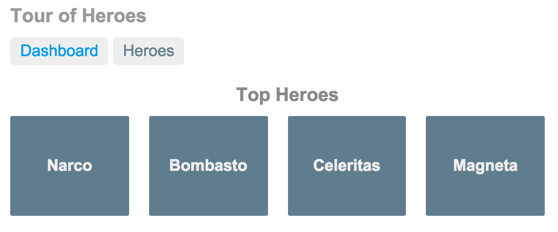
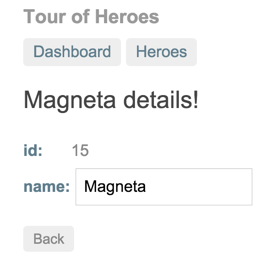
1. use one-way data binding for read-only data.
2. Our final application looks like,



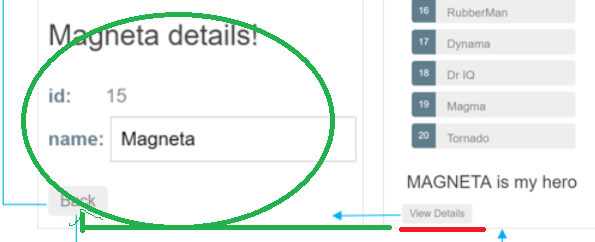
1. You can click the two links above the dashboard ("Dashboard" and "Heroes") to navigate between this Dashboard view and a Heroes view.
2. If you click the dashboard hero "Magneta," the router opens a "**Hero Details**" view where you can change the hero's name.



1. Clicking the "Back" button returns you to the Dashboard. Links at the top take you to either of the main views. If you click "Heroes," the app displays the "Heroes" master list view.
2. When you click a different hero name, the read-only mini detail beneath the list reflects the new choice.



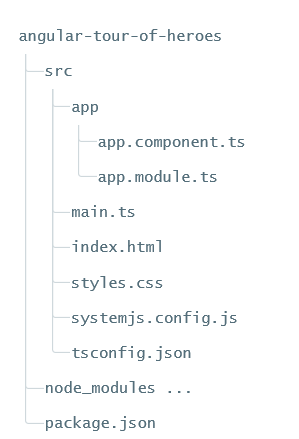
1. You can click the "View Details" button to drill into the editable details of the selected hero.

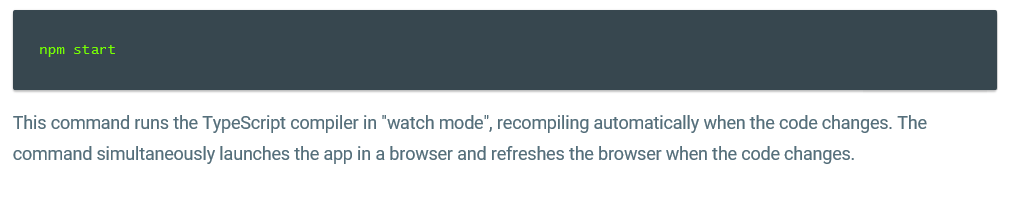
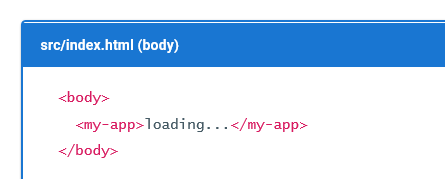


