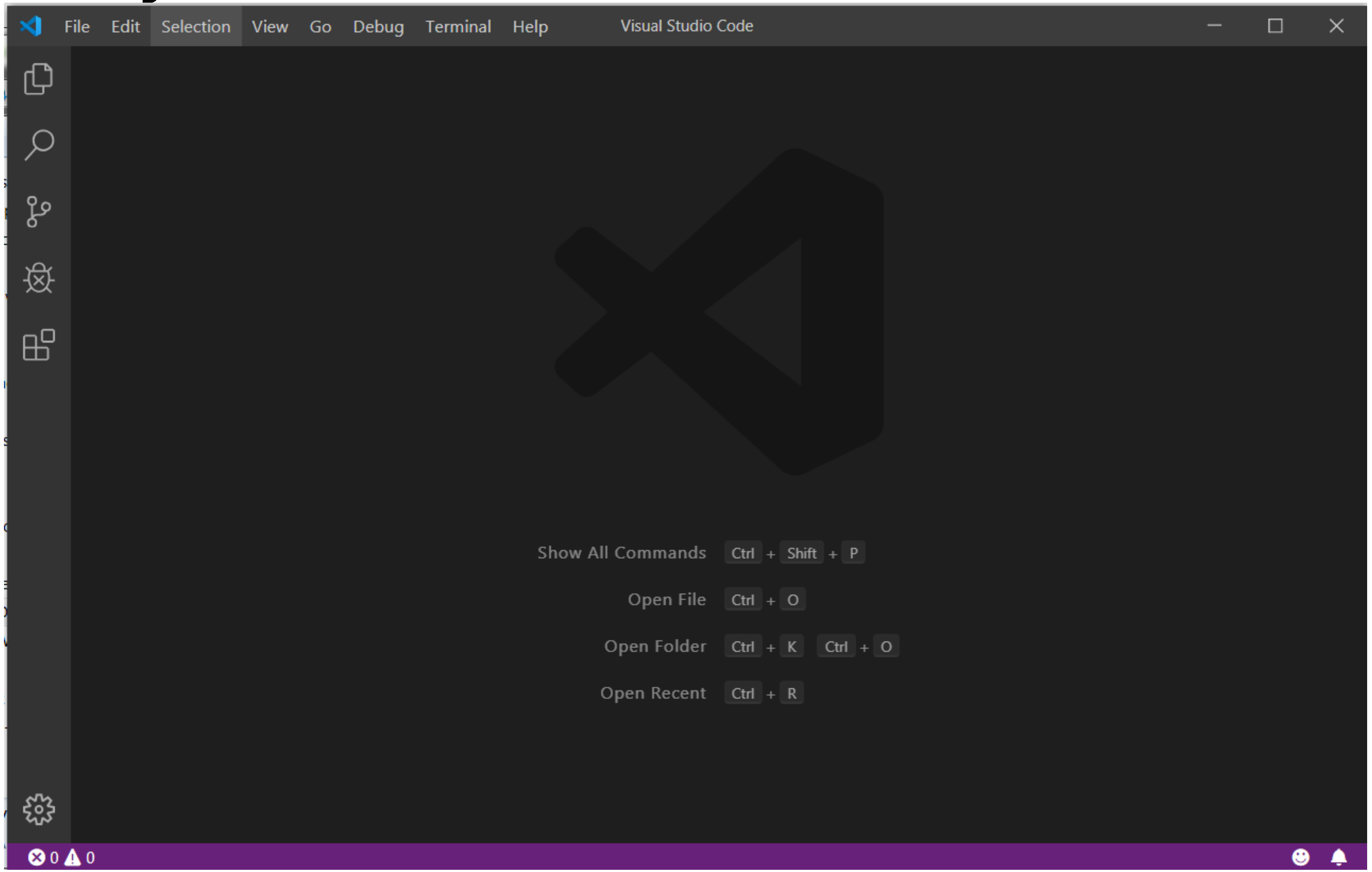
Interface overview
Get

Inte
Try

Help improve VS Code by allowing Microsoft to collect usage data. Read our privacy statement and learn how to opt out.

Install IDE Visual Studio Code

- **Once installed it may show a welcome page that you can close**



Show All Commands Ctrl + Shift + P

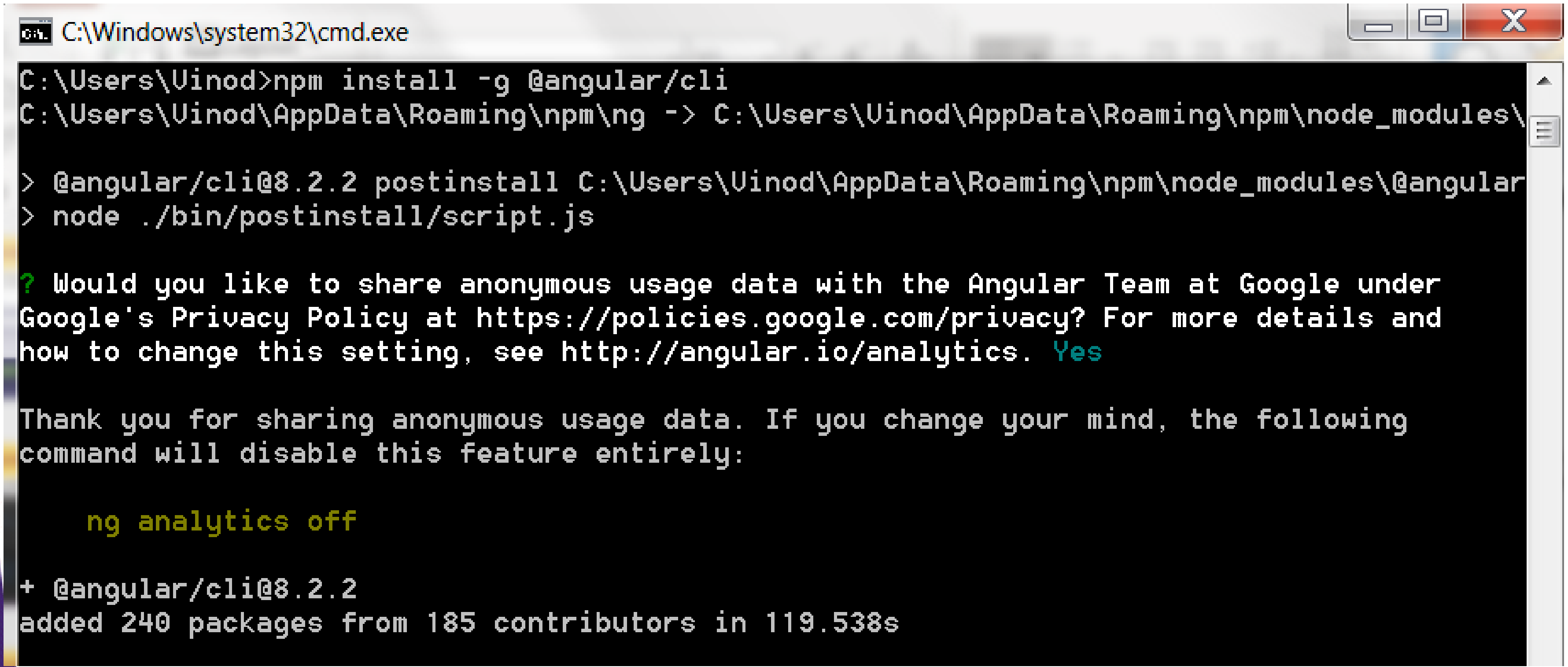
Open File Ctrl + O

Open Folder Ctrl + K Ctrl + O

Open Recent Ctrl + R

Install Angular CLI

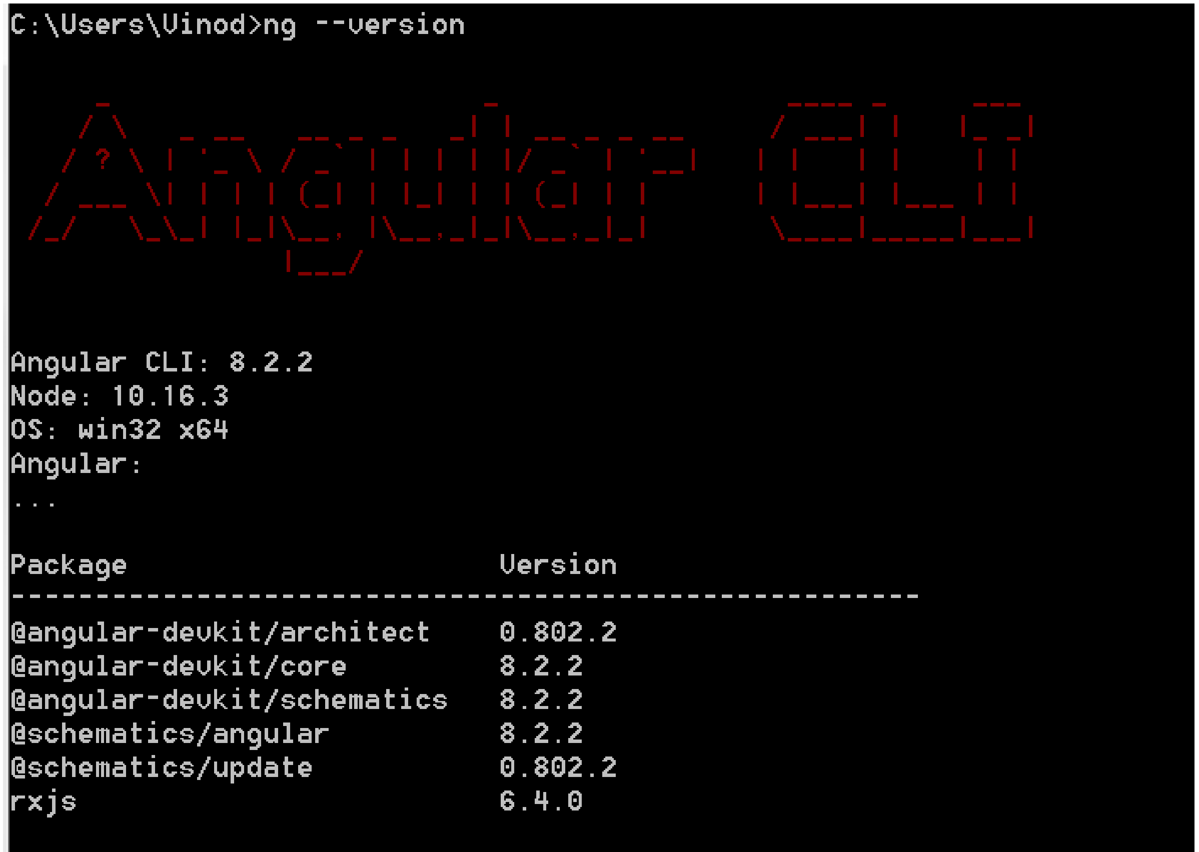
- **Use Node Package Manager (NPM) to install Angular CLI - a third party library**
- $*npm install -g @angular/cli*
- *-g option refers to global installation of CLI*



```
C:\Users\Vinod>npm install -g @angular/cli
C:\Users\Vinod\AppData\Roaming\npm\ng -> C:\Users\Vinod\AppData\Roaming\npm\node_modules\

> @angular/cli@8.2.2 postinstall C:\Users\Vinod\AppData\Roaming\npm\node_modules\@angular
> node ./bin/postinstall/script.js

? Would you like to share anonymous usage data with the Angular Team at Google under
Google's Privacy Policy at https://policies.google.com/privacy? For more details and
how to change this setting, see http://angular.io/analytics. Yes

Thank you for sharing anonymous usage data. If you change your mind, the following
command will disable this feature entirely:

    ng analytics off

+ @angular/cli@8.2.2
added 240 packages from 185 contributors in 119.538s
```

Install Angular CLI

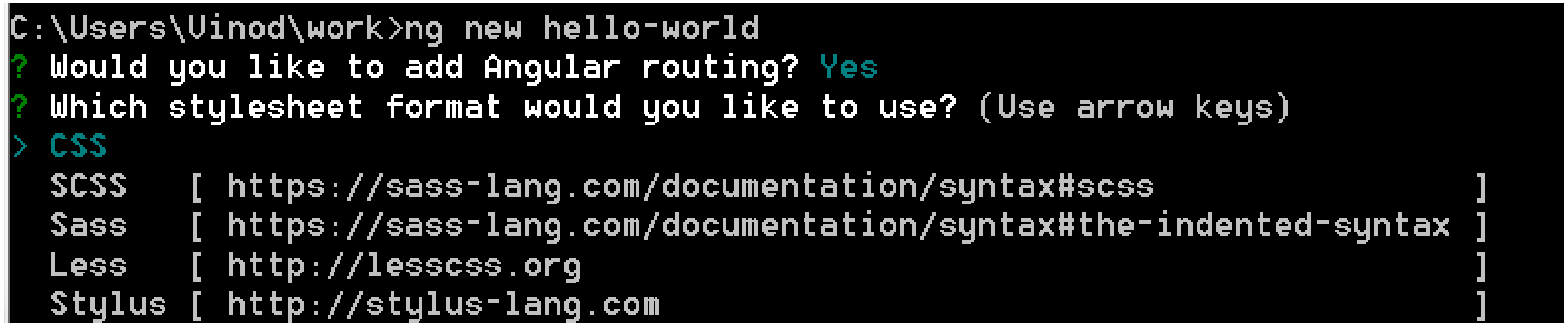
- **To see if Angular CLI is installed, type**
- **$***ng –version*
- *ng is the command to run angular cli*



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

     _                      _                 ____ _     ___
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / ? \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/


Angular CLI: 8.2.2
Node: 10.16.3
OS: win32 x64
Angular:
...

Package                      Version
------------------------------------------------------
@angular-devkit/architect    0.802.2
@angular-devkit/core         8.2.2
@angular-devkit/schematics   8.2.2
@schematics/angular          8.2.2
@schematics/update           0.802.2
rxjs                         6.4.0
```

Let’s create Angular project

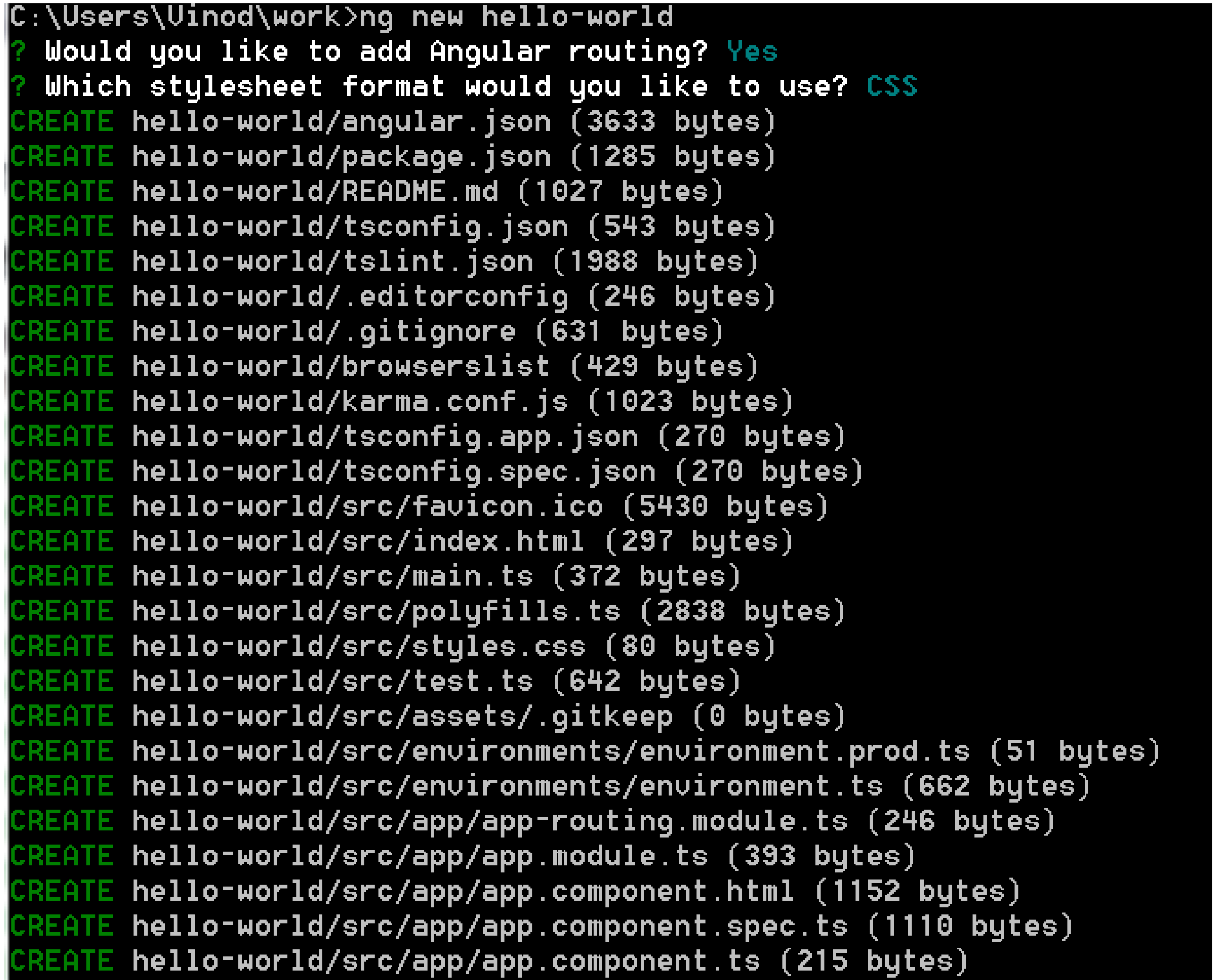
- **The command/syntax to create new angular project**
  - $ng new project-name
- **Create an angular project named hello-world**
  - $ng new hello-world



```
C:\Users\Vinod\work>ng new hello-world
? 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             ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                          ]
  Stylus [ http://stylus-lang.com                                      ]
```

Let’s create Angular project

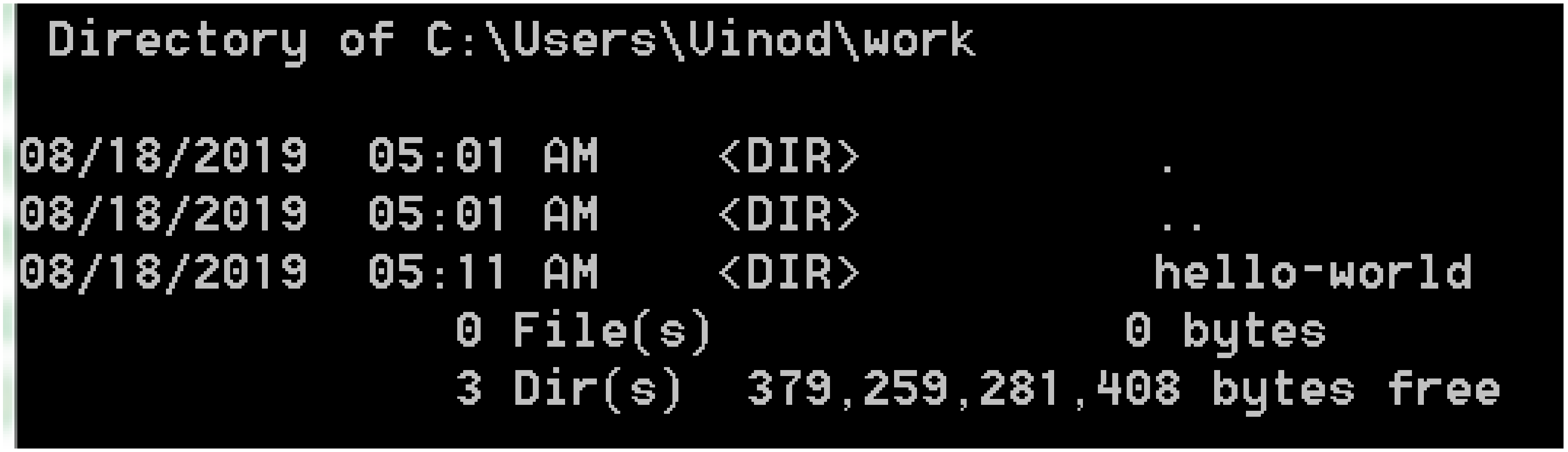
- Create an angular project named hello-world



```
C:\Users\Vinod\work>ng new hello-world
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE hello-world/angular.json (3633 bytes)
CREATE hello-world/package.json (1285 bytes)
CREATE hello-world/README.md (1027 bytes)
CREATE hello-world/tsconfig.json (543 bytes)
CREATE hello-world/tslint.json (1988 bytes)
CREATE hello-world/.editorconfig (246 bytes)
CREATE hello-world/.gitignore (631 bytes)
CREATE hello-world/browserslist (429 bytes)
CREATE hello-world/karma.conf.js (1023 bytes)
CREATE hello-world/tsconfig.app.json (270 bytes)
CREATE hello-world/tsconfig.spec.json (270 bytes)
CREATE hello-world/src/favicon.ico (5430 bytes)
CREATE hello-world/src/index.html (297 bytes)
CREATE hello-world/src/main.ts (372 bytes)
CREATE hello-world/src/polyfills.ts (2838 bytes)
CREATE hello-world/src/styles.css (80 bytes)
CREATE hello-world/src/test.ts (642 bytes)
CREATE hello-world/src/assets/.gitkeep (0 bytes)
CREATE hello-world/src/environments/environment.prod.ts (51 bytes)
CREATE hello-world/src/environments/environment.ts (662 bytes)
CREATE hello-world/src/app/app-routing.module.ts (246 bytes)
CREATE hello-world/src/app/app.module.ts (393 bytes)
CREATE hello-world/src/app/app.component.html (1152 bytes)
CREATE hello-world/src/app/app.component.spec.ts (1110 bytes)
CREATE hello-world/src/app/app.component.ts (215 bytes)
```

Let’s create Angular project

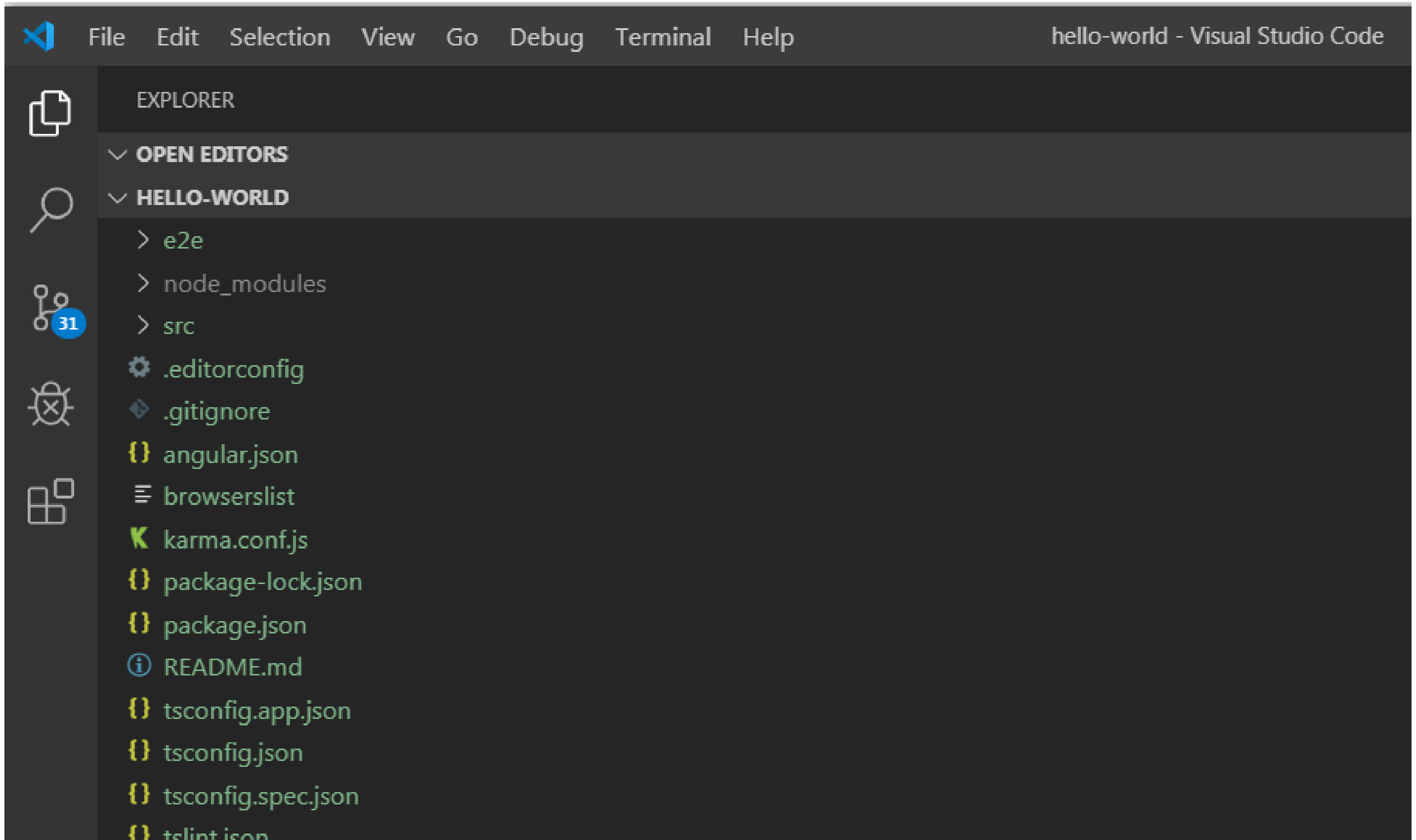
- **Once completed, angular creates a folder** hello-world **with artifacts of angular project**
- **Creates needed files and installs dependencies, such as agnular libraries, which are javascript files**
  - No need to explicitly link js libraries in html.



