Pipes

Overview

Pipes are used to transform data, when we *only* need that data transformed in a template.

If we need the data transformed *generally* we would implement it in our model, for example we have a number `1234.56` and want to display it as a currency such as `$1,234.56`.

We could convert the number into a string and store that string in the model but if the only place we want to show that number is in a view we can use a pipe instead.

We use a pipe with the `|` syntax in the template, the `|` character is called the *pipe* character, like so:

```
{{ 1234.56 | }}
{{ 1234.56 | currency : 'USD' }}
```

This would take the number `1234.56` and convert it into a *currency string* for display in the template like `USD1,234.56`.

We can even *chain* pipes together like so:

```
{{ 1234.56 | currency: 'USD' | lowercase }}
```

The above would print out `usd1,234.56`.

Pipes are just like `filters` in Angular 1

In this section you will learn:

- How to use the set of built-in pipes provided by Angular.
- How to create your own custom pipes.

Built-in Pipes

In this lecture we will cover all of the built-in pipes provided by Angular appart from the *async pipe* which we will cover in detail in a later lecture.

Learning Objectives

- Know the different built-in pipes provided by Angular and how to use them.

Pipes provided by Angular

Angular provides the following set of built-in pipes.

CurrencyPipe

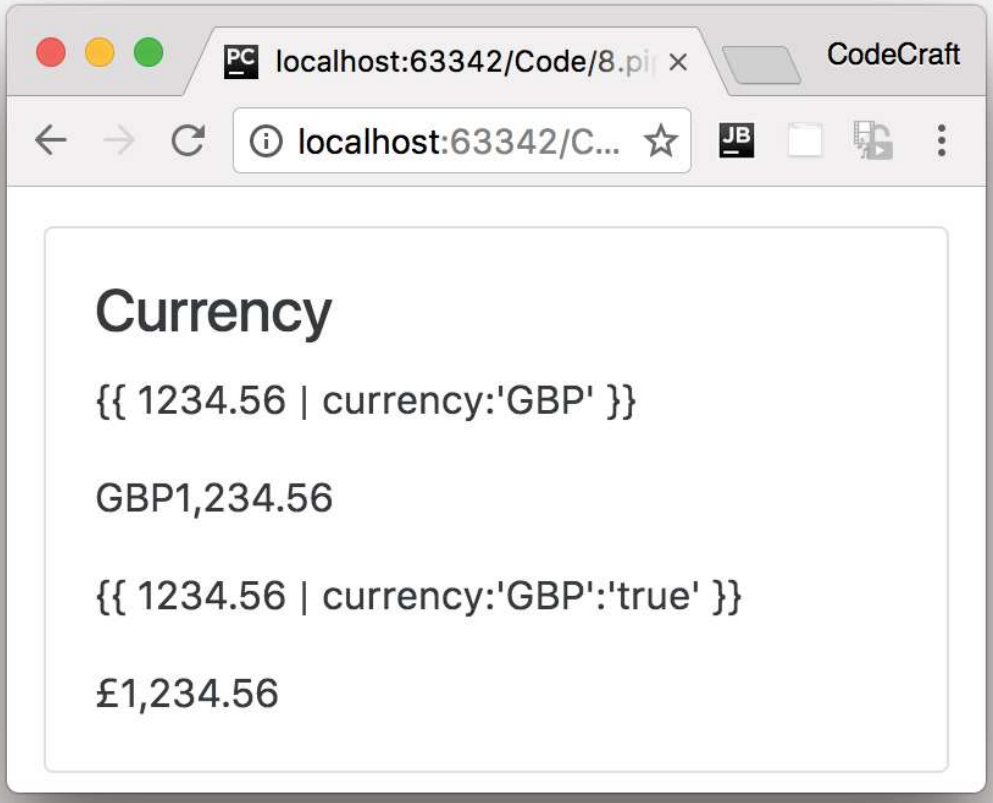
This pipe is used for formatting currencies. Its first argument is an abbreviation of the currency type (e.g. "EUR", "USD", and so on), like so:

```
{{ 1234.56 | currency:'GBP' }}
```

The above prints out `GBP1,234.56`, if instead of the abbreviation of `GBP` we want the currency symbol to be printed out we pass as a second parameter the boolean `true`, like so:

```
{{ 1234.56 | currency:"GBP":true }}
```

