Angular
Lab 1
Data Binding

1 Lab setup

Make sure you have the following items installed

- Latest version of Node and NPM (note: labs are tested with Node v16.13 and NPM v8.1 but should work with later versions with no or minimal changes)
- Latest version of TypeScript (note: labs are tested with TypeScript v4.5.4 but should work with later versions with no or minimal changes)
- Latest version of Angular (note: labs are tested with Angular v13.1 but should work with later versions with no or minimal changes)
- Visual Studio Code (or a suitable alternative IDE for Angular/TypeScript)
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

In each of the main lab folders, there are two subfolders: `changes` and `final`.

- The `changes` subfolder holds the source code and other related files for the lab. The creation of the Angular app will proceed in a step-wise fashion in order to clearly demonstrate the various features of the framework. The various project source files in this folder are numbered

in a way to reflect this gradual construction. For e.g. `xxx.v1.html`, `xxx.v2.html`, etc represents successive changes that are to be made to the `xxxx.html`

- The `final` subfolder holds the complete Angular project. If there was some issue in completing the step-wise construction of the app using the files from the `changes` subfolder, you can use this to run the final working version.

The project folder in `final` is however missing the crucial `node_modules` subfolder which contains all the dependencies (JavaScript libraries) that your Angular app needs in order to be built properly. These dependencies are specified in the `package.json` file in the root project folder. They are omitted in the project commit to GitHub because the number and size of the files are prohibitively large.

In order to run the app via the Angular development server (`ng serve`), you will need to create this subfolder containing the required dependencies. There are two ways to accomplish this:

a) Copy the project root folder to a suitable location on your local drive and run `npm install` in a shell prompt in the root project folder. NPM will then create the `node_modules` subfolder and download the required dependencies as specified in `package.json` to populate it. This process can take quite a while to complete (depending on your broadband connection speed)

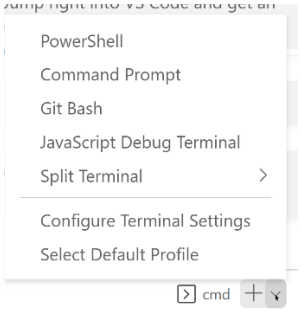
b) Alternatively, you can reuse the dependencies of an existing complete project as most of the projects for our workshop use the same dependencies. In this case, copy the `node_modules` subfolder from any existing project, and you should be able to immediately start the app. If you encounter any errors, this probably means that there are some missing dependencies or Angular version mismatch, in which case you will then have to resort to `npm install`.

The lab instructions here assume you are using the Chrome browser. If you are using a different browser (Edge, Firefox), do a Google search to find the equivalent functionality.

2 Generating a new Angular project

Create an empty folder on your local drive which you will dedicate for use in this workshop. You will generate and/or copy the project folders for the various Angular apps that you will build in this workshop here. Give it any suitable name (e.g. `C:\myprojects`). **DO NOT** generate your project folders directly in your root drive C:\

You will generate the new Angular project using the Angular CLI from the command prompt or shell terminal. You can open a standard Windows command prompt (type `cmd` in Search) or else use the embedded terminal in Visual Studio Code. To access this, select Terminal -> New Terminal from the main menu or use the context menu in the lower right hand corner



At the terminal, type:

```
ng new bindingdemo
```

Press enter to accept the default values for all the question prompts that follow regarding routing and stylesheet.
The process of creating a project will take a while (depending on the speed of your broadband connection) as NPM will need to download and install all the JavaScript modules that the Angular project needs to run properly.

Navigate into the root project folder from the terminal with:

```
cd bindingdemo
```

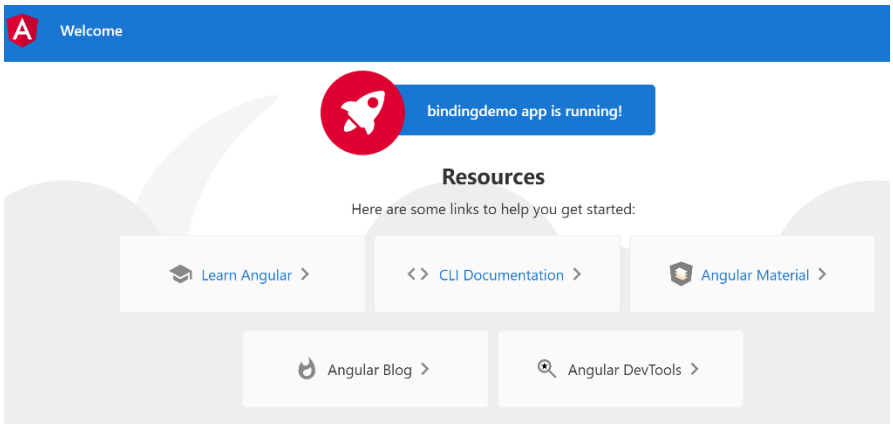
Build the app and serve it via Angular's live development server by typing:

```
ng serve
```

