
ACCELERATING THROUGH
ANGULAR 2

Level 2

Structural Directives

Section 1

Learning Angular Directives

A directive (within Angular) is how we add dynamic behavior to HTML.

There are three different kinds of directives:

Component

Has a template.

Structural

We will define these later.

Attribute

We won't get to these.

Looking Back at Our Code

main.ts TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
   <h2>{{carPart.name}}<h2>
   <p>{{carPart.description}}<p>
   <p>{{carPart.inStock}} in Stock<p>`
})
class AppComponent {
  title = 'Ultra Racing';
  carPart = {
    "id": 1,
    "name": "Super Tires",
    "description": "These tires are the very best",
    "inStock": 5
  };
}


bootstrap(AppComponent);
```

What if we had more than one car part?

Adding an Array of Car Parts

main.ts — TypeScript

```
...
})
class AppComponent {
  title = 'Ultra Racing';
  carParts = [{
    "id": 1,
    "name": "Super Tires",
    "description": "These tires are the very best",
    "inStock": 5
  },
  {
    "id": 2,
    "name": "Reinforced Shocks",
    "description": "Shocks made from kryptonite",
    "inStock": 4
  }, { ... }];
}

bootstrap(AppComponent);
```

Now that we have many car parts, how do we loop through each of these?

Adding an Array of Car Parts

main.ts TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <ul>
      <li *ngFor="let carPart of carParts">
        <h2>{{carPart.name}}<h2>
        <p>{{carPart.description}}<p>
        <p>{{carPart.inStock}} in Stock<p>
      </li>
    </ul>`
})
class AppComponent {
  title = 'Ultra Racing';
  carParts = [{
    "id": 1,
    "name": "Super Tires",
    "description": "These tires are the very best",
    "inStock": 5
  }, { ... }, { ... }];
```

`*ngFor` is a structural directive.

`carPart` is a local variable.

`carParts` is the array to loop through.

The loop is run three times: once for each carPart.

Ultra Racing

- **Super Tires**

  These tires are the very best

  5 in Stock

- **Reinforced Shocks**

  Shocks made from kryptonite

  4 in Stock

- **Padded Seats**

  Super soft seats for a smooth ride

  0 in Stock

Learning Angular Directives

A directive (within Angular) is how we add dynamic behavior to HTML.

Component

Has a template.

Structural

Alters layout by adding, removing, or replacing HTML elements.

*ngFor

When Car Parts Are Out of Stock

When there are none in stock, how can we display “Out of Stock”?

main.ts — TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <ul>
      <li *ngFor="let carPart of carParts">
        <h2>{{carPart.name}}<h2>
        <p>{{carPart.description}}<p>
        <p>{{carPart.inStock}} in Stock<p>
      </li>
    </ul>`
})
class AppComponent {
  ...
```

Should read “Out of Stock”

Using ngIf With a Conditional

main.ts TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <ul>
      <li *ngFor="let carPart of carParts">
        <h2>{{carPart.name}}<h2>
        <p>{{carPart.description}}<p>
        <p *ngIf="carPart.inStock > 0">{{carPart.inStock}} in Stock</p>
        <p *ngIf="carPart.inStock === 0">Out of Stock</p>
      </li>
    </ul>`
})
class AppComponent {
  ...
```

If true, display this.

`*ngIf` is another structural directive. It allows us to evaluate conditionals.

What'd We Learn?

- A directive (within Angular) is how we add dynamic behavior to HTML.
- A component directive has a template.
- A structural directive alters layout by adding, removing, or replacing HTML elements.

`*ngFor`	Loops through an array.
`*ngIf`	Shows content conditionally.


ACCELERATING THROUGH
ANGULAR 2

Level 2

Pipes & Methods

Section 2

Using Pipes to Format Screen Data

A pipe takes in data as input and transforms it to a desired output.

How can we write out car part names in capital letters?