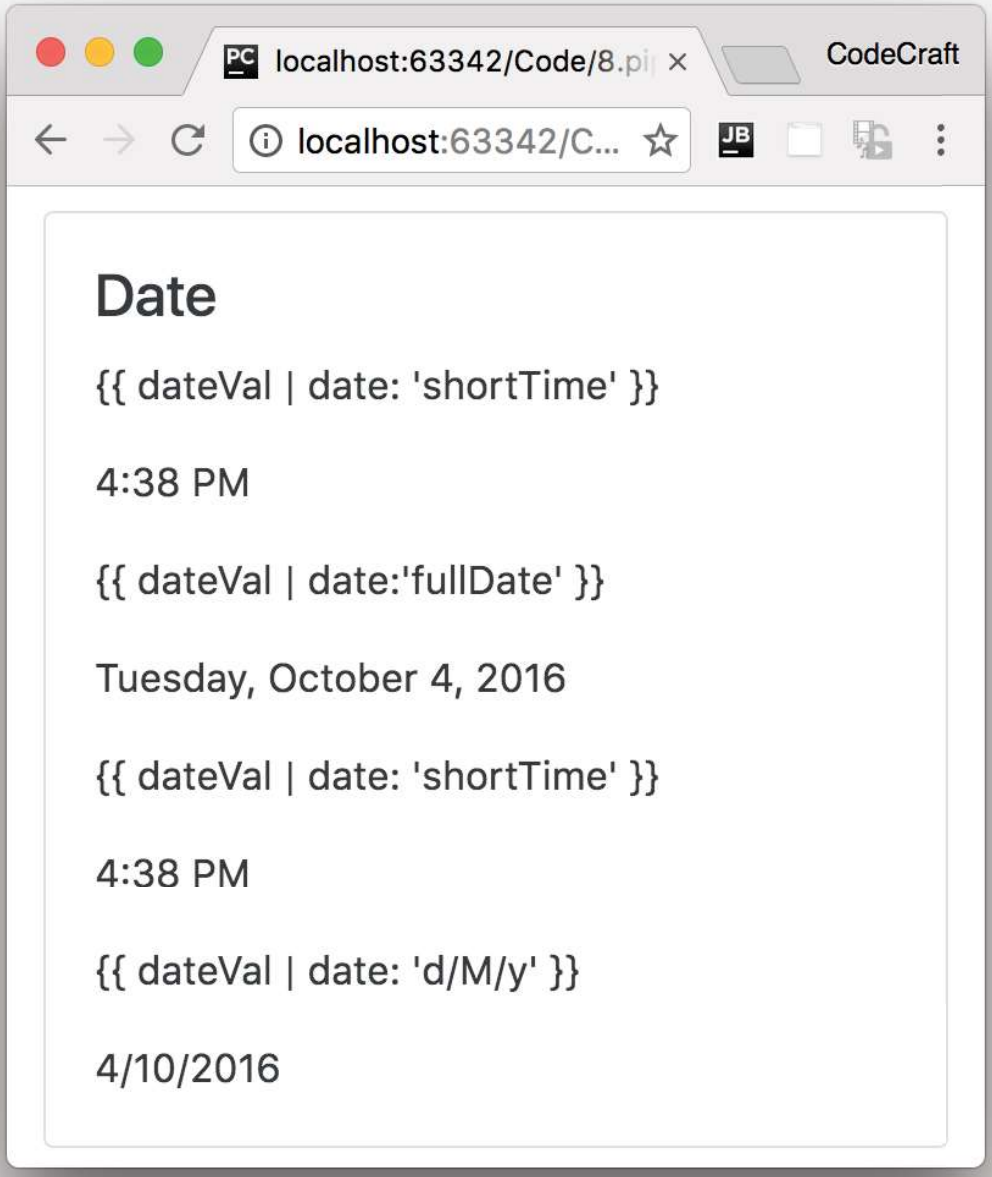
The above prints out `£1,234.56`.




```
<div class="card card-block">
  <h4 class="card-title">Currency</h4>
  <div class="card-text">

    <p ngNonBindable>{{ 1234.56 | currency:'GBP' }}</p>
    <p>{{ 1234.56 | currency:"GBP" }}</p>

    <p ngNonBindable>{{ 1234.56 | currency:'GBP':'true' }}</p>
    <p>{{ 1234.56 | currency:"GBP":true }}</p>
  </div>
</div>
```

DatePipe

This pipe is used for the transformation of dates. The first argument is a format string, like so:


localhost:63342/Code/8.pi
CodeCraft
localhost:63342/C...
Date
{{ dateVal | date: 'shortTime' }}
4:38 PM
{{ dateVal | date:'fullDate' }}
Tuesday, October 4, 2016
{{ dateVal | date: 'shortTime' }}
4:38 PM
{{ dateVal | date: 'd/M/y' }}
4/10/2016

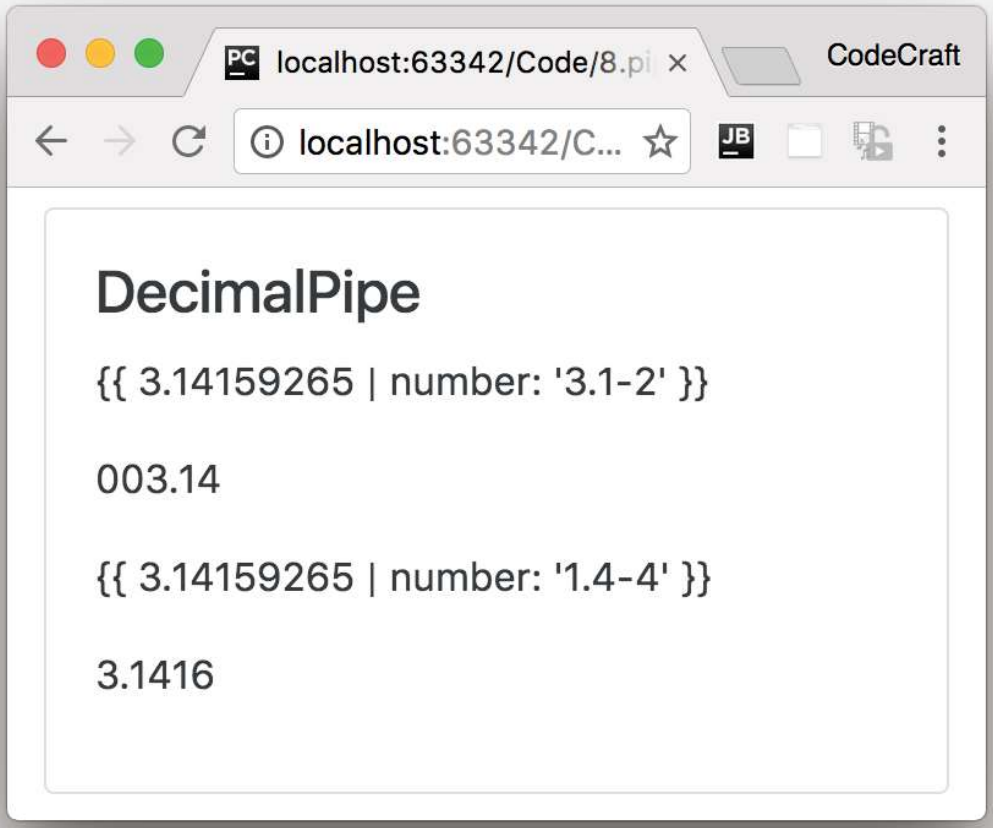
```
<div class="card card-block">
  <h4 class="card-title">Date</h4>
  <div class="card-text">
    <p ngNonBindable>{{ dateVal | date: 'shortTime' }}</p> ①
    <p>{{ dateVal | date: 'shortTime' }}</p>

    <p ngNonBindable>{{ dateVal | date:'fullDate' }}</p>
    <p>{{ dateVal | date: 'fullDate' }}</p>

    <p ngNonBindable>{{ dateVal | date: 'd/M/y' }}</p>
    <p>{{ dateVal | date: 'd/M/y' }}</p>
  </div>
</div>
```

