**Data Binding in Angular Application**

to discuss **Data Binding in Angular Application** in detail. **Data binding** is one of the most important features provided by **Angular Framework** which allows communicating between the **component** and its **view**. So, at the end of this lab, you will understand the following pointers in detail.

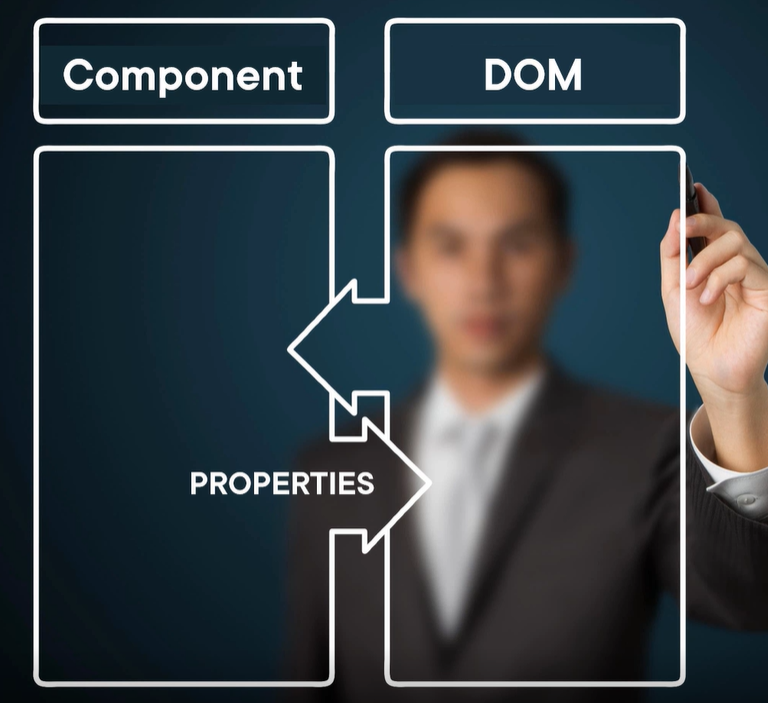
1. **Why do we need Data Binding?**
2. **What is Data Binding in Angular?**
3. **Types of Data Binding**

 When a user interacts with an application in the form of a keyboard movement, a mouse click, or a mouseover, it generates an event. These events need to be handled to perform some kind of action. This is where event binding comes into picture.

**Why do we need Data Binding?**

Whenever you want to develop any data-driven web application, then as a developer you need to keep the focus on two important things i.e. Data and the UI (User Interface) and it is more important for you to find an efficient way to bind them (Data and UI) together. Again, the data can arrive in several chunks and you need to update the user interface with the latest or updated data.

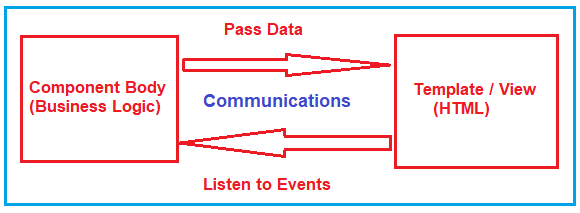
Now a day most of the front-end development uses JavaScript frameworks like Angular which does a lot of work for us. These Frameworks takes the responsibility of synchronizing the data and the user interface. The angular framework provides one concept called Data Binding which is used for synchronizing the data and the user interface (called a view).



**What is Data Binding in Angular Application?**

In Angular, Data Binding means to bind the data (Component’s filed) with the View (HTML Content). That is whenever you want to display dynamic data on a view (HTML) from the component then you need to use the concept Data binding.

Data Binding is a process that creates a connection to communicate and synchronize between the user interface and the data. In order words, we can say that Data Binding means to interact with the data and view. So, the interaction between the templates (View) and the business logic is called data binding.

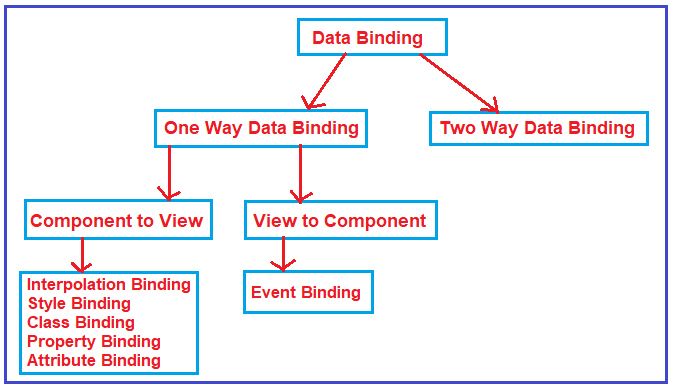


**Types of Data Binding in Angular:**

There are two types of Data binding available. They are as follows

1. **One-way Data Binding-** where a change in the state affects the view (i.e. From Component to View Template) or change in the view affects the state (From View Template to Component).
2. **Two-way Data Binding-** where a change from the view can also change the model and similarly change in the model can also change in the view (From Component to View Template and also From View template to Component).

In order to understand this better and remember, please have a look at the following image which describes the classification of Data Binding.



**Examples of Angular Data Bindings:**

1. [**Interpolation**](https://dotnettutorials.net/lesson/angular-interpolation/)
2. [**Property Binding**](https://dotnettutorials.net/lesson/angular-property-binding/)
3. [**Attribute Binding**](https://dotnettutorials.net/lesson/angular-attribute-binding/)
4. [**Class Binding**](https://dotnettutorials.net/lesson/angular-class-binding/)
5. [**Style Binding**](https://dotnettutorials.net/lesson/angular-style-binding/)
6. [**Event Binding**](https://dotnettutorials.net/lesson/angular-event-binding/)
7. [**Two-way binding**](https://dotnettutorials.net/lesson/angular-two-way-data-binding/)

## one-way Data Binding

One-way data binding will bind the data from the component to the view (DOM) or from view to the component. One-way data binding is unidirectional. You can only bind the data from component to the view or from view to the component.

### From Component to View

There are different techniques of data binding which use one-way data binding to bind data from component to view. Below are some of the techniques, which uses one-way data binding.

* Interpolation Binding: Interpolation refers to binding expressions into marked up language.
* Property Binding: Property binding is used to set a property of a view element. The binding sets the property to the value of a template expression.
* Attribute Binding: Attribute binding is used to set a attribute property of a view element.
* Class Binding: Class binding is used to set a class property of a view element.
* Style Binding: Style binding is used to set a style of a view element.

# Angular Interpolation

## ****Angular Interpolation with Examples****

to discuss the **Angular Interpolation** with Examples. Please read our. As we already discussed in our previous Lab Interpolation Data Binding is used to achieve one way data-binding i.e. From Component to the View Template.

##### ****What is Angular Interpolation?****

If you want to display the read-only data on a view template (i.e. From Component to the View Template), then you can use the one-way data binding technique i.e. the Angular interpolation.

The Interpolation in Angular allows you to place the component property name in the view template, enclosed in double curly braces i.e. **{{propertyName}}**.  So, the Angular Interpolation is a technique that allows the user to bind a value to a UI element.

Interpolation is used for one-way data binding in Angular. It embeds an expression into the HTML template. By default, expression should be surrounded by {{ and }}. This expression is also known as template expression.

{{ expression }}

Angular evaluates an expression surrounded by {{ and }} and then converts a result to a string and assigns it to an element or directive property.

{{ expression }} output=> string

The following example evaluates an arithmetic expression, converts the result into a string.

Example: Interpolation

 Copy

<p> 2 + 2 = {{ 2 + 2 }} </p> <!-- output: "2 + 2 = 4" -->

<p> 2 \* 3 = {{ 2 \* 3 }} </p> <!-- output:`"2 \* 3 = 6" -->

<p> {{ “2 + 2 != ”+ 2 + 2 }} </p> <!-- output:"2 + 2 != 22" -->

<p> {{ “2 + 2 = ”+ (2 + 2) }} </p> <!-- output:"2 + 2 = 4" -->

The context of the interpolation expression is a component class. It means it can access public properties or methods of a component class. However, it cannot access private or protected members.

##### ****Understanding Interpolation in Angular Application:****

Let us understand how to use the Interpolation with some examples.

###### **Step1: Modify the index.html**

Please modify the index.html file as shown below.

<!doctype html>

**<html** lang="en"**>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**MyAngularApp**</title>**

**<base** href="/"**>**

**<meta** name="viewport" content="width=device-width, initial-scale=1"**>**

**<link** rel="icon" type="image/x-icon" href="favicon.ico"**>**

**</head>**

**<body>**

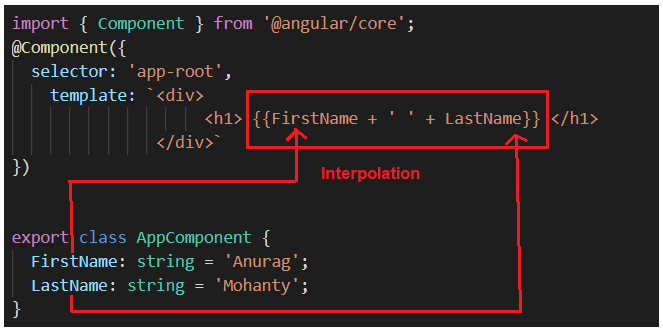
**<app-root></app-root>**

**</body>**

**</html>**

##### ****Step2: modify app.component.ts file.****

In order to understand Angular Interpolation binding, please have a look at the following image. Here, we have created two string variables (FirstName and LastName) within the component (i.e. AppComponent) and initialize these two variables with some default values. Then please focus on the template property of the @Component decorator. Here, the angular framework gets the value of the FirstName and LastName property from the component (i.e. AppComponent) and then inserts the FirstName and LastName value between the opening and closing <h1> element. Here, we are using double curly braces to place to component properties and this is called Interpolation binding in Angular Application.



So, open **app.component.ts** file and then copy and paste the following code in it.

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

@Component**({**

selector: 'app-root',

template: `<div>

<h1> {{FirstName + ' ' + LastName}} </h1>

</div>`

**})**

**export** **class** AppComponent **{**

FirstName: **string** = 'Anurag';

LastName: **string** = 'Mohanty';

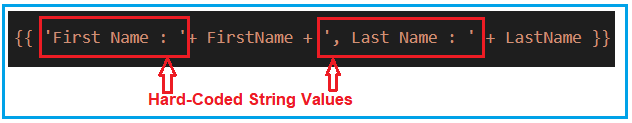
**}**

And, when you run the application, then it should display the FirstName and LastName in the browser as shown in the below image.



##### ****Angular Interpolation with hardcoded string:****

It is also possible in Angular to concatenate some hard-coded string value with the property value. In order to understand this, let us add the hard-coded string First Name before the FirstName Property and Last Name before the LastName property. If this is your requirement then you could do this as shown in the below image.



So, please modify the app.component.ts file as shown below.