main.ts TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <ul>
      <li *ngFor="let carPart of carParts">
        <h2>{{carPart.name | uppercase }}<h2>
        <p>{{carPart.description}}<p>
        <p *ngIf="carPart.inStock > 0">{{carPart.inStock}} in Stock</p>
        <p *ngIf="carPart.inStock === 0">Out of Stock</p>
      </li>
    </ul>`
})
class AppComponent {
  ...
```

Ultra Racing

- **SUPER TIRES**

  These tires are the very best

  5 in Stock

- **REINFORCED SHOCKS**

  Shocks made from kryptonite

  4 in Stock

- **PADDED SEATS**

  Super soft seats for a smooth ride

  Out of Stock

Similar to Linux pipe, if you're familiar with it.

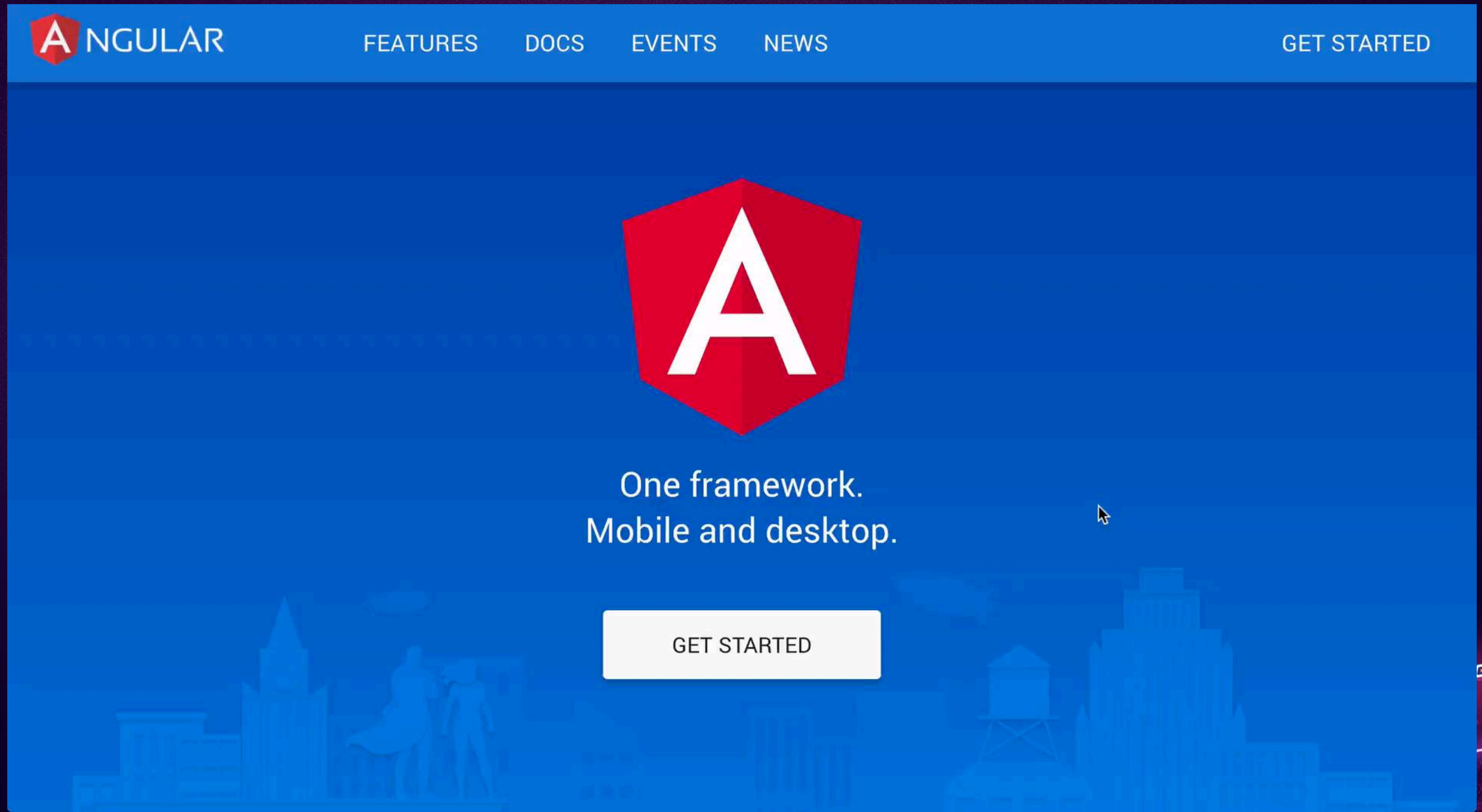
Adding a Price to Our Data

main.ts TypeScript

```
...
})
class AppComponent {
  title = 'Ultra Racing';
  carParts = [{
    "id": 1,
    "name": "Super Tires",
    "description": "These tires are the very best",
    "inStock": 5,
    "price": 4.99
  },
  {
    "id": 2,
    "name": "Reinforced Shocks",
    "description": "Shocks made from kryptonite",
    "inStock": 4,
    "price": 9.99
  }, ...
```

But how do we format this properly?

Using Documentation to Look Up Pipes



Using the Currency Pipe With One Parameter

To format our currency, we use the ISO 4217 currency code.

Ex.: USD, EUR, or CAD

main.ts — TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <ul>
      <li *ngFor="let carPart of carParts" >
        <h2>{{carPart.name | uppercase }}</h2>
        <p>{{carPart.description}}</p>
        <p>{{carPart.price | currency:'EUR'}}</p>
        <p *ngIf="carPart.inStock > 0">{{carPart.inStock}} in Stock</p>
        <p *ngIf="carPart.inStock === 0">Out of Stock</p>
      <li>
    <ul>`
})
class AppComponent {
  ...
```

Ultra Racing

- **SUPER TIRES**

  These tires are the very best

  EUR4.99

  5 in Stock

- **REINFORCED SHOCKS**

  Shocks made from kryptonite

  EUR9.99

  4 in Stock

- **PADDED SEATS**

  Super soft seats for a smooth ride

  EUR24.99

  Out of Stock

But we want the EUR symbol — how do we do that?

