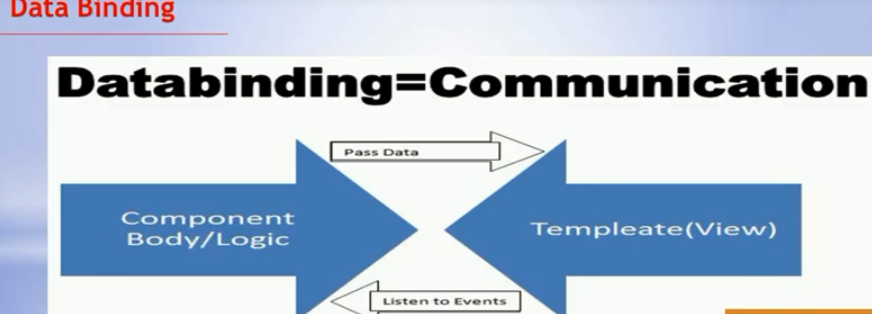
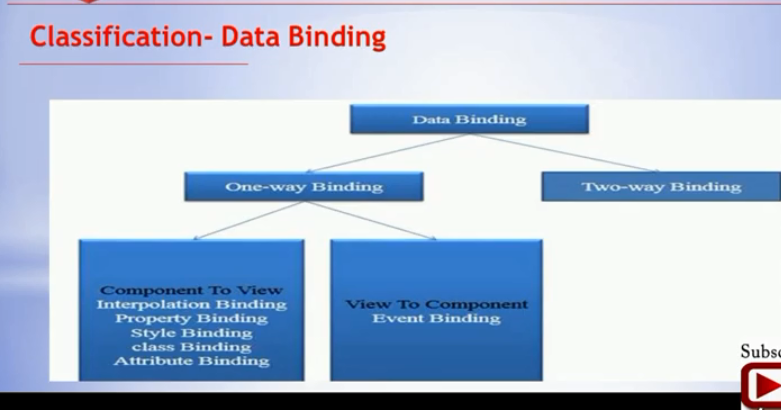
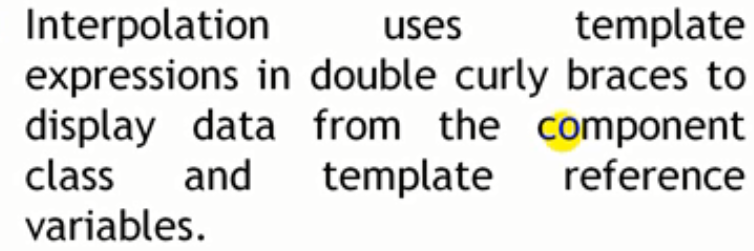
# Data binding is one of the main concepts in Angular. It helps users to establish the communication between a component and the DOM. Using data binding will help users develop interactive applications by letting users exchange data from component to DOM and vice-versa. In the following guides, will explore some of the important techniques used in data binding

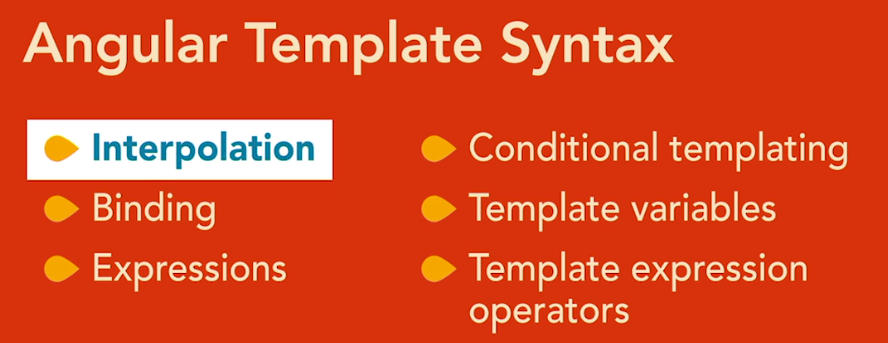
DATABINDING



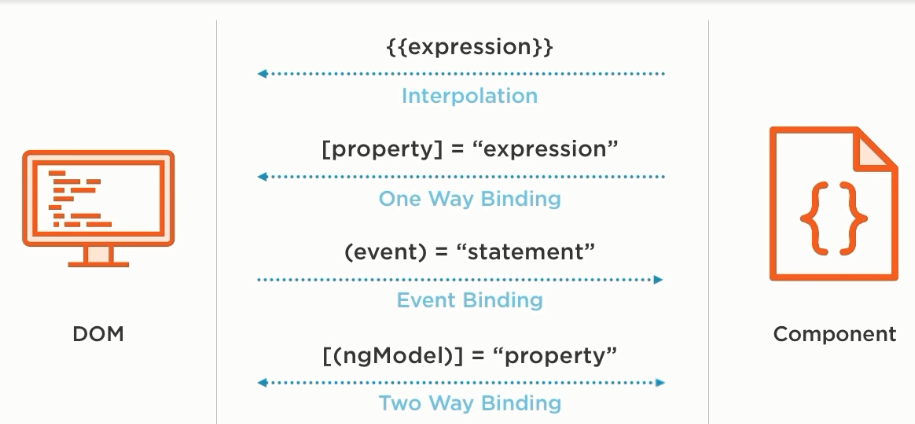


One-way and **two-way** data binding are two of the important ways by which we can exchange data from component to DOM and vice-versa. Data exchange between the component and the view will help us to build dynamic and interactive web applications.