① `dateVal` is an instance of `new Date()`.

DecimalPipe

This pipe is used for transformation of decimal numbers.

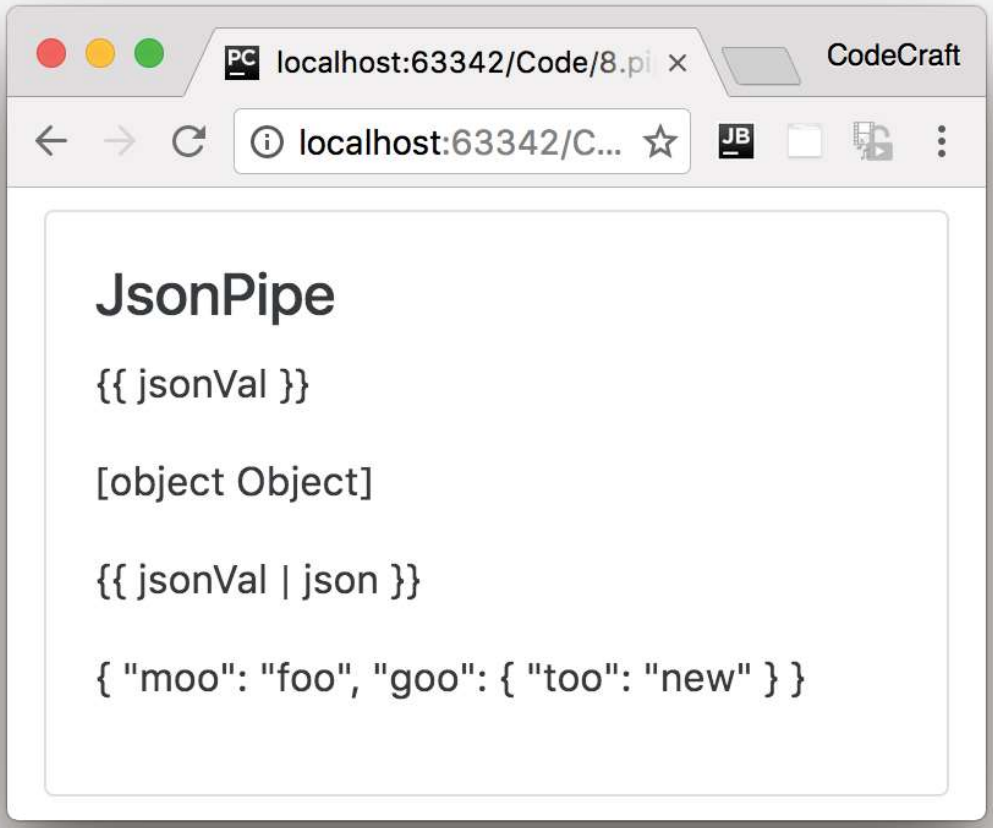
The first argument is a format string of the form "{minIntegerDigits}. {minFractionDigits}-{maxFractionDigits}", like so:




```
<div class="card card-block">
  <div class="card-text">
    <h4 class="card-title">DecimalPipe</h4>
    <p ngNonBindable>{{ 3.14159265 | number: '3.1-2' }}</p>
    <p>{{ 3.14159265 | number: '3.1-2' }}</p>

    <p ngNonBindable>{{ 3.14159265 | number: '1.4-4' }}</p>
    <p>{{ 3.14159265 | number: '1.4-4' }}</p>
  </div>
</div>
```

JsonPipe

This transforms a JavaScript object into a JSON string, like so:

