Angular-Getting Started

Clarification and Hagdarot

* TypeScript is a super set of JavaScript.
* Any – is like auto in c++.
* A list of valid DOM events – <https://developer.mozilla.org/en-US/docs/Web/Events>
* Installs:
  + nodejs + npm from <http://nodejs.org/en/download>

Clone repository from  - <https://github.com/DeborahK/Angular-GettingStarted.git>

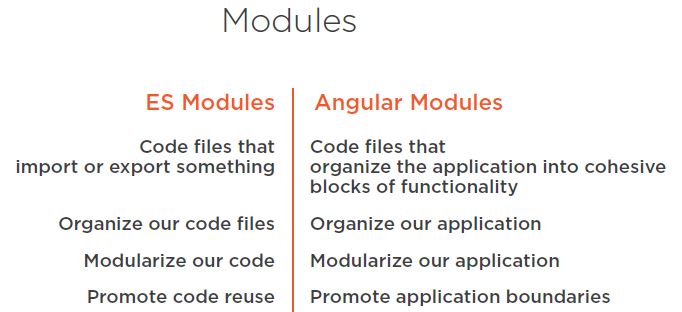
To my repository - <https://github.com/guyma/Angular-GettingStarted.git>

* **Font awesome** – npm install bootstrap font-awsome

# Modules

* + JS have problems, with namespacing
  + Angular JS and TypeScripts have modules that helps us to organized our code, ES2015 set a standard for defining a module
  + **ES2015** defines that a **module is a file** and a **file is a module.**
    - when coding in es2015 we don’t need to define or name modules.
    - angular uses es2015 and we also have angular modules that are different then the implementation of angular to es2015 modules.
  + ES2015 modules implemented by angular
    - Export

if we write export class Product { } in a file named product.ts then this file becomes a module and we can use it in any other class by importing it.

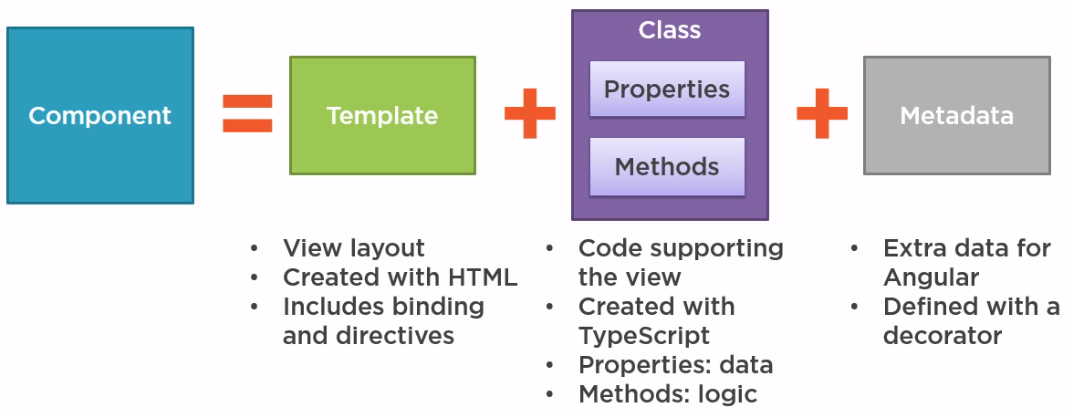
* Import  
  in a file product-list.ts we can write   
  import { Product } from ‘./product’   
  ‘./product’ , we don’t list the extension but it means we specify that we want to import from a file called product.**js** but our file is product.**ts**.  
  so when we compile our code, the typescript file is transpile into an **ES5** js file (**.js)**, so in run-time we are importing from .js file
* this file also becomes a module because we imported something…
  + Angular modules implemented by angular
    - angular modules helps us to organize our application into   
      a cohesive (מגובש) blocks of functionality.
    - Every angular application has at least one angular module, by convention called **app.module.ts**
    - in those modules we declare the set of components and other code files associated with the module and the dependencies needed by those components.
    - Each component we create is declare in and belongs to one module.
    - 