**Syntax for different types of 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.**

**Let's consider an example using the interpolation technique where we are binding two values, firstName and the lastName, to the view, enclosed in double curly braces: {{property Name}}.**

**In this example, the data binding is done from component to the view. Any changes to the values in the component will be reflected in the view not vice-versa.**

# (1)Interpolation: {{...}}Data Binding has been a part of AngularJS since Angular 7|8|9. In the code, you go for curly braces to denote data binding – **{{ variable goes here }}** and this process is referred to as interpolation.





# Template Expression {{}}

This is the one-way binding of a component’s property to the template of the component. So, when the property in the components changes, the template is updated to reflect the new changes. For example

**Component({  
template: `  
<div>  
<h2>  
{{message}}  
</h2>  
</div>  
`  
})  
export class AppComponent {  
message = "My Message"  
}**In essence, we use the double braces {{}} to tell Angular that we want to render a Model data in the View where the {{}} is located.Interpolation helps you to bind the values to the DOM using the template expression({{}}).

The value or text between template expressions will get evaluated first and then converted to a string. The string value will then be bound to the DOM

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

@Component({

selector: 'app-example',

template: `

<div>

<strong>{{firstName}}</strong>

<strong>{{lastName}}</strong>

</div>

`

})

export class AppComponent {

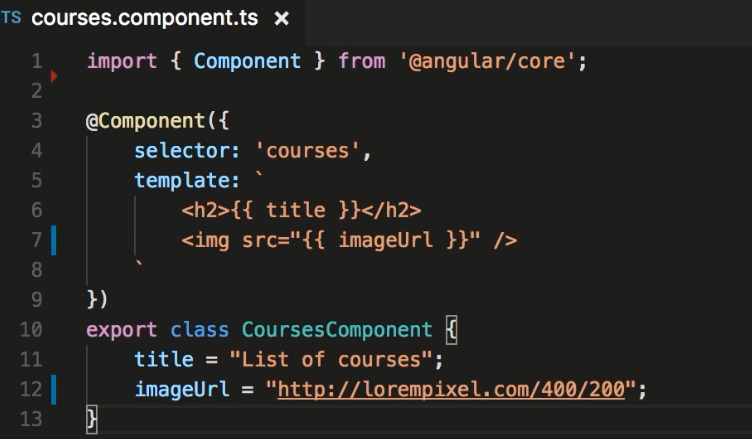
firstName: string = "Yallaling";

lastName:string = "Goudar";

}

**Typescript**

**Example 2: display image using interpolation**



**With interpolation we can use following**

**(1) Expression**

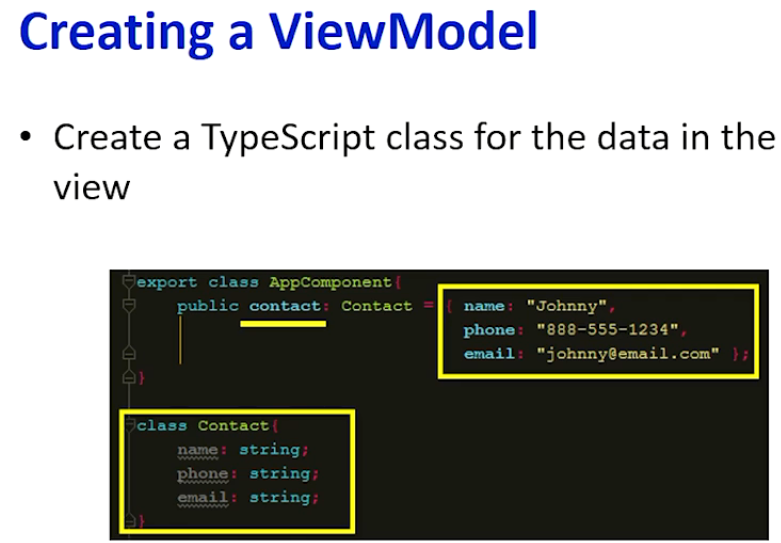
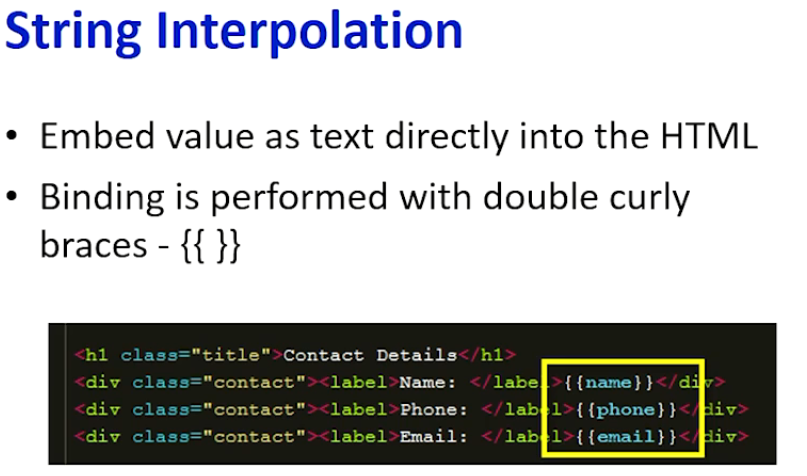
**(2). STRING Concatenations**

