



# FRAMEWORK TO PLATFORM

NOT AN UPGRADE OF ANGULAR 1.x  
COMPLETE RE-WRITE



**Syed Awase Khirni**

RESEARCHER | ENTREPRENEUR | TECHNOLOGY COACH

@sak008 | [sak@sycliq.com](mailto:sak@sycliq.com)/[sak@territorialprescience.com](mailto:sak@territorialprescience.com) | +91. 9035433124

Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project ([www.geo-spirit.org](http://www.geo-spirit.org)). He currently provides consulting services through his startup [www.territorialprescience.com](http://www.territorialprescience.com) and [www.sycliq.com](http://www.sycliq.com)





# Terms of Use

- You shall not circulate these slides without written permission from Territorial Prescience Research I Pvt Ltd.
- If you use any material, graphics or code or notes from these slides, you shall acknowledge the author Dr. Syed Awase Khirni
- If you have not received this material, post-training session, you shall destroy it immediately and not use it for unauthorized usage of the material.
- Angular is copyright of Google

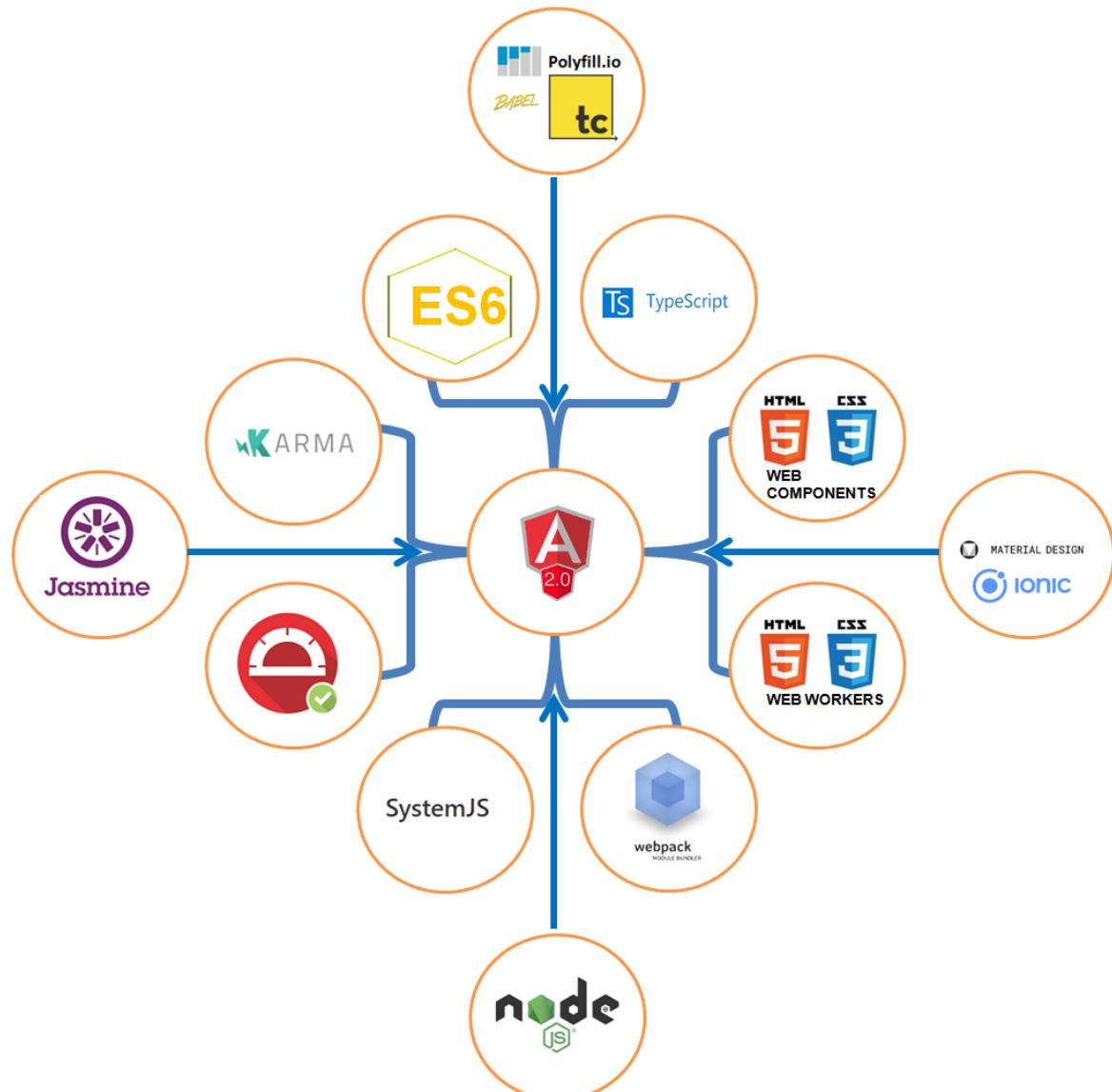




# Slide Version Updates

Last Updated	Angular 2 Version	Release Date	Updated by	Code Plays Done @
21 June 2016	2.0.0-rc3	21 June 2016	Syed Awase Khirni	
27 Aug 2016	2.0.0-rc5	9 Aug 2016	Syed Awase Khirni	JCI Pune
20 Nov 2016	2.1.0	12 Oct 2016	Syed Awase Khirni	ITC Infotech,Bang
21 Dec 2016	2.3.0	8 Dec 2016	Syed Awase Khirni	
27 Jan 2017	2.4.0	20 Dec 2016	Syed Awase Khirni	







# Prerequisites

- HTML5
- CSS3
- JavaScript
  - IIFE
  - Object Construction Pattern
  - Factory Pattern
  - IIFE
  - Module Pattern
  - Revealing Module Pattern
  - Observer Pattern



# Program Agenda

DAY 1

DAY 2

DAY 3

DAY 4





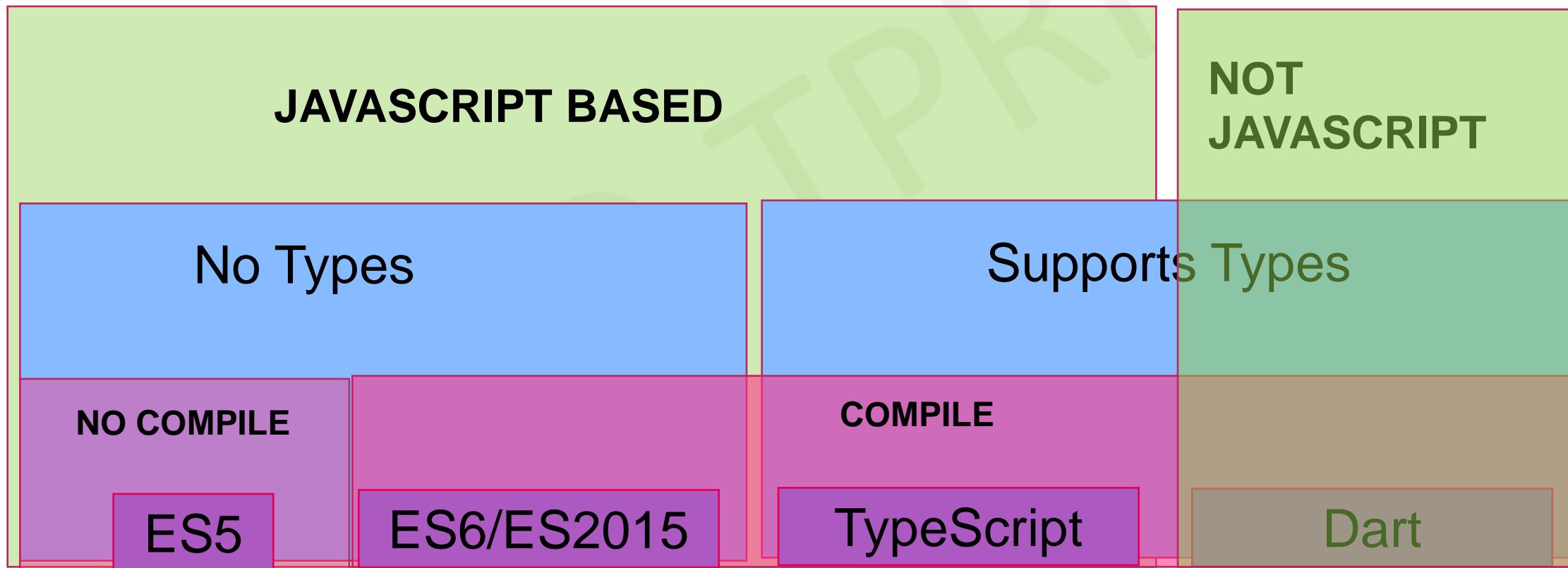
# The JavaScript problem

- No modules, only files
- No linkage
- No code organization features
- No static checking
- Hard to refactor
- So Google created AtScript
  - Google forked TypeScript
  - Called it AtScript for @signed used annotations
  - Google and Microsoft engineers join hands
  - TypeScript evolved.





# Coding Angular 2 (Language Choice)





# JavaScript Language Specification

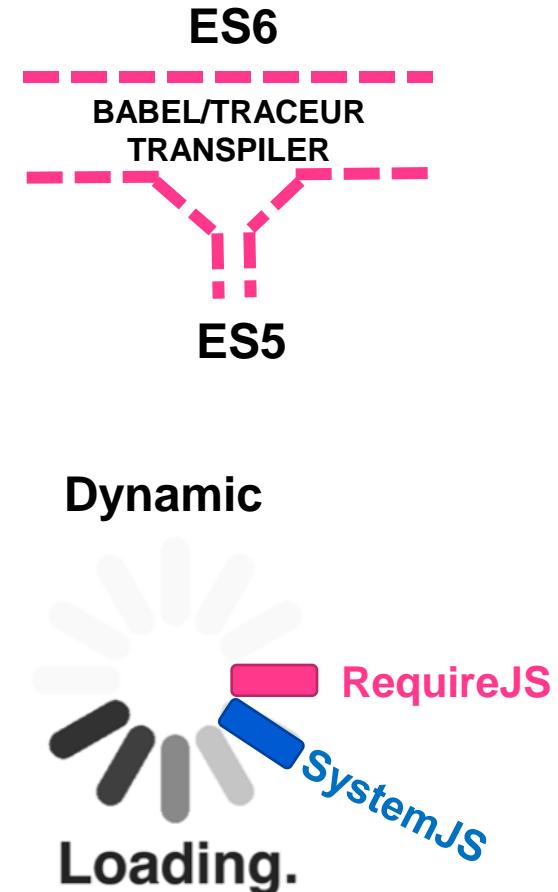
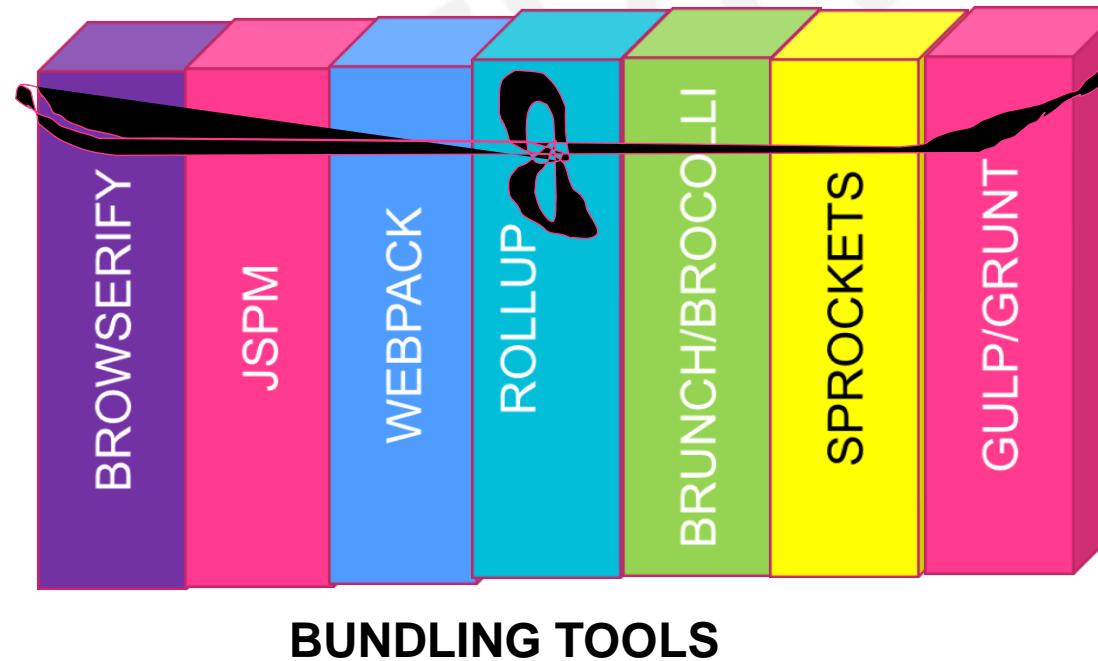
- ECMAScript (ES)
- ES3 – older Browsers
- ES5
- ES 2015 ( formerly known as ES6)
  - Must be **transpiled**
- Building Angular 2 Application using a language choice
  - ES 5 : runs in the browser, no compile required.
  - ES2015 – lots of new features (classes, let, arrow, etc..)
  - TypeScript : a superset of javascript, strong typing and great IDE tooling
  - Dart – A non-javascript based programming for Angular 2 applications.





# Complicated Landscape of JS/ES

ES6 brings a **standard module construct** to JavaScript and a loader specification.





# Difference between A1.x and A2.0

Section I





# Framework to Platform: A2

	Angular-CLI	Language Services	Augury
i18n	Material Design	Mobile First Design	Angular Universal
Animation	Faster Compiler	Change Detector	Rendering model
Router	Support Lazy loading	Extension to JS Language to add metadata to Code	No longer required to call \$scope.apply
ngUpgrade	<b>Dependency Injection</b>	Decorators	Zones





## Angular 1.x

### Angular 1.X

- Factories
- Services
- Providers
- Constants
- Values

ng-repeat

Not built with **mobile support in mind, but**  
**Provides mobile support with RWD Bootstrap**



## vs Angular 2

### TypeScript

JavaScript for tools

### ES6 Compliance

- Class
- Mobile first design
- More choice for languages – ES5, ES6, TypeScript and Dart
- \*ngFor (\* sign is used for structural directives)
- Local variables are defined using hash(#) prefix
- Filters are called pipes in AngularJS 2



## Angular 1.x

- controllers and \$scope services are no longer used.
- Specification for directives are through DDO
- Dependency injection was through **min safe array**.

## vs Angular 2

### TypeScript / ES6 Compliance

JavaScript for tools

- entirely component based, components are directives with a template.
- Specification for directives are with **@Directive** annotations.
- Dependency injection consists of 3 parts, **the injector, which contains the APIs to inject the dependencies and make dependency injection available. Bindings to make it possible for dependencies to be named**. Finally actual dependencies of the object are generated so they can be injected





# Angular 1.x

Ability to cache pre-compiled views  
and multi-touch support

Simple to reason  
Angular2 change detection  
rendering  
Support for server side  
AtScript a superset of ES6

Mobile First Design  
Routing performance  
Web Components  
More transparent  
Internals with zones  
Support for shadow DOM  
Improved modularity

# Angular 2





# Not any more in Angular 2

- \$scope
- Data Definition Object
- Angular module
- Controllers
- jqLite/jQuery





# Angular 2 : New Features

- Form Builder
- Change Detection
- Templating
- Routing
- Annotations
- Observables
- Shadow DOM

	Old School Angular 1.x	Angular 1.x Best Practices	Transitional Architecture	Angular 2
Nested scopes ("\$scope", watches)	Used heavily	Avoided	Avoided	Gone
Directives vs controllers	Use as alternatives	Used together	Directives as components	Component directives
Controller and service implementation	Functions	Functions	ES6 classes	ES6 classes
Module system	Angular's modules	Angular's modules	ES6 modules	ES6 modules
Transpiler required	No	No	TypeScript	TypeScript





# Angular 2 Directives

## Angular 1

- 43 directives

## Angular 2

- [] : Data Binding to Properties
- () : Binding to events





# Ways of Bootstrapping Angular App

## AngularJS 1.5.X

- 2 ways of bootstrapping angular.
  - Using **ng-app attribute in the views**
  - **Alternatively**

```
<script>
  angular.element(document).ready(function () {
    angular.bootstrap(document, ['myApp']);
  });
</script>
```

## Angular 2.1.x

- bootstrapping component through **app.module**

```
1 import { NgModule }      from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { AppComponent }   from './app.component';
4 @NgModule({
5   imports:      [ BrowserModule ],
6   declarations: [ AppComponent ],
7   bootstrap:   [ AppComponent ]
8 })
9 export class AppModule { }
```





# Routing is Changed

## AngularJS 1.5.x

- Routing is provided using ngRoute
- Third party routing is provided using UIRouter(state Router)

## Angular 2.1.x

- Routing is provided using @RouteConfig Decorators





Angular 1.x	Angular 2.x
ES5	TypeScript/ES6
Controllers	Components
Filters	Pipes
Ng-app	Bootstrap
Ng-class	[ngClass]
Ng-click	(click)
Ng-if	*ngIf
Ng-model	[(ngModel)]
Ng-repeat	*ngFor
Ng-show	[hidden]
Promises	Observables





# HTML5 Web Components

SYED AWASE





# Web Components

<http://eisenbergeffect.bluespire.com/all-about-angular-2-0/>

- Web components usually refer to a collection of **four(4)** related W3C specifications
  - Custom elements – enables the extension of HTML through custom tags.
  - HTML Imports – enables packaging of various resources (HTML,CSS, JS, etc..)
  - Template Element – enables the inclusion of inert HTML in a document
  - Shadow DOM- enables encapsulation of DOM and CSS

- 
1. “developers can create declarative components (custom elements) which are fully encapsulated (Shadow DOM)
  2. Components can describe their own views (Template Element) and can be easily packaged for distribution to other developers (HTML imports)
  3. Traditional databinding system works based on the assumption of a small number of known HTML elements with well-known events and behaviours work well. **But for Angular2 -> a new implementation of databinding is needed.”** (rob eisenberg)





# Existing Challenges with Web Development

- Undescriptive Markup in HTML
- Style conflicts, require highly specific CSS selectors
  - use of !important to force styles
  - No guarantee another style won't conflict
- No Native templates – We use `<script type="text/html">`, store HTML in hidden DOM elements and manually extract
- No bundling , to improve user experience, performance and less roundabout to the server.
- No component standard.





# Web-Components

are encapsulated, reusable and composable widgets for the web platform.

## Templates

- Reusable markup

## Custom Elements

- Define our own custom elements

## Shadow DOM

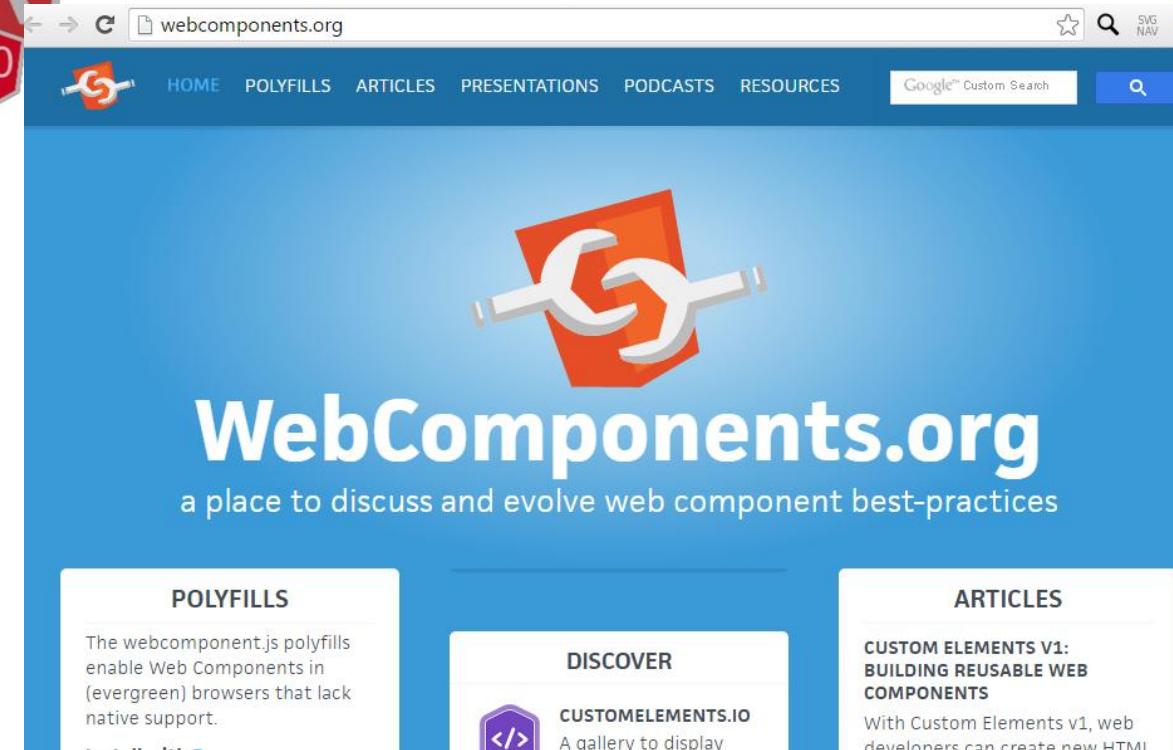
- encapsulated markup and styling  
– DOM and Style boundaries

## Imports

- Bundle HTML, JS and CSS as a single file

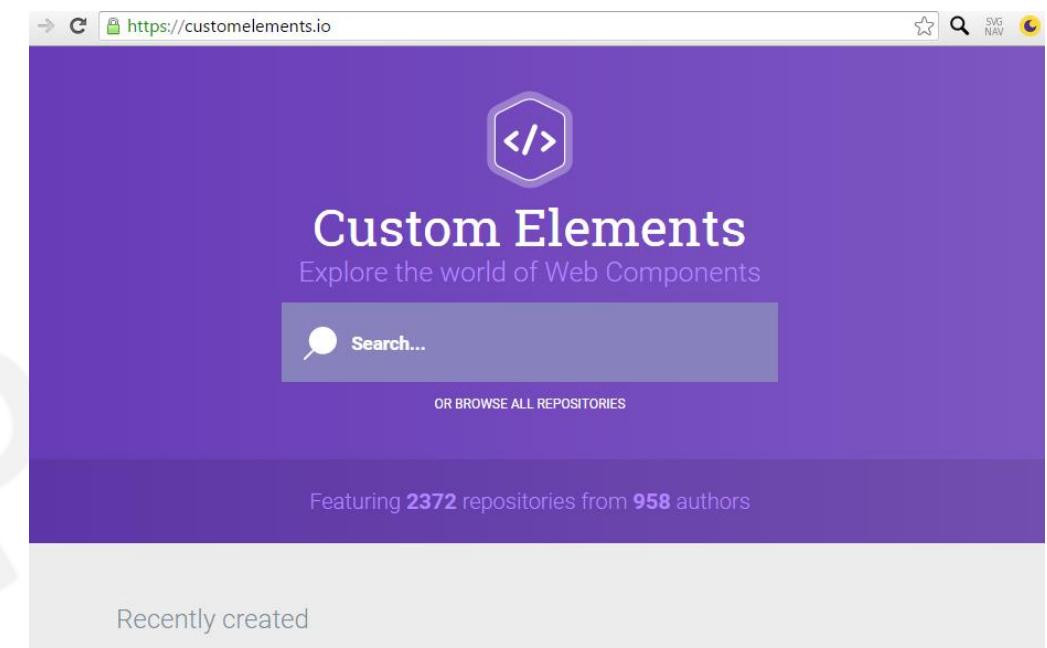


<http://webcomponents.org/>



The screenshot shows the homepage of WebComponents.org. At the top left is a red hexagonal logo with a white 'A' and '2.0'. The address bar shows 'webcomponents.org'. The main header features a large orange icon of two interlocking wrenches inside a hexagon, followed by the text 'WebComponents.org' in large white letters and 'a place to discuss and evolve web component best-practices' in smaller white text. Below this are three main sections: 'POLYFILLS' (describing webcomponent.js polyfills for browsers), 'DISCOVER' (linking to 'CUSTOMELEMENTS.IO'), and 'ARTICLES' (linking to 'CUSTOM ELEMENTS V1: BUILDING REUSABLE WEB COMPONENTS'). A search bar and navigation links for 'HOME', 'POLYFILLS', 'ARTICLES', 'PRESENTATIONS', 'PODCASTS', and 'RESOURCES' are at the top.

<https://customelements.io/>



The screenshot shows the homepage of customelements.io. The address bar shows 'https://customelements.io'. The header features a purple background with a white hexagonal icon containing '</>'. The title 'Custom Elements' is displayed in large white letters, with the subtitle 'Explore the world of Web Components' in smaller white text below it. A search bar with the placeholder 'Search...' is present. Below the search bar is a link 'OR BROWSE ALL REPOSITORIES'. A section titled 'Recently created' shows a list of recently added repositories. At the bottom, it states 'Featuring 2372 repositories from 958 authors'.

<https://html.spec.whatwg.org/multipage/scripting.html#the-template-element>

<https://www.w3.org/TR/html-imports/>

<https://www.w3.org/TR/shadow-dom/>

<https://www.w3.org/TR/custom-elements/>





# Browser Readiness

## Are We Componentized Yet?

Source: <http://jonrimmer.github.io/are-we-componentized-yet/>

Tracking the progress of **Web Components** through standardisation, polyfillification<sup>1</sup>, and implementation.

	Specged	Implementation				
		Polyfill	Chrome / Opera	Firefox	Safari	Edge
Templates			Stable	Stable	8	13
HTML Imports			Stable	On Hold	No Active Development	Vote
Custom Elements			Stable	Flag	Prototype	Vote
Shadow DOM			Stable	Flag	10	Vote

Cells are clickable! Yellow means in-progress; green means mostly finished.





# 3 Major Libraries

**For working with and extending web components**

- X-Tags by Mozilla

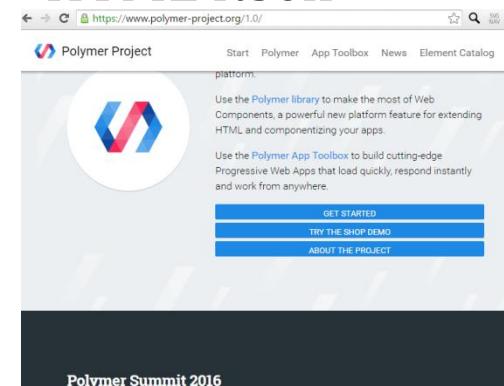
- allows you to easily create elements to encapsulate common behavior or use existing custom elements to quickly get the behavior you are looking for.



<https://x-tag.github.io/>

- Polymer by Google

- encapsulated and interoperable custom elements that extend HTML itself.



<https://www.polymer-project.org/1.0/>

- Bosonic

- Built on top of web components polyfill library.



<http://bosonic.github.io/>





<https://whatwg.org/>



## Welcome to the WHATWG community

Maintaining and evolving HTML since 2004

### FAQ

Get answers to your questions about HTML and the WHATWG

### HTML

Read, use, or implement the HTML Living Standard

<https://component.kitchen/>



## Component.KITCHEN

HOME BLOG TUTORIAL ABOUT

### Web components accelerate app development

Web components let you extend HTML with new capabilities so you can write better web apps faster. Component KITCHEN are experts on this transformational technology. We design and develop web components, we work with a range of industry players to improve the underlying specs and libraries, and we consult with companies who want to use web components to make their products better.

Learn

About

JUNE 13, 2016

#### Can Service Workers service background applications?

I had a thought experiment on how I might port an application I once developed for native Android to a web app, even if it were to run solely on Android devices. The application behaves like a mantle clock: every fifteen or thirty minutes, it wakes up and plays a custom chime. My son used to call it "Big Ben in Your Pocket."

MAY 2, 2016





# Why Web Components?

- 1 • Descriptive markup
- 2 • Bootstrapped elements
- 3 • Easily replaced and upgraded
- 4 • Fewer integration mistakes
- 5 • Clearer interface/api





# HTML Templates

- contains inert chunks of markup intended to be used later
- Existing strategies have Cross Site Scripting risk from using .innerHTML
- Nothing inside runs or renders
- Ng-template (angular)
- Templates in KnockOutJS

Custom template name

```
<script type="text/product-list-stamp">  
<script>
```





# Declaring HTML templates

- Contain inert chunks of markup intended to be used later
- Templates are parsed not rendered
- Working directly with DOM
- Hidden from document, Cannot traverse into its DOM

```
document.querySelector('#product-list-template.sycliq-style_QHY') == null  
  
<template id="product-list-stamp">  
  <!--dynamically populated at runtime-->  
  <img src="" alt="">  
  <div class="sycliq-style_QHY"></div> ←  
  <input type="text" name="input_subscribe" value="subscribe"/>  
</template>
```





# Using template content

- Two ways
- template.content(a document fragment)
- Template.innerHTML

```
<template id="product-list-stamp">
  <!--dynamically populated at runtime-->
  <img src="" alt="">
  <div class="sycliq-style_QHY"></div>
  <input type="text" name="input_subscribe" value="subscribe"/>
</template>
<script>
  var temp = document.querySelector('#product-list-stamp');
  temp.content.querySelector('img').src ="ebayProduct-list-1.jpg";
  document.body.appendChild(temp.content.cloneNode(true));
</script>
```



# Using Template Steps

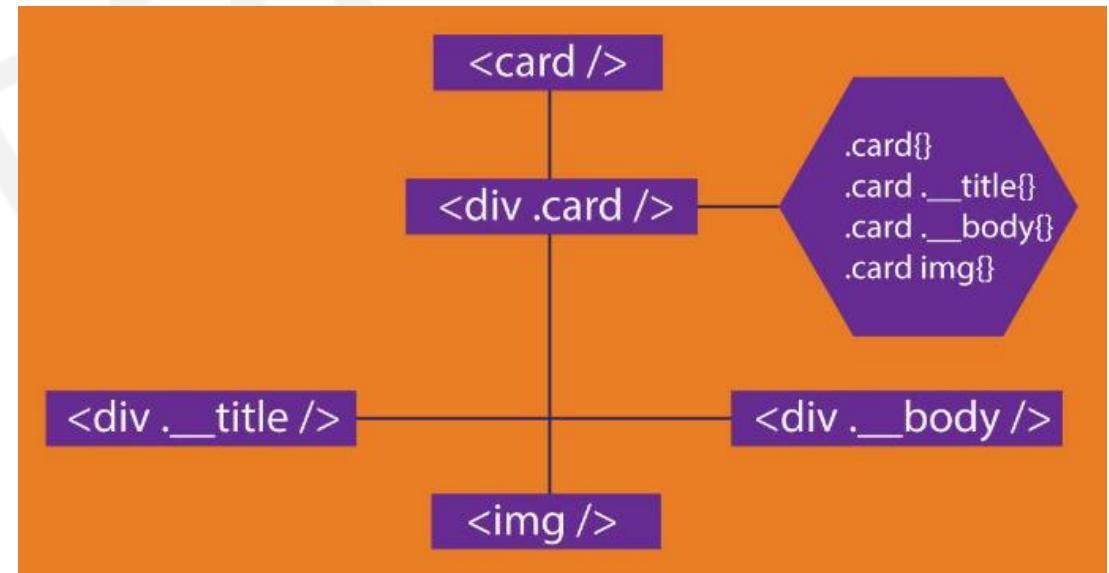
```
//1.Get a reference to the template  
var template = document.querySelector('templateName');  
//2.Use document.importNode to clone the template's Content  
var clone = document.importNode(template.content,true);  
//3.change the target element within the template as desired  
clone.querySelector('.verb').textContent="Syed Awase Khirni";  
//4 Append element to page  
document.body.appendChild(clone);
```





# Shadow DOM

- included in the web components standard by W3C.
- It basically allows group of DOM implementation to be hidden inside a single element and encapsulate styles to the element.
- This means that encapsulated styles will only be available for that group of DOM elements.





# Abstraction with Shadow DOM

- Not all browsers support Shadow DOM.
- Angular 2 allows us to choose whether to implement Shadow DOM, just to emulate it(default) or not use it at all. This technique of handling Shadow DOM in Angular 2 is known as View Encapsulation.
  - There are 3 states of view encapsulation
    - None – All elements are spit out – no shadow DOM at all
    - Emulated – This actually tries to emulate shadow DOM to give us the feel that we are scoping our styles. This is not a real Shadow DOM but a strategy to make all browsers smile at our code
    - Native: Shadow DOM is completely enabled.





# Shadow DOM : ViewEncapsulation

```
//metadata and template
@Component({
  templateUrl:'card.html',
  styles:[
    .card{
      height:70px;
      width:100px;
    }
  ],
  encapsulation:ViewEncapsulation.Native
  //encapsulation: ViewEncapsulation.None
  //encapsulation: ViewEncapsulation.Emulated is default
})
```





# Virtual DOM

- In memory representation of the DOM and events system
- Even HTML5 events in IE8
- Interacting with DOM -> slow process
- Interacting with JS In-Memory Objects – Fast process
  - 60 fps fast
  - Faster parsing
  - Faster validation
- Since no real browser, can easily render on server or client
- Easier to Test
- Autobinding and Event Delegation





# Problems with Existing HTML Markup

- Undescriptive HTML Markup
- Style conflicts – avoiding style conflicts in images requires highly specific CSS selectors
  - No Native templates,
  - No Bundling and minification
  - No Standard



# WEB COMPONENTS

- 1.Templates
- 2.Custom Elements
- 3.Shadow DOM
- 4.Imports





# Why web components?

- a. Descriptive markup syntax
- b. Don't start at square zero
- c. Easily replaced and upgraded
- d. Fewer Integration mistakes
- e. Clearer interface/ API





# Why do we need scoping

1. How do I avoid accidentally styling other parts of the page?
2. How do you avoid other people accidentally styling my component
3. How do I hide away my markup from accidental manipulation?

**Light DOM**

The DOM you know today

**Shadow DOM**

The DOM that hides away complexity

Logical DOM





# HTML5 WEB WORKER

## JAVASCRIPT CONCURRENCY

### Section III

IE10+, Firefox 3.5+, Safari4+, Opera10.6+, Chrome





# WEB WORKER

- A web worker is a JavaScript running in the background, without affecting the performance of the page.
- When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.
- However, a web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.
- `w = new Worker("demo_workers.js");`





# Two kinds of Web Workers

## Dedicated Workers

- A dedicated worker is only accessible from the script that first **spawned it**

## Shared Workers

- A shared worker can be accessed from multiple scripts.
- <http://ejohn.org/blog/web-workers/>
- <https://www.html5rocks.com/en/tutorials/workers/basics/>
- [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Workers\\_API/Using\\_web\\_workers?redirectlocale=en-US&redirectslug=Using%20web%20workers](https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers?redirectlocale=en-US&redirectslug=Using%20web%20workers)
- <https://developers.google.com/web/fundamentals/getting-started/primers/service-workers>





Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project ([www.geo-spirit.org](http://www.geo-spirit.org)). He currently provides consulting services through his startup [www.territorialprescience.com](http://www.territorialprescience.com) and [www.sycliq.com](http://www.sycliq.com)

SYED AWASE

RESEARCHER | ENTREPRENEUR | TECHNOLOGY COACH

# ES6 : ECMA SCRIPT 2015





# Why ECMA 2015/ES6?

- Javascript has no standard library
- Javascript won't run outside the browser
- The DOM is too slow for rendering
- Javascript is single threaded by design
- Asynchronous programming -> callback hell
- Javascript has no packaging or a linker to tie packages together
- Web resources need to be minified and zipped for performance
- Javascript is too slow for videogames
- Machine generated output is more difficult to debug
- Poor performance on mobile devices.
- Ballooning project size and complexity.





kangax.github.io/compat-table/es6/ SVG NAV

**COMPAT** ECMAScript 5 6 2016+ next intl non-standard compatibility table Flattr 43 by kangax Gratipay & webbedspace & zloirock Fork 344

Sort by **Engine types** Show obsolete platforms Show unstable platforms V8 SpiderMonkey JavaScriptCore Chakra Carakan KJS Other  
. Minor difference (1 point) Small feature (2 points) Medium feature (4 points) Large feature (8 points)

Feature name	Current browser	Compilers/polyfills							Desktop browsers															Server		
		Traceur	Babel + core-js <sup>[2]</sup>	Closure	Type-Script + core-js	es6-shim	IE 11	Edge 13 <sup>[4]</sup>	Edge 14 <sup>[4]</sup>	FF 45 ESR	FF 48	FF 49	FF 50	CH 52+, OP 39+ <sup>[1]</sup>	SF 7.1, SF 8	SF 9	SF 10	SFTP	WK	KQ 4.14 <sup>[5]</sup>	PJS	Node 0.12 <sup>[6]</sup>	Node 4 <sup>[6]</sup>			
<b>Optimisation</b>																										
proper tail calls (tail call optimisation)		0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	0/2	
<b>Syntax</b>																										
default function parameters	7/7	4/7	4/7	4/7	5/7	0/7	0/7	0/7	7/7	4/7	4/7	4/7	4/7	7/7	0/7	0/7	7/7	7/7	7/7	0/7	0/7	0/7	0/7	0/7	0/7	
rest parameters	5/5	4/5	3/5	2/5	4/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	5/5	5/5	5/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5
spread (...) operator	15/15	15/15	13/15	12/15	4/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	5/15	9/15	15/15	15/15	15/15	0/15	0/15	0/15	0/15	0/15	0/15	0/15
object literal extensions	6/6	6/6	6/6	4/6	6/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	1/6	5/6	6/6	6/6	6/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6
for..of loops	9/9	9/9	9/9	6/9	3/9	0/9	0/9	7/9	9/9	7/9	7/9	7/9	7/9	9/9	2/9	8/9	9/9	9/9	9/9	0/9	0/9	7/9	7/9	7/9	7/9	7/9
octal and binary literals	4/4	2/4	4/4	4/4	4/4	2/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	4/4	4/4	4/4	4/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
template literals	5/5	4/5	4/5	3/5	3/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	5/5	5/5	5/5	5/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5
RegExp "v" and "u" flags	5/5	3/5	3/5	0/5	0/5	0/5	0/5	5/5	5/5	2/5	5/5	5/5	5/5	5/5	0/5	0/5	5/5	5/5	5/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5
destructuring, declarations	22/22	20/22	21/22	18/22	15/22	0/22	0/22	0/22	22/22	19/22	21/22	21/22	21/22	22/22	9/22	19/22	22/22	22/22	22/22	0/22	0/22	0/22	0/22	0/22	0/22	0/22
destructuring, assignment	24/24	23/24	24/24	16/24	19/24	0/24	0/24	0/24	24/24	21/24	23/24	23/24	23/24	24/24	12/24	21/24	24/24	24/24	24/24	0/24	0/24	0/24	0/24	0/24	0/24	0/24
destructuring, parameters	23/23	19/23	20/23	17/23	15/23	0/23	0/23	0/23	23/23	18/23	19/23	19/23	19/23	23/23	10/23	18/23	23/23	23/23	23/23	0/23	0/23	0/23	0/23	0/23	0/23	0/23
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	0/2	0/2	2/2	2/2	1/2	1/2	1/2	1/2	2/2	0/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2

<http://kangax.github.io/compat-table/es6/>





# Can ES6 be used in browsers?

<http://pointedears.de/scripts/test/es-matrix/>



96%

<https://bugs.chromium.org/p/v8/issues/list?q=label:Harmony>



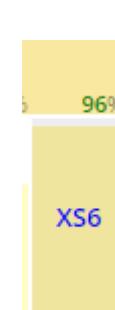
90%

[https://bugzilla.mozilla.org/show\\_bug.cgi?id=694100](https://bugzilla.mozilla.org/show_bug.cgi?id=694100)



86%

EDGE





# Transpilers

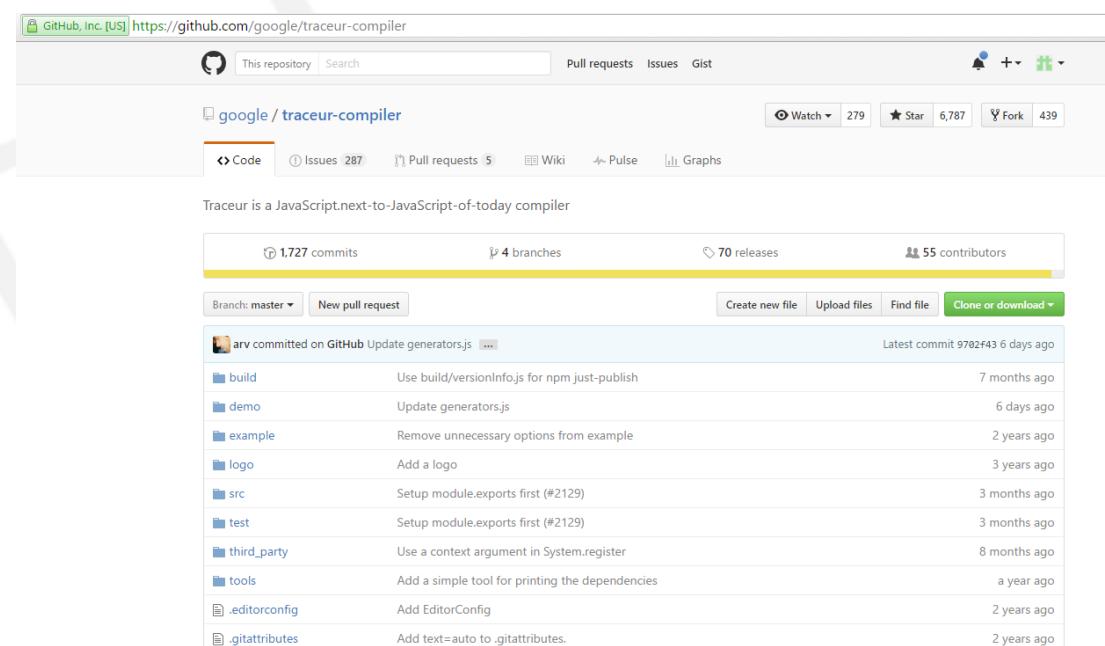
## Babel



The screenshot shows the Babel.js homepage. At the top, there's a navigation bar with links for Learn ES2015, Setup, Plugins, Usage, Try it out, Discuss, Chat, Issues, Blog, Twitter, and GitHub. The main headline reads "Babel is a JavaScript compiler." Below it, a sub-headline says "Use next generation JavaScript, today." A prominent callout box in the center says "v6.14.0 Released!" with a link. To the right of the box is a "Star" button with the number 17,314. The background features abstract black and white brushstrokes.