```
 Directory of C:\Users\Vinod\work

08/18/2019  05:01 AM    <DIR>          .
08/18/2019  05:01 AM    <DIR>          ..
08/18/2019  05:11 AM    <DIR>          hello-world
               0 File(s)              0 bytes
               3 Dir(s)  379,259,281,408 bytes free
```

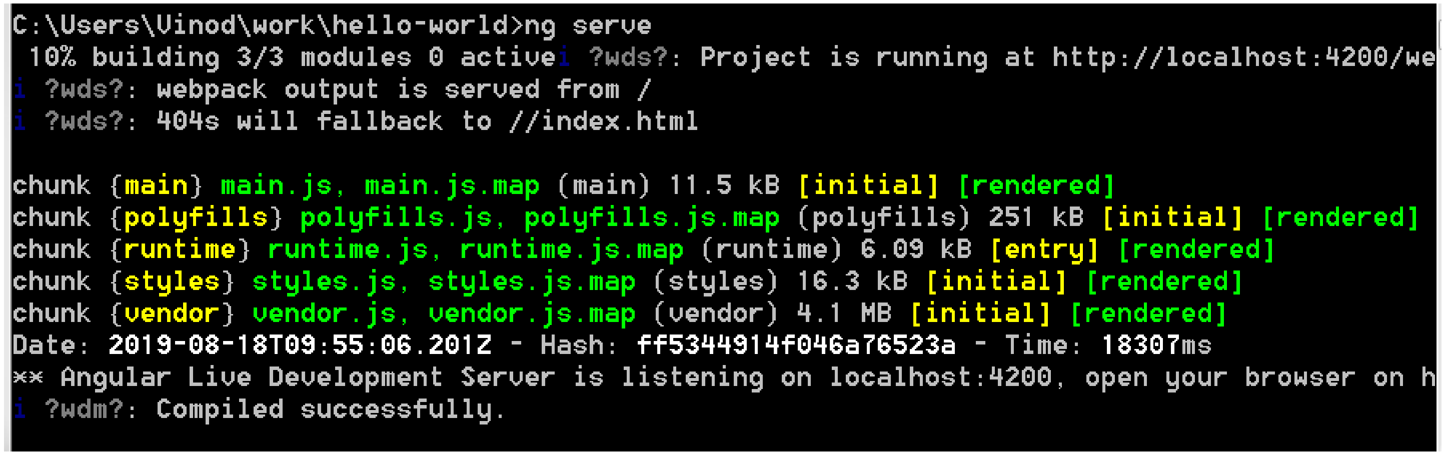
Let's create Angular project

- **Open the hello-world Angular project in Visual Studio Code**
  - **src folder contains all the source code**
  - package.json will contain all dependencies



Running the Angular project

- To run the project type $**ng serve** inside of project folder hello-world to run the development angular server

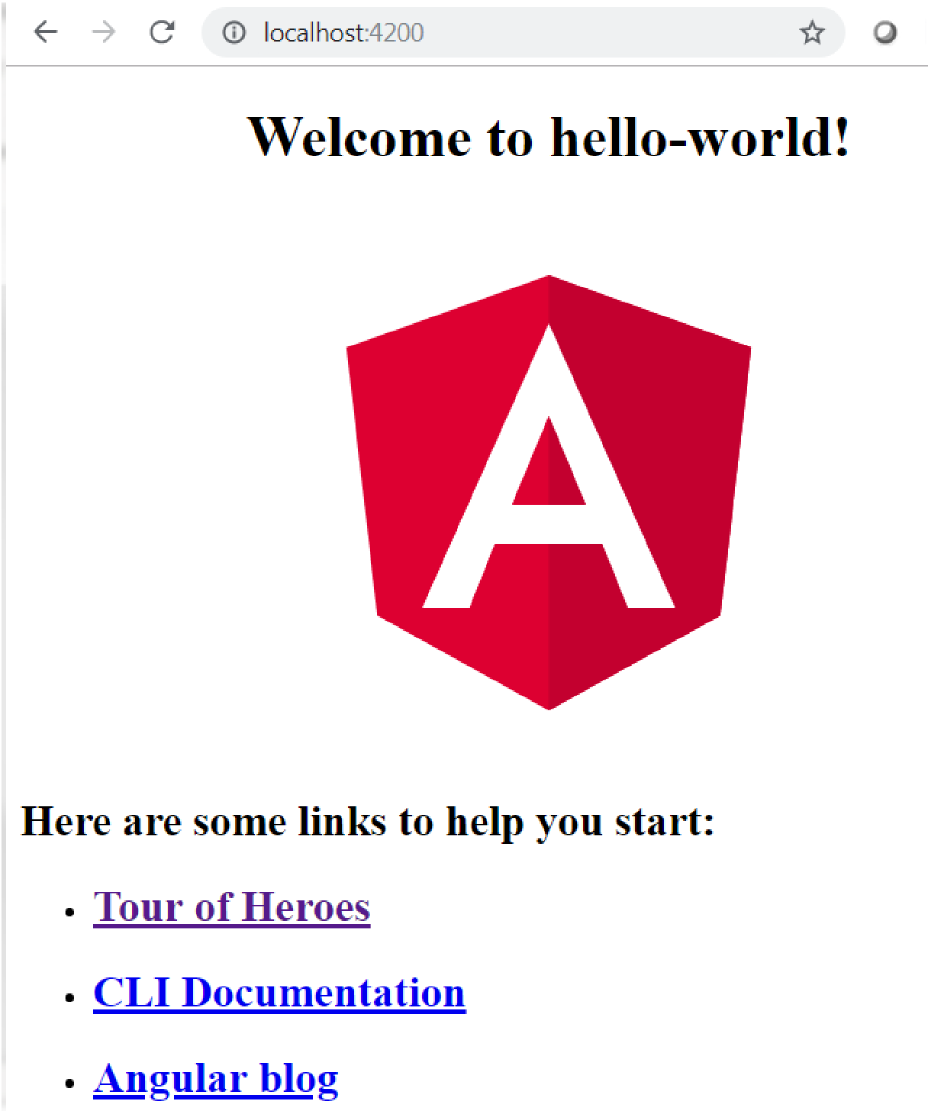


```
C:\Users\Vinod\work\hello-world>ng serve
 10% building 3/3 modules 0 activei ?wds?: Project is running at http://localhost:4200/we
i ?wds?: webpack output is served from /
i ?wds?: 404s will fallback to //index.html

chunk {main} main.js, main.js.map (main) 11.5 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 251 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.09 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.3 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 4.1 MB [initial] [rendered]
Date: 2019-08-18T09:55:06.201Z - Hash: ff5344914f046a76523a - Time: 18307ms
** Angular Live Development Server is listening on localhost:4200, open your browser on h
i ?wdm?: Compiled successfully.
```

- Open the browser on **http://localhost:4200**

Running the Angular project

- Open the browser on **http://localhost:4200**
- **Shows default angular page**


localhost:4200

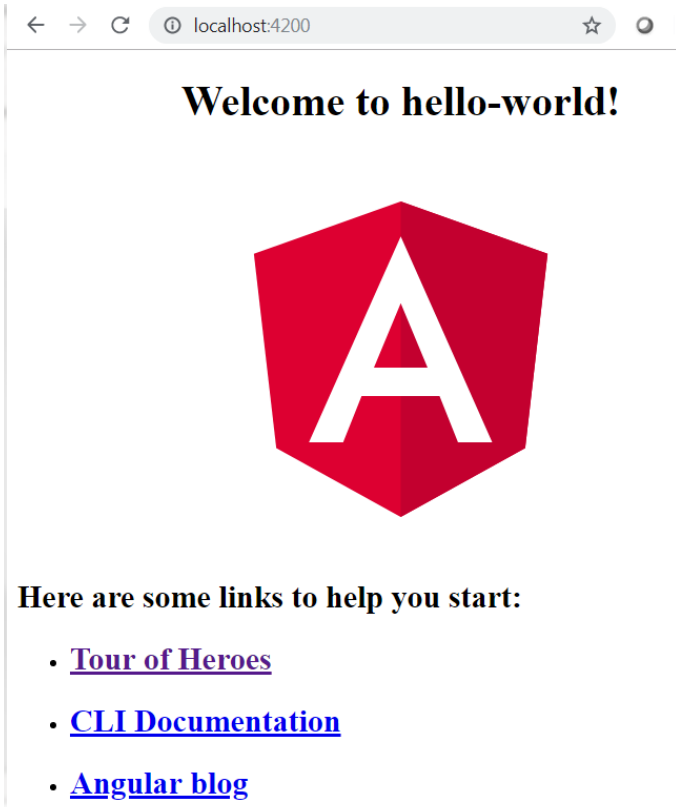
Welcome to hello-world!

Here are some links to help you start:

- Tour of Heroes
- CLI Documentation
- Angular blog

Running the Angular project

- The html of angular page created by default is in app-component.html

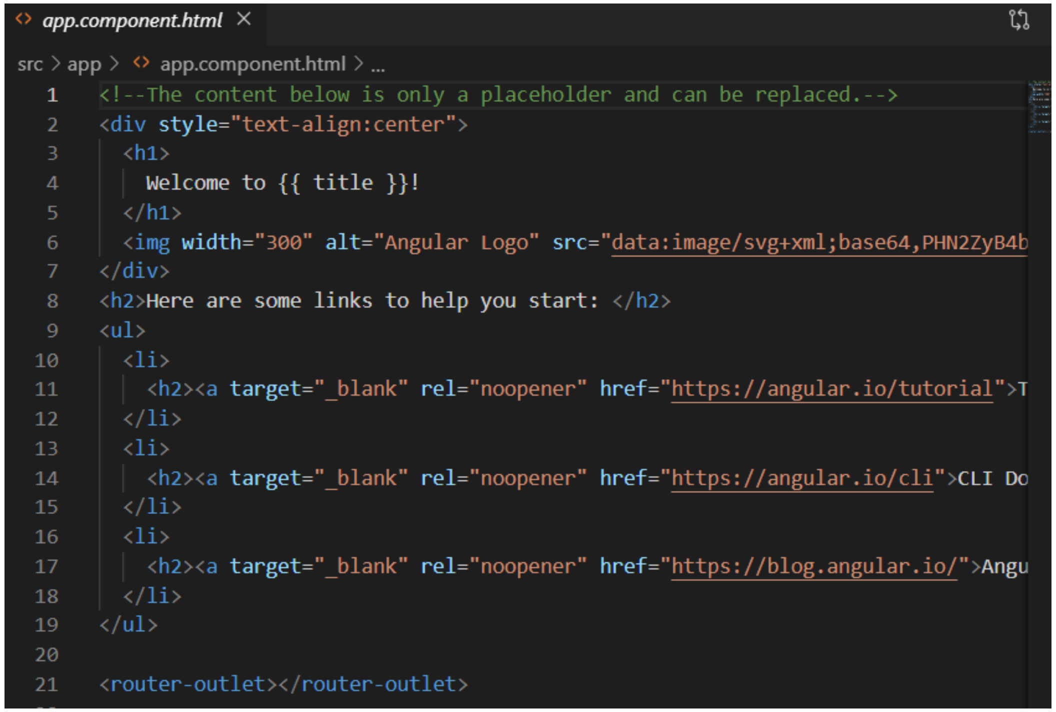


localhost:4200

Welcome to hello-world!

Here are some links to help you start:

- Tour of Heroes
- CLI Documentation
- Angular blog



app.component.html

src > app > app.component.html > ...

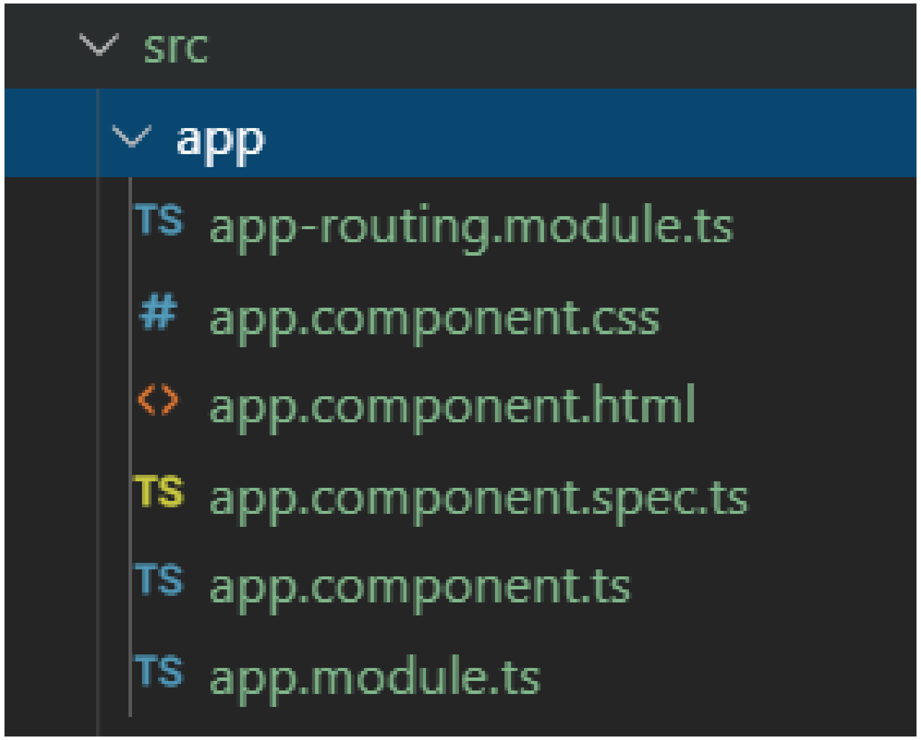
```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="data:image/svg+xml;base64,PHN2ZyB4b
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/tutorial">T
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/cli">CLI Do
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Angu
  </li>
</ul>

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

Angular Components

- **Components define areas of responsibility in the user interface, or UI,**
  - that let you reuse sets of UI functionality.
- **A component consists of three things:**
  - A component class that handles data and functionality.
  - An HTML template that determines the UI.
  - Component-specific styles that define the look and feel.
- **An Angular application comprises a tree of components,**
  - in which each Angular component has a specific purpose and responsibility.

Angular Components

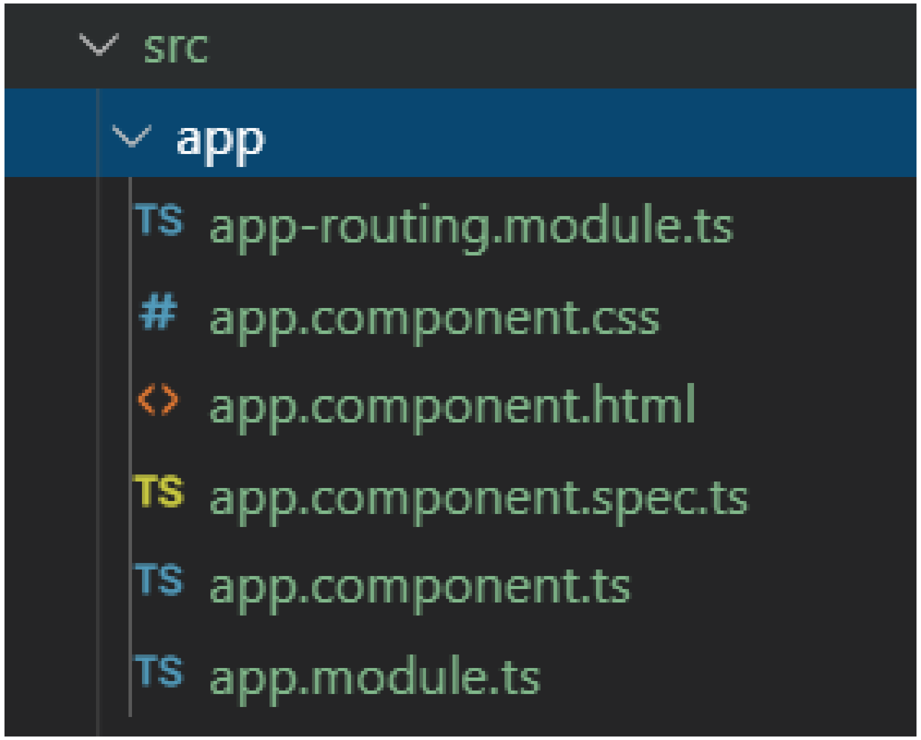
- **Angular is based on components**
– **Angular project creates the main component called app.component**
  – that shows the “Wellcome to App” page on localhost:4200



```
src
  app
    app-routing.module.ts
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
```

Angular Components

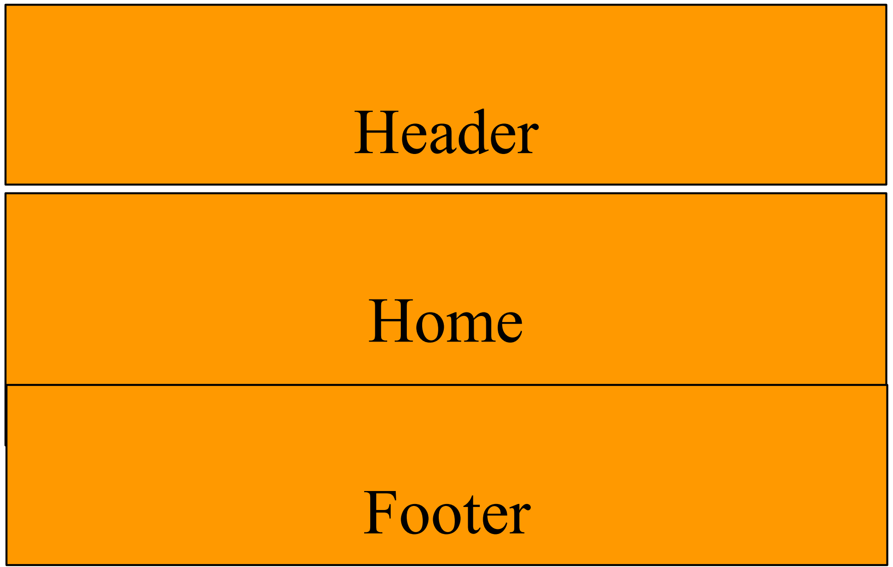
- **Angular component have 3 basics parts**
  - name-component.html (the html code of component)
  - name-component.css (css style of component)
  - name-component.ts (the typescript of component)



```
src
  app
    app-routing.module.ts
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
```

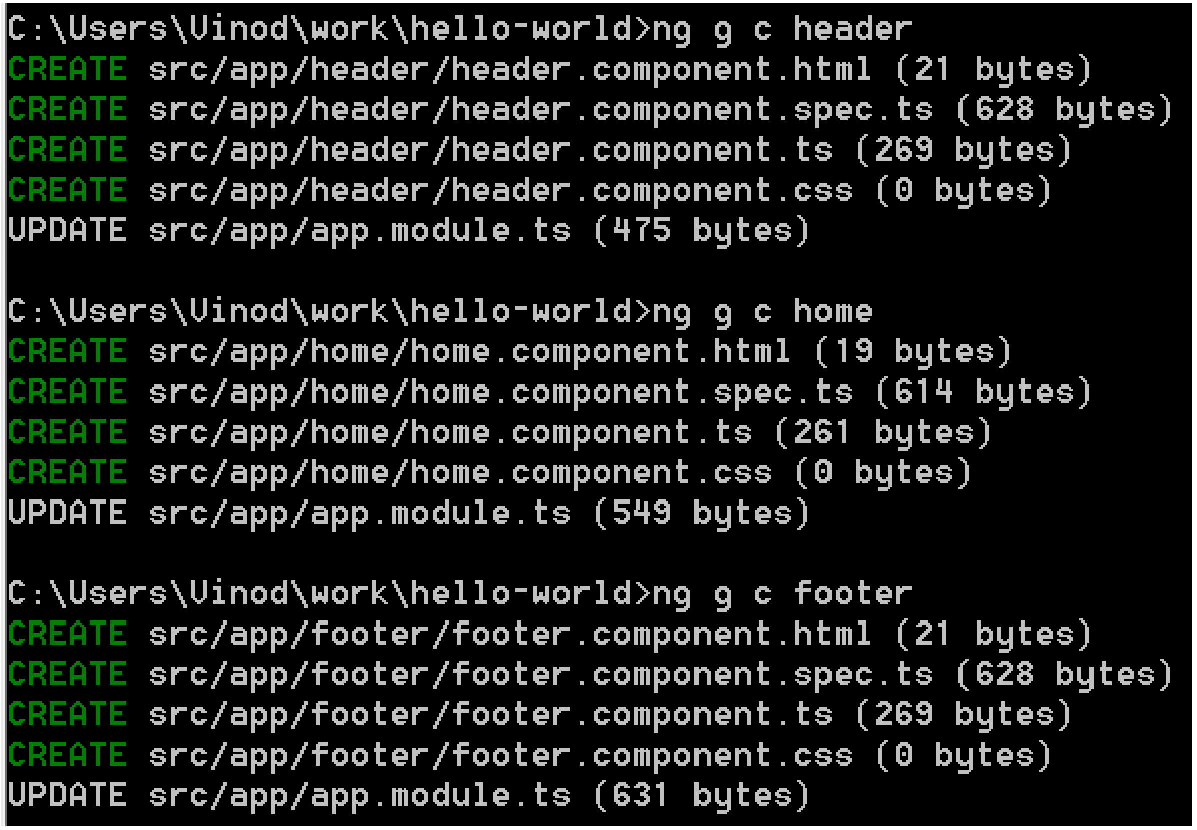
Angular Components

- **Let's say our application will have three components (i.e., parts)**
  - Header, Home, Footer
- **Angular CLI command to create a new component**
  - $ng generate component *name-of-the-component*
  - $ng g c *name-of-the-component* (short notation for the same)