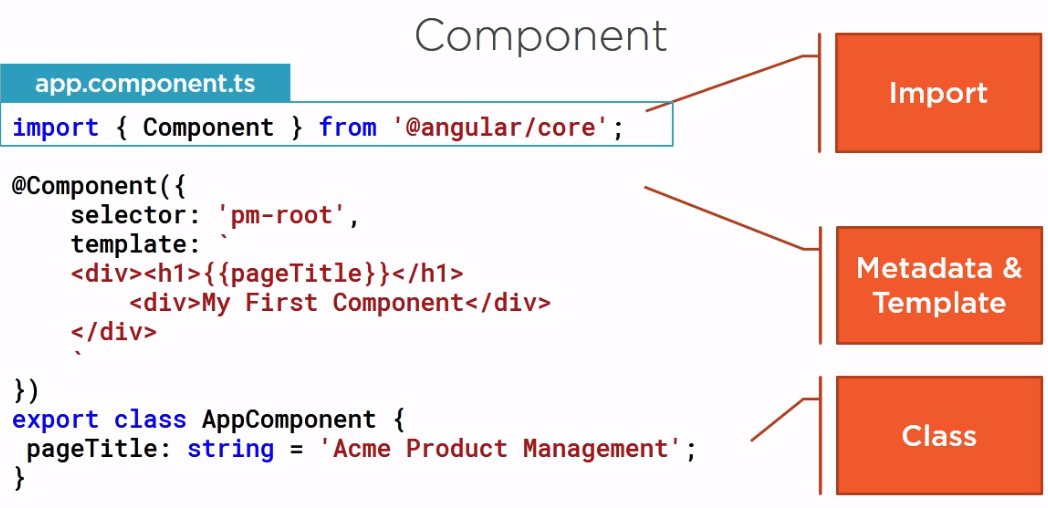
# Components

An angular component includes a template which lays out the user interface fragment which defining a view for the application. It's created with html and defines what is rendered on the page. We use angular binding and directive to power up the view.  
Add to that a class, for the code associated with the view, the class is created with type script. The class contains the properties or data elements available for use in the view.  
For example, if we want to display a title in the view, we define a class property for that title.

The class also contains Methods which are the functions for the logic needed by the view.  
For example, if we want to hide/show we write the logic in a class method.

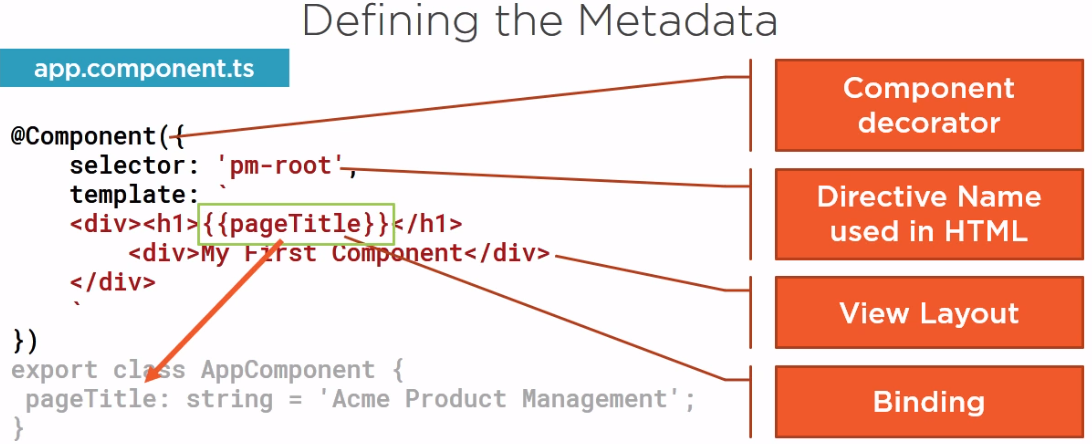
A component also has metadata which provides additional info about the component to angular. It is that metadata that defines that class as an angular component.  
The metadata is defined with decorator.  
**Decorator** is a function that adds a metadata to a class, its members or its methods arguments.