<https://babeljs.io/>

## Traceur



The screenshot shows the GitHub repository page for "google / traceur-compiler". The header includes links for This repository, Search, Pull requests, Issues, Gist, Watch (279), Star (6,787), Fork (439), and Clone or download. The repository description is "Traceur is a JavaScript.next-to-JavaScript-of-today compiler". It shows 1,727 commits, 4 branches, 70 releases, and 55 contributors. A list of recent commits is provided:

Commit	Description	Date
arv committed on GitHub	Update generators.js	6 days ago
build	Use build/versionInfo.js for npm just-publish	7 months ago
demo	Update generators.js	6 days ago
example	Remove unnecessary options from example	2 years ago
logo	Add a logo	3 years ago
src	Setup module.exports first (#2129)	3 months ago
test	Setup module.exports first (#2129)	3 months ago
third_party	Use a context argument in System.register	8 months ago
tools	Add a simple tool for printing the dependencies	a year ago
.editorconfig	Add EditorConfig	2 years ago
.gitattributes	Add text=auto to .gitattributes.	2 years ago

<https://github.com/google/traceur-compiler>





# ES6 New Features

- Classes
  - Modules
  - New Methods for Strings and Arrays
  - Promises
  - Maps, Sets
  - Completely new features
    - Generators
    - Proxies
    - WeakMaps
  - Reflect API
  - Proxy API
- **Static typing is not part of ES6.**
  - **Static typing is available in**
    - Microsoft Typescript : transpiled to ES5 and throws away the type information while doing so.
    - Facebook Flow: A type checker for ECMAScript 6 that is based on flow analysis. It only adds optional type annotations to the language and infers and checks types. DOES NOT HELP IN COMPILING ES6 to ES5.





# Benefits of Static Typing

- Allows you to detect a certain category of errors earlier, because the code is analyzed statically (during development, without running code).
- Static typing is complementary to testing and catches different errors.





# ECMAScript 6 Tools

## Transpilers

Babel	Turns Es6+ code into Vanilla ES5 with no runtime
Traceur	ES6 Features -> ES5 includes classes, generators, promises, destructuring pattern, default parameters
ES6IFY	Traceur compiler wrapped as a Browserify V2 Platform
Babelify	Babel transpiler wrapped as a Browserify transform
Es6-transpiler	ES6-> ES5 Includes classes, destructuring, default parameters, spread
ES6-module-transpiler	ES6 modules to AMD or CJS
Regenerator	Transform ES6 yeild/generator functions to ES5
Jstransform	A simple utility for pluggable JS syntax transforms. Comes with a small set of Es6-> ES5 transforms
Defs	ES6 block-scoped const and let variables to ES3 vars
Es6_module_transpiler-rails	ES6 modules in the Rails Asset Pipeline





# ECMA Script Tools

## Transpiler

Sweet.js	Macros that compile from ES6 to ES5
Bitovi's Transpile	Converts ES6 to AMD, CHS and StealJS
Regexpu	Transform unicode-aware ES6 regular expression to ES5
Lebab	Transformation for ES5 Code to ES6 (approximates)





# Strict mode and ECMAScript 6

- Strict mode was introduced in ECMAScript 5 to clean up the language.
- Strict mode introduces three kinds of breaking changes

1. Syntactic changes: some previously legal syntax is forbidden in strict model
  - **with** - lets users add arbitrary objects to the chain of variable scope
  - **Deleting an unqualified identifier**

[http://exploringjs.com/es6/ch\\_one-javascript.html](http://exploringjs.com/es6/ch_one-javascript.html)

## 2. More errors

- Assigning to an undeclared variable causes a ReferenceError
- Changing read-only properties causes a TypeError

## 3. Different semantics

- Arguments doesn't track the current values of parameters
- this is undefined in non-method functions





# Running ES6

```
//installing traceur  
npm install --g traceur --save -dev  
//running your es6 file  
traceur --out build.js --script myfile.js  
//running build  
traceur build.js
```



<http://google.github.io/traceur-compiler/demo/repl.html#>

- <http://www.es6fiddle.net/> <http://es6console.com/>

<http://babeljs.io/repl/#?babili=false&evaluate=true&lineWrap=false&presets=es2015%2Creact%2Cstage-2&code=>





# New in ECMA 2015

- Arrow function
- Classes
- Modules
- Block Scope (let/const)
- Extended Object Literal
- Default Params
- Rest Params
- Spread Operator
- Destructuring
- Iterator
- Generator
- Template Literal
- Tail Call Optimization





# New built-in classes and objects

- Promise
- Map
- Set
- WeakMap/WeakSet
- TypedArray
- Symbol
- Proxy/Reflect
- Improvement of existing classes
  - String
  - RegExp
  - Array
  - Object
  - Math
  - Number





# Default export/import

- Export : expose a class so that others can use it by importing it
- Import : import external classes to implement reusable functionality

```
//import export example
function exim(){return 'exim';}
function eximbar(){return 'eximbar';}
//export - to expose the functions so that others can use
export {exim, eximbar};
```

```
import {exim, eximbar} from '1-Example';
exim();
eximbar();
```

```
import * as lib from '1-Example';
lib.exim();
lib.eximbar();
```





# Let and Block scoping

## Destructured assignment

- No Hoisting takes place when we use **let**, making sure that the variable declaration takes place before it is being used.

```
// block scoping without hoisting
let [firstson, secondson, thirdson] = [1,2,3];
let {firstplace, secondplace, thirdplace} = {firstplace:1, secondplace:2, thirdplace:3};

console.log(firstson, secondson, thirdson, firstplace, secondplace, thirdplace);

//Block scoping
let productId = 254;
{
  let productId=23213;
}
console.log(productId);
```

```
let a=[12,12,123,123,4123,145,1235,14123123];
let b= [543,623423,6234,234,63,232,23,2,4,2];
for (let i = 0; i < a.length; i++) {
  let x = a[i]
  console.log(x);
}

for (let i = 0; i < b.length; i++) {
  let y = b[i]
  console.log(y);
}

let callbacks = []
for (let i = 0; i <= 2; i++) {
  callbacks[i] = function () { return i * 2 }
  console.log(callbacks[i]());
}

callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
```



```
var updateFunctions =[];  
for (var i =0; i<2;i++){  
    updateFunctions.push(function(){return i;});  
}  
console.log(updateFunctions[0]());|
```

```
//ES2015  
var updateFunctions =[];  
for (let i =0; i<2;i++){  
    updateFunctions.push(function(){return i;});  
}  
console.log(updateFunctions[0]());|
```





# Const

- constant is a variable whose value cannot be changed.

```
//ES2015
const DB_CONNEC = "oracle db";
console.log(DB_CONNEC);
```





# Block Scope in ES6

```
function updateEmployeeId(){
  employeeId = 123123;
}
let employeeId =null;
updateEmployeeId();
console.log(employeeId); → 123
```

Temporal Dead ZONE

```
let itemId=2112;
for(let itemId=0; itemId<100; itemId++){
  console.log(itemId);
}
console.log(itemId); → 2112
```





# Arrow Functions

```
> document.addEventListener('click',
  ()=>console.log(this));
< undefined
```

VM223:2

```
▶ Window {speechSynthesis: SpeechSynthesis,
  caches: CacheStorage, LocalStorage: Storage,
  sessionStorage: Storage, webkitStorageInfo:  

DeprecatedStorageInfo...}
```





# Arrow Functions

```
// we cannot bind the value of a function to a new value with =>
var employee={
    empid: 12123,
    callName: function(){
        return()=>console.log(this.empid);
    }
};
var newEmployee={
    empid: 51222
};
employee.callName().bind(newEmployee)(); //value 12123 is printed
```





# Arrow Functions => (fat arrow symbol)

- they are shorthand form of anonymous function expression that already exist in javascript.
- *this* in arrow function uses lexical scoping, its value is always *inherited* from the enclosing scope.

```
//ES2015
var getTotal = noofitems=> noofitems*99.99;
console.log(getTotal(9));
```

```
const arrvalues = [41, 121, 51];
const squares = arrvalues.map(x => x * x);

console.log(squares);
```

```
let getPrice =()=> 99.99;
console.log(typeof getPrice);
console.log(getPrice());
```





# Arrow Functions

```
//ES2015
let computeTax = (salary, tax)=> salary-(salary*tax);
console.log(computeTax(1000, 0.28));
```

```
//ES2015
let salesTax = (salary,tax)=>{
    let g_salary= salary+ salary*13.8;
    g_salary = g_salary -(g_salary*tax);
    return g_salary;
}
console.log(salesTax(1000,0.15));
```





# String interpolation via Template literals

- We can use string interpolation and multi-line strings via template literals

ES5

```
function printCoord(x, y) {  
  console.log('('+x+', '+y+')');  
}
```

ES6

```
function printCoord(x, y) {  
  console.log(`(${x}, ${y})`);  
}  
  
const HTML5_SKELETON = `  
  <!doctype html>  
  <html>  
    <head>  
      <meta charset="UTF-8">  
      <title></title>  
    </head>  
    <body>  
    </body>  
  </html>`;
```





# Default Function Parameters

```
//ES2015
var stampEmployee = function(empId=1111, designation='SoftwareEngineer', salary='40000'){
  console.log(` ${empId} ${designation} ${salary}`);
};
stampEmployee(undefined, undefined, 45000);
```

---

```
//ES2015
var Payables = function(price, gst=price*0.15){
  console.log(price+gst);
};
Payables(799);
```





# Default Function Parameters

```
//ES2015
let gstTax = 0.15;
let Payables = function(price, gst=price*gstTax){
    console.log(price+gst);
};
Payables(799);
```





# Rest parameter in ES2015

- A ***rest parameter*** is indicated by three dots (...) preceding a named **which becomes an Array *containing the rest of the parameters passed to the function.***
- Allows users to specify that multiple Independent arguments should be Combined into an array.

```
//ES2015
var showCategories = function(itemId, ...categories){
  console.log(categories instanceof Array);
}
showCategories(123, 'electronic item', 'electronic appliance');
```





# Spread operator in ES2015

- Allows you to specify an array that should be split and have its items passed in as separate arguments to a function.

```
//ES2015
let tickets=[12,24,43,145];
var maxTicket = Math.max(...tickets);
console.log(maxTicket);
```





# Object Literal Extensions

- 

```
//object literal extension
var empid= 123214213, empName="Syed Awase", tax=0.18,salary=10000;
var employeeView={
    empid,
    empName,
    salary,
    tax,
    computeTax(){
        return (this.salary-this.salary*(1-tax));
    }
};

console.log(employeeView);
console.log(employeeView.computeTax());
```





# Object Literal Extensions

```
//example3 object literal extension
var attrprps ='nationality';
var dualcitizenship = 'dual nationality';
var dualcitizen={
  [attrprps+"-0001"]:dualcitizenship
}
console.log(dualcitizen);
```

```
//example 4 with getter and setter
var id ='productId';
var productView ={
  get[id](){return true;},
  set[id](value){}
};
console.log(productView.productId);
```





# For..of Loop

- Allows us to easily iterate over elements of a collection.
- For..of iterates over the values of elements of a collection not the keys.
- A collection can be an array, set, list, custom collection object etc.

```
//for..of loop example
var names = ['Syed Awase', 'Syed Azeez', 'Syed Rayyan', 'Syed Ameese'];
for(var name of names){
  console.log(name);
}

var places =[ 'Kashmir','Kanyakumari','Kalangote', 'karimnagar',];
for(var place of places){
  console.log(place);
}
```





# Octal and Binary Literals

- In ES5, we had to use '0' in front of the numbers to specify it's a octal number., which often did not work when used in '***strict mode***'
- "0o" is a new way of creating a number using octal value in ES 2015.
- "0b" lets you create a number using binary of the number directly.

```
//binary values
var b = 0b10; //alternative 0B10 gives same value
console.log(b);
```

---

```
//octal value
var i = 0o10;
console.log(i)
```





# Destructuring

- makes it easier to work with objects and arrays in Javascript.
- Using a pattern syntax similar to object and array literal, we can poke into data structures and pick out the information we want into variables.

```
//destructuring
//object pattern matching
let {lName, fName}= {fName:'Awase Khirni', age:39, lName:'Syed'};
console.log(lName+","+fName);
//Array pattern matching
let feedback= [1,2,3,4,5];
let [stronglyDisagree, Disagree, Neutral, Agree, stronglyAgree] = feedback;
let [poor, ...remaining]=feedback;//rest parameters
console.log(Neutral);
//example 3
let salary=['17000', '50000'];
let [low,average, high="88000"]= salary;
console.log(high);
//example 4
let totalearnings = ['38000', '570231', ['123412512', '87712687']];
let [lowearning,averageearning, [monthlylow, monthlyhigh]]= totalearnings;
console.log(monthlylow);
```





# Modules

- Modules are one of the most important features of any programming language. JS lacks this very basic feature. ES5, AMD was achieved either using commonJS or requireJS.
- In ES6 each module is defined in its own file.
- The functions or variables defined in a module are not visible outside unless you explicitly export them
- ES6 modules are declarative in nature.
- use
  - Export to export
  - Import to import

```
//es6-module loader
npm install -g es-module-loader --save
compile-modules convert -I scripts -o out app.js utility.js --format commonjs
compile-modules convert -I scripts -o out app.js utility.js //alternatively
//running the output
node out|
```





# Basic Module Demo in ECMA 2015

## Onemain.js

```
//onemain.js file using modules
import {projectId as id, projectName} from 'onemodeule.js';
console.log(` ${projectName} has id: ${id}`);
```

```
1 import * as values from 'twomodule.js';
2 console.log(values);
```

## Onemodeule.js

```
//onemodeule.js file using modules
export let projectId=89798798798;
export let projectName ="SycliQ IOT Platform";
```

```
1 let projectId=124123;
2 let projectName = "SycliQ GeoAnalytic Platform";
3 export {projectId, projectName};
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>compile-modules convert -I scripts -o out onemain.js onemodeule.js
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>node out
SycliQ IOT Platform has id: 89798798798
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>
```



# Named Exports in Modules

- A module can export multiple things by prefixing their declarations with the keyword `export`. These exports are distinguished by their names and are called **named exports**.

## Error

```
//named export example one
import {projectId} from 'onemodule.js';
projectId=0912312312;
console.log(projectId);
```

```
//onemodule.js file using modules
export let projectId=89798798798;
export let projectName ="SycliQ IOT Platform";
```





# Named Exports in Module

## Fourmain.js

```
import {sycliqproject} from 'fourmodule.js';
sycliqproject.Id = 123541412312;
console.log(sycliqproject.Id);
```

## Fourmodule.js

```
1 //named export example two
2 export let sycliqproject={
3     Id:3215123123
4 };
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>compile-modules convert -I scripts -o out fourmain.js fourmodule.js
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>node out
123541412312
```





# Named Exports in Module

## Fivemain.js

```
import {employee, showEmployee} from 'fivemodeule.js';
employee.employeeId = 'RD36785';
showEmployee();
console.log(employee.employeeId);
```

## Fivemodeule.js

```
export let employee ={
  employeeId :`RD27865`,
  employeeName:'Syed Awase'
};

export function showEmployee(){
  console.log(employee.employeeId);
}
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>compile-modules convert -I scripts -o out fivemain.js fivemodeule.js
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>node out
RD36785
RD36785
```



# ES2015 Modules

## EXPORT

Company.ts

```
export class Company{  
}
```

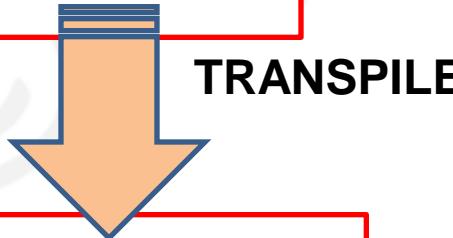
Company.js

```
function Company(){  
}
```

## IMPORT

Company-list.ts

```
import {Company} from './Company'
```





# Class in ES2015

- ES6 classes are not something that is radically new, they are mainly provided with more convenient syntax to create old-school constructor functions.

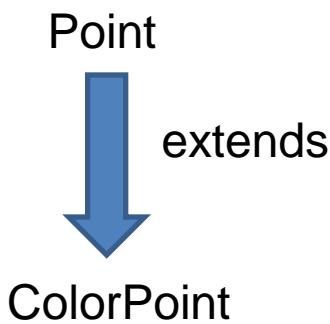
```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur --out build.js --script student.js
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur build.js
Student { id: 1, name: 'SyedAwase', age: 19, section: 'A', level: '+2' }
function
```

```

1  class Student{
2      //constructor of the class
3  constructor(id,name, age, section,level){
4      //this points to the current object
5      this.id =id;
6      this.name = name;
7      this.age =age;
8      this.section = section;
9      this.level = level;
10 }
11 //member functions
12 getId(){ return this.id;    }
13 getName(){ return this.name;   }
14 getAge(){ return this.age;    }
15 getSection(){ return this.section;   }
16 getLevel(){  return this.level;  }
17
18 setId(id){  this.id = id;    }
19 setName(name){  this.name = name;   }
20 setAge(age){ this.age = age;    }
21 setSection(section){  this.section = section;   }
22 setLevel(level){ this.level=level;    }
23 }
24 //creating a new instance of the class student
25 var aks = new Student(1,'SyedAwase',19,'A','+2');
26 console.log(aks);
27 console.log(typeof Student);
```



# Extending Class in ES2015



```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur --out build.js --script point.js
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur build.js
(50,10) in red
true
true
```

```
1 class Point{
2   constructor(x,y){
3     this.x = x;
4     this.y=y;
5   }
6
7   toString(){
8     return `(${this.x},${this.y})`;
9   }
10 }
11 //extending the class point
12 class ColorPoint extends Point{
13   constructor(x,y, color){
14     super(x,y);
15     this.color = color;
16   }
17   toString(){
18     return super.toString() + " in " + this.color;
19   }
20 }
21
22 let cp = new ColorPoint(50,10,'red');
23 console.log(cp.toString());
24 console.log(cp instanceof ColorPoint);
25 console.log(cp instanceof Point);
```





# Class : Additional Insights

```
1 //function
2 function Project(){}
3 console.log(window.Project === Project); //true
4 //class
5 class SoftwareProject{
6
7 };
8 console.log(window.SoftwareProject === SoftwareProject); // false
9 //by creating a class we are not polluting the global context-window
```





# Class :extends



```
1 class Human{
2     canBreathe(){
3         return 'can breathe';
4     }
5     canTalk(){
6         return 'can talk';
7     }
8 }
9 class Man extends Human{
10
11     canTalk(){
12         return 'little talk';
13     }
14 }
15 class Woman extends Human{
16     canTalk(){
17         return 'talks relatively more than Man';
18     }
19 }
20 let kapilsharma = new Man();
21 console.log(kapilsharma.canTalk());
22
23 let gutti = new Woman();
24 console.log(gutti.canTalk());
--
```





# Class: super

```
1 let Project={  
2     getTaskCount(){return 500;}  
3 };  
4  
5 let AgileProject={  
6     getTaskCount(){  
7         return super.getTaskCount()+200;  
8     }  
9 }  
10 Object.setPrototypeOf(AgileProject, Project);  
11 console.log(AgileProject.getTaskCount());
```

```
1 class Project{  
2     getTaskCount(){  
3         return 1500;  
4     }  
5 }  
6 class ScrumProject extends Project{  
7     getTaskCount(){  
8         return super.getTaskCount()+300;  
9     }  
10 }  
11 let csm = new ScrumProject();  
12 console.log(csm.getTaskCount());
```

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur --out build.js --script project.js
```

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur build.js  
1800
```





# Static keyword in ES2015

- static properties or class properties are properties of the class.

```
class Employee {  
    static getemployeeId(){  
        return 1231243;  
    }  
}  
  
console.log(Employee.getemployeeId());
```





# new.target

- ***meta property***
- Sole purpose is to retrieve the current value of the current function environment.
- It is a value that is set when a function is called
- mainly ***used in a constructor***

```
1 class Employee{  
2     constructor(){  
3         console.log(typeof new.target);  
4     }  
5 }  
6  
7 var aks = new Employee();
```

```
1 class Employee{  
2     constructor(){  
3         console.log(new.target);  
4     }  
5 }  
6 class SycliqEmployee extends Employee{  
7     constructor(){  
8         super();  
9     }  
10 }  
11 var sycliqsak = new SycliqEmployee();
```





# Symbols in ES2015

- A symbol is a unique and immutable data type and may be used as an identifier for object properties.
- Purpose of a symbol is to generate a unique identifier, but we never get to access the identifier.

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur --out build.js --script symbolexample2.js
```

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur build.js
[ 'bookTitle' ]
```

```
1 let Book ={
2   bookTitle: 'Fountain Head' ,
3   [Symbol.for('bookAuthor')]: 'Ayn Rand'
4 };
5 console.log(Object.getOwnPropertyNames(Book));
```

Symbols enable access control for object state

Symbols are a new primitive type





## CODE



```
1 let aks = Symbol("Syed Awase");
2 console.log(typeof aks);
3 console.log(aks.toString());
4 //symbols used with constants
5 const S_AK =Symbol("Syed Awase Khirni");
6 console.log(S_AK.toString());
7 let sak1 = Symbol("Syed Rayyan Awais");
8 let sak2 = Symbol("Syed Rayyan Awais");
9 console.log(sak1==sak2);
10 let sak1_addr = Symbol.for("Syed Rayyan Awais");
11 let sak2_addr = Symbol.for("Syed Rayyan Awais");
12 console.log(sak1_addr.toString());
13 console.log(sak2_addr.toString());
14 console.log(sak1_addr==sak2_addr);
15 let desc = Symbol.keyFor(sak1_addr);
16 console.log(desc);
```

## CONSOLE

```
symbol
Symbol(Syed Awase)
Symbol(Syed Awase Khirni)
false
Symbol(Syed Rayyan Awais)
Symbol(Syed Rayyan Awais)
true
Syed Rayyan Awais
```





```
2.1 let Book =  
2     bookTitle: 'Fountain Head' ,  
3     [Symbol.for('bookAuthor')]: 'Ayn Rand'  
4 };  
5 console.log(Object.getOwnPropertyNames(Book));  
6 console.log(Object.getOwnPropertySymbols(Book)); //new method in ES6
```





# Well-known Symbol

```
//well known Symbol
let Vehicle= function(){
};

Vehicle.prototype[Symbol.toStringTag]='Bare Bones';
let truck = new Vehicle();
console.log(truck.toString());
```

```
let carManufacturers = ['mercedes','hyundai','audi','bmw','fiat'];
console.log([] .concat(carManufacturers));
```

```
1 //wellknown Symbol
2 let v1 = [12,24,45,56,78];
3 v1[Symbol.toPrimitive]= function(hint){
4     console.log(hint);
5     return 99;
6 };
7 let total = v1+100;
8 console.log(total);
```

default

199



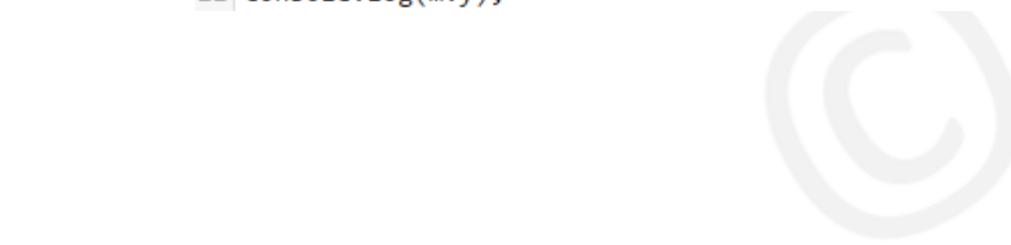


# Object Extensions

```
1 //object extensions example
2
3 let m={
4   x:786
5 };
6
7 let n={
8   y:687
9 };
10 Object.setPrototypeOf(m,n);
11 console.log(m.y);
```

687

```
//Object.assign
let s={a:347}, q={b:743};
let target={};
Object.assign(target,s,q);
console.log(target);
//object.is
let amount =0, total=-0;
console.log(Object.is(amount,total));
```





# String Extensions

```
//string extensions example
let bookTitle = "Exploratory Representations for Geographic Information Retrieved from the Internet";
console.log(bookTitle.startsWith('Exploratory'));//returns true
console.log(bookTitle.endsWith('Internet'));
console.log(bookTitle.includes('ra'));
```





# Number Extensions

```
1 console.log(Number.parseInt === parseInt);
2 console.log(Number.parseFloat === parseFloat);
3 let no='NaN';
4 console.log(isNaN(no));//es5
5 console.log(Number.isNaN(no));//es2015
6 let x='78126';
7 console.log(isFinite(x));//es5
8 console.log(Number.isFinite(x));//es2015
9 let total = 782139.2;
10 console.log(Number.isInteger(total));
11 console.log(Number.isInteger(NaN));
12 console.log(Number.isInteger(Infinity));
13 console.log(Number.isInteger(undefined));
14 console.log(Number.isInteger(10));
15 let a = Math.pow(2,53) -1;
16 console.log(Number.isSafeInteger(a));
17 a=Math.pow(2,53);
18 console.log(Number.isSafeInteger(a));
19 //new constants
20 console.log(Number.EPSILON);
21 console.log(Number.MAX_SAFE_INTEGER);
22 console.log(Number.MIN_SAFE_INTEGER);
```

true  
true  
true  
false  
true  
false  
false  
false  
false  
false  
true  
true  
false  
2.220446049250313e-16  
9007199254740991  
-9007199254740991





# Math Extensions

## Hyperbolic Functions

Cosh()

Acosh()

Sinh()

Asinh()

Tanh()

Hypot()

## Arithmetic Functions

Cbrt() Cube root

Clz32() Count leading zeros (32 bit integer)

Expm1 Equal to  $\exp(x) - 1$

Log2() Log base 2

Log10() Log base 10

Log1p() Equal to  $\log(x+1)$

Imul() 32 bit integer multiplication

**Sign()**

The number' sign: 1, -1, 0, -0, NaN

Trunc()

The integer part of a number

Fround()

Round to nearest 32 bit floating point





# Math Extensions

```
1 console.log(Math.sign(0));  
2 console.log(Math.sign(-0));  
3 console.log(Math.sign(-20));  
4 console.log(Math.sign(20));  
5 console.log(Math.sign(NaN));
```

0  
0  
-1  
1  
NaN





# Regular Expressions in ES 2015

```
1 //use u at the end of pattern to check for astral patterns
2 let pattern = /\u{1f3c4}/u;
3 console.log(pattern.test(``));
4
5 let newpattern = /^.Surfer/u;
6 console.log(newpattern.test('Surfer'));
7
8 //performing the search from the last index using 'y'
9 let numero = /900/y;
10 console.lof(numero.lastIndex);
11 numero.lastIndex=3;
12 console.log(numero.test('7123123900')));
13 let num=/900/yg;
14 console.log(pattern.flags);
```

```
false  
false
```





# Function Extensions

```
1 //function expressions
2
3 let fn = function employee(){
4   return "employee";
5 };
6 console.log(fn.name);
7 //anonymous function
8 let fnone = function (){
9   return "book";
10};
11 console.log(fnone.name);
12 let fntwo = fnone;
13 console.log(fntwo.name);
```

```
employee
fnone
fnone
```

```
1 //class expressions
2
3 class Employee{
4   constructor(){
5   }
6   add(){
7   }
8 }
9
10 let sak = new Employee();
11 console.log(Employee.name);
12 console.log(sak.add.name);
```

```
Employee
add
```





# Iterators in ES2015

- Iterators are used to iterate through a list of collection.

```
1 //iterators
2
3 let names = ['syed awase', 'syed ameese', 'syed azeez'];
4 console.log(typeof names[Symbol.iterator]);
5 let iter = names[Symbol.iterator]();
6 iter.next();
7 console.log(iter.next());
8
9 //iterator using function and spread operator
10 function process(name1,name2,name3){
11   console.log(name3);
12 }
13 process(...names);
```

```
function
[object Object]
syed azeez
```





# Generators in ES2015

- A generator function is a special type of function that when invoked automatically generates a special iterator called a **generator**
- Generator functions are indicated by ***function\**** and make use of the **yield** operator to indicate the value to return for each successive calls to `.next()` on the generator.

```
1 function *process(){
2   yield 8000;
3   yield 8001;
4 }
5 let val = process();
6 console.log(val);
7 console.log(val.next());
8 val.next();
9 console.log(val.next());
10 //generator with a loop
11 function *newprocess(){
12   let nextId = 9000;
13   while(true)
14     yield(nextId++);
15 }
16 for(let id of newprocess()){
17   if(id > 9010)break;
18   console.log(id);
19 }
```

```
[object Generator]
function (arg) { return this._invoke(method, arg); }
[object Object]
9000
9001
9002
9003
9004
9005
9006
9007
9008
9009
9010
```





# Generators in ES2015

```
1 function* range(start, count) {
2     for (let delta = 0; delta < count; delta++) {
3         yield start + delta;
4     }
5 }
6
7 for (let highschoolyears of range(13, 7)) {
8     console.log(`High School Years start from ${highschoolyears}!`);
9 }
```

```
High School Years start from 13!
High School Years start from 14!
High School Years start from 15!
High School Years start from 16!
High School Years start from 17!
High School Years start from 18!
High School Years start from 19!
```





# throw and return

```
1 function* valueProcess(){
2     try{
3         yield 1001;
4         yield 1002;
5         yield 1003;
6         yield 1004;
7         yield 1005;
8     }catch(e){
9     }
10 }
11
12 let it = valueProcess();
13 console.log(it.next().value);
14 console.log(it.throw('error message'));
15 console.log(it.next());
16 console.log(it.next().value);
17 //only works in firefox
18 console.log(it.return('error message'));
```

```
1001
[object Object]
[object Object]
[object Object]
```





# Promises

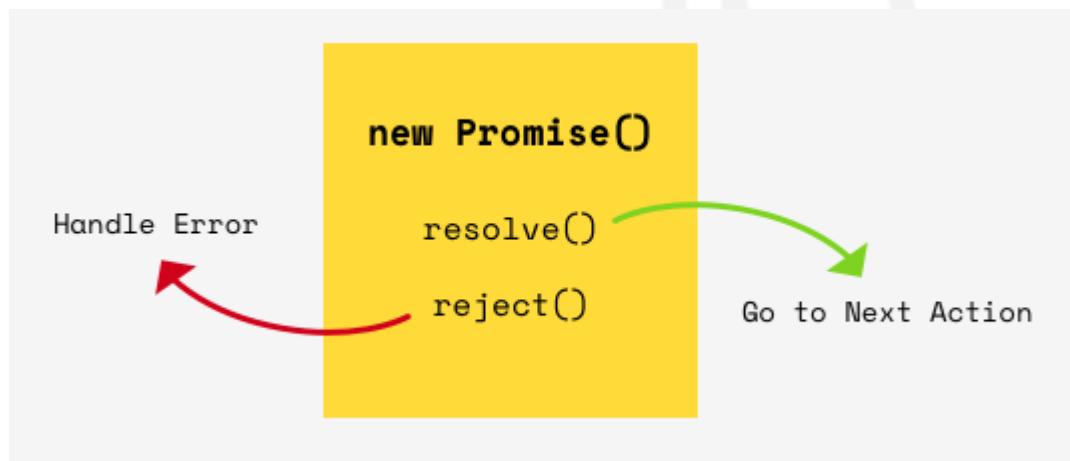
- A Promise specifies some code to be executed later( as with events and callbacks and also explicitly indicates whether the code succeeded or failed at its job).
- You can chain promises together based on success or failure in ways that make your code easier to understand and debug.
- JavaScript engines can only execute one piece of code at a time, so they need to keep track of code that meant to run.
- ES6 has native support for promises. In ES5, we used to achieve this using jquery or \$q libraries.





# Promises in ES2015

- used for deferred and asynchronous computations. A Promise represents an operation that hasn't completed yet, but is expected in the future.

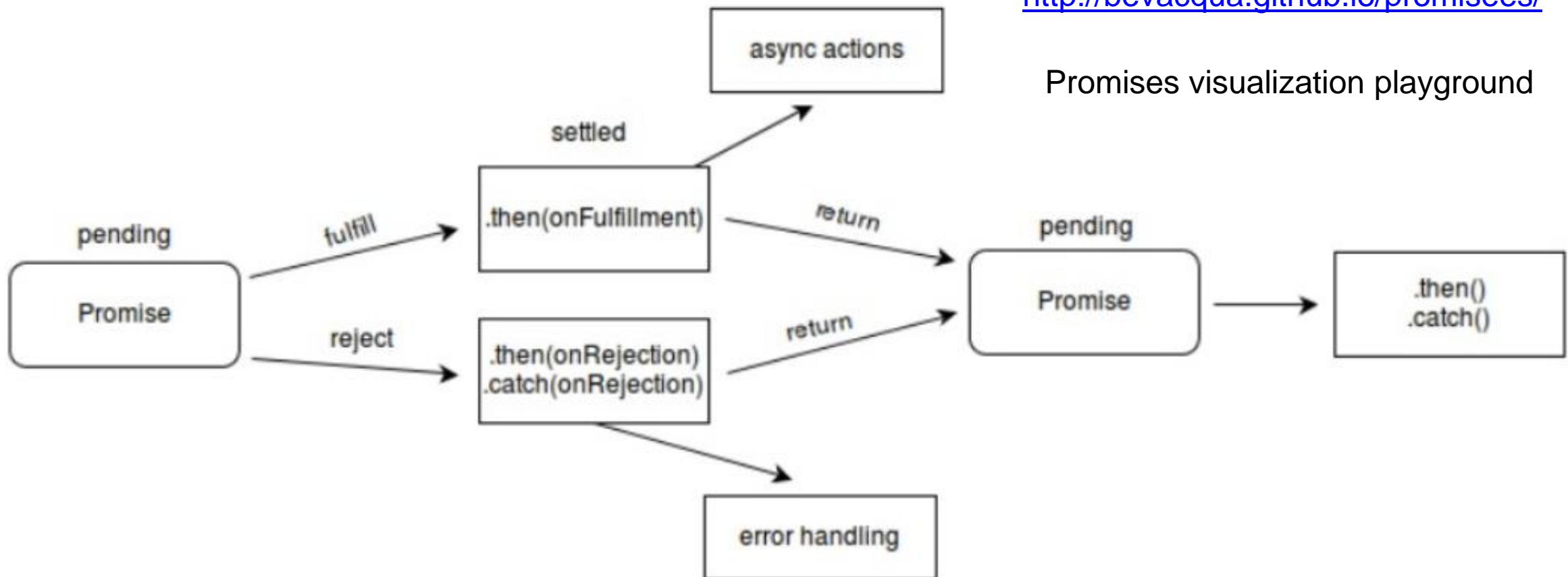


- Promise has **three** states
  - Pending : initial state, not fulfilled or rejected.
  - Fulfilled: meaning that the operation completed successfully
  - Rejected: meaning that the operation failed.





# Promise life cycle in ES2015



<http://bevacqua.github.io/promisees/>

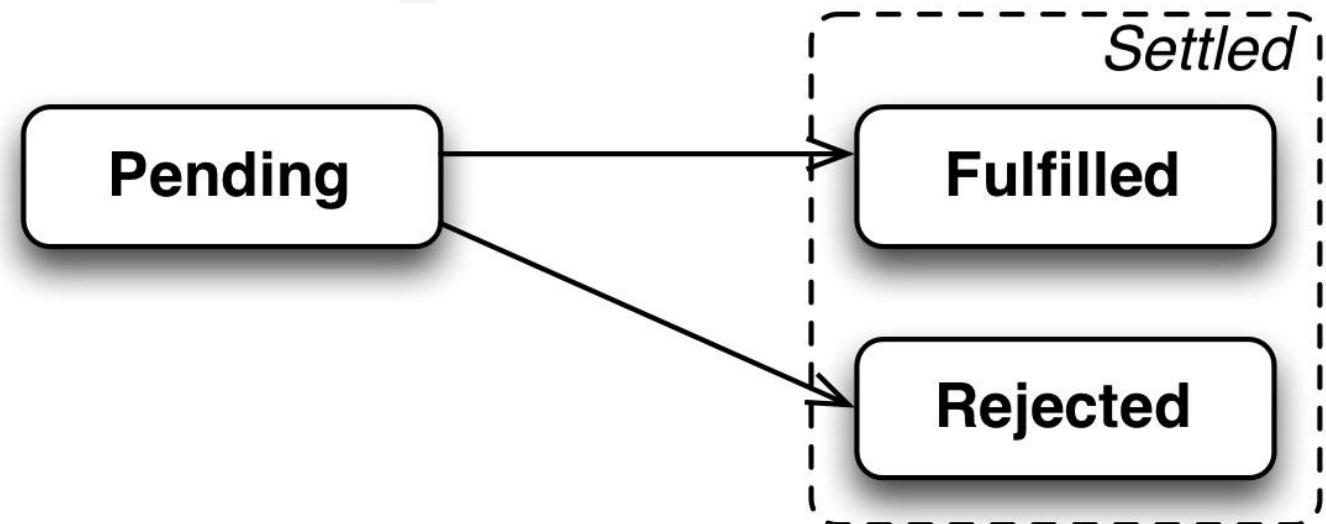
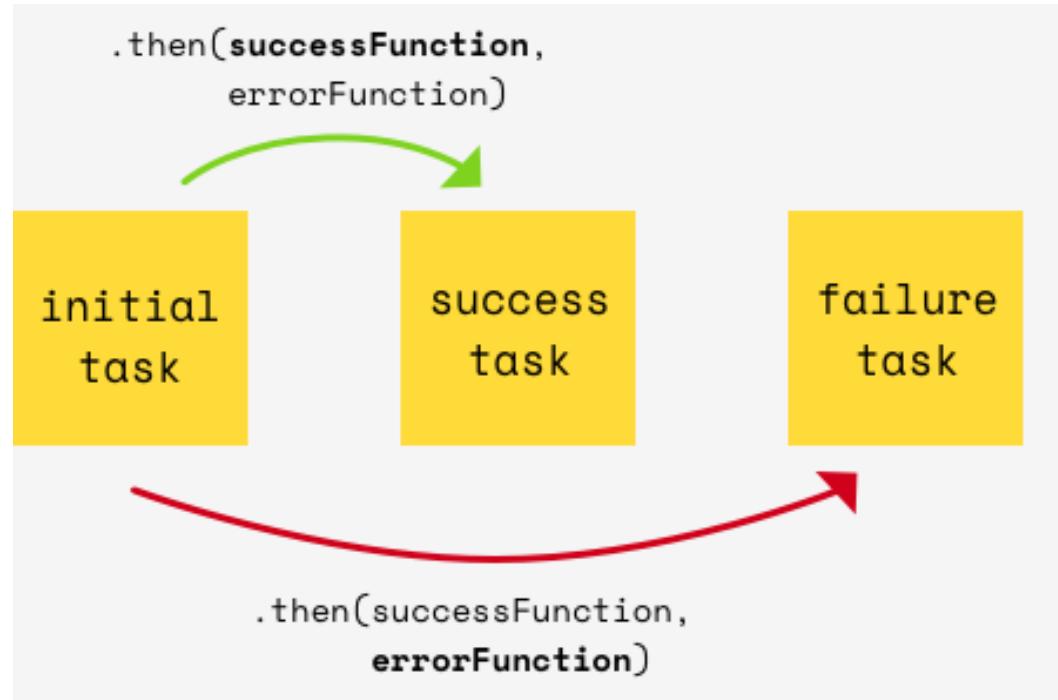
Promises visualization playground

<https://ponyfoo.com/articles/es6-promises-in-depth>





# Promises in ES2015





# Creating a Promise

```
var p = new Promise(function(resolve,reject){  
  if(/* condition* */){  
    resolve(/* value */); //fulfilled successfully  
  }  
  else{  
    reject(/* reason */); //error, rejected  
  }  
});|
```





# Promise

```
// A Promise that throws, rather than explicitly reject
var p1 = new Promise((resolve, reject) => {
  if (true)
    throw new Error("rejected!"); // same as rejection
  else
    resolve(4);
});

// trailing .catch() handles rejection
p1.then((val) => val + 2)
  .then((val) => console.log("got", val))
  .catch((err) => console.log("error: ", err.message));
// => error: rejected!
```





# Promise

```
new Promise(function(resolve, reject) {
  // A mock async action using setTimeout
  setTimeout(function() { resolve(10); }, 3000);
})
.then(function(num) { console.log('first then: ', num); return num * 2; })
.then(function(num) { console.log('second then: ', num); return num * 2; })
.then(function(num) { console.log('last then: ', num);});

// From the console:
// first then: 10
// second then: 20
// last then: 40
```

```
first then: 10
second then: 20
last then: 40
```





```
const all = (...promises) => {
  const results = [];

  const merged = promises.reduce(
    (acc, p) => acc.then(() => p).then(r => results.push(r)),
    Promise.resolve(null));

  return merged.then(() => results);
};

const p1 = Promise.resolve('Game of Throne: Dragon Queen is Promised the 7 kingdoms by birth right');
const p2 = '----- 2---Game of Throne: Ramzi Bolton-Bastard Son is Promised the Warden of North';
const p3 = new Promise((res, rej) => { setTimeout(res, 100, `-----3---Game of Throne: John Stark
- Fights for justice for his family and regains the title as King in the North` )});

all(p1, p2, p3)
  .then(([a1, a2, a3]) => console.log(a1, a2, a3));

const p4 = Promise.reject('---4---Game of Throne: High Sparrow is burnt alive in public');

all(p1, p2, p3, p4)
  .then(([a1, a2, a3]) => console.log(a1, a2, a3))
  .catch(e => console.log(e));
```





# Promise : as a wrapper

```
1 // Creating a promise wrapper for setTimeout
2 function wait(delay = 0) {
3     return new Promise((resolve, reject) => {
4         setTimeout(resolve, delay);
5     });
6 }
7
8 // Using a promise
9 wait(3000)
10    .then(() => {
11        console.log('3 seconds have passed!');
12        return wait(2000);
13    })
14    .then(() => {
15        console.log('5 seconds have passed!');
16        x++; // ReferenceError triggers `catch`
17    })
18    .catch(error => {
19        // output: ReferenceError
20        console.log(error);
21    })
22    .then(() => {
23        // simulate `finally` clause
24        console.log('clean up');
25    });
}
```

```
3 seconds have passed!
5 seconds have passed!
ReferenceError: x is not defined
clean up
```





# Promise.all

- There are times when we trigger **multiple async interactions but only want to respond when all of them are completed.**
- **Promise.all method takes an array of promises and fires one callback once they are ALL resolved.**

```
Promise.all([promise1, promise2]).then(function(results) {  
  // Both promises resolved  
})  
.catch(function(error) {  
  // One or more promises was rejected  
});
```





# Promises Example

```
const promisedTexts= "Game of Thrones: I am promised to be the King in the North";  
  
Promise.all(promisedTexts)  
.then(texts => {  
    for (const text of texts) {  
        console.log(text);  
    }  
})  
.catch(reason => {  
    console.log("Error to read the promised scroll");  
});
```





# Promise.race

- Instead of waiting for all promises to be resolved or rejected, **Promise.race** triggers as soon as any promise in the array is **resolved or rejected**.

```
var req1 = new Promise(function(resolve, reject) {
  // A mock async action using setTimeout
  setTimeout(function() { resolve('I am the first Born'); }, 8000);
});
var req2 = new Promise(function(resolve, reject) {
  // A mock async action using setTimeout
  setTimeout(function() { resolve('I am the second Born'); }, 3000);
});
Promise.race([req1, req2]).then(function(one) {
  console.log('Then: ', one);
}).catch(function(one, two) {
  console.log('Catch: ', one);
});
```

Then: I am the second Born





# ES2015 Promises

1. Promises give us the ability to write asynchronous code in a synchronous fashion, with flat indentation and a single exception channel.
2. Help us unify asynchronous APIs and allow us to wrap non-spec compliant Promise APIs or callback APIs with real Promises.
3. Promises give us guarantees of no race conditions and immutability of future value represented by the Promise.
4. Promises cannot be cancelled, once created it will begin execution, if you don't handle rejections or exceptions, they get swallowed.
5. No way to determine the state of a Promise
6. For Recurring values or events, we can use streams





## Callbacks

1. Callbacks are functions.
2. Just blocks of code which can be run in response to events such as timers going off or messages being received from the server. Any function can be a callback, and every callback is a function
3. They are defined independently of the functions they are called from, they are passed in as arguments. These functions then store the callback and call it when the event actually happens.