Angular components

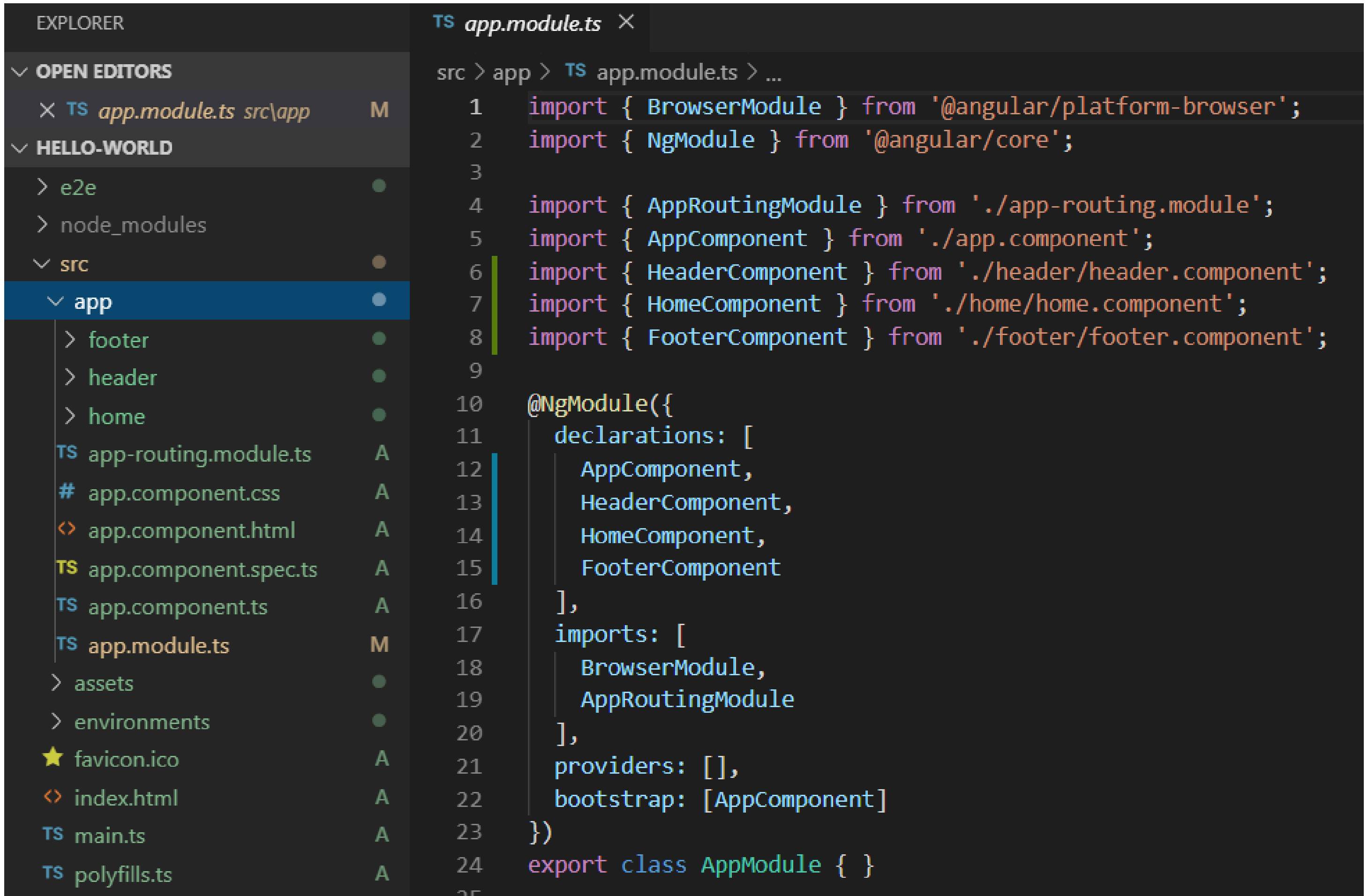
- Let's **create three components** inside **hello-world project**
  - $ng g c header
  - $ng g c home
  - $ng g c footer



```
C:\Users\Vinod\work\hello-world>ng g c header
CREATE src/app/header/header.component.html (21 bytes)
CREATE src/app/header/header.component.spec.ts (628 bytes)
CREATE src/app/header/header.component.ts (269 bytes)
CREATE src/app/header/header.component.css (0 bytes)
UPDATE src/app/app.module.ts (475 bytes)

C:\Users\Vinod\work\hello-world>ng g c home
CREATE src/app/home/home.component.html (19 bytes)
CREATE src/app/home/home.component.spec.ts (614 bytes)
CREATE src/app/home/home.component.ts (261 bytes)
CREATE src/app/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (549 bytes)

C:\Users\Vinod\work\hello-world>ng g c footer
CREATE src/app/footer/footer.component.html (21 bytes)
CREATE src/app/footer/footer.component.spec.ts (628 bytes)
CREATE src/app/footer/footer.component.ts (269 bytes)
CREATE src/app/footer/footer.component.css (0 bytes)
UPDATE src/app/app.module.ts (631 bytes)
```

Angular Components

- New components show up under src/app and gets updated in app.module.ts


EXPLORER

OPEN EDITORS

× TS app.module.ts src\app M

HELLO-WORLD

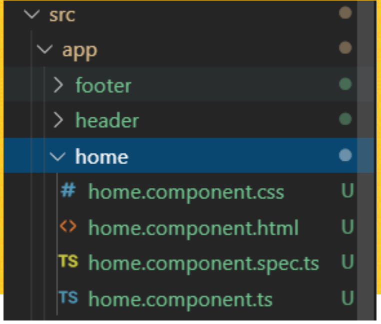
- e2e
- node_modules
- src
  - app
    - footer
    - header
    - home
    - app-routing.module.ts A
    - app.component.css A
    - app.component.html A
    - app.component.spec.ts A
    - app.component.ts A
    - app.module.ts M
  - assets
  - environments
  - favicon.ico A
  - index.html A
  - main.ts A
  - polyfills.ts A

app.module.ts

src > app > app.module.ts > ...

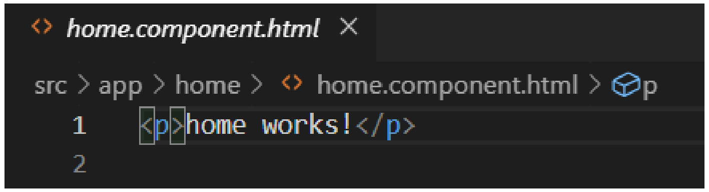
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { HomeComponent } from './home/home.component';
import { FooterComponent } from './footer/footer.component';

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

Angular Components



```
src
  app
    footer
    header
    home
      home.component.css        U
      home.component.html       U
      home.component.spec.ts    U
      home.component.ts         U
```

- Each component has
  - A simple html page

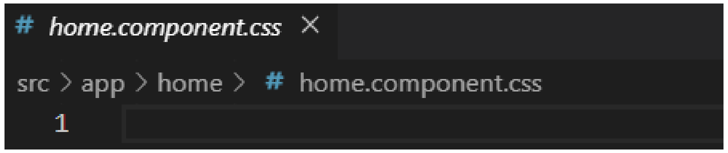


home.component.html

src > app > home > home.component.html > p

```html
<p>home works!</p>

```

  - An empty css file

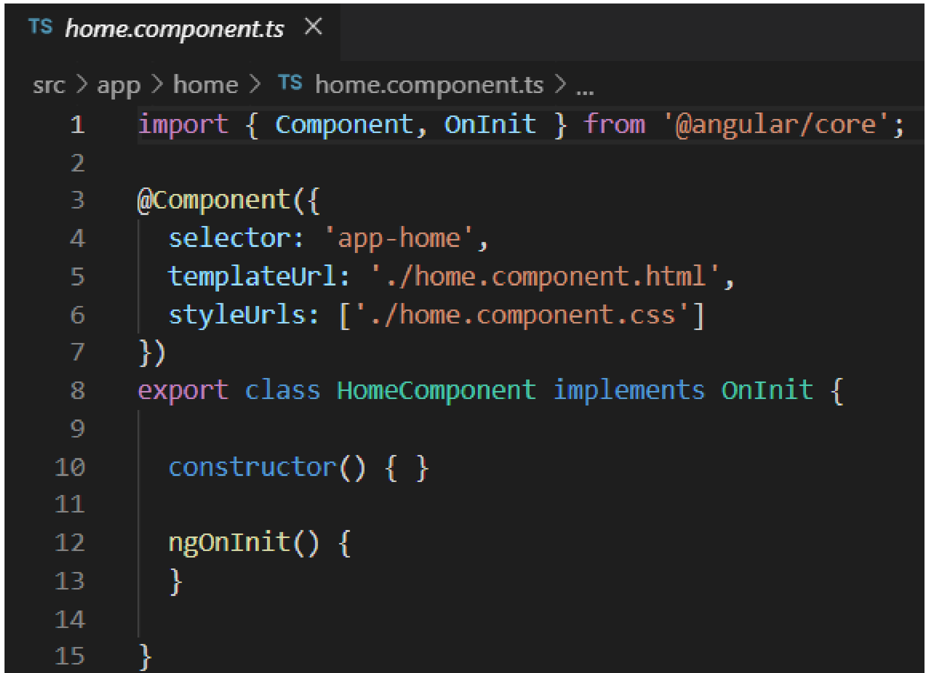


home.component.css

src > app > home > home.component.css

```css

```

  - A typescript class



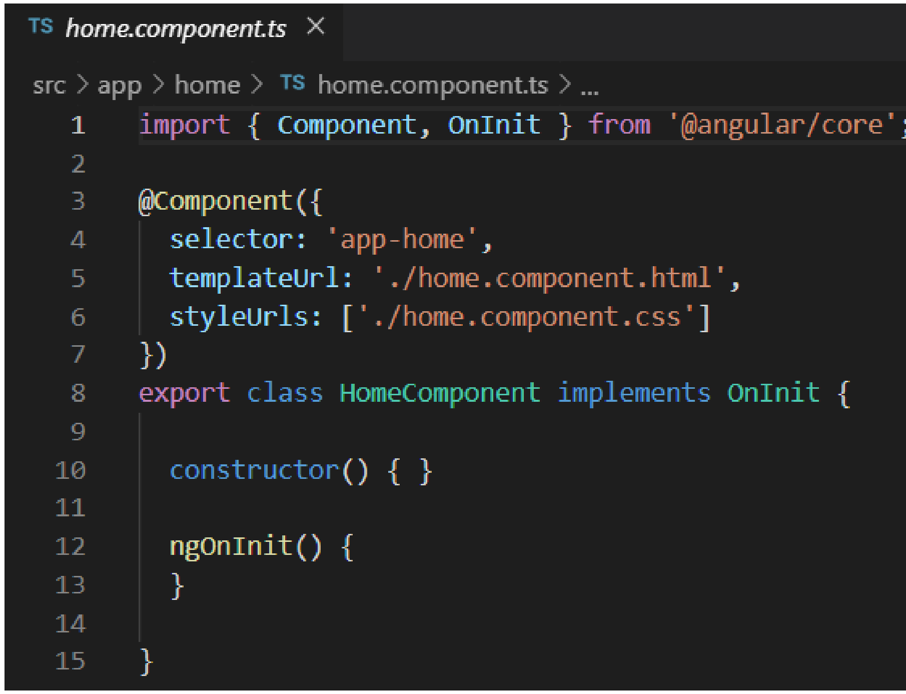
home.component.ts

src > app > home > home.component.ts > ...

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

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

  constructor() { }

  ngOnInit() {
  }

}
```

Angular Components

- **Every component.ts file contains two parts:**
  - The TypeScript class (e.g., HomeComponent) and
  - The registration with the angular using @Component annotation, which is the TypeScript way of adding metadata to the class, such as
    - selector (e.g., app-home) - used as a tag inside html files
    - templateUrl to access the component and
    - the styleUrls



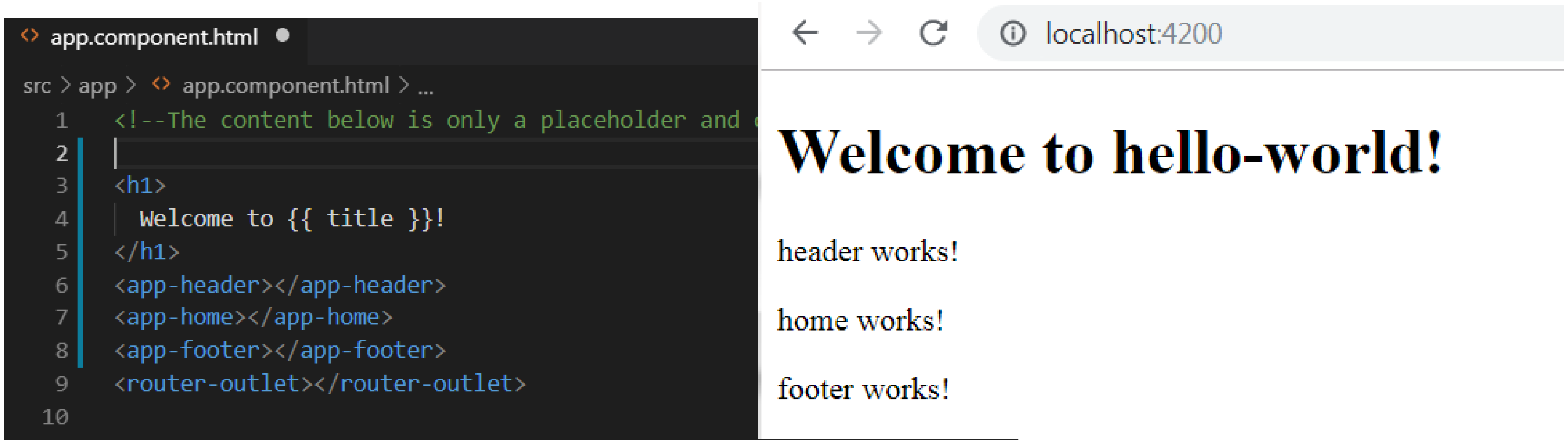
home.component.ts ×

src > app > home > home.component.ts > ...

```
import { Component, OnInit } from '@angular/core'

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

  constructor() { }

  ngOnInit() {
  }

}
```

Angular Component

- Let's **replace the content** of the template **app.component.html** file by **putting components selectors** in the order of the components to be **shown**



app.component.html ●

src > app > app.component.html > ...

```
<!--The content below is only a placeholder and

<h1>
  Welcome to {{ title }}!
</h1>
<app-header></app-header>
<app-home></app-home>
<app-footer></app-footer>
<router-outlet></router-outlet>

```

localhost:4200

Welcome to hello-world!

header works!

home works!

footer works!

- This renders the contents of the three components

Data Binding

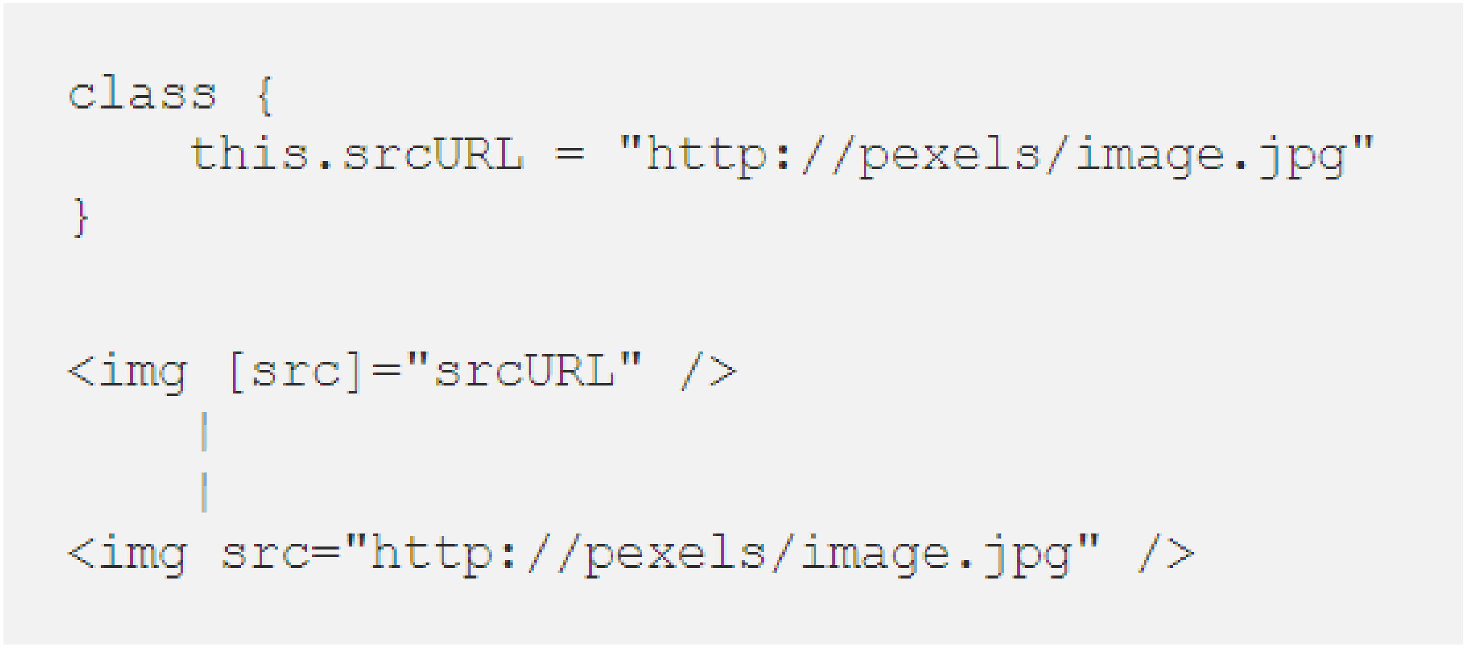
- Data binding is a technique to link your data to your view layer.
- By binding a variable, the angular framework watches it for changes.
- If changes are detected, the framework takes care of updating the view accordingly.
- Data binding consists of *one way data binding* and *two way data binding*.

Property Binding

- Property binding is one way of binding data in Angular.
- The square braces are used to bind data to a property of an element,
- The trick is to put the property onto the element wrapped in brackets: [property].
  - See example on the next page

Property Binding

- The src property of the HTMLElement img is bound to the srcURL property of the class.
- Whenever the srcURL property changes the src property of the img element changes.



```
class {
    this.srcURL = "http://pexels/image.jpg"
}


<img [src]="srcURL" />
     |
     |
<img src="http://pexels/image.jpg" />
```

Angular Interpolation

- **Angular interpolation is used to display a component (class) property in the respective view (html) template with double curly braces {{ }} syntax**
  - We can display all kind of properties data into view e.g. string, number, date, arrays, list or map.
- **Interpolation is used for one way data binding.**
  - Data binding consist of one way data binding and two way data binding.
- **Interpolation moves data in one direction from our components to HTML elements.**

Angular Interpolation

- **The property name to be displayed in the view template is enclosed in double curly braces {{ }} also known as moustache syntax.**
- **Angular automatically pulls the value of the propertyName and object.propertyName from the component and inserts those values into the browser.**
- **Angular updates the display when these properties change.**

Angular Interpolation Usages

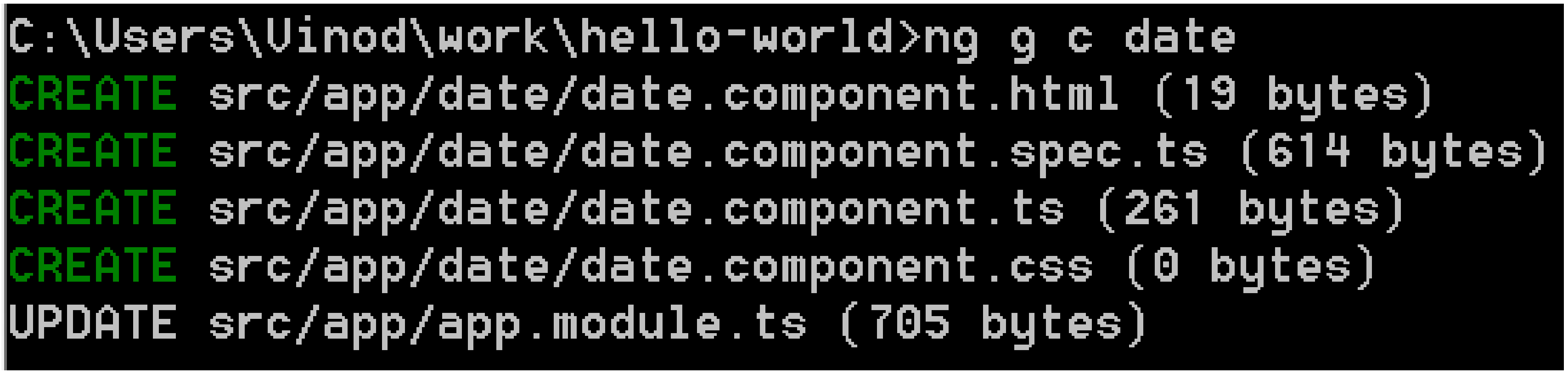
- **Display simple properties**
  - To display and evaluate strings into the text between HTML element tags and within attribute assignments.
- **Evaluate arithmetic expressions**
  - To evaluate arithmetic expressions present within the curly braces.
- **Invoke methods and display return values**
  - To invoke methods on hosting component views within interpolation expressions.
- **Display array items**
  - We can use interpolation along with ngFor directive to display an array of items.

Interpolation Example

- **Refers to a mechanism of binding data in the component class to the html template using {{ }} – the interpolation syntax**
  - This means having some value in the class (.ts file) e.g. with member variables and then showing it in the View (html template)
  - Any changes in the data automatically gets updated in the view
  - This is also referred to as Interpolation
  - <u>This is only one way – from model to view!</u>

Interpolation Example

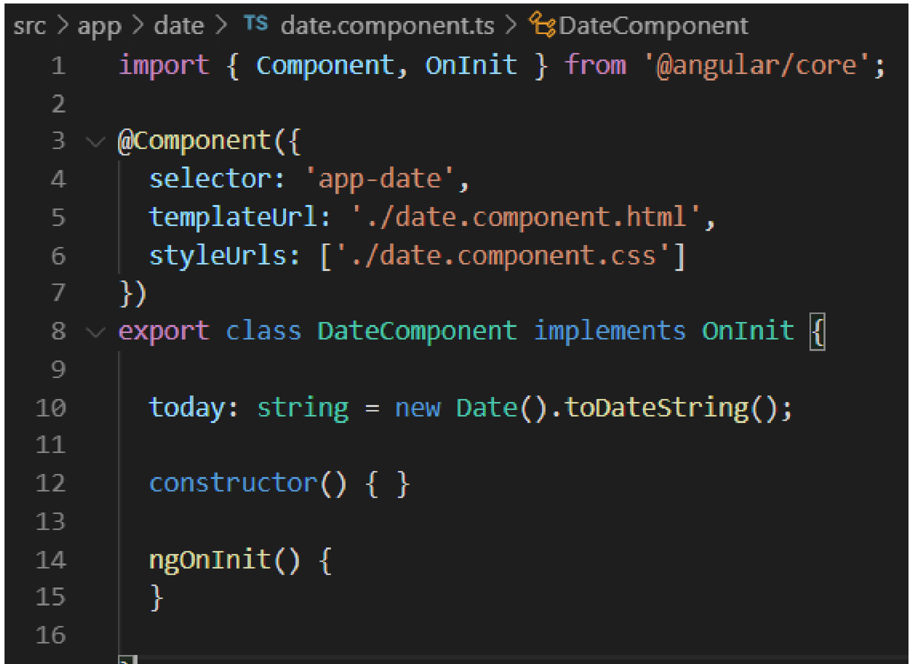
- **Let’s create a new component date inside hello-world**
-



```
C:\Users\Vinod\work\hello-world>ng g c date
CREATE src/app/date/date.component.html (19 bytes)
CREATE src/app/date/date.component.spec.ts (614 bytes)
CREATE src/app/date/date.component.ts (261 bytes)
CREATE src/app/date/date.component.css (0 bytes)
UPDATE src/app/app.module.ts (705 bytes)
```

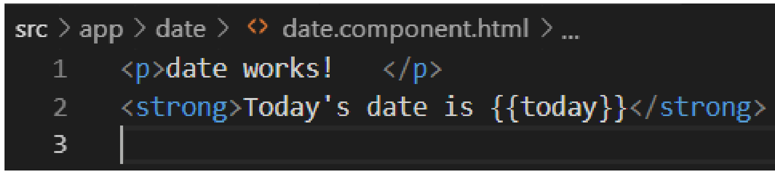
Interpolation Example

- **Let's add some dynamic functionality to the date component**
  - Add a member variable today to the DateComponent class inside date-component.ts file and
  - Then rendering the value of today inside the date-component.html file using {{ }} syntax.



src > app > date > TS date.component.ts > DateComponent

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

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

  today: string = new Date().toDateString();

  constructor() { }

  ngOnInit() {
  }

```

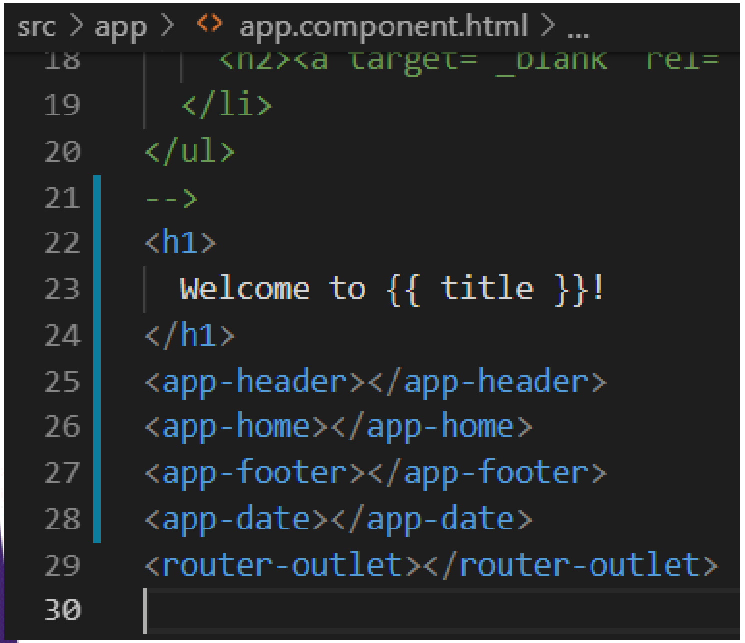


src > app > date > <> date.component.html > ...

```html
<p>date works!   </p>
<strong>Today's date is {{today}}</strong>

```

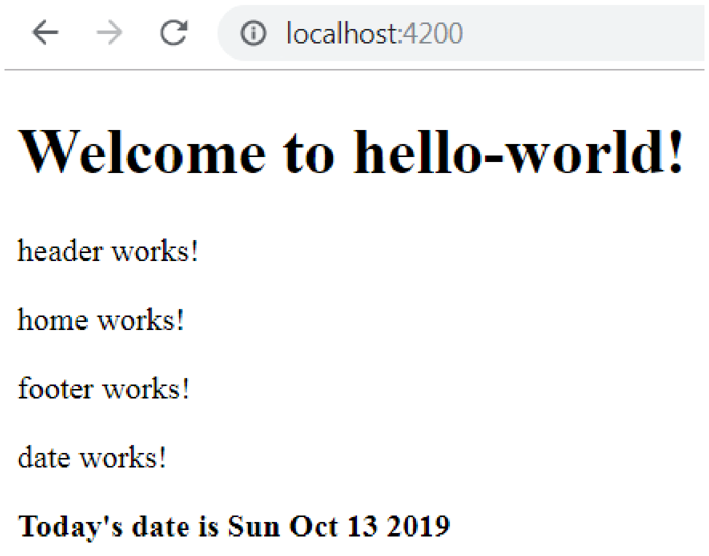
Interpolation Example

- **Add <app-date> component selector in app-component.html**
  - Save all files
  - Restart the server with $ng serve
  - Go to the localhost:4200 to see the value of today dynamically rendered on the page



src > app > app.component.html > ...

```
      </li>
    </ul>
    -->
    <h1>
      Welcome to {{ title }}!
    </h1>
    <app-header></app-header>
    <app-home></app-home>
    <app-footer></app-footer>
    <app-date></app-date>
    <router-outlet></router-outlet>

```



localhost:4200

Welcome to hello-world!

header works!

home works!

footer works!

date works!

