
ACCELERATING THROUGH
ANGULAR 2

Level 1

Our First Component

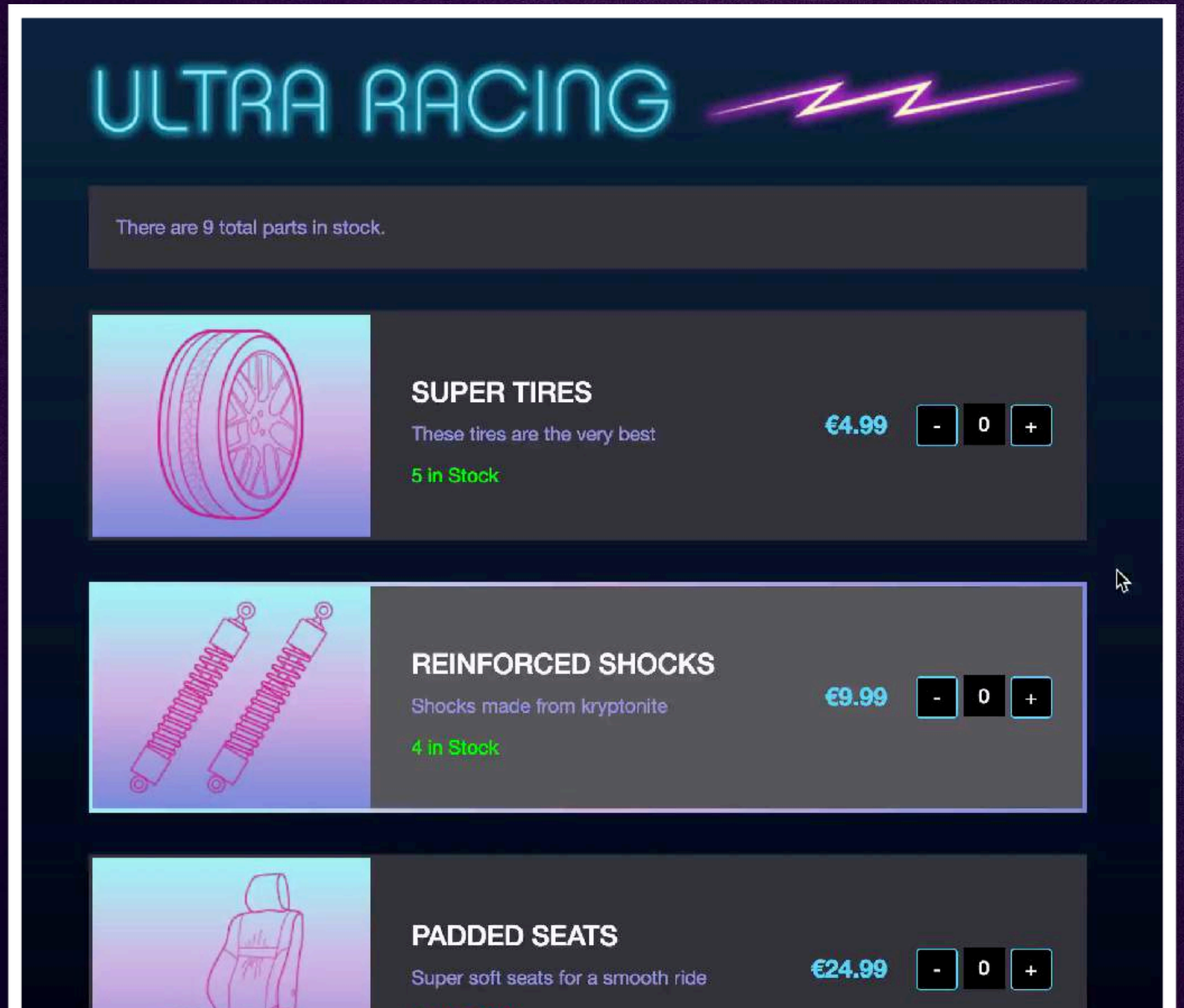
Section 1

What Is Angular?