## Creating the Component Class

* + Crate a file componentName.component.ts
  + By convention
    - Name the class with xxxComponent
    - The root component is called AppComponent
  + We reference to this component by name.
  + Add an export keyword to make it available for use by other components of the application.
  + Note: since this file is exporting something, it became an ES2015 module.
* A class become an angular component when we give it metadata.  
  Angular needs this metadata to understand how to instantiate the component, construct the view and interact with the component. We define a component metadata with the angular component function. In type script we attach that function to a class as a decorator.
* A decorator is a JS language feature that is implemented in type script.
  + the scope to the decorator is limited to the feature it's decorates.
  + Prefix with an @
  + Angular provides built-in decorators we use to provide additional info to angular, we can also build our own decorators.
  + We apply a decorator by positioning it immediately in front of the feature we are decorating. When decorating a class we define it immediately above the class.
  + Directive – is an html tag, whenever the directive is used in the html, angular renders this components template.
  + A component should always have a template, here we define the layout for the user interface fragment or view, managed by this component.
  + The { { } } double curly braces indicated a data binding, we bind the h1 element value to the pageTitle property of the class, so when this html is rendered, the <h1> element will display the content of pageTitle.



* + Selector – here we specify the selector for the name of this component, when used as a directive in the html.

# Importing what we need

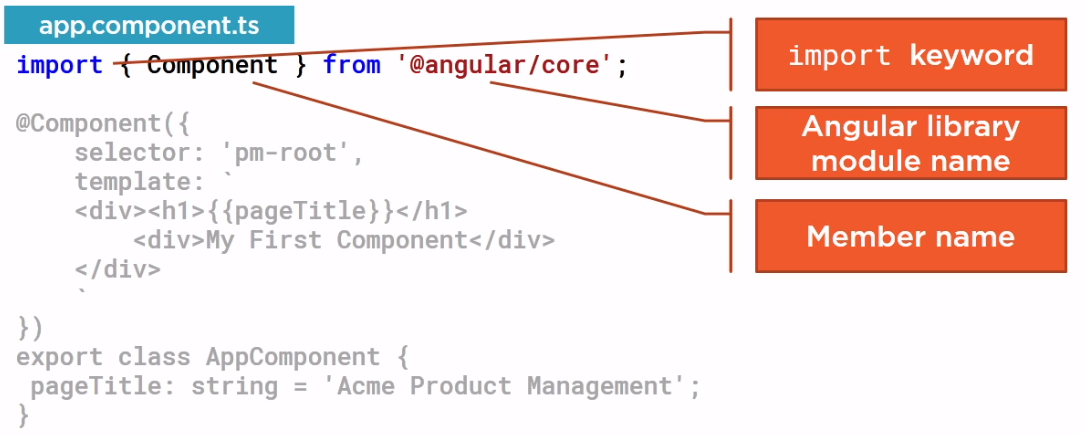
Before we can use an external function or class, we need to define where to find it.

We do this with an **import statement**, this is part of ES2015, and implemented in type script.  
it allows us to use exported members from external ES modules.