## Promises

1. Promises are objects.
2. They are objects which store information about whether or not those events have happened yet, and if they have, what their outcome is.
3. They are created inside of asynchronous functions ( those which might not return a response until later) and then returned. When an event happens, the asynchronous function will update the promise to notify the outside world.





## Callbacks

4. They can be called multiple times by the functions they are passed to.

## Promises

4. They can only represent one event - they are either successful once or failed once

© TPRI





# Array Extensions

- **Array.of**
  - is exactly an incarnation of Array Method, similar to ES5.

```
//es5 arrays strange behaviour
let val= Array(100);
console.log(val.length);
//es6 array behaviour as it should be
let es6val = Array.of(100);
console.log(es6val.length);
```

100  
1

- **Array.from**
  - It has 3 arguments, but only the *input is required*.
    - *Input – the arraylike or iterable object you want to cast*
    - *Map – a mapping function that's executed on every item of input*
    - *context – this binding to use when called map.*

```
let scores = [657,785,857,924];
let subjectScore = Array.from(scores, v=>v+20);
console.log(subjectScore);
```

677,805,877,944





# Array Extensions

## Fill

```
//fill method for an array
let taxVal =[875,895,9125,1754];
taxVal.fill(19000);
console.log(taxVal);
//start filling from the arra[2]
taxVal.fill(22000,2);
console.log(taxVal);
//array filling with start and stop
taxVal.fill(25000, 1,2);
console.log(taxVal);
// start counting from the end of the array
taxVal.fill(35000, -1);
console.log(taxVal);
```

## Find

```
19000,19000,19000,19000
19000,19000,22000,22000
19000,25000,22000,22000
19000,25000,22000,35000
```

- Only returns the first value it finds in the array

```
//find method to search for values in an array
let taxVal =[875,895,9125,1754];
let result = taxVal.find(value=>value >= 9000);
console.log(result);
```

9125



# Array Extensions

```
1 //find method to search for values in an array
2 let taxVal =[875,895,9125,1754];
3 let result = taxVal.find(value=>value >= 9000);
4 console.log(result);
5
6 //findIndex method
7 let searchResult= taxVal.findIndex(function(value
8   return value==this;
9 },9125);
10 console.log(searchResult);|
```

9125  
2

```
//find method to search for values in an array
let taxVal =[875,895,9125,1754];
//copyWithin(destinationPoint, sourceIndex)
taxVal.copyWithin(2,0);
console.log(taxVal);
taxVal.copyWithin(1,3);
console.log(taxVal);|
```

875,895,875,895  
875,895,875,895





# Array Extensions

```
let numeroUno=[11,12,13,14,15,16,17,18,19];
//copyWithin(destPt,srcIndex, HowManyvaluetoCopy)
numeroUno.copyWithin(3,0,4);
console.log(numeroUno);
```

11,12,13,11,12,13,14,18,19

```
let grades=['A','B','C','D','E','F'];
console.log(...grades.entries());
console.log(...grades.keys());
console.log(...grades.values());
```

0,A 1,B 2,C 3,D 4,E 5,F

0 1 2 3 4 5

A B C D E F





# ArrayBuffers

```
let buffer = new ArrayBuffer(1024);
console.log(buffer.byteLength);
buffer[0]=0xff;//hexadecimal
//[] bracket notation can be used with ArrayBuffer
console.log(buffer[0]);|
```

1024

255





# Typed Arrays

1. Int8Array()
2. Uint8Array()
3. Uint8ClampedArray()
4. Int32Array()
5. Uint32Array()
6. Int16Array()
7. Uint16Array()
8. Float32Array()
9. Float64Array()





# Typed Arrays with ArrayBuffer

```
//typed array and array buffer
let buffer = new ArrayBuffer(1024);
console.log(buffer.byteLength);
buffer[0]=0xff;//hexadecimal
//[] bracket notation can be used with ArrayBuffer
console.log(buffer[0]);
//
let buffertwo = new ArrayBuffer(1024);
let a = new Int8Array(buffertwo);
a[0]=0xff; //hexadecimal value
console.log(a[0]);
let b = new Uint8Array(buffertwo);
b[0] = 0xff;
console.log(b[0]);
let c=new Uint8ClampedArray(buffertwo);
c[0] = -15;
console.log(c[0]);
//
let bufferthree = new ArrayBuffer(1024);
let d= new Uint8Array(bufferthree);
let e=new Uint16Array(bufferthree);
d[0]=1;
console.log(e[0]);
```





# Map

- Map is a key/value data structure in ES6. It provides a better data structure to be used for hash-maps.

```
//map example in ES2015
let student1 ={name:"Imad"};
let student2={name:"satnam"};
let students = new Map();
students.set(student1, 'DE12312312');
students.set(student2, 'IN1231232');
console.log(students.get(student1));
console.log(students.get(student2));
// get the size of the map
console.log(students.size);
//delete an entry from the map
students.delete(student2);
console.log(students.size);
//clears out the entire map
students.clear();
console.log(students.size);
let studarr=[
  [student1,'DE12312312'],
  [student2, 'IN1231232']
];
let studentslist = new Map(studarr);
// to check whether a specific key exists or not
console.log(studentslist.has(student2));
let slist = [...studentslist.values()];
console.log(slist);
let smlist = [...studentslist.entries()];
console.log(smlist[0][1]);
```





# WeakMap

1. A WeakMap is a subset of Map, which are not **iterable**
2. Every key must be an object and value types are not admitted as keys.
3. WeakMap enables map keys to be garbage collected when they are only being referenced as WeakMap Keys.





# ES2015 Sets

- Sets are yet another collection type in ES6. Sets are very similar to Map.
- Set is also iterable
- They deal with single value or single objects. The purpose of a set is to guarantee a uniqueness.
- Set constructor also accepts an iterable
- Set also has a .size property
- Keys can also be arbitrary values
- Keys must be unique
- NaN equals NaN when it comes to Set





```
1 let PMsOfIndia = new Set();
2 PMsOfIndia.add("Nehru");
3 PMsOfIndia.add("Indira");
4 PMsOfIndia.add("ManMohanSingh");
5 PMsOfIndia.add("PVNRao");
6 console.log(PMsOfIndia);
7 console.log(PMsOfIndia.size);
8 console.log(PMsOfIndia.values());
9 console.log(PMsOfIndia.keys());
10
11 let PresidentsOfIndia = new Set([
12   `Gyani Zail Singh`, `Abdul Kalam`, `Rajendra Prasad`
13 console.log(PresidentsOfIndia.has('Abdul Kalam'));
14 console.log(PresidentsOfIndia.size);
15 console.log(...PresidentsOfIndia.keys());
16 console.log(...PresidentsOfIndia.values());
17 console.log(...PresidentsOfIndia.entries());
18
19 let myval=new Set([11,'11']);
20 console.log(myval.size);
```

[object Set]

4

[object Set Iterator]

[object Set Iterator]

true

3

Gyani Zail Singh Abdul Kalam Rajendra Prasad

Gyani Zail Singh Abdul Kalam Rajendra Prasad

Gyani Zail Singh,Gyani Zail Singh Abdul Kalam,Abdul  
Kalam Rajendra Prasad,Rajendra Prasad

2





# Subclassing Builtins

- we would like to write extensions to Javascript language builtins.
- The builtin data structures add a huge amount of power to the language, and being able to create new types that leverage that power is amazingly useful

```
//subclassing builtins
class SumArray extends Array{
  sum(){
    let total = 0;
    this.map(v=> total+=v);
    return total;
  }
  let total = SumArray.from([12,13,4212,531,532]);
  console.log(SumArray instanceof Array);
  console.log(SumArray.length);

  let reverseVal = SumArray.reverse();
  console.log(reverseVal instanceof SumArray);
  console.log(reverseVal instanceof Array);
  console.log(SumArray.sum());
```





# ES2015 Reflection API

- Object reflection is a language ability to be able to inspect and manipulate object properties at runtime.
- JS has been supporting APIs for object reflection but these APIs were not organized under a namespace and also they threw exception when they fail to complete an operation.
- ES2015 introduces a new object referred as **Reflect** which exposes methods for object reflection.
- These new methods don't throw exception on failure rather they return error, which makes it easy to write code involving object reflection.
- Reflect object cannot be used with **new operator as it is not a constructor**





```
console.log(typeof Reflect); => Object
```

## Reflect.construct(target, argumentsList[,newTarget]);

- Not a function Object
- Does not have [[construct]] internal method (can't use new)
- Does not have a [[call]] internal method (can't invoke it as a function)
- Check for browser implementations





# Reflect.construct

The [new operator](#) as a function.

```
class Hotel{
    constructor(name, city, rating){
        console.log(` ${name} in ${city} has ${rating}`);
    }
}
function HotelEvaluation(){
    console.log('Evaluate the Quality Standards');
}
let h = Reflect.construct(Hotel, ["Le Meridian", "Bangalore", 8], HotelEvaluation);
Reflect.getPrototypeOf(h);
```





# Reflect.apply

## **Reflect.apply(target, thisArgument, argumentsList)**

*Calls a target function with arguments as specified by the args parameter*

```
class Employee{  
    constructor(){  
        this.empId =7860;  
    }  
    show(){  
        console.log(this.empId);  
    }  
}  
Reflect.apply(Employee.prototype.show, {empId: 7281});  
Reflect.apply(Employee.prototype.show, {empId: 7281}, ['SYCLIQ:']);  
//another example  
Reflect.apply(String.fromCharCode, undefined, [104, 101, 108, 108, 111]);
```





# Reflect and Prototypes

## Reflect.getPrototypeOf(target)

- It is the same method as Object.getPrototypeOf().
- It returns the prototype of the specified object

```
class LatLong{  
    constructor(){  
        console.log('Stamping out Coordinates');  
    }  
}  
class HotelLocation extends Location{  
}  
console.log(Reflect.getPrototypeOf(HotelLocation));
```





# Reflect and Prototypes

`Reflect.setPrototypeOf(target, prototype)`

- It sets the prototype of a specified object to another object or to null.

```
class LatLong{
  constructor(){
    console.log('Stamping out Coordinates');
  }
}
class HotelLocation extends Location{

}
let setup={
  getId(){return 101;}
}
console.log(Reflect.getPrototypeOf(HotelLocation));
let th = new HotelLocation();
Reflect.setPrototypeOf(th,setup);
console.log(th.getId());
```



# Reflect and Properties

`Reflect.get(target, propertyKey[, receiver])`

- The static Reflect.get() method works like getting a property from an object.
- It returns the value of the property.
- It allows you to get a property on an object.

```
class Restaurant{  
  constructor(){  
    this.id=9999;  
  }  
}  
let resty = new Restaurant();  
console.log(Reflect.get(resty,"id"));
```





# Reflect and Properties

`Reflect.set(target, propertyKey, value[, receiver])`

- The static Reflect.set() method works like setting a property on an object.
- It allows you to set a property on an object. It does property assignment and is like the property accessor syntax as a function.

```
class Motel{  
  constructor(){  
    this.id =1111;  
  }  
}  
let m = new Motel();  
Reflect.set(m,'id', 7777);  
console.log(m.id);
```





# Reflect and Properties

`Reflect.has(target, propertyKey)`

- The static **Reflect.has()** method works like the **in operator as a function**.
- It allows you to check if a property is in an object.

```
class LandMark{  
    constructor(){  
        this.city = 'Bangalore';  
    }  
}  
class POI extends LandMark{  
    constructor(){  
        super();  
        this.lat = 12.77;  
        this.long = 77.77;  
    }  
}  
let p = new POI();  
console.log(Reflect.has(p, 'city'));  
console.log(Reflect.has(p, 'lat'));  
console.log(Reflect.has(p, 'long'));
```





# Reflect and Properties

## Reflect.ownKeys(target)

- The **static Reflect.ownKeys()** method returns an array of the target Object's own property keys.
- Its return value is equivalent to `Object.getOwnPropertyNames(target).concat(Object.getOwnPropertySymbols(target))`.

```
class LandMark{  
    constructor(){  
        this.city = 'Bangalore';  
    }  
}  
class POI extends LandMark{  
    constructor(){  
        super();  
        this.lat = 12.77;  
        this.long = 77.77;  
    }  
}  
let p = new POI();  
console.log(Reflect.has(p,'city'));  
console.log(Reflect.has(p, 'lat'));  
console.log(Reflect.has(p, 'long'));  
console.log(Reflect.ownKeys(p));
```





# Reflect and Properties

`Reflect.defineProperty(target, propertyKey, attributes)`

- The **static** **Reflect.defineProperty()** method is like `Object.defineProperty()`, but returns a Boolean.
- It allows precise addition to or modification of a property on an object.

```
class Store{  
}  
let st = new Store();  
Reflect.defineProperty(st, 'id', {  
    name:'Walmart',  
    configurable:true,  
    enumerable:true  
});  
console.log(st['name']);
```





# Reflect and Properties

`Reflect.deleteProperty(target, propertyKey)`

- The static `Reflect.deleteProperty()` method allows to delete properties.
- It returns a Boolean indicating whether or not the property was successfully deleted.

```
let School= {  
  sid: 1298012,  
  name: 'St.louis'  
};  
console.log(School.sid);  
Reflect.deleteProperty(School, 'sid');  
console.log(School.sid);
```





# Reflect and Properties

## `Reflect.getOwnPropertyDescriptor(target, propertyKey)`

- The **static `Reflect.getOwnPropertyDescriptor()` method is similar to `Object.getOwnPropertyDescriptor()`.**
- it returns a property descriptor for the given property if it exists on the object, `undefined` otherwise.

```
let School= {
  sid: 1298012,
  name: 'St.louis'
};
console.log(School.sid);
Reflect.deleteProperty(School, 'sid');
console.log(School.sid);
let d = Reflect.getOwnPropertyDescriptor(School, 'name');
console.log(d);
```





# Reflect and Property Extensions

`Reflect.preventExtensions(target)`

- The **static** `Reflect.preventExtensions()` **method** prevents new properties from ever being added to an object.
- Prevents future extensions of the object.
- Similar to `Object.preventExtensions()`

```
let Person={  
    ssn= 1235412351243  
};  
Person.location="Boston";  
console.log(Person.location);  
Reflect.preventExtensions(Person);  
Person.name="Arijit";  
console.log(Person.name);
```





# Reflect and Property Extensions

## Reflect.isExtensible(target)

- The static Reflect.isExtensible() method determines if an object is extensible ( whether it can have new properties added to it).

```
let Auto={  
  type="three wheeler",  
  make="bajaj"  
};  
console.log(Reflect.isExtensible(Auto));  
Reflect.preventExtenstion(Auto);  
console.log(Reflect.isExtensible(Auto));
```





# ES2015 Proxies : Proxy API

- Proxy is used to determine behavior whenever the properties of a **target** object are accessed.
- **A handler object can be used to configure traps for your Proxy.**
- Used for security, profiling, logging
- New feature in ES2015
- The Proxy object is used to define custom behavior for fundamental operations (e.g. property lookup, assignment, enumeration, function invocation, etc)
- Please check with Kangax for browser compatible implementations.





# Proxy Terminology

Wrapper

## Handler Object (the Proxy)

**TRAPS**

Get()

Set()

Apply()

Target Object  
Or  
Function





# Get Proxy

- A trap for getting a property value
- This trap can intercept these operations
  - Property access
  - Inherited property access
  - Reflect.get()

```
function Store(){  
  
    this.name="WalMart";  
    this.location="Boston";  
    this.MonthlySales = "40000000";  
};  
var ws = new Store();  
var pws = new Proxy(ws, {  
    get:function(target, prop, receiver){  
        return "Successfully Accessed"+ prop;  
    }  
});  
console.log(pws.MonthlySales);
```

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Proxy/handler/get](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/get)





# Get Proxy

```
function Store(){
    this.name="WalMart";
    this.location="Boston";
    this.MonthlySales = "40000000";
};

var ws = new Store();
var pws = new Proxy(ws, {
    get:function(target, prop, receiver){
        return "Successfully Accessed"+ prop;
    }
});
var px = new Proxy(ws,{ 
    get:function(target,prop,receiver){
        return Reflect.get(target, prop, receiver);
    }
})
console.log(pws.MonthlySales);
console.log(px.location);
```





Contact Us

[sak@sycliq.com](mailto:sak@sycliq.com)

[sak@territorialprescience.com](mailto:sak@territorialprescience.com)

For code driven trainings for Technology Firms

Reach out to us +91-9035433124

We are hardcore Technologists/Architects/Programmers  
trainings are offered by Dr. Syed Awase

# Thank You

We also provide Code Driven Open House Trainings

INDIA

HYDERABAD | BANGALORE | CHENNAI | PUNE

OVERSEAS

SINGAPORE | MALAYSIA | DUBAI | EUROPE | NORTH  
AMERICA



## Current Offerings

- AngularJS 1.5.x
- Typescript /CoffeeScript /Dart/
- D3.JS/NVD3, HighCharts
- AngularJS 2 (with NodeJS)
- KnockOutJS (with NodeJS)
- BackBoneJS (with NodeJS)
- Ember JS / Ext JS (with NodeJS)
- Raspberry Pi
- Responsive Web Design with Bootstrap, Google Material Design and KendoUI, Zurb Foundation
- C# ASP.NET MVC , WEB API, WPF
- JAVA , SPRING, HIBERNATE
- Python , Django
- R Statistical Programming
- Android Programming
- Python/Django
- Ruby on Rails
- NoSQL (MongoDB – Essentials & Advanced )
- GIS Tools, SAFE FME, ArcGIS

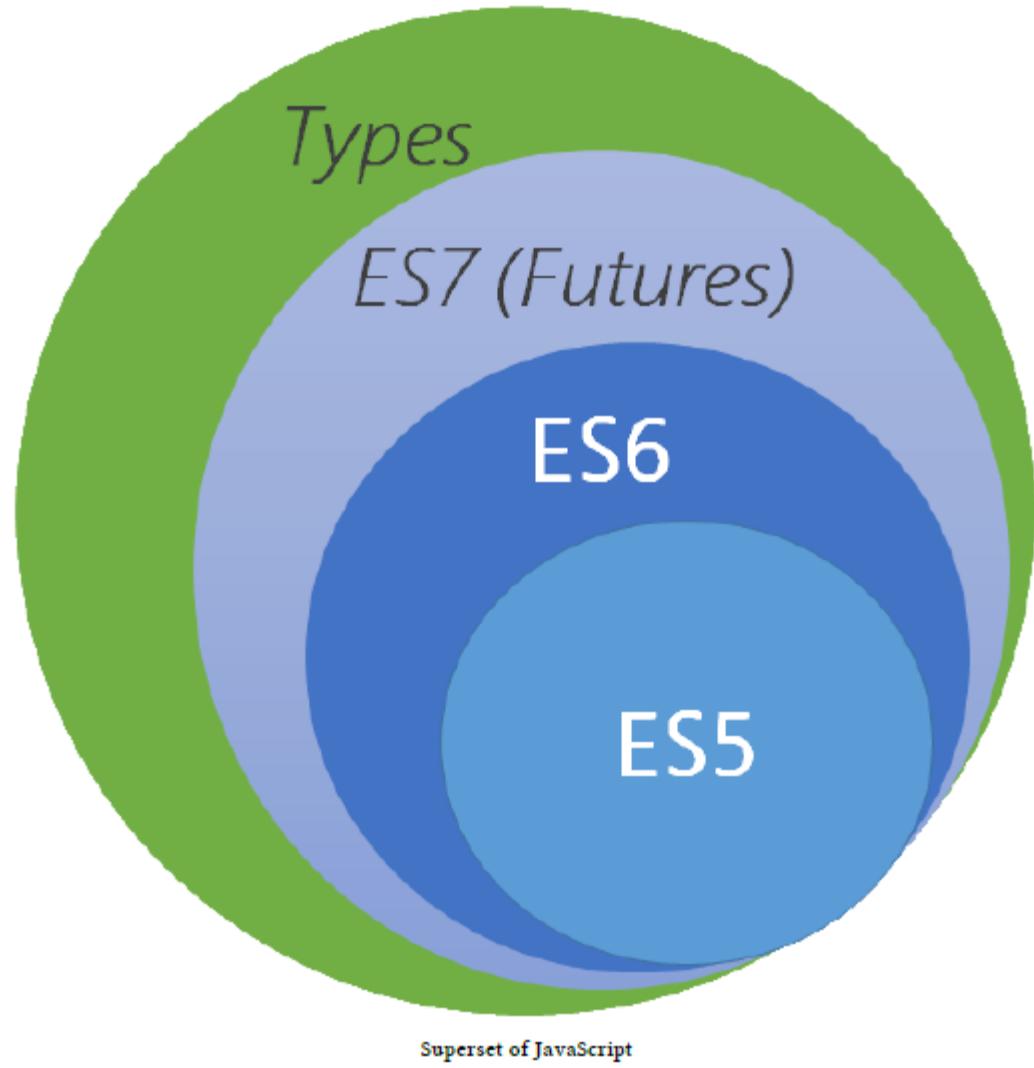
Developed by Microsoft and Apache 2 License

# TypeScript

Syed Awase Khirni

Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project ([www.geo-spirit.org](http://www.geo-spirit.org)). He currently provides consulting services through his startup [www.territorialprescience.com](http://www.territorialprescience.com) and [www.sycliq.com](http://www.sycliq.com)







## Static Type

- Type system
- Rigid
- Promotes stability and maintainability.

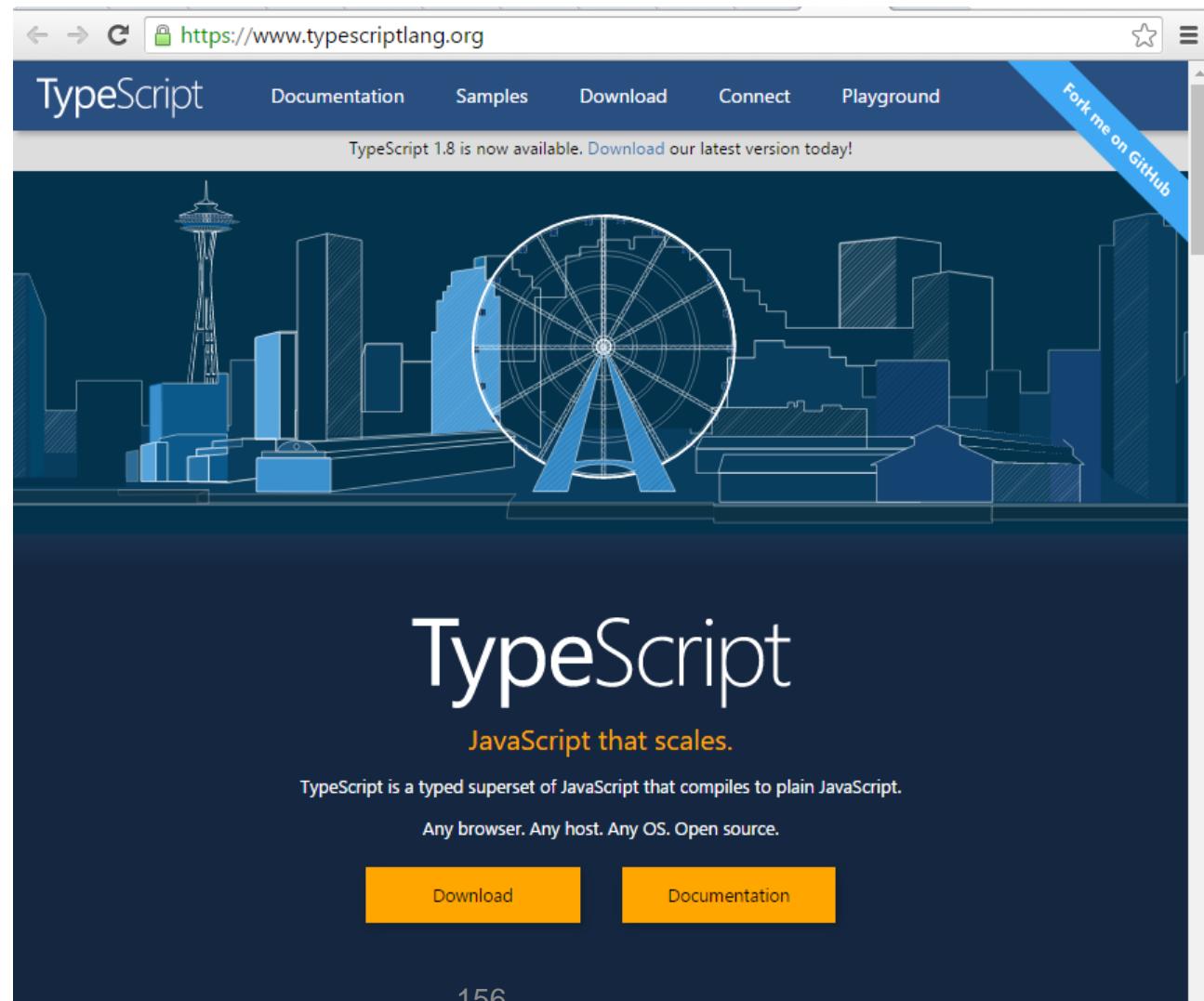
## Dynamic Type

- Type System
- Forgiving
- Great for web browser object model.





# <https://www.typescriptlang.org/>



The screenshot shows the official TypeScript website at https://www.typescriptlang.org/. The page features a blue header with the 'TypeScript' logo and navigation links for Documentation, Samples, Download, Connect, and Playground. A banner at the top right encourages users to 'Fork me on GitHub'. The main content area has a dark blue background with a stylized illustration of a city skyline featuring the Space Needle and a Ferris wheel. The word 'TypeScript' is prominently displayed in large white letters, followed by the tagline 'JavaScript that scales.' Below this, a paragraph explains that TypeScript is a typed superset of JavaScript that compiles to plain JavaScript, supporting 'Any browser. Any host. Any OS. Open source.' At the bottom, there are two orange buttons labeled 'Download' and 'Documentation'.





# Installing TypeScript

## Get TypeScript

### Node.js

The command-line TypeScript compiler can be installed as a Node.js package.

#### INSTALL

```
npm install -g typescript
```

#### COMPILE

```
tsc helloworld.ts
```

### Visual Studio



Visual Studio 2015



Visual Studio 2013



Visual Studio Code

### And More...



Sublime Text



Emacs



WebStorm



Eclipse



Vim





# Development Environment

<http://alm.tools/>

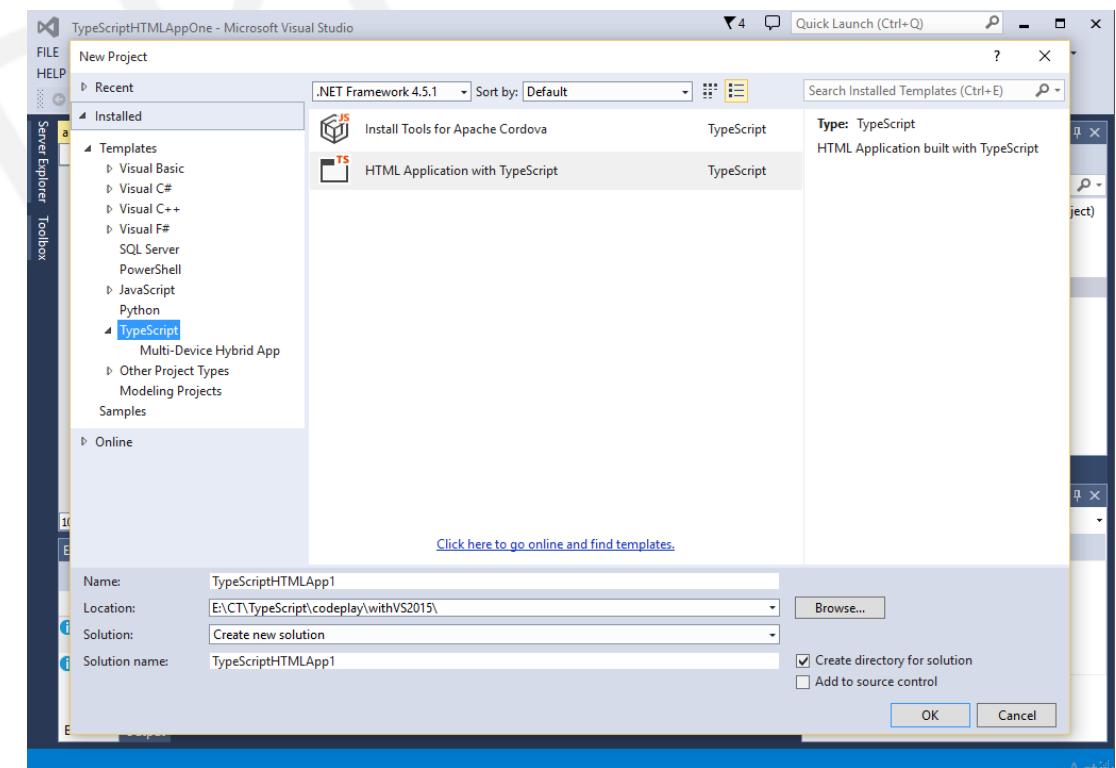
The *best* IDE for TypeScript should be a single npm install away.

Semantic releases start when ★ count > 1000. Don't forget to share ☺

```
bas.tsx
1 declare var React: any;
2 + import {foo} from './foo';
3 const bar = foo;
4 // This is a multi-line comment */
5 function test(a);
6 function test(a, b);
7 + function test(a, b, c) {
8 }
9
10 + let dom = <div>
11   </div>
```

```
PS E:\ct\TypeScript\codeplay> npm install -g alm
LoadRequestedDeps -> netw - |#####-----
```

Visual Studio 2015





# TypeScriptLang.org/play

## TypeScript 1.8 version

TypeScript is a **TRANSPILER**, which takes Your code and changes it, But it's still in the Same language that you started out with.

- Optional static typing and class-based object-oriented Programming, which is a strict superset of JS

The screenshot shows the TypeScript playground interface at <https://www.typescriptlang.org/play/>. The top navigation bar includes links for Documentation, Samples, Download, Connect, Playground (which is selected), and a GitHub fork button. A message at the top says "TypeScript 1.8 is now available. Download our latest version today!". The main area has tabs for "Using Classes", "TypeScript" (selected), and "Share". Below these are two code snippets. The left snippet is in TypeScript:1 class Greeter {  
2 greeting: string;  
3 constructor(message: string) {  
4 this.greeting = message;  
5 }  
6 greet() {  
7 return "Hello, " + this.greeting;  
8 }  
9 }  
10  
11 let greeter = new Greeter("world");  
12  
13 let button = document.createElement('butt  
14 button.textContent = "Say Hello";  
15 button.onclick = function() {  
16 alert(greeter.greet());  
17 }  
18  
19 document.body.appendChild(button);The right snippet is in JavaScript:1 var Greeter = (function () {  
2 function Greeter(message) {  
3 this.greeting = message;  
4 }  
5 Greeter.prototype.greet = function () {  
6 return "Hello, " + this.greeting;  
7 };  
8 return Greeter;  
9 }());  
10 var greeter = new Greeter("world");  
11 var button = document.createElement('butt  
12 button.textContent = "Say Hello";  
13 button.onclick = function () {  
14 alert(greeter.greet());  
15 };  
16 document.body.appendChild(button);  
17 }





# Why use TypeScript?

## Classic Javascript

- Functions are mostly spaghetti code in classic JavaScript unless we use JavaScript design patterns like iife, module pattern or revealing module pattern etc..
- JavaScript code encapsulation
- difficult to ensure proper types are passed without tests
- very few developers use “strict equality” (==>)
- Complex code for enterprise applications and difficult to maintain

## TypeScript

- It is a typed superset of JavaScript that compiles to plain JavaScript
- More structured and More Organized
- Takes care of AMD (Asynchronous Module Definition)
- Appeals Enterprise JavaScript Developers with a whole set of features
- NOT A totally separate JavaScript
- Cross-browser compatible, Works on Any host , Any OS, Open Source, Very Good Tool Support.





# Alternatives to TypeScript?

## CoffeeScript

[coffeescript.org](http://coffeescript.org)

**CoffeeScript** is a little language that compiles into JavaScript. Underneath that awkward Java-esque patina, JavaScript has always had a gorgeous heart. CoffeeScript is an attempt to expose the good parts of JavaScript in a simple way.

The golden rule of CoffeeScript is: *"It's just JavaScript"*. The code compiles one-to-one into the equivalent JS, and there is no interpretation at runtime. You can use any existing JavaScript library seamlessly from CoffeeScript (and vice-versa). The compiled output is readable and pretty-printed, will work in every JavaScript runtime, and tends to run as fast or faster than the equivalent handwritten JavaScript.

Latest Version: 1.10.0

<http://www.coffeescript.org>

## ECMA 6

<https://es6-features.org/#Constants>

**ECMAScript 6 — New Features: Overview & Comparison**

**Constants**

Support for constants (also known as "immutables"), not its assigned content (for instance, `const PI = 3.141593`).

**ECMAScript 6 — syntactic sugar: reduced | traditional**

```
const PI = 3.141593
PI > 3.0
```

**ECMAScript 5 — syntactic sugar: reduced | traditional**

```
// only in ESS through the help of a global context and Object.defineProperty()
// and only in global context and Object.defineProperty()
Object.defineProperty(typeof global,
  'PI', {
    value: 3.141593,
    enumerable: true,
    writable: false,
    configurable: false
})
```

## DART

<https://www.dartlang.org>

**Dart**

Dart is an application programming language that's easy to learn, easy to scale, and deployable everywhere.

Google depends on Dart to make very large apps.

**Get Started**

[ Click underlined text or code to learn more.]

```
import 'dart:async';
import 'dart:math' show Random;
main() async {
  print('Compute π using the Monte Carlo method.');
  await for (var estimate in computePi()) {
    print('n = ${estimate}');
  }
}

/// Generates a stream of increasingly accurate estimates of π.
Stream<double> computePi({int batch: 100000}) async* {
  var total = 0;
  var count = 0;
  while (true) {
    var points = generateRandom().take(batch);
    var inside = points.where((p) => p.isInsideUnitCircle);
    total += batch;
    count += inside.length;
    var ratio = count / total;
    // Area of a circle is A = πr², therefore n = A/r².
    // So, when given random points with x ∈ <0,1>,
}
```

<http://dartlang.org>

Alternatively you can write classical JavaScript with design patterns





# Benefits of TypeScript

- Helps in code structuring
- Uses class based object oriented programming
- Impose coding guidelines
- Offers type checking
- Compile time error checking
- Intellisense
- Optionally statically typed language (using **any** data type, we can assign any type of value to the variable by asking the)
- Components of TypeScript
  - Language
  - Compiler
  - Language Service
- Supports
  - Modules
  - Classes
  - Interfaces
  - Data types
  - Member functions





# TypeScript Features

- 
- Function support for lambdas (writing anonymous functions)
  - Define interfaces
  - Support for constructors, properties and functions
  - Encapsulation through classes and modules
  - Provides static typing
  - Supports standard JavaScript Code





# TypeScript Features

- Type Annotations
- Type inference
- Compile time type checking
- Optional, default and rest parameters
- Classes
- Interfaces
- Structural typing
- Arrow function expressions
- Enums
- Generics
- Modules
- Tuple Types
- Union Types and type guards





Select... TypeScript Share

```

1 class Greeter {
2   greeting: string;
3   constructor(message: string) {
4     this.greeting = message;
5   }
6   greet() {
7     return "Hello, " + this.greeting;
8   }
9 }
10
11 let greeter = new Greeter("world");
12
13 let button = document.createElement('button');
14 button.textContent = "Say Hello";
15 button.onclick = function() {
16   alert(greeter.greet());
17 }
18
19 document.body.appendChild(button);

```

## Static Typing

## Encapsulation

## IIFE

Run JavaScript

```

1 var Greeter = (function () {
2   function Greeter(message) {
3     this.greeting = message;
4   }
5   Greeter.prototype.greet = function () {
6     return "Hello, " + this.greeting;
7   };
8   return Greeter;
9 })();
10 var greeter = new Greeter("world");
11 var button = document.createElement('button');
12 button.textContent = "Say Hello";
13 button.onclick = function () {
14   alert(greeter.greet());
15 };
16 document.body.appendChild(button);
17

```

E:\ct\TypeScript\codeplay\withvs2015> tsc greeter.ts





# Browser Compatibility

[caniuse.com](https://caniuse.com)

About News May 12, 2015 - Feature suggestions now handle... Compare browsers Index

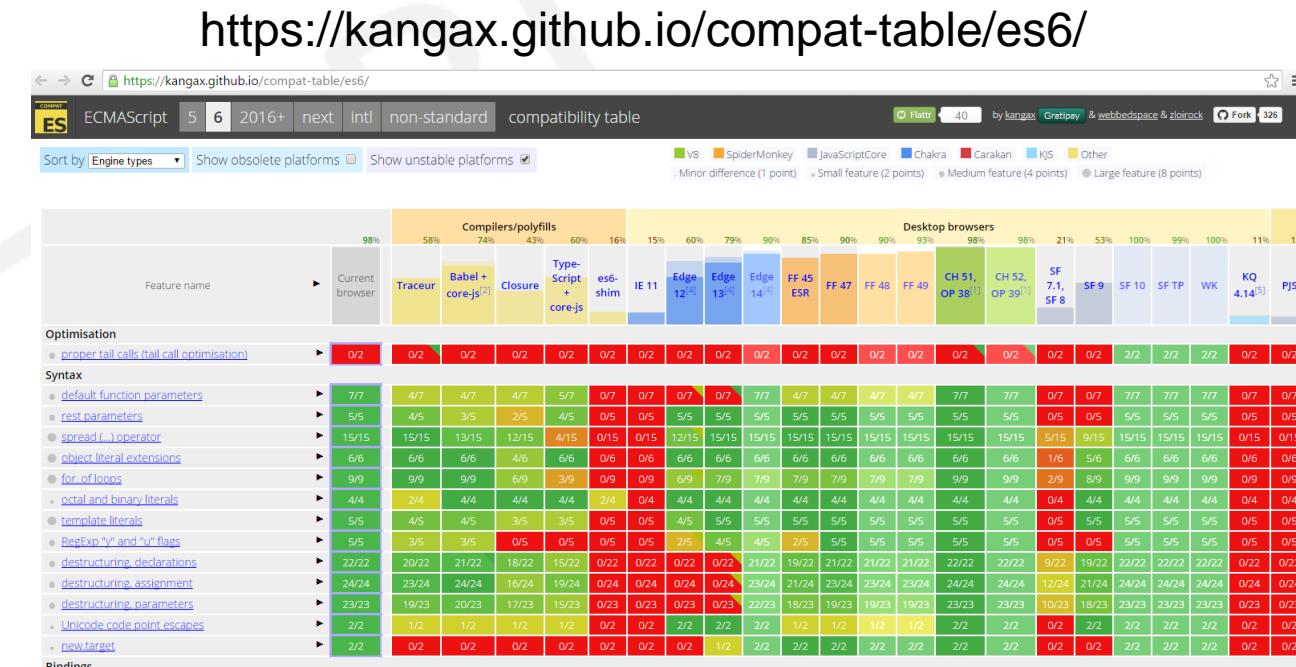
Ads by Google HTML5 CSS3 → HTML5 Browser → HTML5 JavaScript → HTML5 Web Storage →

Can I use ? Settings

CSS	HTML5	SVG
▪ ::first-letter CSS pseudo-element selector	▪ accept attribute for file input	▪ Inline SVG in HTML5
▪ ::placeholder CSS pseudo-element	▪ Audio element	▪ SVG (basic support)
▪ ::selection CSS pseudo-element	▪ Audio Tracks	▪ SVG effects for HTML
▪ ::dir() CSS pseudo-class	▪ Autofocus attribute	▪ SVG favicons
▪ :in-range and :out-of-range CSS pseudo-classes	▪ Canvas (basic support)	▪ SVG filters
▪ :matches() CSS pseudo-class	▪ Canvas blend modes	▪ SVG fragment identifiers
▪ @font-face Web fonts	▪ classList (DOMTokenList)	▪ SVG in CSS backgrounds
▪ Blending of HTML/SVG elements	▪ Color input type	▪ SVG in HTML img element
▪ calc() as CSS unit value	▪ contenteditable attribute (basic support)	▪ SVG SMIL animation
▪ ch (character) unit	▪ Custom Elements	▪ SVG fonts
▪ 2.1 selectors	▪ Custom protocol handling	▪ All SVG features
▪ all property	▪ Datalist element	
▪ Animation	▪ dataset & data-* attributes	
▪ Appearance	▪ Date and time input types	
▪ background-attachment	▪ Details & Summary elements	
	▪ Dialog element	
	▪ Base64 encoding and decoding	
	▪ disabled attribute of the fieldset	