**(3)Java Script Properties Like LENGTH**

**(4).JS Methods LIKE ToLower() OR**

**.ToUpper().**

**5.we Can Call a Method in Component**



**STRING ConCONCATENATION**

.EXPRESSIONS

. **CALL A METHOD in componenent**

property

**JS METHODS**

**Example☹3) interpolation using a method call**

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

**@Component({**

**selector: 'app-root',**

getReversed(str: string){

let reversed = '';

for (let i=str.length-1;i>=0;i--){

reversed += str.substring(i,i+1);

}

return reversed;

}

}

Your app should be working at localhost:4200. Note how the template uses two

expressions: one to show the length of the title and another to reverse the title using a

method in the class.



**template: `**

**<h1>**

**{{title}}**

**</h1>**

**<p>**

**Length: {{title.length}}**

**</p>**

**<p>**

**Reversed: {{getReversed(title)}}**

**</p>**

**`,**

**styles: []**

**})**

**export class AppComponent {**

**title = 'welcome to app!';**

**Not all template expressions are supported the following lists are not supported. How?**

**FOREXAMPLE**

**1.Expression cant be written in Template**

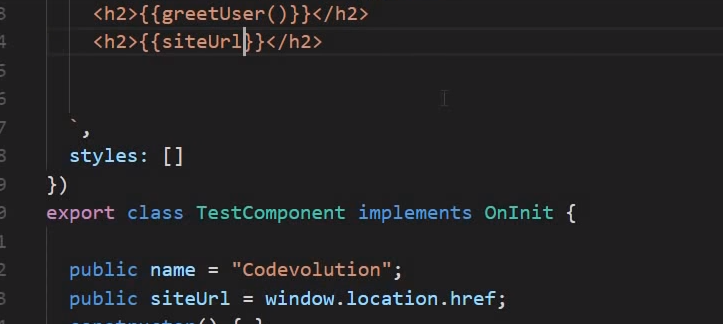
**(2).ACCESS To Global Variables Such as Window and Location not possible Exmple:so if you try to find the current url using**