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

@Component**({**

selector: 'app-root',

template: `<div>

<h1> {{ 'First Name : '+ FirstName + ', Last Name : ' + LastName }} </h1>

</div>`

**})**

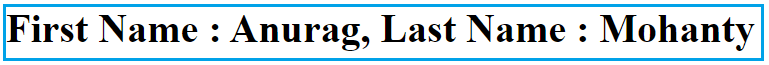
**export** **class** AppComponent **{**

FirstName: **string** = 'Anurag';

LastName: **string** = 'Mohanty';

**}**

With the above changes in place, now you should get the FirstName and LastName property values along with the hardcoded string values as shown in the image below.



##### ****Angular Interpolation with Expression:****

In Angular using Interpolation (i.e. double curly braces), you can also specify some valid expression. For example, if you want to perform mathematical calculations then you can do such calculations using interpolation as shown in the below image.

Angular Interpolation with Expression

So, modify the **app.component.ts** file as shown below and then see the output in the browser.

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

@Component**({**

selector: 'app-root',

template: `<div>

<h1> 10 + 5 \* 7 - 6 = {{ 10 + 5 \* 7 -6 }} </h1>

</div>`

**})**

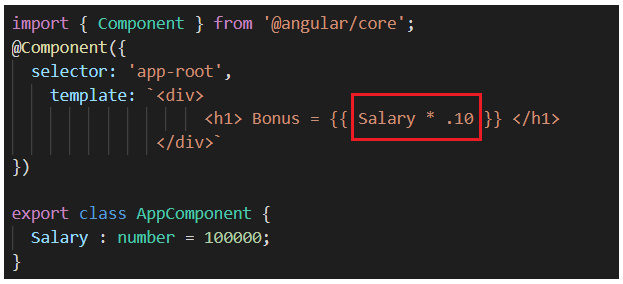
**export** **class** AppComponent **{**

**}**

###### **Output:**

Angular Interpolation with Expression

It is also possible in Angular Interpolation to combine the expression with the property value. Let us understand this with an example. Please have a look at the following image. We have Salary property and we need to apply .10 as a bonus. So, here we can combine the expression with the property using Angular Interpolation.



So, please modify the **app.component.ts** file as shown below and see the output.

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

@Component**({**

selector: 'app-root',

template: `<div>

<h1> Bonus = {{Salary \* .10 }} </h1>

</div>`

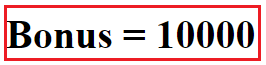
**})**

**export** **class** AppComponent **{**

Salary : **number** = 100000;

**}**

With the above changes in place, now it should display the following output.



##### ****Interpolation in Angular with Ternary Operator:****

The expression that is enclosed in the double curly braces is commonly called **Template Expression** and the template expression can also be a **ternary operator**. Please modify the app.component.ts file as shown below.

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

@Component**({**

selector: 'app-root',

template: `<div>

<h1> Last Name : {{ LastName ? LastName : 'Not Available' }} </h1>

</div>`

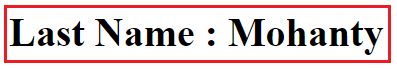
**})**

**export** **class** AppComponent **{**

LastName : **string** = "Mohanty";

**}**

In the above example, the LastName property has a value i.e. Mohanty, So, you can see the last name property in the browser as shown in the below image.



Now let’s modify the LastName property value to null as shown in the below code and then see the output in the browser.

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

@Component**({**

selector: 'app-root',

template: `<div>

<h1> Last Name : {{ LastName ? LastName : 'Not Available' }} </h1>

</div>`

**})**

**export** **class** AppComponent **{**

LastName : **string** = **null**;

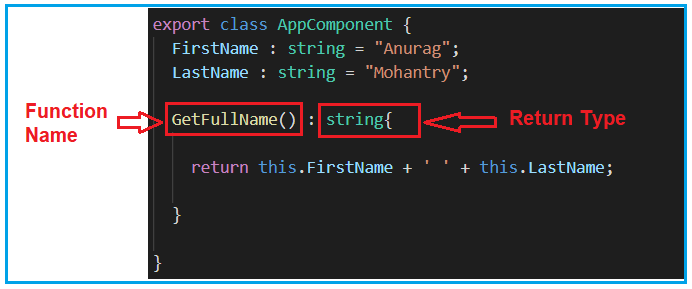
**}**

Now, when you run the application it will display the below output in the browser.

Interpolation with Ternary Operator

##### ****Method Interpolation in Angular Application:****

Let’s see how to create a method using typescript and then we will discuss how to call a class method using interpolation. So, what we will do here is, we will create one method let say GetFullName, and that method will return the full name by combining the first name and last name properties as shown in the below image. Here, you can access the class member using this operator.



Once you have a function created in the component, then you can call this function using the interpolation as shown below.

Method Interpolation Example in Angular

So, modify the **app.component.ts** file as shown below and then see the output.

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

@Component**({**

selector: 'app-root',

template: `<div>

<h1> Full Name : {{ GetFullName() }} </h1>

</div>`

**})**

**export** **class** AppComponent **{**

FirstName : **string** = "Anurag";

LastName : **string** = "Mohantry";

GetFullName**()** : **string{**

**return** this.FirstName + ' ' + this.LastName;

**}**

**}**

When you run the application, you should get the following output in the browser.



##### ****Displaying Images using Angular Interpolation:****

It is also possible in angular to display images using Interpolation. Let us say we want to display the image from the below link.

[**https://dotnettutorials.net/wp-content/uploads/2019/09/cropped-dotnettutorials.png**](https://dotnettutorials.net/wp-content/uploads/2019/09/cropped-dotnettutorials.png)

Modify the **app.component.ts** file as shown below to display images.

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

@Component**({**

selector: 'app-root',

template: `<div>

<img src = {{ImagePath}} />

</div>`

**})**

**export** **class** AppComponent **{**

ImagePath : **string** = "https://dotnettutorials.net/wp-content/uploads/2019/09/cropped-dotnettutorials.png";

**}**

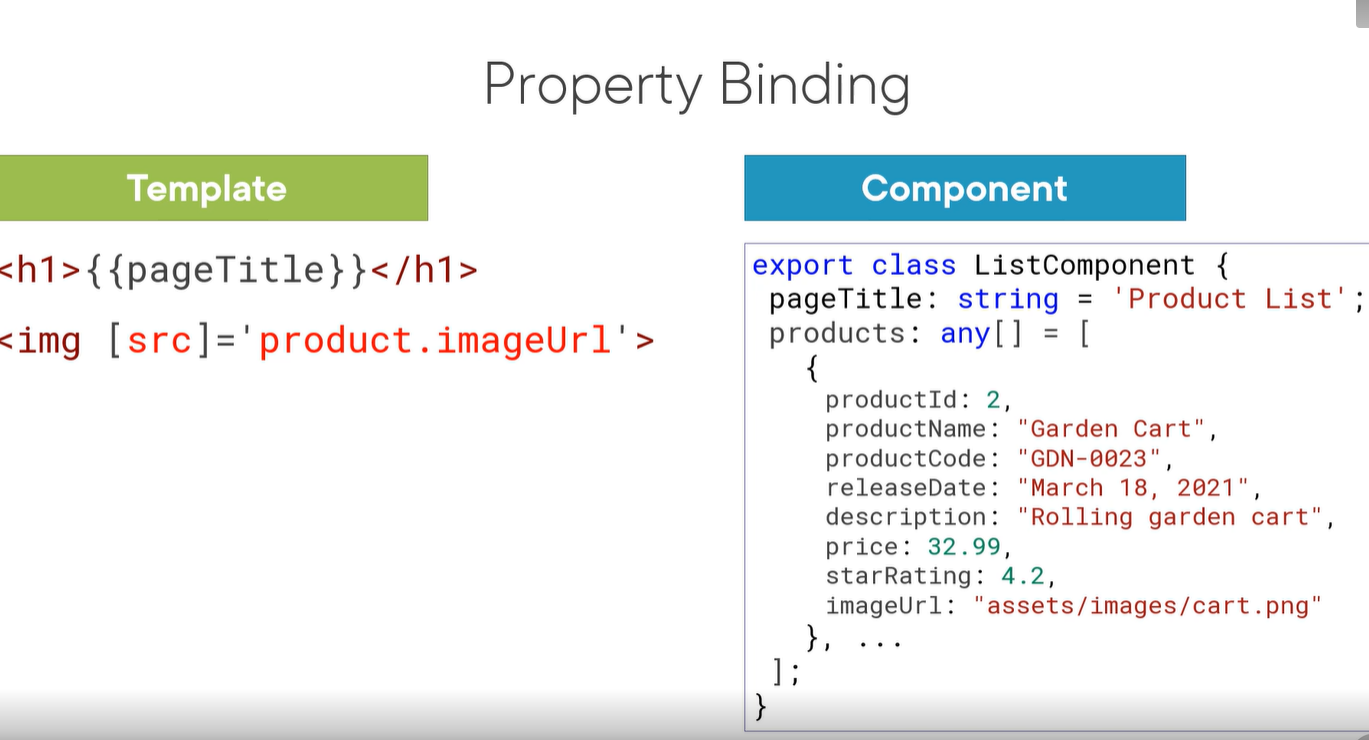
When you run the application, it will display the following image in the browser.

# Angular Property Binding Example

we will practice Angular property binding example.

Property binding is a one-way data binding from data source to view target.

Property binding is performed with component property, HTML element and Angular directives.

Component property binding is used for communication between parent and child component because using this binding we can send property values from parent to child component. 

In element property binding the DOM property of HTML element can be assigned with a value of component property. In directive property binding we can assign component property values to Angular directives.  
Component property binding is performed as below.