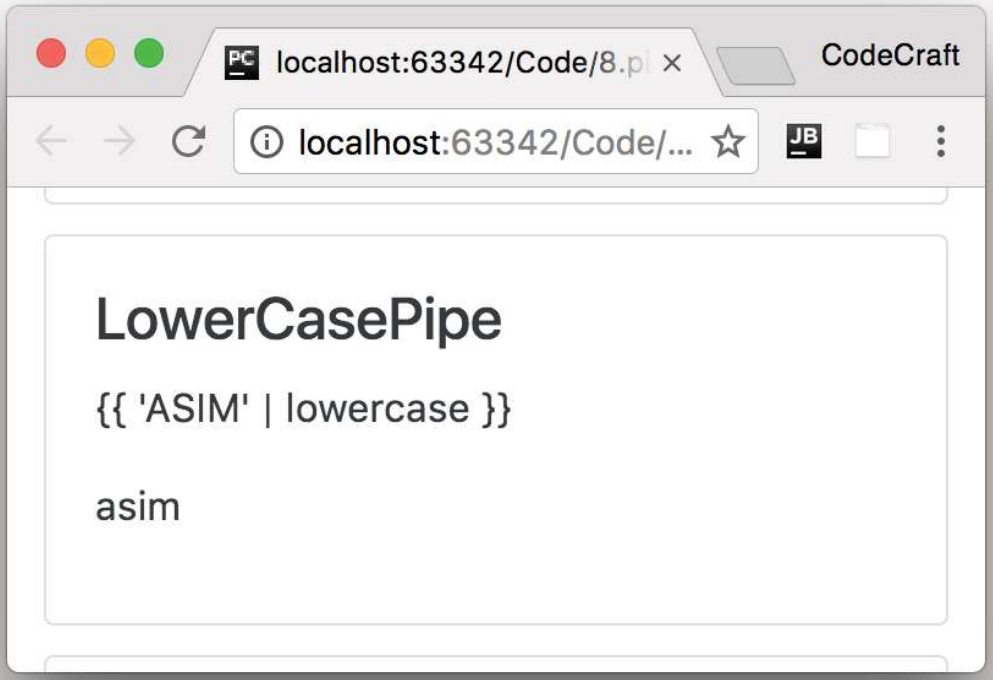



```
<div class="card card-block">
  <h4 class="card-title">JsonPipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ jsonVal }}</p> ①
    <p>{{ jsonVal }}</p>

    <p ngNonBindable>{{ jsonVal | json }}</p>
    <p>{{ jsonVal | json }}</p>
  </div>
</div>
```

① `jsonVal` is an object declared as `{ moo: 'foo', goo: { too: 'new' }}`.

LowerCasePipe

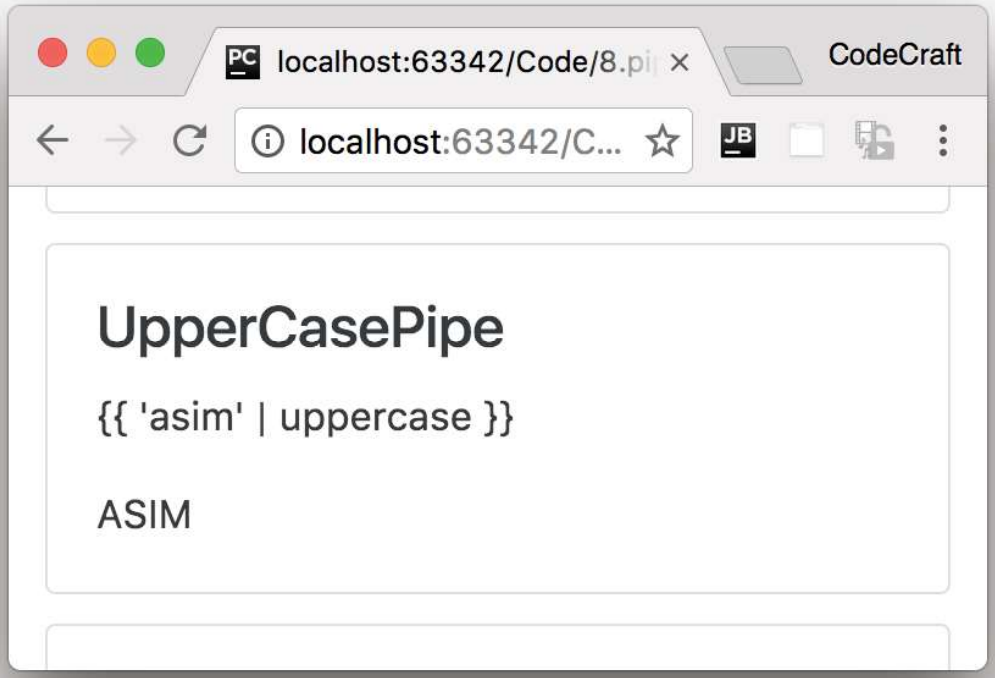
This transforms a string to lowercase, like so:




```
<div class="card card-block">
  <h4 class="card-title">LowerCasePipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ 'ASIM' | lowercase }}</p>
    <p>{{ 'ASIM' | lowercase }}</p>
  </div>
</div>
```

