Angular

By

Srikanth Pragada

What is Angular?

- Angular is a JavaScript framework to build more interactive web applications.
- It is designed for both **Web, Desktop and Mobile** platforms.
- Angular is the next version of most successful **AngularJS 1.x**, which was released in 2010.
- Angular was completely rewritten from scratch in TypeScript, which is superset of JavaScript.
- Angular was finally released on **14th Sep, 2016**. It is called Angular or **Angular 2**.
- **Angular 4** was released in **March, 2017** and there was no Angular 3.
- **Angular 5** was released in **Nov, 2017.**
- **Angular 6** was released in **May, 2018.**
- **Angular 7** was released in **Oct, 2018.**
- It has been optimized for developer **productivity**, small **payload size**, and **performance.**
- Developed using **TypeScript**, which is Microsoft's extension of JavaScript that allows use of all ES 2015 (ECMAScript 6) features and adds type checking and object-oriented features like interfaces.
- You can write code in either **JavaScript** or **TypeScript** or **Dart.**

Setting Up Development Environment

- ❑ Install **Node.js** and **NPM**
- ❑ Install **Angular CLI**
- ❑ Install **Visual Studio Code** or Similar IDE
- ❑ Create a new project using Angular CLI
- ❑ Open project folder in Visual Studio Code
- ❑ Start server using NodeJS

Installing Node.JS and NPM

- Node.js is used to run JavaScript on server and provide environment for build tools.
- NPM is used to manage packages related to JavaScript libraries.
- NPM itself is a Node application.
- To install, go to https://nodejs.org/en/download and download installer (.msi) for 32-bit or 64-bit.
- Run .MSI file to install Node.js into your system. It is typically installed into c:\Program Files\nodejs folder.
- It installs Node.js and NPM.
- It also sets system path to include Node and NPM.
- Make sure Node version is 6.9.x or higher and NPM version is 3.x.x. or higher.
- Go to command prompt and check their versions using following commands in console.

```
> node -v

> npm -v
```

Installing Angular CLI

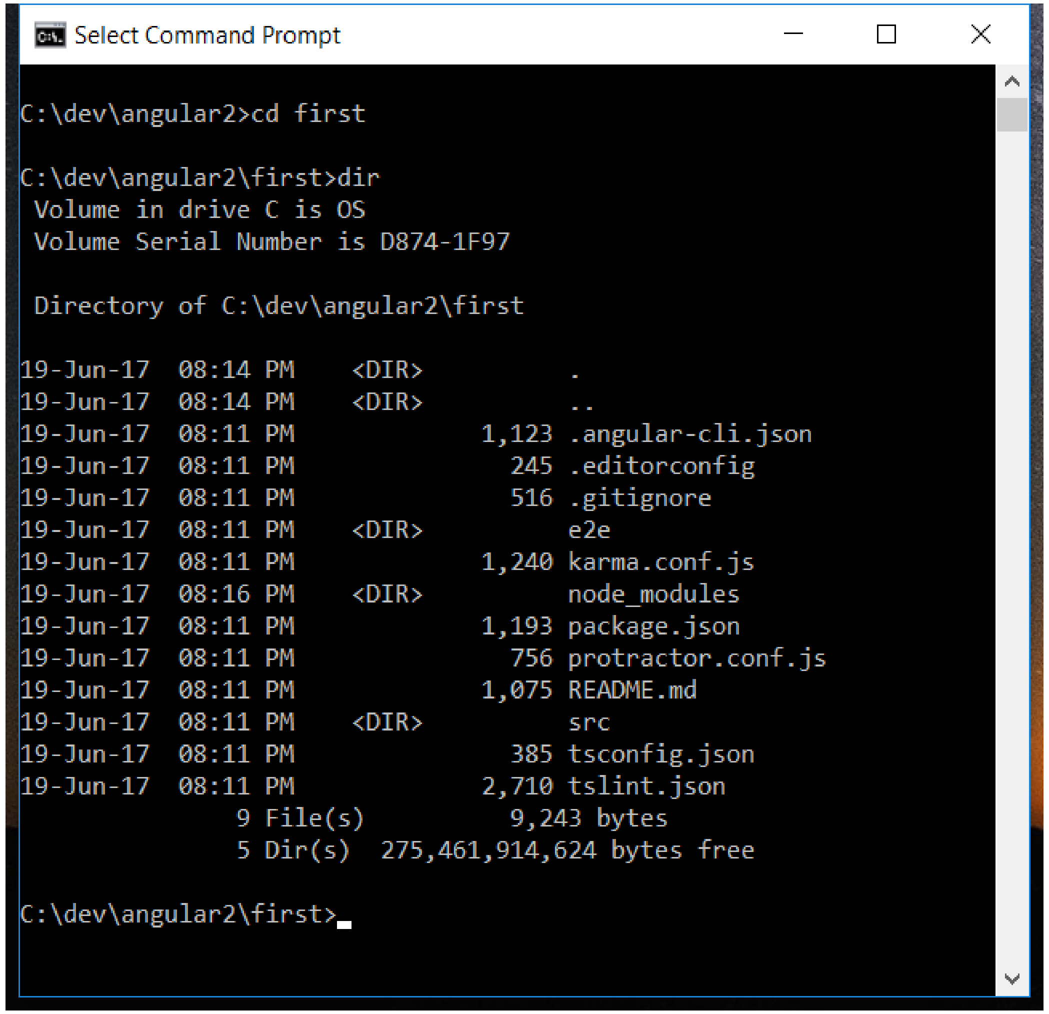
- ❑ Angular CLI - Command Line Interface for Angular Applications.
- ❑ It needs Node 6.9.x or higher, together with NPM 3.x.x or higher.
- ❑ Install Angular CLI using NPM as follows:

```
> npm install -g @angular/cli
```