<my-msg prefixMsg= "Website name is " [siteName] = "website.name"> </my-msg>

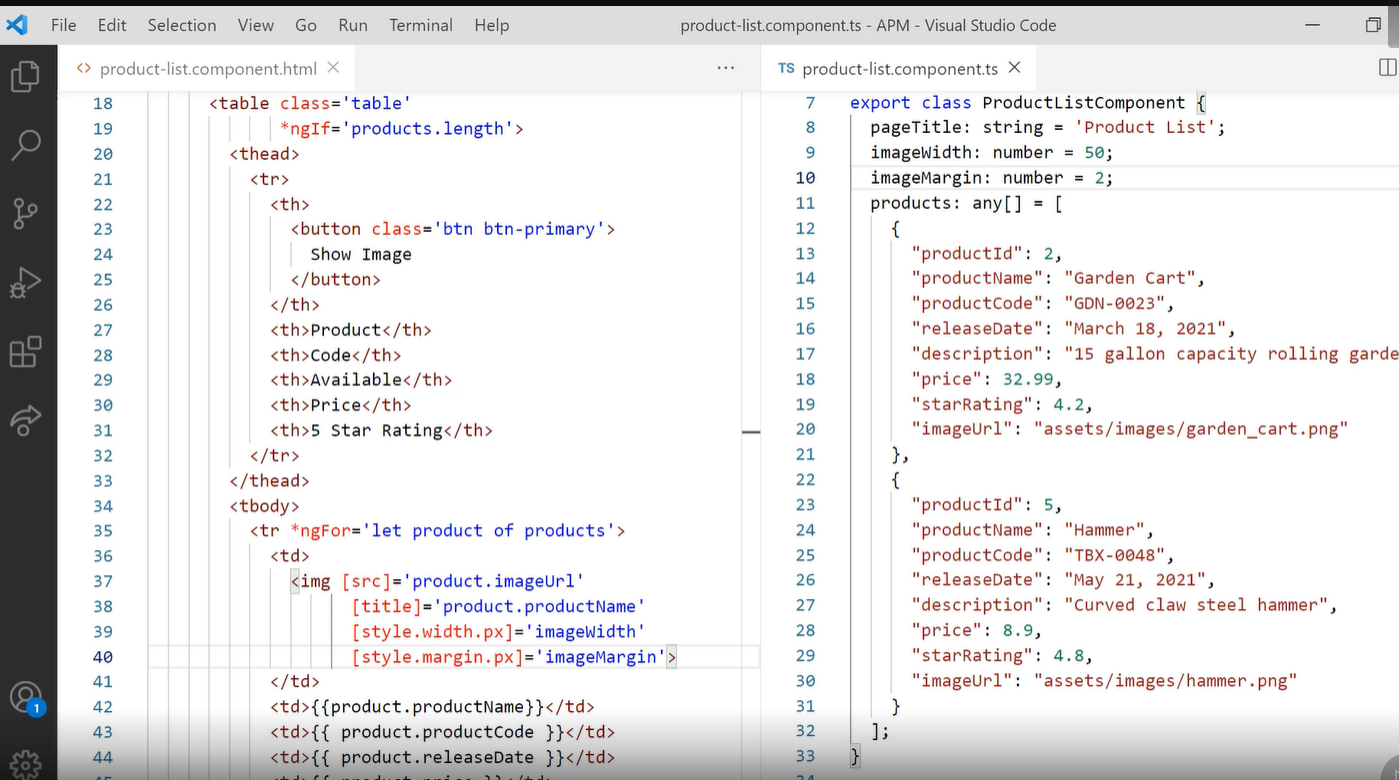
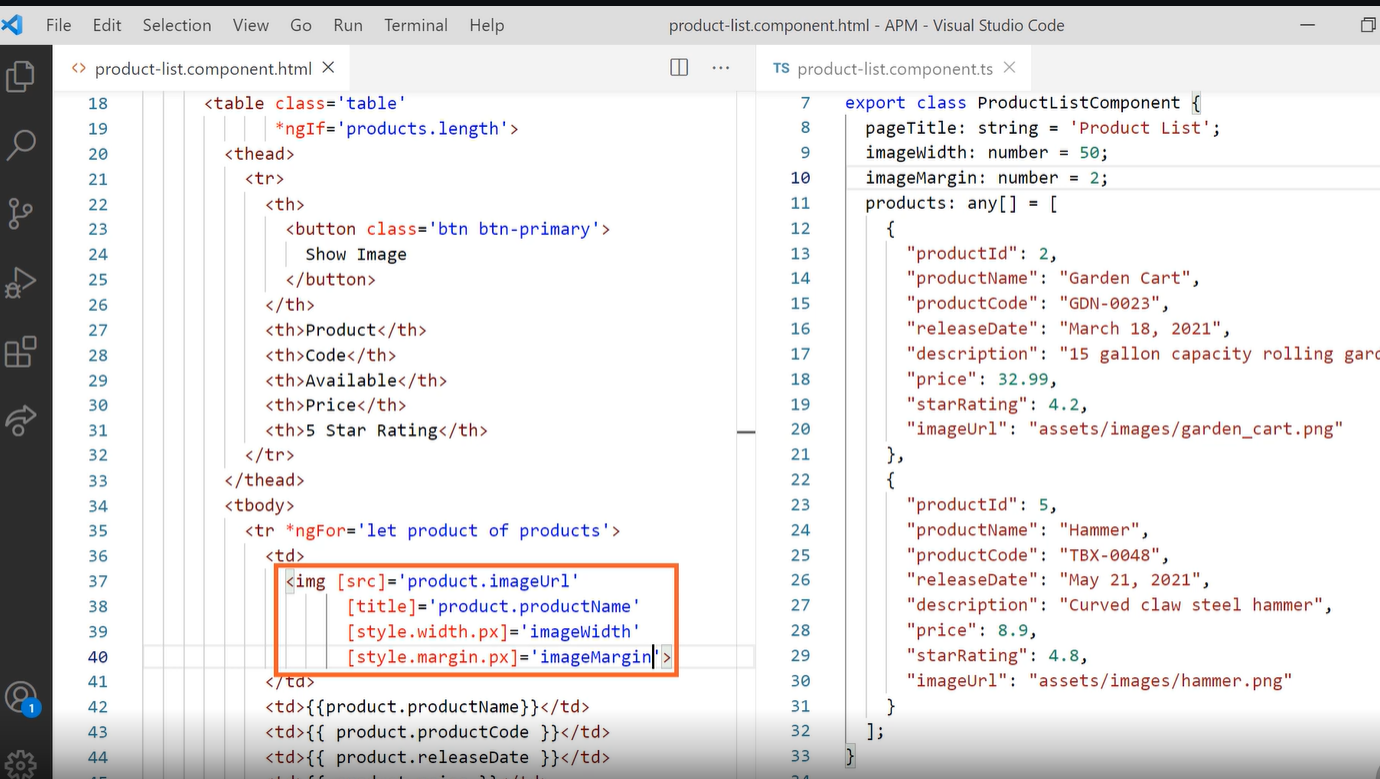
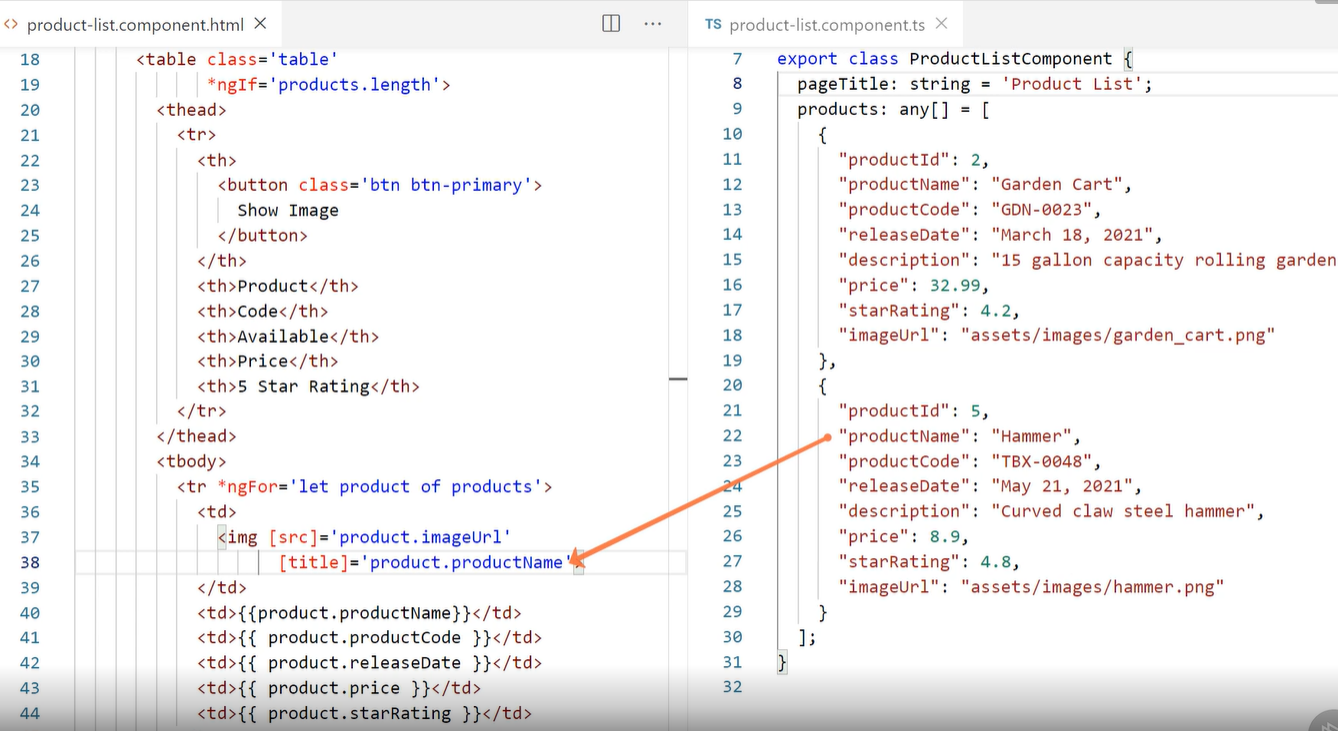
Element property binding is performed as below.

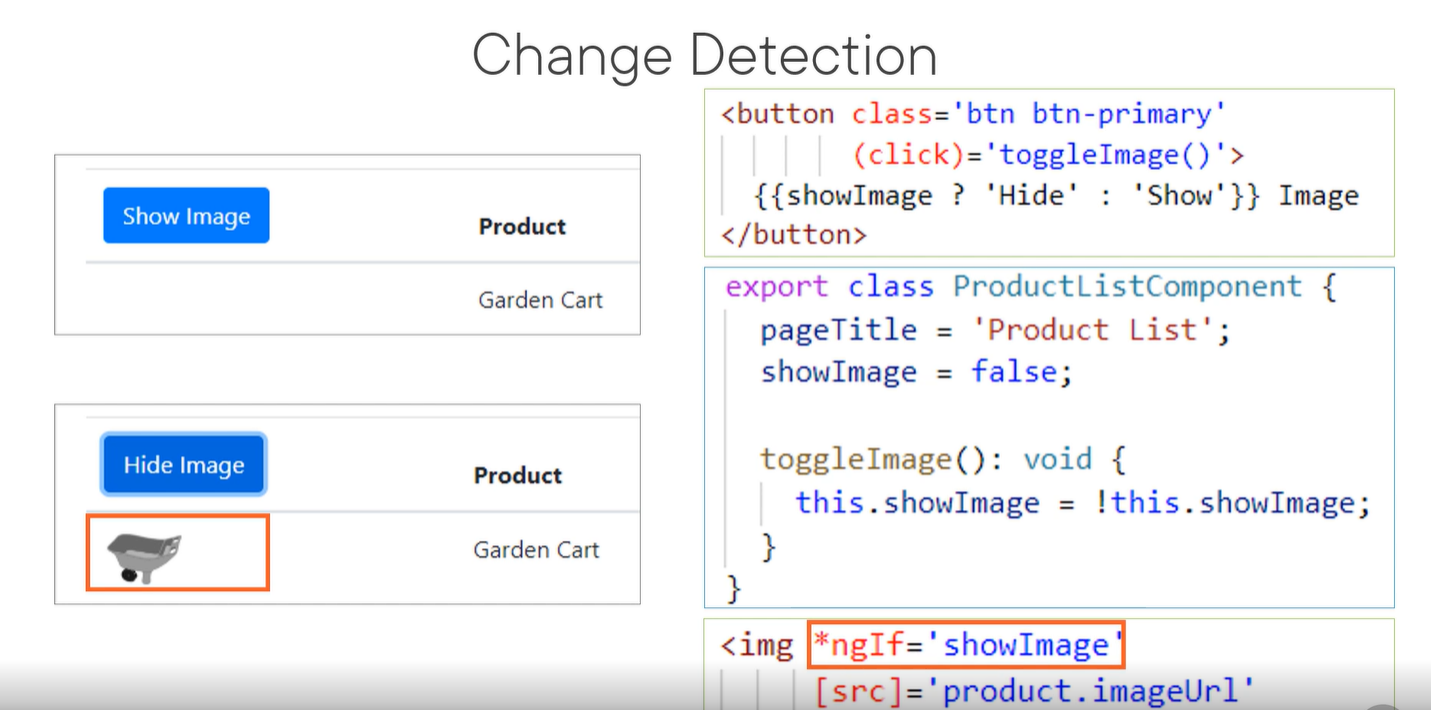
<a [href]="website.url" [textContent]="website.name"> </a>

Directive property binding is performed as below.