JS API

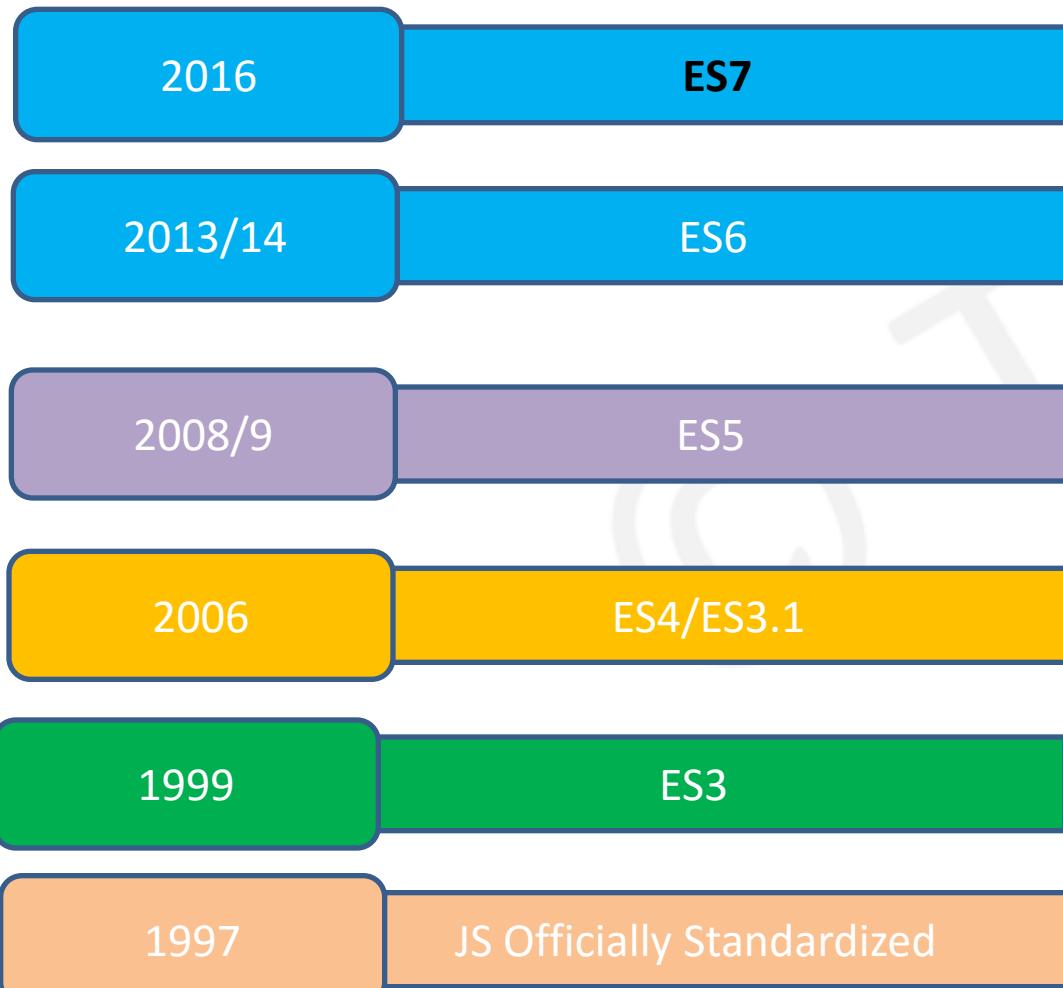
- Ambient Light API
- Arrow functions
- Basic console logging functions





# ECMA TimeLine

European Computer Manufacturers Association



<http://www.ecma-international.org/ecma-262/7.0/index.html>

<https://esdiscuss.org/>

## TC39 Guides ECMA Specification

**ECMAScript -> Language Specification**  
**JavaScript -> implementation of ECMAScript**  
**Java Nashorn -> implementation of ES5.1**  
**Flash/Flex -> implementation of ES3 + custom**





# ES6 CAN I USE

<http://caniuse.com/#search=es6>





# ES6 and TypeScript

- Areas of overlap
  - Classes
  - Modules
  - Arrow functions
  - Rest parameters
  - Default parameter values
- Core TypeScript additions
  - Types (static, compile-time-only) supporting static analysis and tools
  - **compilation to ES3**
- TypeScript Classes
  - ES6-> max-min classes
  - ...plus:
    - Public/private annotations (not runtime enforced currently)
    - Field declarations
    - Statics
    - Constructor parameter initializers
    - Static rejection of
  - ...minus (to be added in future)
    - Class expressions
    - Class-side inheritance





# ES6 and TypeScript

- Classes: Static
  - Statics are used frequently
  - Imperative update is awkward when using an otherwise declarative construct
- Classes: Class Privates
  - Frequent asks for privacy
  - Typescript added compile-time-only privacy
  - Not quite same as current private names syntax proposal

```
class Point {  
  constructor(x,y) { this.x = x; this.y = y; }  
  static origin = new Point(0,0);  
}
```

```
class Point {  
  private x: number;  
  private y: number;  
  constructor(x,y) { this.x = x; this.y = y; }  
}
```





# ES6 and TypeScript

- Classes: Decorators
  - With classes available, teams want to use them
  - Biggest blocker so far is when existing class library supported some extra ‘magic’ associated with class/method declarations
  -
- ES6 modules
  - Compiled to JS which uses AMD or commonJS loaders
  - not yet aligned on import syntax (ES6 not solid here, but may align in future)
  - much richer notion of internal modules to support “namespace” use cases





# ES6 and TypeScript

- Modules: Namespaces use case
  - Two common patterns of large code structure
    1. On demand loaded modules (AMD, CommonJS, others)
    2. Namespace objects to reduce global pollution
  - External modules address #1, internal module do not yet handle #2 well
  - TypeScript allows internal module re-declarations to grow the object
- Modules: “modules.exports=” use case
  - Not yet addressed in TypeScript
  - Critical for interop with existing CommonJS/AMD Code
  - Supportive of “export=” syntax proposal





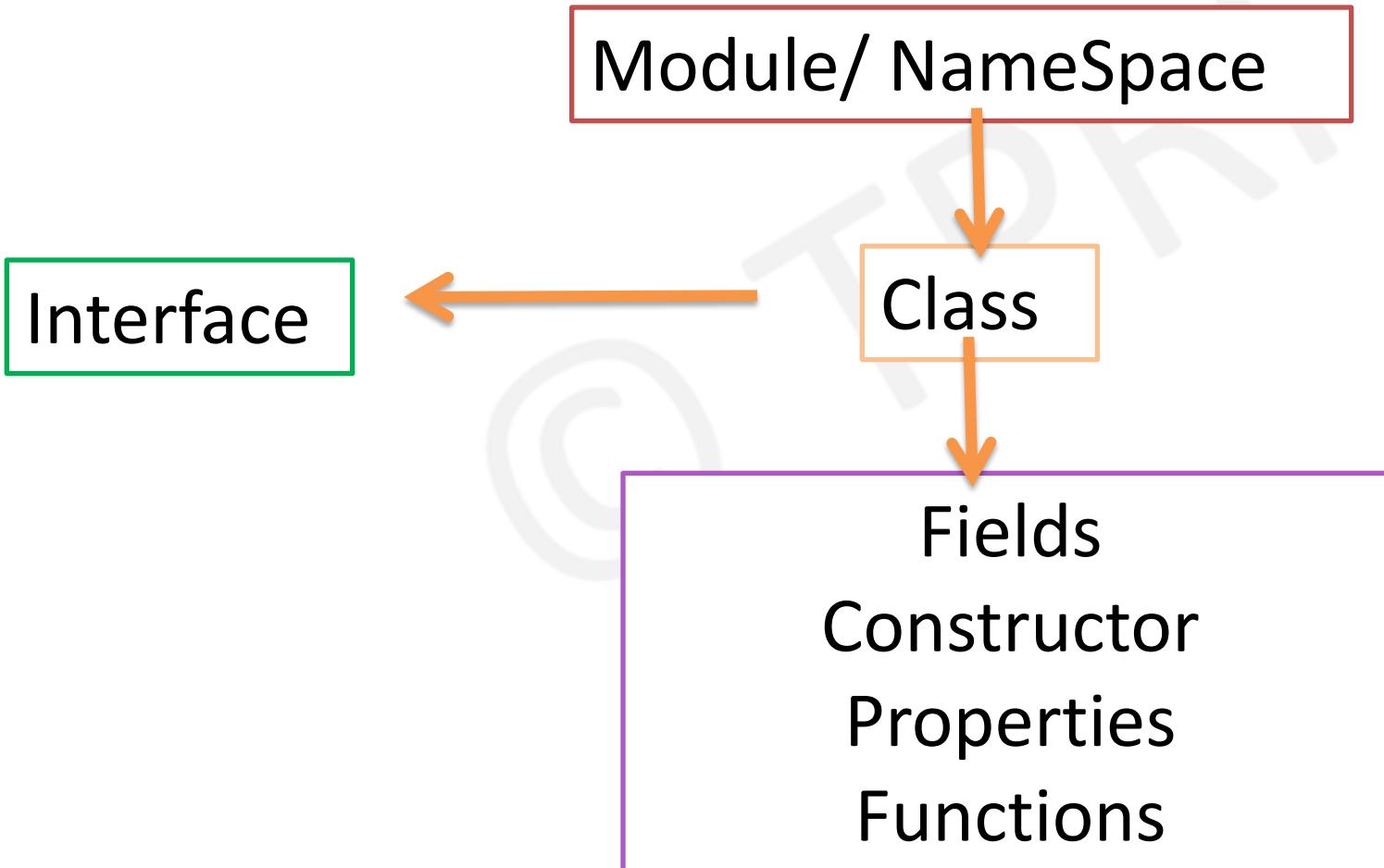
# TypeScript Keywords and Operators

Keyword	Description
Class	Container for members such as properties and functions
Constructor	Provides initialization functionality in a class
Exports	Export a member from a module
Extends	Extend a class or interface
Implements	Implement an interface
Imports	Imports a module
Interface	Defines code contract that can be implemented by types
Module/namespace	Container for classes and other code
Public/private	Member visibility modifiers
...	Rest parameter syntax
=>	Arrow syntax used with definitions and functions
<typeName>	<> Characters use to cast/convert between types
:	Separator between variable/parameter names and types





# TypeScript Hierarchy





# TypeScript Classes

## TypeScript

```
class Car {  
    //variables  
    engine: string;  
    //constructor  
    constructor(engine: string) {  
        this.engine = engine;  
    }  
    //function  
    start() {  
        console.log('engine started:' +  
            this.engine);  
    }  
    //function  
    stop() {  
        console.log('engine stopped:' +  
            this.engine);  
    }  
}  
  
window.onload = function () {  
    var car = new Car('V8');  
    car.start();  
    car.stop();  
};
```



## JavaScript

```
var Car = (function () {  
    //constructor  
    function Car(engine) {  
        this.engine = engine;  
    }  
    //function  
    Car.prototype.start = function () {  
        console.log('engine started:' + this.engine);  
    };  
    //function  
    Car.prototype.stop = function () {  
        console.log('engine stopped:' + this.engine);  
    };  
    return Car;  
})();  
  
window.onload = function () {  
    var car = new Car('V8');  
    car.start();  
    car.stop();  
};
```

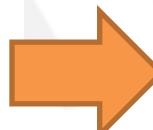




# TypeScript Classes

## TypeScript

```
class Employee {
    //variables
    fname: string;
    lname: string;
    email: string;
    salary: number;
    taxrate: number;
    //constructor
    constructor(fname: string, lname: string, email: string, salary: number, taxrate: number) {
        this.fname = fname;
        this.lname = lname;
        this.email = email;
        this.salary = salary;
        this.taxrate = taxrate;
    }
    //function
    netsalary() {
        var aftertaxsalary = this.salary - (this.salary * this.taxrate);
        console.log("After tax salary is :" + aftertaxsalary);
    }
}
window.onload = function () {
    var sak = new Employee('Awase Khirni', 'Syed', 'awasekhirni@gmail.com', 100000, 0.3);
    sak.netsalary();
};
```



## JavaScript

```
var Employee = (function () {
    //constructor
    function Employee(fname, lname, email, salary, taxrate) {
        this.fname = fname;
        this.lname = lname;
        this.email = email;
        this.salary = salary;
        this.taxrate = taxrate;
    }
    //function
    Employee.prototype.netsalary = function () {
        var aftertaxsalary = this.salary - (this.salary * this.taxrate);
        console.log("After tax salary is :" + aftertaxsalary);
    };
    return Employee;
})();
window.onload = function () {
    var sak = new Employee('Awase Khirni', 'Syed', 'awasekhirni@gmail.com', 100000, 0.3);
    sak.netsalary();
};
//# sourceMappingURL=Employee.js.map
```





# Annotations and inferences

TypeScript

	Description
Var any1;	Type Could be any type (any)
Var num1: numberone;	Type annotation
Var num2: number =2	Type Annotation Setting the value
Var num3 = 3	Type inference (number)
Var num4 = num3 +77	Type inference (number)
Var str1 = num1+ 'this works';	Type inference(string)
Var willnotwork:number = num1+"will give error";	Error





# Static and Dynamic typing

## TypeScript

Static typing (optional)

Typesafety is a compile-time feature

**declare var document;** ambient declaration are available in typescript

## JavaScript

Dynamic typing

Typesafety happens at run-time debugging

Ambient declarations do not appear anywhere in the JavaScript





# Definitely Typed

<http://definitelytyped.org/>

<https://www.nuget.org/packages/jquery.TypeScript.DefinitelyTyped/>

<https://www.nuget.org/packages/knockout.TypeScript.DefinitelyTyped/>

<https://www.nuget.org/packages/angularjs.TypeScript.DefinitelyTyped/>

<https://www.nuget.org/packages/requirejs.TypeScript.DefinitelyTyped/>

<https://www.nuget.org/packages/node.TypeScript.DefinitelyTyped/>





# Any Type and Primitives

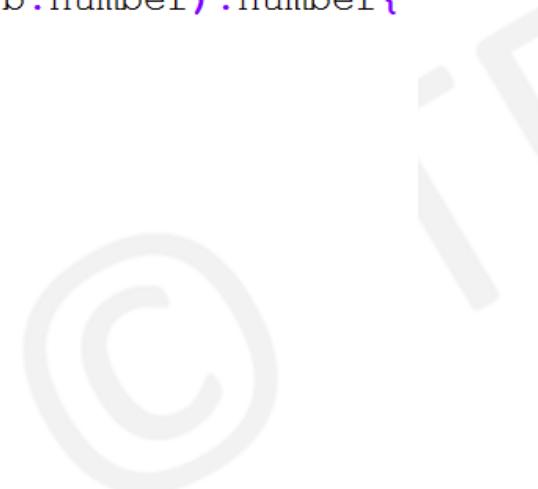
- Represents any JavaScript value
- No Static type checking on “any”  
`var data: any;`  
`var info;`

- Primitive Types
  - Number => `var age: number = 20;`
  - Boolean => `var isactive :boolean = true;`
  - String => `var firstName:string = "Awase"`
  - Arrays and Indexers
    - `var names : string[] = ['sak', 'sas']`
  - Null type – is a subtype of all primitives except for void and undefined
    - `var num: number = null`
  - Undefined => `var customer = undefined;`  
It is a subtype of all types.



# Type Annotations

```
var num =101;  
var num: number =101;  
function multiply(a:number, b:number):number{  
    return a*b;  
}  
  
var x:any;
```



- Optional Arguments

```
function multiply(a:number, b?:number){  
    if(!b){  
        return a;  
    }  
    return a*b;  
}
```

- Default Values

```
function multiply(a:number, b:number=101){  
    return a*b;  
}
```





# Object types

- Examples are Functions, class, module, interface and literal types
- Object types may contain
  - Properties
    - Public or private
    - Required or optional
  - Call signatures
  - Construct signatures
  - Index signatures

```
//object literals
var dimensions = { x: 10, y: 20,
z: 30 };

var room: Object = {
  x: 10, y: 20,
  z: 30
}

//functions are objects in
//typescript
var multiply = function (x:
number) {
  return x * x;
}
```





# Functions

- Parameter types (required and optional)
- Arrow function expressions
  - Compact form of function expressions
  - Omit the function keyword
  - Have scope of “this”
- Void
  - Used as the return type for functions that return no value.

```
//traditional approach
var computearea = function (width:
    number, length: number) {
    return width * length;
}

//using arrow function expression
//omitted the function keyword
//compact return statement with =>
var computevolume = (l: number, w:
    number, h: number) => l * h * w;
```





# void

- used as the return type of functions that returns no value

```
//void example
var loginstatus: (msg: string) =>
  void;

loginstatus = function (msg) {
  console.log(msg);
};

loginstatus('Welcome to
  KingsLanding!');
```





# Rest Parameters

```
function multiply(...factors:number[]) {
  var result = 1;
  for (var i=0;i<factors.length;i++) {
    result *=factors[i];
  }
  return result;
}

//execution
multiply(111,222,333);
```





# Overloading

```
function myFunction(arg:number):void;  
  
function myFunction(arg:string):void;  
  
function myFunction(arg:any):void{  
    //implementation goes here  
}
```





# Class Construct in TypeScript

```
class Post{  
    private title:string;  
    constructor(title:string){  
        this.title=title  
    }  
    toString():string{  
        return this.title;  
    }  
}
```





# Classes in TypeScript

- Classes act as containers for different members that encapsulate code.
- TypeScript class members include
  - Fields
  - Constructors
  - Properties
  - Functions
- Class members are public by default

```
class Book {  
    //fields  
    author: string;  
    isbn: string;  
    publisher: string;  
    bookname: string;  
    //constructor  
    constructor(author: string, bookname: string, publisher: string, isbn: string) {  
        this.author = author;  
        this.bookname = bookname;  
        this.publisher = publisher;  
        this.isbn = isbn;  
    }  
}
```





# Defining Properties in Classes

- Properties act as filters and can have a get or set blocks

```
class Vehicle{  
    //fields  
    private _engine: string;  
    //constructor  
    constructor(engine: string) {  
        this.engine = engine;  
    }  
    //property  
    get engine(): string {  
        return this._engine;  
    }  
    //property  
    set engine(value: string) {  
        if (value == undefined) throw 'Supply an  
            Engine!';  
        this._engine = value;  
    }  
}
```

- Traditionally in JavaScript, we use object.defineProperty() to set and get the properties of the object (ECMA5 Feature)

```
function Book(name) {  
    Object.defineProperty(this, "name", {  
        get: function() {  
            return "Book: " + name;  
        },  
        set: function(newName) {  
            name = newName;  
        },  
        configurable: false  
    });  
}
```





# Defining Properties in Classes

```
class Vehicle{
  //fields
  private _engine: string;
  //constructor
  constructor(engine: string) {
    this.engine = engine;
  }
  //property
  get engine(): string {
    return this._engine;
  }
  //property
  set engine(value: string) {
    if (value == undefined) throw 'Supply an
      Engine!';
    this._engine = value;
  }
}
```

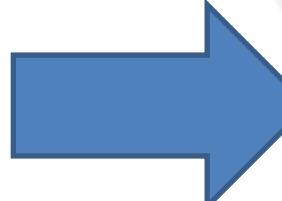
```
var Vehicle = (function () {
  //constructor
  function Vehicle(engine) {
    this.engine = engine;
  }
  Object.defineProperty(Vehicle.prototype, "engine", {
    //property
    get: function () {
      return this._engine;
    },
    //property
    set: function (value) {
      if (value == undefined)
        throw 'Supply an Engine!';
      this._engine = value;
    },
    enumerable: true,
    configurable: true
  });
  return Vehicle;
})();
//# sourceMappingURL=ExampleSix.js.map
```





# Using Complex Types

```
1 class Engine {  
2     constructor(public horsePower: number,  
3                  public engineType: string) { }  
4 }  
5 //new class  
6 class Automotive {  
7     //field declarations  
8     private _engine: Engine;  
9     //constructor  
10    constructor(engine: Engine) {  
11        this._engine = engine;  
12    }  
13 }  
14  
15 var toyotaengine = new Engine(300, 'v8');  
16 var innovacrysta = new Automotive(toyotaengine);
```



```
//using complex types  
var Engine = (function () {  
    function Engine(horsePower, engineType) {  
        this.horsePower = horsePower;  
        this.engineType = engineType;  
    }  
    return Engine;  
})();  
;  
  
//new class  
var Automotive = (function () {  
    //constructor  
    function Automotive(engine) {  
        this._engine = engine;  
    }  
    return Automotive;  
})();  
  
//instantiation  
var toyotaengine = new Engine(300, 'v8');  
var innovacrysta = new Automotive(toyotaengine);  
## sourceMappingURL=Examplethree.js.map
```





# Type Assertions

- TypeScript allows you to override its inferred and analyzed view of types any way you want to. Type assertion are purely YOU telling the compiler that you know about the types better than it does, and it should not second guess YOU.
- A common use case for type assertion is when porting over code from javascript to typescript.
- reason why it is “not called type casting” is that casting generally implies some sort of runtime support.
- assets are purely a compile time construct and a way to inform the compiler how you want your code to be analyzed.

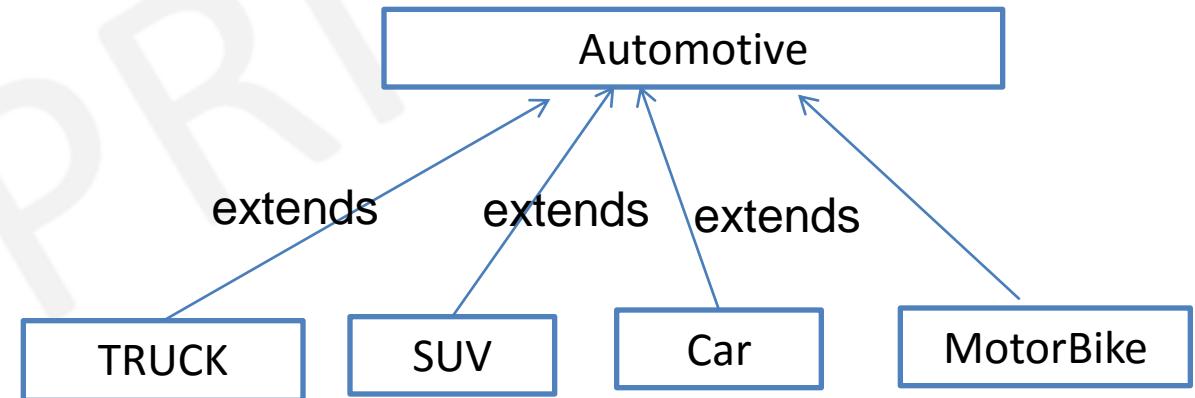




# Extending Types

- Types can be extended using the TypeScript “extends” keyword

```
class ChildClass extends ParentClass{  
    constructor() {  
        super();  
    }  
}
```





# Extending Types

```
class Auto {  
  engine: Engine;  
  constructor(engine: Engine) {  
    this.engine = engine;  
  }  
}  
  
//truck derives from auto base class  
class Truck extends Auto {  
  fourByFour: boolean;  
  constructor(engine: Engine, fourByFour: boolean) {  
    //calling the base class constructor  
    super(engine);  
    this.fourByFour = fourByFour;  
  }  
}
```

```
1 class Animal {  
2   name: string;  
3   constructor(theName: string) {  
4     this.name = theName;  
5   }  
6   move(distanceInMeters: number= 0) {  
7     console.log(`${this.name} moved ${distanceInMeters}m.`);  
8   }  
9 }  
  
0 class Horse extends Animal {  
1   constructor(name: string) {  
2     super(name);  
3   }  
4   move(distanceInMeters= 50) {  
5     console.log("Galloping..");  
6     super.move(distanceInMeters);  
7   }  
8 }  
9 }
```





# Interface in TypeScript

```
interface Printable{
    print():void;
}

class Printer implements Printable{
    print():void;
    console.log("Hello! You fool I love You!");
}
```





# Using Interface

- Interfaces are code contracts. TypeScript supports interfaces.
- An interface is defined as a syntactical contract that all classes inheriting the interface should follow. The interface defines the “what” part of the syntactical contract and the deriving classes define the “how” part of the syntactical contract.
- Interfaces define properties, methods and events, which are members of the interface.
- Interface contain only declaration of the members.

```
1 interface IEngine {  
2     start(callback: (startStatus: boolean, engineType: string) =>  
3         void): void;  
4  
5 }  
6  
7 class AutoEngine implements IEngine {  
8     constructor(public horsePower: number, public engineType:  
9         string) { }  
10  
11     start(callback: (startStatus: boolean, engineType: string) =>  
12         void) {  
13         window.setTimeout(() => {  
14             callback(true, this.engineType);  
15         }, 1000);  
16     }  
17  
18     stop(callback: (startStatus: boolean, engineType: string) =>  
19         void) {  
20         window.setTimeout(() => {  
21             callback(true, this.engineType);  
22         }, 1000);  
23     }  
24 }
```





# Module

- Modules help us keep separation based on similar functionality
- Modules help us in testing individual functionality
- Modules help us make the code reusable and maintainable.
- We can extend modules within or across files
- Each module has a specific

```
1 module dataservice {  
2   export class DataSync {  
3   }  
4 }  
5 //alternatively without module declaration, no exports, no imports  
6 class Library {  
7 }  
8  
9 var universitylib = new Library()
```





# Structural Typing

- TypeScript has a structural type system.
- Two types are compatible if they share the same structure
  - A private fields makes a class incompatible with everything else except its super class
  - <https://github.com/Microsoft/TypeScript/wiki>Type%20Compatibility>

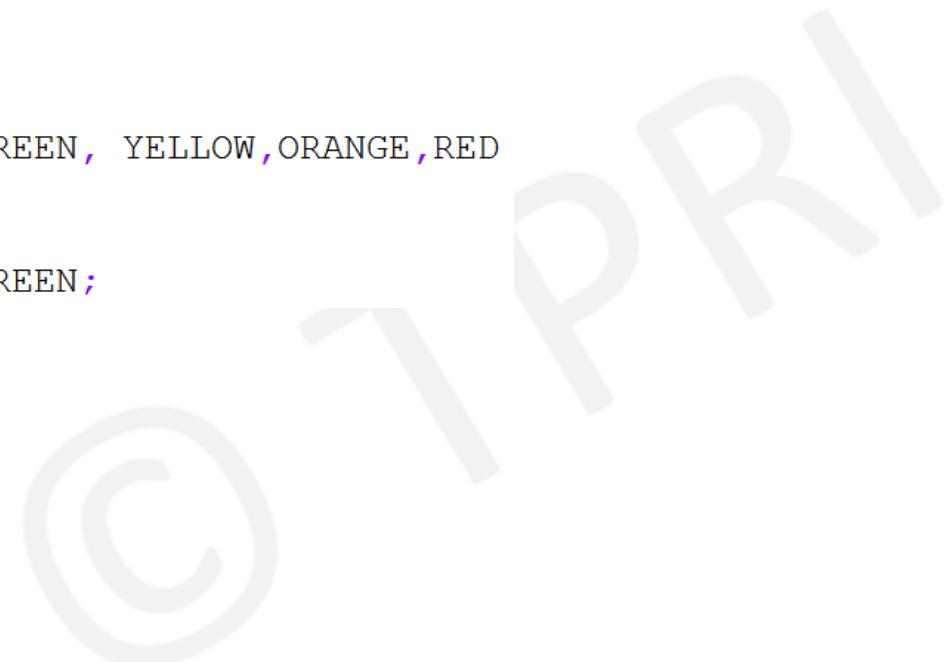
```
interface Point {  
    x:number;  
    y:number;  
    z?:number;  
}  
  
function geoPoint(point:Point) {  
}  
  
geoPoint({x:1,y:2});  
geoPoint({x:1,y:2, z:5});
```





# Enums

```
enum Color{  
    VIOLET, INDIGO, BLUE, GREEN, YELLOW, ORANGE, RED  
}  
  
var pickedColor = Color.GREEN;
```





# Generics

```
class ItemCollection< TThing >
{
    private _things: Array< TThing >

    Add(thing: TThing) {
        this._things.push(thing);
    }
}
```



# Arrow Function Expressions

© TPRI





# Tuple Types

- Providing typing for the elements of an array

```
var tuple:[number,string]=[990008,"Syed Awase", true];
```





# Union Types

- Value can have one of multiple types

```
var ut: string[] | string;
```

```
ut.length === 0
```





# Type Guards

```
function valueformatter(c:string[]|string){  
  if(typeof c ==='string') {  
    return c.trim();  
  }else{  
    return c.join('');  
  }  
}
```





# any

- “any” type annuls type checking
- **Typescript prompts for using “any” when unsure.**

© TPRI



© TPRI

# TYPESCRIPT TESTING





# Unit Testing

- unit tests add clarity to your code
- They favour decoupling
- Encourages simplification
- Supports Extensibility





Contact Us

[sak@sycliq.com](mailto:sak@sycliq.com)

[sak@territorialprescience.com](mailto:sak@territorialprescience.com)

For code driven trainings for Technology Firms

Reach out to us +91-9035433124

No middle men – We understand Technology and Science

We are hardcore Technologists/Architects/Programmers

Most of these trainings are offered by Dr. Syed Awase

# Thank You

We also provide Code Driven Open House Trainings

INDIA

HYDERABAD | BANGALORE | CHENNAI | PUNE

OVERSEAS

SINGAPORE | MALAYSIA | DUBAI



## Current Offerings

- AngularJS 1.5.x
- Typescript /CoffeeScript /Dart
- AngularJS 2 (with NodeJS)
- KnockOutJS (with NodeJS)
- BackBoneJS (with NodeJS)
- Ember JS / Ext JS (with NodeJS)
- Raspberry Pi
- Responsive Web Design with Bootstrap, Google Material Design and KendoUI
- C# ASP.NET MVC
- C# ASP.NET WEB API
- C# ASP.NET WCF, WPF
- JAVA , SPRING, HIBERNATE
- Python , Django
- R Statistical Programming
- Android Programming
- Python/Django
- Ruby on Rails
- NoSQL (MongoDB – Essentials & Advanced )

Section II

# ANGULAR 2 PLATFORM FEATURES



# Develop Once and Deploy Everywhere

- Angular enables you to build applications once and reuse your code and abilities to build apps for any deployment environment.
- Be it web, mobile, native mobile application and native desktop applications.
- With angular you can achieve 5x speed in comparison to angularjs 1.5.x. it exploits HTML5 web-workers and server side rendering making your javascript code concurrent.





A screenshot of the Babel.js website. The header reads 'BABEL'. The main title is 'Babel is a JavaScript compiler.' followed by the subtitle 'Use next generation JavaScript, today.'. Below the title is a button labeled 'Check out the Babel Handbook!'. At the bottom left is a GitHub star icon with the number '17,074'. The background features a dark, abstract graphic of hands.

**Babel transforms your  
JavaScript**



### What is Traceur?

Traceur is a JavaScript.next-to-JavaScript-of-today compiler that allows you to use features from the future. It supports ES6 as well as some experimental ES.next features.

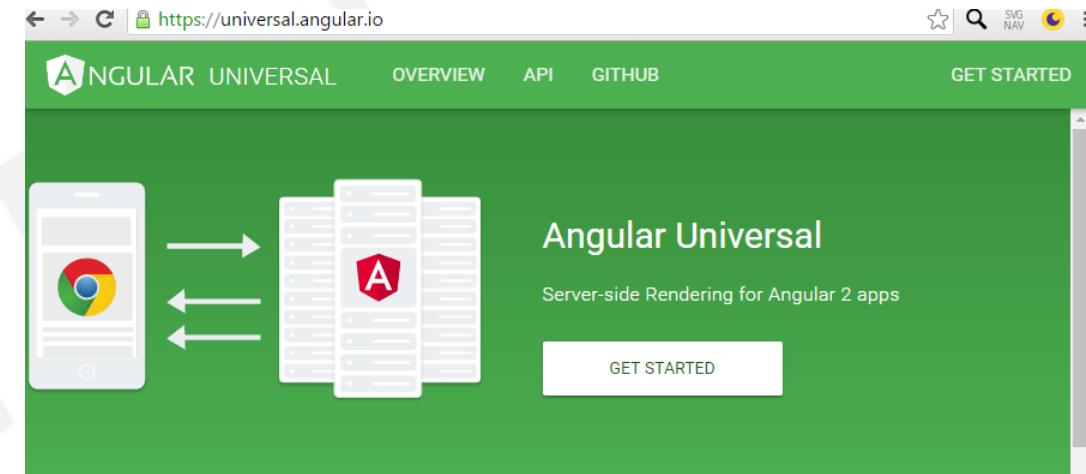
Traceur's goal is to inform the design of new JavaScript features which are only valuable if they allow you to do something useful. Traceur allows you to try out new and proposed language features today, helping you say what you mean and informing the standards process.





# Angular Universal

[angular-universal](#) is a library that lives under the Angular 2 GitHub repository with the goal of making server-side rendering as easy and straightforward as possible. Since Angular 2 is made to be platform agnostic, it can be executed in non-browser environments without too much issues.



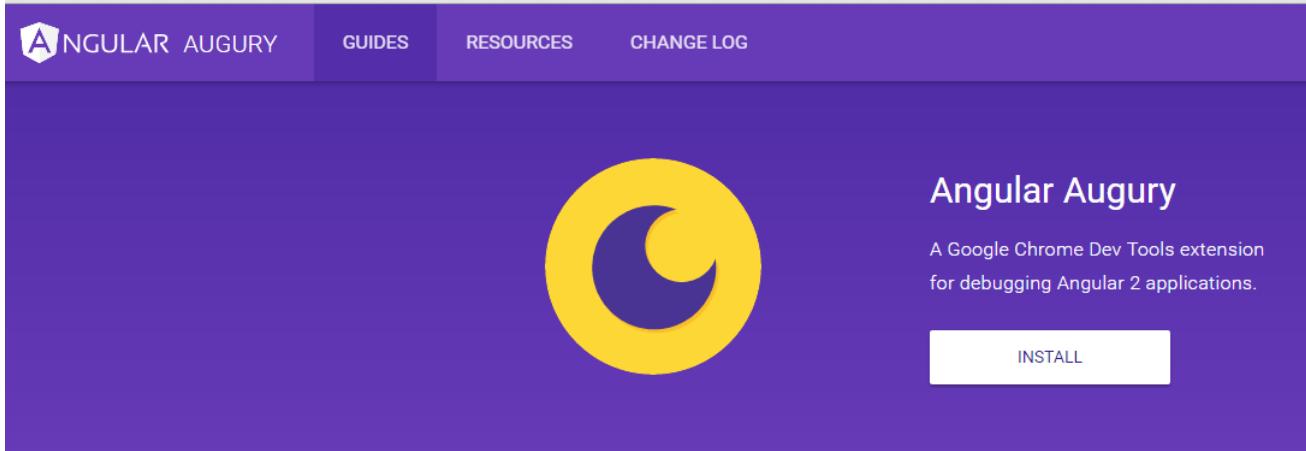
## Better Perceived Performance

First time users of your application will instantly see a server rendered view which greatly improves perceived performance and the overall user experience. According to [research at Google](#), the difference of just 200 milliseconds in page load performance has an impact on





# Productivity Tools : Angular Augury



The screenshot shows the homepage of the Angular Augury website at <https://augury.angular.io>. The header includes navigation links for 'GUIDES', 'RESOURCES', and 'CHANGE LOG'. The main content features a large yellow circular logo with a blue crescent moon shape. To the right of the logo, the text 'Angular Augury' is displayed, followed by a description: 'A Google Chrome Dev Tools extension for debugging Angular 2 applications.' A prominent 'INSTALL' button is located below the description.

## What is Augury

Augury is the most used Google Chrome Developer Tool extension for debugging and profiling Angular 2 applications.

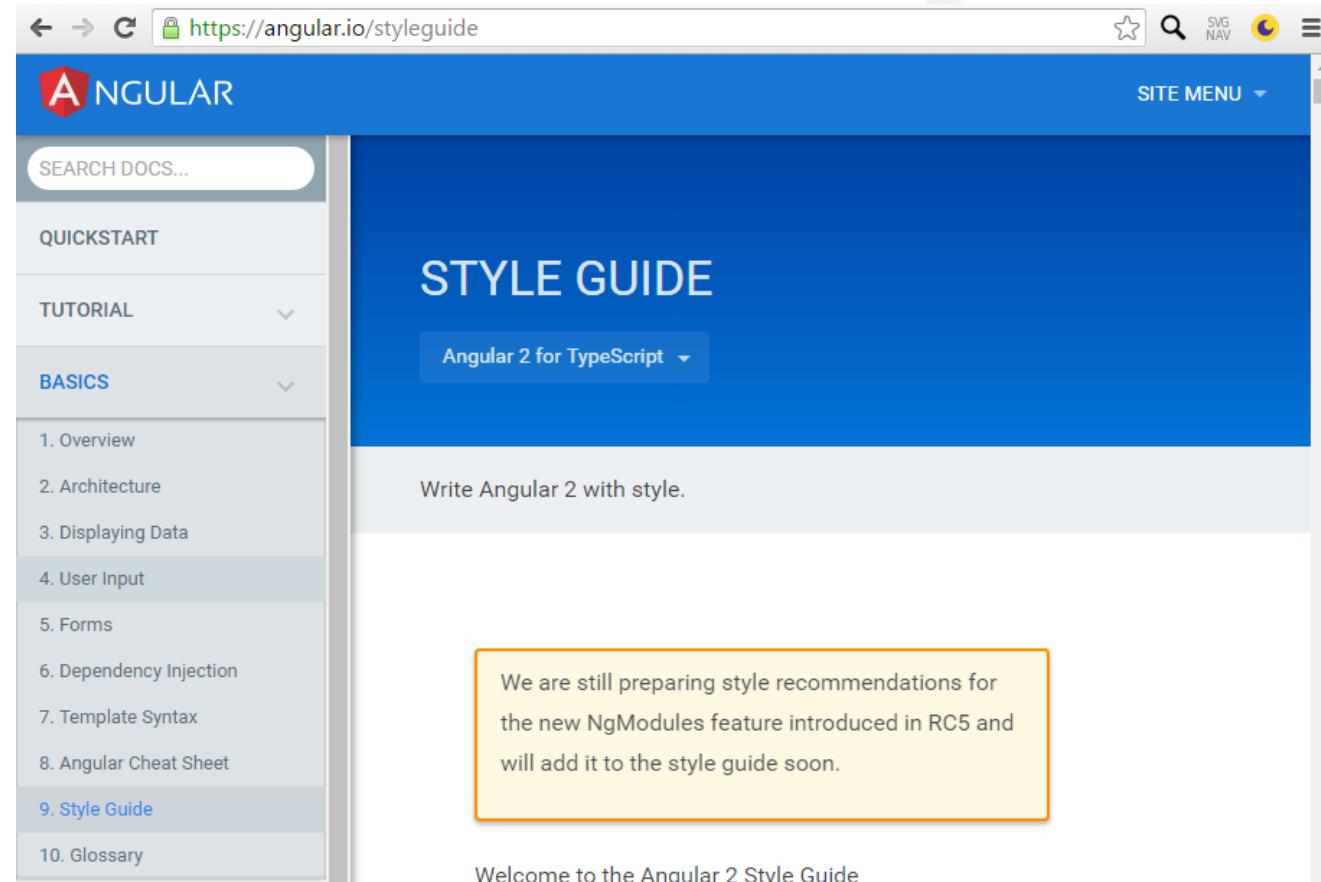
## Why Use Augury?

Augury helps Angular 2.0 developers visualize the application through component trees, and visual debugging tools. Developers get immediate insight into their application structure, change detection and performance characteristics.





# Angular.io/styleguide



The screenshot shows a web browser displaying the Angular Style Guide at <https://angular.io/styleguide>. The page has a blue header with the Angular logo and a search bar. A sidebar on the left contains links for Quickstart, Tutorial, Basics, and a numbered list from 1. Overview to 10. Glossary, with '9. Style Guide' highlighted in blue. The main content area features a large 'STYLE GUIDE' heading and a dropdown menu set to 'Angular 2 for TypeScript'. Below this is a section titled 'Write Angular 2 with style.' and a yellow callout box stating: 'We are still preparing style recommendations for the new NgModules feature introduced in RC5 and will add it to the style guide soon.' At the bottom, a welcome message reads: 'Welcome to the Angular 2 Style Guide'.





# Codelyzer : Architectural feedback/Style Feedback



Minko Gechev

mgechev

<https://github.com/mgechev/codelyzer>

The screenshot shows the GitHub repository page for 'mgechev / codelyzer'. At the top, there's a header with 'This repository' and a search bar. Below the header, it says 'mgechev / codelyzer'. To the right of the repository name are buttons for 'Watch 38', 'Star 468', 'Fork 39', and a dropdown menu. Underneath the repository name, there are links for 'Code' (which is highlighted), 'Issues 18', 'Pull requests 1', 'Wiki', 'Pulse', and 'Graphs'. A note below says 'No description or website provided.' Below this, there's a summary bar with '129 commits', '4 branches', '0 releases', and '8 contributors'. A dropdown menu shows 'Branch: master' and a 'New pull request' button. At the bottom, there's a list of commits:

Author	Commit Message	Date
mgechev	fix(usePipeTransform): handle case of simple decorator	7 days ago
build	chore(links): add basic build script	4 months ago
manual_typings	feat: add ng2 walker	6 months ago
src	fix(usePipeTransform): handle case of simple decorator	7 days ago
test	fix(usePipeTransform): handle case of simple decorator	7 days ago
.gitignore	feat(rules): improve error messages	4 months ago
.npmignore	Update version	6 months ago
README.md	fix: handle the case of alias imports	2 months ago
package.json	chore: bump npm version	9 days ago
tsconfig.json	chore(build): publish rules more convenient	4 months ago
tslint.json	Initial commit	6 months ago
typings.json	feat(typings): upgrade to version 1.3.2	9 days ago

- TypeScript Static Code Analysis and matches against the style guide.
- Actual feedback on your application.





# Mobile.angular.io

https://mobile.angular.io

ULAR MOBILE GUIDES

The screenshot shows a mobile application interface for a weather app. At the top, there's a navigation bar with three horizontal lines and the word "Weather". To the right of the bar are two buttons: one for "°C/F" and another for "55°". Below the bar, there are four cards, each representing a location and its temperature: "San Francisco 55°", "New York 48°", "Toronto 45°", and "Nunavut 25°". Each card has a small plus sign at the bottom right corner. A large orange button labeled "PREVIEW ON GITHUB" is positioned below the cards.

## Angular Mobile Toolkit

All the tools and techniques to build high-performance mobile apps

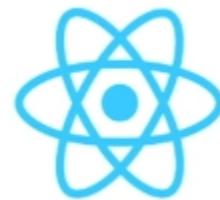
[PREVIEW ON GITHUB](#)

### Performance of native, discoverability of the Web

The Angular Mobile Toolkit makes it easy to build snappy Web apps that load instantly on any device, even without an internet connection. Take advantage of the searchability, shareability, and no-install-required-ability of the Web without compromise.