Using Angular CLI

- Give the following command in console to create a new Angular Project. It creates the following directory structure:

```
ng   new   first
```



Select Command Prompt

```
C:\dev\angular2>cd first

C:\dev\angular2\first>dir
 Volume in drive C is OS
 Volume Serial Number is D874-1F97

 Directory of C:\dev\angular2\first

19-Jun-17  08:14 PM    <DIR>          .
19-Jun-17  08:14 PM    <DIR>          ..
19-Jun-17  08:11 PM             1,123 .angular-cli.json
19-Jun-17  08:11 PM               245 .editorconfig
19-Jun-17  08:11 PM               516 .gitignore
19-Jun-17  08:11 PM    <DIR>          e2e
19-Jun-17  08:11 PM             1,240 karma.conf.js
19-Jun-17  08:16 PM    <DIR>          node_modules
19-Jun-17  08:11 PM             1,193 package.json
19-Jun-17  08:11 PM               756 protractor.conf.js
19-Jun-17  08:11 PM             1,075 README.md
19-Jun-17  08:11 PM    <DIR>          src
19-Jun-17  08:11 PM               385 tsconfig.json
19-Jun-17  08:11 PM             2,710 tslint.json
               9 File(s)          9,243 bytes
               5 Dir(s)  275,461,914,624 bytes free

C:\dev\angular2\first>
```

Usage of Angular CLI

Scaffold	Usage
Component	ng g component my-new-component
Directive	ng g directive my-new-directive
Pipe	ng g pipe my-new-pipe
Service	ng g service my-new-service
Class	ng g class my-new-class
Guard	ng g guard my-new-guard
Interface	ng g interface my-new-interface
Enum	ng g enum my-new-enum
Module	ng g module my-module

Start Server

- ❑ While in project folder (first), enter the following command to start server.
- ❑ Server watches your files and rebuilds app whenever you make changes to files in folder.

```
ng serve
```

- ❑ Go to browser and navigate to following URL:

```
http://localhost:4200
```

- ❑ It is possible to start server on a different port number than default (4200) as follows:

```
ng serve –port 3000
```

Important files in project folder

app/app.module.ts

Defines AppModule, the root module that tells Angular how to assemble the application. Right now it declares only the AppComponent.

app/app.component.{ts,html,css,spec.ts}

Defines the AppComponent along with an HTML template, CSS stylesheet, and a unit test. It is the root component of what will become a tree of nested components as the application evolves.

index.html

The main HTML page that is served when someone visits your site. Most of the time you'll never need to edit it. The CLI automatically adds all js and css files when building your app, so you never need to add any script or link tags here manually.

main.ts

The main entry point for your app. Compiles the application with the JIT compiler and bootstraps the application's root module (AppModule) to run in the browser.