<p [ngClass]="'one two'"> Angular 2 Property Binding Example </p>

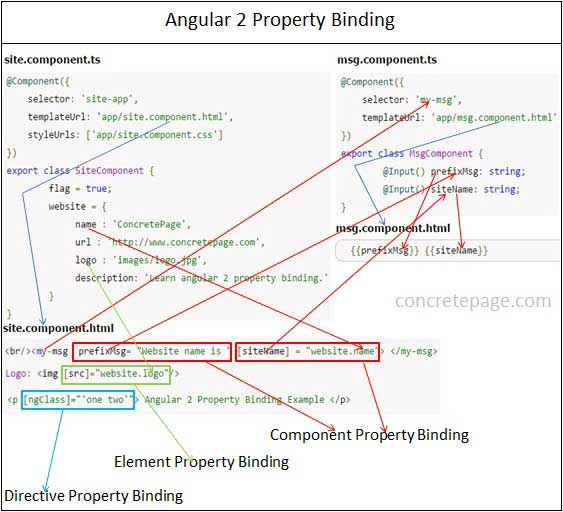
Angular framework ensures content security while displaying data. If we try to display data with <script> tag then it will not allow. Angular filters the data before display. Angular calls such type of coding not only a HTML but HTML Plus because it is more powerful. Now find the complete example of Angular property binding step-by-step.





### Property Binding with Diagram

Find the diagram for Angular property binding.



The description of diagram is as follows.  
1. Angular provides three types of property binding and these are component property binding, element property binding and directive property binding.  
  
2. **Component property binding** works within component element to bind parent component property into child component property. In the diagram the arrows and rectangles in **red** color are displaying the functionality related to component property binding.  
  
3. **Element property binding** works within HTML element and it binds a component property to a DOM property. In the diagram the arrows and rectangle in **green** color are displaying the functionality related to element property binding.  
  
4. **Directive property binding** works within HTML element with angular directives such as NgClass and NgStyle. In directive property binding a component property or any angular expression is bound to angular directive. In the diagram the arrows and rectangles in **light blue** color are displaying the functionality related to directive property binding.  
  
5. In the diagram for the component property binding we have two components. SiteComponent is acting as parent component and MsgComponent is acting as child component. The property website.name from SiteComponent is bound to the property siteName from MsgComponent. So the values of website.name has been copied to siteName. We need to take care that the input property of child component must be decorated with @Input() decorator. The property prefixMsg of MsgComponent has been bound to a constant string using component property binding that is called one-time string initializing.  
  
6. For element property binding demo we are using <img> element. The component property website.logo is getting bound to DOM property src of <img> element.  
  
7. For directive property binding we are using NgClass. We have two CSS classes .one and .two . In <p> element we are bounding these CSS classes to NgClass directive using directive property binding.

### Property Binding Types and Syntax

Property binding is performed as one-way from data source to view target. In property binding there is source and target. For the example we can define it as **[href]="website.url"**. Here **href** is a **target** that is a property of anchor tag and **source** is a component property i.e **website.url**.  
  
**Types of Property Binding**  
In property binding there is source and target. Property binding is performed as one-way from data source to view target. There are three types of property binding.  
1. Element property  
2. Directive property  
3. Component property  
  
**Syntax**  
Property binding target will use the below syntax.  
1. Bracket []  
2. bind- prefix  
3. Interpolation {{expression}}  
We can choose any of the above syntax for property binding that fits to our readability point of view.

### Create Component, HTML Template and CSS

Find the component and its HTML template that is being used in our example.  
**site.component.ts**

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

@Component({

selector: 'app-root',

templateUrl: './site.component.html',

styleUrls: ['./site.component.css']

})

export class SiteComponent {

flag = true;

website = {

name : 'ConcretePage',

url : 'http://www.concretepage.com',

logo : '/assets/images/logo.jpg',

description: 'Learn angular 2 property binding.'

}

}

**site.component.html**

Logo: <img [src]="website.logo"/>

<br/>Logo: <img bind-src="website.logo"/>

<br/>Logo: <img src="{{website.logo}}"/>

<br/>Url: <a [href]="website.url" [textContent]="website.name"> </a>

<br/>Url: <a bind-href="website.url" bind-textContent="website.name"> </a>

<br/>Url: <a href="{{website.url}}" textContent="{{website.name}}"> </a>

<p [textContent]="website.description"> </p>

<p bind-textContent="website.description"> </p>

<p textContent="{{website.description}}"> </p>

<br/><button [disabled]="flag">Submit</button>

<br/><button bind-disabled="!flag">Submit</button>

<p [ngClass]="'one two'"> Angular 2 Property Binding Example </p>

<p bind-ngClass="'one two'"> Angular 2 Property Binding Example </p>

<p ngClass="{{'one two'}}"> Angular 2 Property Binding Example </p>

<br/><my-msg prefixMsg= "Website name is " [siteName] = "website.name"> </my-msg>

<br/><my-msg prefixMsg= "Website name is " bind-siteName = "website.name"> </my-msg>

<br/><my-msg prefixMsg= "Website name is " siteName = "{{website.name}}"> </my-msg>

**site.component.css**

.one {

color: green;

}

.two {

font-size: 20px;

}

### Component Property Binding

We will discuss here component property binding. Using component property binding parent and child component can communicate. The parent component property as source is bound to child component property as target in component property binding. We will understand component property binding step by step.  
  
**A.** In parent component site.component.ts we have a property as follows.

website = {

name : 'ConcretePage',

url : 'http://www.concretepage.com',

logo : '/assets/images/logo.jpg',

description: 'Learn angular 2 property binding.'

}

**B.** For the demo we will send website.name property to child component.  
Find the child component and its HTML template.  
**msg.component.ts**

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

@Component({

selector: 'my-msg',

templateUrl: './msg.component.html'

})

export class MsgComponent {

@Input() prefixMsg: string;

@Input() siteName: string;

}

**msg.component.html**

{{prefixMsg}} {{siteName}}

We will observe in child component that prefixMsg and siteName properties have been decorated with @Input decorator. @Input is responsible to decorate a component property as input property that will be bound with parent component property to accept values from parent.  
  
**C.** Now for component property binding, in site.component.html we are creating a tag with my-msg that is selector of child component. Component property binding can be performed in following ways.  
**1.**Using bracket []

<my-msg [siteName] = "website.name"> </my-msg>

**2.** Using bind-

<my-msg bind-siteName = "website.name"> </my-msg>

**3.** Using Interpolation

<my-msg siteName = "{{website.name}}"> </my-msg>

**D.** In this way website.name property value from site.component.ts has been copied into siteName of msg.component.ts. This is component property binding.  
  
**E. One-time string initialization**  
We can also perform one-time string initialization. In this type of initialization we do not use [ ], bind- or interpolation. But we need to take care that target will accept string only which is fixed and will not change. In the below code snippet prefixMsg is being used as one-time string initialization.

<my-msg prefixMsg= "Website name is " > </my-msg>

<my-msg prefixMsg= "Website name is " [siteName] = "website.name"> </my-msg>

### Element Property Binding

Here we will discuss HTML element property binding.  
**1. Using bracket [ ]**  
Find the property binding in HTML element <img>, <a>, <p> and <button>.

<img [src]="website.logo"/>

<a [href]="website.url" [textContent]="website.name"> </a>

<p [textContent]="website.description"> </p>

<button [disabled]="flag">Submit</button>

The targets such as src, href, textContent and disabled properties has been enclosed within bracket [ ]. Source is a component property. The **src** of <img> tag has been assigned with the value of website.logo. In the same way **href** of anchor tag has been assigned with value of website.url and **textContent** has been assigned with the value of website.name and same in <p> tag. In Angular framework **disabled** property can be assigned with true and false value which in normal HTML coding it does not work.  
  
**2. Using bind-**  
Find the element property using **bind-**. We just have to add it as prefix with element properties as follows.

<img bind-src="website.logo"/>

<a bind-href="website.url" bind-textContent="website.name"> </a>

<p bind-textContent="website.description"> </p>

<button bind-disabled="!flag">Submit</button>

**3. Using Interpolation**  
Now find the element property binding using interpolation.

<img src="{{website.logo}}"/>

<a href="{{website.url}}" textContent="{{website.name}}"> </a>

<p textContent="{{website.description}}"> </p>

### Directive Property Binding

Property binding can also be achieved for Angular directives. In our example we are using ngClass directive. We are assigning CSS classes. For the example we have two CSS classes i.e. .one and .two.  
**1. Using bracket [ ]**  
Find the directive property binding using [ ].

<p [ngClass]="'one two'"> Angular 2 Property Binding Example </p>

**2. Using bind-**  
Find the directive property binding using **bind-**.

<p bind-ngClass="'one two'"> Angular 2 Property Binding Example </p>

**3. Using Interpolation**  
Find the directive property binding using interpolation.

<p ngClass="{{'one two'}}"> Angular 2 Property Binding Example </p>

### Difference between HTML attribute and DOM property

Template binding works with properties and events but not attributes. In angular framework, attribute binding is achieved using prefix **attr.** such as **[attr.colspan]**. Let us understand the differences between HTML attribute and DOM property.  
1. Attributes are defined by HTML and properties are defined by DOM.  
2. The responsibility of HTML attributes is just to initialize DOM properties. And then later DOM properties can change but HTML attributes cannot.  
3. There are some HTML attributes that have corresponding DOM property and those DOM properties have corresponding HTML attributes such as **id**.  
4. There are also some HTML attributes that do not have corresponding DOM property such as **colspan**.  
5. There are also some DOM properties that do not have corresponding HTML attributes such as **textContent**.  
6. The HTML attribute **value** contains initial value whereas DOM property **value** contains current value.

