Angular
Lab 1: Data Binding
Exercise Solutions

1 Lab setup

2 Generating a new Angular project

app.component.ts

```
```

app.component.html

```
```

app.component.css

```css

```

3 General structure of an Angular app

4 Template expressions and interpolations

Q1: Add in a property to the root component called `myAge` and give it any random value

Q2: Display the value of this property with a random string (for e.g. `I am 60 years old`) in the template HTML

Q3: Add in a method called `getAgeMessage` which returns a string depending on the value of `myAge`. The content of the string returned is:

`myAge` less than 30: I am sooo young
`myAge` between 31 and 50 : I am middle aged
`myAge` more than 51: I am soooo old.

Q4: Using this method, display a message similar to the following in the template:
`I am 40 years old, and I am middle aged`
Change the value of `myAge` to check that the correct messages are displayed for the correct age.

app.component.ts

```typescript
// Q1
myAge = 40;

// Q3
getAgeMessage(age: number ) : string {
  if (age < 30)
    return "I am sooo young";
  else if (age >= 31 && age <= 50)
    return "I am middle aged";
  else
    return "I am sooo old";
  }
```

app.component.html

```
<!-- Q2 -->
<p> I am {{ myAge }} years old </p>

<!-- Q4 -->
<p> I am {{ myAge }} years old, and {{ getAgeMessage(myAge) }} </p>
```

5 Angular app scripts and the DOM

6 Property binding

Q1: Add in two properties `socialMediaSite` (which holds any random valid URL) and `myOpenStyle` (which holds the value of either `_self` or `_blank` (see Q6 from Topic 4) to the root component

Q2: Add in an an <a> element into the template which uses these two properties in the syntax of property binding. Check that it works properly.

app.component.ts

```
  // Q1
  socialMediaSite = "https://tiktok.com";

  myOpenStyle = '_blank';
```

app.component.html

```
<!-- Q2 -->
<a [href]="socialMediaSite" [target]="myOpenStyle">My favorite social media
site</a>
```

7 Class binding

First Part: (after `app.component-v3.ts, app.component-v3.html`)

Q1: Create a <div> element and nest a <p> with some text within this <div>. The CSS Box model provides a set of properties to delineate an element and its child elements. You can use a variety of properties to control the border look for the <div>. Set a suitable `padding` for the <div>.

Q2: Add 2 different classes in the template CSS which sets different values for the `border-style` property, another 2 different classes which sets different values for the `border-color` property, and another 2 different classes which sets different values for the `border-width` property. Give these classes any suitable names.

Q3: Add any one of these 6 classes to the <div>, and then use single binding to add another different class to the <div>, depending on a Boolean property in the component.

Q4: Add any 2 or more of these 6 classes to the <div>, and then use single binding to remove one of these classes, depending on a Boolean property in the component.

`app.component.ts`

```
  //Q3
  isItHot = true;

  //Q4
  isItThick = false;
```

`app.component.html`

```
<!-- Q1, Q3 -->
<div class="stylishborder" [class.hotborder]="isItHot">
  <p>Some random text inside this first paragraph </p>
```

```
</div>

<!-- Q4 -->
<div class="boringborder thickborder freezingborder"
[class.thickborder]="isItThick">
  <p>Some random text inside this second paragraph </p>
</div>
```

app.component.css

```

/* Q1. */

.div {
    padding: 10px;
}

/* Q2. */

.stylishborder {
    border-style: inset;

}

.boringborder {
    border-style: dotted;
}

.thickborder {
    border-width: 20px;
}

.thinborder {
    border-width: 2px;
}

.hotborder {
    border-color: red;
}

.freezingborder {
    border-color: blue;
}
```

Second Part:

Reuse the same <div> and CSS classes from First Part: Q1 and Q2

Q5: Add any one of the 6 classes to the <div>, and use a string to add another 2 more of these classes to the <div>

Q6: Add any one of the 6 classes to the <div>, and use an object to add another 2 more of these classes to the <div> while removing the existing class.