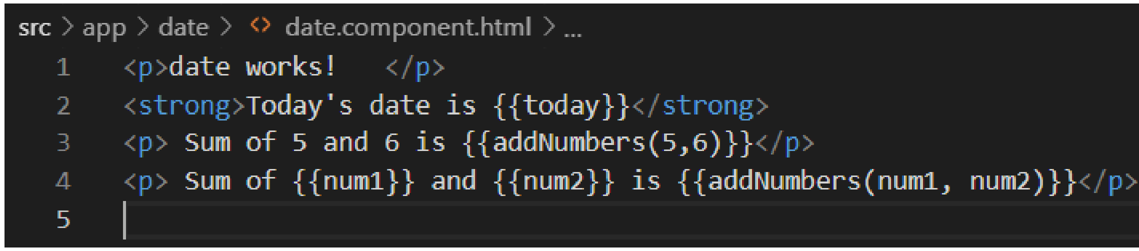
Today's date is Sun Oct 13 2019

Interpolation Example

- **The double curly braces {{ }} in the view component triggers Angular to do interpolation**
  - String interpolation, expression interpolation
- **The content inside the double curly braces in the view component (.html file) gets evaluated and**
  - the result of that evaluation gets plugged in place of double curly.
  - This includes a call to the method of the component class

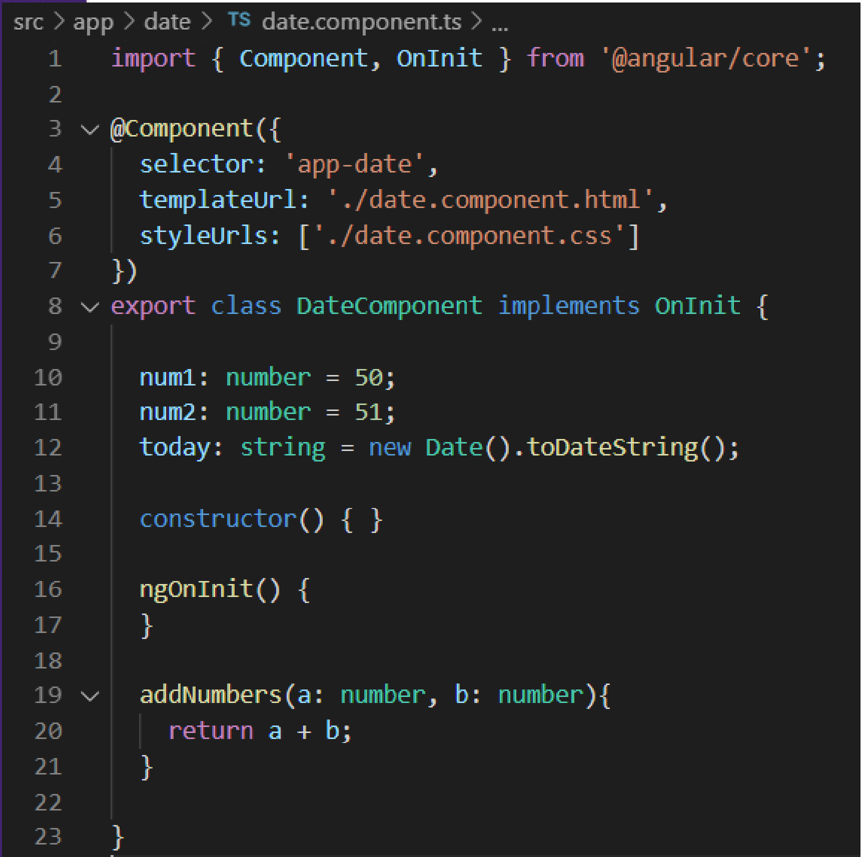
Interpolation Example

- Example



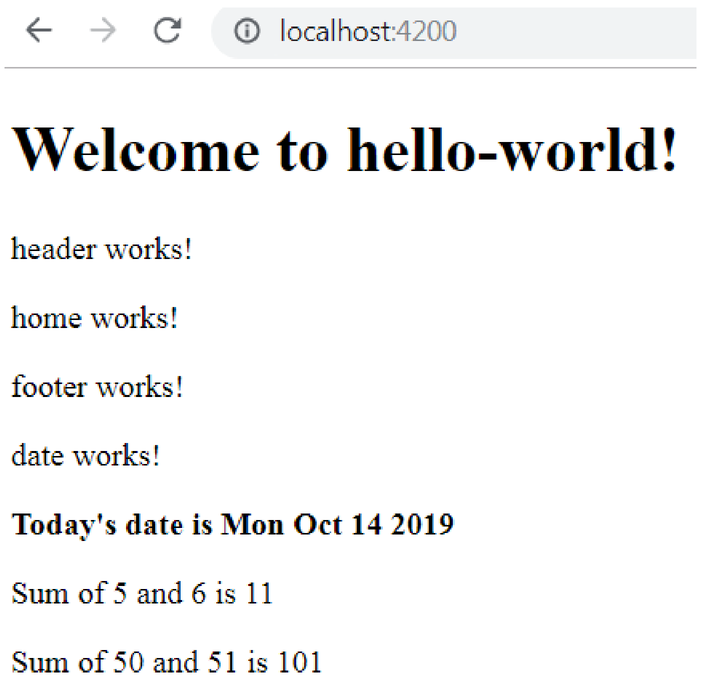
src > app > date > date.component.html > ...

```
<p>date works!   </p>
<strong>Today's date is {{today}}</strong>
<p> Sum of 5 and 6 is {{addNumbers(5,6)}}</p>
<p> Sum of {{num1}} and {{num2}} is {{addNumbers(num1, num2)}}</p>

```



src > app > date > TS date.component.ts > ...

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

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

  num1: number = 50;
  num2: number = 51;
  today: string = new Date().toDateString();

  constructor() { }

  ngOnInit() {
  }

  addNumbers(a: number, b: number){
    return a + b;
  }

}
```

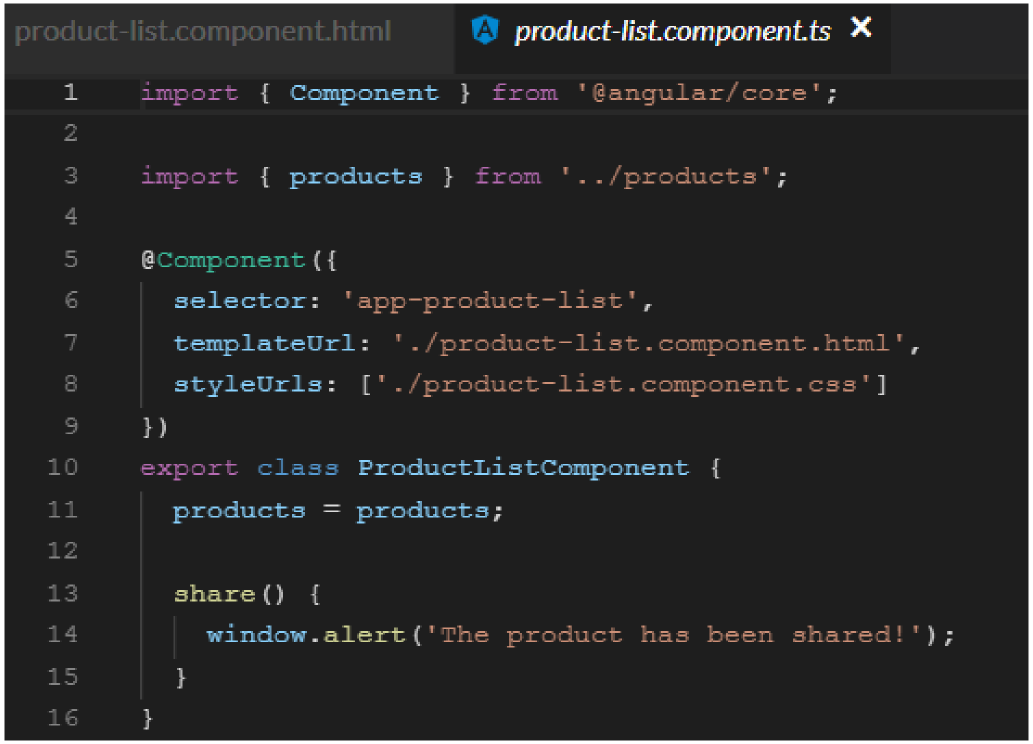


localhost:4200

```
Welcome to hello-world!

header works!

home works!

footer works!

date works!

Today's date is Mon Oct 14 2019

Sum of 5 and 6 is 11

Sum of 50 and 51 is 101
```

Event Binding

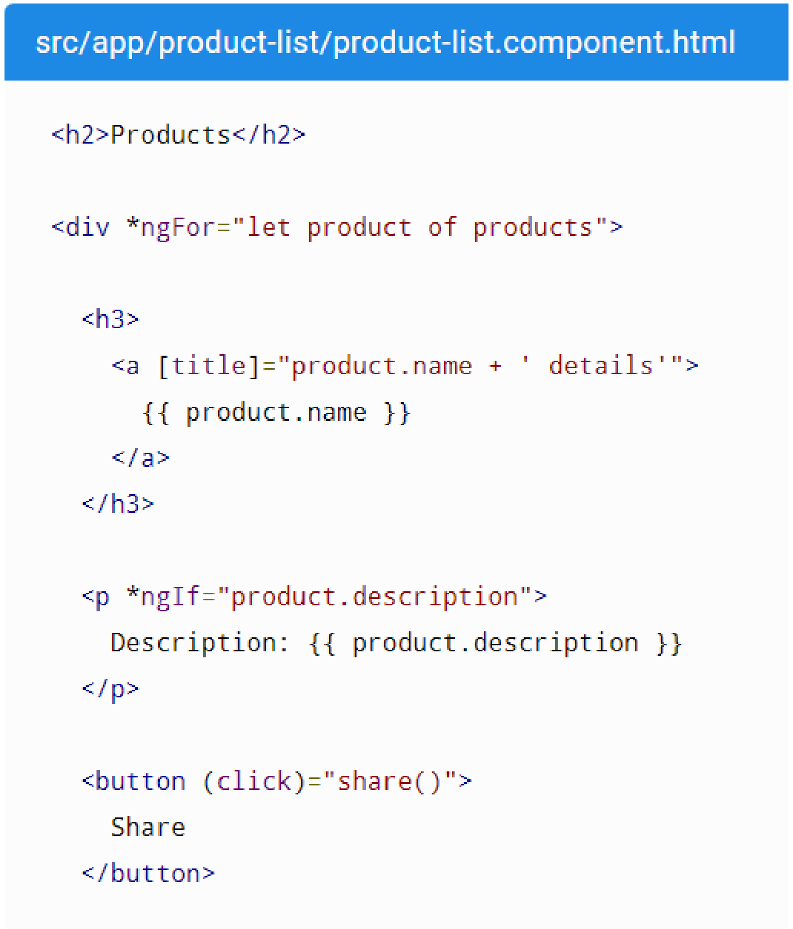
- Event binding **uses a set of parentheses, ( ), around the event**
- To **bind** the button's **click event** to the **share() method** (in product-list.component.ts)



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

import { products } from '../products';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent {
  products = products;

  share() {
    window.alert('The product has been shared!');
  }
}
```

Event Binding

- **Event binding uses a set of parentheses, ( ), around the event,**
  - as putting () around click event in the following button element:

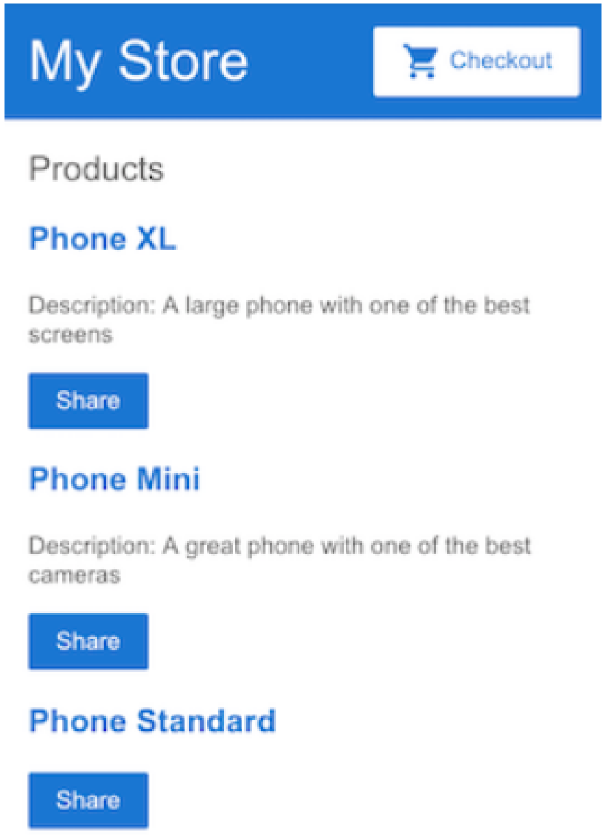


src/app/product-list/product-list.component.html

```
<h2>Products</h2>

<div *ngFor="let product of products">

  <h3>
    <a [title]="product.name + ' details'">
      {{ product.name }}
    </a>
  </h3>

  <p *ngIf="product.description">
    Description: {{ product.description }}
  </p>

  <button (click)="share()">
    Share
  </button>
```

Event Binding

- Each product now has a "Share" button
- Pressing the "Share" button shows the dialog whose content comes from the share() method of product-list.component.ts



Products

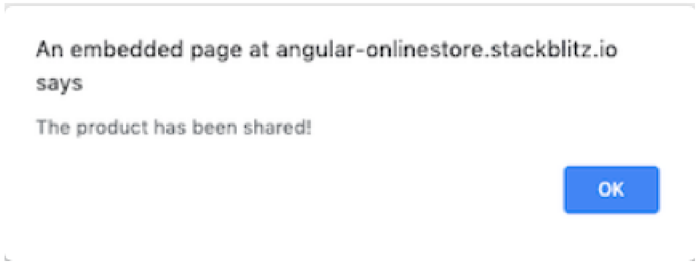
Phone XL

Description: A large phone with one of the best screens

Phone Mini

Description: A great phone with one of the best cameras

Phone Standard



An embedded page at angular-onlinestore.stackblitz.io says

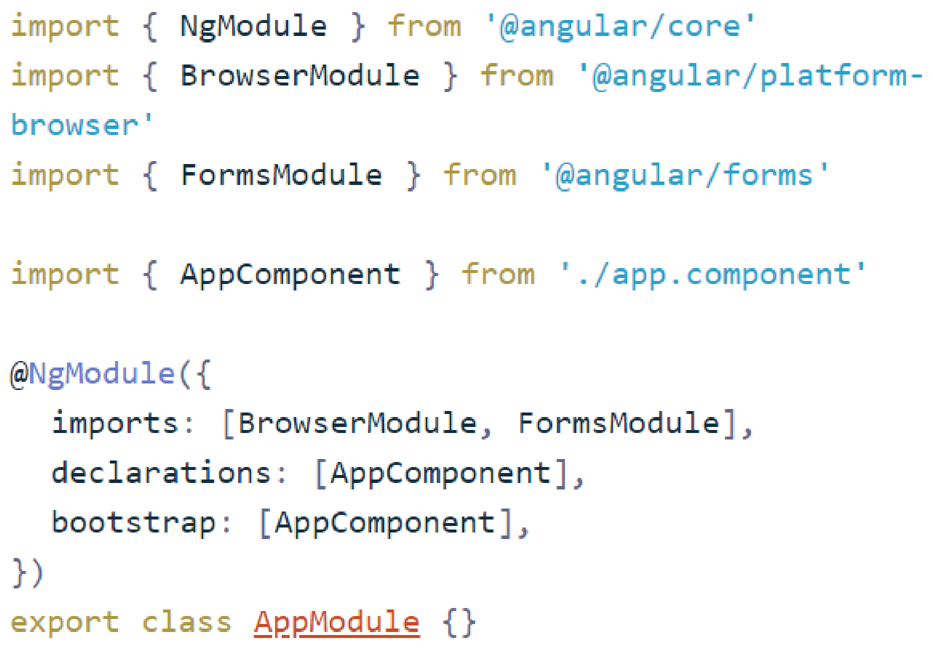
The product has been shared!

Two-way Data Binding

- **This essentially means that if there is data in the component (model), bind it to the view; and if the data changes in view then bind it back to the component.**
- **With two-way data binding, the angular framework not only watches your variables in the model (.ts file) for changes, it also keeps track of changes made by the user (for example with input-elements) and updates the variables accordingly.**
  - That way, the variables in the component code always represent what is displayed in the view.
- **A commonly used directive that makes two-way data binding possible is called ngModel.**

Two-way Data Binding

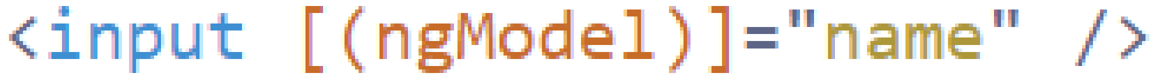
- ngModel is part of the angular "FormsModule" and has to be imported into your module manually.



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

import { AppComponent } from './app.component'

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

Two-way Data Binding

- ngModel can be used with form-elements like inputs to implement two-way data binding.
- To do that, we use a special syntax: [(ngModel)]
  - Its a combination of the one-way- and the event binding syntax.
  - Generally referred to as a “Banana In A Box” syntax
- It is used as follows in the .html file:



```
<input [(ngModel)]="name" />
```

- Using this syntax the value of the variable "name" is not only shown as the value of the input, but both values change when the user types into the input field.

Looping with *ngFor directive

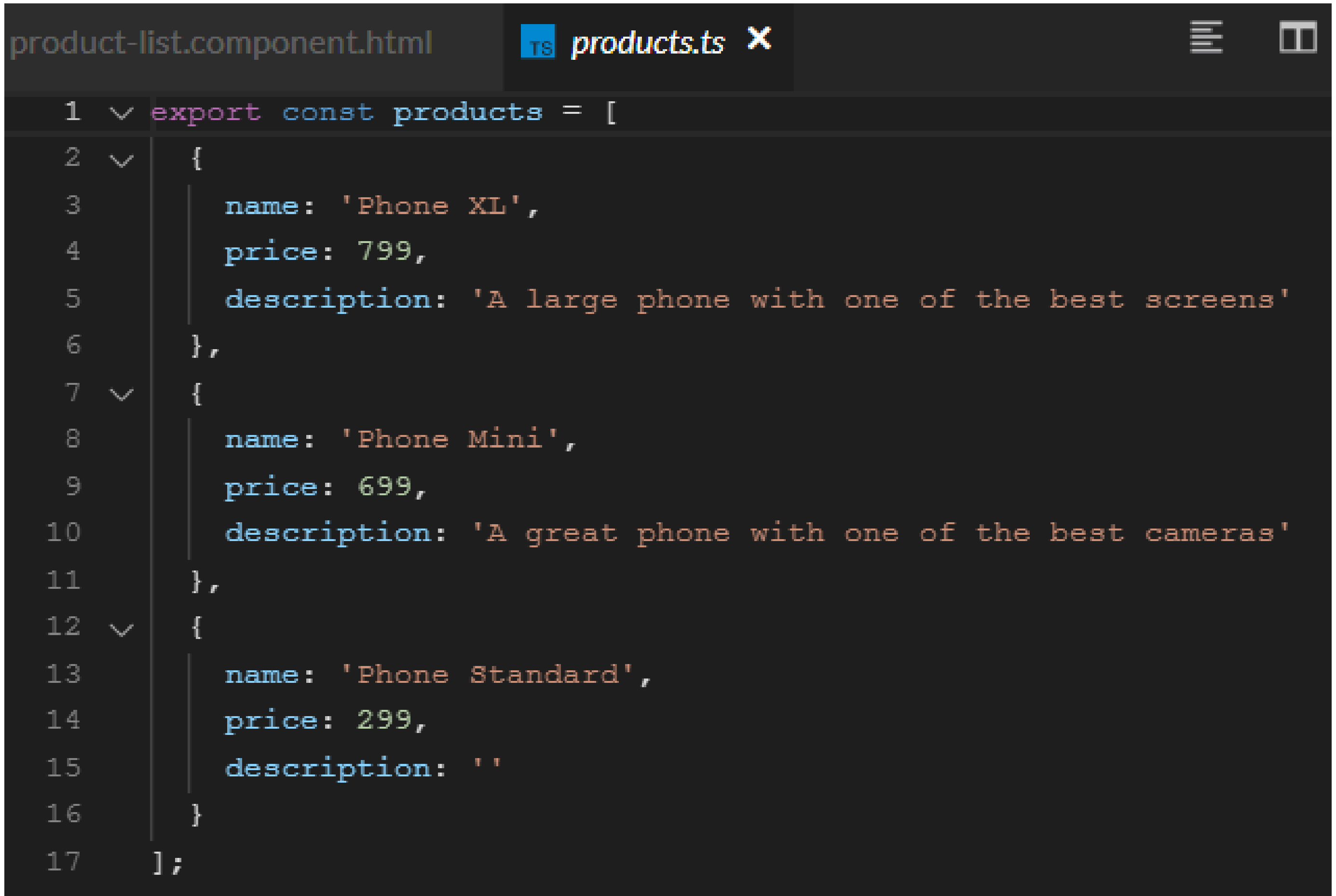
- *ngFor is a "structural directive".
- Structural directives shape or reshape the DOM's structure, typically by
  - adding, removing, and manipulating the elements to which they are attached.
- Any directive with an asterisk, *, is a structural directive.
- We use *ngFor by adding it as an attribute of the html element that you want to repeat, such as <p> element.

Looping with *ngFor directive

- ***ngFor="" takes an expression that consist of two parts:**
  - the array to loop over and
  - the element item that will contain the value during the iteration.
- **The syntax of the expression is "let item of items" where**
  - items is the array you will loop over and
  - the item is the current element in that iteration.

