Exploring Angular 2

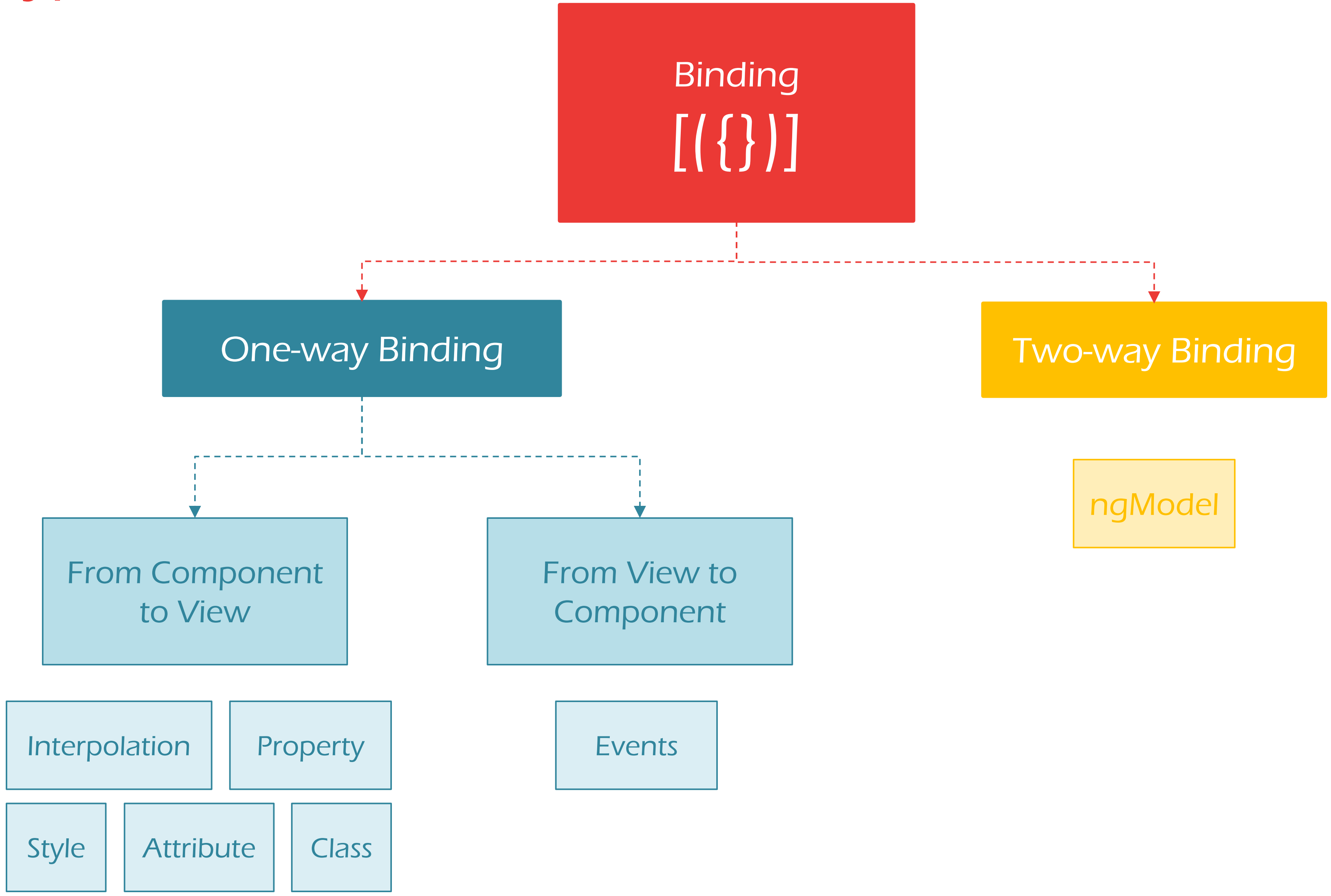
Lecture 2

Templates

Again, but with more details

Data Binding

Types


Binding
[({})]
One-way Binding
Two-way Binding
From Component to View
From View to Component
ngModel
Interpolation
Property
Style
Attribute
Class
Events

Interpolation

```
{{ expression }}
```

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    name: string = "Open Source";
    ....
}
```

app.component.html

```
<p> Hello, {{name}} </p>
```

Hello, Open Source

Property Binding

```
[property] = "expression"
```

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    imageUrl: string = “kiwi-juice.png”;
    ....
}
```

app.component.html

```
<img [src]=“imageUrl” />
```

Attribute Binding

```
[attr.<attr-name>] = "expression"
```

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    imageUrl: string = “kiwi-juice.png”;
    ....
}
```

app.component.html

```
<img [attr.src]=“imageUrl” />
```

Style Binding

```
[style.<style-name>] = "expression"
```

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    bg: string= "#ff0000";
    ....
}
```

app.component.html

```
<div [style.background]="bg"></div>
```

Class Binding

```
[class.<class-name>] = "expression"
```

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    isHidden: boolean= true;
    ....
}
```

app.component.html