- Angular is a framework for dynamic web applications.
- Provides a way to organize your HTML, JavaScript, and CSS to keep your front-end code clean.
- Released in 2011.
- Mainly maintained by Google with the help of the open-source community.

Back-end Server

What Will We Be Building in This Course?

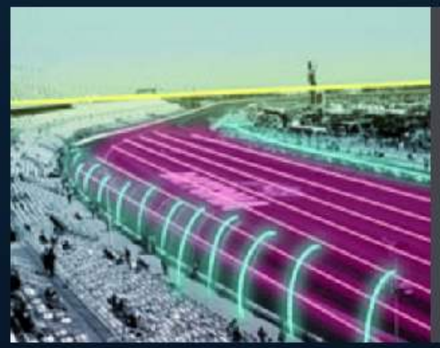


In the Challenges

Ultra Racing Schedule

Money for racing: 10000

Cash left to enter races: $1,000



Daytona Thunderdome
Jan 4, 2512, 9:00 AM

Race through the ruins of an ancient Florida battle arena.

$3,200

Enter Race

San Francisco Ruins
Jul 3, 2512, 4:00 PM

Drift down the streets of a city almost sunk under the ocean.

$4,700

Racing

Cancel Race

New York City

What Will This Course Cover?

Level 1

Our First Component

Level 2

Structural Directives, Pipes & Methods

Level 3

Code Organization & Data Models

Level 4

Data Binding

Level 5

Services & HTTP

With lots of challenges between

What Do You Need to Know to Take This Course?

You don't need any prior experience with Angular 1

Basic JavaScript

JavaScript Road Trip Parts 1, 2 & 3

Basic HTML & CSS

Front-end Foundations & Front-end Formations

(optional)

JavaScript: ES2015

ES2015: The Shape of JavaScript to Come

What Is the Difference Between Angular 1 & 2?

Speed — Angular 2 is faster.

Components — Instead of controllers and scope, we use components, which feel simpler.

Simpler Directives — Creating custom directives is much simpler.

Intuitive Data Binding — When we need to link data to an HTML element or listen for a button clicking on the page, we have an intuitive syntax.

Services are now just a class.

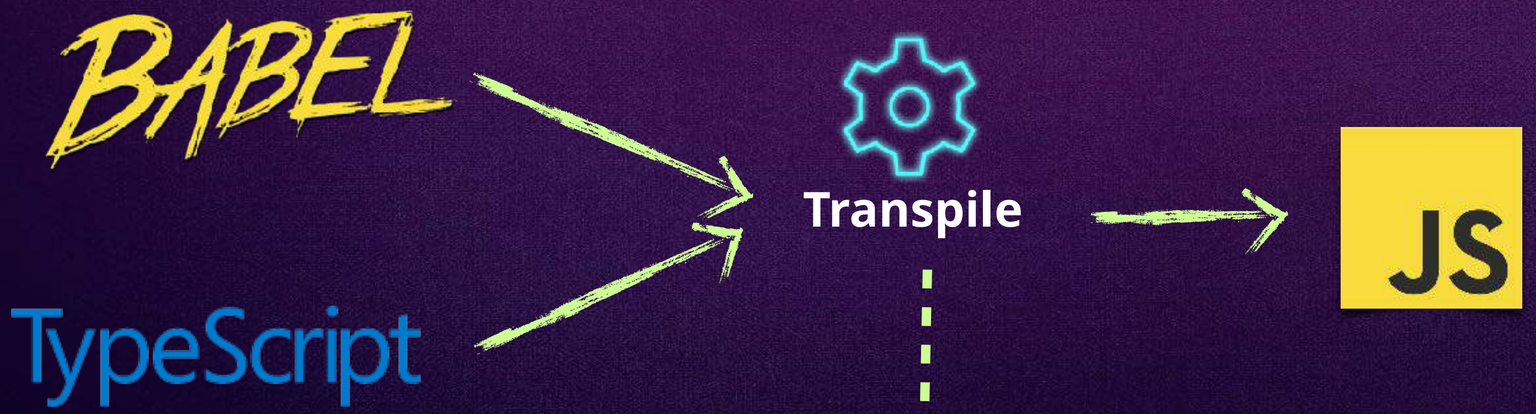
Many more small improvements.