Angular is Modular – there is a collection of library modules, each library is itself a module made of several related featured modules

List of available angular library packages - link

In the component class we use the component decorator function from angular to define our class as a component so we need to **import { Component } from '@angular/core';**



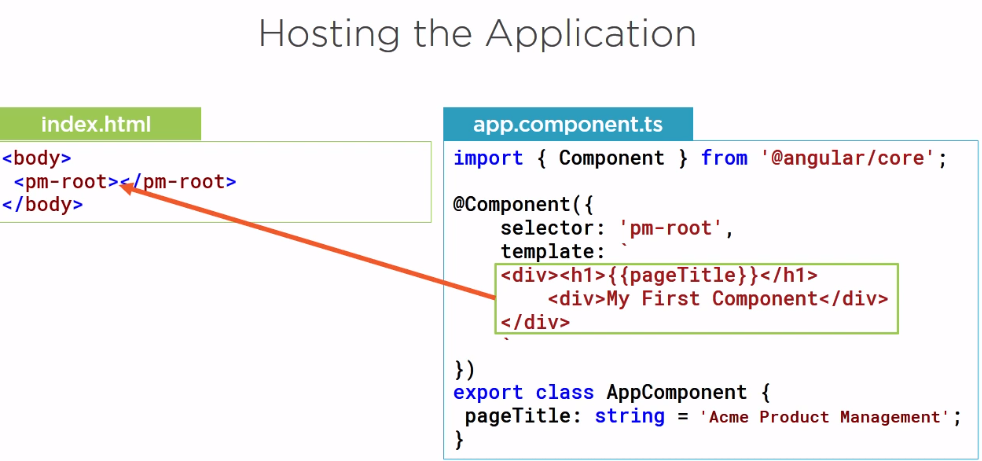
Pulling yourself up by your bootstraps – originally mean to improve you situation by your own efforts.   
In tech it means a self-starting process that loads and goes.

We need to tell angular to load our root component to a process that is called **bootstrapping**

* + First, we setup the index.html file to host our application
  + Define our root angular module to bootstrap our root component

# Single Page Application (SPA)

Most angular application have an index.html file that contains the main page for the application. This is often the only one true Web page of the application.  
Hence an Angular application in often called a Single Page Application – SPA.

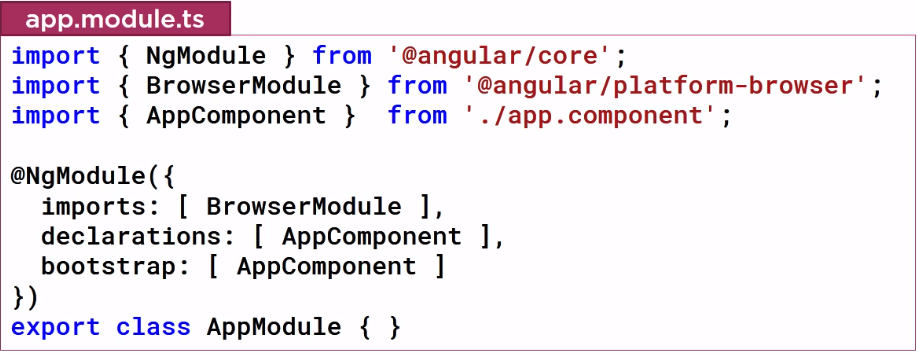


How does the angular compiler know about this custom html element?

It looks in an angular module. Angular modules helps us organize our application, provide boundaries and a Template resolution environment – when an angular compiler sees a directive and a template it looks to the angular module for the definition, so we declare the AppComponent in an angular module so the compiler can find it.  
We also use the module to bootstrap our startup component, which is AppComponent.  
And we want our application to work correctly in the browser so we add angular browser module to our angular modules imports.

# Defining the angular module

In the file app.module.ts



We define an angular module using a class AppModule, then we identify the class as an angular module (with @NgModule decorator)

* + Declaration – state the components that belongs to this module.
  + Imports – defines the external module that we to want to have available to all of the components belongs to this module.
    - browserModule – every browser application must import, it register important application service providers such as error handling
  + Bootstrap – define the startup component of the application which is our AppComponent.
    - The startup component should contain the selector we use in the index.html file.