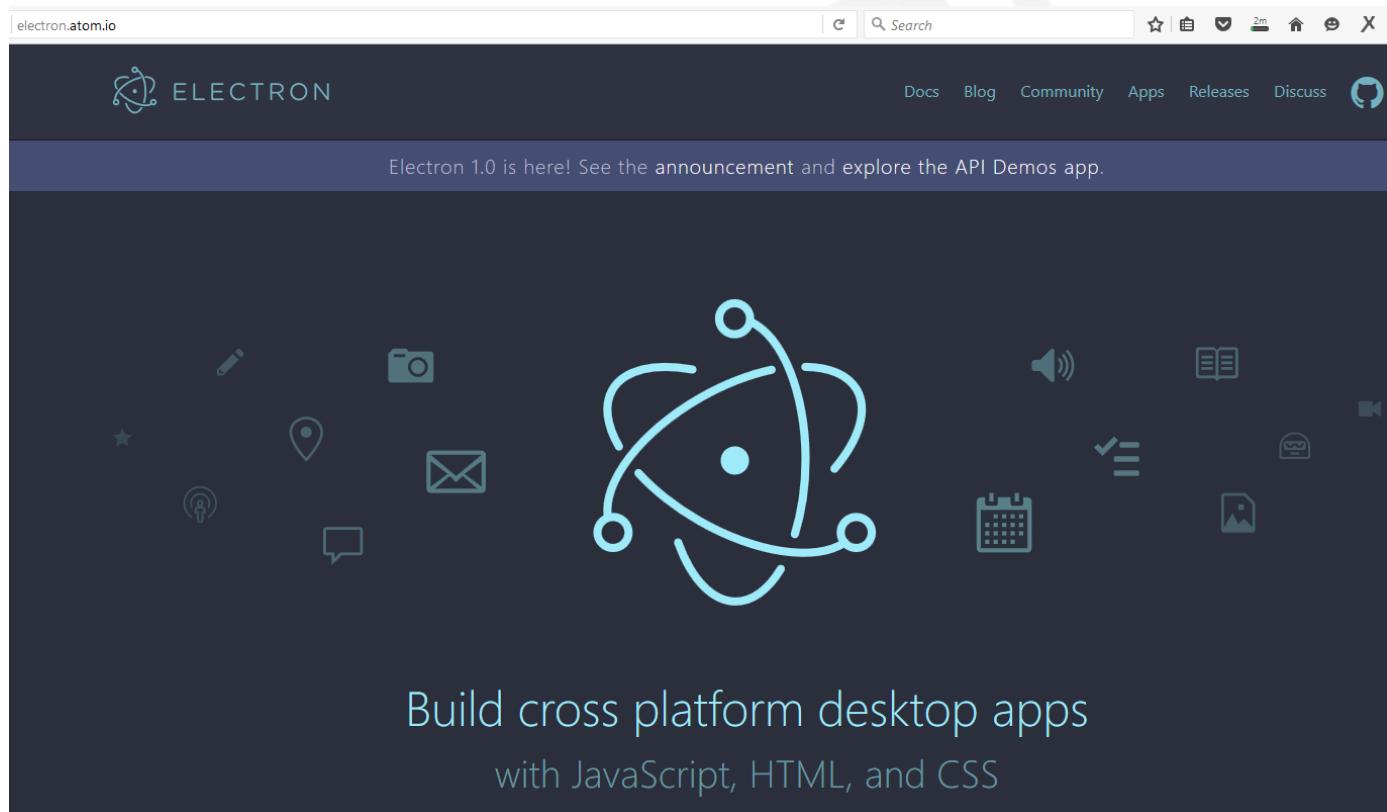


<https://www.nativescript.org/>





# Electron + Angular



Build cross platform desktop apps  
with JavaScript, HTML, and CSS





# UI Components

**Angular Material**  
Material Design components for Angular 2 apps  
[PREVIEW ON GITHUB](#)

<https://material.angular.io/>

wijmo.com  
More Tools | ComponentOne Studio .NET Controls | Xuni Native Mobile | Ultimate bundle  
Search  
PRODUCTS DOWNLOAD SUPPORT BLOG COMPANY PRICING  
wijmo  
Wijmo Enterprise  
Next-Gen HTML5/JavaScript UI Controls  
Lightning-fast, touch-first, flexible controls for your enterprise applications  
What's New Download Free Trial (v 2016)

<http://wijmo.com/>

www.primefaces.org/primeng/#/  
PRIME NG SHOWCASE  
Input Button Data Panel Overlay Menu Charts Messages  
PrimeNG PrimeFaces WIDGETS PRODUCTIVITY  
PrimeNG is a collection of rich UI components for AngularJS. PrimeNG is a sibling of the popular JavaServer Faces Component Suite, PrimeFaces.  
All widgets are open source and free to use under Apache License 2.0, a commercial friendly license.  
PrimeNG is developed by  
Setup Premium Free Themes Forum

<http://www.primefaces.org/primeng/#/>

ng-lightning.github.io/ng-lightning/#/  
NG-LIGHTNING Components Support npm  
NG-LIGHTNING Native Angular 2 components & directives for Lightning Design System  
BUILDING FOR THE FUTURE  
Built upon next generation frameworks like [Angular 2](#) and [Salesforce's Lightning Design System](#), rest assured that you building an application to stand the test of time.  
BEST PRACTICES

<http://ng-lightning.github.io/ng-lightning/#/>

## Ng2-bootstrap

https://vaadin.com/home  
vaadin Framework Elements Community Services Pro Tools Company 12 Sign In Register C  
User interface components for web apps  
Build your apps in Groovy  
See more

<https://vaadin.com/home>





# Benefits of Angular

1. Angular is built using TypeScript, making it easier to switch from Object Oriented Programming Languages like Java, C# etc.
2. TypeScript code analyzer warns of potential errors
3. Using TypeScript classes and interfaces\* makes the code more concise and easy to read and write
4. TypeScript compiler generates Javascript that is readable in ES3, ES5 or ES6 versions.
5. Application development can be done using either webpack or SystemJS. They also take care of bundling and optimization of the code.
6. Clean dependency injection to implement loose coupling between components and services
7. Binding and events allows you to create reusable and loosely coupled components
8. Each component goes through a well defined lifecycle and hooks for intercepting important component events





# Convention for Reading Angular API

D Directive

P Pipe

@ Decorator

C Class

I Interface

F Function

E Enum

T Type Alias

K Const





# @angular/common(2.4.0)

Dec 20, 2016- Release Date

## @angular/common

K APP\_BASE\_HREF  
P CurrencyPipe  
C HashLocationStrategy  
P JsonPipe  
I LocationChangeListener  
D NgClass  
D Nglf  
D NgPlural  
D NgSwitch  
D NgTemplateOutlet  
C PlatformLocation  
P UppercasePipe

P AsyncPipe  
P DatePipe  
P I18nPluralPipe  
C Location  
C LocationStrategy  
D NgComponentOutlet  
C NgLocaleLocalization  
D NgPluralCase  
D NgSwitchCase  
C PathLocationStrategy  
P SlicePipe  
K VERSION

C CommonModule  
P DecimalPipe  
P I18nSelectPipe  
I LocationChangeEvent  
P LowerCasePipe  
D NgFor  
C NgLocalization  
D NgStyle  
D NgSwitchDefault  
P PercentPipe  
P TitleCasePipe

source: <https://angular.io/docs/ts/latest/api/>



# @angular/common/testing(2.4.0)

Dec 20, 2016- Release Date

@angular/common/testing

c MockLocationStrategy

c SpyLocation





## @angular/core(2.4.0)

Dec 20, 2016- Release Date

K ANALYZE\_FOR\_ENTRY\_COMPONENTS

K APP\_INITIALIZER

C AfterContentInit

C AnimationAnimateMetadata

C AnimationKeyframesSequenceMetadata

C AnimationSequenceMetadata

C AnimationStateTransitionMetadata

C AnimationWithStepsMetadata

C ApplicationRef

K CUSTOM\_ELEMENTS\_SCHEMA

F Class

I CollectionChangeRecord

T CompilerOptions

C ComponentFactoryResolver

@ ContentChildren

C DefaultIterableDiffer

K APP\_BOOTSTRAP\_LISTENER

K AUTO\_STYLE

C AfterViewChecked

C AnimationEntryMetadata

C AnimationMetadata

C AnimationStateDeclarationMetadata

C AnimationStyleMetadata

C ApplicationInitStatus

I Attribute

E ChangeDetectionStrategy

I ClassDefinition

C Compiler

@ Component

C ComponentRef

C DebugElement

@ Directive

K APP\_ID

C AfterContentChecked

C AfterViewInit

C AnimationGroupMetadata

C AnimationPlayer

C AnimationStateMetadata

C AnimationTransitionEvent

C ApplicationModule

K COMPILER\_OPTIONS

C ChangeDetectorRef

I ClassProvider

C CompilerFactory

C ComponentFactory

@ ContentChild

C DebugNode

C DoCheck





# @angular/core(2.4.0)

Dec 20, 2016- Release Date

<code>c</code> ElementRef	<code>c</code> EmbeddedViewRef	<code>c</code> ErrorHandler
<code>c</code> EventEmitter	<code>I</code> ExistingProvider	<code>I</code> FactoryProvider
<code>I</code> ForwardRefFn	<code>I</code> GetTestability	<code>@</code> Host
<code>I</code> HostBinding	<code>I</code> HostListener	<code>@</code> Inject
<code>@</code> Injectable	<code>c</code> Injector	<code>I</code> Input
<code>I</code> IterableChangeRecord	<code>I</code> IterableChanges	<code>I</code> IterableDiffer
<code>I</code> IterableDifferFactory	<code>c</code> IterableDifters	<code>I</code> KeyValueChangeRecord
<code>I</code> KeyValueChanges	<code>I</code> KeyValueDiffer	<code>I</code> KeyValueDifferFactory
<code>c</code> KeyValueDifters	<code>K</code> LOCALE_ID	<code>c</code> ModuleWithComponentFactories
<code>I</code> ModuleWithProviders	<code>K</code> NO_ERRORS_SCHEMA	<code>I</code> NgModule
<code>c</code> NgModuleFactory	<code>c</code> NgModuleFactoryLoader	<code>c</code> NgModuleRef
<code>c</code> NgProbeToken	<code>c</code> NgZone	<code>c</code> OnChanges
<code>c</code> OnDestroy	<code>c</code> OnInit	<code>c</code> OpaqueToken
<code>@</code> Optional	<code>I</code> Output	<code>K</code> PACKAGE_ROOT_URL
<code>K</code> PLATFORM_INITIALIZER	<code>I</code> Pipe	<code>I</code> PipeTransform
<code>c</code> PlatformRef	<code>T</code> Provider	<code>c</code> Query
<code>c</code> QueryList	<code>c</code> ReflectiveInjector	<code>c</code> ReflectiveKey
<code>c</code> RenderComponentType	<code>c</code> Renderer	<code>c</code> ResolvedReflectiveFactory





# @angular/core(2.4.0)

Dec 20, 2016- Release Date

I ResolvedReflectiveProvider	C RootRenderer	C Sanitizer
I SchemaMetadata	E SecurityContext	@ Self
C SimpleChange	I SimpleChanges	@ SkipSelf
C SystemJsNgModuleLoader	C SystemJsNgModuleLoaderConfig	K TRANSLATIONS
K TRANSLATIONS_FORMAT	C TemplateRef	C Testability
C TestabilityRegistry	I TrackByFn	@ Type
I TypeProvider	K VERSION	I ValueProvider
C Version	@ ViewChild	@ ViewChildren
C ViewContainerRef	E ViewEncapsulation	C ViewRef
C WrappedValue	I WtfScopeFn	F animate
F asNativeElements	F assertPlatform	F createPlatform
F createPlatformFactory	F destroyPlatform	F enableProdMode
F forwardRef	F getDebugNode	F getModuleFactory
F getPlatform	F group	F isDevMode
F keyframes	K platformCore	F resolveForwardRef
F sequence	F setTestabilityGetter	F state
F style	F transition	F trigger
K wtfCreateScope	K wtfEndTimeRange	K wtfLeave
K wtfStartTimeRange		





# @angular/core/testing(2.4.0)

Dec 20, 2016- Release Date

## @angular/core/testing

**c** ComponentFixture

**c** InjectSetupWrapper

**c** TestComponentRenderer

**F** discardPeriodicTasks

**F** get TestBed

**F** tick

**K** ComponentFixtureAutoDetect

**T** MetadataOverride

**T** TestModuleMetadata

**F** fakeAsync

**F** inject

**F** withModule

**K** ComponentFixtureNoNgZone

**c** TestBed

**F** async

**F** flushMicrotasks

**F** resetFakeAsyncZone





## @angular/forms

## @angular/forms

**c** AbstractControl**I** AsyncValidatorFn**c** ControlContainer**I** Form**c** FormBuilder**D** FormControlName**D** FormGroupName**D** MinLengthValidator**K** NG\_VALUE\_ACCESSOR**D** NgControlStatusGroup**D** NgModelGroup**D** RadioControlValueAccessor**D** SelectControlValueAccessor**I** Validator**c** AbstractControlDirective**D** CheckboxControlValueAccessor**I** ControlValueAccessor**c** FormArray**c** FormControl**c** FormGroup**c** FormsModuleModule**K** NG\_ASYNC\_VALIDATORS**c** NgControl**D** NgForm**D** NgSelectOption**c** ReactiveFormsModuleModule**D** SelectMultipleControlValueAccessor**I** ValidatorFn**c** AbstractFormGroupDirective**D** CheckboxRequiredValidator**D** DefaultValueAccessor**D** FormArrayName**D** FormControlDirective**D** FormGroupDirective**D** MaxLengthValidator**K** NG\_VALIDATORS**D** NgControlStatus**D** NgModel**D** PatternValidator**D** RequiredValidator**K** VERSION**c** Validators



## @angular/http

## @angular/http

- c BaseRequestOptions
- c Connection
- c Headers
- c JSONPBackend
- c JsonpModule
- c Request
- i RequestOptionsArgs
- c ResponseOptions
- c URLSearchParams
- c XHRConnection

## @angular/http/testing

- c MockBackend

- c BaseResponseOptions
- c ConnectionBackend
- c Http
- c JSONPConnection
- c QueryEncoder
- e RequestMethod
- c Response
- t ResponseOptionsArgs
- k VERSION
- c XSRFStrategy

- c MockConnection

Dec 20, 2016- Release Date

- c BrowserXhr
- c CookieXSRFStrategy
- c HttpModule
- c Jsonp
- e ReadyState
- c RequestOptions
- e ResponseContentType
- e ResponseType
- c XHRBackend





# @angular/platform-browser

## @angular/platform-browser

**c** AnimationDriver

**k** DOCUMENT

**c** EventManager

**c** Meta

**i** SafeHtml

**i** SafeStyle

**k** VERSION

**k** platformBrowser

**c** BrowserModule

**c** DomSanitizer

**k** HAMMER\_GESTURE\_CONFIG

**i** MetaDefinition

**i** SafeResourceUrl

**i** SafeUrl

**f** disableDebugTools

**d**ec 20, 2016- Release Date

**c** By

**k** EVENT\_MANAGER\_PLUGINS

**c** HammerGestureConfig

**c** NgProbeToken

**i** SafeScript

**c** Title

**f** enableDebugTools

## @angular/platform-browser/testing

**c** BrowserTestingModule

**k** platformBrowserTesting

**c** BrowserDynamicTestingModule

**k** platformBrowserDynamicTesting

## @angular/platform-browser-dynamic

**k** RESOURCE\_CACHE\_PROVIDER

**k** VERSION

**k** platformBrowserDynamic





## @angular/platform-server

c ServerModule

k VERSION

k platformDynamicServer

k platformServer

## @angular/platform-server/testing

c ServerTestingModule

k platformServerTesting

## @angular/platform-webworker

c ClientMessageBroker

c ClientMessageBrokerFactory

c FnArg

c MessageBus

i MessageBusSink

i MessageBusSource

k PRIMITIVE

c ReceivedMessage

c ServiceMessageBroker

c ServiceMessageBrokerFactory

c UiArguments

k VERSION

k WORKER\_APP\_LOCATION\_PROVIDERS

k WORKER\_UI\_LOCATION\_PROVIDERS

c WorkerAppModule

f bootstrapWorkerUi

k platformWorkerApp

k platformWorkerUi

## @angular/platform-webworker-dynamic

k VERSION

k platformWorkerAppDynamic





# Angular Router

## @angular/router

I ActivatedRoute	I ActivatedRouteSnapshot	I CanActivate
I CanActivateChild	I CanDeactivate	I CanLoad
T Data	C DefaultUrlSerializer	T DetachedRouteHandle
T Event	I ExtraOptions	T LoadChildren
T LoadChildrenCallback	C NavigationCancel	C NavigationEnd
C NavigationError	I NavigationExtras	C NavigationStart
C NoPreloading	K PRIMARY_OUTLET	T Params
C PreloadAllModules	C PreloadingStrategy	K ROUTER_CONFIGURATION
K ROUTER_INITIALIZER	I Resolve	T ResolveData
I Route	C RouteReuseStrategy	C Router
D RouterLink	D RouterLinkActive	D RouterLinkWithHref
C RouterModule	D RouterOutlet	C RouterOutletMap
C RouterPreloader	I RouterState	I RouterStateSnapshot
T Routes	C RoutesRecognized	C UrlHandlingStrategy
C UrlSegment	C UrlSegmentGroup	C UrlSerializer
I UrlTree	K VERSION	F provideRoutes



## @angular/router/testing

C RouterTestingModule

C SpyNgModuleFactoryLoader

F setupTestingRouter





# Angular Upgrade

## @angular/upgrade

c UpgradeAdapter

c UpgradeAdapterRef

k VERSION

## @angular/upgrade/static

c UpgradeComponent

c UpgradeModule

f downgradeComponent

f downgradeInjectable





# Angular 2 Insights

Section III





# AngularJS 2.0

## PRE-REQUISITE

- Understanding of Angular JS 1.5.x or JavaScript
- TypeScript programming knowledge (Covered as part of the course)
- ES6 Specification and Constructs
- Knowledge of Object-oriented programming concepts , previous experience with C++/C#/Java



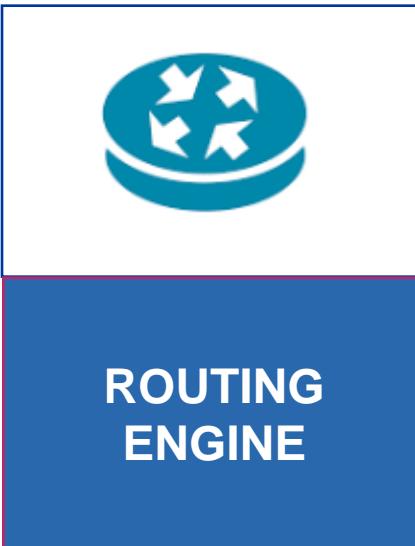
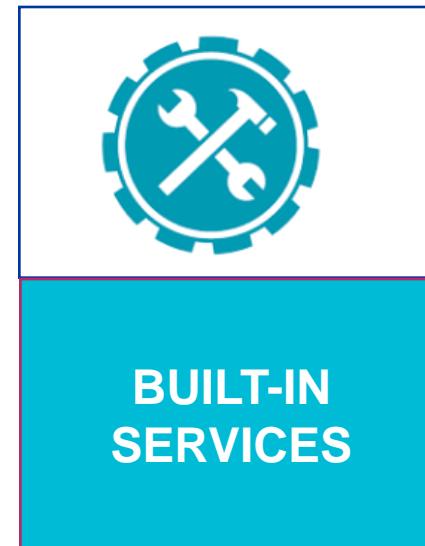
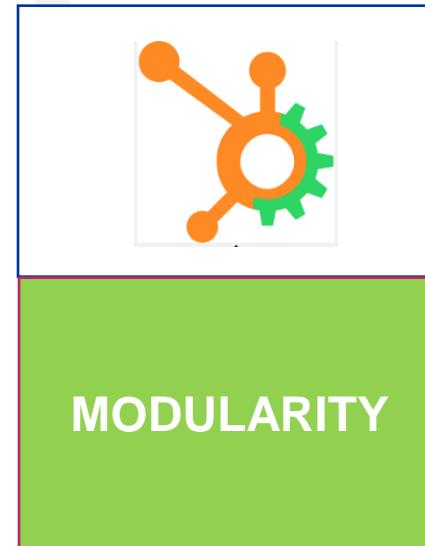
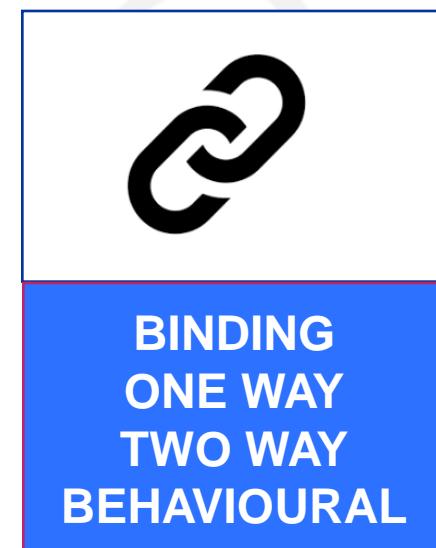


# AngularJS 2.0

- A JavaScript framework
- for building client-side application using HTML5, CSS3 and JavaScript(TypeScript)

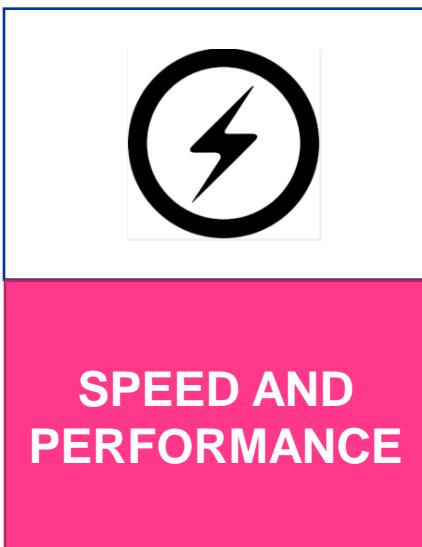
<http://eisenbergeffect.bluespire.com/all-about-angular-2-0/>

- Component based Programming
- application platform





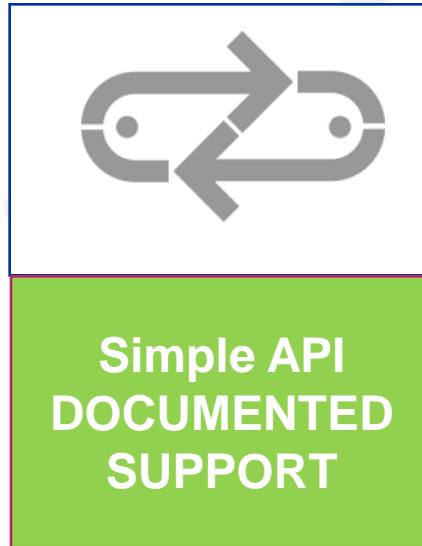
# Why AngularJS 2.0



SPEED AND PERFORMANCE



Support for ES6 Compliance



Simple API DOCUMENTED SUPPORT



PRODUCTIVITY TOOLS

- Applications can be written in
  - ES5
  - ES6
  - CoffeeScript
  - Dart
  - TypeScript

almost impossible to build a proper cross-browser site without help from something like jQuery

Web Components are on the horizon



# Angular 2 is Modular

Angular 2  
/Core

Angular 2  
/animate

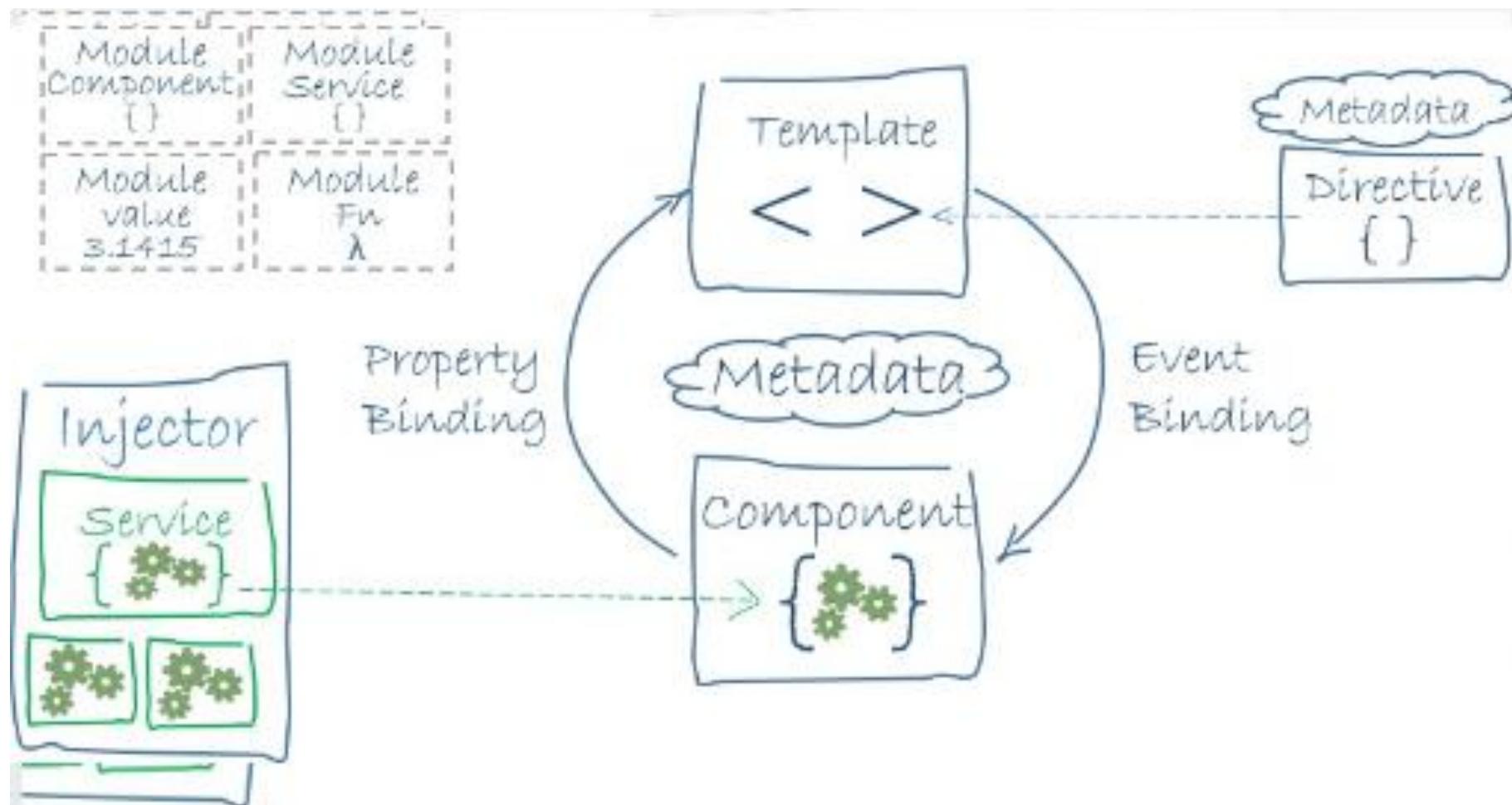
Angular 2  
/http

Angular 2  
/router





# Angular App Architecture



Source: <https://angular.io/docs/ts/latest/guide/architecture.html>





# Component System Architecture

- Components are small, encapsulated pieces of software that can be reused in many different contexts
- Angular 2 strongly encourages the component architecture by making it easy (and necessary) to build out every feature of an app as a component
- Angular components contain their own templates, styles and logic so that they can easily be ported elsewhere





# 8 MAIN BUILDING BLOCKS OF ANGULAR





**Webpack**

<http://webpack.github.io>

# WEB PACK

SYED AWASE

**MODULE BUNDLER**





# Why webpack?

- JavaScript is unorganized, not modularized, no automatic dependency injection, loosely structured, loosely typed language.
- Simplifies web development by solving a fundamental problem – bundling, transforming the assets to be easily consumed by the browser.
- Allows you to treat your project as a **dependency graph**.
- Webpack does all the preprocessing and gives the bundles needed through configuration.





# Task Runners vs Bundlers

## Task Runners

- Task runners such as Grunt and Gulp allow users to perform operations in a cross-platform manner, loading all assets at once.
- They are powerful means of putting assets together to serve specific functionality by loading dependent libraries.

## Bundlers

- Bundlers are needed when it is required to splice various assets together and produce bundles on the fly.
- Browserify, Brunch or Webpack are bundlers.
- JSPM pushes package management directly to the browser. It relies on System.JS a dynamic module loader.





# Webpack

- It is driven by configuration
- Webpack.config.js
- Configuration defines the inputs and the outputs of your project.
- It describes the types of transformations you perform. These transformations are defined using loaders and plugins each of which serves as a purpose of its own.

```
e:\CT\ReactJS>mkdir webpack-codeplay
```

```
e:\CT\ReactJS>cd webpack-codeplay
```

```
e:\CT\ReactJS\webpack-codeplay>npm init -y  
Wrote to e:\CT\ReactJS\webpack-codeplay\package.json:
```

```
{  
  "name": "webpack-codeplay",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC"  
}
```

---

```
npm i webpack --save-dev # or just -D if you want to save typing
```

```
e:\CT\ReactJS\webpack-codeplay\node_modules\.bin>webpack  
webpack 1.13.2
```

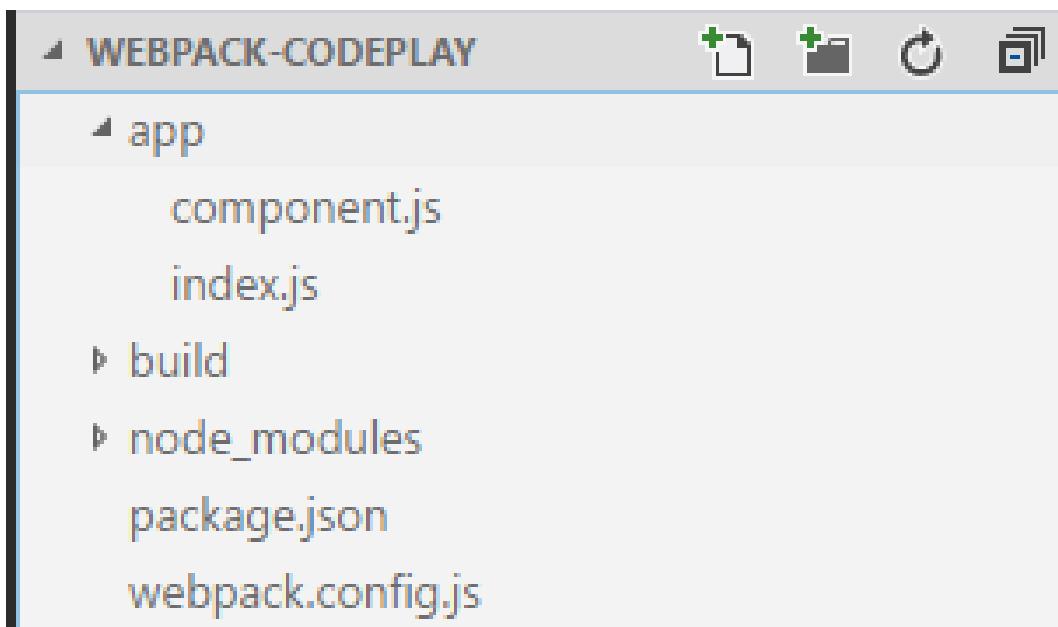
```
Usage: https://webpack.github.io/docs/cli.html
```





# Webpack

## Application-Directory Structure



## Webpack configuration

- Using **html-webpack-plugin** to wire up the generated assets with it.

```
e:\CT\ReactJS\webpack-codeplay>npm i html-webpack-plugin --save-dev
```

## Why use Webpack?

- Hot Module Replacement
- Bundle Splitting
- Asset Hashing
- Loaders and Plugins





# Webpack.config.js

```
1 const path = require('path');
2 const HtmlWebpackPlugin=require('html-webpack-plugin');
3 const PATHS ={
4     app:path.join(__dirname,'app'),
5     build:path.join(__dirname, 'build')
6 };
7 module.exports={
8     //Entry accepts a path or any object of entries
9     entry:{
10         app:PATHS.app
11     },
12     output:{
13         path:PATHS.build,
14         filename:'[name].js'
15     },
16     plugins:[
17         new HtmlWebpackPlugin({
18             title:'Webpack demo'
19         })
20     ]
21 };
```





# Webpack.config.js : alt config

- As the project grows, it becomes necessary to split it up per environment so as to have enough control over the build result.
- Maintaining configuration with a single file and branching using **webpack-merge**

```
e:\CT\ReactJS\webpack-codeplay>npm i webpack-merge --save-dev
```

```
1 const path = require('path');
2 const HtmlWebpackPlugin=require('html-webpack-plugin');
3 const merge = require('webpack-merge');
4 const PATHS ={
5   app:path.join(__dirname,'app'),
6   build:path.join(__dirname, 'build')
7 };
8 const common={
9   //Entry accepts a path or any object of entries
10  entry:{
11    app:PATHS.app
12  },
13  output:{
14    path:PATHS.build,
15    filename:'[name].js'
16  },
17  plugins:[
18    new HtmlWebpackPlugin({
19      title:'Webpack demo'
20    })
21  ]
22 };
23 var config;
24 switch(process.env.npm_lifecycle_event){
25   case 'build':
26     config=merge(common,{});
27     break;
28   case 'test':
29     config=merge(common,{});
30     break;
31   default:
32     config=merge(common,{});
33 }
34 module.exports = config;
```





# Integrating webpack-validator

- Validating your web-pack configuration against a schema and warn if we are trying to do something not sensible.

```
e:\CT\ReactJS\webpack-codeplay>npm i webpack-validator --save-dev
```

## Enable watch mode

- **webpack --watch**

```
const path = require('path');
const HtmlWebpackPlugin=require('html-webpack-plugin');
const merge = require('webpack-merge');
const validate=require('webpack-validator');
const PATHS ={
  app:path.join(__dirname,'app'),
  build:path.join(__dirname, 'build')
};

const common={
  //Entry accepts a path or any object of entries
  entry:{
    app:PATHS.app
  },
  output:{
    path:PATHS.build,
    filename:'[name].js'
  },
  plugins:[
    new HtmlWebpackPlugin({
      title:'Webpack demo'
    })
  ]
};
var config;
//detect how npm is run and branch based on that
switch(process.env.npm_lifecycle_event){
  case 'build':
    config = merge(common,{});
    break;
  case 'test':
    config = merge(common,{});
    break;
  default:
    config= merge(common,{});
}
module.exports =validate(config);
```



# Tools

LiveReload 2 proudly presents...

## The Web Developer Wonderland

(a happy land where browsers don't need a Refresh button)

CSS edits and image changes apply live.  
CoffeeScript, SASS, LESS and others just work.

Citizenship is granted through the Mac App Store.  
Windows permanent residency issues are being worked out,  
temporary stay already allowed.

Contact Support

MONITORED FOLDERS

- tools
- Web View Resources
- MWCPerfMon
- lib
- backend
- compass\_960

site  
-/Dropbox/Projects/LiveReload

Insert this snippet before </body> or [install browser extensions](#):

```
npm install -g browser-sync
```

## Browsersync ([www.browsersync.io](https://www.browsersync.io))

Browsersync Documentation Community

Time-saving synchronised browser testing.  
It's wicked-fast and totally free.

npm install -g browser-sync

Get Started

Your indispensable test





# Webpack-dev-server

e:\CT\ReactJS\webpack-codeplay>npm i webpack-dev-server --save-dev

- Webpack-dev-server is a development server running in-memory. It refreshes content automatically in the browser, while you develop your application.
- Also supports an advanced Webpack feature known as **Hot Module Replacement (HMR)** which provides a way to patch the browser state without a full refresh.

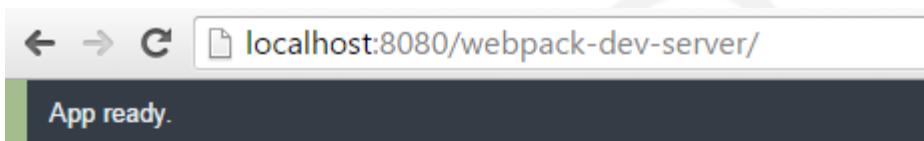
```
package.json x webpack.config.js Default Settings index.js component.js
1  {
2    "name": "webpack-codeplay",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\"Error: no test specified\\" && exit 1",
8      "start": "webpack-dev-server",
9      "build": "webpack"
10 },
11 "keywords": [],
12 "author": "",
13 "license": "ISC",
14 "devDependencies": {
15   "html-webpack-plugin": "^2.22.0",
16   "webpack": "^1.13.2",
17   "webpack-dev-server": "^1.15.1",
18   "webpack-merge": "^0.14.1",
19   "webpack-validator": "^2.2.7"
20 }
21 }
```





# Running your application with Webpack

```
e:\CT\ReactJS\webpack-codeplay>npm build  
  
e:\CT\ReactJS\webpack-codeplay>npm start  
  
> webpack-codeplay@1.0.0 start e:\CT\ReactJS\webpack-codeplay  
> webpack-dev-server  
  
http://localhost:8080/webpack-dev-server/
```





<https://webpack.github.io/docs/comparison.html>

## COMPARISON

Feature	webpack/webpack	jrburke/requirejs	substack/node-browserify	jspm/jspm-cli	rollup/rollup
CommonJS <code>require</code>	yes	only wrapping in <code>define</code>	yes	yes	commonjs-plugin
CommonJS <code>require.resolve</code>	yes	no	no	no	no
CommonJS <code>exports</code>	yes	only wrapping in <code>define</code>	yes	yes	commonjs-plugin
AMD <code>define</code>	yes	yes	deamdfy	yes	no
AMD <code>require</code>	yes	yes	no	yes	no
AMD <code>require</code> loads on demand	yes	with manual configuration	no	yes	no
ES2015 <code>import / export</code>	yes(vr. 2)	no	no	yes	yes
Generate a single bundle	yes	yes♦	yes	yes	yes
Load each file separate	no	yes	no	yes	no
Multiple bundles	yes	with manual configuration	with manual configuration	yes	no
Additional chunks are loaded on demand	yes	yes	no	System.import	no
Multi pages build with common	with manual		with manual		





# SourceMap

- To enable sourcemaps during development, we can use a default known as eval-source-map

Webpack.config.js

```
//detect how npm is run and branch based on that
switch(process.env.npm_lifecycle_event){
  case 'build':
    config = merge(common,{devtool:'eval-source-map'});
    break;
  case 'test':
    config = merge(common,{devtool:'eval-source-map'});
    break;
  default:
    config = merge(common,{devtool:'eval-source-map'});
}
module.exports =validate(config);
```



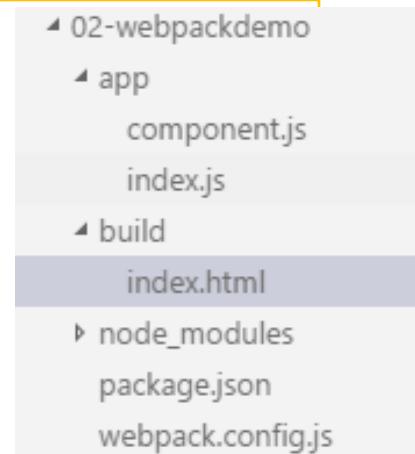


# Webpack Application: Playbook

**STEP: I**

```
npm init -y
npm i webpack --save-dev
npm i webpack-merge --save-dev
npm i webpack-validator --save-dev
npm i html-webpack-plugin --save-dev
npm i webpack-dev-server --save-dev
npm i npm-install-webpack-plugin --save-dev
```

**STEP: II**





# Webpack Application: Playbook

STEP: III

- create webpack.config.js

```
const path = require('path');
const HtmlWebpackPlugin=require('html-webpack-plugin');
const merge = require('webpack-merge');
const validate=require('webpack-validator');
const PATHS ={
    app:path.join(__dirname,'app'),
    build:path.join(__dirname, 'build')
};
const common={
    //Entry accepts a path or any object of entries
    entry:{
        app:PATHS.app
    },
    output:{
        path:PATHS.build,
        filename:'[name].js'
    },
    plugins:[
        new HtmlWebpackPlugin({
            title:'Webpack demo'
        })
    ]
};
var config;
//detect how npm is run and branch based on that
switch(process.env.npm_lifecycle_event){
    case 'build':
    config = merge(common,{});
    break;
    case 'test':
    config = merge(common,{});
    break;
    default:
    config= merge(common,{});
}
module.exports =validate(config);
```





# Webpack Application: Playbook

**STEP: IV:**  
app/component.js

```
module.exports = function(){
  var element = document.createElement('h1');
  element.innerHTML ='Hello Syed Awase';
  return element;
};
```

**STEP: V:**  
app/index.js

```
var component = require('./component');
var app = document.createElement('div');
document.body.appendChild(app);
app.appendChild(component());
```





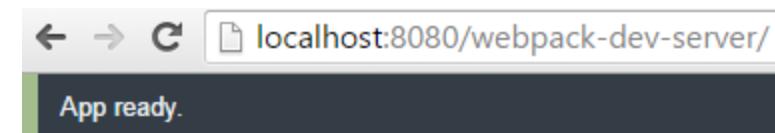
# Webpack Application:Playbook

**STEP: VI:**  
build/index.js

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>WebPack Second Demo</title>
</head>
<body>
<div id="app"></div>
<script src=".bundle.js"></script>
</body>
</html>
```

**STEP: VII:**

```
e:\CT\ReactJS\webpack-codeplay\02-webpackdemo>npm start
> 02-webpackdemo@1.0.0 start e:\CT\ReactJS\webpack-codeplay\02-webpackdemo
> webpack-dev-server --content-base build
http://localhost:8080/webpack-dev-server/
```



**RESULT =>**

Hello Syed Awase





# SystemJS: Universal dynamic module loader

Section VI





# JSPM and SystemJS

- ES6 brings a **standard module construct** to JavaScript and a loader specification.

[www.jspm.io](http://www.jspm.io)

The screenshot shows the official website for jspm. It features a large yellow cube logo followed by the text "jspm.io". Below this is the tagline "Frictionless browser package management". A "Fork me on GitHub" button is visible in the top right corner. The main content area lists several key features:

- jspm is a package manager for the [SystemJS universal module loader](#), built on top of the dynamic [ES6 module loader](#)
- Load any module format (ES6, AMD, CommonJS and globals) directly from any registry such as [npm](#) and [GitHub](#) with flat versioned dependency management. Any custom registry endpoints can be created through the Registry API.
- For development, load modules as separate files with ES6 and plugins compiled in the browser.
- For production (or development too), optimize into a bundle, layered bundles or a self-executing bundle with a single command.

<https://github.com/systemjs/systemjs>

The screenshot shows the GitHub repository page for "systemjs/systemjs". The repository has 1,618 commits, 7 branches, 125 releases, 70 contributors, and is licensed under MIT. The commit history is listed below, showing recent activity from "guybedford".

Commit	Author	Message	Date
bench	guybedford	js deps extraction benchmark	4 months ago
dist		0.19.39	27 days ago
docs		docs/config-api: babelOptions: stage no longer exists; replaced examp...	18 days ago
lib		remove redundant \$__curScript assignment path	18 days ago
test		0.19.37	2 months ago
.ignore		Ag search ignores dist	2 years ago
.gitignore		0.13.2	2 years ago
.travis.yml		add node 6	6 months ago
LICENSE		transpiler tweaks	10 months ago





# JSPM

- Jspm is a package manager for the SystemJS universal module loader, built on top of the dynamic ES6 module loader.
- Loads any module format (ES6, AMD, CommonJS and globals)
- `npm install jspm –save-dev`
- `jspm init` or `jspm init -y`
- `jspm init –p`
- `jspm –v`