Building Blocks

- ❑ Modules
- ❑ Components
- ❑ Templates
- ❑ Metadata
- ❑ Data binding
- ❑ Directives
- ❑ Services
- ❑ Dependency injection

Module

- ❑ A module is a class that is decorated with **@NgModule** decorator.
- ❑ Every application contains at least one module called root module, conventionally called as **AppModule**.
- ❑ NgModule decorator provides information about module using properties listed below:
  - ✓ **Declaration** – Classes that belong to this module. They may be components, directives and pipes.
  - ✓ **Exports** – The subset of declarations that should be visible to other modules.
  - ✓ **Imports** – Specifies modules whose exported classes are needed in this module.
  - ✓ **Bootstrap** – Specifies the main application view – root component. It is the base for the rest of the application.

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

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

Angular Module vs. JavaScript Module

- A module in Angular is a class decorated with **@NgModule** decorator.
- Whereas a module in JavaScript is a file and all objects defined in the file belong to that module.
- JavaScript module exports some objects using **export** keyword and they can be imported in other modules using **import** statement.
- Angular ships a couple of JavaScript modules each beginning with **@angular** prefix.
- In order to use objects in those modules, we need to import them using **import** statement in JavaScript.

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

Component

- A component controls a part of the screen called view.
- Every component is a class with its own data and code.
- A component may depend on services that are injected using dependency injection.
- The template, metadata, and component together describe a view.
- Components are decorated with **@Component** decorator through which we specify **template, styles** and **selector** (tag) related to component.
- Properties like **templateUrl, styleUrl** and **providers** can also be used.

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

@Component({
  selector: 'my-first',                        // tag to be used in view
  templateUrl : './first.component.html'      // html page that describes view
})
export class FirstComponent {
   title : string = "Srikanth Technologies";
}
```

Template

- ❑ Component's view is defined with its companion template.
- ❑ A template is in the form of HTML that tells Angular how to render the component.
- ❑ Template is the user interface (UI) of component.
- ❑ Template can use the following:
  - ✓ HTML
  - ✓ Interpolation {{ …}}
  - ✓ Structural and Attribute directives
  - ✓ Data binding
  - ✓ Template reference variables
  - ✓ Pipes

```
<html>
<body>
<h1> {{courseName}} </h1>
<ul>
    <li *ngFor="let topic of topics"> {{ topic }} </li>
</ul>
</body>
</html>
```

Interpolation

- When you put properties in view template using {{ }}, it is called interpolation.
- Angular updates the display whenever property changes.
- The context of the expression is component instance that is associated with the view.
- The expression is evaluated, converted to string and then sent to client.
- From the expression, we can call methods of component associated with view.
- The target of the template expression may be HTML element, a component, or a directive.
- In case of assigning value to a property, template expression may appear in double quotes.

```
<span [hidden]="isNew">Modified</span>
<img  src="{{filename}}" />
```

Template Statement

- A template **statement** responds to an **event** raised by a binding target such as an element, component, or directive.
- The template statement parser differs from the template expression parser and specifically supports both basic assignment (=) and chaining expressions (with ; or ,).
- Template statements cannot refer to anything in the global namespace. They can NOT refer to window or document. They cannot call console.log or Math.max.
- The following expressions of JavaScript are NOT allowed:
    - ✓new
    - ✓increment and decrement operators, ++ and --
    - ✓operator assignment, such as += and -=
    - ✓the bitwise operators | and &
    - ✓the template expression operators

```
<button (click)="calculate()">Calculate</button>
<button (click)="onSave($event)">Save</button>
```

Data Binding

- ❑ Data binding allows copying values from Component properties to attributes of HTML elements.
- ❑ Following are different bindings available :
  - ✓ Property Binding
  - ✓ Event Binding
  - ✓ Attribute binding
  - ✓ Class binding
  - ✓ Style binding

Property Binding

- ❑ Binds value to a property in HTML element or Angular Component.
- ❑ Target property must be a property and not just an attribute in DOM element.