### Create Module

**app.module.ts**

import {NgModule} from '@angular/core';

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

import {SiteComponent} from './site.component';

import {MsgComponent} from './msg.component';

@NgModule({

imports: [BrowserModule],

declarations: [SiteComponent, MsgComponent],

bootstrap: [SiteComponent]

})

export class AppModule { }

### Run Application

To run the application, find the steps.  
**1.** Download source code using download link given below on this page.  
**2.** Use downloaded **src** in your Angular CLI application. To install Angular CLI, find the [link](https://angular.io/start).  
**3.** Run **ng serve** using command prompt.  
**4.** Access the URL **http://localhost:4200**  
Find the print screen of the output.



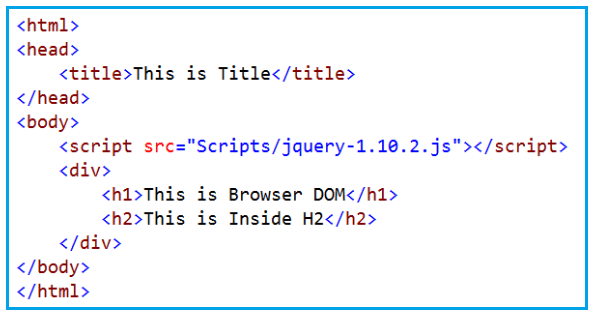
HTML Attribute VS DOM Property

**HTML Attribute VS DOM Property with Example**

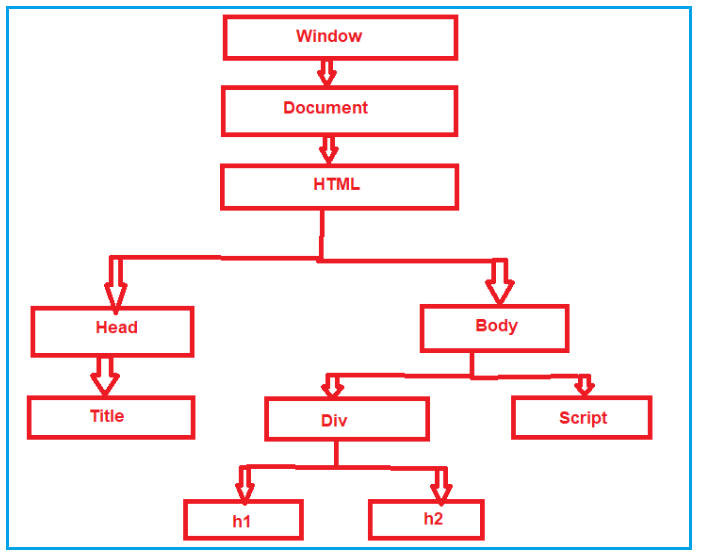
I am going to discuss the **HTML Attribute VS DOM Property** with an example. Please read our previous article where we discussed [**Angular Property Binding**](https://dotnettutorials.net/lesson/angular-property-binding/) in detail. At the end of this article, you will understand what exactly DOM is and the difference between HTML Attribute and DOM Property.

**What is DOM?**

The DOM stands for Document Object Model. When a browser loads a web page, then the browser creates the Document Object Model (DOM) for that page. For example, let say we have a page with the following HTML.



When the above HTML is loaded by the browser, then it creates the Document Object Model (DOM) as shown in the below image.



So in simple words, we can say that the DOM is an application programming interface (API) for the HTML, and we can use the programming languages like JavaScript or JavaScript frameworks like Angular to access and manipulate the HTML using their corresponding DOM objects.

In other words, we can say that the DOM contains the HTML elements as objects, their properties, methods, and events and it is a standard for accessing, modifying, adding or deleting

**Interpolation example: <button disabled='{{IsDisabled}}’>Click Me</button>**

**Property binding example: <button [disabled]=’IsDisabled’>Click Me</button>**

If you look at the above two examples, you may feel that you are binding to the Button’s disabled attribute, but that is not true.

You are actually binding to the disabled property of the button object.

* **So,** **the Angular data-binding is all about binding to the DOM object properties and not the HTML element attributes**.

**What is the difference between the HTML element attribute and DOM property?**

1. The Attributes are defined by HTML whereas the properties are defined by the DOM.
2. The attribute’s main role is to initializes the DOM properties. So, once the DOM initialization complete, the attributes job is done.
3. Property values can change, whereas the attribute values can never be changed.

**Let’s prove this –** The Property values can change, whereas the attribute values can neven be changed with an example. In the below example, we have set the value attribute of the input HTML element to Anurag.

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

@Component**({**

selector: 'app-root',

template: `<div>

<input id='inputId' type='text' value='Anurag'>

</div>`

**})**

**export** **class** AppComponent **{**

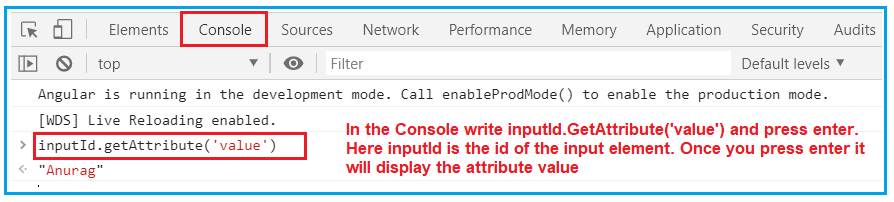
**}**

At this point, run the application and you will see the value Anurag in the textbox as expected as shown in the below image.

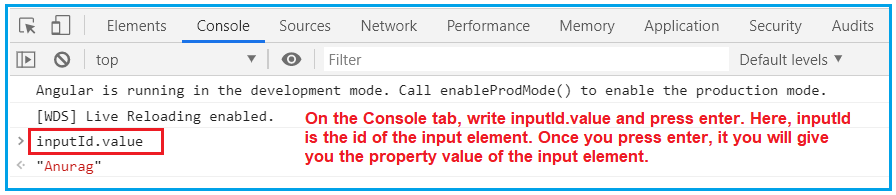
HTML Attribute VS DOM Property

**Getting the Attribute and Property Value**

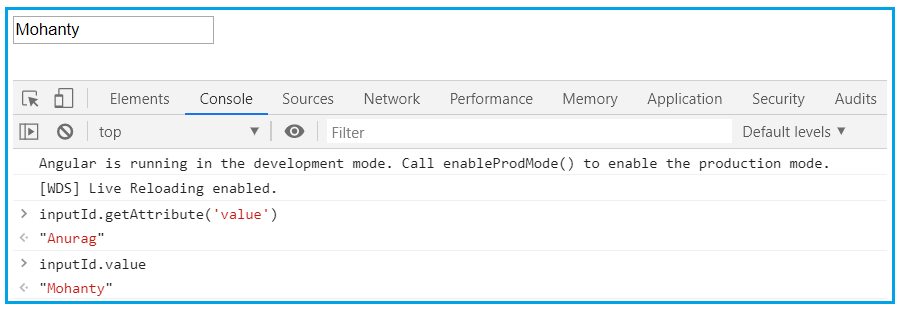
Now, launch the browser developer tools by pressing the F12 key and then click on the Console Tab. On the ‘**Console**‘ tab, use the **getAttribute()** method and the **value** property of the input element to get the attribute and property values. Notice at the moment both have the value Anurag. The **GetAttribute()** Method provides you with the attribute value of the HTML Element.



The **Value** Property of the DOM object provides you the property value as shown in the below image.



Now, change the value in the textbox to Mohanty on the webpage. And do the same thing once more as shown below.



Now, when we query for the attribute and property values, the attribute value is still Anurag but the property value is changed to Mohanty.

* So this proves that the Property values changes whereas the attribute values never changed.
* **So, the points to remember is that,**

1. The HTML attributes and DOM properties are two different things.
2. ***Angular binding works with the properties and events,*** and not with the attributes.
3. The job of attributes is to initialize the DOM object properties.

Angular Attribute Binding

**Angular Attribute Binding with Examples**

1. **What is Attribute Binding in Angular?**
2. **Why we need Attribute Binding in Angular Application?**
3. **Understanding Angular Attribute Binding with an Example**
4. **Using Angular Interpolation**
5. **Using Property Binding and Attribute Binding in Angular**

**Why we need Attribute Binding in Angular Application?**

* **,** we discussed that they both (Interpolation and Property Binding) are dealing with the DOM Properties but not with the HTML attributes
* . But there are some HTML elements (such as colspan, area, etc) that do not have the DOM Properties.
* Now the question is how to deal with such elements that do not have DOM Properties as we can’t use Interpolation and propertyBinding

. The Answer is Angular Attribute Binding.

With Attribute Binding in Angular, you can set the value of an HTML Element Attribute directly. So, the Attribute Binding is used to bind the attribute of an element with the properties of a component dynamically. Suppose we have defined a property clspn with some numeric value in our component.

clspn = 2;

1. Find the attribute binding using bracket [].

<td [attr.colspan]="clspn">

2. Attribute binding using bind- keyword.

<td bind-attr.colspan = "clspn">

3. Attribute binding using interpolation.

<td attr.colspan = "{{clspn}}">

### Complete Example

**app.component.ts**

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html'

})

export class AppComponent {

h = 300;

w = 200;

bdr = 5;

clspn = 2;

}

**app.component.html**

<table [border]="bdr" [attr.height]="h" [width]="w">

<tr>

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

</tr>

<tr>

<td> C </td>

<td> D </td>

</tr>

</table>

In the above code border and width are using element property binding and height and colspan are using attribute binding.  
  
**app.module.ts**

import {NgModule} from '@angular/core';

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

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

@NgModule({

imports: [BrowserModule],

declarations: [AppComponent],

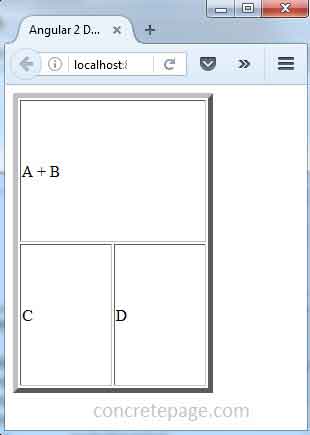
bootstrap: [AppComponent]

})

export class AppModule { }

### Run Application

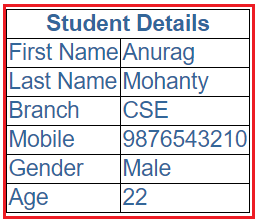
To run the application, find the steps.  
**1.** Download source code using download link given below on this page.  
**2.** Use downloaded **src** in your Angular CLI application. To install Angular CLI, find the [link](https://angular.io/start).  
**3.** Run **ng serve** using command prompt.  
**4.** Access the URL **http://localhost:4200**  
Find the print screen of the output.