**Window.location.href it throughs error**



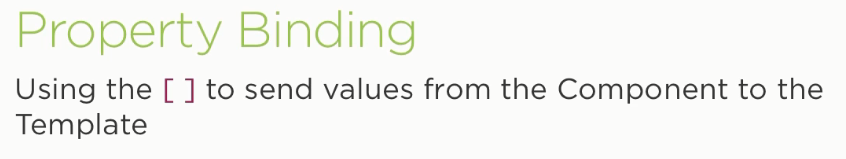


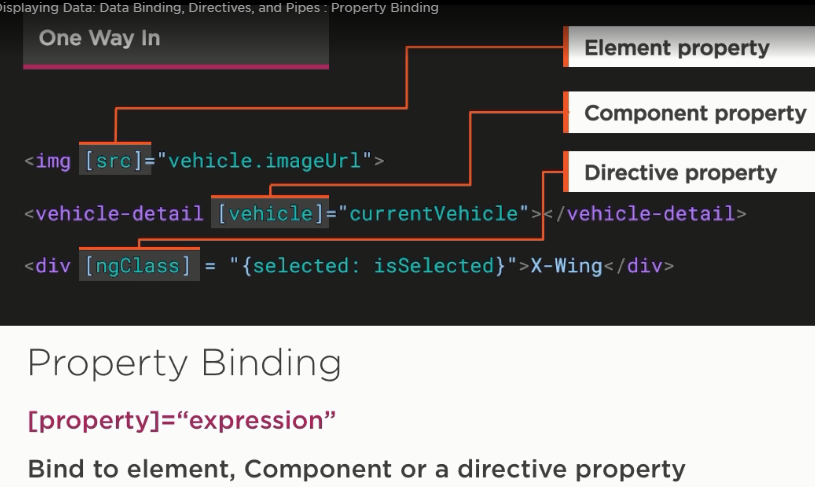
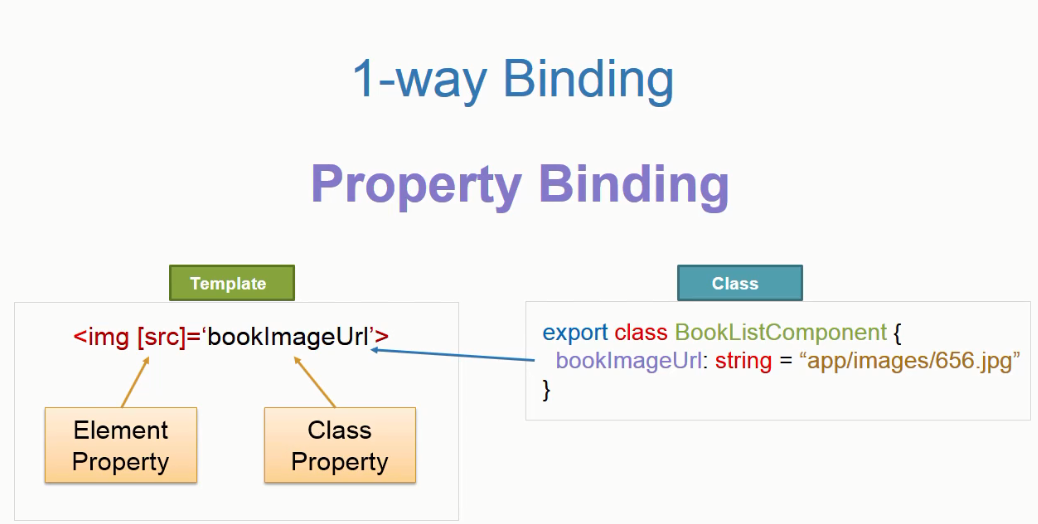
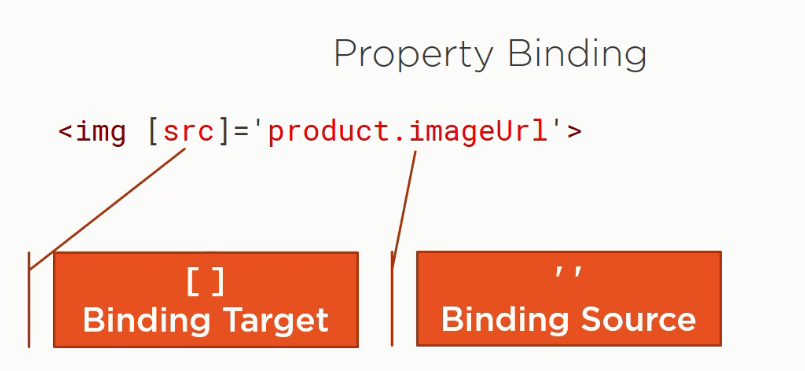
**If you want to use Global variables write them in the class and bind it to template**





**🡪Template is not aware of Global js Variables**





**There are three types of property binding:**

1. Component property binding

2. Element property binding

3. Directive property binding

**Component property binding**

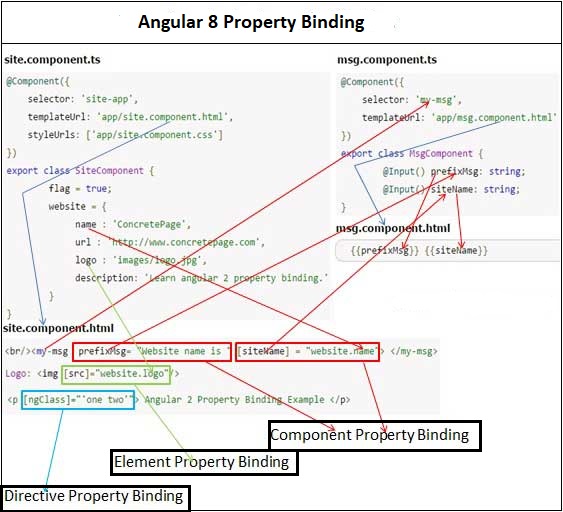
It workswithin the component element to bind parent component property. In the diagram, the arrows and rectangles in red color are displaying the functionality related to component property binding.

