# Install

In order to install Angular CLI, the following should be installed in the development environment:

node (at least version 4.0)

npm (at least version 3.0)

#Installing Angular CLI is as easy as typing the follow command:

npm install -g @angular/cli

# create a new project called my-project.

ng new my-project

This will cause several things to happen:

directory my-project will be created

directory structure and source files will be generated

any needed dependencies will be installed

TypeScript will be configured

Karma test runner will be configured

Protractor end-to-end test runner will be configured

environment files will be created

# Build the application

ng build

this will generate a dist folder in homepath which contains a folder with the similar name to application. Copy this folder to any server, then open index.html from that folder.

# Run Application

Change to the my-project directory and run the following:

ng serve

Then open a browser and point it to http://localhost:4200/.

# Create elements

either "ng generate" or "ng g" command can be used

class

interface

enumeration

component

service

module

## Create A Class

To create a new class, run the following:

ng g class my-class-a

This will create a file called my-class-a.ts under the src/app directory.

## Create An Interface

To create a new interface, run the following:

ng g interface my-interface-a

This will create a file called my-interface-a.ts under the src/app directory.

## Create An Enumeration

To create a new enumeration, run the following:

ng g enum my-num

This will create a file called my-enum.enum.ts under the src/app directory.

## Create A Component

To create a new component, run the following:

ng g component my-component-a

This will do several things:

create a directory called my-component-a under src/app directory

generate four files under that directory

my-component-a.component.css

Contains any css that would be needed for this component

Optional file that is pointed to by the component.ts file

my-component-a.component.html

Contains any html that would be needed for this component

Optional file that is pointed to by the component.ts file

html could be contained within the component.ts file, if desired

my-component-a.component.spec.ts

unit test skeleton for this component

my-component-a.component.ts

exports a class called MyComponentAComponent

implements an interface called OnInit

generates empty function called ngOnInit for OnInit interface

generates empty constructor function

decorates class with @Component

add selector for component, app-my-component-a

adds templateUrl, points to generated html file for component

adds styleUrls array, points to generated css file for component

modifies app.module.ts file, added MyComponentAComponent to declarations (every component has to belong to a module)

## Create A Service

To create a new service, run the following;

ng g service my-service-a

This will generate a couple of files under the src/app directory:

my-service-a.service.spec.ts

unit test skeleton for this service

my-service-a.service.ts

exports a class called MyServiceAService

generates empty constructor function

decorates class with @Injectable

## Create A Module

To create a new module, run the following:

ng g module my-module-a

This will do a couple things:

create a directory my-module-a under src/app

generate a file under that directory called my-module-a.module.ts

exports a class name MyModuleAModule

decorates that class with @NgModule

## Create Component In A Module

Components can be added to generated module by changing to the module directory:

cd src/app/my-module-a

ng g component my-subcomponent-a

or by prefixing the module name to the front of the new component name:

ng g component my-module-a/my-subcomponent-a

This will do several things:

create a directory my-subcomponent-a under the src/app/my-module-a directory

generate all the component files under this directory (see Create a component section for description of files)

add MySubcomponentAComponent to the my-module-a.module.ts file

# Types of Bindings ( data flow from component to html )

## Interpolation

---- This will be in html file

<button id='{{buttonName}}'>myButton</button>

---- This will be in component file.

let buttonName = 'addButton';

## PropertyBinding

---- This will be in html file

<button [id]='buttonName'>myButton</button>

---- This will be in component file.

let buttonName = 'addButton';

## AtrributeBinding

Attribute binding is same as property binding with the difference that we need to add attr. as prefix with attribute name.

---- This will be in html file

<tr>

<td [attr.colspan]="clspn"> A + B </td>

</tr>

## ClassBinding

The Angular Class binding is used to add or remove classes to and from the HTML elements. You can add CSS Classes conditionally to an element, hence creating a dynamically styled element.

---- This will be in html file

<button [ngClass]='addClasses()'>myButton</button>

---- This will be in component file. boldClass and italicClass are two css classes declared in css file.

addClass(){

let classes = {

boldClass: true;

italicClass: true;

}

return classes;

}

## StyleBinding

We can set the inline styles of a HTML element using the style binding in angular.

---- This will be in html file

<button [style.fontSize.px]="'20'" >Big Button</button>

## Event binding ( data flow from html to component )

---- This will be in html file

<button (click)='onClickMethod()'>myButton</button>

OR

<button on-click='onClickMethod()'>myButton</button> ----- This is canonical form

---- This will be in component file. boldClass and italicClass are two css classes declared in css file.

onClickMethod(){

console.log("button clicked");

}

# Two ways binding

This we can achieve using FormsModule. Here we are using ngModule in example.