Using the Currency Pipe With Two Parameters

The second parameter is a boolean indicating if we should use the currency symbol.

main.ts — TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <ul>
      <li *ngFor="let carPart of carParts" >
        <h2>{{carPart.name | uppercase }}</h2>
        <p>{{carPart.description}}</p>
        <p>{{carPart.price | currency:'EUR':true }}</p>
        <p *ngIf="carPart.inStock > 0">{{carPart.inStock}} in Stock</p>
        <p *ngIf="carPart.inStock === 0">Out of Stock</p>
      <li>
    <ul>`
})
class AppComponent {
  ...
```

Notice the colon between parameters.

Ultra Racing

- **SUPER TIRES**

  These tires are the very best

  €4.99

  5 in Stock

- **REINFORCED SHOCKS**

  Shocks made from kryptonite

  €9.99

  4 in Stock

- **PADDED SEATS**

  Super soft seats for a smooth ride

  €24.99

  Out of Stock

Additional Pipes to Use

`lowercase`	Well, lowercase...
`date`	Formats dates how you like them.
`number`	Formats numbers.
`decimal`	Formats decimals.
`replace`	Creates a new string, replacing specified characters.
`slice`	Creates a new list or string containing a subset of the elements.
`json`	Transforms any input to a JSON-formatted string.

You can also create custom pipes!

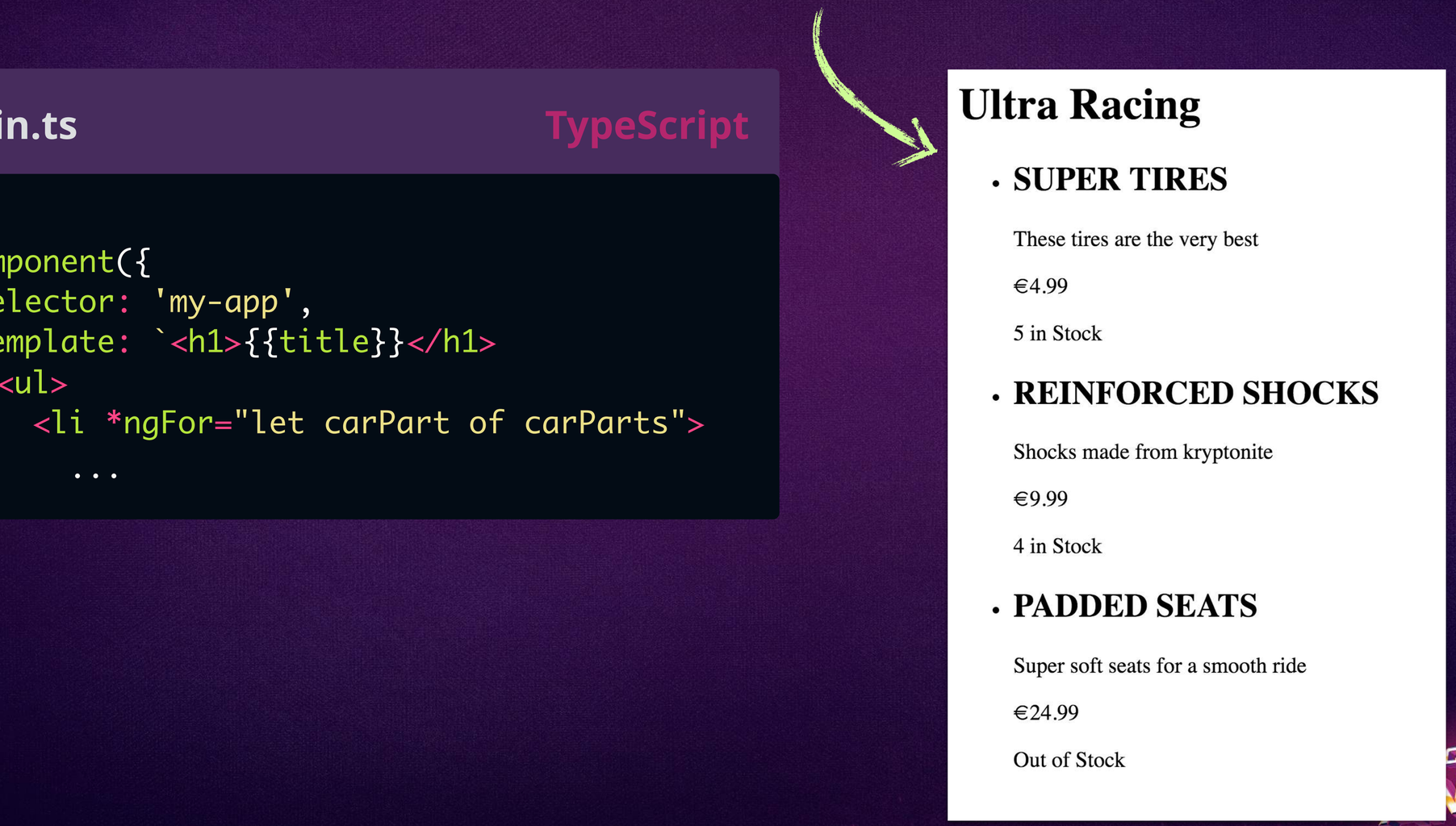
Great for debugging

Listing the Number of Car Parts in Stock

How could we display the total number of car parts in stock?

main.ts — TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <ul>
      <li *ngFor="let carPart of carParts">
        ...
```