The final output line from this should indicate that the compilation process was successful and identify the port that the live development server is currently serving up the Angular app dynamically from (by default this will be port 4200)

Open a browser tab at:
http://localhost:4200/

You should be able to see the default landing page for all new autogenerated Angular projects.

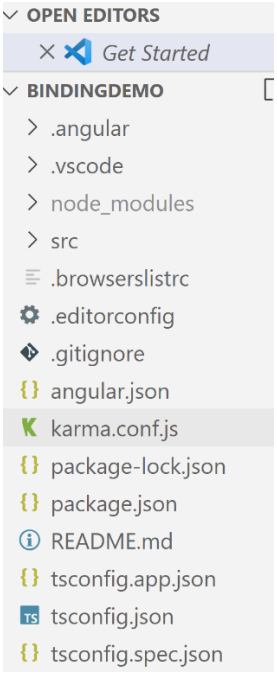


To stop the development server, type Ctrl+C in the terminal window that it is running in. You can restart again anytime by typing `ng serve` in the project root folder.

Additional references:
https://ccbill.com/kb/install-angular-on-windows

3 General structure of an Angular app

Open the root folder of this Angular project using VS Code.
You should see a variety of files listed in the Explorer tab.



Most of these files are in JSON format and provide configuration for the Angular app as well the TypeScript compiler used to compile it. We will examine these in more detail in a subsequent lab.

Navigate in the Explorer into `src/app`. You will see the core files that constitute a minimal Angular app. Open `app.component.ts`

This is the root component of the Angular app, which is always a TypeScript class and has the default name of AppComponent. It typically has a single property title which by default will be the name of the project (bindingdemo in this case).
It has a @Component decorator which identifies the class as a proper Angular component (rather than just an ordinary TypeScript class). The decorator specifies several important metadata items:

- `selector`: This is a CSS selector that tells Angular to create and insert an instance of this component wherever it finds the corresponding tag in template HTML of any other component in the app (or `index.html` for the case of the root component). While we normally use element selectors, any valid CSS selector (https://www.w3schools.com/css/css_selectors.asp) can be used as well. Note that there can only be one component instantiated for a given element selector in a template. When the component is instantiated, its corresponding template is then substituted

into the place of its selector tag in the target template HTML and Angular then renders this as a view.

- `templateUrl`: The module-relative address of this component's HTML template. Alternatively, you can provide the HTML template inline, as the value of the *template* property (which we will see later in another lab)

- `styleUrls`: Specifies external CSS files to load styles from to style the HTML template. Alternatively, you can use the *styles* array property to define CSS rules as strings directly (which we will see later in another lab)

There are other metadata items that can be specified here, which we will examine in a later lab.

Open `app.component.html`
This contains the template HTML corresponding to the root component (notice that the name of this file is specified in the `templateUrl` property of the @Component decorator metadata). This contains the HTML and styling that produces the default landing page for the Angular app that we saw earlier.

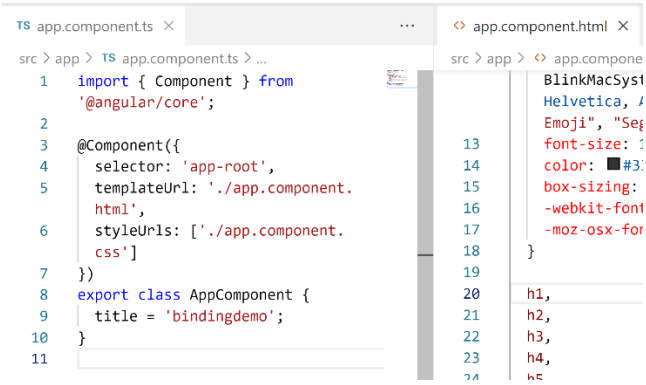
Open `app.component.css`
This is the stylesheet where we can specify CSS styling rules (https://www.w3schools.com/css/css_syntax.ASP) for the template HTML.
Notice that is currently empty. All the styling for the landing page is currently achieved via embedded style rules in the HTML within the <style> tags.