1. The following diagram captures all of the navigation options.

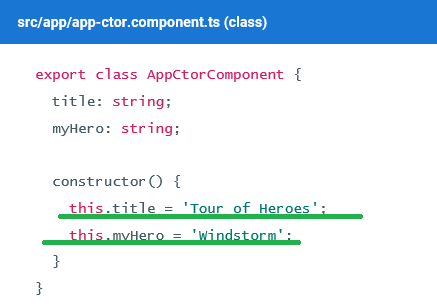


1. The file structure should look like this:

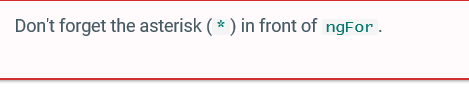


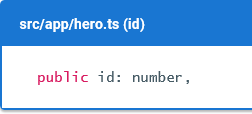
1. 
2. The double curly braces are Angular's interpolation binding syntax.
3. Property binding helps show app data in the UI
4. The template is a multi-line string within ECMAScript 2015 backticks (`). The backtick (`)—which is not the same character as a single quote (')—allows you to compose a string over several lines, which makes the HTML more readable
5. 🡺

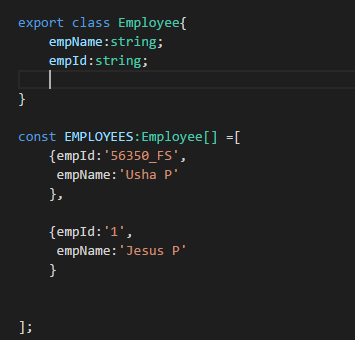
When you bootstrap with the AppComponent class (in main.ts), Angular looks for a <my-app> in the index.html, finds it, instantiates an instance of AppComponent, and renders it inside the <my-app> tag.

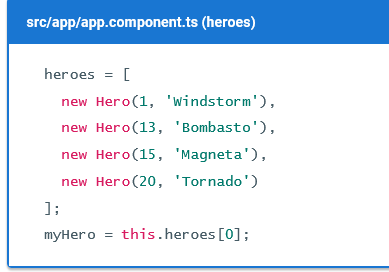
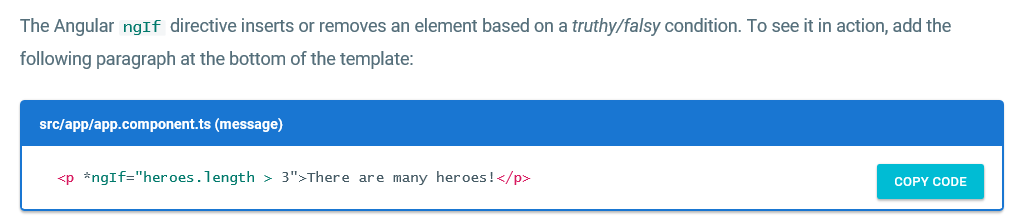
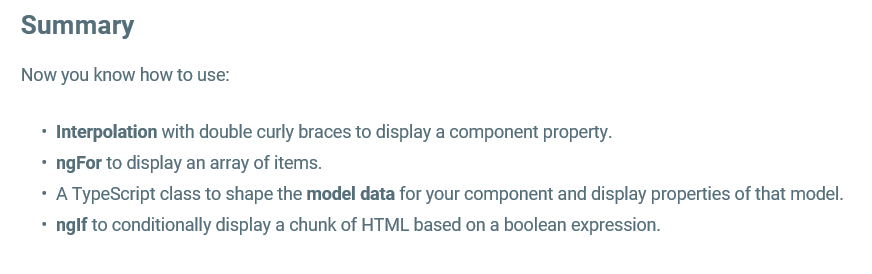
1. Template inline or template file? 🡺 You can store your component's template in one of two places. You can define it inline using the template property, or you can define the template in a separate HTML file and link to it in the component metadata using the @Component decorator's templateUrl property.
2. 
3. 

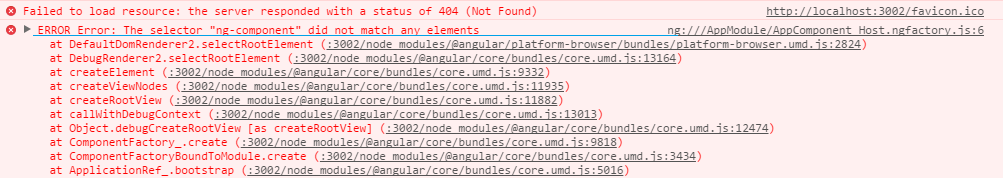




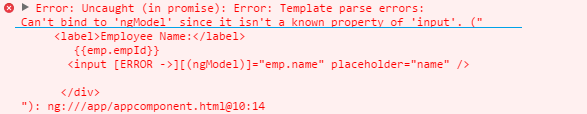
1. 