```
{{expression}}
[targetproperty]="expression"
bind-targetproperty="expression"
```

```
<img [src]="heroImageUrl">
<img bind-src="heroImageUrl">
<img src="{{imageUrl}}">
```

HTML Attribute vs. DOM Property

- ❑ Attributes are defined by HTML.
- ❑ Properties are defined by the DOM (Document Object Model).
- ❑ A few HTML attributes have 1:1 mapping to properties. Example: id
- ❑ Some HTML attributes don't have corresponding properties. Example: colspan
- ❑ Some DOM properties don't have corresponding attributes. Example: innerHTML
- ❑ Value of property might change at runtime but value of attribute doesn’t change. Example: value

Angular Property Binding deals with DOM properties and not HTML attributes

Attribute Binding

- ❑ Sets the value of an attribute directly.
- ❑ Must be used when there is no element property to bind.
- ❑ Attribute binding syntax resembles property binding.
- ❑ Start with the prefix **attr**, followed by a dot (.) and the name of the attribute.
- ❑ You then set the attribute value, using an expression that resolves to a string.

```
<td [attr.colspan]="noCols">Value</td>
```

Angular Expression

Class Binding

- ❑ Adds or removes CSS class names from element's **class** attribute.
- ❑ Class binding syntax resembles property binding.
- ❑ Instead of an element property between brackets, start with the prefix class, optionally followed by a dot (.) and the name of a CSS class **- [class.classname].**
- ❑ Angular adds the class when the template expression evaluates to true. It removes the class when the expression is false.
- ❑ **NgClass** directive is usually preferred when managing multiple class names at the same time.

```
<div [class.special]="isSpecial">Special Offer</div>
```

Angular Expression

Style Binding

- Sets inline styles.
- Instead of an element property between brackets, start with the prefix style, followed by a dot (.) and the name of a CSS style property: **[style.style-property].**
- **NgStyle** directive is usually preferred when setting multiple styles at the same time.

```
<button [style.color]="isSpecial ? 'red': 'green'">Show Details</button>
<button [style.background-color]="canSave ? 'cyan': 'grey'">Save</button>
```

Event Binding

- ❑ Event binding syntax consists of a target event name within parentheses on the left of an equal sign, and a quoted template statement on the right.
- ❑ The binding conveys information about the event, including data values, through an event object named **$event**.
- ❑ If the name fails to match an element event, Angular reports an error.

```
<button (click)="onSave()">Save</button>
<button on-click="onSave()">Save</button>
```

Attribute Directives

- ❑ These attributes are used like attributes of HTML elements.
- ❑ They modify the behavior of HTML elements.

ngClass	Add and remove a set of CSS classes
ngStyle	Add and remove a set of HTML styles
ngModel	Two-way data binding to an HTML form element

ngClass Directive

- ❑ Used to add or remove many CSS classes at the same time.
- ❑ Bind ngClass to a key:value object where each key of the object is a CSS class name; its value is true if the class should be added, false if it should be removed.