What Language to Use With Angular 2?

JavaScript

But all browsers don't support the newest version of JavaScript.

There are ways to access these features:



Means it gets changed into JavaScript

TypeScript: Our Language of Choice

TypeScript is Microsoft's extension of JavaScript that allows the use of all ES2015 features and adds powerful type checking and object-oriented features.

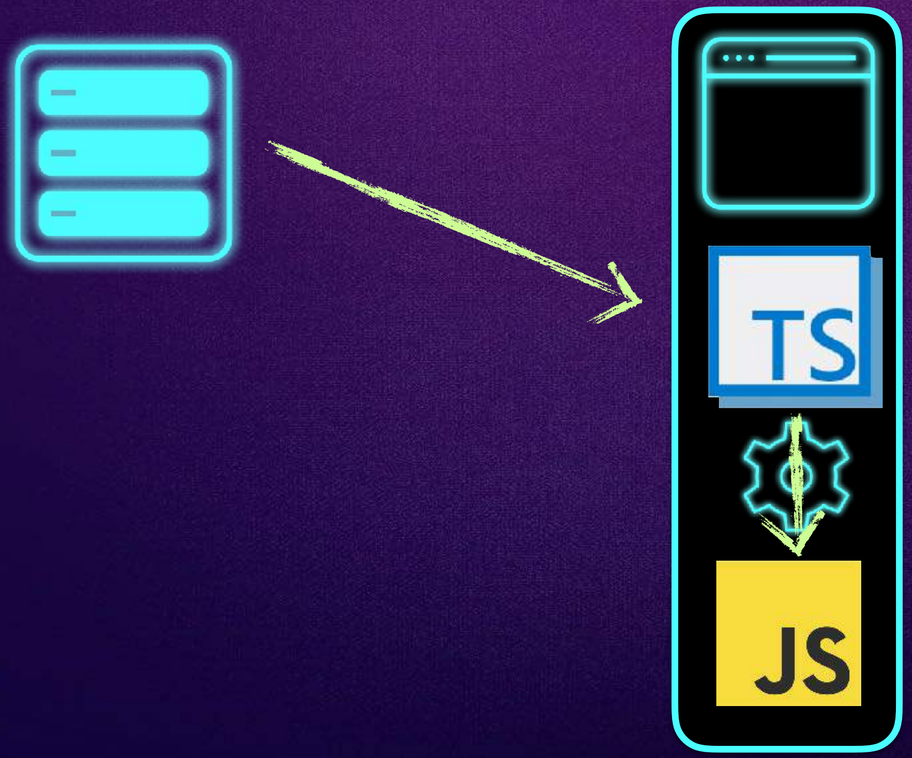
The Angular 2 source is programmed with TypeScript.