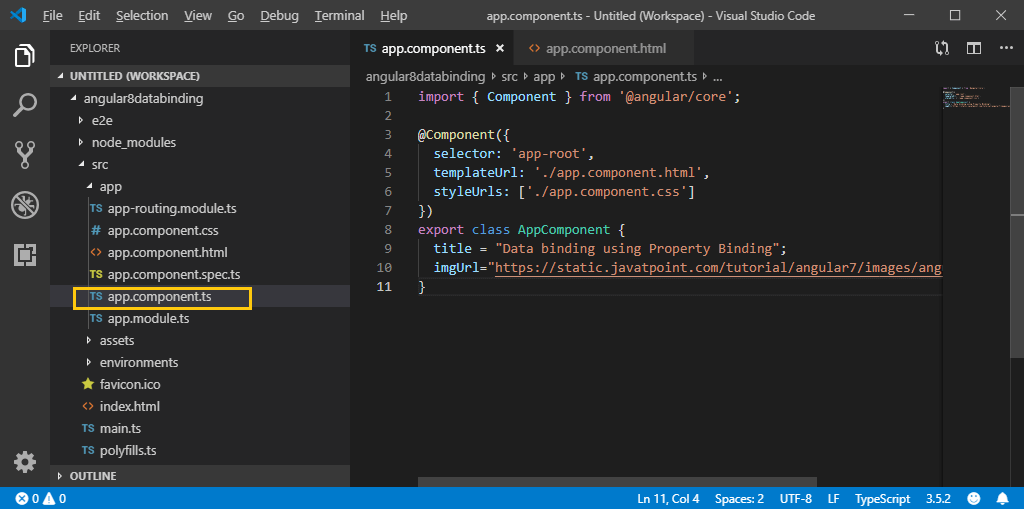
**Element property binding**

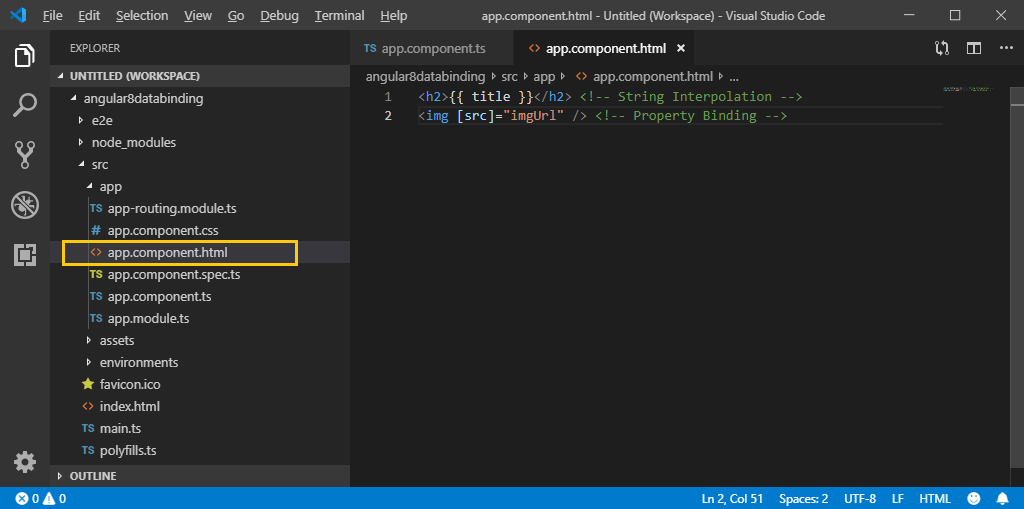
Element property binding works within HTML element, and it binds a component property to a DOM property. In the diagram, the arrows and rectangle in green color are displaying the functionality related to element property binding.

**Directive property binding**

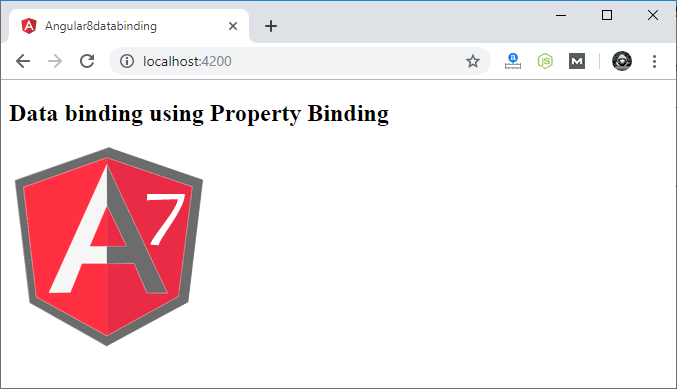
It also works within HTML element with directives such as **Ngclass** and **NgStyle**. In the directive, property binding a component properly or any angular expression is linked to the angular directive.





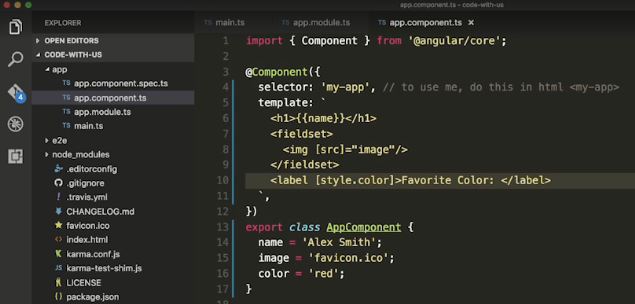


Run the **ng serve** command and open localhost 4200 to see the result.