# Component Checklist

* + Create a class for the component with code to support the view.
  + Use a decorator to define the component metadata which include the html for the component template.
  + Import what we need for any third party library from our modules or form angular itself
  + **Create a Class**
    - Give it a clear name – it's the component name.
      * Use PascalCasing – JS is case sensitive.
      * Append component to the end of the name (best practice)
    - Export keyword on the class signature.
    - Data properties – if the component view displays data, like title in our example, the data defines as a property of the class.
      * Give it a data type and a default value.
      * Use camelCase for the property name.
    - If the component view has functionality such as show/hide an image, the logic is defined as methods in the class.
      * Use camelCase for the method name.
  + **Metadata**
    - How to define metadata for our component?
    - With a component decorator.
      * @Component({})
      * Use the **selector** property to define the name of the component when used as a directive in html, if the component is not used in any html, the selector property is not needed.
      * Use the template property in the component metadata, to defined the view's html.
      * Import – it tells angular where to find the members that this component needs.

# Something Wrong! Checklist

* + F12 – in chrome, to view the developer tools.
  + Check the console for error messages.
  + <http://blogs.msmvps.com/deborahk/angular-2-getting-started-problem-solver/>

# Lesson 5

To build the UI in angular we build a template with html.

## Build a template

Define a template in a component metadata

* + Inline Template – define a property 'template:'.
    - For one-line use " to '.
    - For multi-line use `. this back-ticks is ES2015.
    - When we use inline template, we don't have intellisence.
  + Linked Template - define a property: templateUrl: 'path to html file'.
    - templateUrl: './product-list.component.html'

To make our page look nice, we use the twitter bootstrap styling framework.

For the rating start we use the font-awesome icon set and tool kit.

* + npm install bootstrap font-awsome

installing the packages does not provide access to their style sheets, for that we import the styles for these packages into our global application stylesheet, which is the style.css file

* + @import "~bootstrap/dist/css/bootstrap.min.css"
  + @import "~font-awesome/css/font-awesome.min.css"

Now these style sheets are available for any template in our application.

Best Practice – for each feature we create a new folder for all the files related.

1. Create a Folder products
2. Create product-list.component.ts
   1. Create a class inside with selector and templateUrl
3. Create product-list.component.html
   1. Write all the html code inside
4. Add the selector as the directive to the app.component.ts
5. Declare the component to the application modules in the declaration array.

## Data Binding

Coordinates communication between the component's class and its template and often involves passing data.  
We can provide value from the class to the template for display.  
The template raises events to pass user actions or user entered values back to the class.  
The binding syntax is always defined in the template.

Interpolation – is a one way binding, from the class property to the template, so the value in the class shows up in the template.   
Example: {{ pageTitle }}

Interpolation lets us perform concatenation or simple calculations like:  
{{ 'Title:: ' + pageTitle }} or {{ 2\*20+1}}

We can also call a class method like

{{ 'Title:: ' + getTitle() }}

We use interpolation to insert the interpolated string into the text between html elements.  
Or we can use interpolation with element property assignments:  
<h1 innerText={{ pageTitle }}></h1>

## Angular Directives

The angular built in directives will look at our structural directives   
A structural directives modify the structure or layout of a view, By adding/removing/modifying elements end their children.

* + \*ngIf : if logic
  + \*ngFor: For loops

The \* marks the directive as a structural directive.

**\*ngIf** – is a structural directive that removes or recreates a portion of the DOM tree, based on an expression.  
If the expression assigned to the ngIf evaluate to:  
a false value the elements and it's children are remove from the DOM.  
a true value a copy of the elements and it's children are reinserted to the DOM.

Example:   
let's assume we have an array of products as a property in the product-list component.  
we want to show the product table, only if we have products in our list, we use ngIf on the <table> element: *\*ngIf='products && products.length'*Explanation: if the product elements have a value and the product length is not 0 then the table element will appear in the DOM, else it will be removed.

How our application finds this ngIf directive?  
We import BrowserModule, and he exposes ngIf and ngFor directives.

