## Property & Event Binding

Our goal in this lecture is to hide the punchline of our jokes and only reveal them when the user clicks a button.

**Learning Outcomes**

* How to use the *hidden* DOM property to hide/show elements.
* Difference between DOM properties and HTML attributes.
* How to perform input property binding with []
* How to perform output event binding with ()

**Hiding and showing elements**

We can hide any element by adding an *attribute* called hidden to the element in HTML, so we could hide the punchline like so:

<

p class="card-text" hidden>{{joke.punchline}}</p

>

This is a core feature of HTML, not some functionality provided by Angular.

We want to add the hidden property to the element by default and then remove it when the user clicks a button.

So we add the following markup:

<

p class="card-text" [hidden]="true">{{joke.punchline}}</p

>

Specifically we added the markup [hidden]="true" and again it works, the element is hidden.

A few things:

1. We wrapped the attribute with a [], more on that later.
2. We made the attribute equal to true, if we made it equal to false it shows the element.

We say we have bound the value true to the property called hidden.

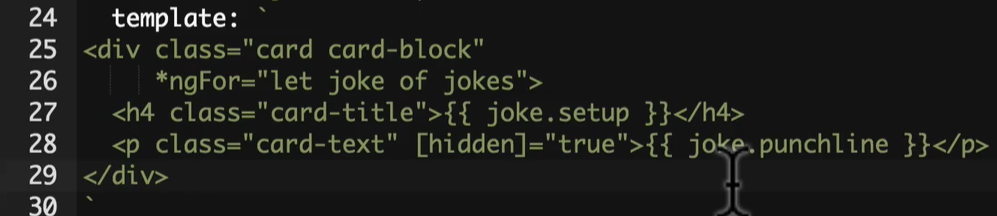
  This is called **Input Property Binding** and it’s a very important concept in Angular.

**HTML Attribute vs DOM Property**

The distinction between an HTML attribute and a DOM property is important in understanding binding in Angular.

HTML is a set of written instructions for *how* to display a web page.

The browser reads the HTML and creates something called a *DOM*, a *Document Object Model*. This is the manifestation of those HTML instructions in memory.



Ex: In the above template

There would be a DOM object for div and 2 other DOM objects H4 and P tag DOM object.

Changing the HTML doesn’t automatically update the webpage unless the user refreshes the browser, changing the DOM however instantly updates the webpage.

The DOM is the HTML in memory of the browsers. Updating DOM updates the instant current version of the browser.

There is mostly a 1-to-1 mapping between the names and values of HTML attributes and their equivalent DOM properties, but not always.

The hidden HTML attribute is a good example, it only needs to exist on an HTML element to instruct the browser to hide the element.

So hidden="true" hides the element but confusingly so does hidden="false"(i.e. this will not display the element because HTML just expects hidden word for the element to hide that element and it doesn’t require any value. Hence hidden=”true” or hidden=”false” are of same meaning) in HTML we just need to add hidden to hide the element.

The DOM representation of the hidden attribute is a property also called hidden, which if set to true hides the element and false shows the element.

Angular doesn’t manipulate HTML attributes, it manipulates DOM properties because the *DOM* is what actually gets displayed.

So when we write [hidden] we are manipulating the *DOM property* and not the *HTML attribute*.

This is why the above is called Input *Property* Binding and not Input Attribute Binding.

**Input Property Binding**

Looking back at our use of the hidden property:

<

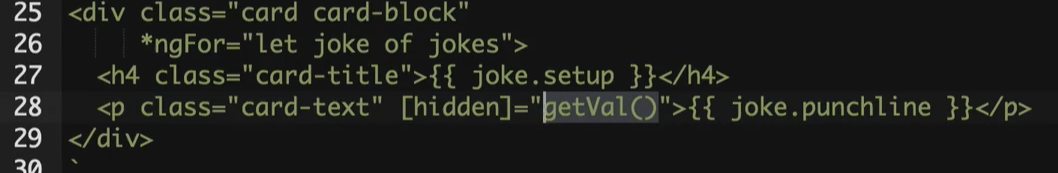
p class="card-text" [hidden]="true">{{joke.punchline}}</p

>

The *target* inside [] is the name of the DOM property. In the example above the target is the hidden DOM property.

The *text* to the right of = is JavaScript code that gets executed and the resulting value is assigned to the target.

 true is still JavaScript code which if executed returns true.



In the above example, the JavaScript function is evaluated and the result is assigned to the hidden property.

So in summary, we are binding to the DOM property hidden and setting it to true so the element is hidden.

In other parts of the web you’ll see this referred to as just *property binding*.

However to distinguish it from the other type of binding in Angular I like to call this **input** property binding.

We can only use this type of binding to change the value of the target. We can’t use it to get *notified* when the target’s value changes, to do that we need to use something called *Output Event Binding*, more on that soon.