[← Previous](https://www.tutorialandexample.com/event-binding-in-angular-8/)

**Example -1 binding a Color property to a**





**To avoid creation of specification files while creating angular component use –is - -spec false**

**Example:2**

**Product.component**

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

**@Component({**

**selector: 'app-productinfo',**

**templateUrl: './productinfo.component.html',**

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

**})**

**export class ProductinfoComponent {**

**productId:any;**

**productname:string;**

**price:number;**

**imgurl:string;**

**imgUrl:string;**

**url:string;**

**constructor(){**

**this.productId=100;**

**this.productname="mobile";**

**this.price=50000;**

**this.imgurl ="assets/images/shopping.png" this.imgUrl="https://static.javatpoint.com/tutorial/angular7/images/angular-7-logo.png";**

**this.url="https://angular.io/";**

**} }**

**Component.html**

**<h1>productInfo </h1>**

**<div>**

**<table style="width:100%; border-style: double;">**

**<tr><th>productID</th>**

**<th>productname</th>**

**<th>price</th>**

**<th>productimage</th>**

**<th>productimage</th>**

**<th>LeaarnAngular</th></tr> <tr><td>{{productId}}</td>**

**<td>{{productname}}</td>**

**<td>{{price}}</td>**

**<td><img width:200 src={{imgurl}}/></td>**

**<td><img [src]="imgUrl"/></td>**

**<td><a [href]="url">{{url}}</a>**

**</tr>**

**</table></div>**

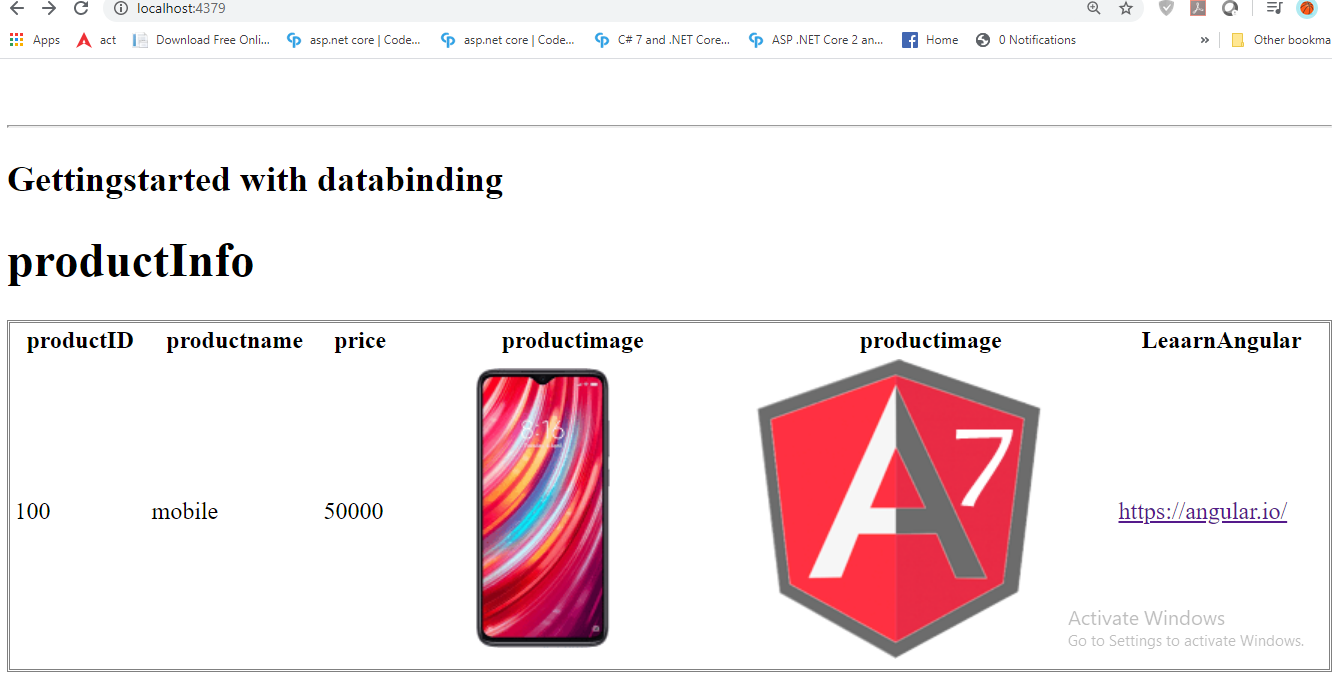