UpperCasePipe

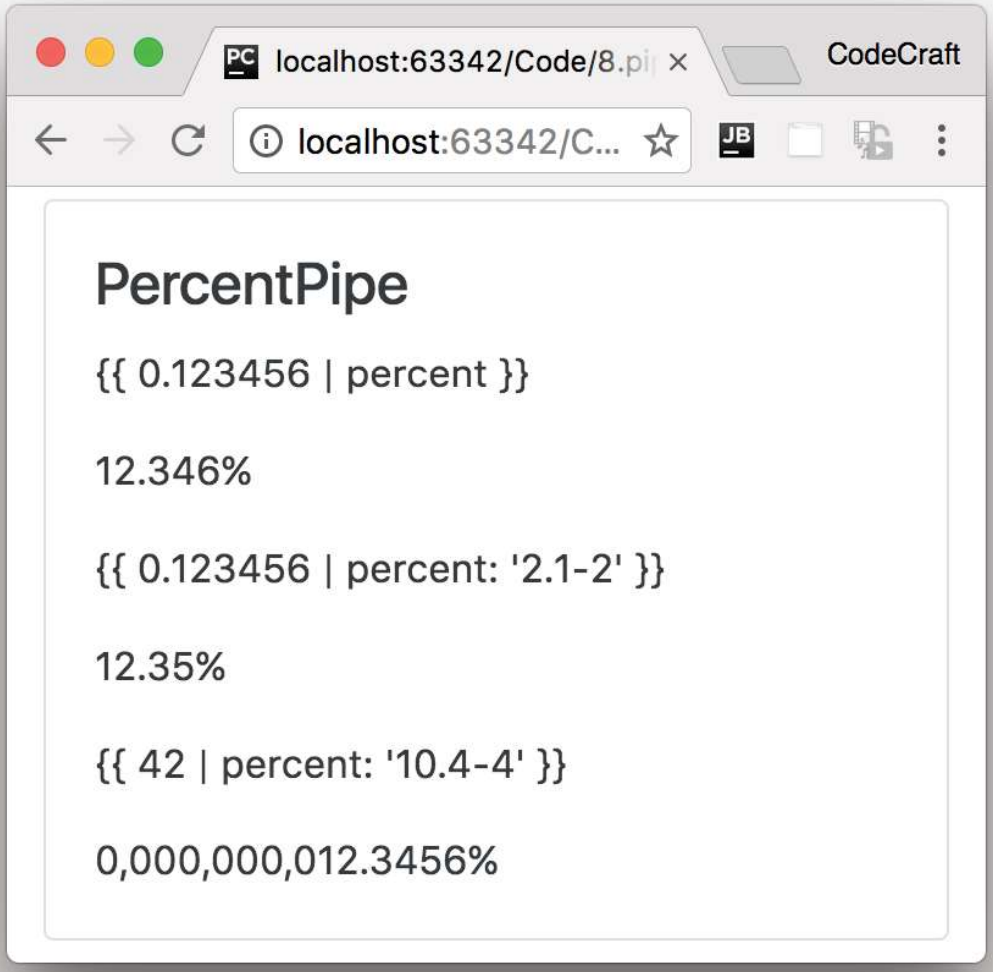
This transforms a string to uppercase, like so:




```
<div class="card card-block">
  <h4 class="card-title">UpperCasePipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ 'asim' | uppercase }}</p>
    <p>{{ 'asim' | uppercase }}</p>
  </div>
</div>
```

PercentPipe

Formats a number as a percent, like so:



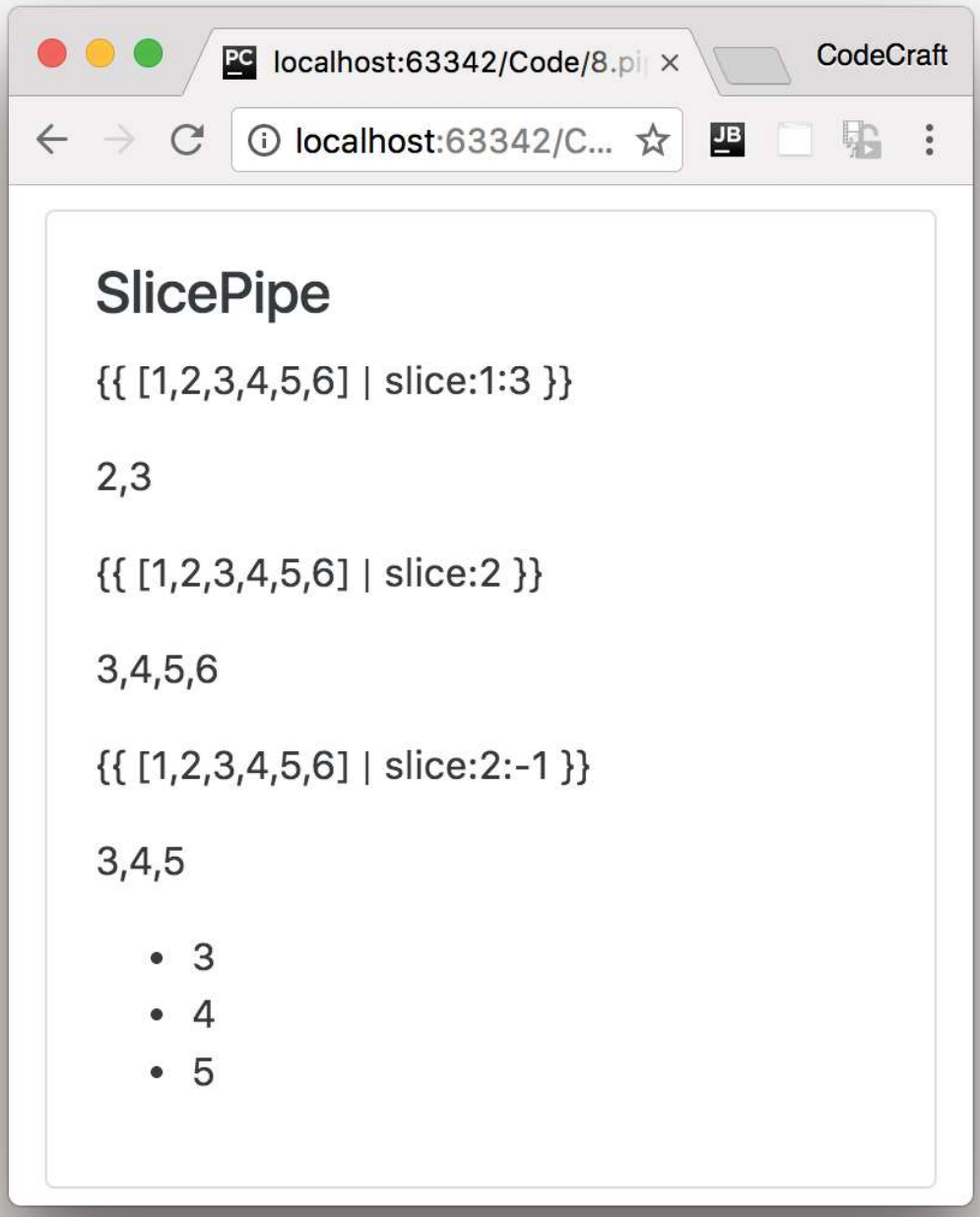

```
<div class="card card-block">
  <h4 class="card-title">PercentPipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ 0.123456 | percent }}</p>
    <p>{{ 0.123456 | percent }}</p>

    <p ngNonBindable>{{ 0.123456 | percent: '2.1-2' }}</p> ①
    <p>{{ 0.123456 | percent: '2.1-2' }}</p>

    <p ngNonBindable>{{ 42 | percent: '10.4-4' }}</p>
    <p>{{ 0.123456 | percent : "10.4-4" }}</p>
  </div>
</div>
```