Open `app.component.spec.ts`
This contains the code to perform basic unit testing on the Angular app, which we can expand upon if we wish. Angular uses the Jasmine test framework by default for creating tests and uses the Karma test runner to run the tests.

Open `app.module.ts`
Angular applications are modular and Angular has its own modularity system called NgModules. Every Angular application has at least one NgModule class, the root module, which is conventionally named AppModule and resides in `app.module.ts`. You launch your application by bootstrapping the root NgModule. Notice that it has a declaration for AppComponent (the root component) and also specifies that the AppComponent is the component that will be used to bootstrap the app.
Notice that it has to import the AppComponent class from the specified file, which is the reason why the AppComponent class is declared with the `export` keyword.

Open `src/index.html`
This is the HTML file (the single page of a SPA) that will be served up by the Angular live development server. Notice that the <body> element of this file contains the CSS selector `(<app-root>)` for the root component AppComponent. As explained earlier, Angular will replace this selector with the HTML from the template when it builds and serves up the app.

You can use View -> Editor layout to split the editors for the different files in order to lay them out in different orientations to simplify working with them. It is often useful to have both the component and its template viewable simultaneously in two different editors because the content in both these files are tightly bound with each other, as we will see in a later lab.
You can also use View -> Word Wrap to toggle word wrap if necessary to avoid scrolling across editor views for long lines of code.



Additional references:
https://angular.io/guide/architecture-components#introduction-to-components-and-templates
https://angular.io/guide/architecture-components#component-metadata
https://angular.io/api/core/Component#description

4 Template expressions and interpolations

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v1.ts`

`app.component-v1.html`

`app.component-v1.css`

Place the following files from `changes` into `src/assets` (navigate to this folder in the Explorer View in VS Code and drag and drop the images)

`cat.jpg`
`dog.jpg`
`horse.jpg`

Notice that after you have made these changes, the Angular server automatically rebuilds the app and serves it at the port that is currently listening on (default 4200). There is no need to restart the server again.

Notice that the style rule specified in `app.component.css` is applied to `app.component.html`, as discussed earlier

Make changes to the values of the properties and methods in AppComponent and notice that the changes are updated immediately to the app landing page. The Angular development server recompiles and updates the application automatically so that the new content is displayed in the browser view without the need to reload. This uses the hot module replacement feature of webpack, which is the main module bundler used in Angular

Template expressions are used primarily in data binding. The most common form is to use them in interpolations within {{ }}

When used in interpolation, template expressions will always produce a string value that can be incorporated into a HTML element at 2 specific places:

1. In the text portion of HTML elements accept text (for e.g. <p>, , <break>,), see: https://flaviocopes.com/html-text-tags/

   ```
   <p>{{title}}</p>
   ```

2. Assignment to an element attribute that expects a string, for e.g.

   ```
   <div><img src="{{itemImageUrl}}"></div>
   <a href="/product/{{productID}}">{{productName}}</a>
   ```

The same result for using interpolation to assign a value to an element attribute can be achieved via: property binding. In general, while assignment to an attribute that expects a string value can be achieved using interpolation, it is recommended to use any appropriate form of binding (property, class, style, etc) to achieve the same result.

On the same note, since template expressions in interpolation will always produce a string value, you cannot use it to assign value to a non-string attribute - in which case, property binding MUST be used.

The expression context refers to the location where the particular variable names used in a template expression can be resolved; this is typically the component that the template is directly associated with. Thus, the template expression will typically reference component properties and methods.