### Reference

**Understanding Angular Attribute Binding with an Example:**

Let us understand the need and use of Angular Attribute Binding with an example. We want to do here is, we want to show the Details of a Student on a webpage as shown in the below image.



Let us see how to achieve this step by step.

**Step1: Modify the app.component.css file**

First, open app.component.css file which you can find inside the app folder and then copy and paste the following styles into it. These are the styles that we want to apply to the table.

table **{**

color: *#369*;

font-family: Arial, Helvetica, sans-serif;

font-size:large;

border-collapse: collapse;

**}**

td **{**

border: 1px solid black;

**}**

thead**{**

border: 1px solid black;

**}**

**Step2: Modify the app.component.html file**

Open **app.component.html** file and then copy and paste the following code in it. You can also find this file inside the app folder. Here, we have created one table to display the student details. Again here we are using angular interpolation i.e. double curly braces to bind the student data.

**<table>**

**<thead>**

**<tr>**

**<th** colspan="2"**>**

{{pageHeader}}

**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr>**

**<td>**First Name**</td>**

**<td>**{{FirstName}}**</td>**

**</tr>**

**<tr>**

**<td>**Last Name**</td>**

**<td>**{{LastName}}**</td>**

**</tr>**

**<tr>**

**<td>**Branch**</td>**

**<td>**{{Branch}}**</td>**

**</tr>**

**<tr>**

**<td>**Mobile**</td>**

**<td>**{{Mobile}}**</td>**

**</tr>**

**<tr>**

**<td>**Gender**</td>**

**<td>**{{Gender}}**</td>**

**</tr>**

**<tr>**

**<td>**Age**</td>**

**<td>**{{Age}}**</td>**

**</tr>**

**</tbody>**

**</table>**

**Step3: Modify the app.component.ts file**

Modify the **app.component.ts** file as shown below. Here within the component decorator, we are providing the stylesUrl and templateUrl values to the external files that we just modified in our previous two steps. Then as part of the Component class, we are creating the required student properties with some value as well as the header value.

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

@Component**({**

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: **[**'./app.component.css'**]**

**})**

**export** **class** AppComponent **{**

pageHeader: **string** = 'Student Details';

FirstName: **string** = 'Anurag';

LastName: **string** = 'Mohanty';

Branch: **string** = 'CSE';

Mobile: **number** = 9876543210;

Gender: **string** = 'Male';

Age: **number** = 22;

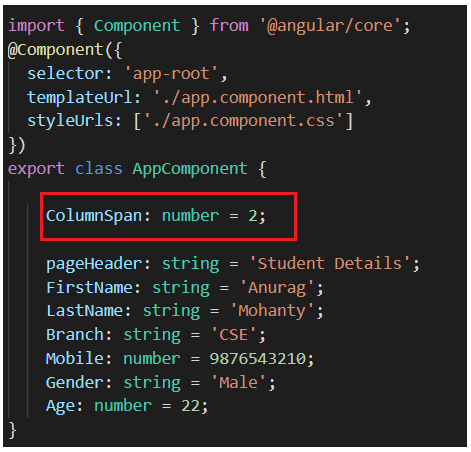
**}**

Now, with the above changes in place, if you browse the application (**ng serve -o**), you will get the output as expected.

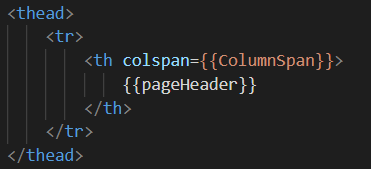
**Using Angular Interpolation:**

If you notice, here we have hardcode the colspan value to 2 in the HTML file i.e. in the app.component.html file.

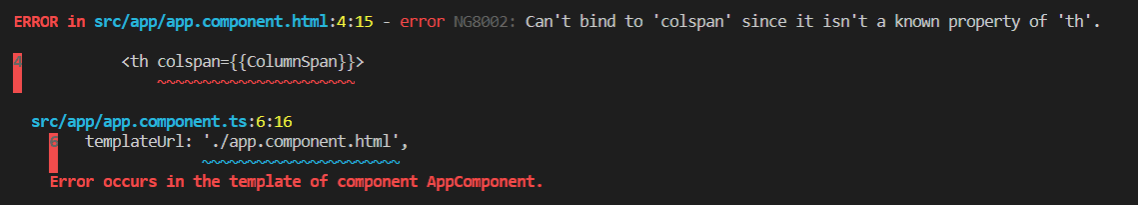
Let us say we also want to bind this (colspan) value from a property of the component class. To do so, let us add a new property with the name ColumnSpan in the AppComponent (app.component.ts file) class as shown in the below image.



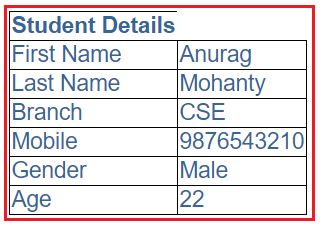
Now, you may be intended to use the ColumnSpan property to set the colspan attribute of the th element using angular interpolation as shown in the below image.



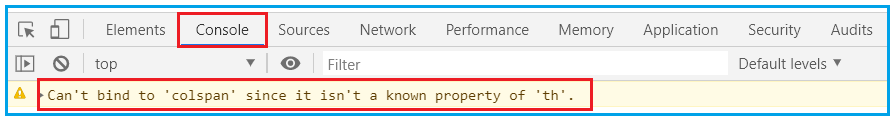
Once you compiled, you will get the following error.



Now, have a look at the browser, and you will not get the output as expected as shown in the below image. The colspan is not working here.



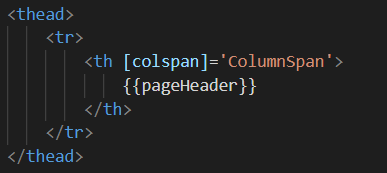
In order to see what went wrong, launch the browser developer tools by pressing the F12 key and open the Console tab as shown in the below image.



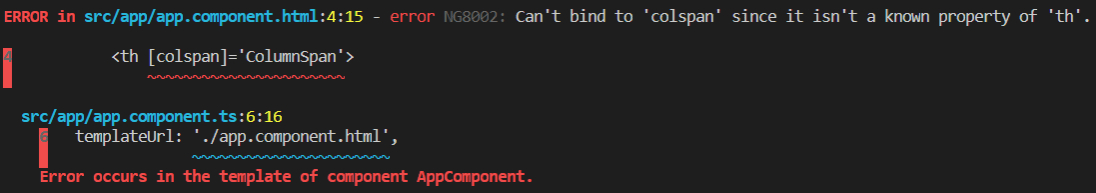
It clearly shows that you “**Can’t bind to ‘colspan’ since it isn’t a known property of ‘th’**”. So, you cannot use angular interpolation to bind the colspan attribute of th element.

**Using Property Binding in Angular:**

You will also get the same error if you try to bind the colspan property using the Property Binding Technique. Let us prove this. First, modify the colspan binding in the app.component.html file as shown below.



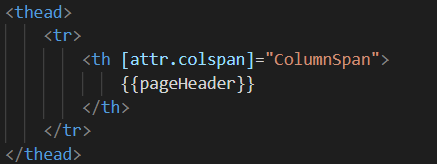
With the above Property Binding in place, now if you compile the project, then you will get the same error as shown in the below image.



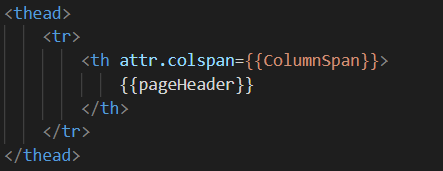
We get the above error. The reason for this error is, we do not have a corresponding property in the DOM for the colspan attribute. In order to solve the above error, what you need to do is you need to use Angular Attribute Binding, which will set the colspan attribute value.

**Using Attribute Binding in Angular:**

To tell the angular framework that we are setting an attribute value, we have to prefix the attribute name with the **attr** and a DOT as shown below.



The same is also true when we are using angular interpolation as shown in the below image.



Now run the application and you should get the output as expected.

**Note:** The Angular team recommends using the property binding or Interpolation whenever possible and use the attribute binding only when there is no corresponding element property to bind.

### Differences between HTML Attribute and DOM Property

Angular 2 attribute binding is required in those cases when HTML attribute has no corresponding DOM property. Find some differences between HTML attribute and DOM property.  
  
1. Attributes are defined by HTML and properties are defined by DOM.  
  
2. The responsibility of HTML attributes is just to initialize DOM properties. And then later DOM properties can change but HTML attributes cannot.  
  
3. There are some HTML attributes that have corresponding DOM property and those DOM properties have corresponding HTML attributes such as **id**.  
  
4. There are also some HTML attributes that do not have corresponding DOM property such as **colspan**.  
  
5. There are also some DOM properties that do not have corresponding HTML attributes such as **textContent**.  
  
6. The HTML attribute **value** contains initial value whereas DOM property **value** contains current value.

Angular Class Binding

***Angular Class Binding with Examples***

going to discuss the **Angular Class Binding** with examples. Please read our previous lab where we discussed [**Angular Attribute Binding**](https://dotnettutorials.net/lesson/angular-attribute-binding/) in detail. At the end of this, you will understand what exactly Angular Class Binding is and when and how to use Class Binding in Angular Application.

* **What is Angular Class Binding?**

The Angular Class Binding is basically used to add or remove classes to and from the HTML elements.

It is also possible in Angular to add CSS Classes conditionally to an element, which will create the dynamically styled elements and this is possible because of Angular Class Binding.

**Understanding Class Binding in Angular Application:**

Let us understand Angular Class Binding with an example. First, open the **styles.css** file and then copy and paste the following code in it. You can find styles.css file within the src folder of your project.

**.boldClass{**

font-weight:bold;

**}**

**.italicClass{**

font-style:italic;

**}**

**.colorClass{**

color:red;

**}**

**Note:** Please make sure to have a reference to this **styles.css** file in the **index.html** file which is your host page as shown in the below image.



***Modifying app.component.ts file:***

Modify the app.component.ts file as shown below. Here we are just adding a button element as well as we also set the class attribute of the button element to ‘**colorClass**‘. If you remember this **colorClass** class is defined in the styles.css file.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button class='colorClass'>Click Me</button>

</div>`

**})**

**export** **class** AppComponent **{**

**}**

Now run the application and you should see the **‘colorClass’** is applied to the button element as expected as shown in the below image.

without using Class Binding in Angular

**Using Class Binding in Angular:**

Let us see how to use class binding in angular. To do so, first, modify the **app.component.ts file** as shown below. Here, as you can see, we have created a property called **ClassesToApply** in the **AppComponent** class. We have also specified the class binding for the button element. The word ‘**class**‘ is in a pair of square brackets and it is bound to the property ‘**ClassesToApply**‘.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button [class] = 'ClassesToApply' >Click Me</button>

</div>`

**})**

**export** **class** AppComponent **{**

ClassesToApply : **string** = 'italicClass boldClass';

**}**

Once you save changes, then you will get the following output. Here you can see the button is italic as well as bold.