① Percent can be passed a format string similar to the format passed to the `DecimalPipe`.

SlicePipe

This returns a *slice* of an array. The first argument is the start index of the slice and the second argument is the end index.

If either indexes are not provided it assumes the start or the end of the array and we can use negative indexes to indicate an offset from the end, like so:



```
<div class="card card-block">
  <h4 class="card-title">SlicePipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ [1,2,3,4,5,6] | slice:1:3 }}</p> ①
    <p>{{ [1,2,3,4,5,6] | slice:1:3 }}</p>

    <p ngNonBindable>{{ [1,2,3,4,5,6] | slice:2 }}</p> ②
    <p>{{ [1,2,3,4,5,6] | slice:2 }}</p>

    <p ngNonBindable>{{ [1,2,3,4,5,6] | slice:2:-1 }}</p> ③
    <p>{{ [1,2,3,4,5,6] | slice:2:-1 }}</p>

    <pre ngNonBindable>
&lt;ul&gt;
  &lt;li *ngFor=&quot;let v of [1,2,3,4,5,6] | slice:2:-1&quot;&gt;
    {{v}}
  &lt;/li&gt;
&lt;/ul&gt;
    </pre>

    <ul>
      <li *ngFor="let v of [1,2,3,4,5,6] | slice:2:-1"> ④
        {{v}}
      </li>
    </ul>
  </div>
</div>
```

① `slice:1:3` means return the items from the 1st to the 3rd index inclusive (indexes start at 0).

② `slice:2` means return the items from the 2nd index to the end of the array.

③ `slice:2:-1` means return the items from the 2nd index to one from the end of the array.

④ We can use slice inside for loops to only loop over a subset of the array items.

AsyncPipe

This pipe accepts an observable or a promise and lets us render the output of an observable or promise without having to call `then` or `subscribe`.

We are going to take a much deeper look at this pipe at the end of this section.

Summary

Pipes enables you to easily transform data for display purposes in templates.

Angular comes with a very useful set of pre-built pipes to handle most of the common transformations.

One of the more complex pipes to understand in Angular is the async pipe that's what we'll cover

next.