Let’s add a property called hide on each joke and set it to true, like so:

[

{

setup: "What did the cheese say when it looked in the mirror?",

punchline: "Hello-Me (Halloumi)",

hide: true

},

{

setup: "What kind of cheese do you use to disguise a small horse?",

punchline: "Mask-a-pony (Mascarpone)",

hide: true

},

{

setup: "A kid threw a lump of cheddar at me",

punchline: "I thought

‘

That

’

s not very mature

’

",

hide: true

},

]

Now we can set the hidden input property to joke.hide in the template, like so:

<

div class="card card-block"

\*ngFor="let joke of jokes">

<h4 class="card-title">{{joke.setup}}</h4>

<p class="card-text"

[hidden]="joke.hide">{{joke.punchline}}</p>

<

/div

>

**Output Event Binding**

We want to show or hide the punchline when a user clicks a button, so lets add a button with the label *Tell Me* to the bottom of each card, like so:

<

div class="card card-block"

\*ngFor="let joke of jokes">

<h4 class="card-title">{{joke.setup}}</h4>

<p class="card-text"

[hidden]="joke.hide">{{joke.punchline}}</p>

<a class="btn btn-primary">Tell Me</a>

<

/div

>

We want to set joke.hide to false when the user clicks the button, and then back to true again when they click the button a second time.

To have Angular call some code every time someone clicks on the button we add some special markup to our button:

<

a class="btn btn-primary"

(click)="joke.hide = !joke.hide">Tell Me

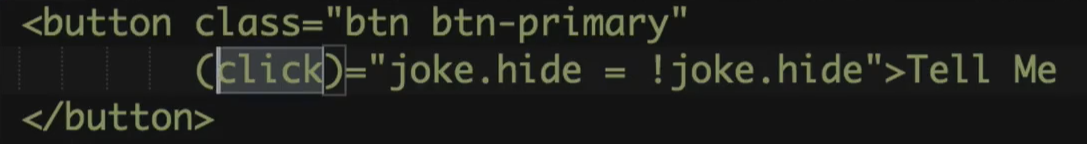
<

/a

>

We have some new syntax with (). The *target* inside the () is an event we want to listen for, we are listening for the click event.

The *text* to the right of = is some javascript which will be called every time a click event occurs.



joke.hide = !joke.hide toggles the value of joke.hide, so if it’s false clicking the button will change it to true, if it’s true clicking it will change it to false.

We can just as easily make the expression to the right of = call a function on our component instead, like so:

<

a class="btn btn-primary"

(click)="toggle(joke)">Tell Me

<

/a

>

Now when the button gets clicked it calls the toggle(…) function on the JokeListComponent and passes it the joke object the card is bound to, like so:

toggle(joke) {

joke.hide = !joke.hide;

}

Now when we click the button, we set the hide property to false which then *unhides* the element.

() => This type of binding is called Output event binding.

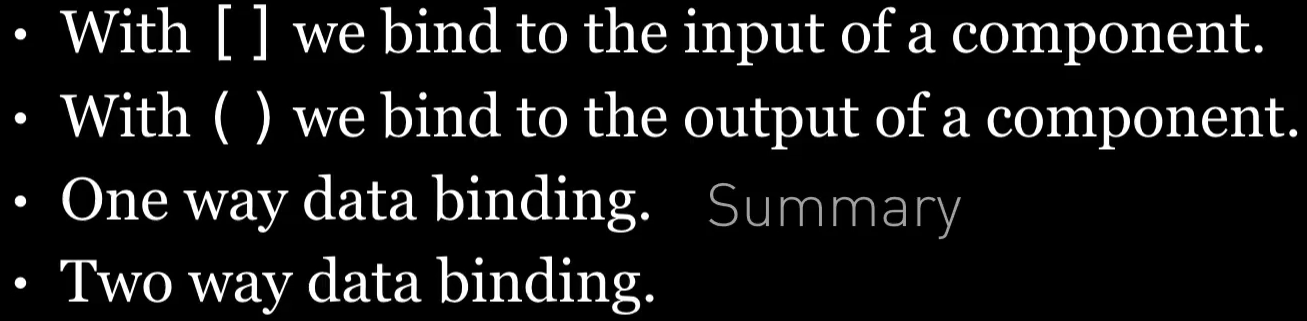
**Summary**

The way think about these two different ways of binding is in terms of inputs and outputs.

* With the [] we are binding to an input of a Component. i.e. input to view
* With the () we are binding to an output event of a Component or an element. i.e. output to view

This is what we call *one-way data binding*, since data only flows one way, either into or out of a component.

It is possible to *simulate* two-way data binding in Angular and we’ll cover that in a later section on Forms.



**Listing**

<http://plnkr.co/edit/6BGJzWKFuPLFeBdKZ6z2?p=preview>