Angular2 Notes

* Typescript
  + Declaring types
    - myVar: any
    - myVar: number[]
    - Type of a variable cannot be changed once defined or declared
  + Classes
    - class Car {

constructor() { }

<properties and methods> }

* + - car: Car = new Car()
  + Application is written is TypeScript and converted to JavaScript (and outputs it in the distribution folder)
    - Browsers cannot understand TypeScript
* When importing TypeScript files, no need to put ‘.ts’ at the end
* Components
  + - Application is built out of various components
    - app.component is the root component
    - app.module.ts bootstraps the app component
    - Each component consists of 3 parts
      * componentName.component.ts (logic)
      * componentName.component.html (view/template)
      * componentName.component.css (styles)
    - ng generate component component\_name
    - Nested components
      * import { ComponentClassName } from ‘./<path>/index’ (inside the parent component)
        + The index.ts exports everything in ./<path>/something.component.ts
      * Add ‘directives: [ComponentClassName]’ in the @Component decorator of the parent component
* CSS files (for the entire application, if any) should be put inside the ‘public’ directory and can be referenced in the index.html file as “/styles.css”
* **Data Flow**
  + **Into the view**
    - Interpolation (evaluate content inside curly braces)
      * cannot contain assignments
      * cannot access global JS variables
    - Property binding
      * Attribute (HTML) vs. Property (DOM)
        + Attributes initialize DOM properties
        + Once attribute is initialized, it cannot be changed
      * <input [id]=“<someVar>”> (OR) <input bind-disabled=“<boolVar>”>
        + How to set disabled attribute for AngularJS?
        + When to use “{{ varName }}” and when to use just “varName” in AngularJS?
        + <https://stackoverflow.com/questions/18487480/angular-expression-in-attribute>
    - Class binding
      * Regular class attribute becomes a dummy in the presence of class binding: [class]=“<someVar>” OR [class.className] = “<boolVar>”
      * [ngClass]=“<classObj>”, where classObj is an object of the form,

{ “className”: bool }

* + - Style binding
      * [style.propertyName]=“<someVar>”
      * [ngStyle]=“<styleObj>”, where styleObj is an object of the form,

{ property: “value” }

* + **Out of the view**
    - Event binding
      * (eventName) = “func($event)” (OR) (eventName) = “someVar=‘Hello, World!’ ”
        + $event is a special variable that provides information about the DOM event that was raised
    - Template reference variables
      * <input **#myInput** ... >
      * myInput can be passed as an argument (to a function) from the HTML page
      * All DOM properties of this element can be accessed using myInput
  + 2-way data binding
    - app.module.ts
      * import forms module
      * add FormsModule to the imports array
    - <input [(ngModel)]=“varName”>
* **Pipes** {{ expr | pipe}} – to transform data in the view
  + String pipes
    - lowercase
    - uppercase
    - titlecase
    - slice:start:end-1 (0-based indexing)
  + Number pipe {{ 5.678 | number:‘<min-digits>.<min-digits>-<max-digirs>’ }} ({{ 5.678 | number:‘3.2-3’ }})
  + Percent pipe {{ 25 | percent }}
  + Currency pipe {{ 25 | currency: ‘GBP’: ‘code’}}
  + Date pipe
    - {{ date | date:short }} (where the 1st date is a Date object)
    - shortDate
    - shortTime
    - *short* can be replaced with *medium* or *long*
* **Service** – a class with a specific task (filename.service.ts; class: filenameService)
  + ng g s <service>
  + Applications
    - To share data
    - Implement application logic (which can be used by various components)
    - External interaction (connecting to DB)
  + Dependency Injection
    - Assume we have 3 classes – Engine, Tyre and Car. Car depends on Engine and Tyre classes
    - Code without DI
      * The Car class creates instances of Engine and Tyre in its constructor
      * Code is not flexible
        + If the parameters to the constructors of Engine and/or Tyre change, the change has to be made in the code of the Car class as well
      * Testing is difficult
    - DI as a design pattern
      * Pass dependencies as parameters
      * Disadvantage: We need to manually create all dependencies for a class(?) as they need to be passed as parameters. This is difficult if the number of dependencies are large (and recursive)
    - DI as a framework (provided by Angular)
      * Regsiter dependencies in an Injector
      * **Define service class** (use @Injectable decorator since a service may have another service as dependency) **->** **register with injector** (Provider metadata) **-> declare as dependency wherever required**
* Lifecycle Hooks
  + ngOnChanges – <https://www.tektutorialshub.com/angular/angular-ngonchanges-life-cycle-hook/>
  + All
    - <https://indepth.dev/posts/1494/complete-guide-angular-lifecycle-hooks>
    - <https://blog.logrocket.com/angular-lifecycle-hooks/>
* Component Interaction
  + @Input and @Output decorators – <https://angular.io/guide/inputs-outputs>
  + 2-way binding – <https://stackoverflow.com/questions/41464871/update-parent-component-property-from-child-component-in-angular-2>
  + Comprehensive – <https://stackoverflow.com/questions/37587732/how-to-call-another-components-function-in-angular2>
  + Change Detection – <https://www.sitepoint.com/change-detection-angular/>
* HTTP and Observables (Observable => HTTP response)
  + app.module.ts

import { HTTPClientModule } ...

Include HTTPClientModule in the imports property

* + In the service, provide the HTTPClient class in the constructor and make a HTTP get request
  + Cast observable into suitable format
  + Subscribe to observable
  + Using promises – <https://stackoverflow.com/questions/50303033/how-to-return-a-promise-from-subscribe-in-angular-5>

Doubts

* ngModel
* Sharing data between siblings – <https://stackoverflow.com/questions/43940351/how-do-i-share-data-between-sibling-components-in-angular>