Looping with *ngFor directive

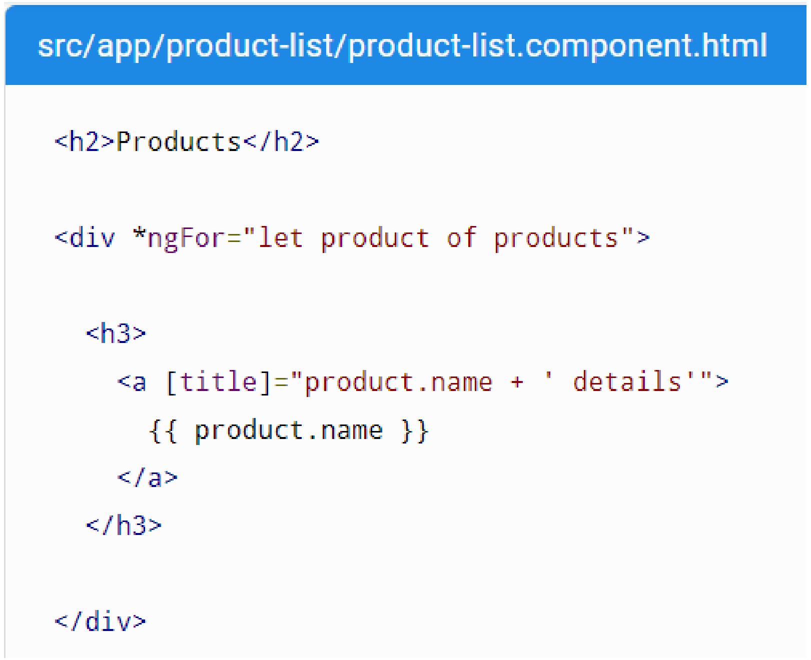
- Consider **products.ts** that defines an **array of products**



```
export const products = [
  {
    name: 'Phone XL',
    price: 799,
    description: 'A large phone with one of the best screens'
  },
  {
    name: 'Phone Mini',
    price: 699,
    description: 'A great phone with one of the best cameras'
  },
  {
    name: 'Phone Standard',
    price: 299,
    description: ''
  }
];
```

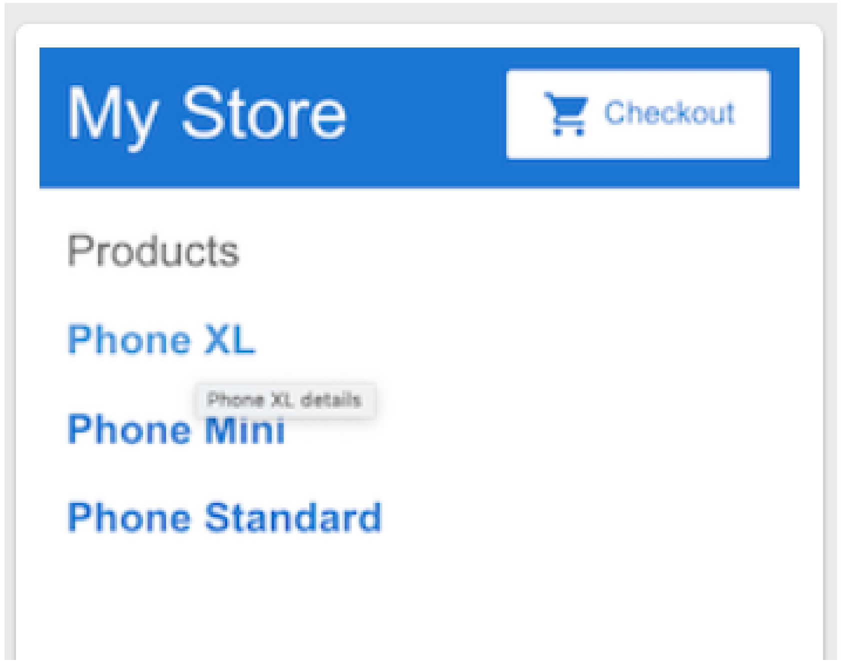
Looping with *ngFor directive

- Consider products.ts that defines an array of products



src/app/product-list/product-list.component.html

```
<h2>Products</h2>

<div *ngFor="let product of products">

  <h3>
    <a [title]="product.name + ' details'">
      {{ product.name }}
    </a>
  </h3>

</div>
```

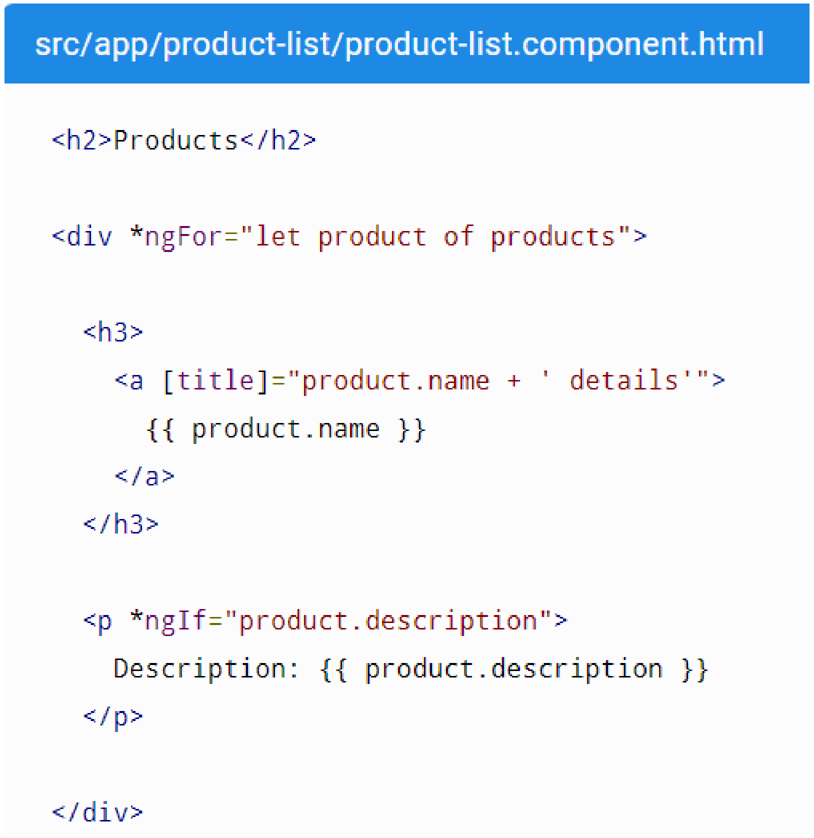


Using *ngIf directive

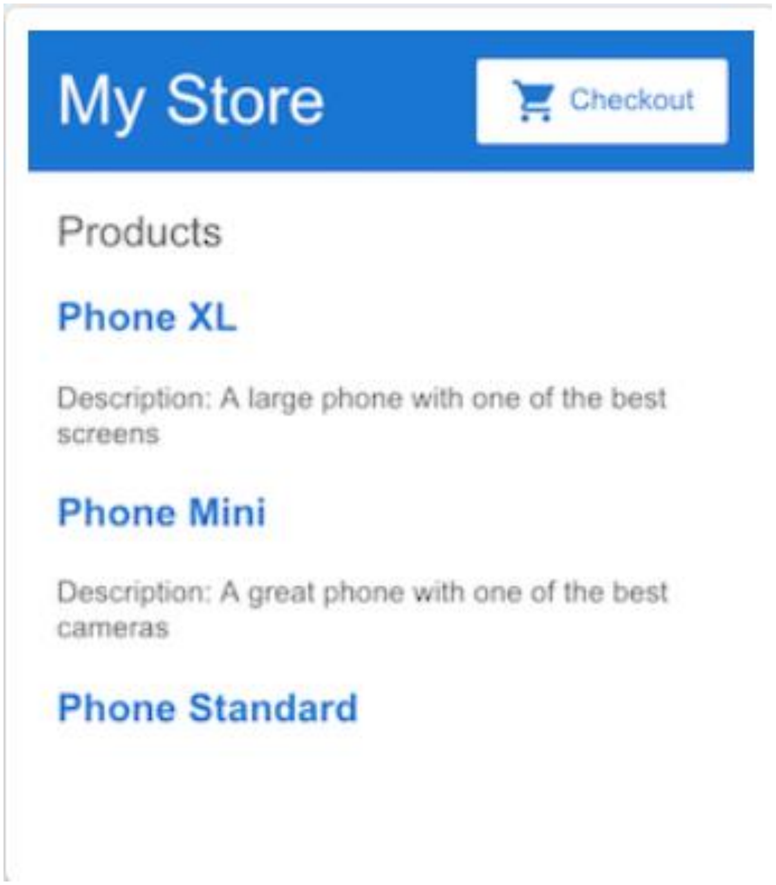
- ***ngIf correspond to if block in programming languages.**
- **It evaluates a condition and if the condition is true only then the template bound to that ngIf is displayed.**
  - In fact, ngIf removes the element from the DOM if the corresponding expression evaluates to false.
- **For example, if product.description is empty, then the inner <p> should not be rendered**
  - See an example on the next page

Using *ngIf directive

- **Phone Standard does not show description as it's missing in the component**


src/app/product-list/product-list.component.html

```
<h2>Products</h2>

<div *ngFor="let product of products">

  <h3>
    <a [title]="product.name + ' details'">
      {{ product.name }}
    </a>
  </h3>

  <p *ngIf="product.description">
    Description: {{ product.description }}
  </p>

</div>
```



Products

Phone XL

Description: A large phone with one of the best screens

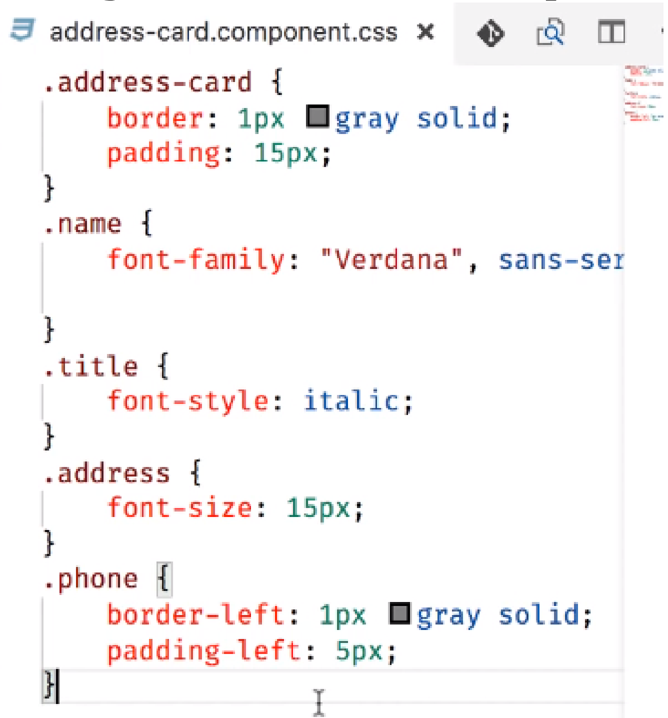
Phone Mini

Description: A great phone with one of the best cameras

Phone Standard

Styles for Angular Components

- Styles of a component for all html tags/elements used in xxxx-component.html file can be specified inside xxxx.component.css file.
- Styles specified in xxxx.component.css file applies only to the component it belongs to.



address-card.component.css ×

```
.address-card {
    border: 1px ■gray solid;
    padding: 15px;
}
.name {
    font-family: "Verdana", sans-ser

}
.title {
    font-style: italic;
}
.address {
    font-size: 15px;
}
.phone {
    border-left: 1px ■gray solid;
    padding-left: 5px;
}
```

Styles for Angular Components

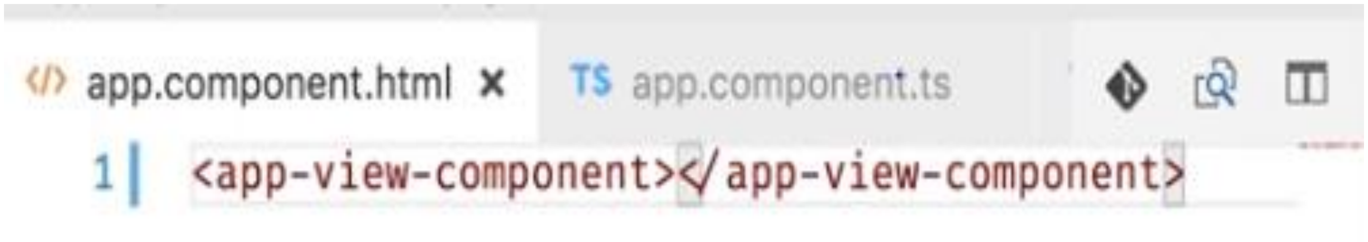
- If you really want a global style that could be applied to all components, then those styles can specified inside *styles.css* file which is at the root of your project inside src folder of the, **e.g., hello-world**

Creating and using multiple modules

- **To create a module and be able to use its component outside this module, we need to do the following:**
  - Module that needs this new module needs to import it
  - Whatever module the component is declared in has to export the component
- **That is, what needs to be imported is the module itself, e.g., AppModule imports ViewModule.**
- **And what is being exported is the component, that is the module that contains the component, i.e., ViewModule needs to export the component.**

Creating and using multiple modules

- This allows the component declared in a module to be usable in another module.
- This will allow the app-component.html which is part of AppModule to be able use <app-view-component> tag, which is part of ViewModule.



```
<app-view-component></app-view-component>
```

Creating a Service

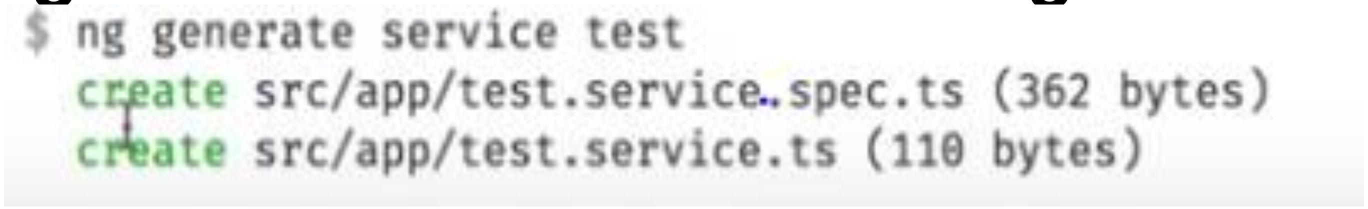
- Remember a component is something that you create to render some functionality in a portion of the user's view,
  - so you write view and write backend functionality that goes with it, and thus you create reusable components.
- However, not all reusable elements are actually views.
- You could create some reusable elements that are just some functionality in terms of a service or a method that needs to be used in multiple places
  - but don't come with any view attached.

Creating a Service

- So to create those kind of business services that don’t have views, you can create something called Services in Angular.
- Services are also classes like components are; and contains methods that contain functionality that you can reuse across multiple different components.

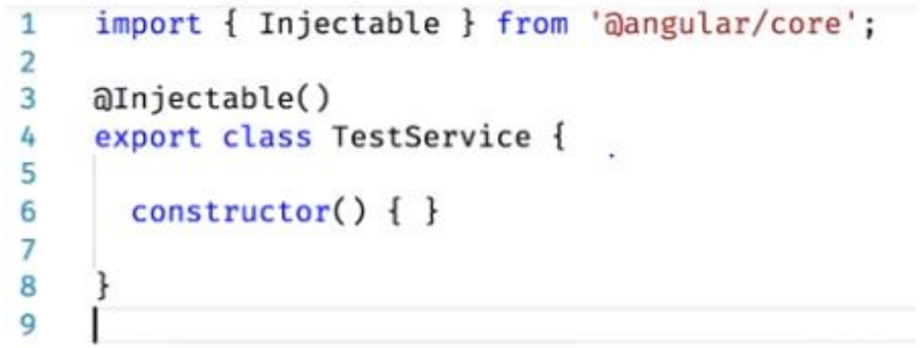
Creating a Service

- A **service** called '**test**' can be created using Angular **CLI** with the following **command**



```
$ ng generate service test
  create src/app/test.service.spec.ts (362 bytes)
  create src/app/test.service.ts (110 bytes)
```

- **test.service.ts** files essentially contains a **class** called **TestService** decorated with an annotation **@Injectible(),** which tells the Angular that this **class is a service.**



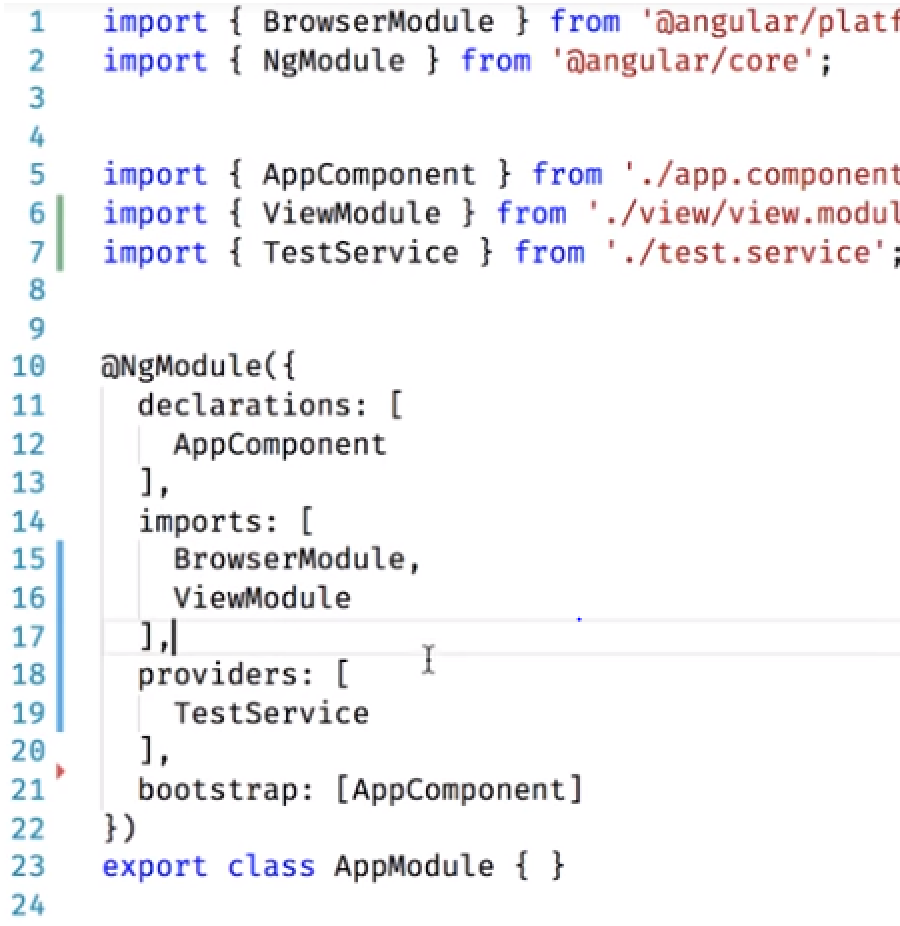
```
import { Injectable } from '@angular/core';

@Injectable()
export class TestService {

  constructor() { }

}

```

Creating a Service

- **Services that you declare in your module needs to be listed in the providers: section of @NgModule, which is an annotation for AppModule class in src/app/app.module.ts**
- **Please note that**
  - declarations: section contain all the components that are part of the module,
  - providers: contain all services that are part of the module,
  - imports: contain all other modules that this module depends on.
- **In order to add the test service in the AppModule, add TestService class into the providers: array**
  - See an example on next page

Creating a Service

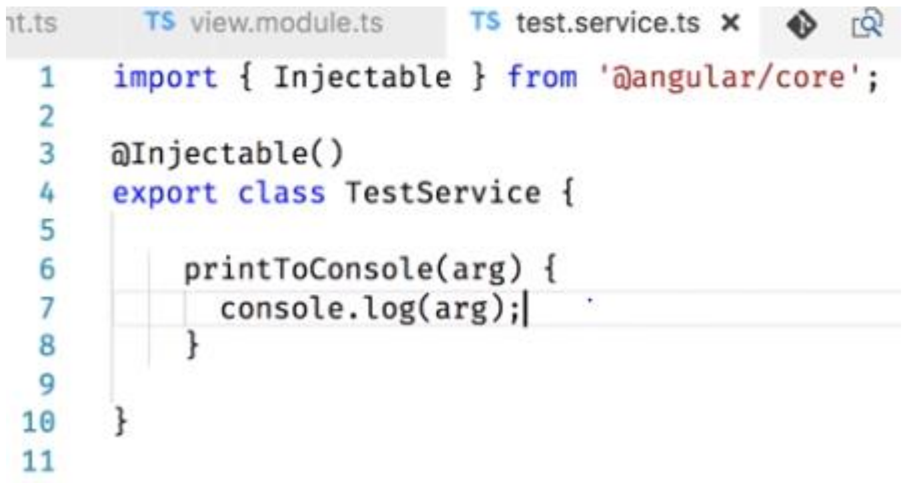
- To add test service in the AppModule in app.module.ts, **add TestService** class **into** the **providers:** array of @NgModule
- Now the TestService is available for other components to use.



```
import { BrowserModule } from '@angular/platf
import { NgModule } from '@angular/core';


import { AppComponent } from './app.component
import { ViewModule } from './view/view.modul
import { TestService } from './test.service';


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

Creating a Service

- Let's add a simple log method called **printToConsole()** to TestService



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

@Injectable()
export class TestService {

    printToConsole(arg) {
        console.log(arg);
    }

}

```

- Now **to call printToConsole()** method **of TestService** in my AppComponent defined **in app.component.ts** is done by **dependency injection**
  - as discussed in the next section

Dependency Injection

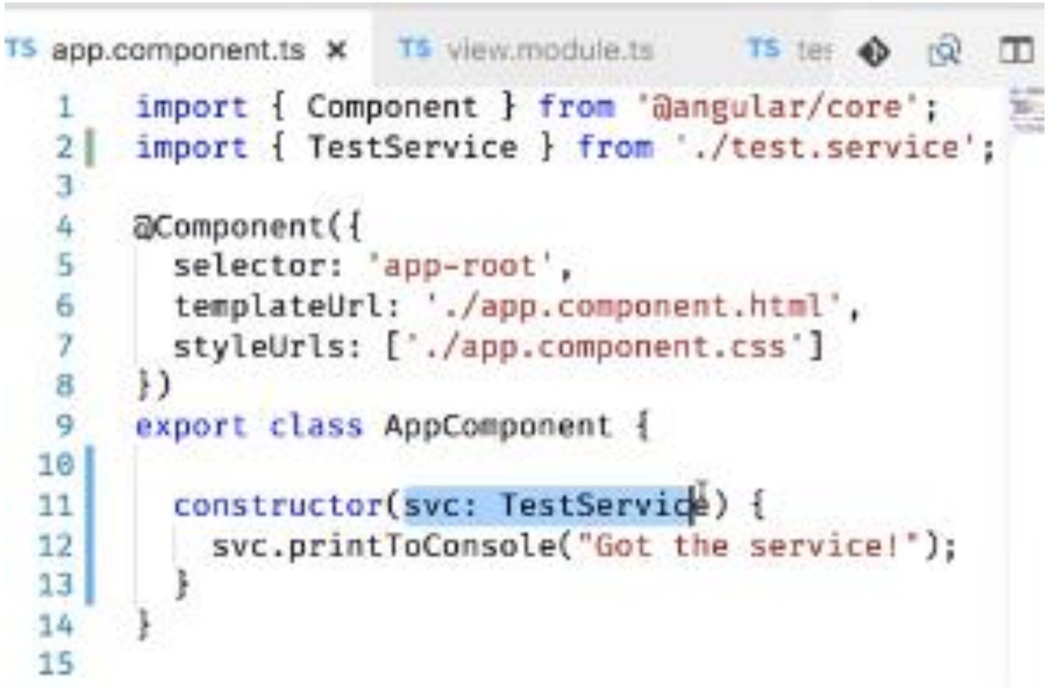
- **When you have a class A that is dependent on another class B,**
  - You don't have class A create the instance of class B,
  - Rather have the class A declare its dependency and have the dependency get injected by the framework.