**\*ngFor** – repeats a portion of the DOM tree, once for each item in iterable list.  
We define a block of html that defines how we want display a single item, and tell angular to use that block for displaying each item in the list.

For example, say we want to display each product in a row of a table, we define one table row and its child table data elements. That table row element and its children are then repeated for each product in the list of product.

The **let** keyword create a template input variable called product, we can reference this keyword anywhere on this element, on any sibling element or on any child element.

<tr \*ngFor='let product of products'>

<td></td>

<td>{{ product.productName }}</td>

<td>{{ product.productCode }}</td>

<td>{{ product.releaseDate }}</td>

<td>{{ product.price }}</td>

<td>{{ product.starRating }}</td>

</tr>

We know the product properties from the products-list we defined (in the Json)

for…of vs for…in

The reasoning for this has to do with ES2015 for loop.

ES2015 has both *for of* and a *for in*

The *for of* loop is similar to a for each style loop, it iterate over an iterable object, such as an array. For example, say we have an array of person nick name, if we use *for of*, we'll see each nickname log to the console.

The *for in* loop iterates over the properties of an object.  
For example, in the nickname array, the array indexes are enumerable properties with integer names. So, we'll see the indexes log to the console.

*For in* – iterating the index  
since the ngFor directive iterates over iterable objects and not their properties, angular selected to use the *of* in the ngFor expression.

Checklist: Template

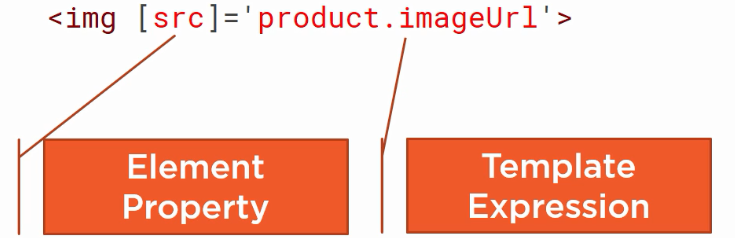
* + Inline template
    - use an inline template when building a shorter template.
    - Specify the – template: - property in the component's decorator, use double or simple quoted to define a template string.
    - Use the ES2015 back ticks to for multiple lines
  + Linked templates
    - For longer templates.
    - Specify the – templateUrl: - property in the component's decorator, and define the path to the file contains the html code.

Checklist: Component as a Directive

After building the template we build its component and learned how to use this component as a directive .

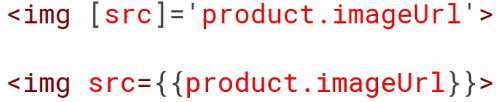
# Lesson 6 – Data Bindings & Pipes

Property Binding – allows us to set a property of an element to the value of a template expression.



Here we bind the *src* property of the *img* element to the *imgUrl* property of the product,  
effectively defining the source of the image from information in our component class.

Our binding target [*src*] is always enclosed with square brackets to the left of the equal sign, and identifies a property of the element. The binding source is always enclosed with   
quotes ‘ ‘ to the right of the equals and specifies the template expression.

For comparison, here is a similar binding using interpolation.  
  
Note that in this case the element property is not enclosed in square brackets and the template expression is enclosed in curly braces with no quotes. If you need to include the template expression as part of a larger expression, such as:  
  
you may need to use interpolation.   
Like interpolation, property binding is one way from the source class property to the target element property. It effectively allows us to control our template’s DOM, from our component class.