```
E:\CT\Angular2\Scenarios-A2-1.0\1-JSPMDemos\1-basicjspm>jspm init
Would you like jspm to prefix the jspm package.json properties under jspm? [yes]:
Enter server baseURL (public folder path) [./]:
Enter jspm packages folder [.\jspm_packages]:
Enter config file path [.\config.js]:jspmconfig.js
Configuration file jspmconfig.js doesn't exist, create it? [yes]:
Enter client baseURL (public folder URL) [/]:
Do you wish to use a transpiler? [yes]:
Which ES6 transpiler would you like to use, Babel, TypeScript or Traceur? [babel]:Traceur
ok  Verified package.json at package.json
    Verified config file at jspmconfig.js
    Looking up loader files...
```





# SystemJS

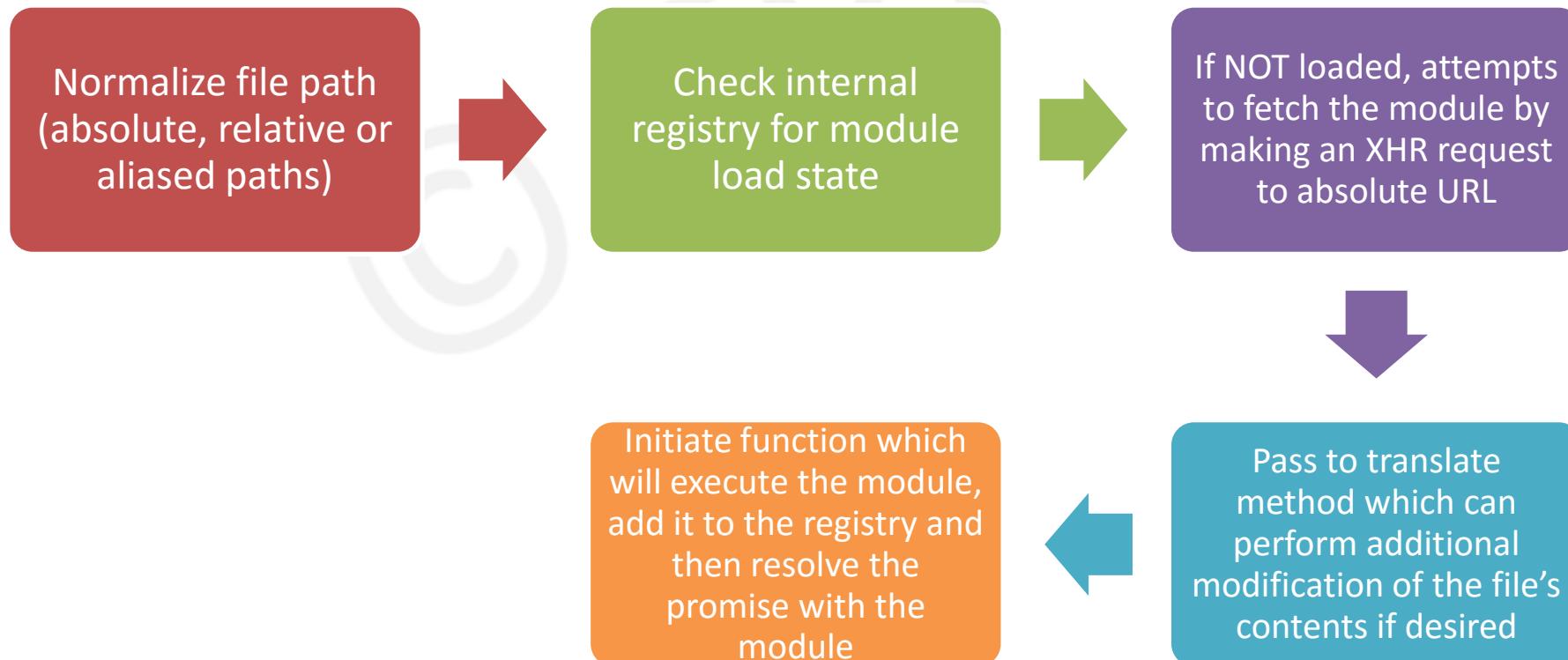
- Universal dynamic module loader to load ES6 modules, AMD and CommonJS and global scripts in the browser
  - Loads any module format with exact circular reference and binding support
  - Loads ES6 modules compiled into the System.register bundle format for product
  - Supports RequireJS-style map, paths, bundles and global shims
  - Loader plugins allow custom transpilation or asset loading.
- SystemJS is a module loader built on top of the original ES6 module loader polyfill and soon to be the polyfill for the WhatWG (web hypertext application technology working group) module loader.





# SystemJS

- It loads files in order from top to bottom and **then instantiates from bottom to top**.





# Simple jspm example

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Document</title>
  </head>
  <body>
    <div>
      <h1>JSPM Basic Demo</h1>
    </div>
    <script src="jspm_packages/system.js"></script>
    <script>
      System.import("lib/main.js");
    </script>
  </body>
</html>
```

- main.js

```
1  console.log("dynamically loaded");
2  console.log("jspm first demo");
```

RUN

E:\CT\Angular2\Scenarios-A2-1.0\1-JSPMDemos\1-basicjspm>live-server





# omdbApi Example-1

MAIN.JS

```
icjspm-notes.txt main.js x
1 import OmdbApi from './omdb-api'
2 import ExtractPosters from './ExtractPoster'
3 import DisplayPosters from './DisplayPoster'
4 console.log("dynamically loaded");
5 console.log("jspm first demo");
6 OmdbApi.load()
7 .then(ExtractPosters)
8 .then(DisplayPosters)
9
10 export default {}
```

OmdbApi.js

```
1-basicjspm-notes.txt main.js x
1 import jsonp from 'jsonp'
2 class OmdbApi{
3   constructor(){
4     this.Url="http://www.omdbapi.com/?s=batman"
5   }
6   load(){
7     return new Promise((resolve,reject)=>{
8       jsonp(this.Url, {}, (err,data)=>{
9         err? reject(err):resolve(data.Search)
10        console.log(data)
11      })
12    })
13  }
14 }
15 export default new OmdbApi()
```





# OmdbApi Example -1

ExtractPoster.js

```
1-basicjspm-notes.txt  main.js  omdb-api.js  ExtractPoster.js ×
1  export default(posts)=>{
2      console.log("ExtractPoster")
3      console.log(posts)
4      return posts.map(post=>post.Poster)
5 }
```

DisplayPoster.js

```
1-basicjspm-notes.txt  main.js  omdb-api.js  DisplayPoster.js ×
1  export default(Posters)=>{
2      var elem = document.querySelector("#poster")
3      console.log("DisplayPoster")
4      console.log(Posters)
5      elem.innerHTML=Posters.map(
6          Poster=>``)
7          .join("\n")
8
9 }
```

Index.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Document</title>
6  </head>
7  <body>
8      <div>
9          <h1>JSPM Basic Demo</h1>
10         <div id="poster"></div>
11     </div>
12     <script src="jspm_packages/system.js"></script>
13     <script src="jspmconfig.js"></script>
14     <script>
15         System.import("lib/main.js");
16     </script>
17 </body>
18 </html>
```



# SETTING UP....PART-I : BUILDING THE MINIMAL ANGULAR APP ANGULAR 2

**STEP I**

## Creating the basic structure for a minimal angular 2 application

```
>mkdir minimalapp  
·cd minimalapp
```

```
E:\CT\Angular2\Scenarios-A2-1.0\1-JSPMDemos\minimalapp>npm init -y  
Wrote to E:\CT\Angular2\Scenarios-A2-1.0\1-JSPMDemos\minimalapp\package.json
```

**STEP II**

- a. typical Angular project needs several configuration files:  
*package.json, tsconfig.json and systemjs.config.js*
- b. *Copy the code as instructed from  
<https://angular.io/docs/ts/latest/quickstart.html>*
- c. *Remove “~” or “^” from dependencies or devdependencies  
from package.json*





# Role of Package.json: Dependencies

*helps in identifying npm package dependencies for the project*

Package/library	Version	
@angular/common	2.1.1	Commonly needed services pipes and directives provided by the Angular team
@angular/compiler	2.1.1	
@angular/core	2.1.1	
@angular/forms	2.1.1	
@angular/http	2.1.1	
@angular/platform-browser	2.1.1	
@angular-platform-browser-dynamic	2.1.1	
@angular/router	3.1.1	
@angular/upgrade	2.1.1	





# Role of Package.json: Dependencies

*helps in identifying npm package dependencies for the project*

Package/library	Version	
Angular-in-memory-web-api	0.1.13	
Bootstrap	3.3.7	For rendering view (bootstrap –mobile first)
Core-js	2.4.1	brings ES2015/ES6 capabilities to ES5 browsers
Reflect-metadata	0.1.8	
Rxjs	5.0.0-beta.12	Reactive js extensions
Systemjs	0.19.39	Module loader and packaging
Zone.js	0.6.25	





# Role of Package.json:DevDependencies

*helps in identifying npm package dependencies for the project*

Package/Libraries	Version	
@types/core-js	0.9.34	
@types/node	6.0.45	
Concurrently	3.0.0	
Lite-server	2.2.2	
TypeScript	2.0.3	





## @angular/common

- Commonly needed services pipes and directives provided by the Angular team

## @angular/compiler

- Angular's Template Compiler. It understands templates and can convert them to code that makes the app run and render.
- No direct interaction with compiler directly.
- Called when they use **platform-browser-dynamic** or the **offline template compiler**





## @angular/platform-browser

- Everything DOM and browser related, especially the pieces that help render into DOM.
- This package also includes the **bootstrapStatic** method for bootstrapping applications for production builds that pre-compile templates offline

## @angular/platform-browser-dynamic

- Providers and a bootstrap method for applications that compile templates on the client.
- We use this package for bootstrapping during development .





## @angular/http

- Angular's http client

## @ angular/upgrade

- A set of utilities for upgrading Angular 1.x applications

## @ angular/router

- Component router

## @ angular/system.js

- A dynamic module loader compatible with ES2015 module specification.





## Polyfill Packages

- Polyfills plug gaps in the browser's javascript implementations.
- Angular requires certain polyfills in the application environment.
- Core-js : monkey patches the global context (`window`) with essential features of ES2015(ES6).
- Reflect-metadata : a dependency shared between Angular and the TypeScript compiler.
- Rxjs- A polyfill for the observables specification currently before the TC39 committee.

## Zone.js

- A polyfill for the **zone specification currently before the TC39 committee that determines standards for the JavaScript language.**
- can be compared to the equivalent of a thread-local context in Java.





# Helper Libraries

## Angular2-in-memory-web-api

- An angular-supported library that simulates a remote server's web api without requiring an actual server or real http class.

## bootstrap

- A popular HTML and CSS framework for designing responsive web apps.





# Role of tsconfig.json

*defines how the TypeScript compiler generates JavaScript from the project's files.*

Configuration Property	Value	Explanation
CompilerOptions		
Target	Es5	
Module	Commonjs	
ModuleResolution	Node	
sourceMap	True	
emitDecoratorMetadata	True	
experimentalDecorators	True	
removeComments	False	
<b>nolImplicitAny</b>	<b>False</b>	





# devDependencies

- Concurrently – a utility to run multiple npm commands concurrently on OS/X, windows and linux O.S
- Lite-server – a light weight static file server, written and maintained by John papa with excellent support for angular apps that use routing.
- TypeScript – tsc typescript compiler
- typings – A manager for typescript definition files.



**STEP III**

```
E:\CT\Angular2\Scenarios-A2-1.0\1-JSPMDemos\minimalapp>npm install
```

```
▲ minimalapp
  ▶ node_modules
    package.json
    systemjs.config.js
    tsconfig.json
```

**STEP IV**

```
▲ minimalapp
  ▲ app
    app.component.ts
    app.module.ts
    main.ts
  ▶ node_modules
    index.html
    package.json
    style.css
    systemjs.config.js
    tsconfig.json
```

Follow the instructions outlined at <https://angular.io/docs/ts/latest/quickstart.html>

```
1-JSPMDemos\minimalapp>mkdir app
```





STEP V

```
E:\CT\Angular2\Scenarios-A2-1.0\1-JSPMDemos\minimalapp>npm start
```

```
> minimalapp@1.0.0 start E:\CT\Angular2\Scenarios-A2-1.0\1-JSPMDemos\minimalapp  
> tsc && concurrently "tsc -w" "lite-server"
```



# My First Angular App





# Running NPM scripts

1. Running **npm scripts**
  1. **Npm start** -> runs the compiler and a server at the same time in watch mode.
  2. **Npm run tsc** -> runs the typescript compiler once
  3. **Npm run tsc:w** -> runs the typescript compiler in watch mode, the process keeps running, awaiting changes to the typescript files and recompiling when it sees them
  4. **Npm run lite** – runs the lite-server – a light weight, static file server
  5. **npm, run typings** – runs the typings tool separately
  6. **Npm run post-install**





# Angular 2 Dev Env Setup

[https://github.com/adeveloperdiary/angular2\\_setup](https://github.com/adeveloperdiary/angular2_setup)

```
git clone https://github.com/adeveloperdiary/angular2 setup.git
```

PS E:\CT\Angular2\CodePlay\angular2\_setup-master\angular2\_setup-master>ls

Directory: E:\CT\Angular2\CodePlay\angular2\_setup-master\angular2\_setup-master

Mode	LastWriteTime	Length	Name
d----	5/3/2016 12:44 PM		src
----	5/3/2016 12:44 PM	1164	package.json
----	5/3/2016 12:44 PM	385	README.md
----	5/3/2016 12:44 PM	419	tsconfig.json
----	5/3/2016 12:44 PM	298	typings.json
----	5/3/2016 12:44 PM	1797	webpack.config.js

```
PS E:\CT\Angular2\CodePlay\angular2_setup-master\angular2_setup-master> npm install -g webpack-dev-server typings typescript fetchMetadata -> network | | #####-----|
```

```
PS E:\CT\Angular2\CodePlay\angular2_setup-master\angular2_setup-master> npm install
```

```
PS E:\CT\Angular2\CodePlay\angular2_setup-master\angular2_setup-master> npm start
```

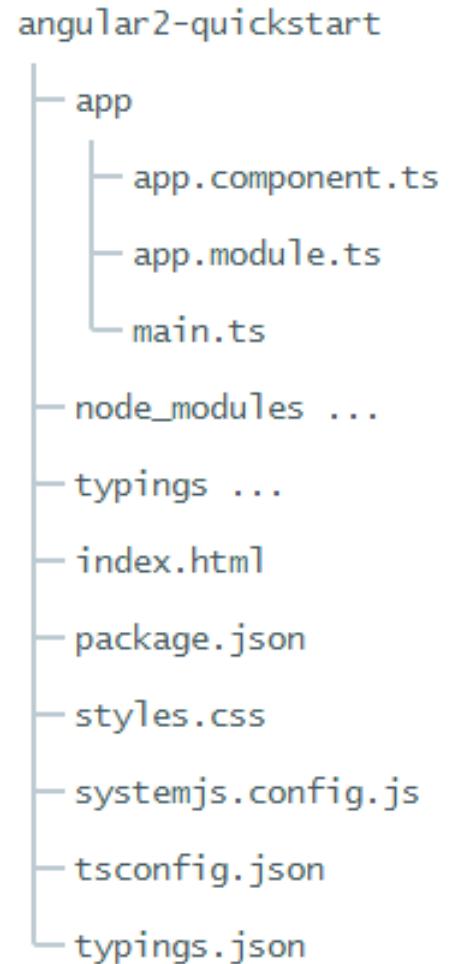
```
> setup_angular2@1.0.0 start E:\CT\Angular2\CodePlay\angular2_setup-master\angular2_setup-master  
> npm run server
```





# Step 4: first angular component

- mkdir app
- create the component file  
app/app.component.ts
- create the file  
app/app.module.ts





# A2.0 View

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular 2 Styling Techniques</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- 1. Load libraries -->
    <!-- IE required polyfills, in this exact order -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.16/system-polyfills.js"></script>

    <script src="https://code.angularjs.org/2.0.0-beta.3/angular2-polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.3/Rx.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.3/angular2.dev.js"></script>

    <!-- 2. Configure SystemJS -->
    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}}
      });
      System.import('app/main')
        .then(null, console.error.bind(console));
    </script>

  </head>
  <!-- 3. Display the application -->
  <body>
    <my-app>Loading...</my-app>
  </body>
</html>
```





# Core Libraries

```
<!-- 1. Load libraries -->
<!-- IE required polyfills, in this exact order -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.16/system-polyfills.js"></script>

<script src="https://code.angularjs.org/2.0.0-beta.3/angular2-polyfills.js"></script>
<script src="https://code.angularjs.org/tools/system.js"></script>
<script src="https://code.angularjs.org/tools/typescript.js"></script>
<script src="https://code.angularjs.org/2.0.0-beta.3/Rx.js"></script>
<script src="https://code.angularjs.org/2.0.0-beta.3/angular2.dev.js"></script>
```





# SystemJS Configuration

```
<!-- 2. Configure SystemJS -->
<script>
  System.config({
    transpiler: 'typescript',
    typescriptOptions: { emitDecoratorMetadata: true },
    packages: {'app': {defaultExtension: 'ts'}}})
  System.import('app/main')
    .then(null, console.error.bind(console));
</script>
```





# Angular 2 webpack env setup

<https://github.com/AngularClass/angular2-webpack-starter>

© TPRI





# Building Blocks of Angular 2

Section VI





# BASIC BUILDING BLOCKS OF ANGULAR 2

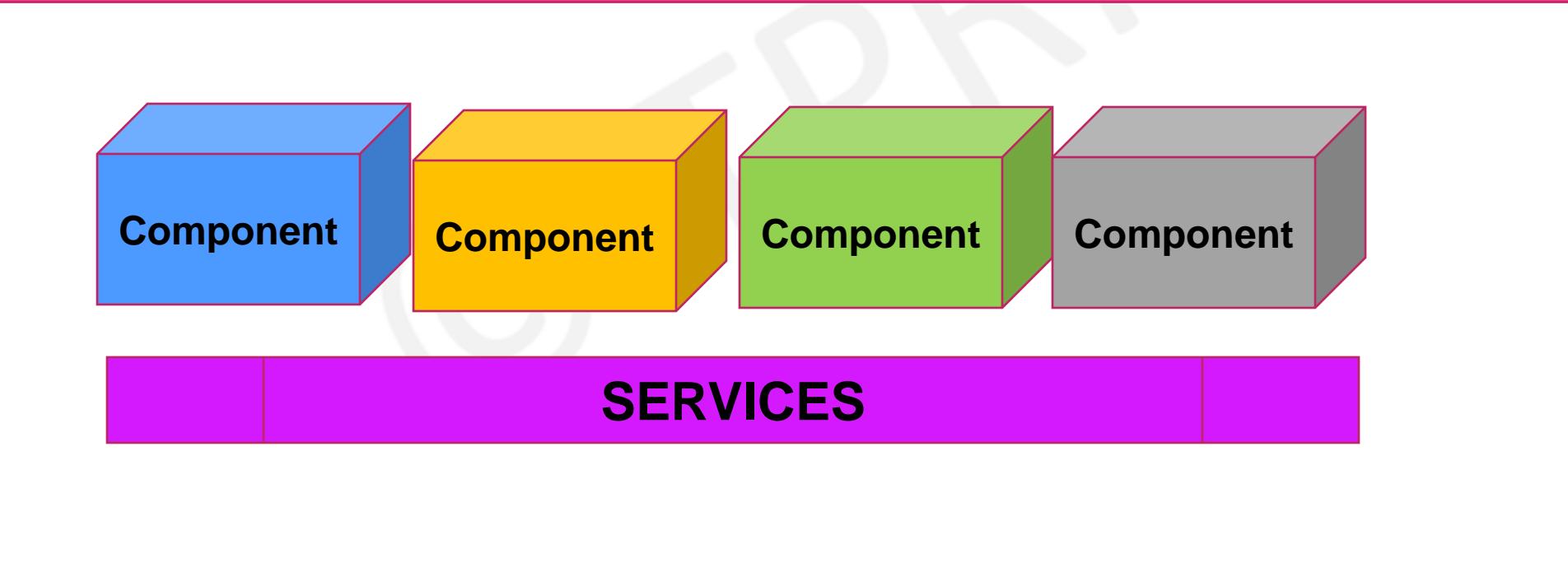
1. Modules
2. Components
3. Templates
4. Metadata
5. Data binding
6. Directives/Structural Directives
7. Services
8. Dependency injection

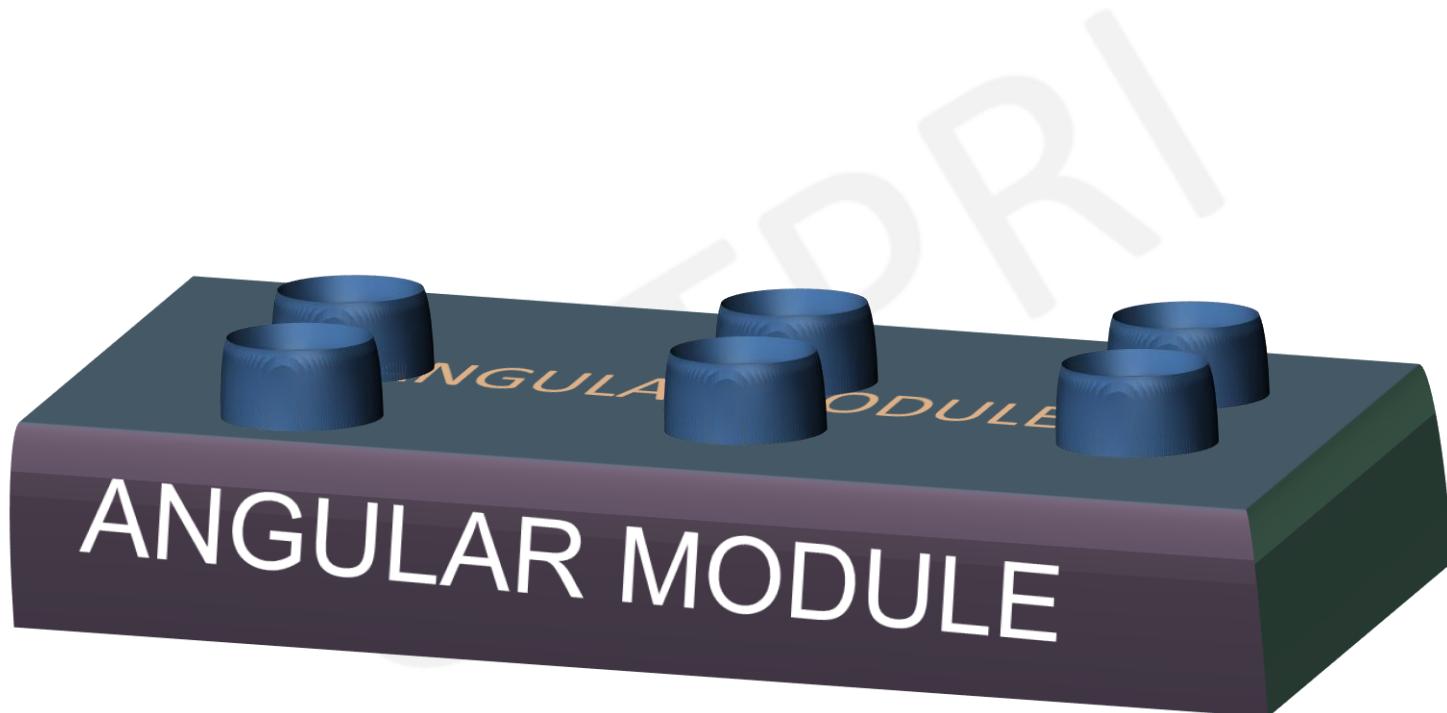




# Anatomy of an Angular 2 Application

## Container Component







# AngularModule

- A great way to organize the application and extend it with capabilities from external libraries
- Angular Libraries
  - FormsModule
  - HttpClientModule
  - RouterModule
- Third Party
  - Material Design
  - Ionic
  - AngularFire2
- They consolidate components, directives and pipes in cohesive blocks of functionality.
- Modules can also add services to the application
- Modules can be loaded eagerly when the application starts or lazy loaded asynchronously by the router





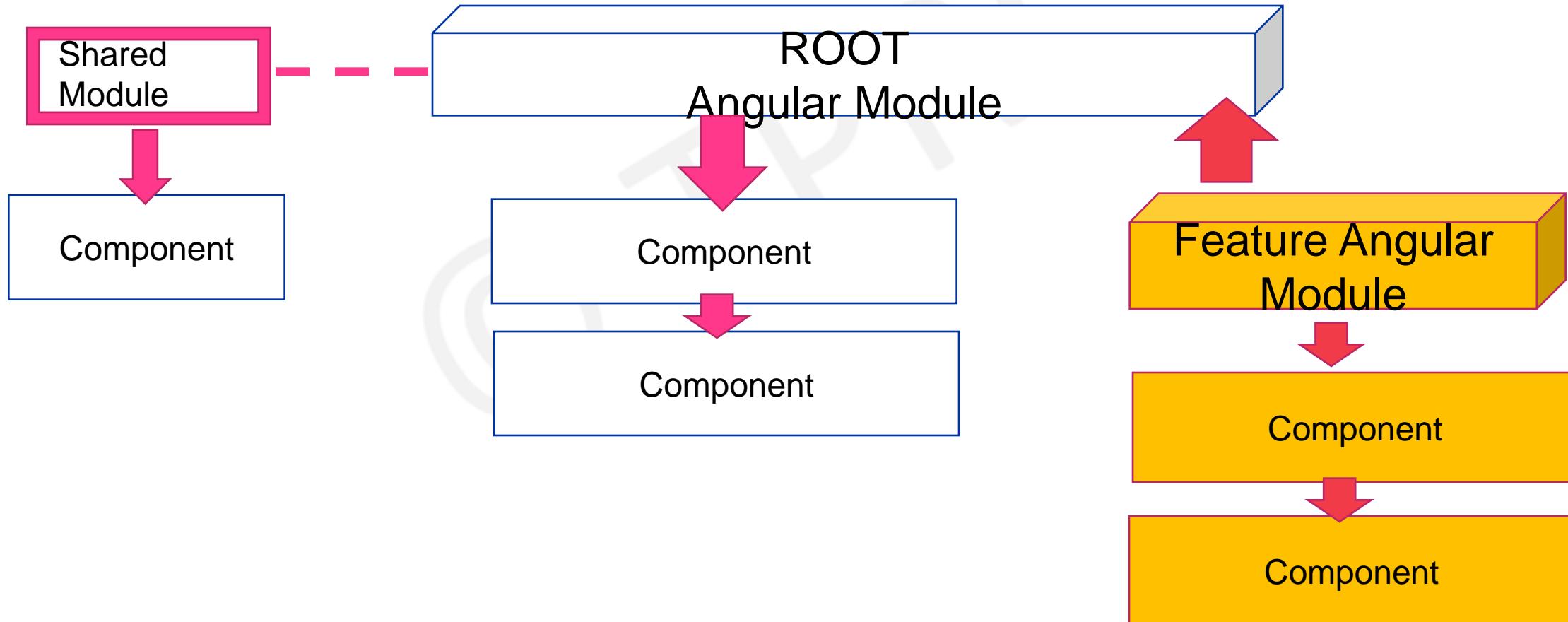
# AngularModule

- An angular module is a class decorated with **@NgModule metadata**
  - **Declare which components, directives and pipes belong to the module**
  - **Make some of those classes public so that other component templates can use them**
  - **Import other modules with components, directives and pipes needed by the components in this module**
  - **Provide services at the application level that any application component can use.**
- Every Angular app has **atleast** one module class the **root module**.





# Angular Modules





# Module

- We use ES6 style modules with Angular 2
- Export tells TypeScript that the resource is a module available for other modules
- Import tells TypeScript the resource in a module.
- Angular ships a collection library modules.

- We assemble our application from modules
- A module exports an asset such as a Service, Component, or a shared value.
- Assets can be exported using the **export** keyword.
- Modules and their contents can be imported using **import** keyword





# ES Modules

- Code files that import or export something
- Organize our code files
- Modularize our code
- Promote code reuse

# Angular Modules

- Code files that organize the application into cohesive blocks of functionality
- Organize our application
- Modularize our application
- Promote application boundaries.





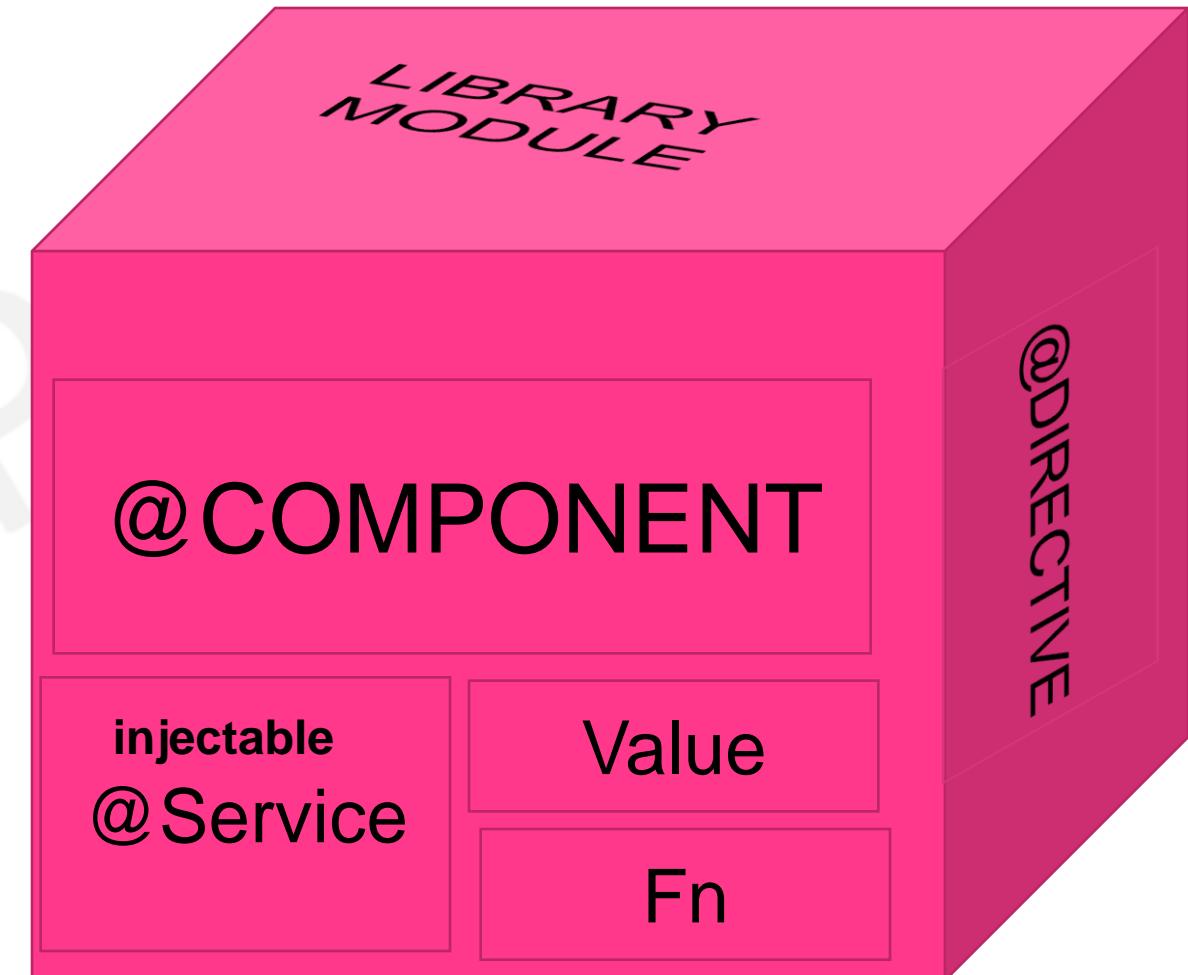
# Angular Module

- Angular ships as a collection of JavaScript Modules, You can think of them as library modules
- Each angular library name begins with **@angular prefix**

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

import { BrowserModule } from '@angular/platform-browser';

imports:      [ BrowserModule ],
```





# NgModule

- A decorator function that takes a single metadata object whose properties describe the module.
  - Declarations: the view classes that belong to this module. Angular has **three kinds of view classes**
    - a. Components
    - b. Directives
    - c. Pipes
  - Exports: the subset of declaration that should be visible and usable in the component templates of other modules
  - Imports: other modules whose exported classes are needed by component templates declared in this module

- Providers: creators of services that this module contributes to the global collection of services, they become accessible in all parts of the app.
- Bootstrap: the **main application view, called the root component, that hosts all other app views. Only the root module should set this bootstrap property.**
- BrowserModule, the module every browser app must import. It also includes some of the common directives like NgIf and NgFor.
- declarations is a list of components, directives, and pipes belonging to the module.
- imports is a list of components, directives, and pipes needed and used by this module.
- bootstrap indicates that the application has start at the AppComponent which will replace a correspondingly named directive in index.html





# app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

**Import other modules**

**Exported to be used else where**





# Bootstrapping Angular Application

Angular offers a variety of bootstrapping options (launch application by bootstrapping the AppModule in main.ts)

## JUST-IN-TIME COMPILER

Angular compiler compiles the application  
In the browser

```
// The browser platform with a compiler
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

// The app module
import { AppModule } from './app.module';

// Compile and launch the module
platformBrowserDynamic().bootstrapModule(AppModule);
```

## AHEAD-OF-TIME COMPILER

Static Alternative which can produce a much smaller application that launches faster, especially for mobile devices and high latency networks

```
// The browser platform without a compiler
import { platformBrowser } from '@angular/platform-
browser';

// The app module factory produced by the static offline
compiler
import { AppModuleNgFactory } from
'./app.module.ngfactory';

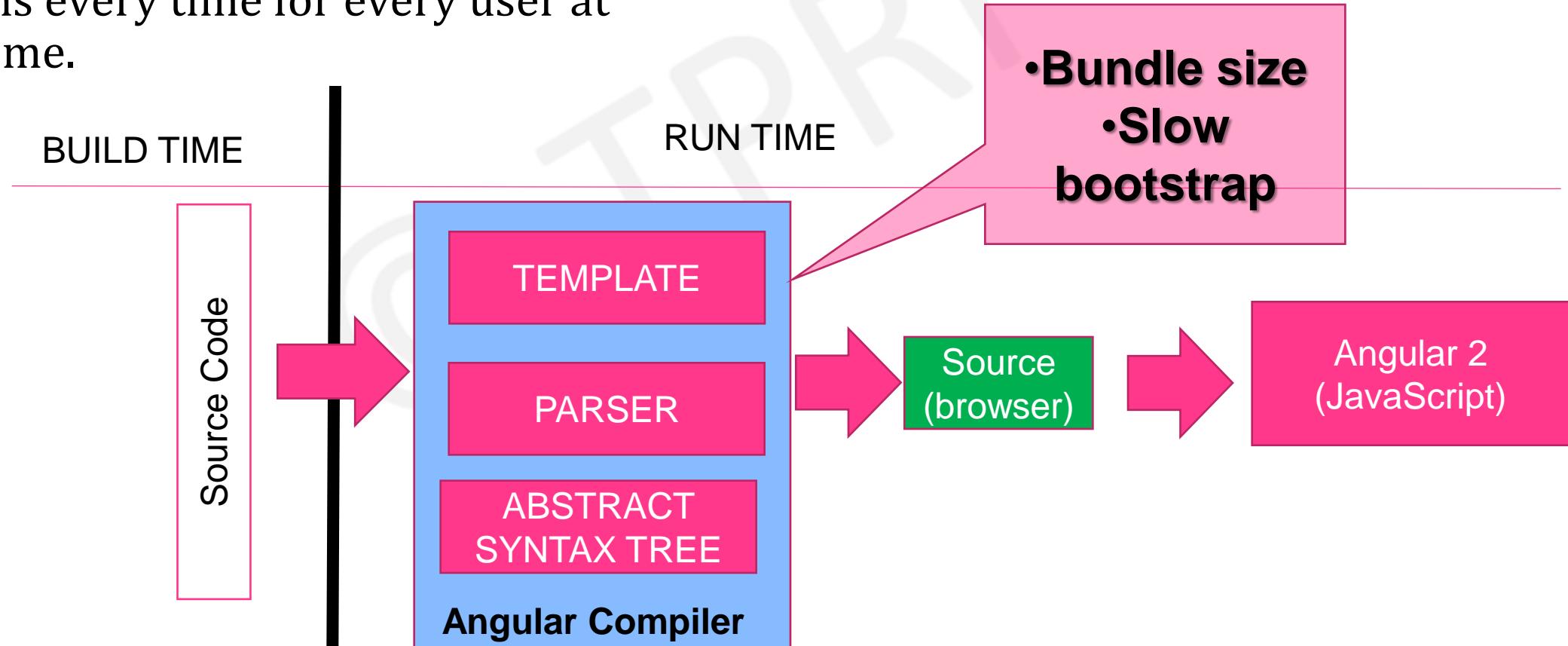
// Launch with the app module factory.
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```





# Just-in-Time Compiling: Angular2

- Runs every time for every user at runtime.