```
<div [class.hide]="isHidden"></div>
```

Event Binding

```
(event) = "statement"
```

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    save(){
        console.log("Saved");
    }
}
```

app.component.html

```
<button (click)="save()">Save</button>
```

console

```
Saved
```

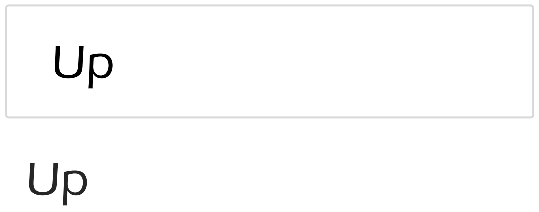
$event

$event is the Event Object that contains all the data of the Event Occurs to the target

Example

app.component.ts

```
export class AppComponent{
    movie="Prestige";
    changeIt(e){
        this.movie= e.target.value;
    }
}
```



Up

Up

app.component.html

```
<input (input)="changIt($event)">

<p>{{movie}}</p>
```

ngModel

```
[(ngModel)] = “expression”
```

Example

app.component.ts

```
export class AppComponent{

    movie=“Prestige”;

}
```

app.component.html

```
<input [(ngModel)]=“movie”>

<p>{{movie}}</p>
```

Up

Up

Pipe Operator (|)

Pipes are simple functions that accept an input value and return a transformed value

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    name: string = "Open Source";
    ....
}
```

app.component.html

```
<p> Hello, {{ name | uppercase }} </p>
```

Hello, OPEN SOURCE

Date Pipe

```
date_expression | date[:date_format]
```

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    today: number = Date().now();
    ....
}
```

Today is 02/22/2017

app.component.html

```
<p> Today is {{ today | date: "MM/dd/yy" }} </p>
```

Decimal Pipe

```
number_expression | number[:digitsInfo]
```

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    pi: number = 3.1415233455;
    ....
}
```

PI: 3.1415

app.component.html

```
<p> PI: {{ pi | number: '2.1-4' }} </p>
```

Safe Navigation Operator (?)

Safe navigation operator (?.) is a fluent and convenient way to guard against null and undefined values in property paths

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    movie: any = {name: "up"};
    ....
}
```

Movie Name is up

app.component.html

```
<p> Movie Name is {{ movie?.name }} </p>
```

Reference Variables (#)

Reference variable is a reference to a DOM element or directive within a template.

Example

app.component.ts

```
@Component({ .... })
export class AppComponent{
    movie: any = "Prestige";
    ....
}
```

Google
http://www.google.com

app.component.html

```
<a href="http://www.google.com" #spy >Google</a>
<p>{{ spy.href }}</p>
```

Structural Directives

*ngFor

app.component.ts

```
@Component({ .... })
export class AppComponent{
    movies: string[] = ["Forrest Gump", "Prestige", "Up" ]
    ....
}
```

app.component.html

```
<ul>
  <li *ngFor="let m of movies">
      {{ m }}
  </li>
</ul>
```

- Forrest Gump
- Prestige
- Up

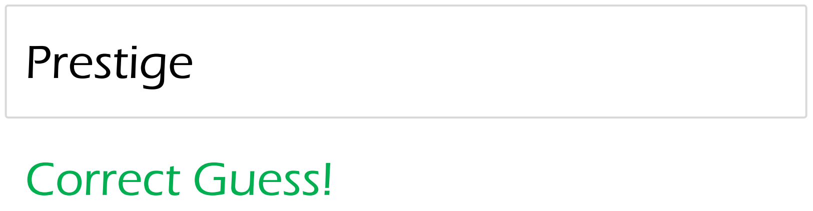
*ngIf

app.component.ts

```
@Component({ .... })
export class AppComponent{
    movies: string[] = [“Forrest Gump”, “Prestige”, “Up” ];
    movie: string = “Prestige”;
    temp = “”;
}
```

app.component.html

```
<input [(ngModel)]=“temp” >
<p *ngIf =“temp == movie”>Correct Guess!</p>
```



Prestige

Correct Guess!

ngSwitch

Practical Report

Use ngSwitch in Lab

Attribute Directives

ngClass

NgClass directive may be the better choice when we want to add or remove many CSS classes at the same time.

Example

app.component.ts

```
export class AppComponent{

    setClasses() {
        let classes =  { saveable: this.canSave,
                         modified: this.isModified};
        return classes;
    }
} // canSave is true, isModified is false.
```

app.component.html

```
<p [ngClass]=‘setClasses()’>Saveable but not modified</p>
```

```
<p class=“saveable” >Saveable but not modified</p>
```

ngStyle

NgStyle directive may be the better choice when we want to modify many CSS styles at the same time.

