BINDINGS

[Drop a comment for suggestions, requests & fixes]

Data Binding is communication between the application logic (TS) and the user interface (HTML)

String Interpolation
Read var from typescript and output data to html
String interpolation accepts any expression that could be resolved to a string in the end

TS
```
varName = "online";
get_id() { return "235"; } // function that returns a string
```
HTML
```
<p> {{ "Server" }} is {{ varName }} and it's id is: {{ get_id() }}</p>
```

Property binding
Dynamically bind html tag properties
Any TypeScript code that will be placed between the quotation marks will work

TS
```
textValue = "Paragraph one";
```
HTML
```
<p [innerText]="text Value~~textValue~~"></p>
```

Disable button via Property binding
The disable keyword can be anything, this line will toggle the "disabled" keyword visibility in the HTML

TS
```
buttonIsDisabled = true;
```
HTML
```
<button [disabled]="!(buttonIsDisabled)"> Button Text </button>
<p [innerText]="buttonIsDisabled ? 'btn disabled' : 'btn enabled'"></p>
```

Event binding

HTML
```
<button (click)="onButton1Clicked()">Button 1</button>
```

Grab text input via Event binding
The input keyword is a standard dom event provided by the element. The $event keyword is a reserved variable that can be used only in the template when using event binding

TS
```
onInput( event:Event ) {
  // explicit casting event.target to HTMLInputElement
  inputTextValue = (<HTMLInputElement>event.target).value; }
```
HTML
```
<input type="text" (input)="onInput( $event )">
```

Two way data binding
Combination of output data and event binding using the **ngModel** directive (which have to be imported in the application module)

TS
```
variableX = "my text value"
```
HTML
```
<input type="text" [(ngModel)]="variableX">
<p>{{ variableX }}</p>
```

Structural Directive `**ngIf**`
* accepts boolean expression, method/variable

HTML
```
<p *ngIf="isParagraph1Visible()">Paragraph 1</p>
<p *ngIf="paragh2Visible">Paragraph 2</p>
```

Structural Directive `**Ng if else**`
* Swaps the element with the given local reference
* The local reference is sometimes called a marker
* It's possible to use to ngIf statements with the logical not flag on the second ngIf instead

HTML
```
<p *ngIf="flagVariable; else pNoVisible"> flag is True </p>
<ng-template #pNoVisible>
        <p> flag is false </p>
</ng-template>
```

Attribute directive **ngStyle**
* Accepts argument as javascript object in the standard structure {style:value,…} pairs
* Object's key must be camel case if it's not a String
* Variables and functions are also valid as long as they return a js style object

HTML
```
<p [ngStyle]="{'background-color': getColor() }" >Example 1</p>
<p [ngStyle]="{backgroundColor: 'red' }" >Example 2</p>
<p [ngStyle]="getStyleColor( 'red' )"> Example 3</p>
```
TS
```
getPStyle( c ) {   return { color : c };   }
```

Attribute directive **ngClass**
Accepts argument as javascript object, were the key is the class name and value is a boolean that enables/disables the same class

HTML
```
<p [ngClass]="{cssClassName : classIsActive === 'yes' }"> Paragraph </p>
```

Structural Directive `**ngFor**`
* Bind to array
* Replicate DOM elements
* Let i = index, is optional

TS
```
myList = [ "dog" , "cat", "bird" ];
```
HTML
```
<p *ngFor="let listItem_value of myList">{{listItem_value}}</p>
<p *ngFor="let item of myList; let i = index">{{item}} at {{index}}</p>
```

ANGULAR CLI

Ng Serve Options

Official description	**Option**	**Default Value**	**Explanation**
Listens only on localhost by default	-H	localhost	
Whether to reload the page on change, using live-reload	-lr	true	
Opens the url in default browser	-o	false	
Port to listen to for serving	-p	4200	
Log progress to the console while building	-sm	N/A	