Modifying the Template

We'll add new code to our HTML template and print the result of a method we're about to define.

main.ts — TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <p>There are {{totalCarParts()}} total parts in stock.</p>
    <ul>
      <li *ngFor="let carPart of carParts">
        ...
```

We define this method inside of our component class.

Modifying the Template

Let’s do the simplest thing and implement a class method that returns 10.

main.ts — TypeScript

```
...
@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <p>There are {{totalCarParts()}} total parts in stock.</p>
    ...
  `
})
class AppComponent {
  title = 'Ultra Racing';
  carParts = [...];

  totalCarParts() {
    return 10;
  }
}
```

Inside a TypeScript class, we don’t use the word “function,” just like we don’t use “let” to declare the properties.

ES2015 functionality enabled by TypeScript!

Implementing the Sum of Car Parts

Let's use an ES2015 `for of` loop, like in our template.

main.ts — TypeScript

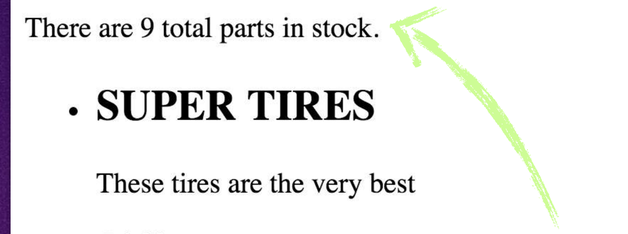
```
class AppComponent {
  title = 'Ultra Racing';
  carParts = [...];

  totalCarParts() {
    let sum = 0;

    for (let carPart of this.carParts) {
       sum += carPart.inStock;
    }

    return sum;
  }
}
```

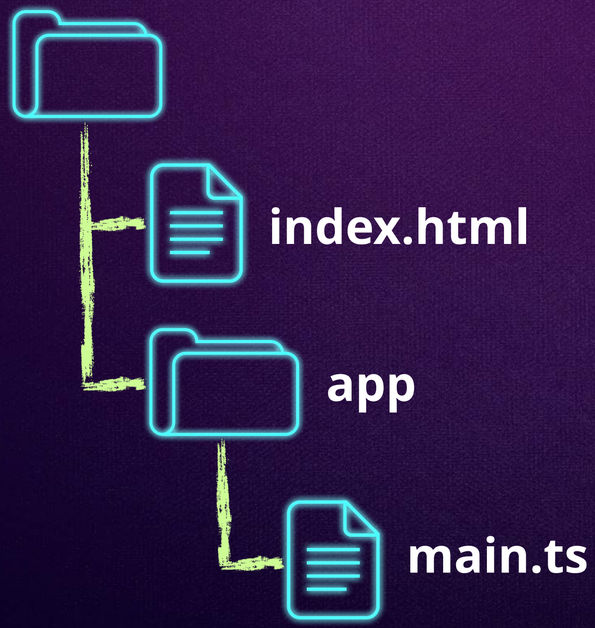
ES2015



What'd We Learn?

- We can use pipes to transform our template output.
- How to create and use methods in our components.

Our Current Application Structure




ACCELERATING THROUGH
ANGULAR 2

Bonus: Simplifying a Sum

```
totalCarParts() {
  let sum = 0;

  for (let carPart of this.carParts) {
     sum += carPart.inStock;
  }

  return sum;
}
```

Just for fun, let's go through a few ways we could simplify this code.

```
totalCarParts() {
  return this.carParts.reduce(function(prev, current) { return prev + current.inStock; }, 0 );
}
```

```
totalCarParts() {
  return this.carParts.reduce((prev, current) => prev + current.inStock, 0 );
}
```

The fat arrow (ES2015)