Using Class Binding in Angular

If you want then you can also combine both class binding with the normal class as shown in the below example.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button class='colorClass' [class] = 'ClassesToApply' >Click Me</button>

</div>`

**})**

**export** **class** AppComponent **{**

ClassesToApply : **string** = 'italicClass boldClass';

**}**

Now you should get the button with all the three CSS classes like the color red, bold and italic as shown in the below image.

Using both Class Attribute and Class Binding in Angular Application

**Adding or removing a single class:**

Suppose you want to add or remove a single class, then you need to include use the prefix ‘class’ within a pair of square brackets and followed by a DOT (.) and the name of the class that you want to add or remove.

The following example adds the boldClass to the button element. Notice it does not remove the existing colorClass which is added using the class attribute.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button class='colorClass' [class.boldClass]='ApplyBoldClass'>Click Me</button>

</div>`

**})**

**export** **class** AppComponent **{**

ApplyBoldClass: **boolean** = **true**;

**}**

Now, if run the application, then you should see the button with red color and bold. If you set the ApplyBoldClass property value to false or remove the property altogether from the AppComponent class, then the CSS class i.e. boldClass will not be added to the button element. In the following example, we have set the ApplyBoldClass property value to false and when you browse the application, you will see the button without bold.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button class='colorClass' [class.boldClass]='ApplyBoldClass'>Click Me</button>

</div>`

**})**

**export** **class** AppComponent **{**

ApplyBoldClass: **boolean** = **false**;

**}**

**Angular Class Binding using “!” symbol:**

In Angular Class Binding, it is also possible to use the “!” symbol. Notice in the following example, we have set ApplyBoldClass property value to false. As we use the “!” symbol in the class binding, the class is going to be added to the button element.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button class='colorClass' [class.boldClass]='!ApplyBoldClass'>Click Me</button>

</div>`

**})**

**export** **class** AppComponent **{**

ApplyBoldClass: **boolean** = **false**;

**}**

Now, if you run the application, then you should the bold class is applied to the button element as expected.

**How to remove an existing class using Class Binding in Angular?**

Let us understand this with an example i.e. how to remove an existing class that is already applied by using Angular Class Binding. Consider the below example. Here we have 3 classes (colorClass, boldClass, and italicsClass) added to the button element using the class attribute. Then the class binding removes the boldClass.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button class='colorClass italicClass boldClass' [class.boldClass]='ApplyBoldClass'>Click Me</button>

</div>`

**})**

**export** **class** AppComponent **{**

ApplyBoldClass: **boolean** = **false**;

**}**

That’s fine. But if we want to add or remove multiple classes then how we can do this?

**Add or Remove multiple classes in Angular:**

In order to add or remove multiple style classes in angular, the angular framework provides one directive called ngClass directive which you can use to remove or add multiple classes as shown in the below example. What is a directive and what are the different types of directives that we will discuss in detail in our upcoming articles. For now, let us see how to use the ngClass directive to add or remove multiple classes.

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

@Component({

selector: 'app-root',

template: `**<div>**

**<button** class='colorClass' [ngClass]='AddCSSClasses()'**>**Click Me**</button>**

**</div>**`

})

export class AppComponent {

ApplyBoldClass: boolean = true;

ApplyItalicsClass: boolean = true;

AddCSSClasses() {

let Cssclasses = {

boldClass: this.ApplyBoldClass,

italicsClass: this.ApplyItalicsClass

};

return Cssclasses;

}

}

**What we are doing here?**

1. Here the colorClass is added using the class attribute
2. The ngClass directive is bind to the AddCSSClasses() method of the AppComponent class
3. Here, AddCSSClasses() is method that returns an object with 2 key/value pairs. The key is the CSS class name and the value can be true or false. If the value is true then it will add the class and when the value is False then it will remove the class.
4. Since both the keys (boldClass & italicsClass) are set to true, both classes will be added to the button element
5. The **let**is a new type of variable declaration in JavaScript. **let**is similar to var in some respects. In our upcoming article, we will discuss let and var in detail as well as discuss the differences between them. For this example, you can also use var and it should work.

### Create CSS Classes

Find the CSS classes which we will use in our example.

.required{

color: green;

font-size: 30px;

}

.optional {

color: red;

background-color: cyan;

font-family: cursive;

}

### CSS Class Binding

Class binding is same as property binding with the difference that we need to prefix class name with class. . The syntax of using class binding is as follows.

<div [class.required]="isReq">Hello Wordl!</div>

Here required is a CSS class name. We need to prefix it with class. . Here we are using bracket **[]** for class binding. We can also use **bind-** keyword with the target. Class binding also works with interpolation. Find the code snippet.

<div bind-class.required ="isReq">Hello Wordl!</div>

<div class.required ="{{isReq}}">Hello Wordl!</div>

In the above code snippet first line is using **bind-** keyword and second line is using interpolation. We have defined isReq a component property which is Boolean. In the above class binding we control our class dynamically. If the value of isReq is **true** only when the CSS class required will be applied in the <div> element.  
In normal HTML coding CSS class is used as follows.

<div class="required">Hello Wordl!</div>

Now to understand more about class binding, I am creating a method in component as follows.

isOptional(data) {

if (data == 'yes') {

return true;

} else {

return false;

}

}

In the above method when we pass "yes", method will return true otherwise false. We will discuss different scenarios now. Here we are using HTML element class attribute as well as angular class binding. We are adding two CSS classes that is required and optional .  
1. The component method isOptional('yes') returns true, so <div> tag will have both the CSS classes. The CSS properties of optional class will override the CSS properties of required class if there is any common property.

<div class="required" [class.optional]="isOptional('yes')">Hello Wordl!</div>

2. Here optional class will not include in <div> because isOptional('no') will return false value.

<div class="required" [class.optional]="isOptional('no')">Hello Wordl!</div>

3. Here we consider a scenario in which we are using both CSS classes using regular HTML element class attribute. Now using angular class binding we can on/off CSS classes. If isOptional() method returns true, then both the CSS classes will work and if isOptional() method returns false, then optional CSS class will be removed.

<div class="required optional" [class.optional]="isOptional('no')">Hello Wordl!</div>

In above code only required CSS class will work.

### Complete Example

**app.component.ts**

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

isReq = true;

isOptional(data) {

if (data == 'yes') {

return true;

} else {

return false;

}

}

}

**app.component.css**

.required{

color: green;

font-size: 30px;

}

.optional {

color: red;

background-color: cyan;

font-family: cursive;

}

**app.component.html**

<div class="required">Hello Wordl!</div>

<div [class.required]="isReq">Hello Wordl!</div>

<div class="required" [class.optional]="isOptional('yes')">Hello Wordl!</div>

<div class="required" [class.optional]="isOptional('no')">Hello Wordl!</div>

<div class="required optional" [class.optional]="isOptional('no')">Hello Wordl!</div>

**app.module.ts**

import {NgModule} from '@angular/core';

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

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

@NgModule({

imports: [BrowserModule],

declarations: [AppComponent],

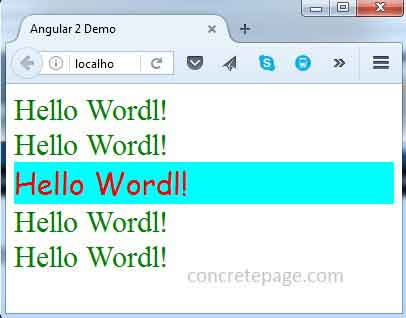
bootstrap: [AppComponent]

})

export class AppModule { }

### Run Application

To run the application, find the steps.  
**1.** Download source code using download link given below on this page.  
**2.** Use downloaded **src** in your Angular CLI application. To install Angular CLI, find the [link](https://angular.io/start).  
**3.** Run **ng serve** using command prompt.  
**4.** Access the URL **http://localhost:4200**  
Find the print screen of the output.



**Inline style binding in Angular is similar to interpolation**

**{{}}**

**syntax:**

* **[style.style-property]="style-value";**
* **style binding uses [] place the css style property inside the square brackets**
* **css sytle must begin with style keyword followed by a dot and then style name**

Here style is a prefix and style-property is a name of a CSS style property. style prefix and style property are concatenated using dot (.) . Style property binding can be achieved with bracket **[]**, **bind-** keyword and interpolation **{{}}**. Now find the code to use style binding.

<p [style.color] = "result > 30 ? 'blue' : 'green'"> Hello Color World! </p>

<p bind-style.color = "result > 30 ? 'blue' : 'green'"> Hello Color World! </p>

<p style.color = "{{result > 30 ? 'blue' : 'green'}}"> Hello Color World! </p>

* In our style binding example we are using conditional operator. When the condition result > 30 returns **true** then **color** style will be set to **blue** otherwise **green**. Let us know about conditional operator.  
    
  **Conditional (ternary) Operator**  
  Conditional operator that is also called ternary operator, is used as a short cut of if else statement. Find the syntax as below.  
  condition ? expr1 : expr2  
  When the **condition** is true then **expr1** will be returned otherwise **expr2** will be returned.