Example

app.component.ts

```
export class AppComponent{

  setStyles() {
    let styles =  { 'font-style': this.canSave ? 'italic': 'normal'
                     'color': this.isModified ? 'orange': 'green'
                  };
    return styles;
  }

} //canSave is true, isModified is false.
```

app.component.html

```
<p [ngStyle]='setStyles()'> Hello Open Source </p>
```

Hello, Open Source

@Input

Input directive let the component receive inputs from other components

Example

movie.component.ts

```
@Component({
    selector: 'app-movie',
    template:`<p>{{movieName}}</p>`
})
export class MovieComponent{
    @Input() movieName;
}
```

Forrest Gump

app.component.html

```
<div>
   <app-movie [movieName]='movie.name'></app-movie>
</div>
```

@Output

Output directive let the component send data to the other components

Example

movie.component.ts

```
@Component({ ... })
export class MovieComponent{
    @Output() editMovie = new EventEmitter();
    newMovie = {title: ‘The Dark Knight’, Actor: ‘Cristian Pale’ }
    constructor(){ this.editMovie.emit(this.newMovie)}
}
```

app.component.html

```
<div>
   <app-movie (editMovie)=‘getNewMovie($event)’></app-movie>
</div>
```

Forms

A new way to treat with HTML Forms

Intro

A form creates a cohesive, effective, and compelling **data entry experience**.

An Angular form coordinates a set of data-bound user controls, tracks changes, validates input, and presents errors.

State Tracker

It tracks every change of the form input state and add a class corresponding to it's current state.

```
<form>

   <input type="text" id="name" required [(ngModel)] ="movie" name="n">

</form>
```

Prestige

ng-pristine	ng-untouched	ng-valid
ng-dirty	ng-touched	ng-invalid

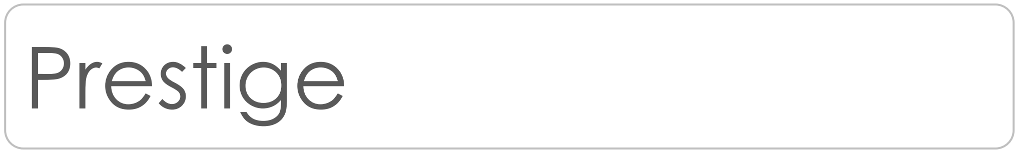
State Tracker

It tracks every change of the form input state and add a class corresponding to it’s current state.

```
<form>

   <input type="text" id="name" required [(ngModel)] ="movie" name="n">

</form>
```


Prestige

ng-pristine	ng-untouched	ng-valid
ng-dirty	ng-touched	ng-invalid

State Tracker

It tracks every change of the form input state and add a class corresponding to it's current state.

```
<form>

   <input type="text" id="name" required [(ngModel)] ="movie" name="n">

</form>
```

Prestige |

$\boxed{\text{ng-pristine}}$	ng-untouched	$\boxed{\text{ng-valid}}$
ng-dirty	$\boxed{\text{ng-touched}}$	ng-invalid

State Tracker

It tracks every change of the form input state and add a class corresponding to it's current state.

```
<form>

   <input type="text" id="name" required [(ngModel)] ="movie" name="n">

</form>
```

Pres |

ng-pristine ng-untouched ng-valid

ng-dirty ng-touched ng-invalid

State Tracker

It tracks every change of the form input state and add a class corresponding to it's current state.

```
<form>

   <input type="text" id="name" required [(ngModel)] ="movie" name="n">

</form>
```

ng-pristine	ng-untouched	ng-valid
ng-dirty	ng-touched	ng-invalid

State Tracker Benefits

1 Add Custom CSS for every state:

```
.ng-invalid{
        border-color: red;
}
```

2 Add Custom Error messages based on the State:

```
<input type="text" id="name" required [(ngModel)] ="movie"

name="name" #name=ngModel>

<div [hidden] =“name.pristine || name.valid”>Input Not Valid</div>
```

FormsModule

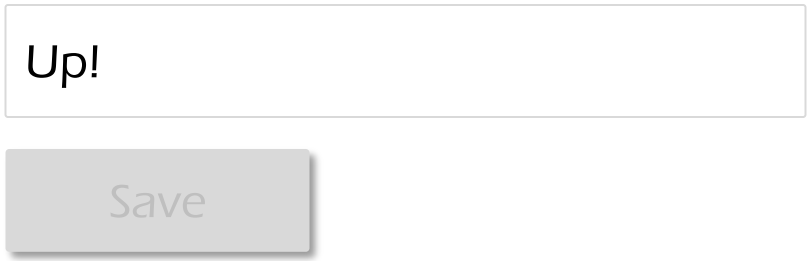
app.module.ts

```
import { NgModule }  from '@angular/core';
import { BrowserModule }  from '@angular/platform-browser';
import { FormsModule }  from '@angular/core';
import { MovieFormComponent }  from './movie-form.component';
@NgModule({
       imports: [ BrowserModule, FormsModule ],
       declarations: [ MovieFormComponent ],
       bootstrap: [ MovieFormComponent ]
})
export class AppModule{}
```

ngSubmit & ngForm

movie-form.component.ts

```
export class MovieFormComponent{
    submitted = false;
    submitIt() {
        this.submitted = true;
        //Any other procedures
    }
}
```



movie-form.component.html

```
<form (ngSubmit)="submitIt()"  #movieForm="ngForm">
    <input type="text" id="name" required name="n">
    <input type="submit" [disabled]="!movieForm.form.valid">
</form>
```

Thank You