Dependency Injection

- **Let AppComponent is dependent on TestService then using dependency injection, the runtime creates an instance of the TestService and then injects it into your component,**
  - You don’t have to explicitly create any instance of the service,
  - Just tell Angular that you need an instance of the service and Angular magically hands over that instance to your component – see an example on the next page

Dependency Injection

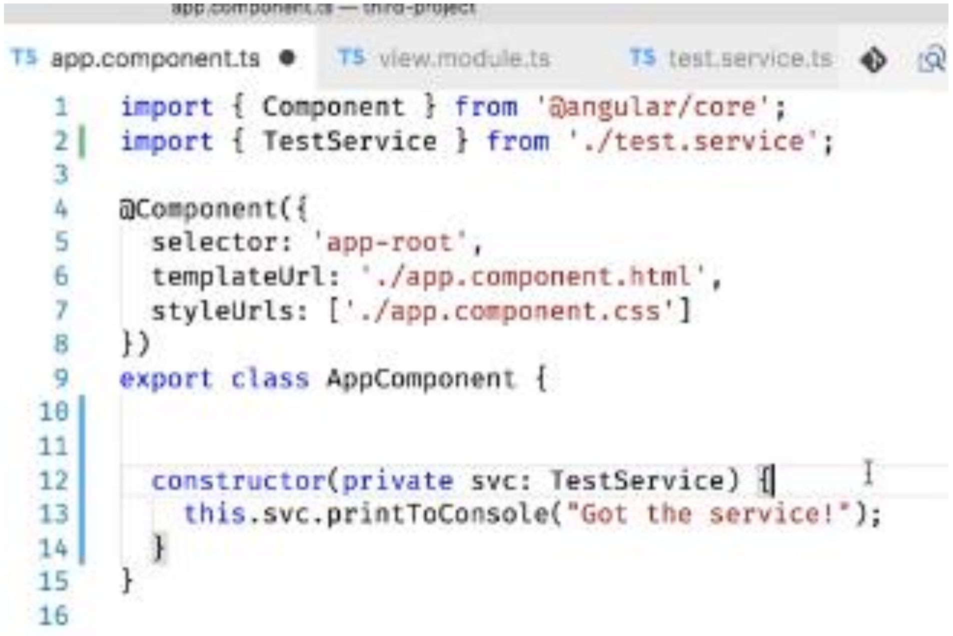
- In order to allow Angular to give us an instance of the service, the convention is to create an argument to the AppComponent constructor of the service type (i.e., the TestService) that you need



```
import { Component } from '@angular/core';
import { TestService } from './test.service';

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

  constructor(svc: TestService) {
    svc.printToConsole("Got the service!");
  }
}
```

Dependency Injection

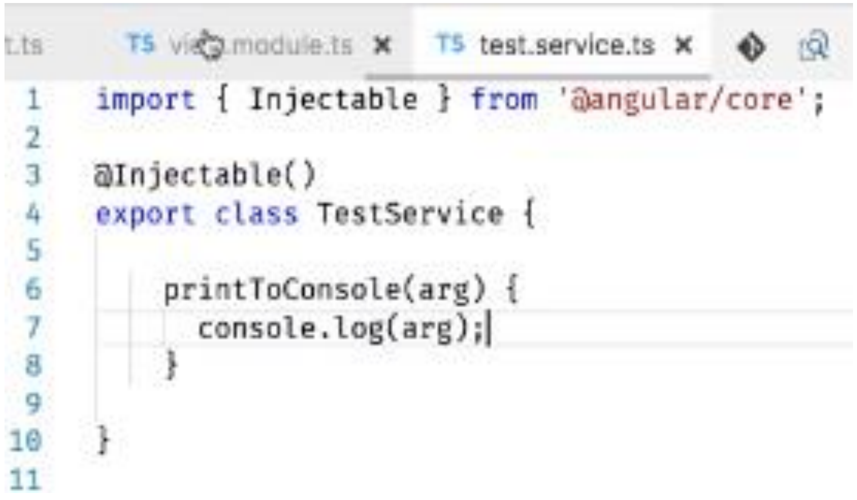
- **To make sure the svc is a member variable, you use private before svc as shown below.**
  - This is equivalent to really have private member variable and then Angular creating an instance of the TestService and then assigning its instance to the member variable svc so that it could be used outside constructor as well.



```
import { Component } from '@angular/core';
import { TestService } from './test.service';

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


  constructor(private svc: TestService) {
    this.svc.printToConsole("Got the service!");
  }
}

```

Dependency Injection

- **Having TestService as an argument of the constructor, Angular creates an instance of TestService and passes the instance to svc in the previous example.**
- **This is done after Angular verifies the TestService to make sure that the TestService is Injectible as shown below.**



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

@Injectable()
export class TestService {

    printToConsole(arg) {
        console.log(arg);
    }

}

```

Making REST calls with HttpClient

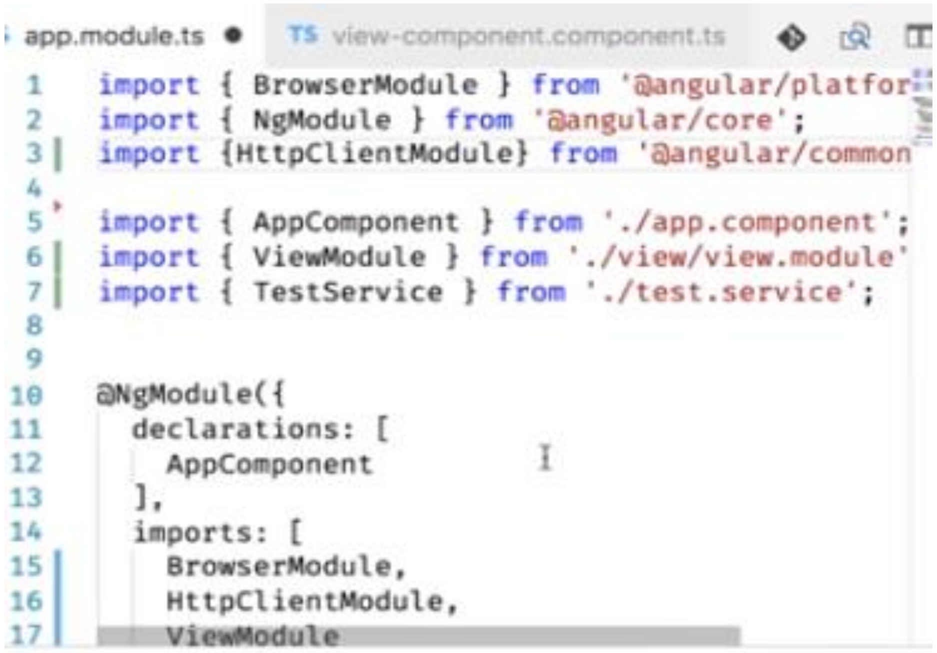
- **Angular provides a handy service called HttpClient out of the box that can be used to make REST API calls from Angular.**
- **To use that service, we need to import the module that the service comes with.**
  - The module contains service in the providers: section. (In this case, if's internal angular module.)
  - When you import that module in any one of your modules, that service gets added to the injection context, the global service context so that it's available for all your components to use.

Making REST calls with HttpClient

- Typically, HttpClient module is imported in the root module (i.e., AppModule inside app-module.ts).
- The name of the module is HttpClientModule, as imported in the AppModule next.

Making REST calls with HttpClient

- **The name of the module is HttpClientModule, as imported in the AppModule in app.module.ts next.**

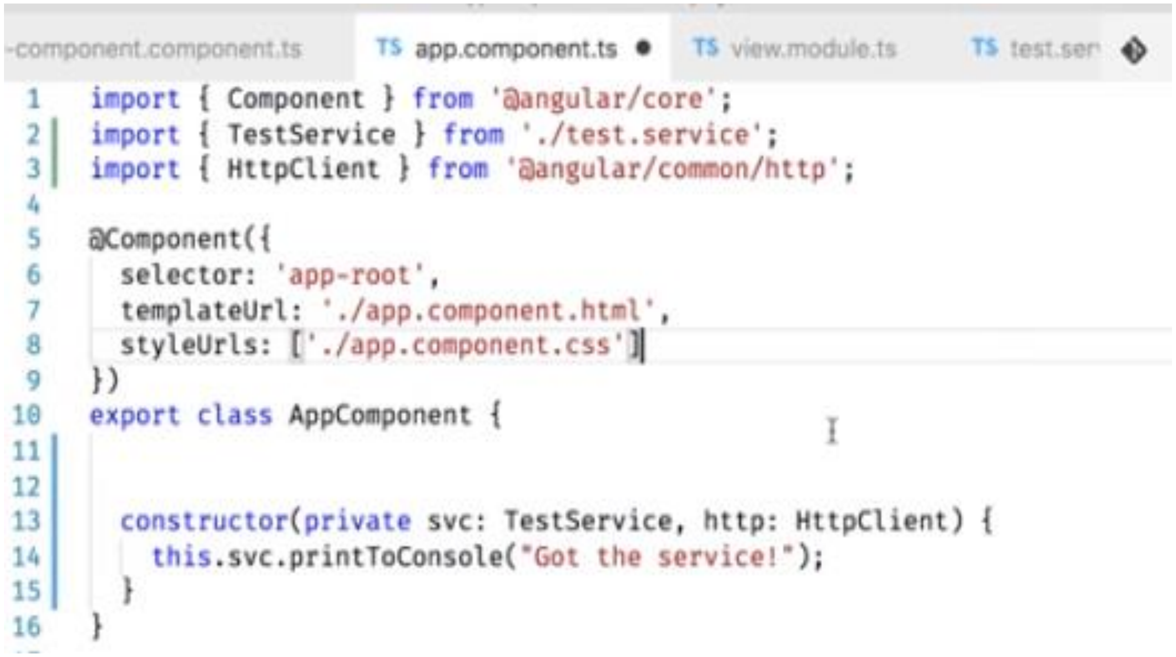


```
app.module.ts ●    TS view-component.component.ts

import { BrowserModule } from '@angular/platfor
import { NgModule } from '@angular/core';
import {HttpClientModule} from '@angular/common

import { AppComponent } from './app.component';
import { ViewModule } from './view/view.module'
import { TestService } from './test.service';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    ViewModule
```

- **Now, the HttpClientModule is part of your application.**
- **All the services/providers in HttpClientModule are now part of injection context and can be used wherever needed.**

Making REST calls with HttpClient

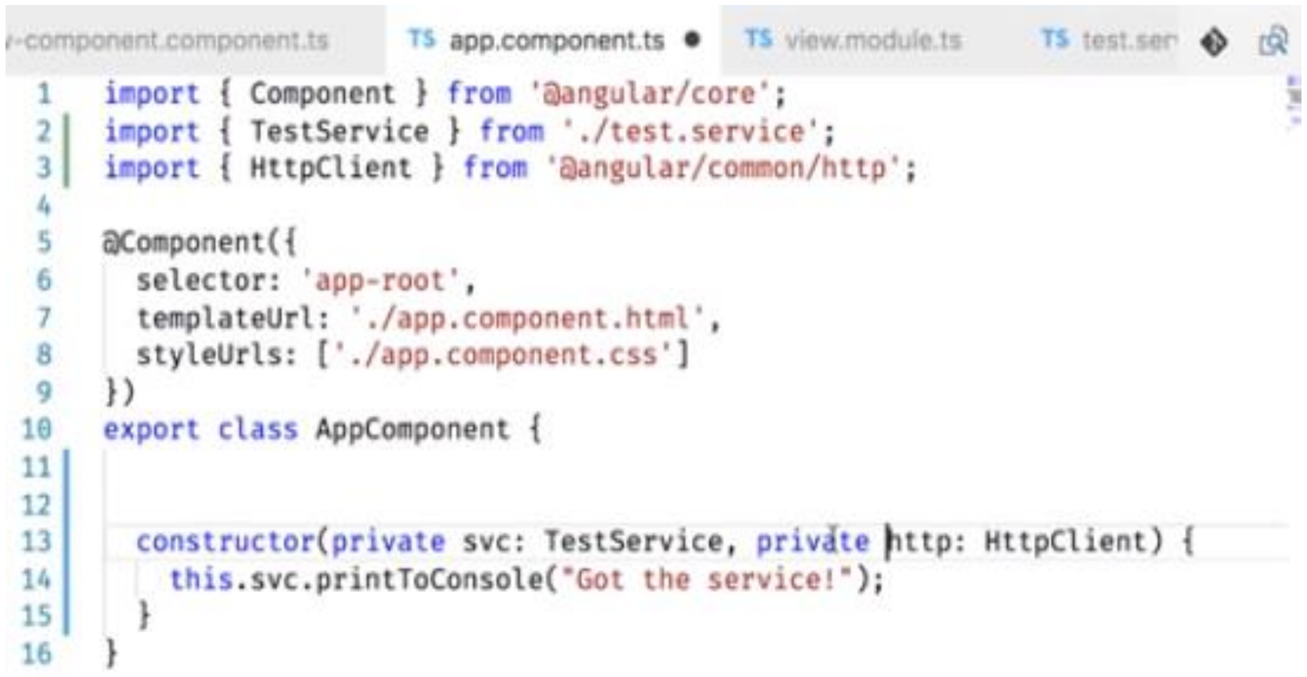
- **Let's inject the service called HttpClient (which allows to make http call) in the constructor of AppComponent**
  - (inside app-component.ts) by adding an argument http: followed by type HttpClient



```
import { Component } from '@angular/core';
import { TestService } from './test.service';
import { HttpClient } from '@angular/common/http';

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


  constructor(private svc: TestService, http: HttpClient) {
    this.svc.printToConsole("Got the service!");
  }
}
```

Making REST calls with HttpClient

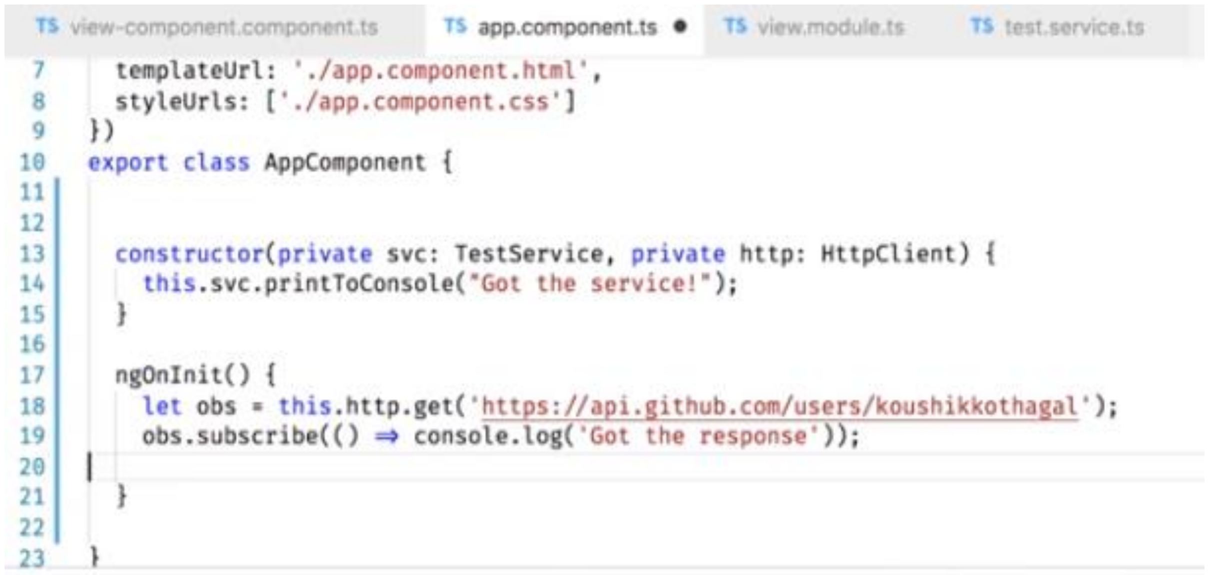
- **To make the http as a member variable in the above code, mark it <u>private</u> in the constructor argument as shown below.**



```
import { Component } from '@angular/core';
import { TestService } from './test.service';
import { HttpClient } from '@angular/common/http';

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


  constructor(private svc: TestService, private http: HttpClient) {
    this.svc.printToConsole("Got the service!");
  }
}
```

Making REST calls with HttpClient

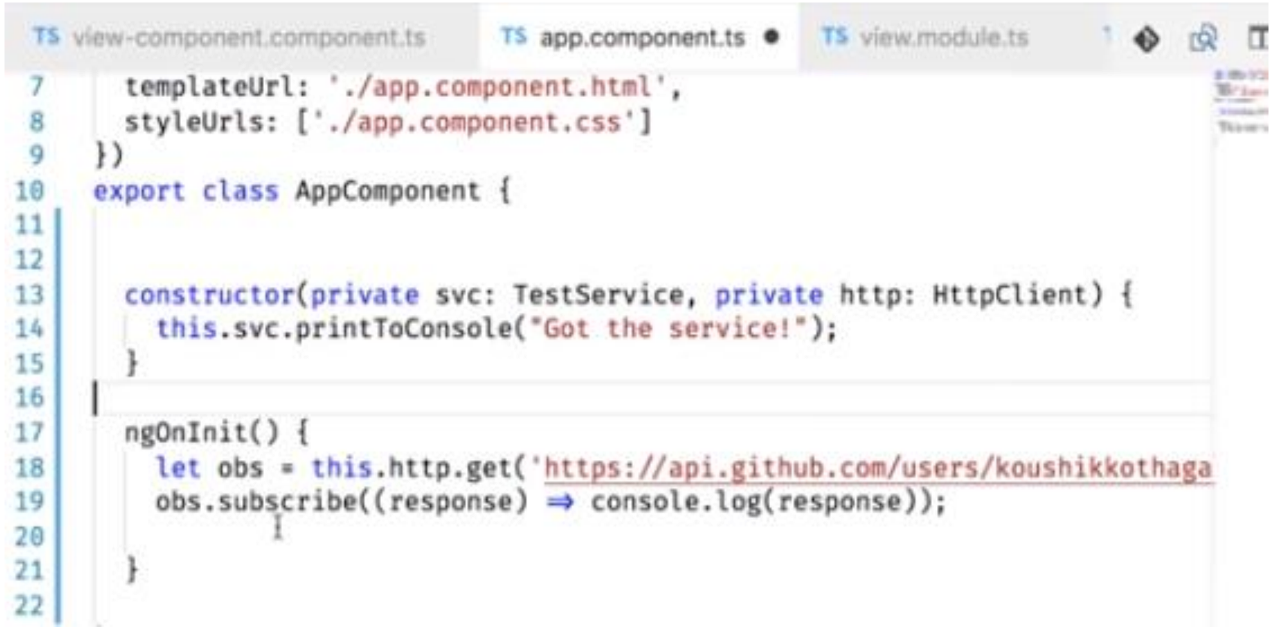
- **Let's make API call (e.g., get request) in the ngOnInit() method, as shown below.**
  - Please note that HttpClient get is an asynchronous call and returns an object (asynchronously), which, in the angular world, is called observable obs
  - You can pass observable to a function that you want to execute when the asynchronous operation completes.
  - You do this by calling subscribe method and giving it to your function



```
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {


  constructor(private svc: TestService, private http: HttpClient) {
    this.svc.printToConsole("Got the service!");
  }

  ngOnInit() {
    let obs = this.http.get('https://api.github.com/users/koushikkothagal');
    obs.subscribe(() => console.log('Got the response'));

  }

}
```

Making REST calls with HttpClient

- **To figure out what the response is, do the following.**
  - As observable calls a function (e.g., console.log() when job is done, this function can ask the observable to pass the API response to it when done fetching the API response.
  - Done by passing an argument to the function (line 18, 19)



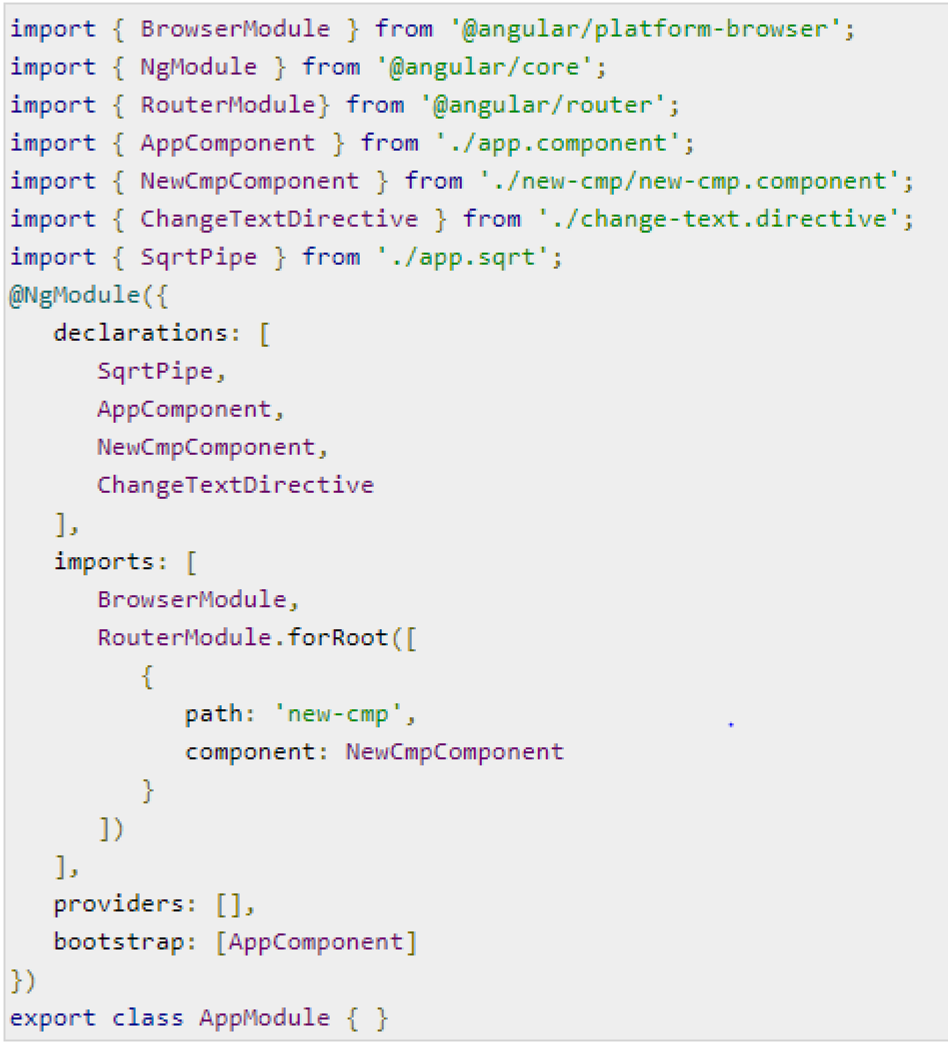
```
7      templateUrl: './app.component.html',
8      styleUrls: ['./app.component.css']
9    })
10   export class AppComponent {
11
12
13     constructor(private svc: TestService, private http: HttpClient) {
14       this.svc.printToConsole("Got the service!");
15     }
16
17     ngOnInit() {
18       let obs = this.http.get('https://api.github.com/users/koushikkothaga
19       obs.subscribe((response) => console.log(response));
20
21     }
22
```

Angular Routes

- **Routing basically means navigating between pages.**
  - It helps your application to become a Single Page Application (SPA)
- **Here the pages that we are referring to will be in the form of components.**
  - It redirects the user to another component without reload the page or call the back end.

Angular Routes

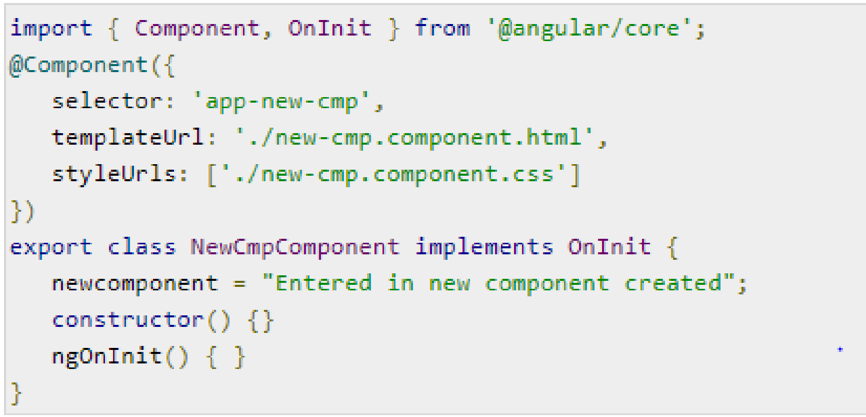
- **In the main parent component app.module.ts, we import the** RouterModule from angular/router and include in the imports:
- RouterModule refers to the **forRoot** which takes an input as an array, which in turn has the object of the path and the component.
  - Path is the name of the route and
  - component is the name of the class, i.e., the component created.