Listing

http://plnkr.co/edit/UG4SwlJ0DQEGjkhbbQz9?p=preview

script.ts

```
import {NgModule, Component} from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';
import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';

@Component({
  selector: 'pipe-builtins',
  template: `<div class="card card-block">
  <h4 class="card-title">Currency</h4>
  <div class="card-text">
    <p ngNonBindable>{{ 1234.56 | currency:'CAD' }}</p>
    <p>{{ 1234.56 | currency:"CAD" }}</p>

    <p ngNonBindable>{{ 1234.56 | currency:'CAD':'code' }}</p>
    <p>{{ 1234.56 | currency:'CAD':'code'}}</p>

    <p ngNonBindable>{{ 1234.56 | currency:'CAD':'symbol' }}</p>
    <p>{{ 1234.56 | currency:'CAD':'symbol'}}</p>

    <p ngNonBindable>{{ 1234.56 | currency:'CAD':'symbol-narrow' }}</p>
    <p>{{ 1234.56 | currency:'CAD':'symbol-narrow'}}</p>
  </div>
</div>

<div class="card card-block">
  <h4 class="card-title">Date</h4>
  <div class="card-text">
    <p ngNonBindable>{{ dateVal | date: 'shortTime' }}</p>
    <p>{{ dateVal | date: 'shortTime' }}</p>

    <p ngNonBindable>{{ dateVal | date:'fullDate' }}</p>
    <p>{{ dateVal | date: 'fullDate' }}</p>

    <p ngNonBindable>{{ dateVal | date: 'shortTime' }}</p>
    <p>{{ dateVal | date: 'shortTime' }}</p>

    <p ngNonBindable>{{ dateVal | date: 'd/M/y' }}</p>
    <p>{{ dateVal | date: 'd/M/y' }}</p>
  </div>
</div>

<div class="card card-block">
  <div class="card-text">
    <h4 class="card-title">DecimalPipe</h4>
```

```
    <p ngNonBindable>{{ 3.14159265 | number: '3.1-2' }}</p>
    <p>{{ 3.14159265 | number: '3.1-2' }}</p>

    <p ngNonBindable>{{ 3.14159265 | number: '1.4-4' }}</p>
    <p>{{ 3.14159265 | number: '1.4-4' }}</p>
  </div>
</div>

<div class="card card-block">
  <h4 class="card-title">JsonPipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ jsonVal }}</p>
    <p>{{ jsonVal }}</p>

    <p ngNonBindable>{{ jsonVal | json }}</p>
    <p>{{ jsonVal | json }}</p>
  </div>
</div>


<div class="card card-block">
  <h4 class="card-title">LowerCasePipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ 'ASIM' | lowercase }}</p>
    <p>{{ 'ASIM' | lowercase }}</p>
  </div>
</div>

<div class="card card-block">
  <h4 class="card-title">UpperCasePipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ 'asim' | uppercase }}</p>
    <p>{{ 'asim' | uppercase }}</p>
  </div>
</div>

<div class="card card-block">
  <h4 class="card-title">PercentPipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ 0.123456 | percent }}</p>
    <p>{{ 0.123456 | percent }}</p>

    <p ngNonBindable>{{ 0.123456 | percent: '2.1-2' }}</p>
    <p>{{ 0.123456 | percent: '2.1-2' }}</p>

    <p ngNonBindable>{{ 42 | percent: '10.4-4' }}</p>
    <p>{{ 0.123456 | percent : "10.4-4" }}</p>
  </div>
</div>

<div class="card card-block">
```

```
  <h4 class="card-title">SlicePipe</h4>
  <div class="card-text">
    <p ngNonBindable>{{ [1,2,3,4,5,6] | slice:1:3 }}</p>
    <p>{{ [1,2,3,4,5,6] | slice:1:3 }}</p>

    <p ngNonBindable>{{ [1,2,3,4,5,6] | slice:2 }}</p>
    <p>{{ [1,2,3,4,5,6] | slice:2 }}</p>

    <p ngNonBindable>{{ [1,2,3,4,5,6] | slice:2:-1 }}</p>
    <p>{{ [1,2,3,4,5,6] | slice:2:-1 }}</p>

    <pre ngNonBindable>
&lt;ul&gt;
  &lt;li *ngFor=&quot;let v of [1,2,3,4,5,6] | slice:2:-1&quot;&gt;
    {{v}}
  &lt;/li&gt;
&lt;/ul&gt;
    </pre>

    <ul>
      <li *ngFor="let v of [1,2,3,4,5,6] | slice:2:-1">
        {{v}}
      </li>
    </ul>
  </div>
</div>

 `
})
class PipeBuiltinsComponent {
  private dateVal: Date = new Date();
  private jsonVal: Object = {moo: 'foo', goo: {too: 'new'}};

}

@Component({
  selector: 'app',
  template: `
<pipe-builtins></pipe-builtins>
 `
})
class AppComponent {
}


@NgModule({
  imports: [BrowserModule],
  declarations: [AppComponent,
    PipeBuiltinsComponent
  ],
  bootstrap: [AppComponent],
```

```
})
class AppModule {

}

platformBrowserDynamic().bootstrapModule(AppModule);
```

Async Pipe

Learning Objectives

- When to use the *async* pipe.
- How to use async pipe with Promises and also Observables.

Overview

Normally to render the result of a promise or an observable we have to:

1. Wait for a *callback*.
2. Store the result of the callback is a *variable*.
3. *Bind* to that variable in the template.

With `AsyncPipe` we can use promises and observables directly in our template, without having to store the result on an intermediate property or variable.

`AsyncPipe` accepts as argument an observable or a promise, calls `subcribe` or attaches a `then` handler, then waits for the asynchronous result before passing it through to the caller.

AsyncPipe with promises

Lets first create a component with a promise as a property.

```
@Component({
  selector: 'async-pipe',
  template: `
 <div class="card card-block">
  <h4 class="card-title">AsyncPipe</h4>
  <p class="card-text" ngNonBindable>{{ promiseData }}</p> ①
  <p class="card-text">{{ promiseData }}</p> ②
 </div>
  `
})
class AsyncPipeComponent {
  promiseData: string;
  constructor() {
        this.getPromise().then(v => this.promiseData = v); ③
  }

  getPromise() {  ④
     return new Promise((resolve, reject) => {
       setTimeout(() => resolve("Promise complete!"), 3000);
     });
  }
}
```

① We use `ngNonBindable` so we can render out `{{ promiseData }}` as is without trying to bind to to the property `promiseData`

② We bind to the property `promiseData`

③ When the promise resolves we store the data onto the `promiseData` property

④ `getPromise` returns a promise which 3 seconds later resolves with the value `"Promise complete!"`

In the constructor we wait for the promise to resolve and store the result on a property called `promiseData` on our component and then bind to that property in the template.