Component properties must be public (this is the default access modifier if none is specified) in order to be accessible in the template expression

References:

https://angular.io/guide/interpolation#text-interpolation
https://angular.io/guide/interpolation#displaying-values-with-interpolation
https://angular.io/guide/interpolation#template-expressions

Issues to keep in mind regarding template expressions:
https://angular.io/guide/interpolation#syntax
https://angular.io/guide/interpolation#expression-best-practices

A listing of good sites with comprehensive collection of royalty free images that you can use for your projects:
https://buffer.com/library/free-images/

5 Angular app scripts and the DOM

On the main landing page of the Angular app, right click anywhere in the page and select View Page Source from the context menu. You will be able to see the same HTML file; you will be able to see the

same HTML file; but with an additional 5 JavaScript scripts referenced from it at the bottom. These 5 scripts together implement the functionality of the Angular app that will generate a live DOM from the static HTML to create the view that you see in your browser:

- runtime.js – Webpack runtime file. Webpack is a static module bundler for JavaScript applications — it bundles all the code from the Angular app and makes it usable in a web browser
- polyfills.js –provides the polyfill features that are not necessary for app compatibility with older browsers, particularly those that do not support newer JavaScript versions such as ES6.
- styles.js - Global style files bundled in a single file. This files are declared in `angular.json`
- vendor.js - Angular and other 3rd party vendor scripts bundled together
- main.js – contains the application codebase; including components (ts, html and css), pipes, directives, services and all other imported modules (including third party)

You can click on any of these script references to view their contents. For e.g. click on `main.js` to view the transpiled Javascript (ES5) for your AppComponent Typescript class. The largest of these Javascript files is `vendor.js`, which contains the compiled Angular library itself along with any other 3rd party library modules.

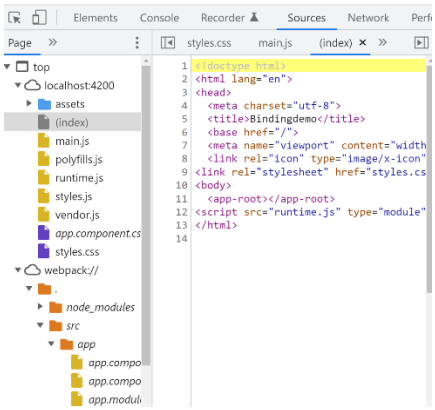
In the browser, select More Tools -> Developer Tools, and click on the Elements tab to switch to the Element view which shows the DOM tree that the current view in the browser is rendered from. You will notice that the <app-root> element is now populated with content through the operation of these 5 scripts:

```
<!DOCTYPE html>
<html lang="en">
▶ <head>…</head>
▼ <body> == $0
  ▼ <app-root _nghost-def-c11 ng-version="13.1.1">
      <h1 _ngcontent-def-c11> My first Angular app </h1>
      <h2 _ngcontent-def-c11> This app is hosted in Kuala
      Lumpur </h2>
      <h3 _ngcontent-def-c11> KUALA LUMPUR is a great place
      </h3>
      <p _ngcontent-def-c11> There are 3 languages that I
      like: Python, Java and JavaScript</p>
      <p _ngcontent-def-c11> Incorrect addition : 23 </p>
      <p _ngcontent-def-c11> Correct addition : 5 </p>
      <img _ngcontent-def-c11 alt="some animal" src="../asse
      ts/dog.jpg">
```

Notice that there are the standard HTML elements such as <h1>, <h2> and <p> present; but these are augmented with Angular-specific attributes such as `_ngcontent-xxxx` and `_nghost-xxx`, which Angular uses isolate component styles from one another (to be covered in a later lab).

The style sheet for this component (`app.component.css`) is also embedded directly in the page via <style> tags in the <head> element.