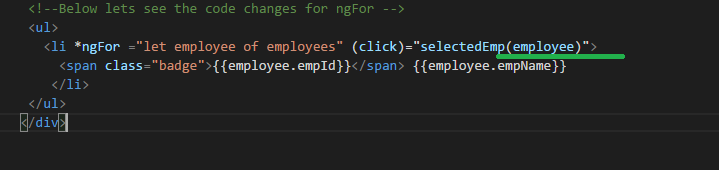
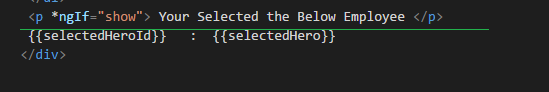


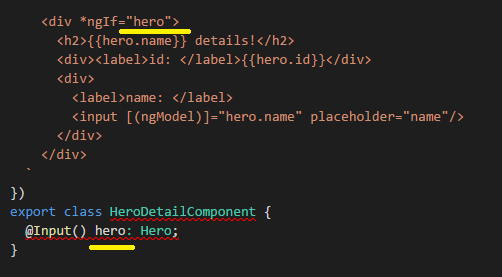
1. 
2. 
3. Don't forget the leading asterisk (\*) in \*ngIf.
4. 
5. Although NgModel is a valid Angular directive, it isn't available by default. It belongs to the optional FormsModule
6. import the FormsModule symbol from the @angular/forms library.
7. @NgModule metadata's imports array, **which contains the list of external modules that the app uses**
8. Practical Scenario🡺 I had not used the Selector in my AppComponent this resulted in the following error



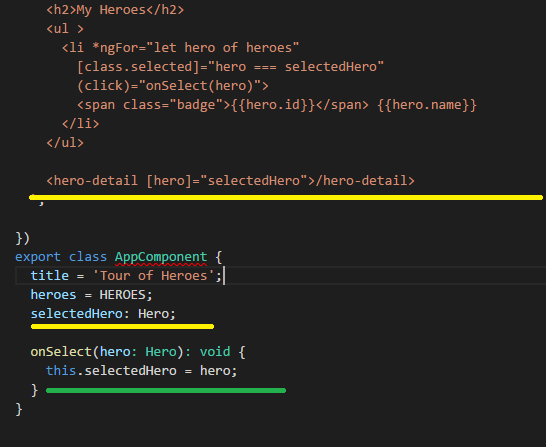
1. Import BrowserModule in the root module and CommonModule in other modules where you want to use common directives
2. CAN I USE THE "const" key word inside the class? 🡺 I was getting some error in the Studio
3. Without the FormModule, the Error in the Console is🡺