```
currentClasses = {"special"  : true, "big"  : true};
```

```
<div [ngClass]="currentClasses"> … </div>
```

ngStyle Directive

- Used to set many inline styles at the same time.
- Bind ngStyle to a **key:value** control object where each **key** of the object is a **style name** and its **value** is whatever is **appropriate for that style.**

```
brightRed = {"color"  : "red", "font-size"  : "30pt"};
```

```
<div [ngStyle]="brightRed">Some Text Here!</div>
```

Structural Directives

- ❑ These directives are responsible for HTML layout.
- ❑ They typically add, remove, and manipulate the host elements to which they are attached.

ngIf	Conditionally adds or removes an element from DOM
ngFor	Repeats a template for each item in the list
ngSwitch	Adds one element from a set of possible elements based on condition. Used with ***ngSwitchCase*** and ***ngSwitchDefault***

ngIf Directive

- ❑ Conditionally adds or removes an element from the DOM.
- ❑ Adding and removing DOM elements is different from displaying and hiding element.
- ❑ It is possible to have **if then else** structure with true block and false block using **ng-template.**

```
<div *ngIf="condition">Add Div to DOM when condition is true </div>
```

```
<div *ngIf="condition; else falseBlock">True Block</div>
<ng-template  #falseBlock><div>False Block</div></ng-template>
```

```
<div *ngIf="condition; then trueBlock; else falseBlock"></div>
<ng-template  #trueBlock><div>True Block</div></ng-template>
<ng-template  #falseBlock><div>False Block</div></ng-template>
```

ngFor Directive

- Repeats a template for each item in the given list.
- NgFor is a repeater directive — a way to present a list of items. You define a block of HTML that defines how a single item should be displayed.
- It provides some **exported values** that can be assigned to local variables.

index	The index of the current item in the list. It is a number
first	True if item is first item in the list
last	True if item is last item in the list
even	True if item has even index
odd	True if item has odd index

ngFor Directive

```
<div *ngFor="let topic of topics; let i=index; let first=first;let oddRow=odd">
        <h2 *ngIf="first">List Of Topics</h2>
        <span [class.red]="oddRow">
             {{i + 1}} - {{topic}}
        </span>
</div>
```

ngSwitch Directive

- It is like the JavaScript switch statement.
- It can display one element from among several possible elements, based on a switch condition.
- Angular puts only the selected element into the DOM.
- NgSwitch is actually a set of three, cooperating directives: **NgSwitch**, **NgSwitchCase**, and **NgSwitchDefault.**

```
<div [ngSwitch]="mode">
   <h1   *ngSwitchCase="'f'">Flight {{flight}} </h1>
   <h2   *ngSwitchCase="'t'">Train  {{train}}  </h2>
   <h3   *ngSwitchDefault>Private Transport</h3>
</div>
```

Template Reference Variable (#var)

- ❑ A template reference variable is often a reference to a DOM element within a template.
- ❑ It can also be a reference to an Angular component or directive.
- ❑ Use the hash symbol (#) to declare a reference variable.
- ❑ You can refer to a template reference variable anywhere in the template.
- ❑ The scope of a reference variable is the *entire template*.
- ❑ You can use the ref- prefix alternative to #.

```
<input  #email  type="email" />
<button (click) ="search(email.value)">Search</button>
```

```
<input  ref-email type="email" />
<button (click) ="search(email.value)">Search</button>
```

Pipes

- ❑ A pipe takes data as input and transforms it to required output.
- ❑ Angular provides built-in pipes for many common requirements.
- ❑ Pipes are used in template.
- ❑ Pipes can also take optional parameters to fine tune output.
- ❑ To add parameters to a pipe, follow the pipe name with a colon (:) and then the parameter value (such as currency : 'EUR').
- ❑ If the pipe accepts multiple parameters, separate the values with colons (such as slice:1:5).
- ❑ You can chain pipes together so that output of one pipe goes to another pipe as input.

- ✓ DatePipe
- ✓ JsonPipe
- ✓ LowerCasePipe
- ✓ CurrencyPipe
- ✓ DecimalPipe
- ✓ PercentPipe
- ✓ SlicePipe
- ✓ UpperCasePipe
- ✓ TitleCasePipe

DatePipe

- ❑ Formats a date according to locale rules.