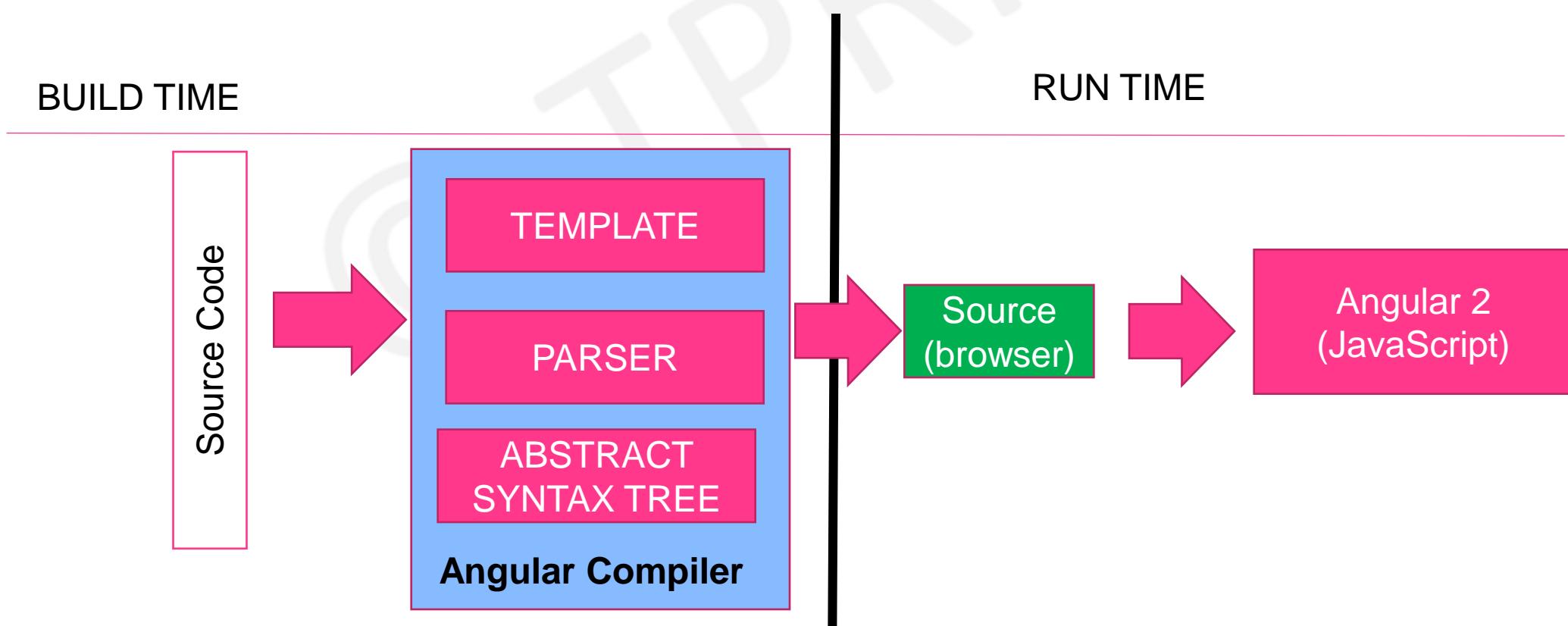


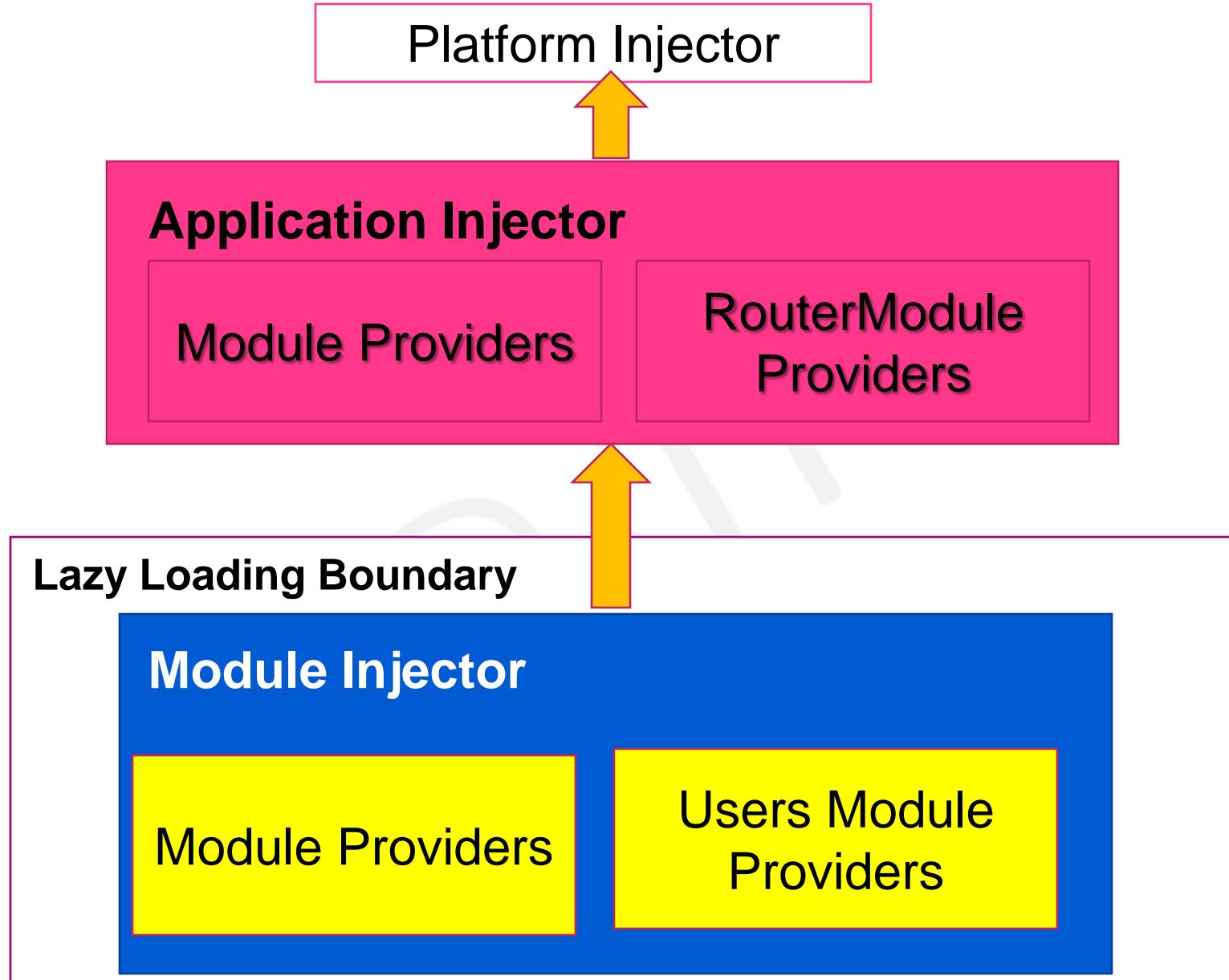


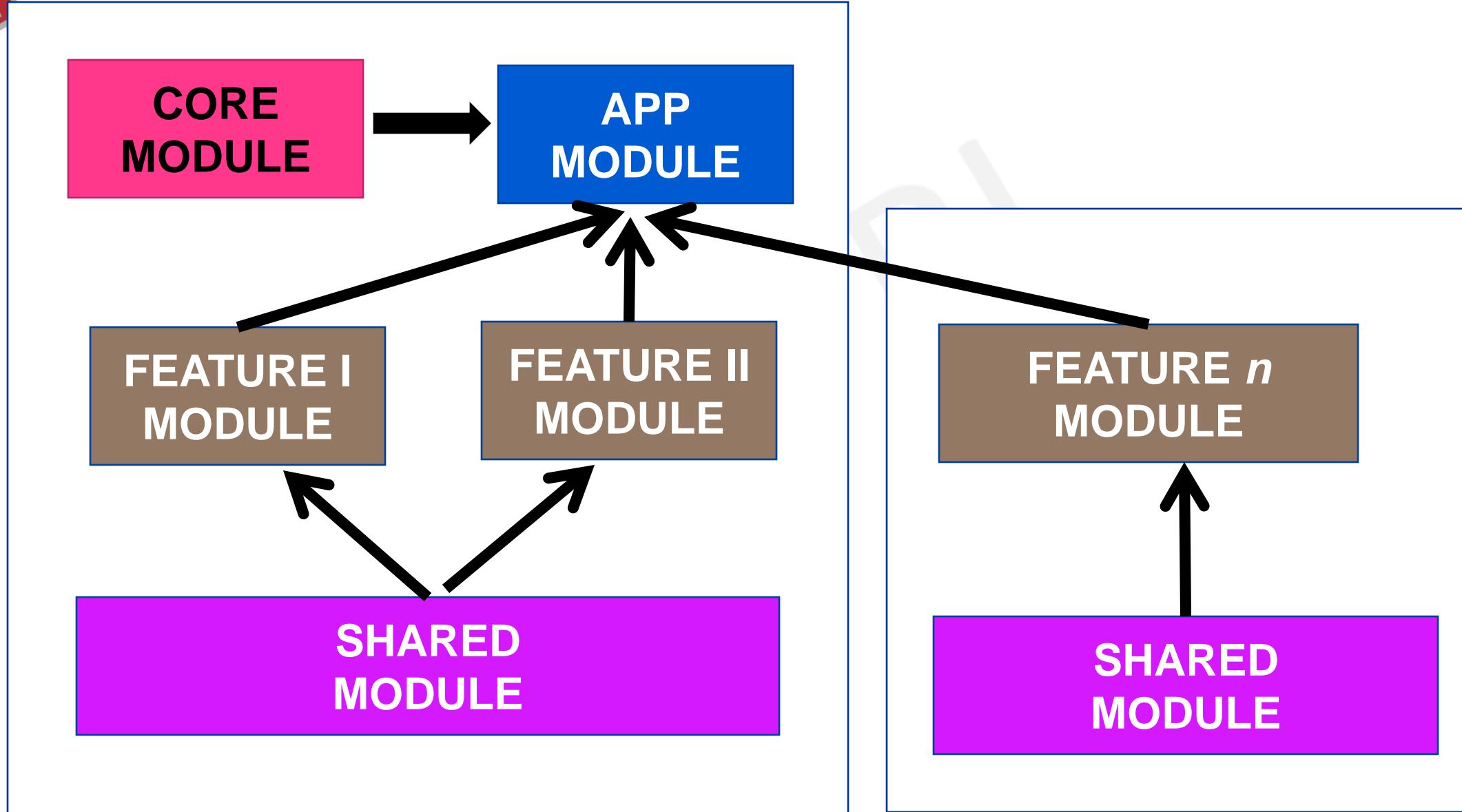
# Ahead of Time Compiling :Angular 2

- Run once at build time

```
// Launch with the app module factory.  
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

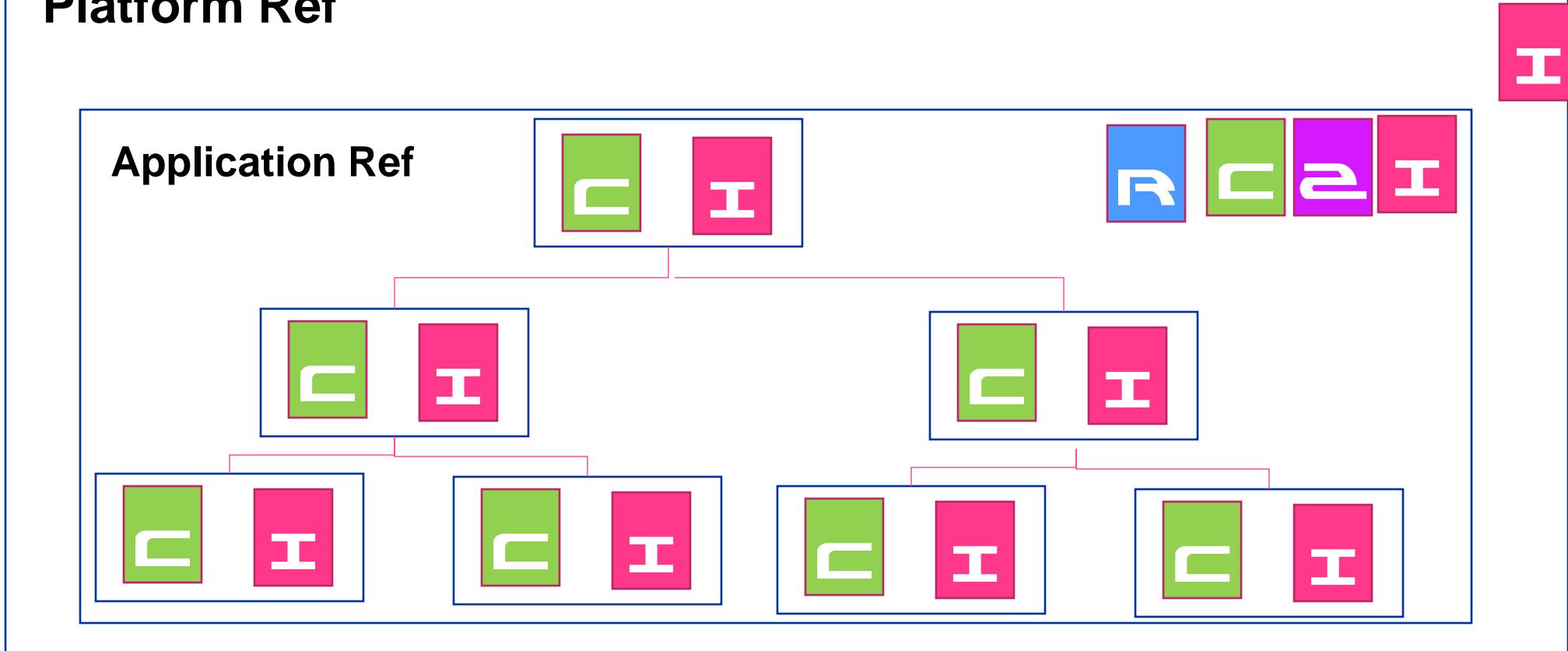








## Platform Ref



**Injector**



**Zone**



**Change Detection**



**Renderer**





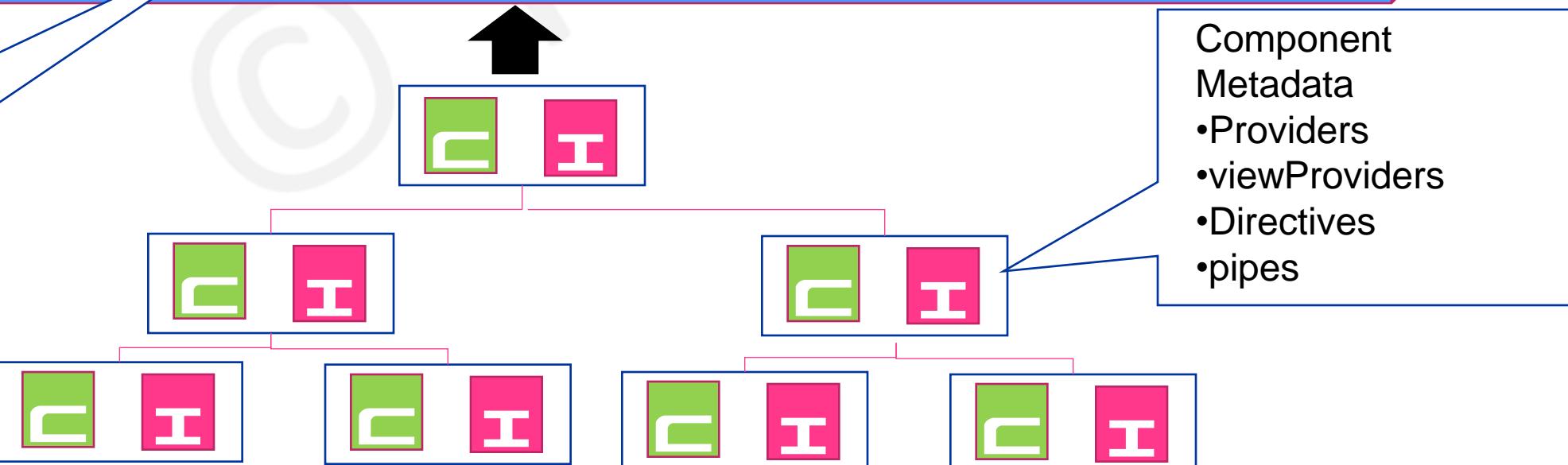
# Injectors Tree

- Platform\_initializer
- Reflector
- Console



- Platform\_pipes
- Platform\_directives
- Form\_providers
- Document
- DomRootRenderer

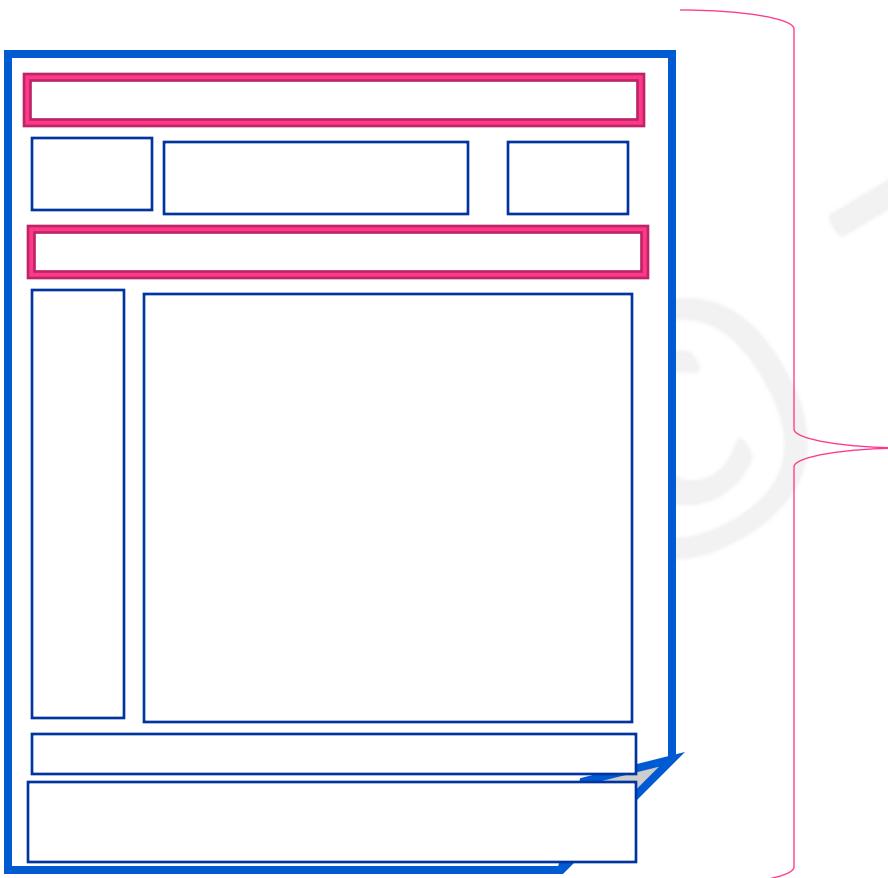
- Component Metadata
- Providers
- viewProviders
- Directives
- pipes





# Component Tree

All Angular 2 applications are **HTML5 WEB COMPONENTS**



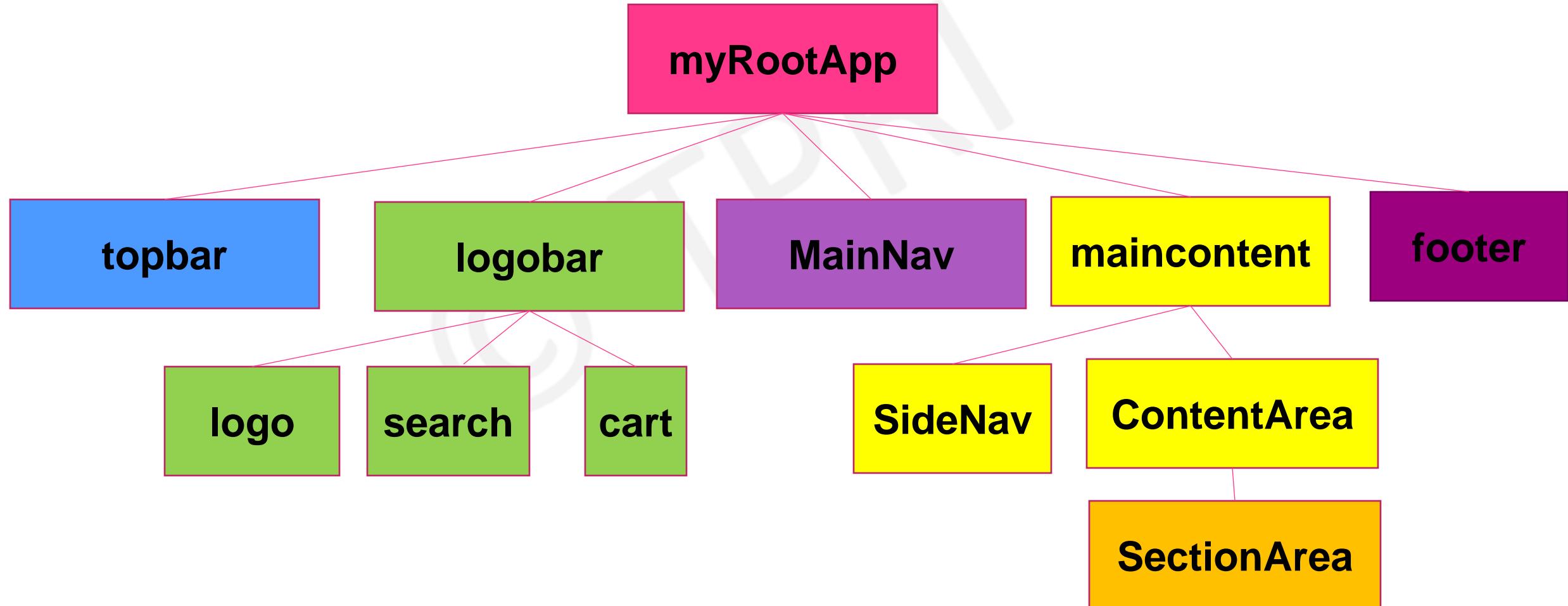
The components have a tree structure, with the root as the *bootstrapped* component.

Single Web Page  
divided into multiple  
Web Components





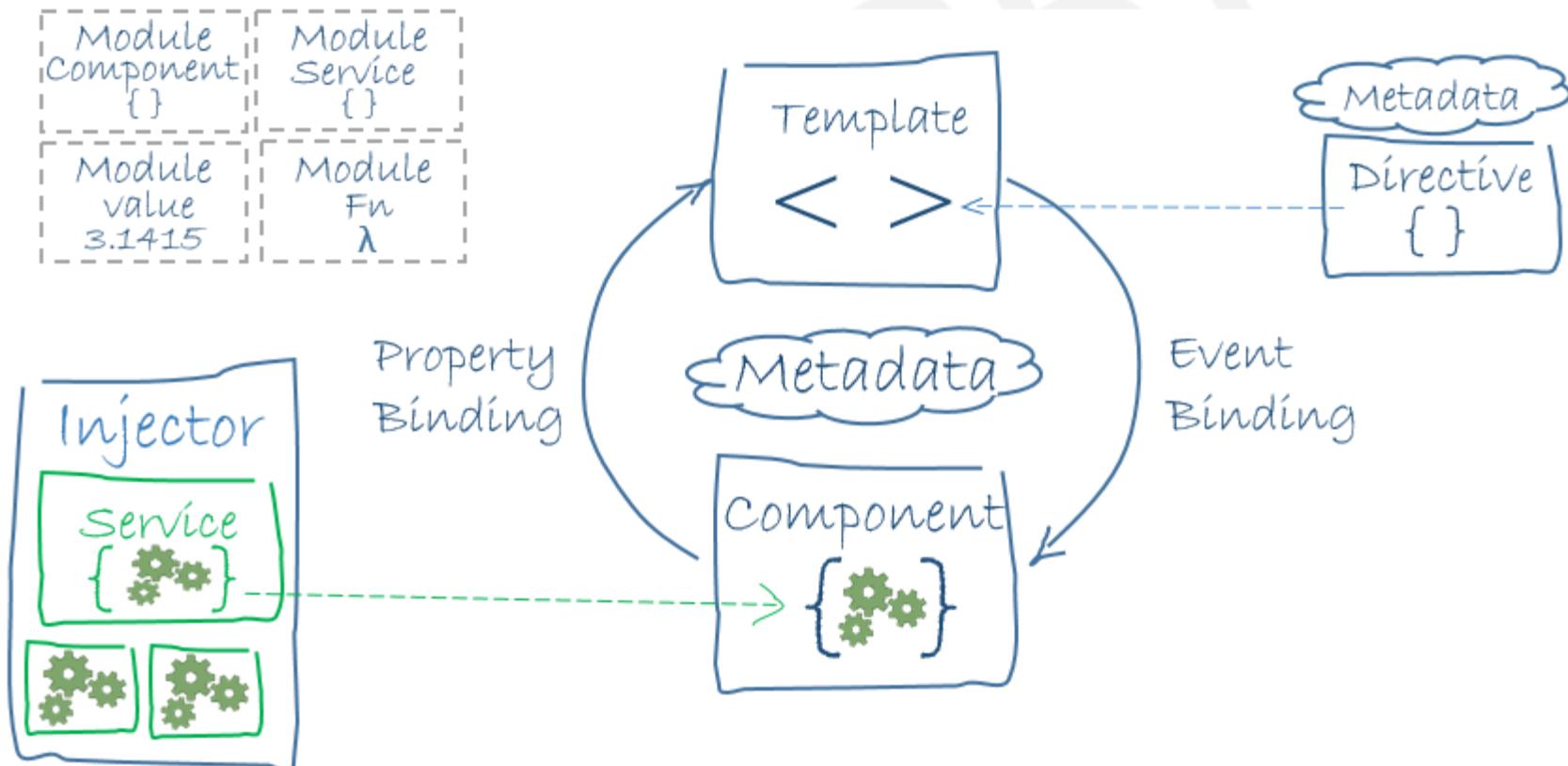
# Component Tree

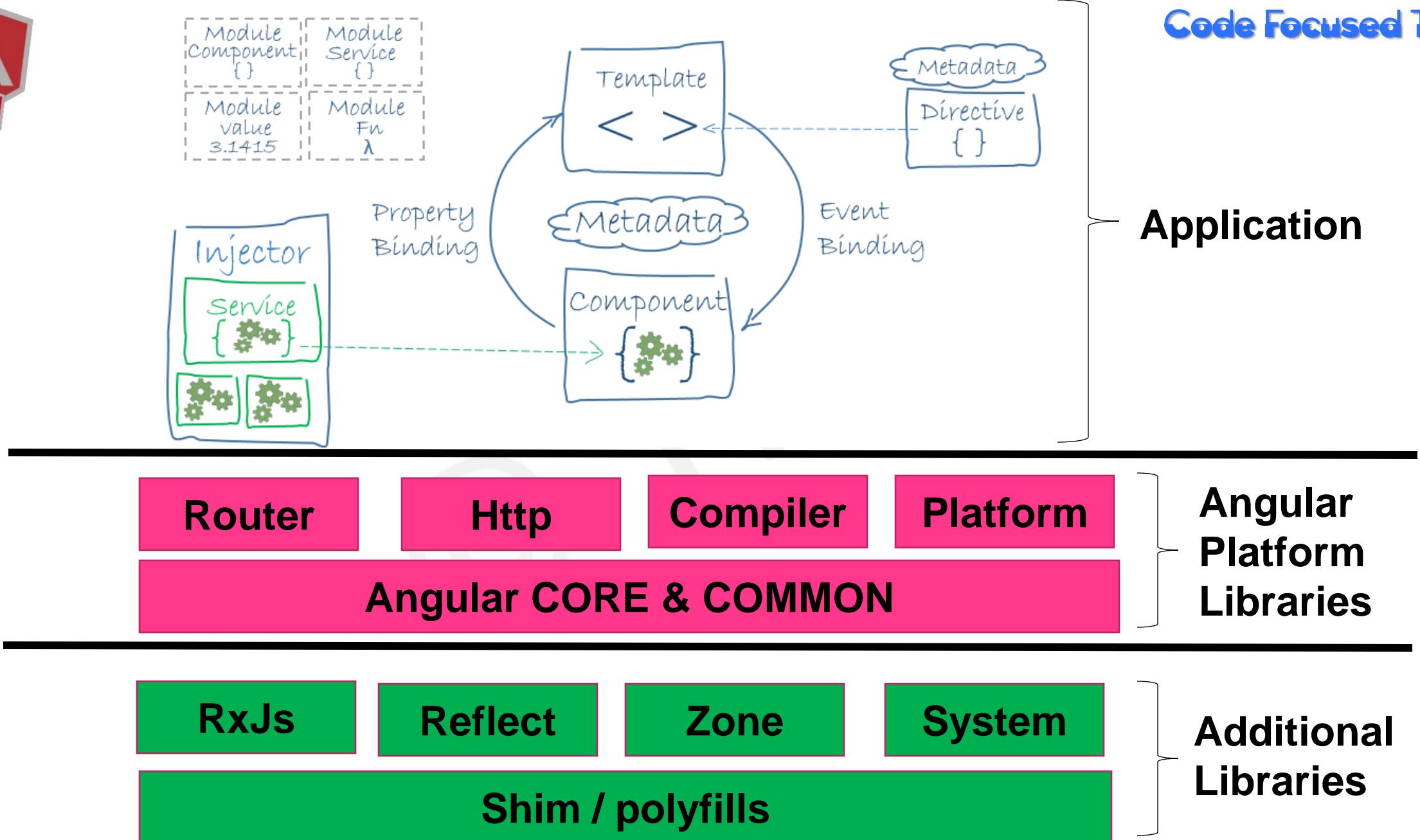




# Module

A bunch of components could be packaged in a module

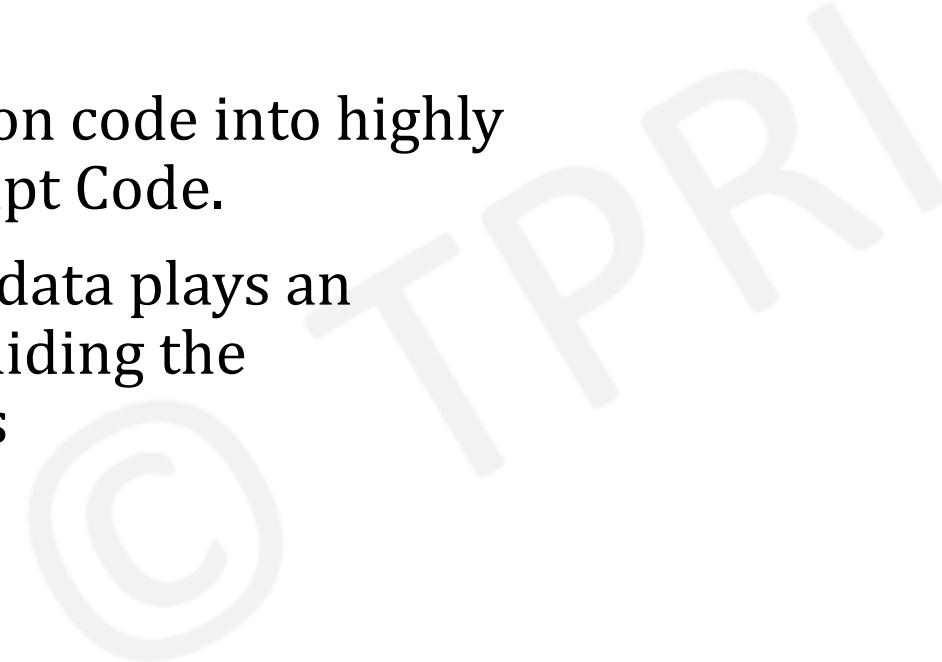






# Angular Compiler

- converts application code into highly performant JavaScript Code.
- **@NgModule** metadata plays an important role in guiding the compilation process





# Building with Angular 2 Components





# Angular 2 Components

A component contains application logic that controls a region of the user interface that we call a view

Imports use other modules inside a component



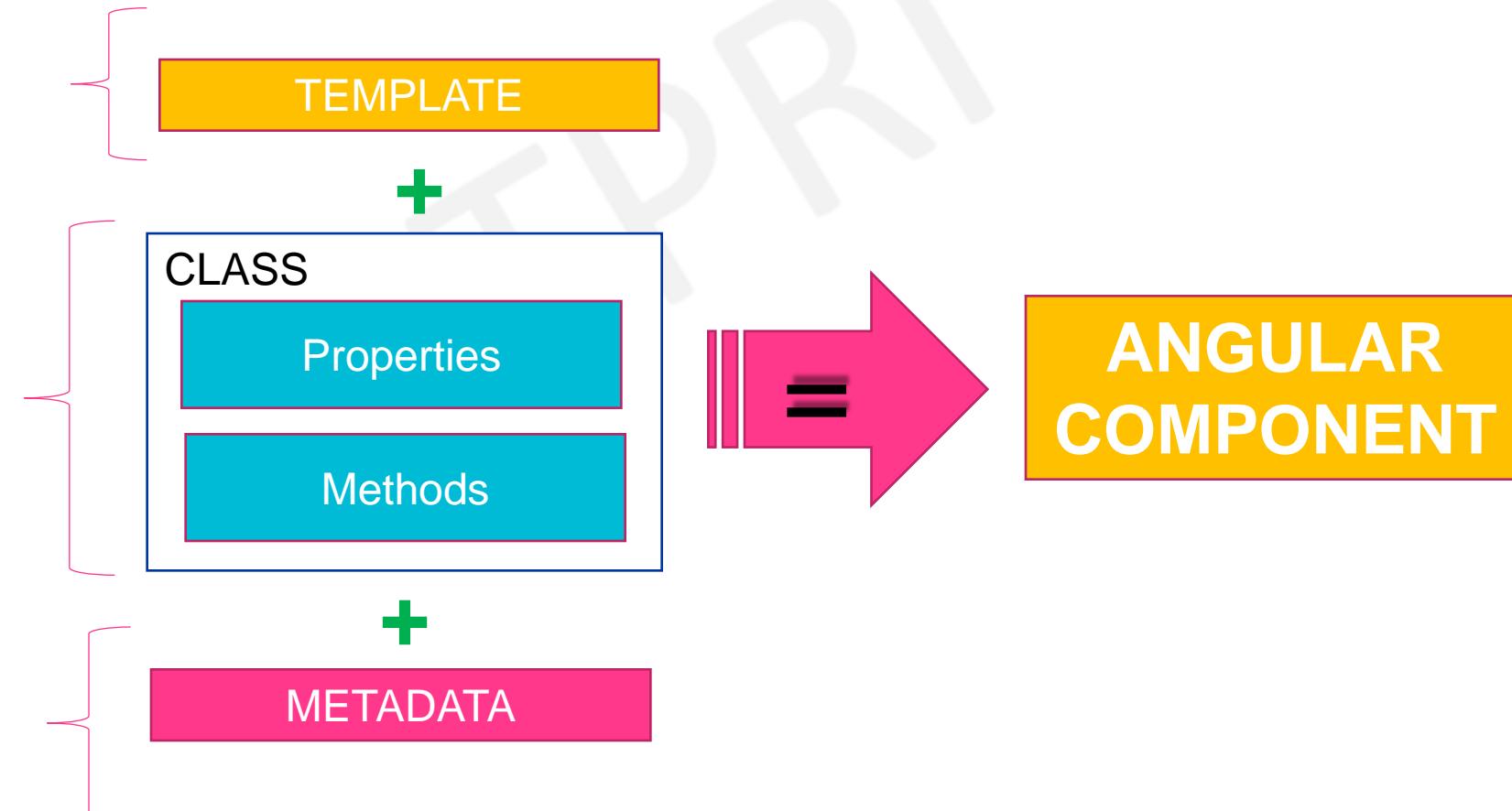


# AngularJS 2.0 Component Anatomy

- VIEW
- created with HTML
- includes binding and directives

- code supporting the view
- created with TypeScript
- Properties: data
- Methods:logic

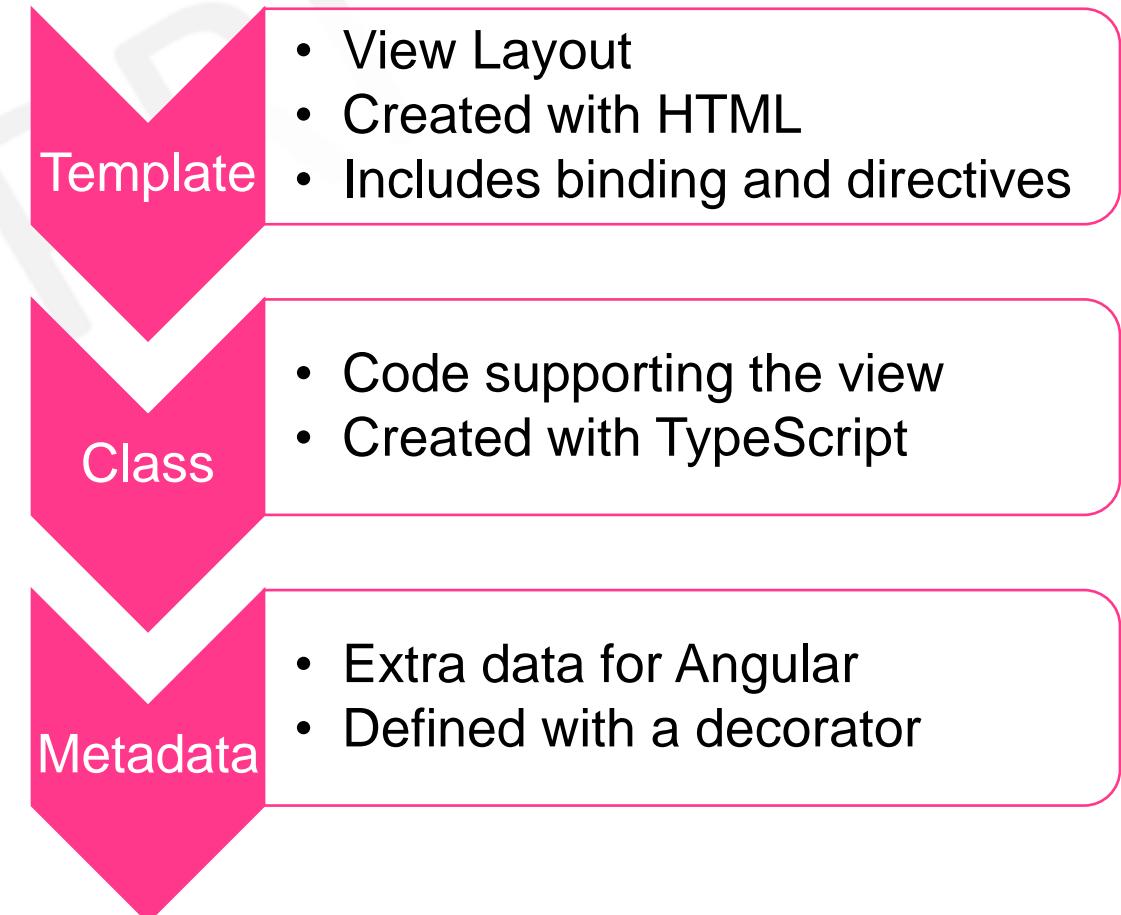
- Extra data for Angular
- Defined with a decorator



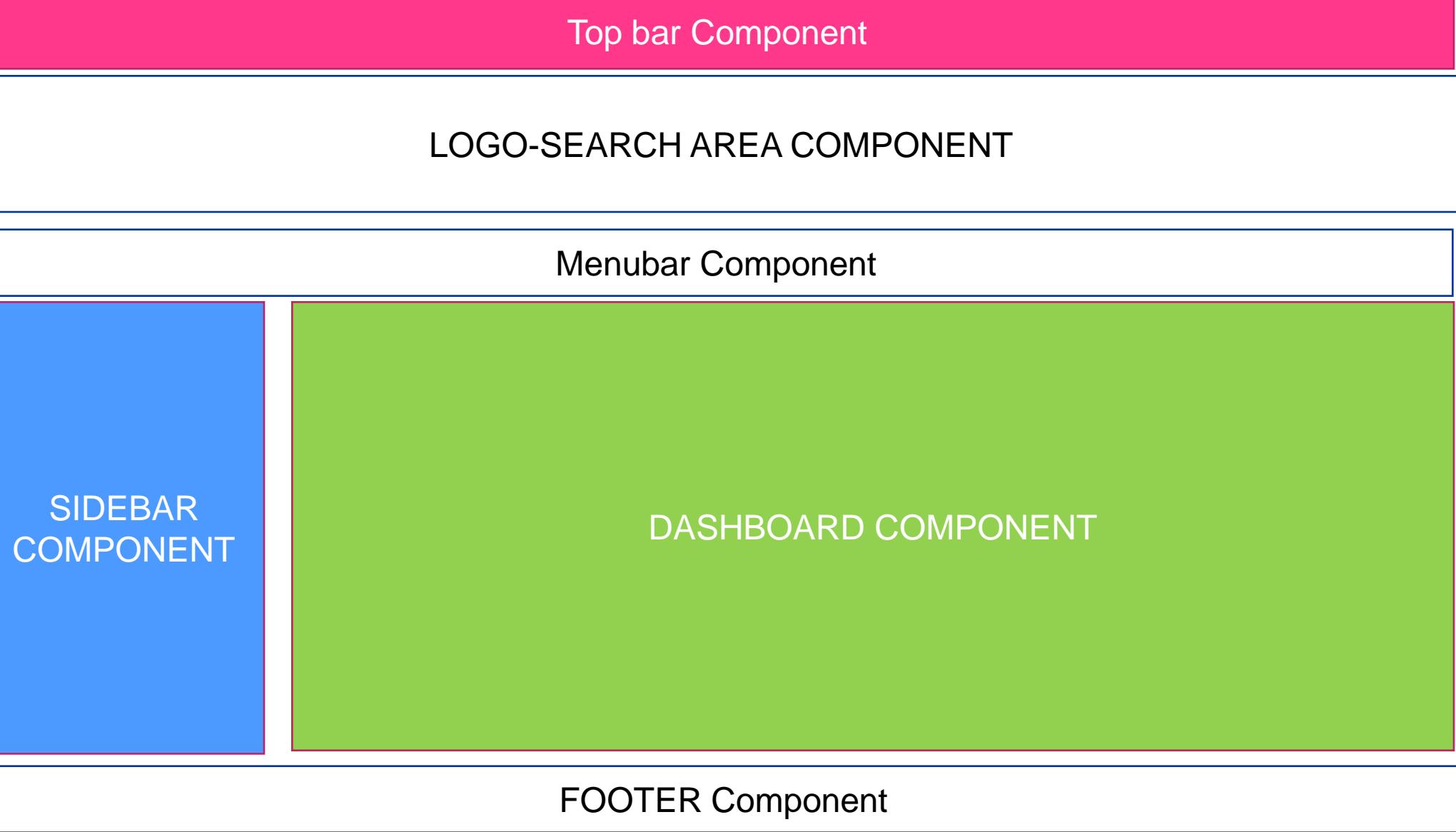


# Component

- A class with component metadata
- Defining the metadata with a decorator
- Importing what we need.
- Responsible for a piece of the screen referred to as view
- Template is a form HTML that tells angular how to render the component.
- Metadata tells Angular how to process a class.



## App Component | Container





# Metadata

- Metadata is extra information which gives angular more info
  - @Component tells angular the class is a component
  - @Directive tells angular the class is a directive





# Bootstrapping

- Angular apps built with components
- One component is chosen by the developer to kickoff the app
- This process is known as **bootstrapping**.

## Angular 1.x

```
<!--bootstrapping angular1.x application-->
<html ng-app="app">
```

## Angular 2

```
import {bootstrap} from 'angular2/platform/browser';
import {AppComponent} from './app.component';

bootstrap(AppComponent);
```

- In Angular 2 a component identifies its Template





# Component in TypeScript

```
//import from angular2-core
import {Component} from 'angular2/core';

//metadata and template
@Component({
    selector: 'sycliq-one',
    template: `
        <div>
            <h1>{{pageTitle}}</h1>
            <div> SycliQ Component </div>
        </div>
    `
})

//class
export class AppComponent{
    pageTitle: string = "SycliQOne Platform";
}
```

import

Metadata and Template

Class Construct





# The Component Class

```
//class  
export class AppComponent{  
  pageTitle: string = "SyycliQOne Platform";  
}
```

Class keyword

Component Name  
When used in code

Export keyword

Class Name=  
FeatureName+Component(Suffix)

Property Name

Data Type

Value

This diagram illustrates the structure of an Angular component class. It highlights several key components: the 'class' keyword, the component name 'AppComponent', the 'export' keyword, the class name formula 'FeatureName+Component(Suffix)', the property 'pageTitle', its data type 'string', and its value 'SyycliQOne Platform'. Arrows point from each annotation to its corresponding element in the code.





# Defining the Metadata

- Decorator Function adds Metadata to a class, its members, Or its method arguments (ES2016)
- Prefixed with an @
- Angular provides built-in decorators

```
//metadata and template
@Component({
  selector: 'sycliq-one',
  template: '
    <div>
      <h1>{{pageTitle}}</h1>
      <div>SycliQ Component</div>
    </div>
  '
})
```

**Component Decorator** → `@Component({`

**View layout** → `template: '`

**Reference in HTML – Directive Name Used in HTML** → `selector: 'sycliq-one'`

**Data Binding Expression** →  `{{pageTitle}}`





# Importing

- We use **import** keyword to use an external function or class.
- ES2015 feature
- Import statement
- Import allows us to use exported members from external modules
- Import is used to import a third-party library, our own modules, or from Angular

```
//import from angular2-core  
import {Component} from 'angular2/core';
```

Member Name

Angular Library  
Module name





# Bootstrapping application

## HTML PAGE

```
//index.html
System.import('app/main')

//HTML Directive Includes
<body>
  <sycliq-one>Application Loading...
  </sycliq-one>
</body>
```

## Main.ts (bootstrapper)

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

import {AppComponent}
  from './app.component';

bootstrap(AppComponent);
```



- Load the root component(bootstrap)

- Host the application

## Component

```
//metadata and template
@Component({
  selector: 'sycliq-one',
  template: `
    <div>
      <h1>{{pageTitle}}</h1>
      <div>Sycliq Component</div>
    </div>
  `
})
//class
export class AppComponent{
  pageTitle: string = "SycliqOne Platform";
}
```





# Component Checklist

Import what we need

Defines where to find the members that this component needs

Member names are case sensitive and Module path should be specified.