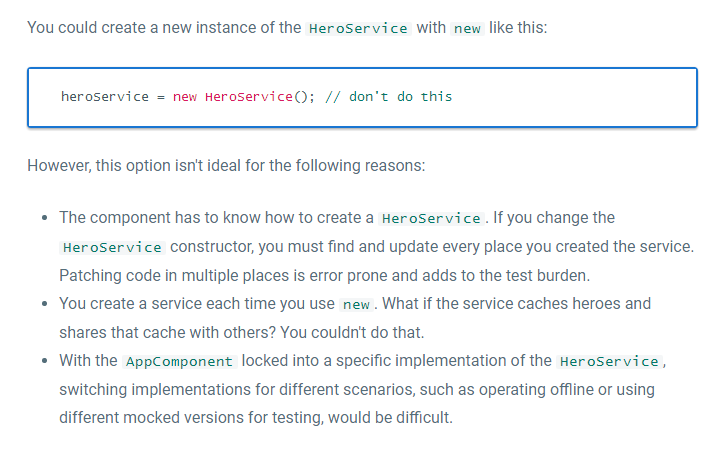


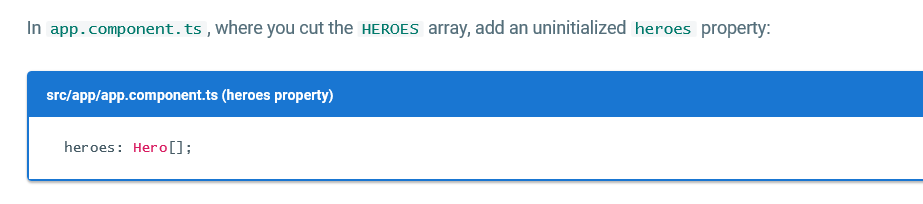
1. 
2. Syntax for \*ngIf is 🡺
3. The @Component decorator provides the Angular metadata for the component.
4. In general, the declarations array contains a list of application components, pipes, and directives that belong to the module.A component must be declared in a module before other components can reference it
5. Note🡺I thought that \*ngIf can accept value only true or false, but its actually like javaScript which accepts, non zero +ve and –ve no, non empty string, any object which is not undefined, for example see the below screen shot



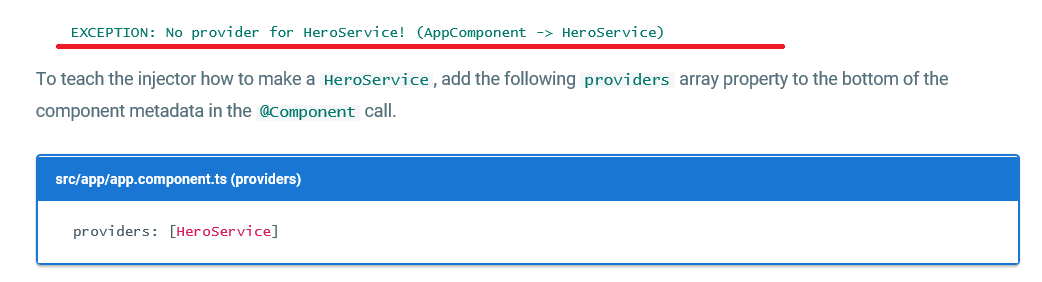
See here it's Quite Different way how the data is sent from parent to child compared to our Video Tutorials



1. Although the HeroService doesn't have any dependencies at this moment, applying the @Injectable() decorator ​from the start ensures consistency and future-proofing.
2. 
3. Don't forget the parentheses. Omitting them leads to an error that's difficult to diagnose.
4. 



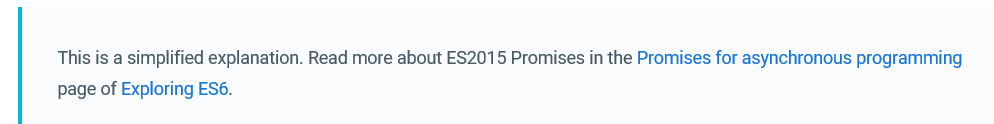
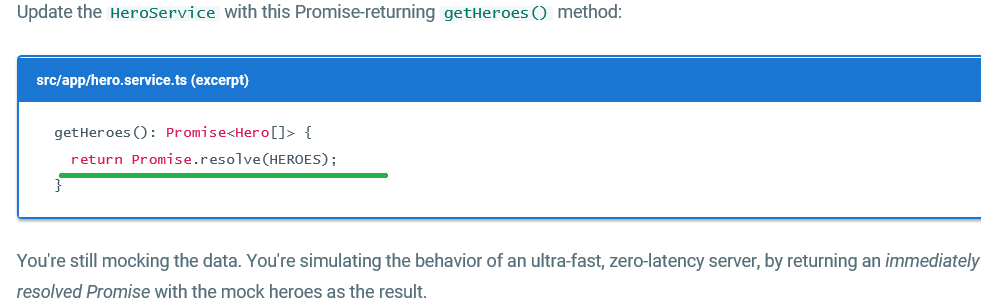


