Pipes

- Pipes allow you to transform values in templates before they are outputed to the view.
- Pipes were formerly known as filters in Angular 1.x
- A pipe is defined using the `@pipe` class decorator
- The pipe decorator takes name as a parameter defining the name of the pipe: `@pipe({ name: 'myPipe' })`
- Every pipe class has a `transform` method that transforms input to outputs:
  - The first parameter is the input to the pipe
  - The second parameter is the list of arguments passed to the pipe
- Give the following pipe in a template: `{{ data | somePipe:1:'px'}}` :
  - `data` is the input to pipe -- the first parameter of the transform method
  - `[1, 'px']` is the arguments to the pipe -- the second parameter of the transform method
- A pipe can be as simple as:

```
@pipe({name: 'simplePipe'})
class MyPipe {
  transform(input, args) { return input + 'px'; }
}
```

- If you want to use a pipe, you need to register your pipe class with the components in the pipes array:

```
@component({
  selector: '...',
  pipes: [MyPipe] // adding pipe to the array of pipes.
})
class MyComponent {}
```

- Pipes can be chained: `input | pipe1 | pipe2 | pipe3`

- `input | pipe1 : output1`
- `output1 | pipe2: output2`
- `output2 | pipe3 : finalOutput`

Basic Pipe

Let's make a basic pipe called `pixel` that takes a value as the input and appends 'px' to the end of it. The project files for this section are in angular2-intro/project-files/angular-examples/pipes/basic-pipe.

Start by making a copy of the "starter" folder and call it "basic-pipe" and put it in `project-files/angular-examples`. Then, open the folder in VSCode: `code project-files/angular-examples/basic-pipe` and start the build with `command + shift + b`.

Then, create a file for the pipe and call it `pixel.pipe.ts` in the root of the project.

After that we need to do couple of things to define the pipe:

- Import the Pipe Class Metadata from angular core: `import {Pipe} from 'Angular/core'`
- Then create a class defining the Pipe:

```
class PixelPipe {

}
```

- Implement the `transform` method in the class:

```
class PixelPipe {
  transform(input) {
    return input + 'px';
  }
}
```

- After implementing the method, we need to decorate the class and give the pipe a name that we want to use in our templates:

```
@Pipe({name: 'pixel'}) // <- adding the decorator
class PixelPipe {
  transform(input) {
    return input + 'px';
  }
}
```

- As the last step we are going to export the class by putting the `export` keyword behind the class:

```
...
export class PixelPipe {
  ...
}
```

Now, your file should look like the following:

```
import {Pipe} from 'angular2/core';
@Pipe({name: 'pixel'}) // <- adding the decorator
export class PixelPipe {
  transform(input) {
    return input + 'px';
  }
}
```

Now, let's go back to the `main.ts` file and import our pipe:

```
import {Component} from 'angular2/core';
import {bootstrap} from 'angular2/platform/browser';
```

```
3 import {PixelPipe} from './pixel.pipe'; // <- importing pipe
```

After importing our pipe, we should register it with our component by adding it to the `pipes` array:

```
1 @Component({
2   selector: 'app',
3   templateUrl : 'templates/app.tpl.html',
4   pipes: [PixelPipe] // <- registering the pipe
5 })
```

Now that we have registered the pipe, we can use it in our template in `templates/app.tpl.html` :

```
1 <h1>{{ name }}</h1>
2 <p>Pixel value: {{ 25 | pixel }}</p>
```

You should be all set now. You can set the url in your `launch.json` file and hit F5:

```
1 ...
2 "url": "http://localhost:8080/project-files/angular-examples/basic-pipe/index.
3 ...
```

If your server is running you should be able to see the following output:

Running the pixelPipe in the browser

Chaining Pipes

Let's continue where we left off with the "pixelPipe" and add another pipe called "round" that rounds down given values, that is:

```
25.3 | round | pixel -> 25px
```

The project files for this section are in angular2-intro/project-files/angular-examples/pipes/pipe-chaining.

We are going to add the "roundPipe" to our "basic-pipe" project. Let's get started by adding the `round.pipe.ts` file in the root of the project:

```
import {Pipe} from 'angular2/core';
@Pipe({name: 'round'})
export class RoundPipe {
  transform (input) {
    return Math.floor(+input); // <- convert input to number and then floor it
  }
}
```

This Pipe is not complicated at all. We are just returning the floor of the input. We are also converting the input to number by putting a `+` before input.

Now, let's import the pipe into our `main.ts` file:

```
import {Component} from 'angular2/core';
import {bootstrap} from 'angular2/platform/browser';
import {PixelPipe} from './pixel.pipe';
import {RoundPipe} from './round.pipe'; // <- importing `RoundPipe`
```

and then we have to add the pipe to the list of pipe array:

```
@Component({
  selector: 'app',
  templateUrl : 'templates/app.tpl.html',
  pipes: [PixelPipe, RoundPipe] // <- registering the pipe
})
```

after that we are going to add the following to our `templates/app.tpl.html` file:

```
<p>Pixel value: {{ 34.4 | round | pixel }}</p>
```

After running the app you should see `34.px` as the output on the page.

Pipes with Parameters

In this section we are going to extend our 'pixel' pipe to accept an optional parameter to set the unit. As a result, we are going to rename the 'pixel' pipe to 'unit' to make it more generic. This pipe will take the unit as an optional argument. If no argument is passed, it will default to 'px'. That is:

```
25 | unit -> 25px
25 | unit:'em' -> 25em
34.5 | round | unit:'%' -> 34%
```

You can look at the project files in angular2-intro/project-files/angular-examples/pipes/pipe-unit.. AFter refactoring the name of the Pipe, we just need to change the implementation of the "UnitPipe":

`unit.pipe.ts`

```
import {Pipe} from 'angular2/core';
@Pipe({name: 'unit'})
export class UnitPipe {
  transform(input, args:string) {
    const unit = args[0] || 'px';
    return input + unit;
  }
}
```

- On line 5, we are grabbing the first parameter that is passed in and setting it to the `unit` variable. And if the value is not set, we are setting 'px' as the default value.
- And finally we are returning `input + unit` .

That's basically all we have to do. Note that you can pass multiple parameters separated by `:` and they all become available in the `args` array. So if you wanted to expand this pipe, this is how your template would look like:

```
1  {{ 25 | unit:'em':2}}
```

And the `args` array would be: `['em', 2]`.

Async Pipes

Async Pipes can be used for values that will be resolved after some asynchronous operation like getting a value after making a http call.

TODO