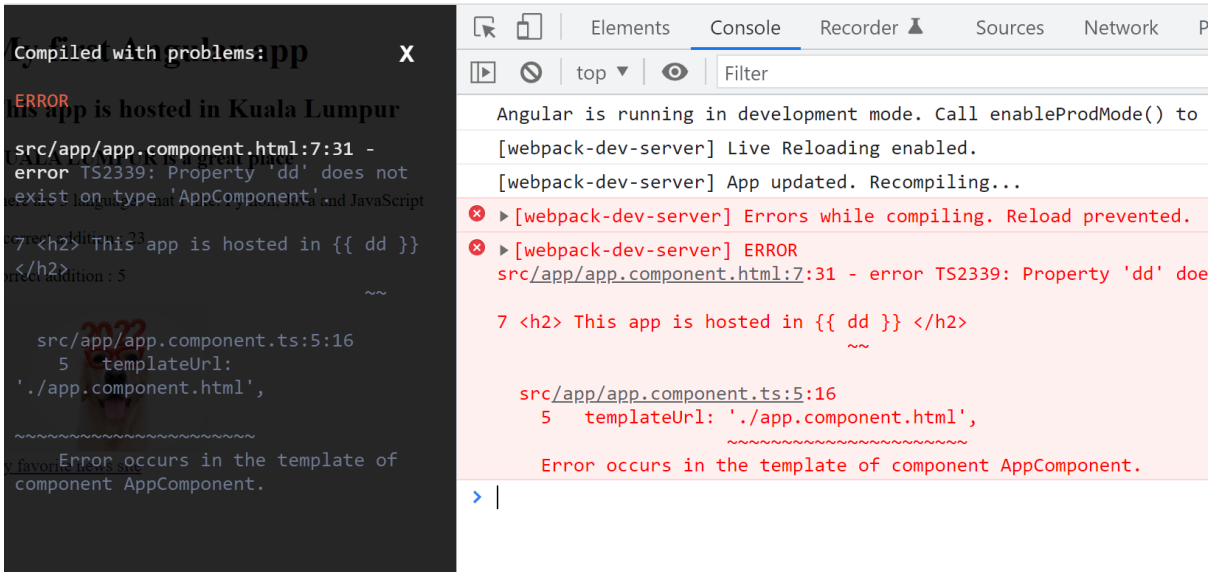
If you navigate to the Sources view, you can view all the 5 Javascript files mentioned earlier. Further at the bottom, if you navigate to `webpack://  .  src/app`, you will also be able to view the source code for your app (`app.component.html, app.component.ts`, etc)



Purposely introduce an error into the component or the template and save it. For e.g. in app.component.html, try to reference a property that does not exist in the component.

```
<h2> This app is hosted in {{ xyz }} </h2>
```

Notice that a compilation error is flagged in the window running the Angular development server, with more specific information on the error.
The compilation error is also shown on the landing page as well as in the Console tab of DevTools.



Correct the error and save. The app should render normally again.

6 Property binding

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v2.ts`

`app.component-v2.html`

Make changes to the values of the properties and methods in AppComponent and notice that the changes are updated immediately to the app landing page. The Angular development server recompiles and updates the application automatically so that the new content is displayed in the browser view without the need to reload.

Template expressions are also used in property binding. In this case, they will appear between double quotes (unlike in interpolations, where they appear within {{ }}).

When template expressions are used, the value from the template expression is assigned to the property of a binding target. This target could be a HTML element, a component or a directive. It is important to keep in mind that when it is a HTML element, we are assigning the value to a property of the DOM of that element and `*NOT*` a HTML tag attribute.
There is however one single exception, which is the case of Attribute binding , where the template expression value is assigned to the attribute of a HTML element (typically an `aria` attribute).

Attributes initialize DOM properties and you can configure them to modify an element's behavior. Properties are features of DOM nodes.

- A few HTML attributes have 1:1 mapping to properties; for example, `id`.
- Some HTML attributes don't have corresponding properties; for example, `aria-*`.
- Some DOM properties don't have corresponding attributes; for example, `textContent`.

In this example, we are binding to the various specific properties of the `h3, button, img` and `a` elements.

All DOM HTML elements have a set of common properties and methods:
https://www.w3schools.com/jsref/dom_obj_all.asp

In addition, each DOM element will have a further subset of properties and methods that are common to it, for e.g.

DOM Button
https://www.w3schools.com/jsref/dom_obj_pushbutton.asp

DOM Image
https://www.w3schools.com/jsref/dom_obj_image.asp

DOM Anchor
https://www.w3schools.com/jsref/dom_obj_anchor.asp

In some situations, you can achieve equivalent results using interpolations and property binding.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v2-2.html`

Note that when you change the value of `isDisabled` in `app.component.ts`, there is no effect on the button where interpolation is used on the `disabled` attribute.