This example contains ‘property Binding’ + ‘event Binding’

<input [(ngModule)] = ‘name’>

* Square bracket indicates event binding and parenthesis indicates property binding.

You entered: {{name}}

# ternary operator

isPass ? 'passed' : 'failed'

# Structural Directives

---- \*ngIf ( this works as if condition)

<div \*ngIf="hero" class="name">{{hero.name}}</div>

---- \*ngFor ( this works as forEach loop)

<ul>

<li \*ngFor="let hero of heroes">{{hero.name}}</li>

</ul>

---- \*ngSwitch ( this works as Switch condition)

<div [ngSwitch]="hero?.emotion">

<app-happy-hero \*ngSwitchCase="'happy'" [hero]="hero"></app-happy-hero>

<app-sad-hero \*ngSwitchCase="'sad'" [hero]="hero"></app-sad-hero>

<app-confused-hero \*ngSwitchCase="'confused'" [hero]="hero"></app-confused-hero>

---- \*ngIf ( this works as if condition)

<app-unknown-hero \*ngSwitchDefault [hero]="hero"></app-unknown-hero>

</div>

# Pipes

## CurrencyPipe

Transforms a number to a currency string, formatted according to locale rules that determine group sizing and separator, decimal-point character, and other locale-specific configurations.

Ex:

{{ salary | currency : ‘USD’ : true : ‘1.3-3’}}

Currency pipe on salary with 3 parameters. 1) currency in USD, 2) true – indicates currency symbol not the code, 3) 1.3-3 – at least 1 digit before decimal, at least 3 and at max 3 digits after decimal.

## DatePipe

Formats a date value according to locale rules.

Ex: {{ dateOfBirth | date: ’fulldate’ }}

## DecimalPipe

Transforms a number into a string, formatted according to locale rules that determine group sizing and separator, decimal-point character, and other locale-specific configurations.

## DeprecatedCurrencyPipe

Formats a number as currency using locale rules.

## DeprecatedDatePipe

Formats a date according to locale rules.

## DeprecatedDecimalPipe

Formats a number as text. Group sizing and separator and other locale-specific configurations are based on the active locale.

## DeprecatedPercentPipe

Formats a number as percentage according to locale rules.

## I18nPluralPipe

Maps a value to a string that pluralizes the value according to locale rules.

## I18nSelectPipe

Generic selector that displays the string that matches the current value.

## JsonPipe

Converts a value into its JSON-format representation. Useful for debugging.

## KeyValuePipe

Transforms Object or Map into an array of key value pairs.

## LowerCasePipe

Transforms text to all lower case.

## PercentPipe

Transforms a number to a percentage string, formatted according to locale rules that determine group sizing and separator, decimal-point character, and other locale-specific configurations.

## SlicePipe

Creates a new Array or String containing a subset (slice) of the elements.

## TitleCasePipe

Transforms text to title case. Capitalizes the first letter of each word, and transforms the rest of the word to lower case. Words are delimited by any whitespace character, such as a space, tab, or line-feed character.

## UpperCasePipe

Transforms text to all upper case.

Ex: {{ fullName | uppercase }} – this will print fullName value in capital or uppercase letters.

# Reactive Forms

Should be import

1. import { [FormBuilder](https://angular.io/api/forms/FormBuilder), FormGroup, FormControl } from '@angular/forms';

## FormBuilder

An injectable service that provides factory methods for creating control instances.

## AbstractControl

The abstract base class for the concrete form control classes FormControl, FormGroup, and FormArray. It provides their common behaviors and properties.

## FormControl

Manages the value and validity status of an individual form control. It corresponds to an HTML form control such as <input> or <select>.

Ex: name : new FormControl(‘ ‘, Validators.required); // while using direct form control.

name: [‘ ‘] // while using form builder

## FormGroup

Manages the value and validity state of a group of AbstractControl instances. The group's properties include its child controls. The top-level form in your component is FormGroup.

Ex :

address : new FormGroup( {

\*\* create form controls here\*\*

}); // while using direct form control.

--------------------------------------------------------------------------------------

constructor(private fb: [FormBuilder](https://angular.io/api/forms/FormBuilder)) { }

address: this.fb.group({

\*\* create form controls here\*\*

}); // while using form builder

## FormArray

Manages the value and validity state of a numerically indexed array of AbstractControl instances.

1. phones: this.fb.array([
2. this.fb.control('')
3. ])