**App.component.html**

**<div>**

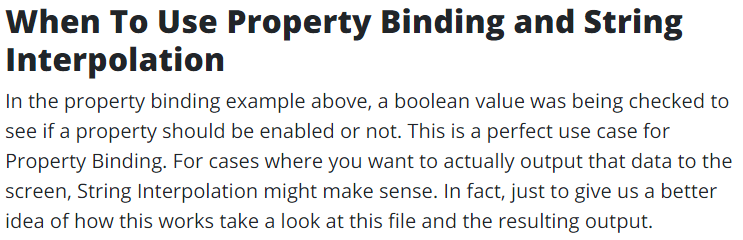
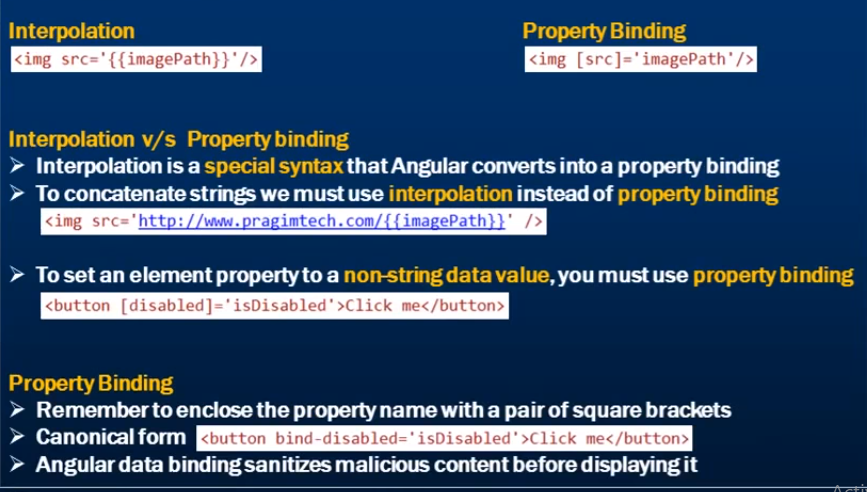
**<h2>Gettingstarted with databinding</h2>**

**<app-productinfo></app-productinfo>**

**</div>**

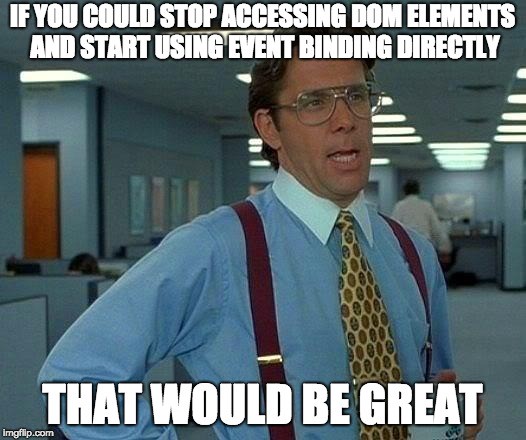


**>**

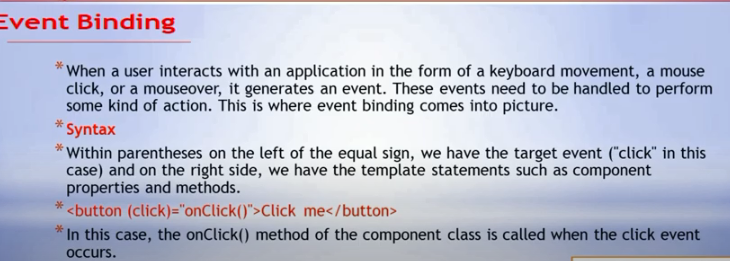


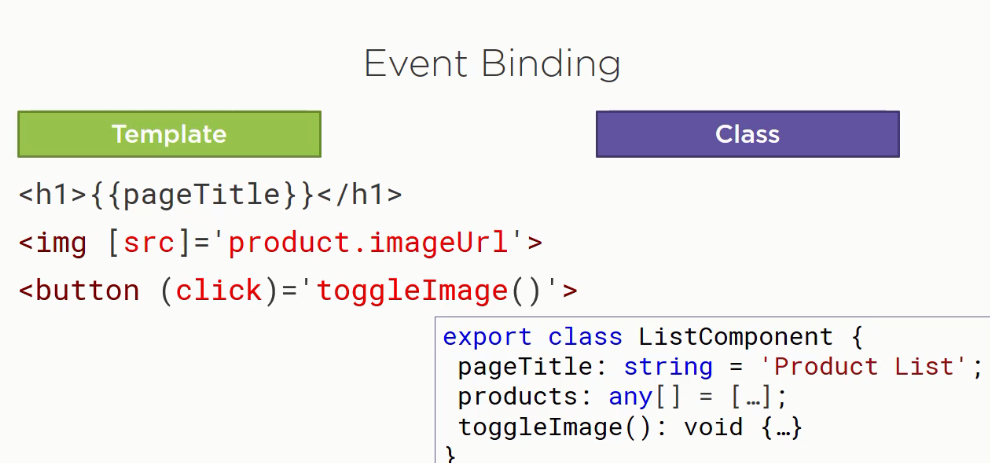
**Angular Event Binding**

Listening to DOM events and keeping Angular application performant can be challenging. Angular offers several methods to listen to events. By examining those available options and methods, we can understand and be aware of the characteristics that each one has. The four options for listening and reacting to DOM events that will be covered in this blog series are:

1. **Event Binding**: One-way data binding, in which information is sent from a component’s template to the component’s class
2. **@HostListener**: Angular decorator that handles events on the host element
3. **Renderer2:** Using Renderer2 .listen() method for a target event and element
4. **RxJS:**Using RxJS .fromEvent() operator that turns events into observable sequences.