Ng Generate Component

Official description	**Option**	**Default Value**	**Explanation**
Allows for skipping the module import	**--skip-import**	false	Will skip the declaration import to declarations array in app.module.ts, @NgModule({ **declarations**: […], … });
Flag to indicate if a dir is created.	**--flat**	false	Prevent creating a folder for the component and place it in app folder instead
Specifies if the style will be in the component typescript file	**--inline-style** (alias) -is	false	
Specifies if the template will be in the component typescript file	**--inline-template** (alias) -it	false	
Allows specification of the declaring module's file name (e.g `app.module.ts`).	**--module** (alias) -m	N/A	If --skip-import is set to false, this will tell angular cli which module the component will be imported into.
Specifies if a spec file is generated.	**--spec**	true	Set this parameter to false if you wish to not create the .spec.ts file

Examples

Create component comp1, place it into comp1 folder, without the style, html and spec files. Without importing component into module.

```
> ng g c comp1 --skip-import --is -it --spec false
```

Create component comp2 with a folder named comp2, then place comp2 folder inside comp1 folder

```
> ng g c comp1/comp2
```

AngularJS Version based installation

```
> npm install -g @angular/cli@1.0.6
```

ABOUT

Component - (Creating a new component by hand)While it's recommended to create a new folder for each component, it is not necessary.

Since angular will use each component to create objects from it, every component needs to be a TypeScript class (with the .ts file extension). For example i have create **myComp.ts** and placed it in to **project_folder/src/app** directory.

To allow angular use this class globally it must be first exported

```
export class MyComp { … }
```

A component-decorator tells angular that this class is a component (note: decorators comes with the at `@` sign in front of them)

```
// import component decorator from angular core package
import { Component } from '@angular/core';
// pass a js object to configure the component decorator, this will be stored as a metadata for this class
@Component( {
    // the html tag, each selector within your app must be unique
    selector: 'comp-selector-name',
    // templateUrl will link any html document to this component, to use inline html code use 'template'
    templateUrl: 'compPage.html'
    // optional -> styleUrls
} )

Export class MyComp { … }
```

Available component selector types	TypeScript component decorator	Html document example
Html tag	selector: 'comp-name'	<comp-name></comp-name>
Attribute	selector: '[comp-name]'	<div comp-name></div>
Element class	selector: '.comp-name'	<div class="comp-name"></div>

Angular uses components to build web pages and uses modules to bundle components into packages, generally a small application will be sufficient with the main app/app.**module**.ts . A module provides angular about the futures that the app have.
To make the new component work it has to be declared within the main module. In app/app.**module**.ts:

```
…
// import component class from the provided directory including the filename ( without the .ts extension )
import { MyComp } from './myComp'; // don't forget to include folder to path if comp isn't located in the app directory
…
@NgModule( {
    declarations: [
        … ,
        MyComp  // include component declaration into module
    ],
    imports : [ … ],
    providers : [],
    bootstrap : [ AppComponent ] // the root component that will be loaded into index.html
} )
…
```

To see your component working add the following into the html documents

myComp.html (your html file linked to component decorator)	app.component.html (main app component)
<h1>My Component</h1>	<comp-selector-name></comp-selector-name>

Directives

Directives are instructions to the DOM. Any component is a directive with a template. A custom directive can be simply used as a parameter in any html element. Here is an example of a new directive structure in the code:

somePage.html	app.MyDirectiveName.ts

`<div paramaterName> … </div>`	`@Directive({ Selector: '[paramaterName]' })`
	`export class MyDirectiveName { … }`

Structural directives such as **ngIf** will change the Html structure, all structural directives must have a star as the first character.

IF / Else directive example

```
// The structural directive will include the element only if the typescript condition is true
<button *ngIf="buttonIsVisible">Click Me</button>
// If the condition is false and the else keyword is present, ngIf will look up the local reference and display it instead
<h1 *ngIf="myText.length == 0; else altHeader">
   Text is empty
</h1>
// The local reference must be contained within the ng-template html element
<ng-template #altText>
   // Everything inside the ng-template element will be swapped with the target ‘ngIf’ element
   <h1>
      My text is {{myText}}
   </h1>
</ng-template>
```

Attribute directive won’t modify the dom structure and don’t need the star character at the beginning of the keyword.
All non-structural directives must follow the property binding syntax (wrapped around with two square brackets), this will bind the directives to the html element. For example the **ngStyle** directive will control the elements **style** parameter as this is how that directive was structured. The attribute directive doesn't necessary have to share a common naming between the directive name, and it’s target control parameter, but it is a good practice to do so.

(page break)