```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { RouterModule} from '@angular/router';
import { AppComponent } from './app.component';
import { NewCmpComponent } from './new-cmp/new-cmp.component';
import { ChangeTextDirective } from './change-text.directive';
import { SqrtPipe } from './app.sqrt';
@NgModule({
   declarations: [
      SqrtPipe,
      AppComponent,
      NewCmpComponent,
      ChangeTextDirective
   ],
   imports: [
      BrowserModule,
      RouterModule.forRoot([
         {
            path: 'new-cmp',
            component: NewCmpComponent
         }
      ])
   ],
   providers: [],
   bootstrap: [AppComponent]
})
export class AppModule { }
```

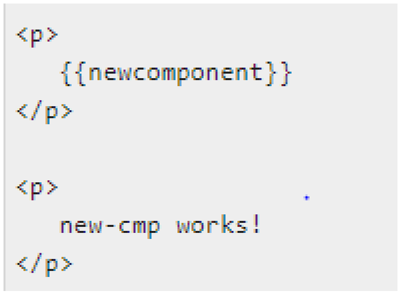
Angular Routes

- **new-cmp.component.ts** defines a class named NewCmpComponent,
  - which is mentioned in the imports of the main module app.module.ts



```
import { Component, OnInit } from '@angular/core';
@Component({
   selector: 'app-new-cmp',
   templateUrl: './new-cmp.component.html',
   styleUrls: ['./new-cmp.component.css']
})
export class NewCmpComponent implements OnInit {
   newcomponent = "Entered in new component created";
   constructor() {}
   ngOnInit() { }
}
```

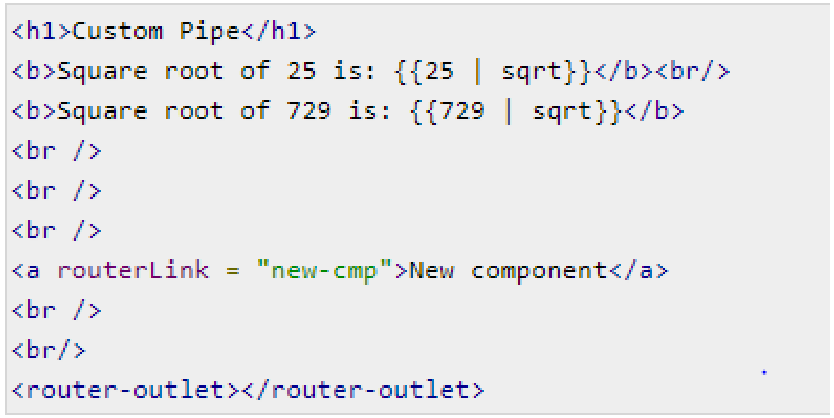
- **new-cmp.component.html**



```
<p>
   {{newcomponent}}
</p>

<p>
   new-cmp works!
</p>
```

Angular Routes

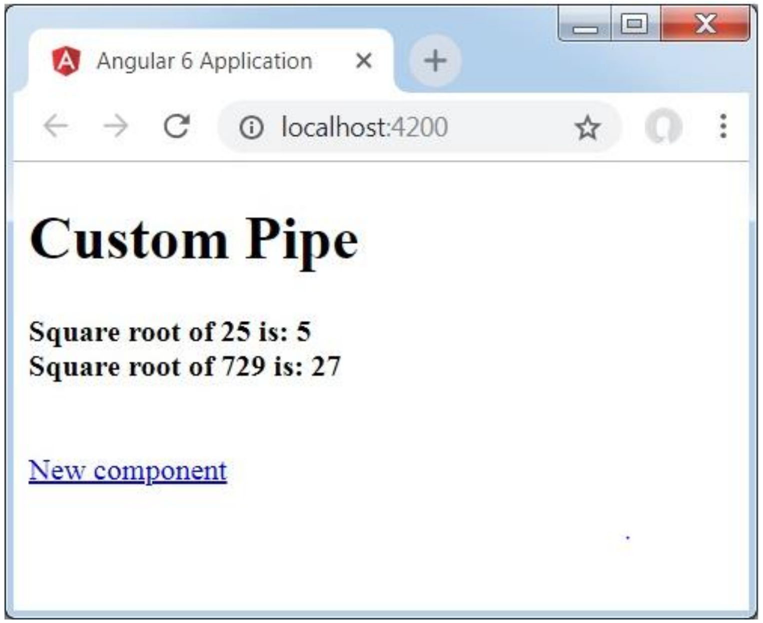
- To display the content from the **new-cmp.component.html** file whenever required or clicked from the main module, we need to add the router details in the **app.component.html** by adding the anchor link <a> tag that uses attribute routerLink and **"new-cmp“ its value**.
  - new-cmp is referred in app.module.ts as the path.
- When a user clicks N**ew component** link, the page displays the content. For this, we need the following tag - **<router-outlet></router-outlet>**.
  - The above tag ensures that the content in the new-cmp.component.html will be displayed on the page when a user clicks new component.



```
<h1>Custom Pipe</h1>
<b>Square root of 25 is: {{25 | sqrt}}</b><br/>
<b>Square root of 729 is: {{729 | sqrt}}</b>
<br />
<br />
<br />
<a routerLink = "new-cmp">New component</a>
<br />
<br/>
<router-outlet></router-outlet>
```

Angular Routes

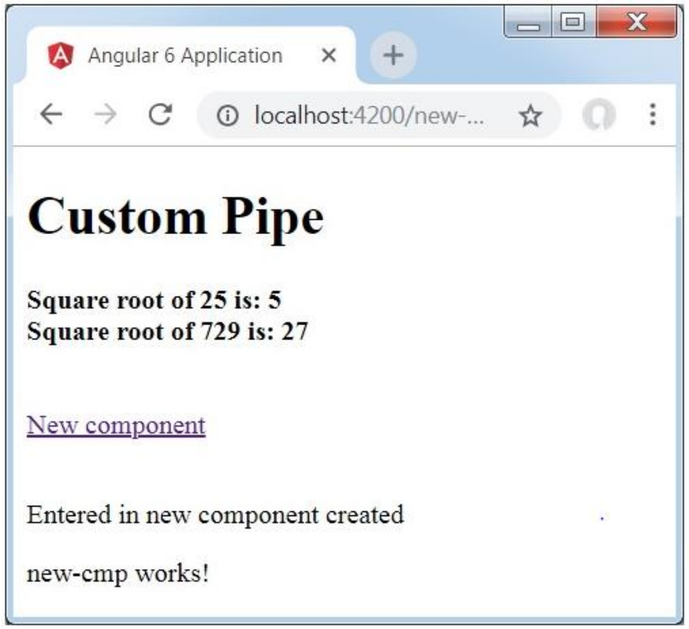
- When a user clicks **New component**, you see the following in the browser – the url contains **http://localhost:4200/new-cmp**.
  - Here, the new-cmp gets appended to the original url, which is the path given in the app.module.ts and the router-link in the app.component.html.
- When a user clicks **New component**, the contents are shown to the user without any reloading.
  - Only a particular piece of the site code will be reloaded when clicked.



Custom Pipe

Square root of 25 is: 5
Square root of 729 is: 27

New component



Custom Pipe

Square root of 25 is: 5
Square root of 729 is: 27

New component

Entered in new component created

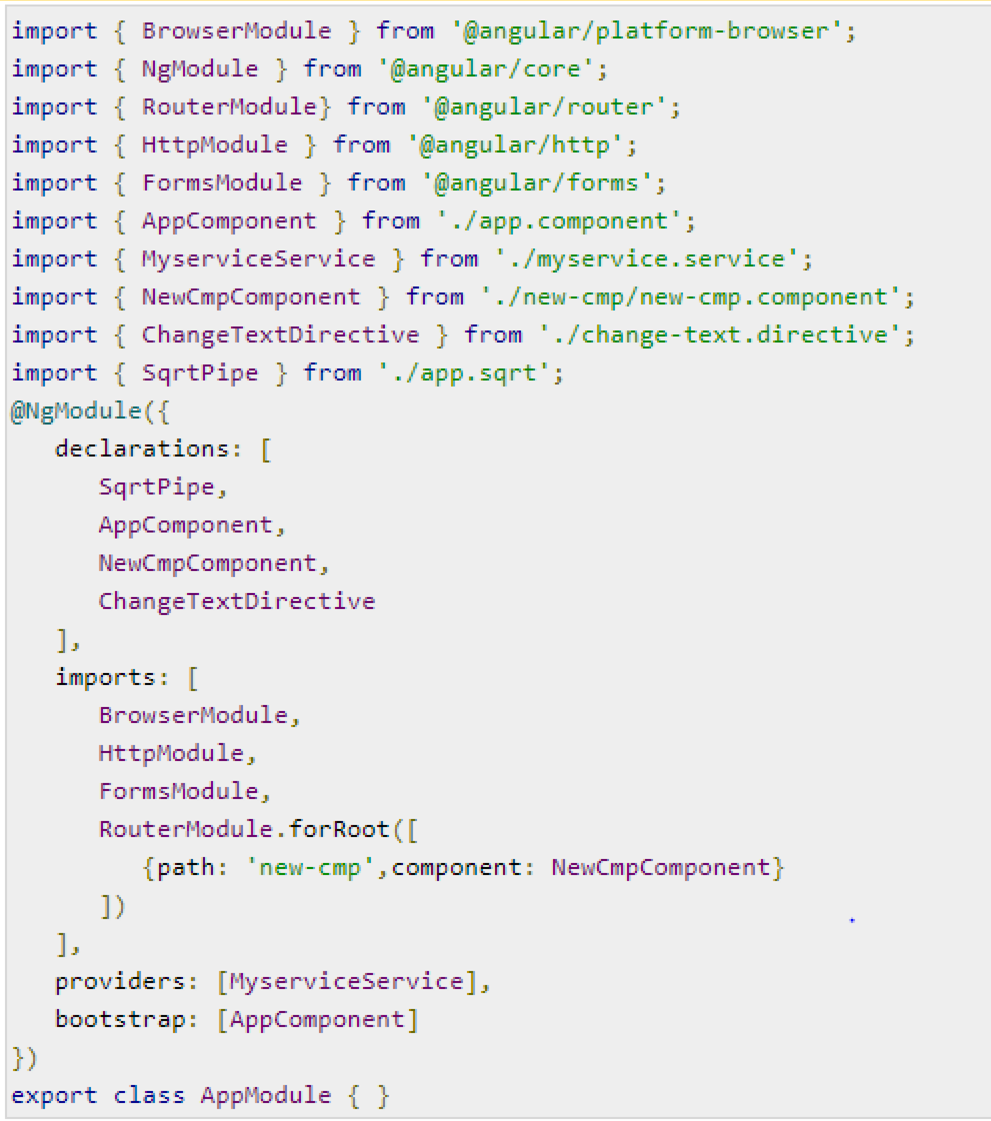
new-cmp works!

Forms

- There are two ways of working with forms: **Template driven form** and Model driven forms.
  - With a template driven form, most of the work is done in the template; and
  - With the model driven form, most of the work is done in the component class.
- **For now, we will focus on Template Driven Form**

Template Driven Forms

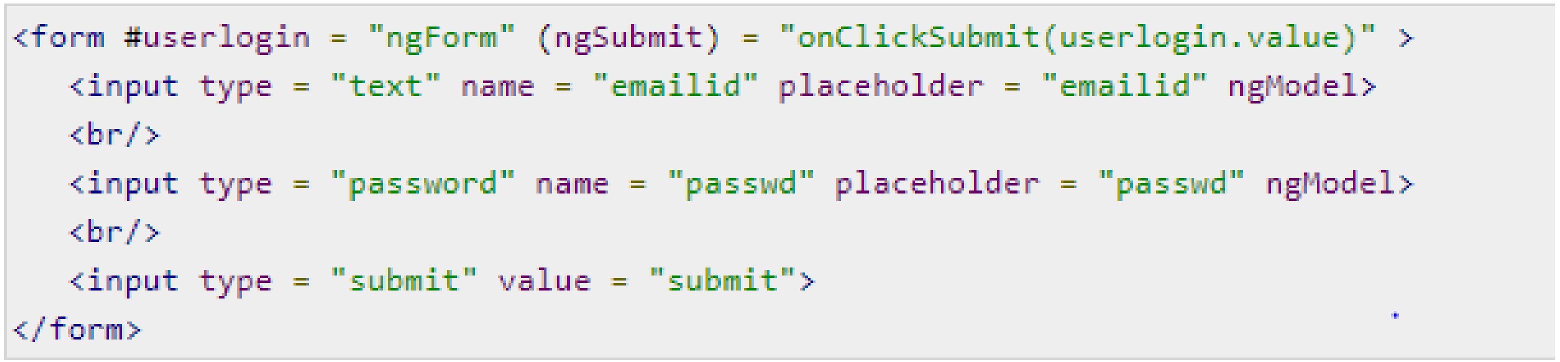
- Let's create a simple login form with email id, password fields and a submit button.
- Need to **import FormsModule** from @angular/core, and **add** it in the **imports array** of @NgModule of AppModule class in **app.module.ts**
  - as shown here



```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { RouterModule} from '@angular/router';
import { HttpModule } from '@angular/http';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { MyserviceService } from './myservice.service';
import { NewCmpComponent } from './new-cmp/new-cmp.component';
import { ChangeTextDirective } from './change-text.directive';
import { SqrtPipe } from './app.sqrt';
@NgModule({
   declarations: [
      SqrtPipe,
      AppComponent,
      NewCmpComponent,
      ChangeTextDirective
   ],
   imports: [
      BrowserModule,
      HttpModule,
      FormsModule,
      RouterModule.forRoot([
         {path: 'new-cmp',component: NewCmpComponent}
      ])
   ],
   providers: [MyserviceService],
   bootstrap: [AppComponent]
})
export class AppModule { }
```

Template Driven Forms

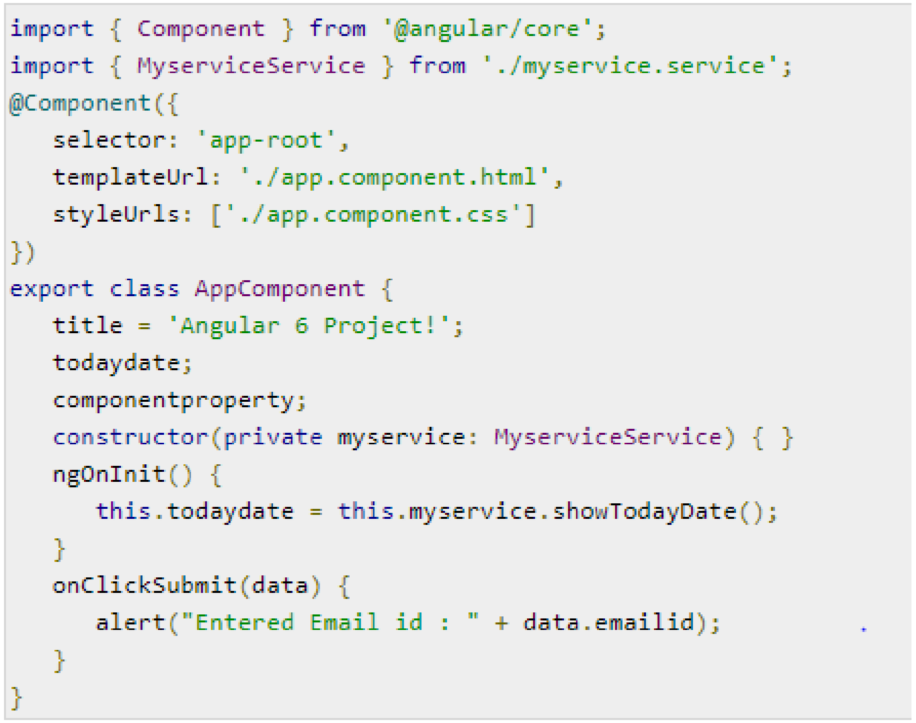
- The **app.component.html** file creates a simple form with input tags having email id, password and the submit button.
  - We have assigned type, name, and placeholder to it.
- In template driven forms, we need to create the **model form controls** by adding the **ngModel** directive and the **name** attribute.
  - Wherever we want Angular to access our data from forms, add ngModel to that tag as shown below. If we have to read the emailid and passwd, we need to add the ngModel across it.
- Also, assigned the **ngForm** to the **#userlogin**
  - The ngForm directive needs to be added to the form template.
  - **userlogin** here represents the form.
- We have also added function **onClickSubmit** that takes **userlogin.value** as an argument



```
<form #userlogin = "ngForm" (ngSubmit) = "onClickSubmit(userlogin.value)" >
   <input type = "text" name = "emailid" placeholder = "emailid" ngModel>
   <br/>
   <input type = "password" name = "passwd" placeholder = "passwd" ngModel>
   <br/>
   <input type = "submit" value = "submit">
</form>
```

Template Driven Forms

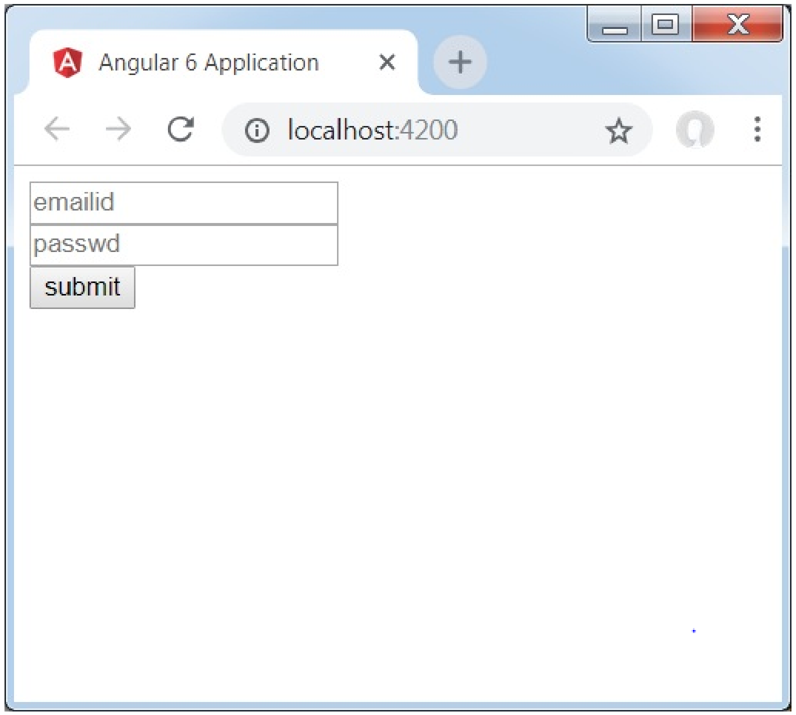
- In the **app.component.ts** file, we define the function **onClickSubmit()**, which fetches the values entered in the form.
  - When you click on the form submit button, the control will come to the above function.
  - If you get an error of an implicit type *any* for the function onClickSubmit() in the app.component.ts file, one solution is to update the onClickSubmit() header to be onClickSubmit(data: any)

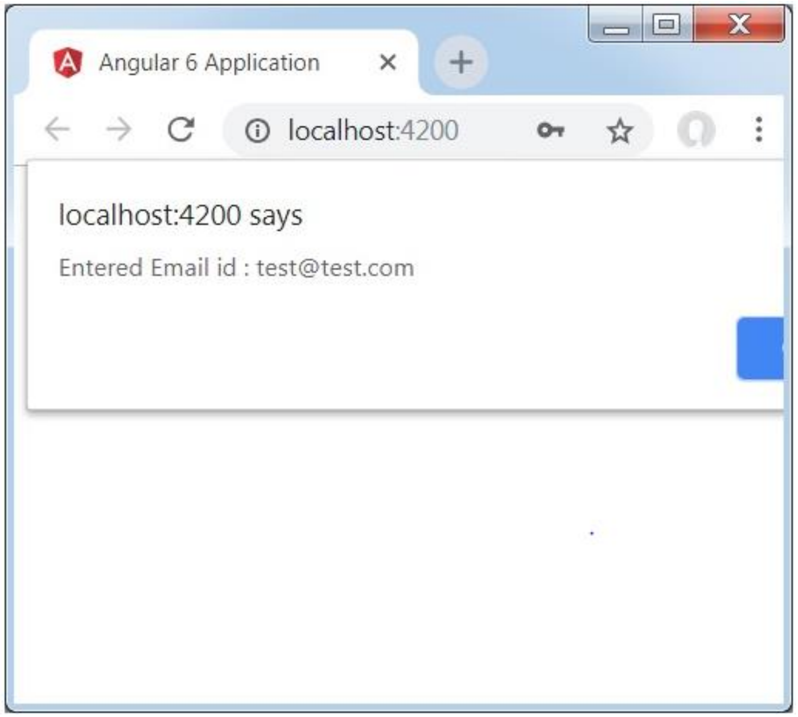


```
import { Component } from '@angular/core';
import { MyserviceService } from './myservice.service';
@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.css']
})
export class AppComponent {
   title = 'Angular 6 Project!';
   todaydate;
   componentproperty;
   constructor(private myservice: MyserviceService) { }
   ngOnInit() {
      this.todaydate = this.myservice.showTodayDate();
   }
   onClickSubmit(data) {
      alert("Entered Email id : " + data.emailid);
   }
}
```

Template Driven Forms

- The form looks like as shown below.
- After enter the data, e.g., email id, in it and pressing the submit function, shows the screen on right





localhost:4200 says

Entered Email id : test@test.com

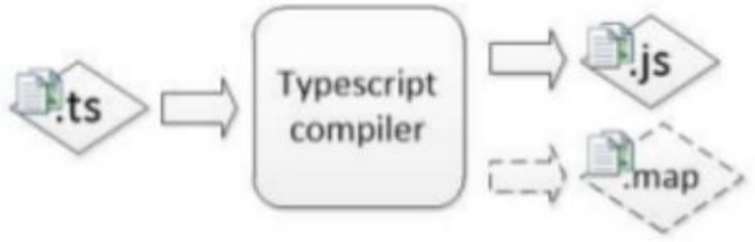
TypeScript Primer

Typescript

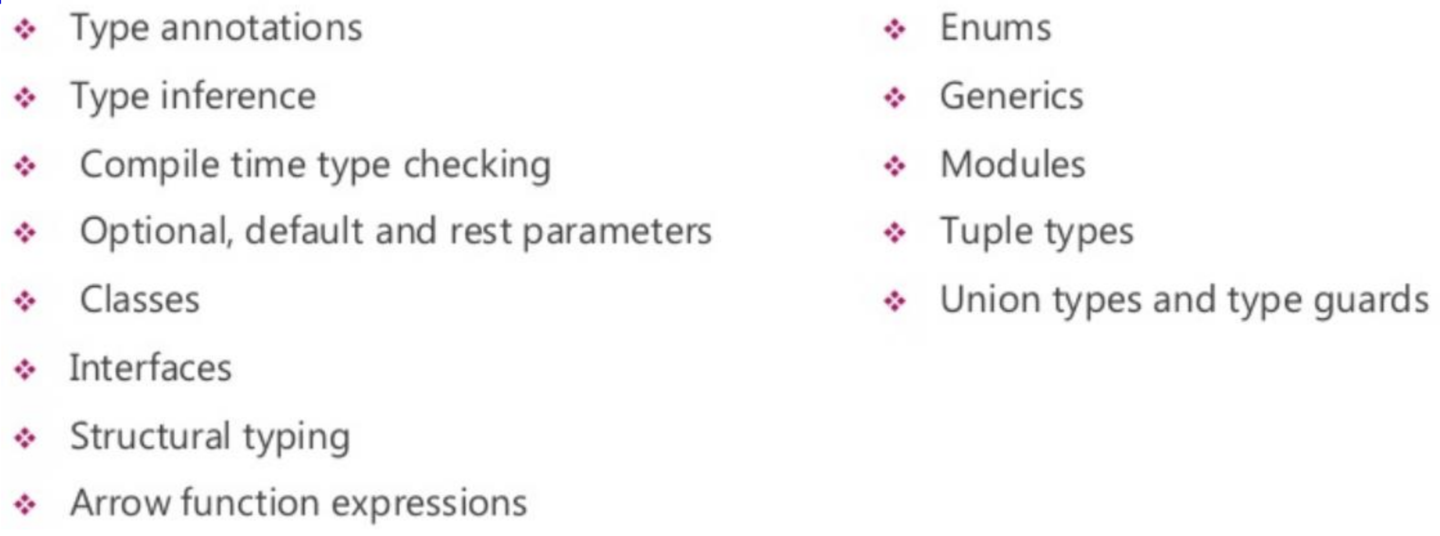
- **TypeScript is a superset of JavaScript**
  - Adds optional static typing to the language.
  - The TypeScript compiles to JavaScript
  - Developed by Microsoft
- **Anders Hejlsberg led efforts to develop TypeScript**
  - A lead architect of C#
  - Originally a Danish software engineer
- **Angular 2 and later versions use TypeScript**

Typescript

- **Typescript is written in .ts file**
- **The TypeScript compiles to JavaScript**



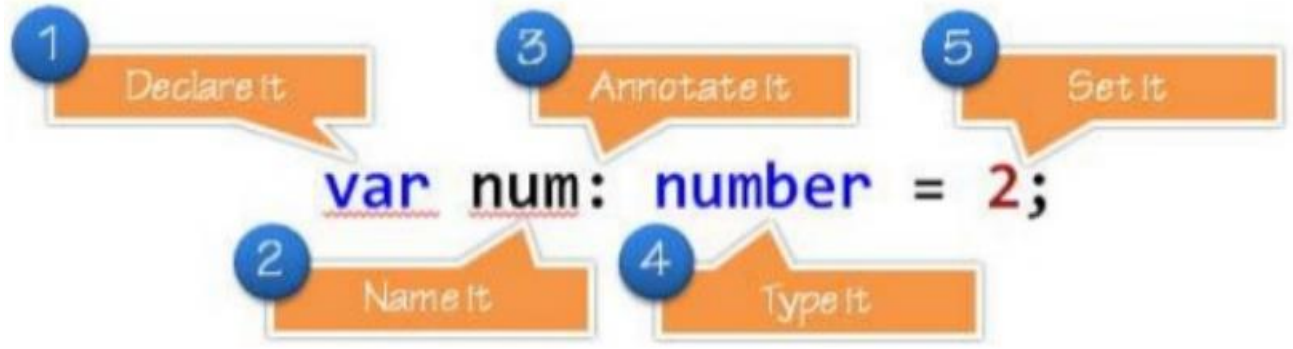
- **TypeScript Features**



- Type annotations
- Type inference
- Compile time type checking
- Optional, default and rest parameters
- Classes
- Interfaces
- Structural typing
- Arrow function expressions
- Enums
- Generics
- Modules
- Tuple types
- Union types and type guards

Typescript

- TypeScript Grammar



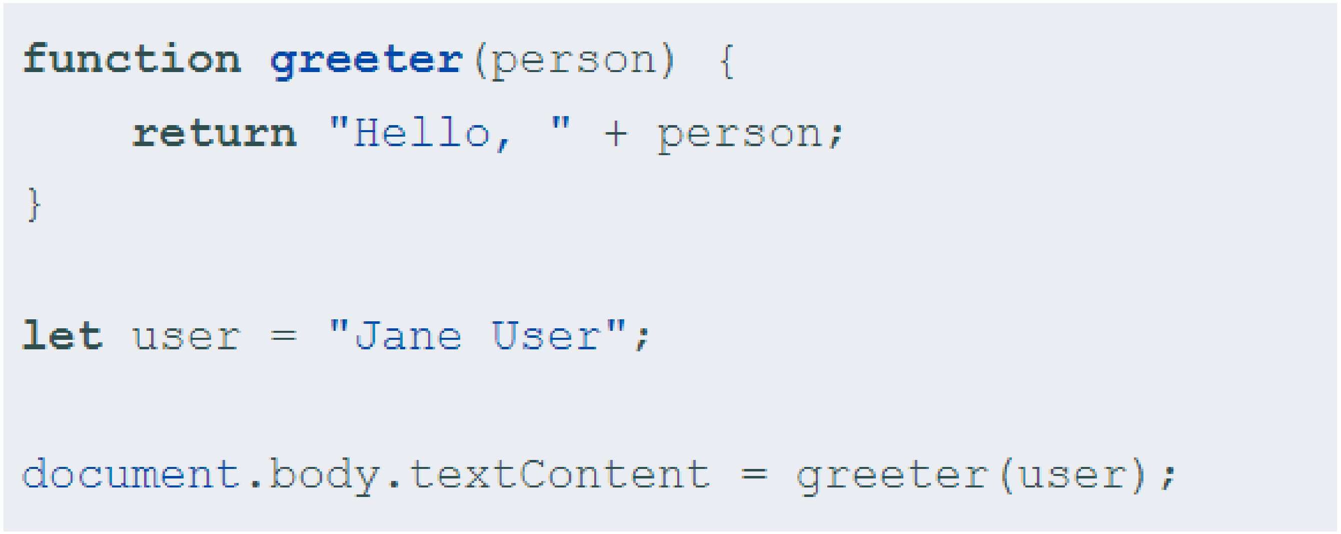
- **let** and **const** are two **new types** of variable declarations in JavaScript.
- **let** is similar to **var** in some respects
- **const** is an augmentation of **let** in that it **prevents re-assignment** to a variable

Typescript in five minutes

- **Reference:**
  - https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html
- **There are two main ways to get the TypeScript tools:**
  - Via npm (the Node.js package manager)
  - By installing TypeScript's Visual Studio plugins
- **Visual Studio includes TypeScript by default.**
- **For NPM users:**
  - npm install -g typescript

Building your first TypeScript file

- **In your editor, type the following JavaScript code i**n `greeter.ts`:



```
function greeter(person) {
    return "Hello, " + person;
}

let user = "Jane User";

document.body.textContent = greeter(user);
```

- We used a .ts extension, but this code is just JavaScript.

Type Annotations

- **We can take advantage of some of the new tools TypeScript offers.**
- **Let’s add a `: string` type annotation to the ‘person’ function argument**

```
function greeter(person: string) {
    return "Hello, " + person;
}

let user = "Jane User";

document.body.textContent = greeter(user);
```

- **Type annotations is used to record the intended contract of the function or variable.**

Type Annotations

- In this case, we intend the `greeter` function to be called with a single string parameter.
- Try changing the call to greeter to pass an array instead

```
function greeter(person: string) {
    return "Hello, " + person;
}

let user = [0, 1, 2];

document.body.textContent = greeter(user);
```

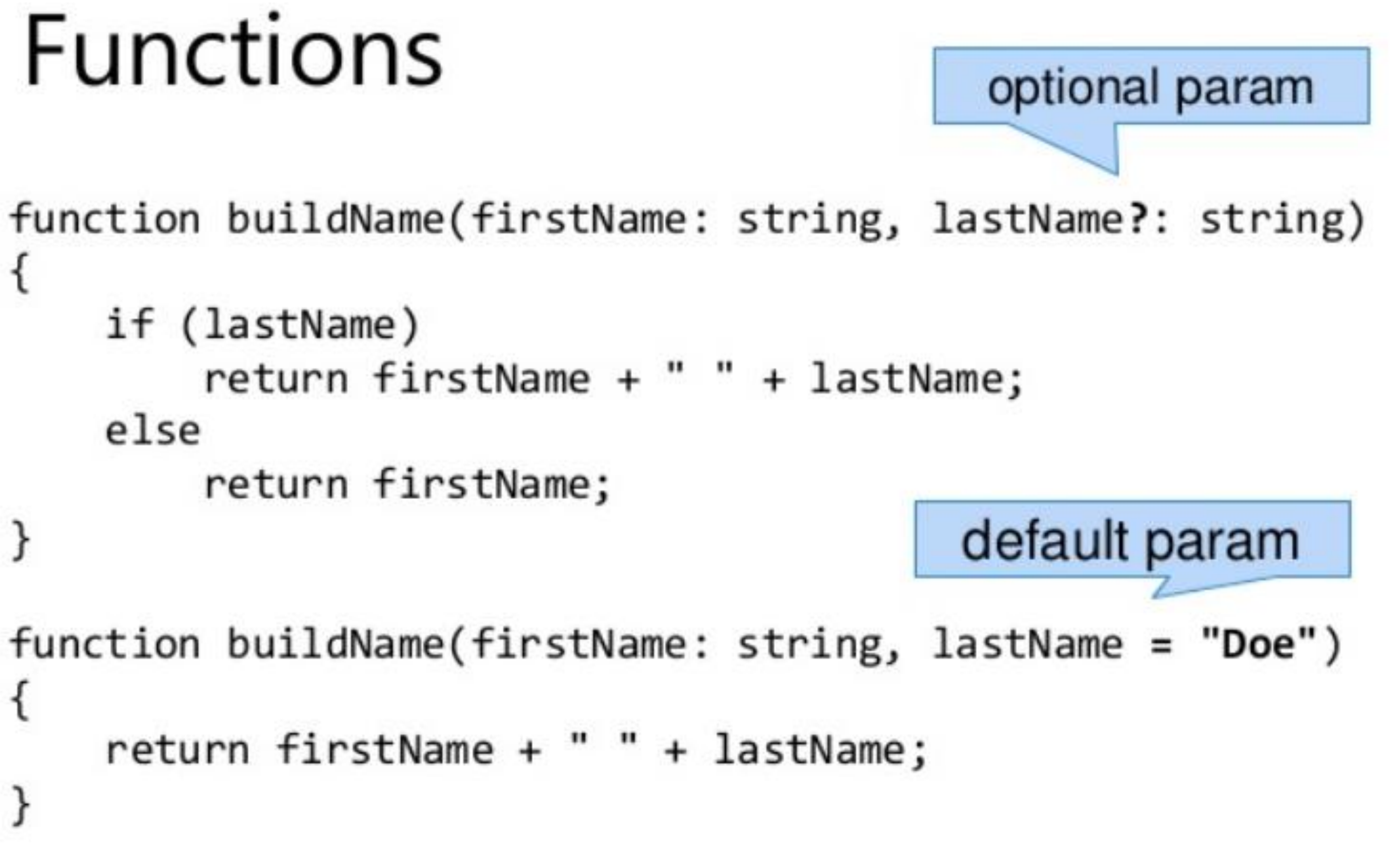
- Re-compiling, you'll now see an error:

```
error TS2345: Argument of type 'number[]' is not assignable to parameter of type 'strin
g'.
```

- Similarly, removing all the arguments to the greeter call gives error
  - that the function called with an unexpected number of parameters.

TypeScript Functions Params

- TypeScript **functions** allow **optional** and **default parameters**


Functions

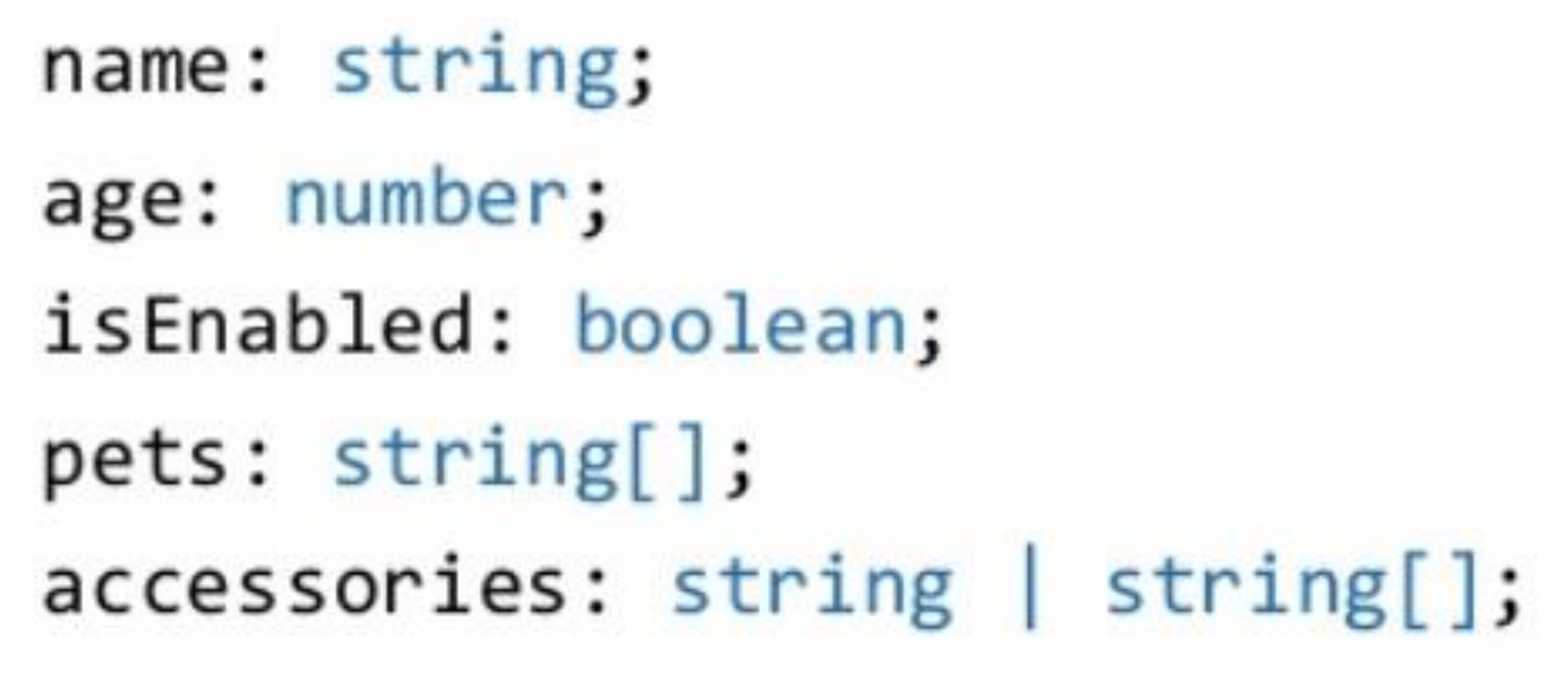
```
function buildName(firstName: string, lastName?: string)
{
    if (lastName)
        return firstName + " " + lastName;
    else
        return firstName;
}

function buildName(firstName: string, lastName = "Doe")
{
    return firstName + " " + lastName;
}
```

TypeScript Types

- **Built-In types**
  - string
  - number
  - boolean
  - Date
  - Array
  - any
- **Custom types**

TypeScript Types

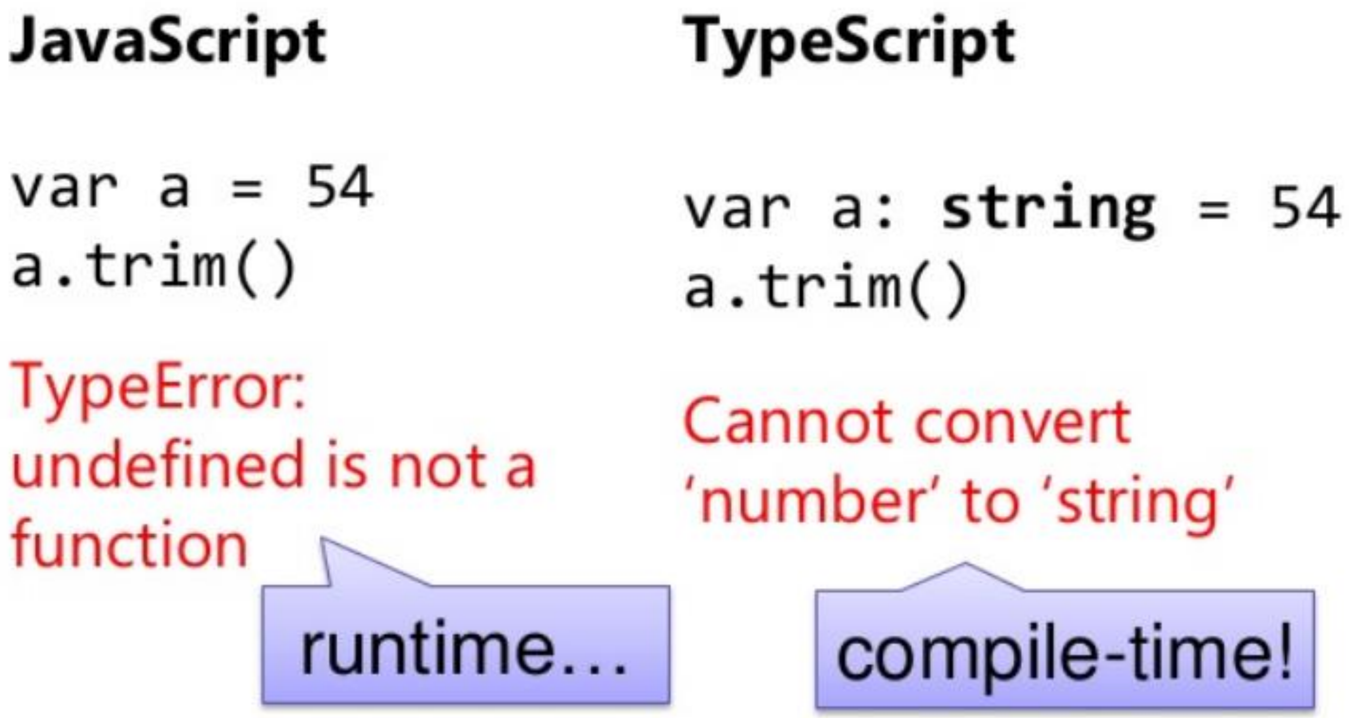
- **TypeScript Types Annotations**



```
name: string;
age: number;
isEnabled: boolean;
pets: string[];
accessories: string | string[];
```

TypeScript Types

- **TypeScript Types enforces compile time errors**



JavaScript

```
var a = 54
a.trim()
```

TypeError: undefined is not a function

runtime…

TypeScript

```
var a: string = 54
a.trim()
```

Cannot convert ‘number’ to ‘string’

compile-time!

TypeScript Interfaces

- **TypeScript interfaces provide a code contract**

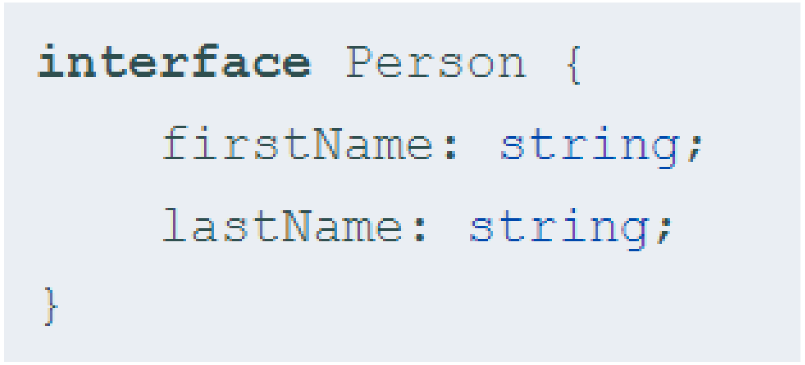
```
interface Person {
    firstName: string;
    lastName: string;
}
```

- **An example of a valid satisfied contract**

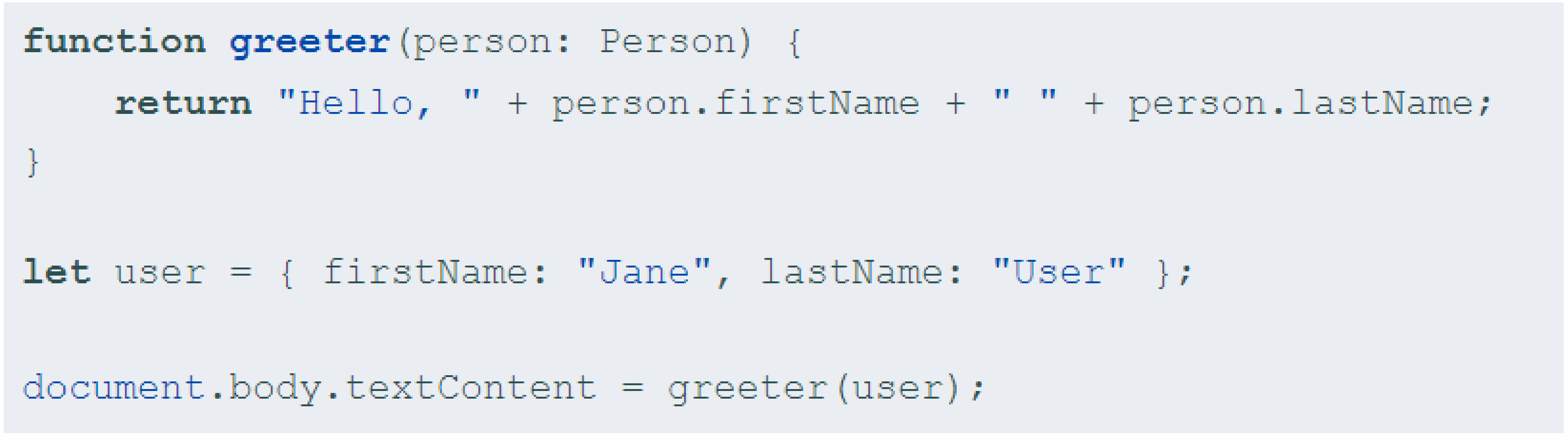
```
let user = { firstName: "Jane", lastName: "User" };
```

TypeScript Interfaces

- **An example of using the interface in function**



```
interface Person {
    firstName: string;
    lastName: string;
}
```



```
function greeter(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = { firstName: "Jane", lastName: "User" };

document.body.textContent = greeter(user);
```

TypeScript Class

- **TypeScript supports class-based object-oriented programming.**
- **Let's create a Student class with a constructor and a few public fields.**

```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public lastNam
e: string) {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}
```

  – Note, the use of public on arguments to the constructor is a shorthand that allows us to automatically create properties with that name.

TypeScript Interface and Class

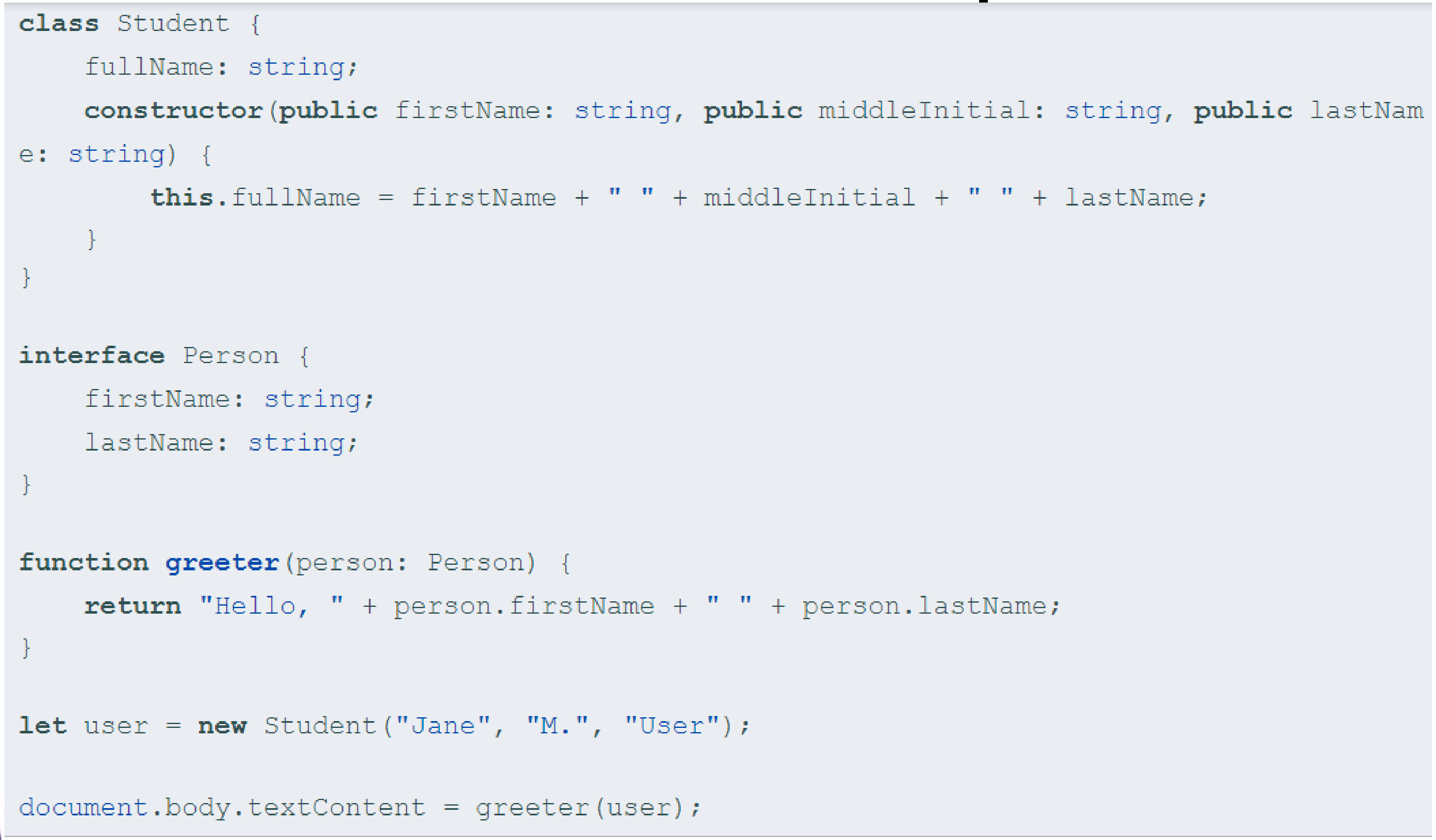
- **In TypeScript, the two types (i.e., Interface and Class) are compatible if their internal structure is compatible.**
  - This allows us to implement an interface just by having the shape the interface requires, without an explicit implements clause.

```
interface Person {
    firstName: string;
    lastName: string;
}
```

```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public lastNa
e: string) {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}
```

TypeScript Interface and Class

- **In TypeScript, Interface and Class are compatible if their internal structure is compatible**



```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public lastNam
e: string) {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = new Student("Jane", "M.", "User");

document.body.textContent = greeter(user);
```