1. 
2. The providers array tells Angular to create a fresh instance of the HeroService when it creates an AppComponent. The AppComponent, as well as its child components, can use that service to get hero data.
3. The ngOnInit lifecycle hook
4. AppComponent should fetch and display hero data with no issues.
5. You might be tempted to call the getHeroes() method in a constructor, but a constructor should not contain complex logic, especially a constructor that calls a server, such as as a data access method. The constructor is for simple initializations, like wiring constructor parameters to properties.
6. To have Angular call getHeroes(), you can implement the Angular *ngOnInit lifecycle hook*. Angular offers interfaces for tapping into critical moments in the component lifecycle: at creation, after each change, and at its eventual destruction.
7. Each interface has a single method. When the component implements that method, Angular calls it at the appropriate time.
8. 

Async services and Promises

1. The HeroService returns a list of mock heroes immediately; its getHeroes() signature is synchronous.
2. Eventually, the hero data will come from a remote server. When using a remote server, users don't have to wait for the server to respond; additionally, you aren't able to block the UI during the wait.
3. To coordinate the view with the response, you can use *Promises*, which is an asynchronous technique that changes the signature of the getHeroes() method.

The hero service makes a Promise

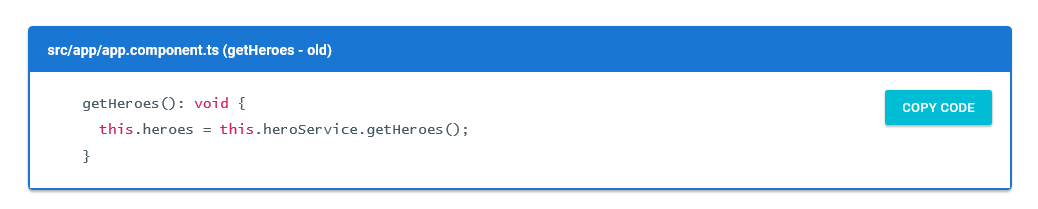
1. A Promise essentially promises to call back when the results are ready. You ask an asynchronous service to do some work and give it a callback function. The service does that work and eventually calls the function with the results or an error.
2. 
3. 

Act on the Promise

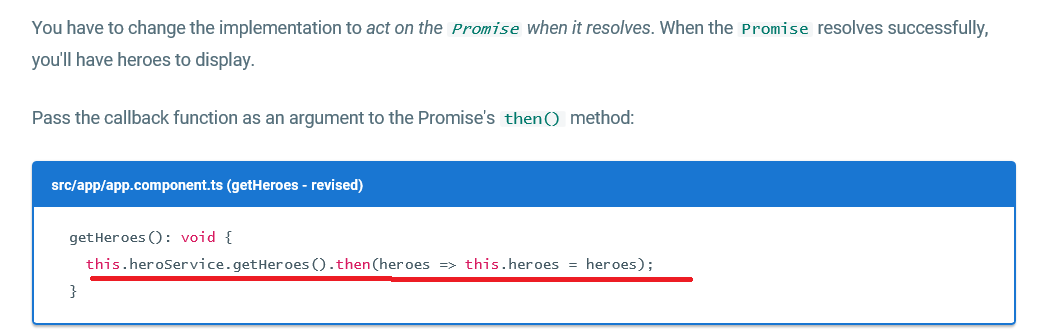
1. As a result of the change to HeroService, this.heroes is now set to a Promise rather than an array of heroes.