In general, while assignment to an attribute that expects a string value can be achieved using interpolation, it is recommended to use any appropriate form of binding (property, class, style, etc) to

achieve the same result. The only time we should use interpolation is to place string values in HTML elements that accept text (for e.g. <p>, , <break>,), see: https://flaviocopes.com/html-text-tags/

References:

https://angular.io/guide/binding-syntax#types-of-data-binding
https://angular.io/guide/binding-syntax#binding-types-and-targets
https://angular.io/guide/binding-syntax#data-binding-and-html
https://angular.io/guide/binding-syntax#html-attributes-and-dom-properties

7 Class binding

Class binding can be used for binding to a single CSS class or to multiple classes

To create a single class binding, use the prefix class followed by a dot and the name of the CSS class—for example, `[class.sale]="onSale"`. Angular adds the class when the bound expression, `onSale` is truthy, and it removes the class when the expression is `falsy`—with the exception of undefined. Single class binding does not remove any existing classes on that element unless those classes have the same name as the one used in the binding.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v3.ts`

`app.component-v3.html`

`app.component-v3.css`

The CSS stylesheet defines a variety of style rules based on the `class` selector: https://www.w3schools.com/cssref/sel_class.asp

Change the values of `isThisForReal` and `isItDangerous` properties in the AppComponent and verify that the classes on the 2 paragraphs change accordingly.

For binding to multiple CSS classes, use `[class]="classExpression"`. The expression can be one of:

- A space-delimited string of class names.
- An object with class names as the keys and truthy or falsy expressions as the values.
- An array of class names.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v3-2.ts`

`app.component-v3-2.html`

Verify that the classes on the 3 paragraphs change accordingly. Notice that for the 3rd case where there are two classes with conflicting style rules now applied to the <p> element simultaneously (`normal` and `medium`), the style rule for the class that appears last in the list of class names takes effect.

You can change the truthy/falsy values of the object used in the 2nd approach and verify that the correct classes appear on the affected element.

Note that class binding is not technically property binding per se, since for a HTML DOM element, there is no `class` property per se, only `className` and `classList`.
https://www.w3schools.com/jsref/dom_obj_all.asp

In addition to class binding, we can also use the NgClass built-in directive to add or remove multiple classes (to be covered in a later lab).

References:

https://angular.io/guide/attribute-binding#binding-to-the-class-attribute
https://www.netjstech.com/2020/04/angular-class-binding-with-examples.html

8 Style binding

Style binding is used to set the `style` attribute. This is one of the 3 different ways to apply CSS style rules to elements within a HTML document:
https://www.tutorialrepublic.com/html-tutorial/html-styles.php

To create a single style binding, use the prefix style followed by a dot and the name of the CSS style property—for example, `[style.width]="width"`. Angular sets the property to the value of the bound expression, which is usually a string. Optionally, you can add a unit extension like `em` or `%`, which requires a number type.

For binding to multiple style simultaneously, use `[style]="styleExpression"`. The `styleExpression` can be one of:

- A string list of styles such as `"width: 100px; height: 100px; background-color: cornflowerblue;"`.
- An object with style names as the keys and style values as the values, such as `{width: '100px', height: '100px', backgroundColor: 'cornflowerblue'}`.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v4.ts`

`app.component-v4.html`

Verify that the style rules on the various elements involved change accordingly. You can change the value of the properties in the component referenced in the template and verify the changes are reflected in the DOM.

References:

https://angular.io/guide/attribute-binding#binding-to-the-style-attribute
https://www.tektutorialshub.com/angular/angular-style-binding/

9 Event binding

Event bindings use template statements which are executed when the target event occurs. In an event binding, Angular sets up an event handler for the target event. When the event is raised, the handler executes the template statement. The template statement typically invokes a component method which performs an action in response to the event, for e.g. storing a value from a HTML control element into a component property that will typically be a data model.

Lists of common events for event binding when this is used on a standard HTML element (such as button, text, etc):
https://www.positronx.io/useful-list-of-angular-7-event-types-for-event-binding/

The binding conveys information about the event (the event payload) through an event object named `$event`. The type of the event object is determined by the target event. If the target event is a DOM element event, then `$event` is a DOM event object:
https://www.w3schools.com/jsref/dom_obj_event.asp
The `$event` object can be passed as a parameter to a component method in the template expression of the event binding. The parameter type will be identical to the event type used in the binding (for e.g. `click`, `input`, `submit`, `keydown`, etc) shown in:
https://www.positronx.io/useful-list-of-angular-7-event-types-for-event-binding/

We can use the base Event type for the parameter
https://developer.mozilla.org/en-US/docs/Web/API/Event
and then cast this to the appropriate type, for e.g.
https://developer.mozilla.org/en-US/docs/Web/API/InputEvent
https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent
https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent
and then access properties and methods for these types where appropriate. An example:

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v5.ts`

`app.component-v5.html`

Switch to the Console view in Devtools to be able to see the console log output from the various handler methods that are invoked when the 3 types of events occur (`input`, `keydown` and `click`). Notice that when we bind to the `keydown` event for an input text form (instead of an `input` event), we are able to detect keypresses of all keys such as Caps Lock, shift, etc.

However, most of the time, we are interested in information regarding the element that triggered the event, rather than the event itself. All standard DOM event objects have a `target` property, a reference to the element that raised the event.
For the later versions of Angular, we need to cast `event.target` to the correct `HTMLxxxElement` type in order to extract the relevant properties of that particular element (for e.g. the content for the case of most <input> elements).
For a standard input type element (`<input type="text">`), we will need to cast to HTMLInputElement:
https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement
For a button element (<button>), we will need to cast to HTMLButtonElement:
https://developer.mozilla.org/en-US/docs/Web/API/HTMLButtonElement
Since all these specific elements inherit from the generic HTMLElement, we can also access any of the properties here that might be useful for our purposes:
https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v6.ts`

`app.component-v6.html`

Verify that the appropriate console log output occurs corresponding to interaction with the form elements (text field, checkbox and button).

Note that most use cases for event binding for button will not pass the $event object since all that is required typically is just to know that a specific button has been clicked and to execute some logic to handle that event in the component method.

References:

https://angular.io/guide/event-binding#binding-to-events
https://angular.io/guide/event-binding-concepts#how-event-binding-works
https://angular.io/guide/event-binding-concepts#handling-events

https://www.tektutorialshub.com/angular/event-binding-in-angular

9.1 Template reference variables for event binding

The template reference variable can be used to reference any DOM element, component or a directive directly. It can then be subsequently accessed elsewhere in the template as well as being passed onto a method in the component.

It is often used as an alternative to the `$event` object in order to access the event content as it eliminates the need for knowledge and typing of the `$event` object prior to use (as is the case in event binding). The template variable when passed as a parameter to the component method through event binding is already of the correct HTMLxxxElement type and no further casting is required as is the case with the `$event` object

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v7.ts`

`app.component-v7.html`

References:

https://angular.io/guide/template-reference-variables#template-variables
https://angular.io/guide/template-reference-variables#syntax

9.2 Combining interpolation, property, class and event binding

We can combine the different types of binding we have covered so far to perform dynamic changes to the template DOM in response to events that occur. This will follow the basic pattern of:

- Using event binding to change some component property / template variable in response to a particular event
- Using property / class / style binding or interpolation to transfer the updated value from this property back to the appropriate element in the template

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v8.ts`

`app.component-v8.html`

`app.component-v8.css`

Note that an event binding is required for Angular to detect an event and perform its change detection process, where by Angular traverses the view hierarchy and updates the view. Sometimes, you may wish to just access the updated `value` property of the template variable in the template itself without actually calling a component method or updating a component property. In that case, you will need to include a "dummy" event binding statement (`(keyup)="0"`) :

10 Two way binding with ngModel

In order to use ngModel, you will have to import the FormsModule and add it to the `imports` list in the root module AppModule (app.module.ts)

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v9.ts`

`app.component-v9.html`

`app.module-v9.ts`

Two-way binding is useful for situations where the value of a property in a component is synchronized to the value of a HTML element (typically a form control like a text field), where by changes in the property are reflected in the value and vice versa.

This can be achieved by combining the standard event and property bindings together on a single element.

Alternatively, we can use the ngModel and ngModelChange event and property bindings to generalize the creation of these two-way bindings. In this case, the `$event` object no longer refers to the input event, but rather to the latest value for the <input> element - therefore there is no need for all the additional preliminary type casting that needs to be done for the conventional event binding to obtain this latest value.

Finally, the ngModel and ngModelChange event and property bindings can be combined into the most common form of two-way binding syntax: banana-in-the-box syntax.

We can still use ngModel and ngModelChange event and property bindings if we do not want complete synchronization between the element value and the component property. For e.g. we might want to perform a preliminary operation on the element value before storing it in the component property or vice versa.

ngModel is primarily used in template-driven forms.

References:

https://angular.io/guide/built-in-directives#displaying-and-updating-properties-with-ngmodel
https://www.netjstech.com/2020/04/angular-two-way-data-binding-with-example.html