## Event Binding

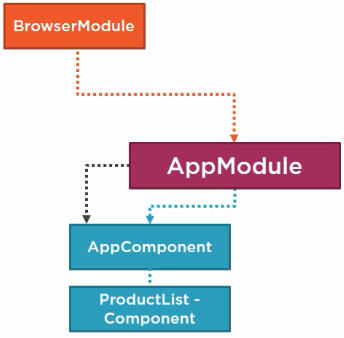
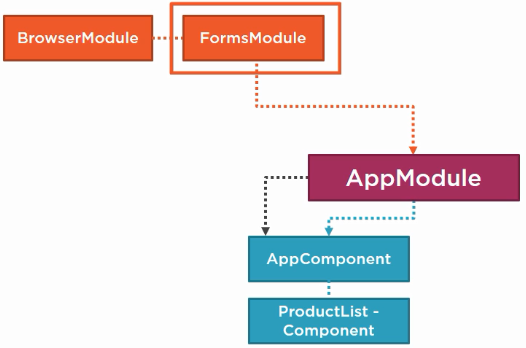
We want to send information from the DOM to our component, so we can respond to user event.   
For example, to perform an operation when the user clicks a button, a component listens for user actions using event binding as shown:  
  
The component listen to the click event on a button, the name of the bound event is enclosed in parentheses *(click)*, identifying it as the target event. To the right of the equal sign, is the template statement. This is often the name of a component class method followed by open and closing parentheses and enclosed in quotes. If the defined event occurs, the template statement is executed, calling the specified method in the component. Wondering where you might a list of valid DOM events?   
<https://developer.mozilla.org/en-US/docs/Web/Events>

**Note:** we use interpolation to change the button text between hide/show using a JavaScript conditional operator. We specify the condition and a question mark, then we specify the true expression. So when the condition is we’ll display this value.  
{{ showImage ? ‘Hide’ : ‘Show’ }}

## Two-way Binding

When working with user-entry HTML element, such as an input element, we often want to display a component class property in the template and update that property when the user makes a change. This process requires two-way binding. To specify two-way binding in angular, we use the *ngModel* directive.  
  
We enclose *ngModel* in square brackets to indicate property binding from the class property to the input element, and parentheses to indicate event binding to send a notification of the user-entered data back to the class property. We assign this directive to a template expression. To help us remember which order to put the two sets of brackets, visualize a banana in a box [()] ☺

Every time we want to use an angular directive in a template, we need to consider how to make that directive visible to the component associated with that template.  
Recall that an angular module defines the boundary or context within which the component resolves its directives and dependencies.

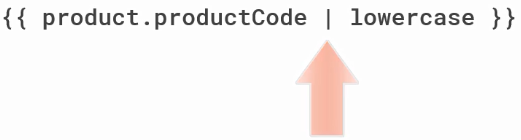
The illustrative of our AppModule currently looks like this:  
 🡪  
We want to use the *ngModel* directive in our Product-List Component, which is owned by AppModule. So in the AppModule, we need to import the appropriate system module that exposes the ngModel directive. Since the ngModel directive is most often used when building date entry forms, ngModel is part FormsModule, so we import that here.  
Now the ngModel directive and any of the other Angular forms directives are visible to ant component declared by AppModule, including our Product-List component.

In order to use the *ngModel* in Product-List Component we need to:

1. Import the FormsModule in the module that owns the product list component, which is AppModule.
2. We add the imported FormModule to the imports array for the *NgModule* decorator. We don’t add it to the declaration array because our directives, components and pipes are declared here in declarations array. Directives, components and pipes we use from other sources, such as Angular itself or third parties, are defined in external angular modules we add to the imports arrat here.