Ms 🡺 thus now let’s see the differece/ comparison between older code and latest revisited/modified code in app component

OLDER🡺



LATEST🡺



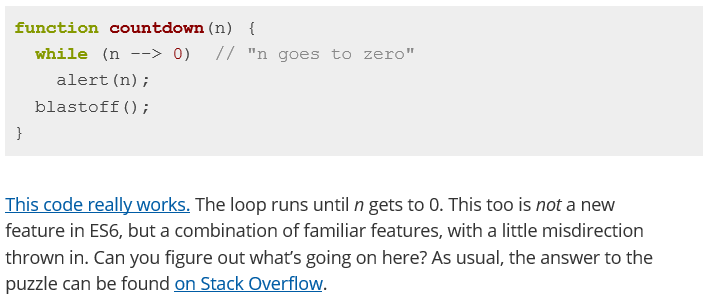
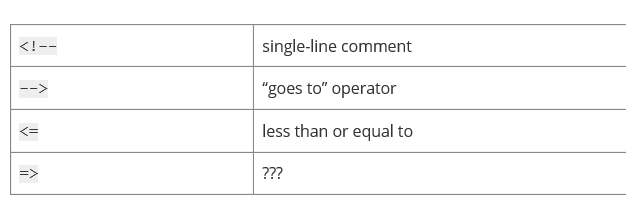
Ms🡺 lets see few concept with respect to Arrow functions,

* 1. Arrow functions make it easy to write anonymous functions, and also bind to the current context.
  2. Example

Case 1: let’s see the syntax for anonymous functions

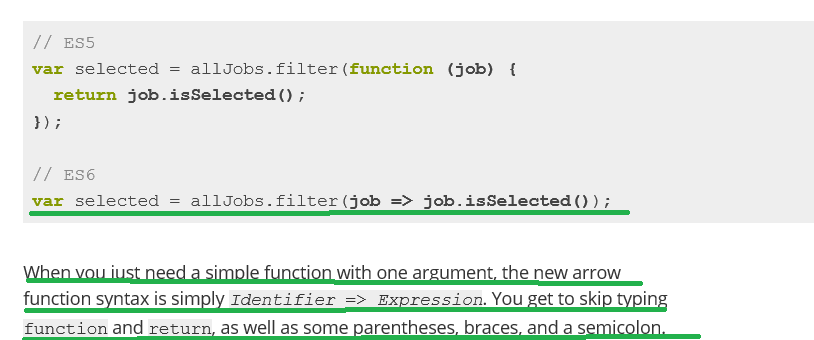


1. Arrow function in Es6🡺 <https://hacks.mozilla.org/2015/06/es6-in-depth-arrow-functions/>
   1. -->

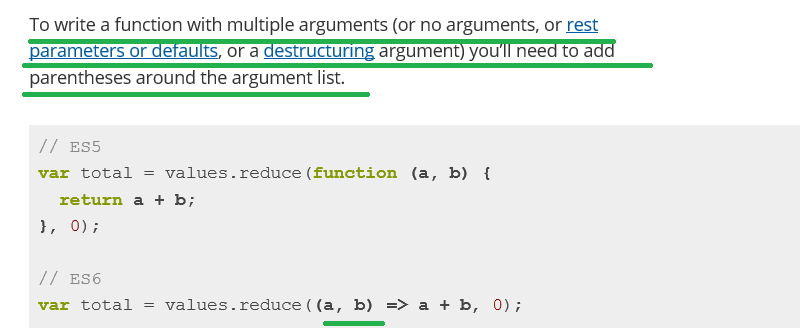
* This arrow indicates a comment *only* when it appears at the start of a line. That’s because in other contexts, --> is an operator in JS, the “goes to” operator!
* 
* 
* What happened to =>? Today, we find out. First, let’s talk a bit about functions.

1. A new arrow in your quiver🡺ES6 introduces a new syntax for writing functions.

Example 1:



Example 2:



Example 3:

