14/08/2018 – Angular – Gilad

1) npm init -y

2) npm install @angular/cli – add –g flag for global, otherwise angular will only work on the current directory.

2\*) node "node\_modules/@angular/cli/bin/ng"

to access ng on stupid PCs who don't work right.

src/app/app.component.ts:

@Component: (@ = Decorator)

selector – the html element angular creates and searches to fill with

templateUrl and its style, styleUrls.

After that, there is an export of a class with items in it: these are like "props". We access those props in the HTML item with {{ }}

3) ng generate component \*name\*

It will generate a new component, with the same 4 elements of every app: html, css, js and the test for the component.

It will come with a constructor and a life cycle function: ngOnInit() – which is like ComponentDidUpdate.

In the app.component.html item, we need to add the element: <app-\*name\*> in order to render it.

We can create a class like normal, with a .ts in the src/app/

now, in order to show the class items, we need to access them individually in the html.

It's all **Pipes**!

In Angular, a pipe is a function that gets a parameter and returns something.

Example: {{\*string\* | [uppercase](https://angular.io/api/common/UpperCasePipe)}} will result in the entered string to be IN ALL CAPS.

We can also create our own pipes!

Binding (Or: the red string of fate)

1 way: output: () (click makes a callback, thus "calling the parent"),   
input: [] (puts stuff on the html element)

2 way binding: (input/output) [()] ( [(ngModel)] )

Binding helps us "glue" the class items on the html to the "real" ones in the class.ts, like when we want to update in real-time with an input box.

In order to add ngModule, we need to add it to app.module.ts in the @NgModule -> imports

Let's make a list (Or: use loops)

We add a <ul> and just one <li>, like so: <li \*[ngFor](https://angular.io/api/common/NgForOf)="let \*itemName\* of \*array\*">

Event Binding: like click!

<li \*ngFor="let hero of heroes"

[class.selected]="hero === selectedHero"

(click)="onSelect(hero)">

<span class="badge">{{hero.id}}</span> {{hero.name}}

</li>

Services: a component should not have logic that fetches data, it should only get it, somehow (like from an outer source) and render it. A service is just that: logic that does something, like fetching data, that a component can use.

Routing:

Navigating between different pages in a single page application. Usually we see a home page (landing page) and there's a nav (a menu) that gives us links to different pages. Instead of moving from page to page (refresh into a different page), we can simply show and hide components with the Router. The Router needs a path and a component to render, and if the URL of the browser matches the "path" in the Router, it will show the corresponding component and hides the rest.

We show these with <router-outlet>

and allow nav with

<nav>

<a routerLink="/heroes">Heroes</a>

</nav>

Life hack: adding " + " before a number string makes it a real bone-a-fide number!