In event binding, information flows from elements in a component template to the corresponding component’s class. With event binding, you don’t need to give the target element an identifier in order to access and attach your listeners to it because you are dealing with the target event and the target element directly in the template.



**The Data-Model may have functions containing code for business rules and for processing the property values. These functions are executed using the events of HTML elements .e.g. Button Click, TextBox Blur, keypress, keyup etc.** 

**These events execute a function defined in the component class as a response to the action the end-user takes on the UI element,**

1. **clicking on the button.**

**2) pressing a key on the keyboard.**

The event-binding features in Angular can be used to bind the functions defined in the Data-Model with the events exposed by HTML elements.

The event uses parenthesis ( ) notation in HTML.

The syntax for event BINDING:

( EventName)= ”function name in component class”

In ABOVE example, the component listens for the click event on a button. The name of the bound event is enclosed in parentheses, identifying it as the target event. To the right of the equals 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.

There are two syntaxes to add event binding.

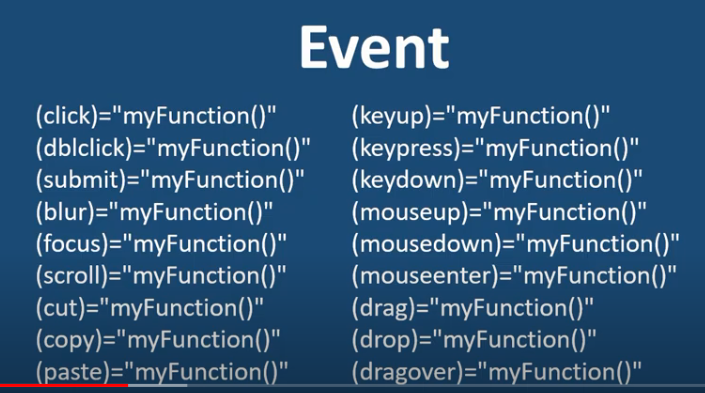
(1)The event of the HTML element is enclosed.in the parentheses with the function name passed to it. For example, in the above listing, the **click** event of the button is bound to the **save()** function.



(2). we can also use the **‘on-’** prefix for the click event, as used for binding the **clear()** function in the Clear button.

If you need access to the event payload object, you can pass $event as an argument to your handler function:

<button (click)="handleClick(**$event**)">Save</button>

here is another peculiar Angular event feature in relation to **keyup**and **keydown**events. If you only need to listen to specific keys, Angular offers shortcuts for that. For example, in order to listen to only the ENTER key, you can specify that key as a property of keyup or keydown events: **keyup.enter** or **keydown.enter.** These specific key events are called pseudo-events. You can use pseudo-events with event binding:

**<input (keydown.enter)="onEnterKey($event)">**

We can even listen to key combinations:

**<input (keyup.control.shift.enter)="onCtrlShiftEnter($event)">**

**Eventbubling**

### Event Bubbling:Event bubbling is used to identify an order in which event handlers are called when one element is nested inside a second element. And both parts have registered a listener for the event (i.e., click).

We are using a div wrapper on the button in component HTML, and div has also a click event handler. It is only to show some message if the div has been clicked.

Use the given code in **app.component.ts** file:

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

onSave($event){

console.log("Save button is clicked!", $event);

}

onDivClick(){

console.log("DIV is clicked!");

}

}

## app.component.html:

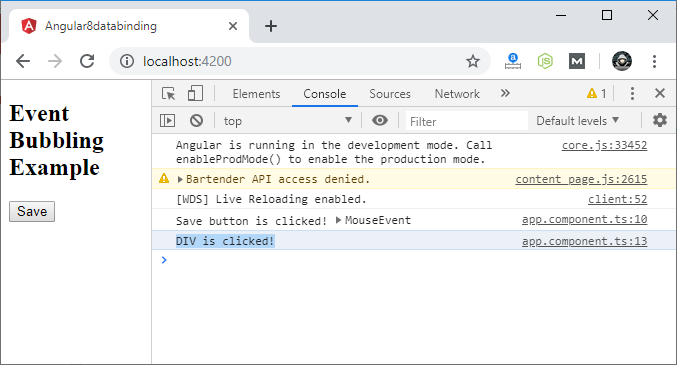
<h2>Event Bubbling Example</h2>

<!-- Event Bubbling --

<div (click)="onDivClick()">

<button (click)= "onSave($event)">Save</button> <!-- Event Binding -->

</div>

Click on the “save” button and open the console to see the result. 

Here, we can see that our div message has also occurred. This is due to bubbling where we have specified **onDivClick** button.