**Example:<p [style.color**]="red">giveme red</p>

We can apply multiple styles to html element in Angular

Template:

<button [style.background-color]='getcolor()' [style.font.size.px]="5">click me  </button>

  </div>`

  ,

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'Welcome to First Angular App';

  typesofDatabinding='OnewayDataBinding'

  getcolor()

  {

      return "red";

  }

### Complete Example

**app.component.ts**

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html'

})

export class AppComponent {

result = 50;

colorFlag = false;

isSmall = true;

isBackgroundRed = false;

small = 10;

big = 15;

num = 10;

isRed(num) {

if (num > 10) {

return false;

} else {

return true;

}

}

allRequiredStyles(styleSet) {

let myStyles;

if(styleSet == 'one') {

myStyles = {

'color': this.colorFlag ? 'black' : 'yellow',

'font-size.em': this.isSmall ? this.small/5 : this.big/5,

'background-image': !this.isBackgroundRed ? 'url(\'/assets/images/red.gif\')' : 'url(\'/assets/images/green.gif\')'

};

} else if(styleSet == 'two') {

myStyles = {

'color': !this.colorFlag ? 'black' : 'yellow',

'font-size.em': !this.isSmall ? this.small/5 : this.big/5,

'background-image': this.isBackgroundRed ? 'url(\'/assets/images/red.gif\')' : 'url(\'/assets/images/green.gif\')'

};

} else {

myStyles = {

'background-color': this.colorFlag ? 'cyan' : 'grey',

'font-size.%': !this.isSmall ? this.small \* 10: this.big \* 10

};

}

return myStyles;

}

}

**app.component.html**

<p [style.color] = "result > 30 ? 'blue' : 'green'"> Hello Color World! </p>

<p bind-style.color = "result > 30 ? 'blue' : 'green'"> Hello Color World! </p>

<p style.color = "{{result > 30 ? 'blue' : 'green'}}"> Hello Color World! </p>

<button

[style.background-color] = "colorFlag ? 'cyan' : 'grey'"

[style.color] = "isRed(8) ? 'red' : 'black'" > Button </button>

<div [style.font-size.em] = "isSmall ? small/8 : big/8" > Font Size Test with em</div>

<div [style.font-size.px] = "!isSmall ? small : big" > Font Size Test with px</div>

<div [style.font-size.pt] = "isSmall ? small : big" > Font Size Test with pt</div>

<div [style.font-size.%] = "!isSmall ? small \* 10: big \* 10" > Font Size Test with %</div>

<div [style.background-image] = "isBackgroundRed ? 'url(\'/assets/images/red.gif\')' : 'url(\'/assets/images/green.gif\')'">

Style Binding Example

</div>

<br/><br/>

<div [ngStyle]="allRequiredStyles('one')">

NgStyle Example

</div>

<br/><br/>

<div bind-ngStyle="allRequiredStyles('two')">

NgStyle Example

</div>

**app.module.ts**

import {NgModule} from '@angular/core';

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

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

@NgModule({

imports: [BrowserModule],

declarations: [AppComponent],

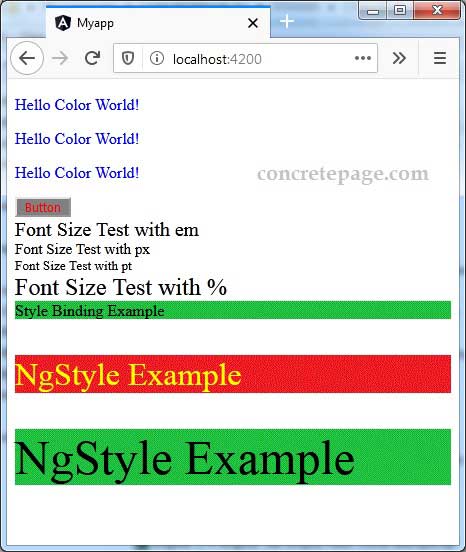
bootstrap: [AppComponent]

})

export class AppModule { }

### Run Application

To run the application, find the steps.  
**1.** Download source code using download link given below on this page.  
**2.** Use downloaded **src** in your Angular CLI application. To install Angular CLI, find the [link](https://angular.io/start).  
**3.** Run **ng serve** using command prompt.  
**4.** Access the URL **http://localhost:4200**  
Find the print screen of the output.



Angular Style Binding

**Angular Style Binding with Examples**

I am going to discuss the **Angular** **Style Binding**with examples. Setting inline styles with style binding is very much similar to setting CSS classes with class binding.

So, please read [**Angular Class Binding**](https://dotnettutorials.net/lesson/angular-class-binding/) before proceeding to this lab., you will understand what exactly Angular Style Binding is and when and how to use style binding in Angular Application.

**What is Angular Style Binding?**

The Angular Style Binging is basically used to set the style in HTML elements. You can use both inline as well as Style Binding to set the style in the element in Angular Applications. Here, in this article, I will show how to use both inline as well as style binding to style the HTML Elements with examples.

**Understanding Style Binding in Angular:**

Let’s understand how to use style binding as well as Inline style to style the HTML Elements. In the following example, we use inline style set the font color of the button element i.e. using the style attribute of the HTML Element.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button style='color:red'>Click Me</button>

</div>`

**})**

**export** **class** AppComponent **{**

**}**

When you run the application, it will display the button in red color as expected as shown in the below image.

Angular Style Binding with Examples

In the below example, it will set the style (font-weight). If the property **‘IsBold’** (this boolean property is defined in the AppComponent class) is true, then the font-weight style is set to bold else it is set to normal.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button style='color:red' [style.font-weight]="IsBold ? 'bold' : 'normal'">Click Me

</button>

</div>`

**})**

**export** **class** AppComponent **{**

IsBold: **boolean** = **true**;

**}**

The style property name can be written in either dash-case or camelCase. For example, the font-weight style can also be written using camel case **fontWeight** as shown in the below example.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button style='color:red' [style.fontWeight]="IsBold ? 'bold' : 'normal'">Click Me

</button>

</div>`

**})**

**export** **class** AppComponent **{**

IsBold: **boolean** = **true**;

**}**

Some styles like font-size have a unit extension. To set the font-size in pixels, you need to use the following syntax. This example sets font-size to 40 pixels.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button style='color:red' [style.font-size.px]="FontSize">Click Me

</button>

</div>`

**})**

**export** **class** AppComponent **{**

FontSize: **number** = 40;

**}**

**Multiple Inline Styles in Angular Application:**

If you want to set multiple inline styles in the angular application, then you need to use **NgStyle** directive as shown in the below example.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button style='color:red' [ngStyle]="AddCSSStyles()">Click Me </button>

</div>`

**})**

**export** **class** AppComponent **{**

IsBold: **boolean** = **true**;

FontSize: **number** = 40;

IsItalic: **boolean** = **true**;

AddCSSStyles**()** **{**

let CssStyles = **{**

'font-weight': this.IsBold ? 'bold' : 'normal',

'font-style': this.IsItalic ? 'italic' : 'normal',

'font-size.px': this.FontSize

**}**;

**return** CssStyles;

**}**

**}**

**What we did here?**

1. Here, we add the color style using the style attribute
2. The ngStyle is bounded to **AddCSSStyles()** method of the AppComponent class. In our [**Angular ngStyle Directive**](https://dotnettutorials.net/lesson/angular-ngstyle-directive/) article, we will discuss this ngStyle in detail.
3. The **AddCSSStyles()** method returns an object with 3 key/value pairs. The key is a style name, and the value is a value for the respective style property or an expression that returns the style value.
4. The let is a new type of variable declaration in JavaScript. Instead of let you can also use var here.

**How To Apply Conditional Styles to Components Using Angular Declarations**

If you need to apply dynamic styles to a HTML element in your Angular applications, there are a few different options available, including ngClass and ngStyle. In today’s article, we’ll explore both in detail.

Component Styling Using ngClass

Most of the time, we want to apply component styles at all times. We can do that by simply adding classes to our templates, like so:

<label>A Bootstrap Button

<button class="btn btn-primary">Click me</button>

</label>

And then, there are styles that we want to apply conditionally based a certain condition being met. For times such as these, when an element’s style may have multiple states, the ngClass directive is just the trick!

The ngClass directive will take an expression that can be one of three types:

* an object
* an array
* a string

Let’s go over each one of the above three cases with examples.

We can toggle CSS classes based on a condition, by passing in a JavaScript object where the keys of the object denote the class names and the values assign the conditions:

<div [ngClass]="{

'conditional-class': condition

}">

</div>

One simple use of ngClass is to assign multiple static class names all at once. To do that, we pass in an array of class names:

<div [ngClass]="['class1', 'class2']"></div>

Finally, we can use ngClass to assign multiple CSS attributes based on different conditions:

<div [ngClass]="{

'class1': condition,

'class2': !condition

}">

Adding Embedded Styles to Our Templates Using ngStyle

We can invoke a component method from the template to calculate the styles at runtime:

<label>A Bootstrap Button

<button [ngStyle]="setStyles()">Button</button>

</label>

One caveat with using the ngStyle declaration is that it causes CSS styles to be applied directly to the component. While there certainly are some valid reasons for applying directly styles to an HTML element, it’s generally a bad idea to do this and should be avoided. The reason is that embedded styles take precedence over any CSS styles except those that are marked with the “!important” qualifier.

To illustrate, take the following component:

<label>A Bootstrap Button

<button [ngStyle]="{background: 'gray'}">Button</button>

</label>

Here is the resulting HTML and embedded style:

<label>A Bootstrap Button

<button style="background: gray">Button</button>

</label>

Be that as it may, sometimes an element needs an embedded HTML style, as it’s simply not known up-front. In Angular, the ngStyle built-in core directive comes in handy.

To recap the use of the ngClass and ngStyle directives:

* For state styles that don’t have a pseudo-class selector linked to them, your best choice is to use ngClass.
* If the ngClass expressions get too big, it’s a good idea to move the calculation of the styles to the component class.
* For situations where you have a dynamically calculated embedded style, use ngStyle. This should be needed only rarely.

Conclusion

Angular offers a few different options for applying dynamic styles to page elements, including ngClass and ngStyle. Both accept a variety of arguments that should satisfy any use case that you may have.

Class Example

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

@Component({

  selector: 'app-root',

  template: `<div>

    <p>css classBinding</p>

    <br>

<button class="colorClass">classBinding  </button>

<br>

<p>inline styling</p>

<br>

<button [style.background-color]='getcolor()' [style.font.size.px]="5">click me  </button>

  </div>`

  ,

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'Welcome to First Angular App';

  typesofDatabinding='OnewayDataBinding'

  getcolor()

  {

      return "red";

  }

  employeeid:number=100;

  employeename:string='john';

  age:number=28;

  Designation:string='Manager';

  title1='Propertybinding';

  definition='property binding is enclosed in a squarebracket[]'

  +'it should match  the name of property'

  +'Binding source is enclosed in single quotes'+'whenever the'

  +'binding source changes the  view will be updated automatically'

syntax='[Binding-tartge]="bindingsource"';

isboolean=true;

greet='goodeveing';

getgreetmessage()

{

  return 2;

}

border:number=20;

}

Appcomponent.css

p {

    border-color: rebeccapurple;

}

body {

    background-color: blueviolet;

    font-family: Georgia, 'Times New Roman', Times, serif;

}

.boldClass {

    font-weight: bold;

}

.italicClass {

    font-style: italic;

}

.colorClass {

    color: red;

}

Appcomponent.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <link rel="stylesheet" href="./app.component.css">

</head>

<body>

    <h1 [innerText]="title1"></h1>

    <p>welcome to Angular</p>

    <p>{{employeeid}} </p>

    <p>{{employeename}}</p>

    <p>{{age}}</p>

    <p>{{Designation}}</p>

    <hr>

    < br>

        <p>we can not use wit h property Binding Assignments= +=,-=, keywords like new typeof instanceof etc changing expressions with; the operators increment and decrement ++ and -- can not be used bitwise bitwise operat operators such as || and &&

        </p>

        <table border="1">

            <tr>

                <td></td>

                <td>

                    col1

                </td>

                <td>

                    col2

                </td>

                <td>

                    col3

                </td>

                <td>

                    col4

                </td>

            </tr>

            <tr>

                <td [attr.colspan]="2 ">col and col2

                </td>

                <td bind.attr.colspan="getgreetmessage()">

                    col4

                </td>

            </tr>

        </table>

</body>

</html>