http://www.typescriptlang.org

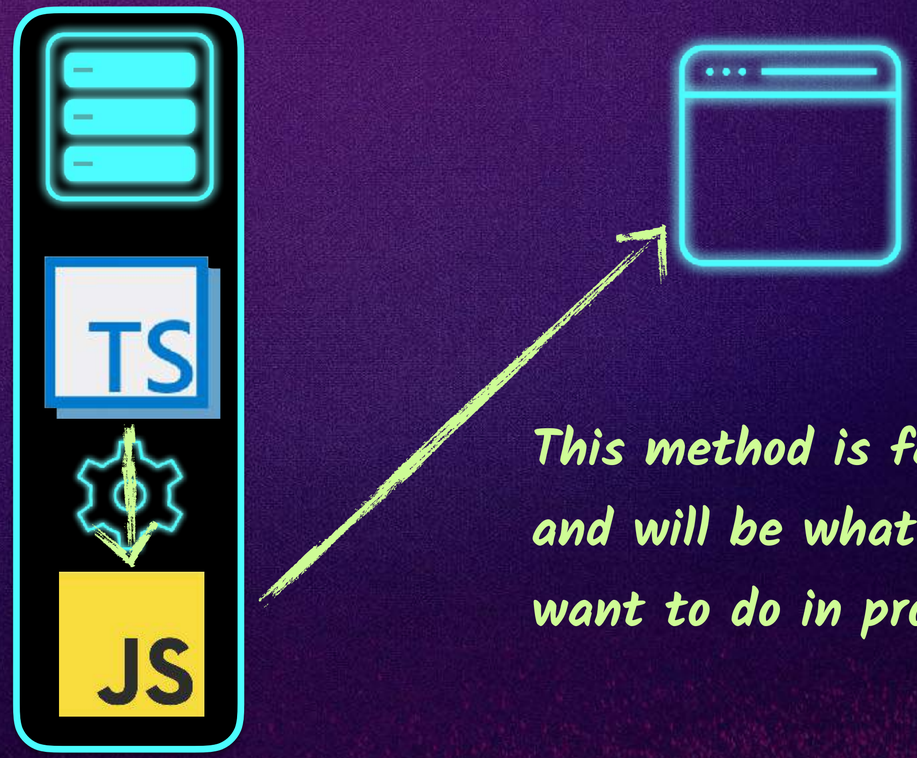
Transpiling Locations

Our browsers don’t know how to read TypeScript out of the box, so we have two options when it comes to changing our TypeScript code into JavaScript.

Transpile to JavaScript in the browser



Transpile to JavaScript before shipping to browser



This method is faster and will be what you want to do in production.

Building Our Index

index.html HTML

```
<!DOCTYPE html>
<html>

  <head>
    <!-- All the Angular 2 libraries needed -->
  </head>

</html>
```

We won't be covering all the libraries you need to load up Angular 2.

When you're ready to start developing, we suggest you go through the **5-minute QuickStart Guide**.

http://go.codeschool.com/angular2start

Creating Our First Custom Element

index.html HTML

```
<!DOCTYPE html>
<html>

  <head>
    <!-- All the Angular 2 libraries needed -->
  </head>

  <body>
    <my-app>Loading App ...</my-app>
  </body>

</html>
```

This is where our Angular 2 application will load.

This could be named anything, even `<racing-app>`

Until our app gets loaded in the browser, we see:

Loading a JavaScript File Using SystemJS

index.html HTML

```
<!DOCTYPE html>
<html>

  <head>
    <!-- All the Angular 2 libraries needed -->
    <script>
      ...
      System.import('app')
       .catch(function(err){ console.error(err); });
    </script>

  </head>
  <body>
    <my-app>Loading App ...</my-app>
  </body>

</html>
```

SystemJS is a JavaScript library that allows us to import other libraries.

app/main.ts

This loads our application's code.

Error messages should be printed out to the browser console.

Writing Our First TypeScript File

app/main.ts — TypeScript

```ts
import { Component } from '@angular/core';
```

Angular 2 library

`import` — ES2015 feature used to import functions, objects, or primitives.

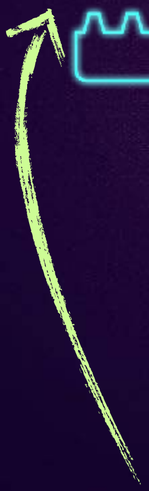
`Component` — Function we will use to create our first Component.

Components are the basic building blocks of Angular 2 applications. A component controls a portion of the screen.

Component Is a Decorator Function

app/main.ts — TypeScript

```
import { Component } from '@angular/core';

Our component decorator code goes here.

class AppComponent { }
```



A decorator  **adds more behavior to our class from outside the class.**

It must be declared immediately before the class.

The decorator turns our plain old JavaScript class into a component.