## Transforming Data with Pipes

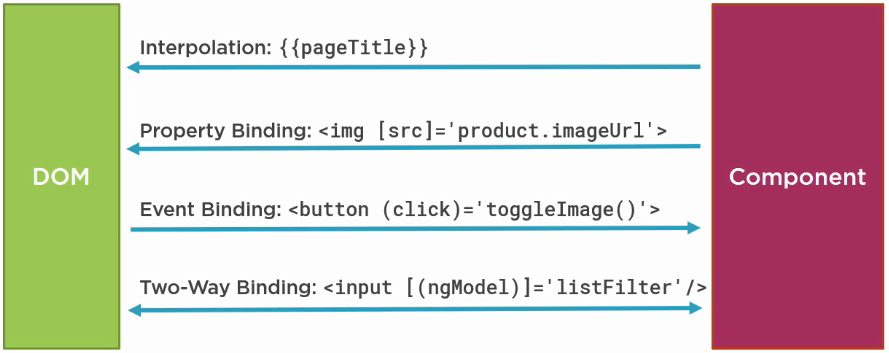
**Pipes** – transform bound properties before they are displayed, so we can alter the property values to make them more user-friendly or more locale-appropriate.  
Angular provides some built-in pipes for formatting values, such as date, number, decimal, percent, currency and so on. Angular also provides a few pipes for working with objects, such as the json pipe to display the content of an object as a json string, which is helpful when debugging, and a slice pipe, which selects a specific subset of elements from a list. We can also build our own custom pipes as we’ll see in the next module.

Example: say we want to display the product code in lowercase. We can add the pipe character after the property in the template expression.  
  
And then specify the lowercase pipe. The product code is then transformed into lower case before it’s displayed, We can also use pipes in property bindings. Add pipe after the property in the template expression and specify the desire pipe.  
  
We can also chain pipes   
By default, the currency pipe adds the all caps, three-letter abbreviation of the local currency to the amount. If we want to display that abbreviation in lowercase, we can use another pipe.  
Some pipe support parameters, Parameters are defined by specifying a colon and the parameter value.  
  
Example: The currency pipe has three parameters: the desired currency code, a string defining how to show the currency symbol, and digit info.   
The digit consists of the minimum number of integer digits (1), the minimum number of fractional digits, and the maximum number of fractional digits.

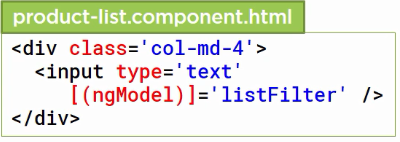
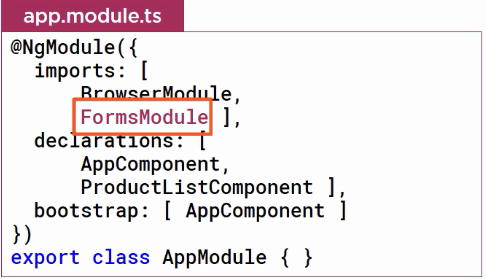
## Checklist

Data binding makes it easy to display class properties from our component and set DOM element properties to our class property values to better control the view, and the component can listen for events from the DOM to respond as needed for an interactive user experience.   
There are **four** basic types of bindings in Angular. Here is a diagram as a memory aid.

1. **Interpolation** inserts interpolated strings into the text between HTML elements or assigns elements properties. Be sure to wrap the template expression in curly braces and no quotes.
2. **Property binding** sets an elements property to the value of a template expression.  
   The binding target specifies a property of the element and must be enclosed in square brackets. The binding source specifies the template expression and must be enclosed in quotes.
3. **Event binding** listens for events and executes a template statement when the event occurs. The target event (click) specifies an event name and must be enclosed in parentheses. The template statement often defines the name of the method to call when the event occurs and must be enclosed in quotes.
4. Two-way binding display a component class property and update the property when the user makes a change. Use the banana in a box syntax with the ngModel directive. The binding source specifies the template expression and must be enclosed in quotes.



## Checklist: ngModel

* + Define ngModel within [()] for two-way binding.  
     
  + Be sure to add FormsModule from the angular/forms package to the imports array of an appropriate angular module, in this case, AppModule. This ensures that the ngModel directive is available to any template defined in a component associated with that module.   
    

## Checklist: Pipes

The data we have in our component may not be in the format we want for display.  
We can use a pipe and a template to transform that data to a more user-friendly format.  
To use a pipe:

* + specify the pipe character **|**
  + pipe name
  + pipe parameters, separated with :