To save time we can use the `async` pipe in the template and bind to the promise *directly*, like so:

[source,javascript]cript.ts

```
@Component({
  selector: 'async-pipe',
  template: `
 <div class="card card-block">
  <h4 class="card-title">AsyncPipe</h4>
  <p class="card-text" ngNonBindable>{{ promise }}</p>
  <p class="card-text">{{ promise | async }}</p> ①
 </div>
  `
})
class AsyncPipeComponent {
  promise: Promise<string>;
  constructor() {
        this.promise = this.getPromise(); ②
  }

  getPromise() {
     return new Promise((resolve, reject) => {
       setTimeout(() => resolve("Promise complete!"), 3000);
     });
  }
}
```

① We pipe the output of our `promise` to the `async` pipe.

② The property `promise` is the actual unresolved *promise* that gets returned from `getPromise` without `then` being called on it.

The above results in the same behaviour as before, we just saved ourselves from writing a `then` callback and storing intermediate data on the component.

AsyncPipe with observables

To demonstrate how this works with *observables* we first need to setup our component with a simple *observable*, like so:

```
import { Observable } from 'rxjs/Rx';
.
.
.
@Component({
  selector: 'async-pipe',
  template: `
 <div class="card card-block">
  <h4 class="card-title">AsyncPipe</h4>
  <p class="card-text" ngNonBindable>{{ observableData }}
  <p class="card-text">{{ observableData }}</p> ①
 </div>
`
})
class AsyncPipeComponent {
  observableData: number;
  subscription: Object = null;

  constructor() {
    this.subscribeObservable();
  }

  getObservable() { ②
    return Observable
        .interval(1000)
        .take(10)
        .map((v) => v * v);
  }

  subscribeObservable() { ③
    this.subscription = this.getObservable()
        .subscribe( v => this.observableData = v);
  }

  ngOnDestroy() { ④
    if (this.subscription) {
      this.subscription.unsubscribe();
    }
  }

}
```

① We render the value of `observableData` in our template.

② We create an observable which publishes out a number which increments by one every second then squares that number.

③ We subscribe to the output of this observable chain and store the number on the property `observableData`. We also store a reference to the subscription so we can unsubscribe to it later.

④ On destruction of the component we unsubscribe from the observable to avoid memory leaks.

We should also be destroying the subscription when the component is destroyed. Otherwise we will start leaking data as the old observable, which isn't used any more, will still be producing results.

Again by using `AsyncPipe` we don't need to perform the `subscribe` and store any intermediate data on our component, like so:

```
@Component({
  selector: 'async-pipe',
  template: `
 <div class="card card-block">
  <h4 class="card-title">AsyncPipe</h4>
  <p class="card-text" ngNonBindable>{{ observable | async }}
  <p class="card-text">{{ observable | async }}</p> ①
 </div>
`
})
class AsyncPipeComponent {
  observable: Observable<number>;

  constructor() {
    this.observable = this.getObservable();
  }

  getObservable() {
    return Observable
      .interval(1000)
      .take(10)
      .map((v) => v*v)
  }
}
```

① We pipe our `observable` directly to the `async` pipe, it performs a subscription for us and then returns whatever gets passed to it.

By using `AsyncPipe` we: 1. Don't need to call `subscribe` on our observable and store the intermediate data on our component. 2. Don't need to remember to `unsubscribe` from the observable when the component is destroyed.

Summary

`AsyncPipe` is a convenience function which makes rendering data from observables and promises much easier.

For promises it automatically adds a `then` callback and renders the response.

For Observables it automatically `subscribes` to the observable, renders the output and then also `unsubscribes` when the component is destroyed so we don't need to handle the clean up logic ourselves.

That's it for the built-in pipes, next up we will look at creating out own custom pipes.

Listing

http://plnkr.co/edit/gHIalfn10CfocBwCE6UG?p=preview

script.ts

```
import {NgModule, Component, OnDestroy} from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';
import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';
import { Observable } from 'rxjs/Rx';


@Component({
  selector: 'async-pipe',
  template: `
 <div class="card card-block">
  <h4 class="card-title">AsyncPipe</h4>

  <p class="card-text" ngNonBindable>{{ promise | async }}  </p>
  <p class="card-text">{{ promise | async }}  </p>

  <p class="card-text" ngNonBindable>{{ observable | async }}  </p>
  <p class="card-text">{{ observable | async }}</p>

  <p class="card-text" ngNonBindable>{{ observableData }}  </p>
  <p class="card-text">{{ observableData }}</p>
 </div>
  `
})
class AsyncPipeComponent implements OnDestroy {
  promise: Promise<string>;
  observable: Observable<number>;
  subscription: Object = null;
  observableData: number;

  constructor() {
    this.promise = this.getPromise();
    this.observable = this.getObservable();
    this.subscribeObservable();

  }

  getObservable() {
    return Observable
        .interval(1000)
        .take(10)
        .map((v) => v * v);
  }
```

```
  // AsyncPipe subscribes to the observable automatically
  subscribeObservable() {
    this.subscription = this.getObservable()
        .subscribe((v) => this.observableData = v);
  }

  getPromise() {
    return new Promise((resolve, reject) => {
      setTimeout(() => resolve("Promise complete!"), 3000);
    });
  }

  // AsyncPipe unsubscribes from the observable automatically
  ngOnDestroy() {
    if (this.subscription) {
      this.subscription.unsubscribe();
    }
  }
}


@Component({
  selector: 'app',
  template: `
  <async-pipe></async-pipe>
 `
})
class AppComponent {
  imageUrl: string = "";

}


@NgModule({
  imports: [BrowserModule],
  declarations: [AppComponent,
    AsyncPipeComponent
  ],
  bootstrap: [AppComponent],
})
class AppModule {

}

platformBrowserDynamic().bootstrapModule(AppModule);
```