Decorating Our First Component

app/main.ts TypeScript

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: '<h1>Ultra Racing</h1>'
})
class AppComponent { }
```

index.html

```html
<body>
  <my-app>Loading App ...</my-app>
</body>
```

Often called metadata

`@Component` Used to apply our component decorator to our class.

Decorators are a TypeScript feature.

`selector` The CSS selector for the HTML element where we want the component to load.

`template` The content we want to load inside our selector.

Declaring the Root Angular Module

Modules are how we organize our application in Angular. Every Angular application must have a “root module,” which we’ll need to launch it.

app/main.ts — TypeScript

```ts
import { NgModule, Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: '<h1>Ultra Racing</h1>'
})
class AppComponent { }

@NgModule({
  declarations: [ AppComponent ]
})
class AppModule { }
```

List of all components within the module

Dependencies to Render the Application

app/main.ts TypeScript

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

`BrowserModule` Module needed for running Angular websites.

`platformBrowserDynamic` Angular library that will render the website.

This will allow us to bootstrap, or launch, the app.

Bootstrapping Our Component

Using our new dependencies.

app/main.ts — TypeScript

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

...

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

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

Loads required dependencies to launch our app in the browser

Indicates our root component

Render application using AppModule

Previewing our Rendered Application

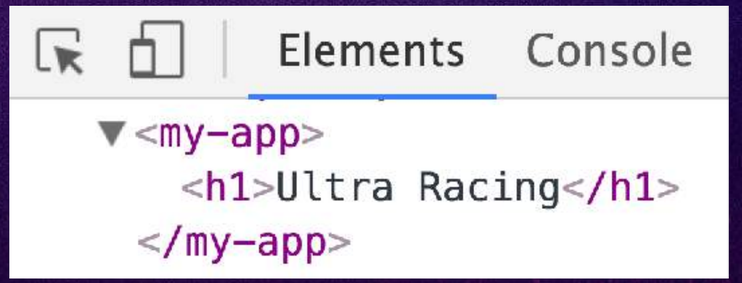
app/main.ts TypeScript

```
...
@Component({
  selector: 'my-app',
  template: '<h1>Ultra Racing</h1>'
})
class AppComponent { }


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

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

index.html

```
<body>
  <my-app>Loading App ...</my-app>
</body>
```

Viewing the Source



Our App Is Full of Components

Components are the building blocks of Angular 2 applications.

And they easily nest one inside the other.

Each component may have its own:

- *class file*
- *html file*
- *css file*

Sending Data Around

How do we send a property from our component class into our HTML?

app/main.ts — TypeScript

```
...

@Component({
  selector: 'my-app',
  template: '<h1>???</h1>'
})
class AppComponent {
  title = 'Ultra Racing';
}

...
```

Inside a TypeScript class, we don't use the `var` or `let` keywords to declare class properties.

Though we do in regular methods.

Using Interpolation to Print Properties

Curly braces allow us to load in component properties — this is called interpolation.

app/main.ts — TypeScript

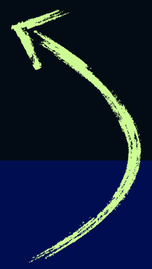
```
...

@Component({
  selector: 'my-app',
  template: '<h1>{{title}}</h1>'
})
class AppComponent {
  title = 'Ultra Racing';
}

...
```

Loading an Object

What if we have an object we want to print out onto the screen?

app/main.ts — TypeScript

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

...
```



Template with Back Ticks

app/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
  };
}
```

Our template now uses back ticks instead of single quotes.

Single Quote

Back Tick

Using the back ticks allows us to have template strings, which allows us to be multiline.

This is another ES2015 feature.

What'd We Learn?

- Angular is a framework for dynamic web applications.
- We are coding Angular using TypeScript, a language that transpiles into JavaScript.
- NgModules group Angular code into blocks of functionality.
- Components are the basic building blocks of any Angular application.
- We use a custom HTML tag (aka, selector) to show where we want our component to load inside our HTML.
- Decorators are what turn our plain TypeScript classes into Components.


ACCELERATING THROUGH
ANGULAR 2