```
date_expression | date [:format]
```

'medium'	Equivalent to 'yMMMdjms' (e.g. Sep 3, 2010, 12:05:08 PM for en-US)
'short'	Equivalent to 'yMdjm' (e.g. 9/3/2010, 12:05 PM for en-US)
'fullDate'	Equivalent to 'yMMMMEEEEd' (e.g. Friday, September 3, 2010 for en-US)
'longDate'	Equivalent to 'yMMMMd' (e.g. September 3, 2010 for en-US)
'mediumDate'	Equivalent to 'yMMMd' (e.g. Sep 3, 2010 for en-US)
'shortDate'	Equivalent to 'yMd' (e.g. 9/3/2010 for en-US)
'mediumTime'	Equivalent to 'jms' (e.g. 12:05:08 PM for en-US)
'shortTime'	Equivalent to 'jm' (e.g. 12:05 PM for en-US)

COMPONENT	**SYMBOL**
year	y
month	M
day	d
weekday	E
hour	j
hour12	h
hour24	H
minute	m
second	s
timezone	z
timezone	Z
timezone	a

DatePipe - Example

```
{{ dateObj | date }}              // output is 'Apr 11, 2017'
{{ dateObj | date:'medium' }}     // output is 'Apr 11, 2017, 09:43:11 PM'
{{ dateObj | date:'shortTime’}}  // output is '9:43 PM'
{{ dateObj | date:'mmss' }}       // output is '43:11'
```

DecimalPipe

```
number_expression | number [:digitInfo]
```

```
{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}
```

minIntegerDigits
Is the minimum number of integer digits to use. Defaults to 1.

minFractionDigits
Is the minimum number of digits after fraction. Defaults to 0.

maxFractionDigits
Is the maximum number of digits after fraction. Defaults to 3.

CurrencyPipe

- Formats a number as currency using locale rules.

```
number_expression | currency [:currencyCode[:symbolDisplay [:digitInfo]]]
```

currencyCode
Is the currency code, such as USD for the US dollar and EUR for the euro.

symbolDisplay
Is a boolean indicating whether to use the currency symbol or code - true: use symbol (e.g. $), false(default): use code (e.g. USD).

PercentPipe

- ❑ Formats a number as a percentage.

```
number_expression | percent [:digitInfo]
```

```
{{ .20 | percent }}                    //  results 20%
{{ 0.125539 | percent:'2.2-3' }}    //  results in 12.554
```

SlicePipe

- ❑ Creates a new List or String containing a subset (slice) of the elements.

```
array_or_string_expression | slice :start [:end]
```

start
The starting index of the subset to return.

end
The ending index of the subset to return.

```
{{str | slice:0:4}}
{{str | slice:-4}}
{{str | slice:4:0}}
```

Custom Pipes

- ❑ Create a class and decorate it with @**Pipe** decorator.
- ❑ The pipe class implements **PipeTransform** interface's transform method that accepts an input value followed by optional parameters and returns the transformed value.
- ❑ There will be one additional argument to the transform method for each parameter passed to the pipe.
- ❑ The **@Pipe** decorator allows you to define the **pipe name** that you'll use within template expressions. It must be a valid JavaScript identifier.
- ❑ You must include your pipe in the **declarations** array of the **AppModule**.

brackets.pipe.ts

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

@Pipe({name: 'brackets'})
export class BracketsPipe implements PipeTransform
{
  transform(value: string, newcase : string = 'n') {
      if (newcase == 'u')
          value = value.toUpperCase();
      else
         if (newcase == "l")
            value = value.toLowerCase();

      return "[" + value + "]";
  }
}
```

pipesdemo.component.ts

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

@Component({
  selector: 'my-pipes',
  template: `<h2>Custom Pipes Demo </h2>
     {{ title | brackets }}
     <p></p>
     {{ title | brackets:'u'}}
  `
})
export class PipesDemoComponent{
    title : string = "Srikanth Technologies";
}
```

app.module.ts

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

import {PipesDemoComponent}  from './pipesdemo.component';
import {BracketsPipe} from './brackets.pipe'



@NgModule({
  declarations: [
    PipesDemoComponent, BracketsPipe
  ],
  imports: [
    BrowserModule

  ],
  providers: [],
  bootstrap: [PipesDemoComponent]
})
export class AppModule { }
```