Q7. Add any one of the 6 classes to the <div>, and use an array of strings to add another 2 more of these classes to the <div>

`app.component.ts`

```
  // Q5.
  classesForBorder = "thinborder hotborder";

  // Q6.
  borderClassesToAdd = {
    stylishborder : false,
    boringborder: true,
    thickborder : true,
    freezingborder :  true
  };

 // Q7.
  arrayOfClassesToAdd = ['thickborder','stylishborder','hotborder'];
```

`app.component.html`

```
<!-- Q5 -->
<div class="stylishborder" [class]="classesForBorder">
  <p>Some random text inside this paragraph </p>
</div>

<!-- Q6 -->
<div class="stylishborder" [class]="borderClassesToAdd">
  <p>Second div with paragraph in it </p>
</div>

<!-- Q7 -->
```

```
<div class="freezingborder" [class]="arrayOfClassesToAdd">
  <p>Second div with paragraph in it </p>
</div>
```

8 Style binding

Q1: Create a <p> element and use single style binding to set its `font-family` property.

Q2: Create a <div> element and nest a <p> with some text within this <div>. Use multiple style binding with a string containing the relevant properties to style the <div>

Q3: Create another <div> element and nest a <p> with some text within this <div>. Use an object containing the relevant properties to style this <div>

`app.component.ts`

```
  //Q1
  fontFamilySettings = `"Lucida Console", "Courier New", monospace`;

  //Q2

  stylingForFirstDiv = "border: 5px dotted green;";

  //Q3
  stylingForSecondDiv = {
    border: "2px solid blue",
    "border-radius": "10px",
  };
```

`app.component.html`

```
<!-- Q1 -->
<p [style.font-family]="fontFamilySettings">This is the first paragraph </p>


<!-- Q2 -->
<div [style]="stylingForFirstDiv" >
```

```
    <p>Some random text inside this first div </p>
</div>

<!-- Q3 -->
<div [style]="stylingForSecondDiv" >
    <p>Some random text inside this second div </p>
</div>
```

9 Event binding

Q1: Add in 3 radio buttons, each with different values.

Q2. Provide event binding for the click event for all these 3 radio buttons to the same component method

Q3. Implement this component method so that it logs the value of the particular radio button clicked.

app.component.ts

```
  //Q3
  processRadioButton(event: Event) {
    let hie = event.target as HTMLInputElement;
    console.log("The value of the control is : " + hie.value);
  }
```

app.component.html

```
  <!-- Q1 and Q2 -->
  <input type="radio" id="html" name="fav_language" value="HTML"
(click)="processRadioButton($event)">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS"
(click)="processRadioButton($event)">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaScript"
(click)="processRadioButton($event)">
  <label for="javascript">JavaScript</label>
```

9.1 Template reference variables for event binding

Repeat Q1 - Q3 from the previous section, but implement the event binding using template reference variables instead.

`app.component.ts`

```
    //Q3
    processRadioButton(val: string) {
      console.log("The value of the control is : " + val);
    }
```

`app.component.html`

```
    <!-- Q1 and Q2 -->
    <input #radiobutton1 type="radio" id="html" name="fav_language"
value="HTML" (click)="processRadioButton(radiobutton1.value)">
    <label for="html">HTML</label><br>
    <input #radiobutton2 type="radio" id="css" name="fav_language"
value="CSS" (click)="processRadioButton(radiobutton2.value)">
    <label for="css">CSS</label><br>
    <input #radiobutton3 type="radio" id="javascript" name="fav_language"
value="JavaScript" (click)="processRadioButton(radiobutton3.value)">
    <label for="javascript">JavaScript</label>
```

9.2 Combining interpolation, property, class and event binding

Q1. Using any 2 or more HTML elements of your choice (text field, normal button, radio button, check box, paragraph, etc) create a scenario of your design that combines interpolation, property, class and event binding.

Many possible solutions here, so no possible sample solution

10 Two way binding with ngModel

Q1. Create a radio button. Use banana-in-the-box syntax to perform 2 way binding between the button value and a component property and display the property via interpolation.

app.component.ts

```
//Q1
radioButtonContent = '';
```

app.component.html

```
<!-- Q1 -->
<input type="radio" id="html" name="fav_language" value="HTML"
[(ngModel)]="radioButtonContent">
<label for="html">HTML</label><br>

<p>Value of radio button : {{radioButtonContent}}</p>
```