Define the decorator with Metadata

Prefix with @; suffix with (), Use selector for component name in HTML

Use template – View's HTML



Class -> code

Use PascalCasing, Append “Component” to the name

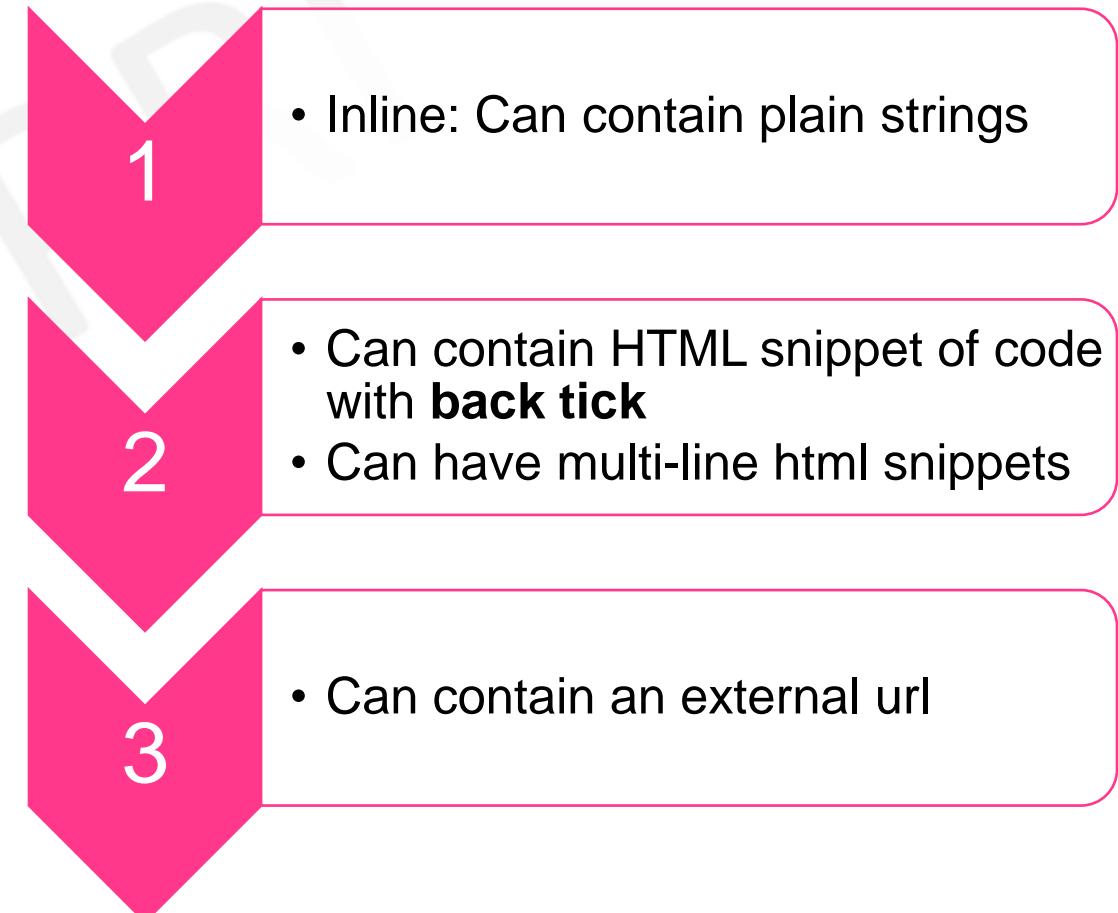
Export Keyword, Data in properties, camelCase (Property and Method Names)





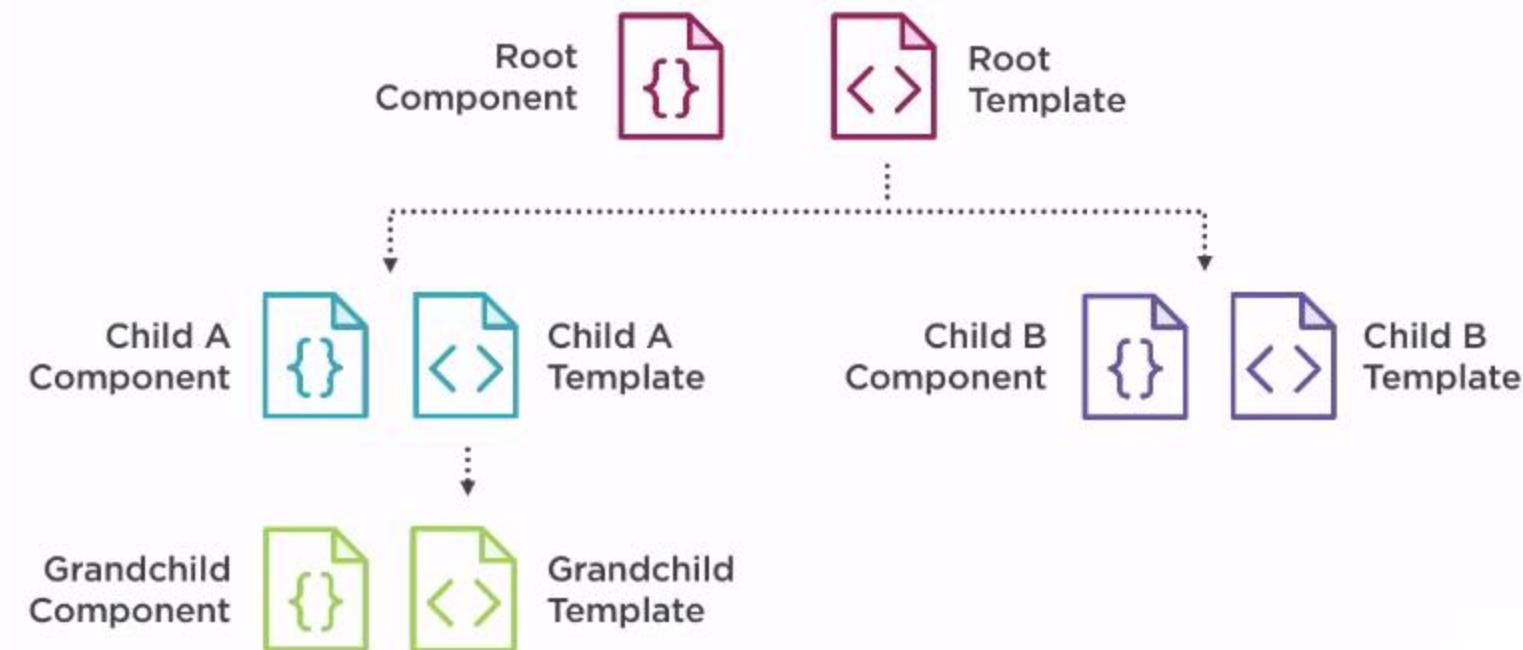
# Template

- Is a way to describe a view using HTML
- Templates can be included with the component
- Or as a URL link to an HTML file
- Best practice is to use an HTML file





Components have templates which may contain other components. This is how we can create a **Component Tree**





# Defining a template in a component

## Inline Template

```
template:  
"<h1>{{pageTitle}}</h1>"
```

## Inline Template

```
template: `  
<div>  
  <h1>{{pageTitle}}</h1>  
  <div>  
    My First Component  
  </div>  
</div>`
```

## Linked Template

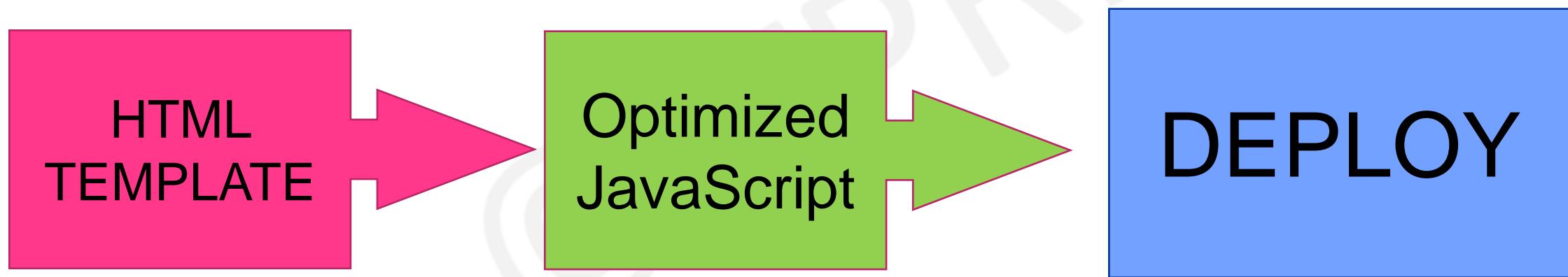
```
templateUrl:  
'product-list.component.html'
```

ES 2015  
Back Ticks





# Template Compiler



**Offline Compile Step  
Template Compiler**





# Using a component as a Directive

- 1 • Build the component
- 2 • Identify the container component for the application
- 3 • Add the component selector to the container component template
- 4 • Add directives metadata in the @Component decorator and the name of the component
- 5 • Import the component reference in the container component.





## Index.html

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

### Container Component

Step 3:

```
import { Component } from '@angular/core';
import {TodoInput} from './todo-input';
import {EventSample} from './app.eve.component';
```

Step 2:

```
@Component({
  selector: 'my-app',
  directives: [TodoInput, EventSample],
  template: `<h1>My First Angular 2 App</h1>
<div><todo-input></todo-input></div>
<div><aevent-example></aevent-example></div>
`
```

Step 1:

```
)>
export class AppComponent { }
```

## component

```
import{Component} from "@angular/core";
@Component({
  selector: 'aevent-example',
  template: `
    <div>
      <input type="text" #myInput/>
      <button (mouseover)="onClick($event,myInput.value)">Click me</button>
    </div>
  `})
export class EventSample{
  onClick(event:any,value:any){
    console.log(value);
  }
}
```





# Directive

- A custom HTML element or attribute used to power up and extend our HTML
  - Built-in Directives
  - Custom Directives
- Angular Built-in Directives
  - Structural Directives
    - \*ngIf : IF logic
    - \*ngFor: For Loops





# \*ngIf Built-In Directive : Structural Directive

- \*ngIf is used to check for **true** or **false** value to execute specific behaviour

```
<table *ngIf='products.length'>
  <thead>
    </thead>
  <tbody>
    <tbody>
  </table>
```





# \*ngFor Built-In Directive : Structural Directive

- \*ngFor repeats a portion of the DOM tree once for each item in an iterable list.

```
<table *ngFor="#emp of employees">
  <tr>
    <th>Employee Name</th>
    <th>Employee Department</th>
    <th>Salary</th>
  </tr>
  <tr>
    <td>{{ emp.empName }}</td>
    <td>{{ emp.empDept }}</td>
    <td>{{ emp.salary }}</td>
  </tr>
</table>
```





# ES2015 : for..of vs for...in Loop

## For..of

- Iterates over iterable objects, such as an array

## For..in

- Iterates over properties of an object

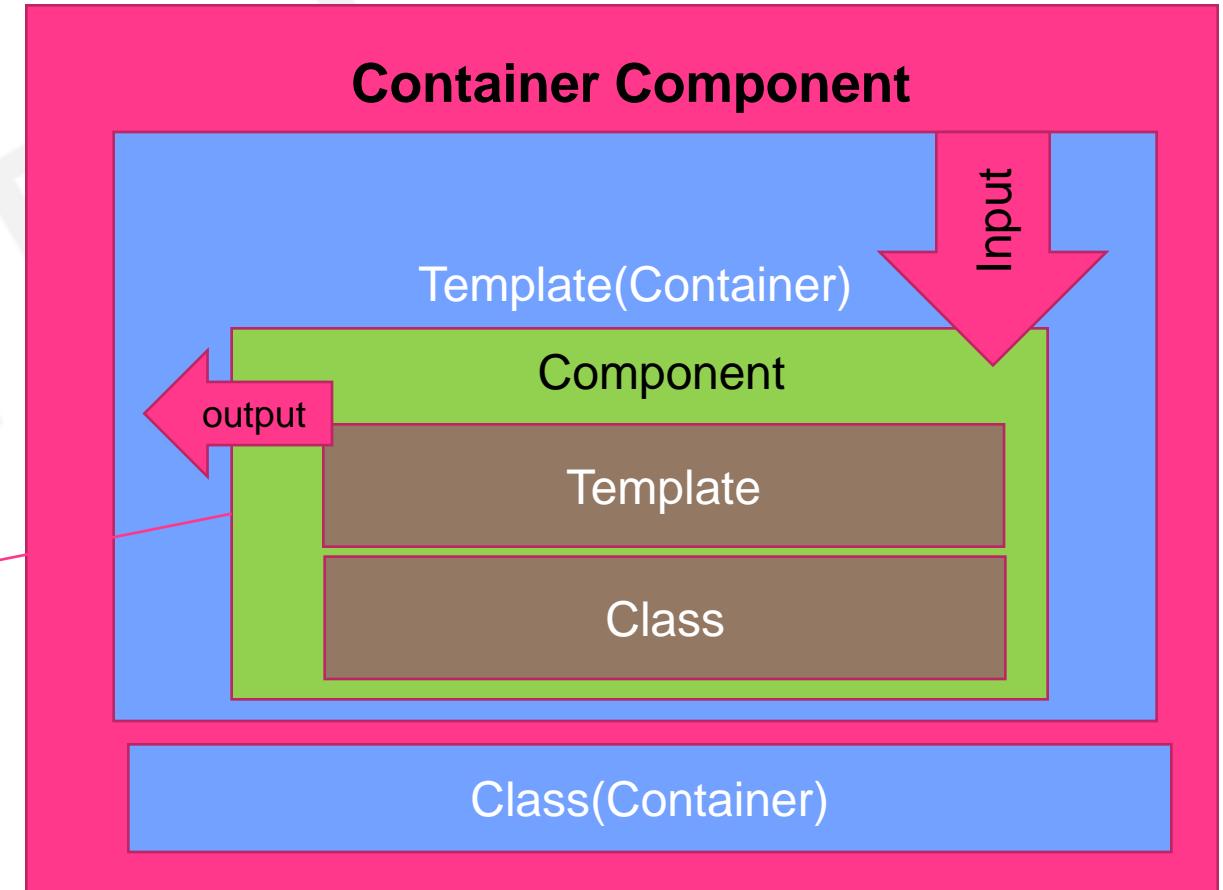




# Nest-able Components in A2

- Its template only manages a fragment of a larger view
- It has a **selector**
- **It optionally communicates with its container**

Referenced through <template1/>





# Using a Nested Component

```
import { Component } from '@angular/core';
import {TodoInput} from './todo-input';
import {EventSample} from './app.eve.component';
@Component({
  selector: 'my-app',
  directives: [TodoInput, EventSample],
  template: `<h1>My First Angular 2 App</h1>
<div><todo-input></todo-input></div>
<div><aevent-example></aevent-example></div>
`})
export class AppComponent { }
```

```
import{Component} from "@angular/core";
@Component({
  selector:'aevent-example',
  template: `
    <div>
      <input type="text" #myInput/>
      <button (mouseover)="onClick($event,myInput.value)">Click me</button>
      <div>
        <ng-content></ng-content>
      </div>
    </div>
  `})
export class EventSample{
  onClick(event:any,value:any){
    console.log(value);
  }
}
```





# Passing Data to a Nested Component

## product-list.component.ts

```
@Component({
  selector: 'pm-products',
  templateUrl: 'product-list.component.html',
  directives: [StarComponent]
})
export class ProductListComponent { }
```

## star.component.ts

```
@Component({
  selector: 'ai-star',
  templateUrl: 'star.component.html'
})
export class StarComponent {
  @Input rating: number;
  starWidth: number;
}
```

## product-list.component.html

```
<td>
  <ai-star></ai-star>
</td>
```





# Passing Data to a Nested Component

## product-list.component.ts

```
@Component({
  selector: 'pm-products',
  templateUrl: 'product-list.component.html',
  directives: [StarComponent]
})
export class ProductListComponent { }
```

## product-list.component.html

```
<td>
  <ai-star [rating]='product.starRating'>
  </ai-star>
</td>
```

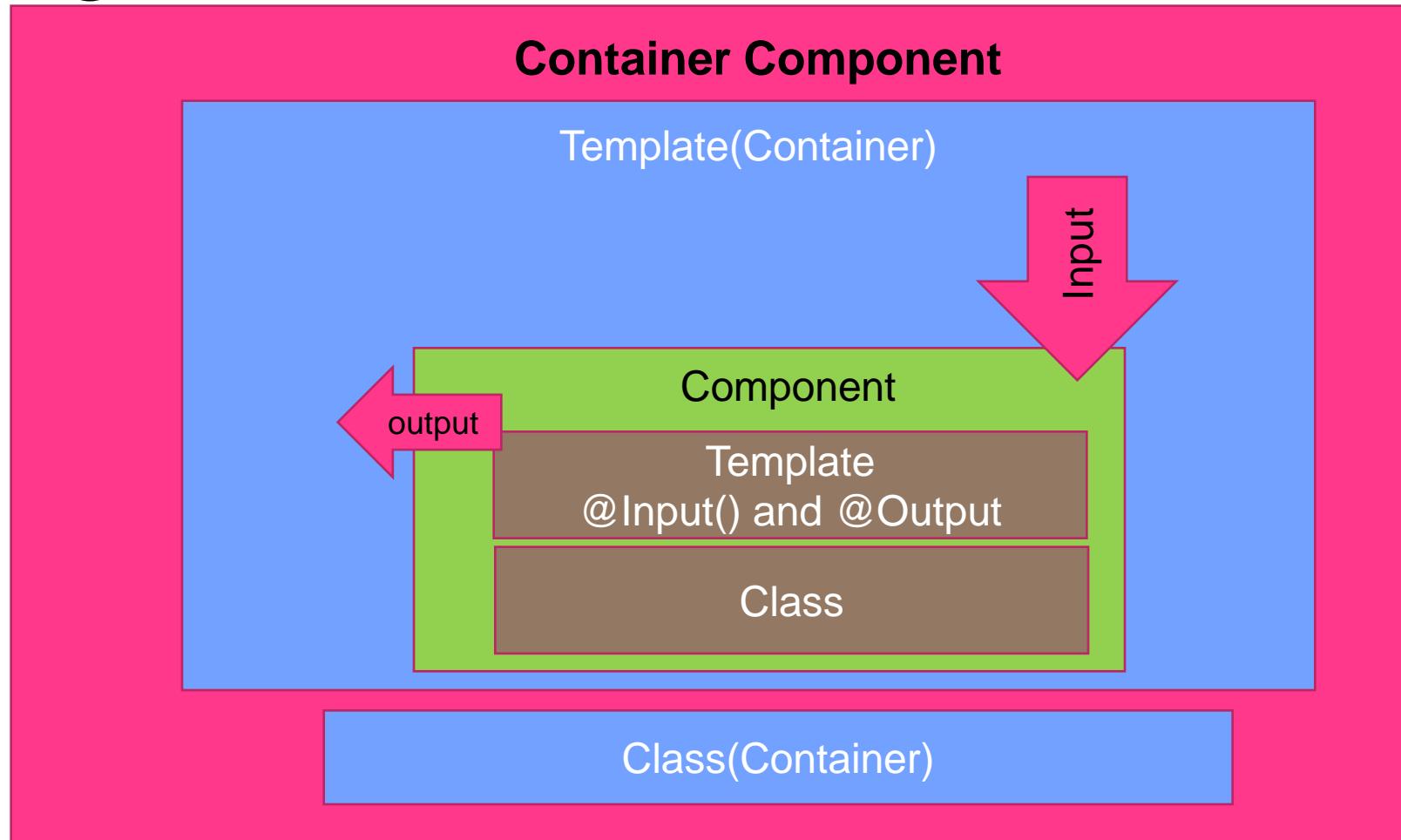
## star.component.ts

```
@Component({
  selector: 'ai-star',
  templateUrl: 'star.component.html'
})
export class StarComponent {
  @Input rating: number;
  starWidth: number;
}
```





# Raising an Event(@Output)





# Nested Component

- Input decorator
  - Attached to a property of any type
  - Prefix with @ and suffix with ()
- Output decorator
  - Attached to a property declared as an Event Emitter
  - Use the generic argument to define the event payload type
  - Use the new keyword to create an instance of the Event Emitter
  - Prefix with @ and suffix with ()





# Container Component

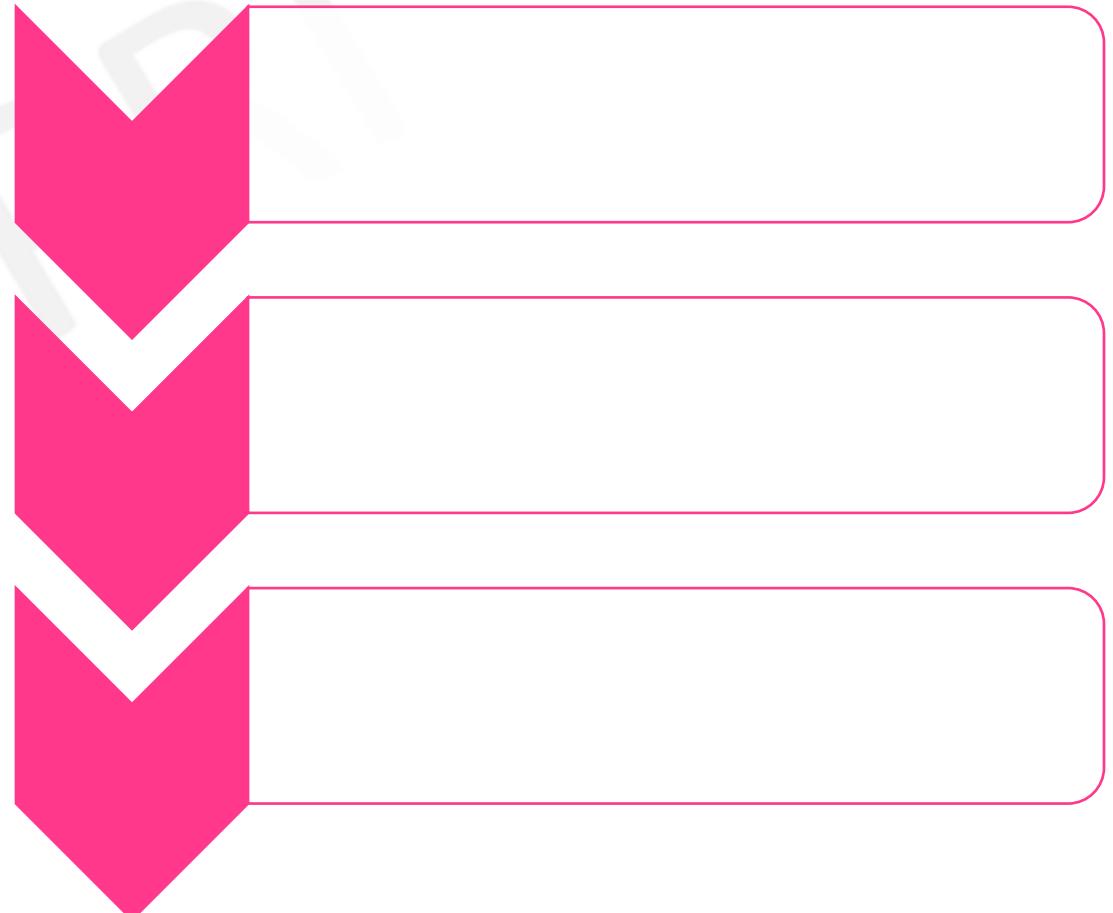
- Use the directive
  - Directive name -> nested component's selector
  - We use property binding to pass data to the nested component
  -
- We use event binding to respond to events from the nested component
  - Use \$event to access the event payload passed from the nested component





# Data binding

- Component -DOM
- {{value}} => interpolation
- [property] = “value” -> property binding (used to pass data from parent component to child)
- (event) = “handler” <- event binding
- [(ng-model)] = “property” <-> two way binding.





# Service

- Substitutable objects that are wired together using dependency injection (DI)
- Used to share code across an app
- Lazily instantiated
- Angular has no “Service” defined type





# Directives

- A class with directive metadata
- Two kinds
  - Attribute :
    - Attribute directives alter the look or behaviour of an existing element
  - Structural
    - Structural directives alter the layout by adding, removing and replacing elements in the DOM
  - A Component is a directive with a view.





# Dependency Injection

- A way to supply a new instance of a class with the fully-formed dependencies it needs
- Most dependencies are services
- Angular know which services a components by looking at the types of its constructor parameters
- Services are injected by an injector which uses a provider to create the service.





# Angular 2 Bindings and Events



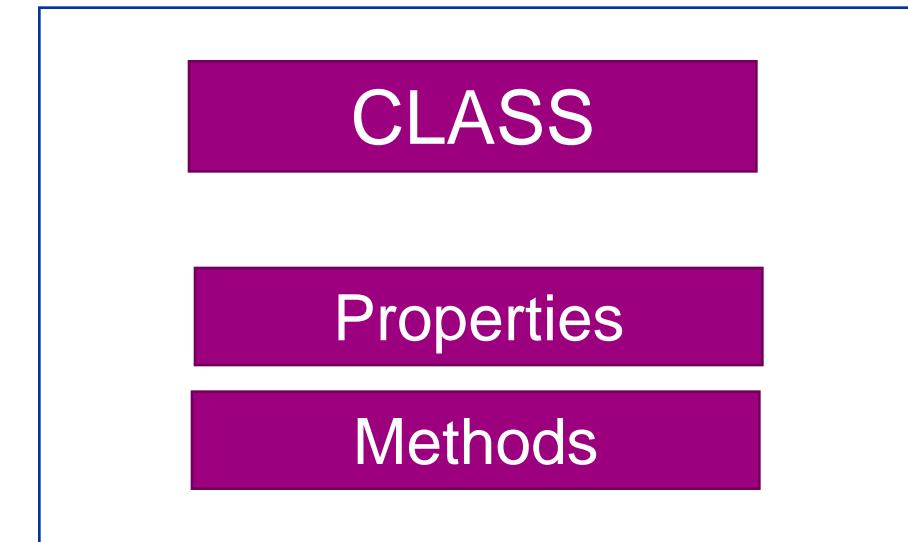
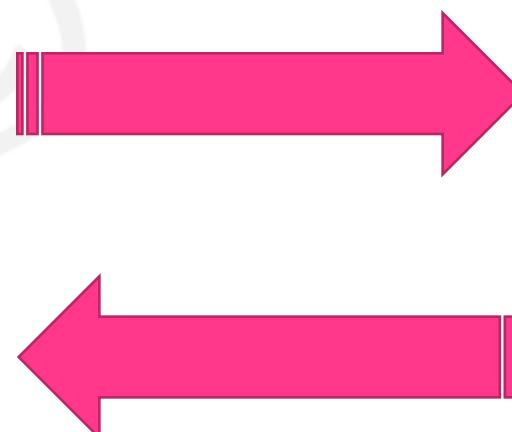


# Binding

- Coordinates communication between the component's class and its template and often involves passing data.

**{{Template Expression}}**

TEMPLATE





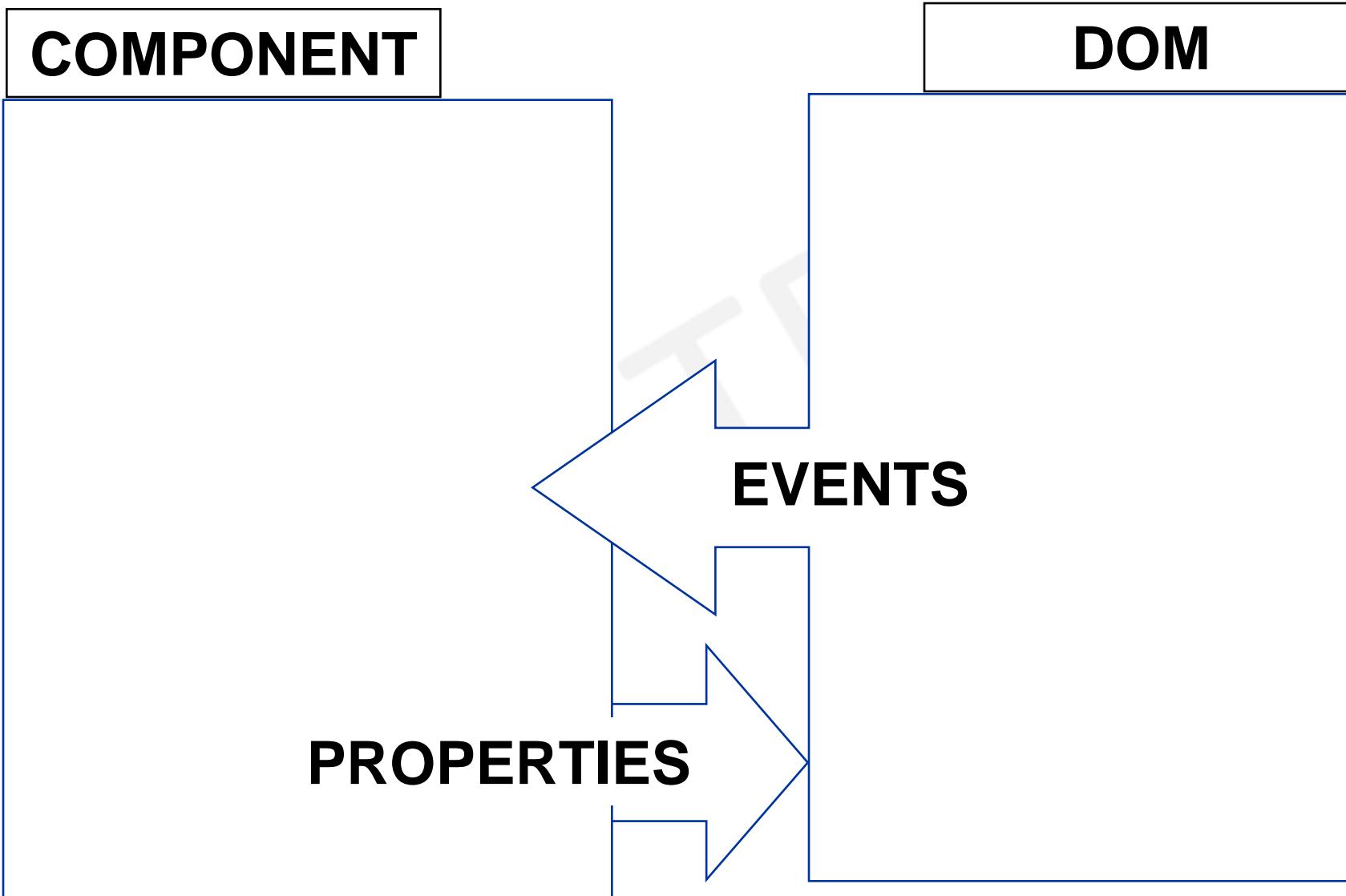
# BINDING

COMPONENT

DOM

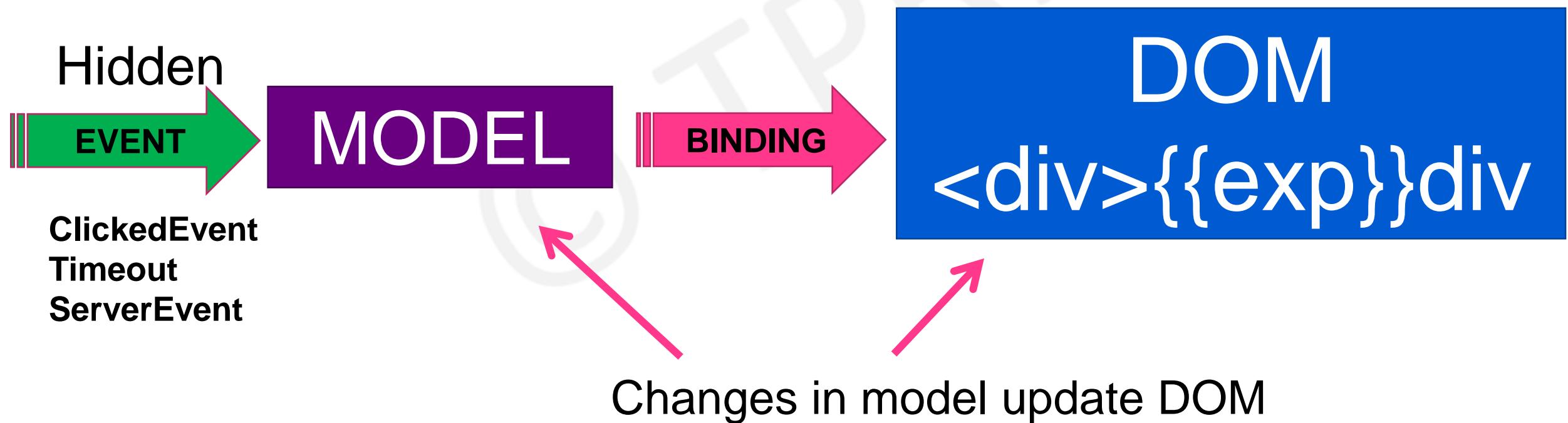
EVENTS

PROPERTIES





# One-way data binding

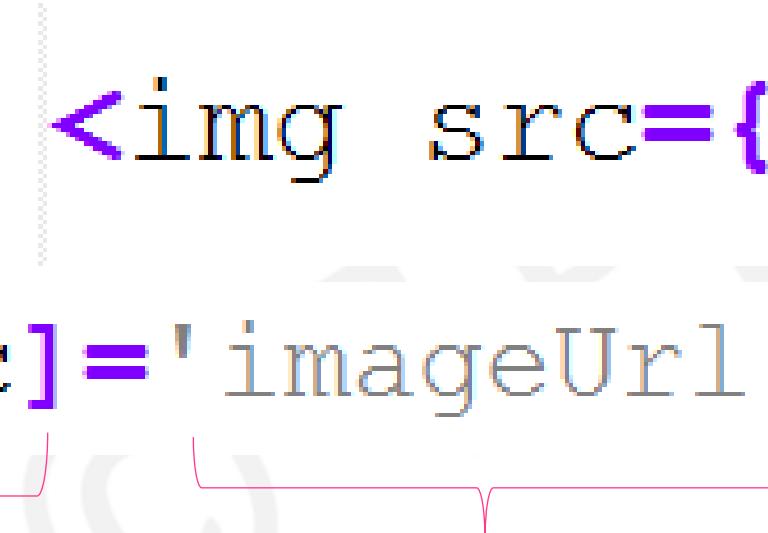




# Property Binding : One-way Binding

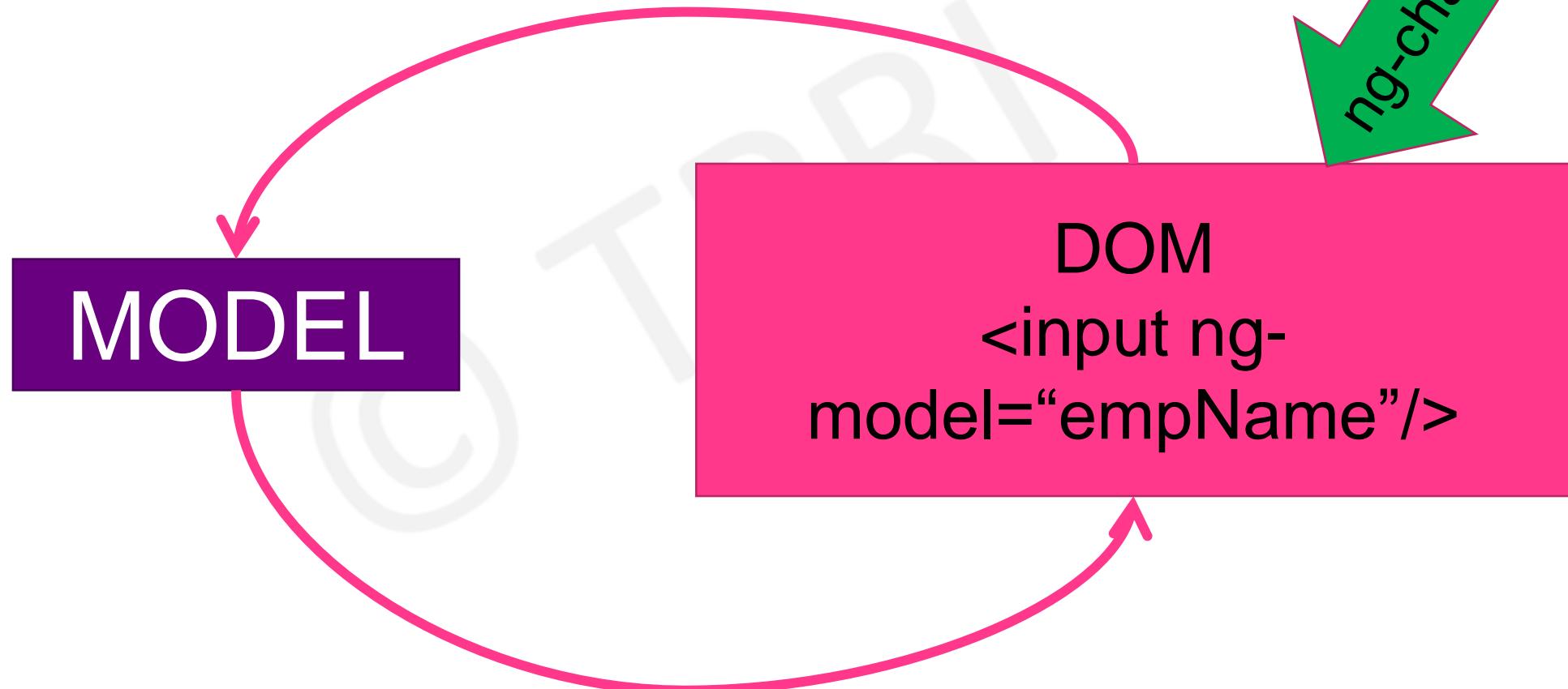
```
<img src={{imageUrl}}/>  
  
<img [src]='imageUrl'>
```

**Binding Target Element Property**      **Binding Source Template Expression**





# Two-way data binding (Angular 1.x)

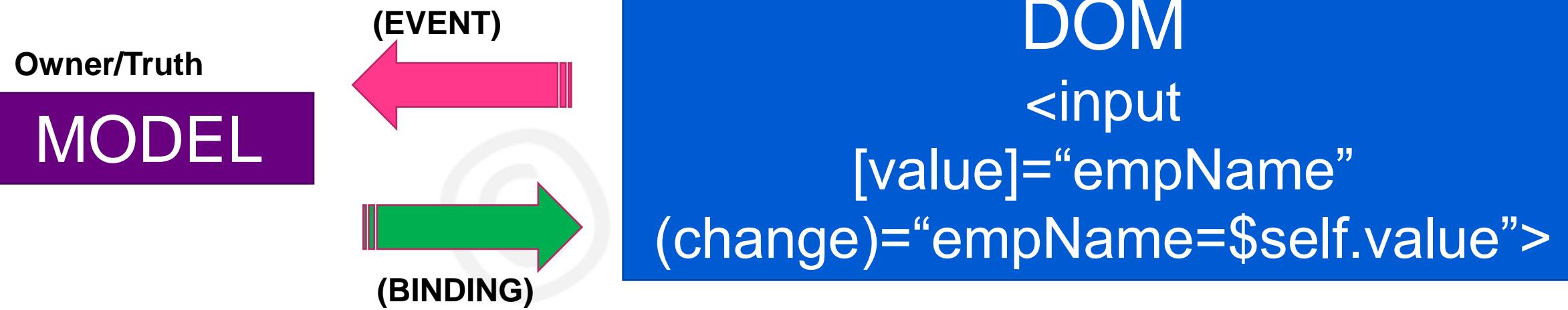


\$Digest triggers the update





# A2 : two-way data binding





# A2: Two way data binding: [(ngModel)]



```
<div *ngIf="selectedHero">
  <h2>{{selectedHero.name}} details!</h2>
  <div><label>id: </label>{{selectedHero.id}}</div>
  <div>
    <label>name: </label>
    <input [(ngModel)]="selectedHero.name" placeholder="name"/>
  </div>
</div>
```





# Event Reference: MDN

<https://developer.mozilla.org/en-US/docs/Web/Events>

- HTML DOM EVENTS
- DOM EVENTS
- CSSOM EVENTS
- DEVICE ORIENTATION EVENTS
- DEVICE STORAGE API EVENTS
- DOWNLOAD API EVENTS
- FILE API EVENTS
- FULLSCREEN API EVENTS
- BROWSER API EVENTS
- BATTERY API EVENTS
- APP CACHE EVENTS
- HTML DRAG AND DROP API EVENTS
- INDEXEDDB EVENTS
- WEB SOCKETS API EVENTS
- WEB STORAGE API EVENTS
- WEBRTC EVENTS
- WEB NOTIFICATION EVENTS





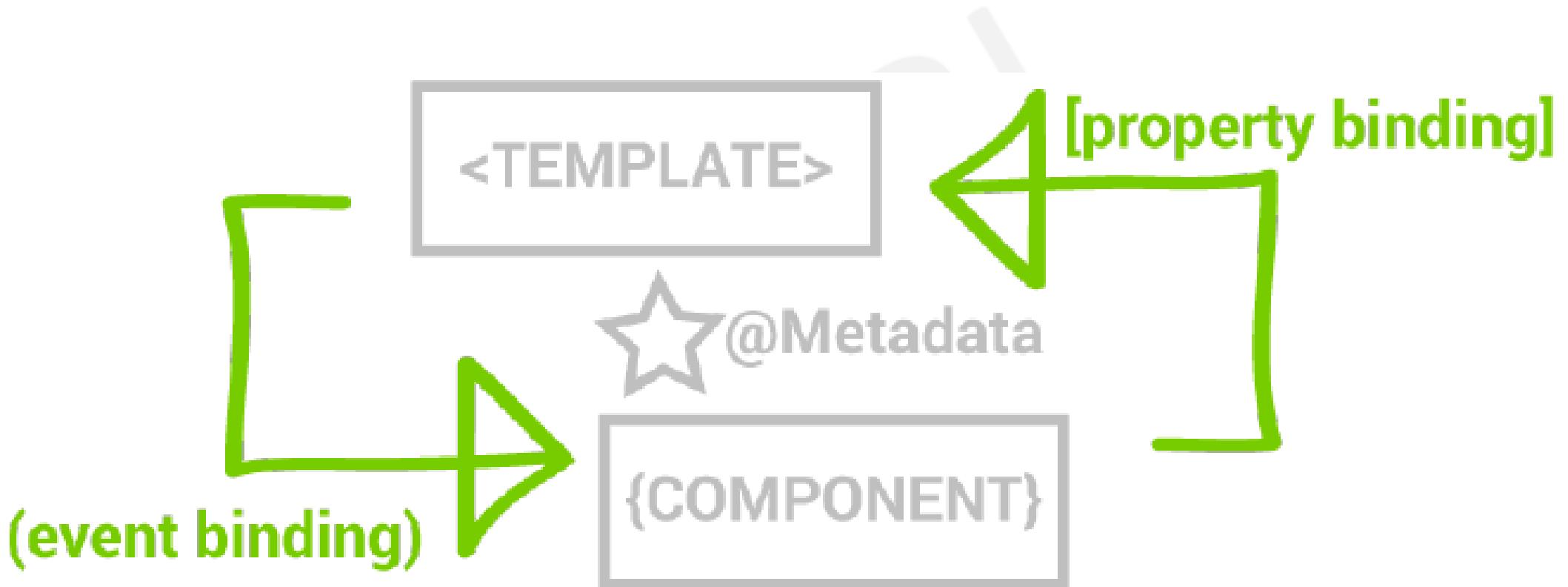
# Event Binding /Behaviour Binding

```
import{Component} from "@angular/core";
@Component({
  selector:'aevent-example',
  template: `
    <div>
      <input type="text" #myInput/>
      <button (mouseover)="onClick($event,myInput.value)">Click me</button>
    </div>
  `
})
export class EventSample{
  onClick(event:any,value:any){
    console.log(value);
  }
}
```





# Custom Data Binding





# A2: Data Binding Summary

DOM

COMPONENT

1 way Binding / Interpolation: {{empName}}



1 way Binding / Property Binding: <img [src]:'emp imgUrl'>

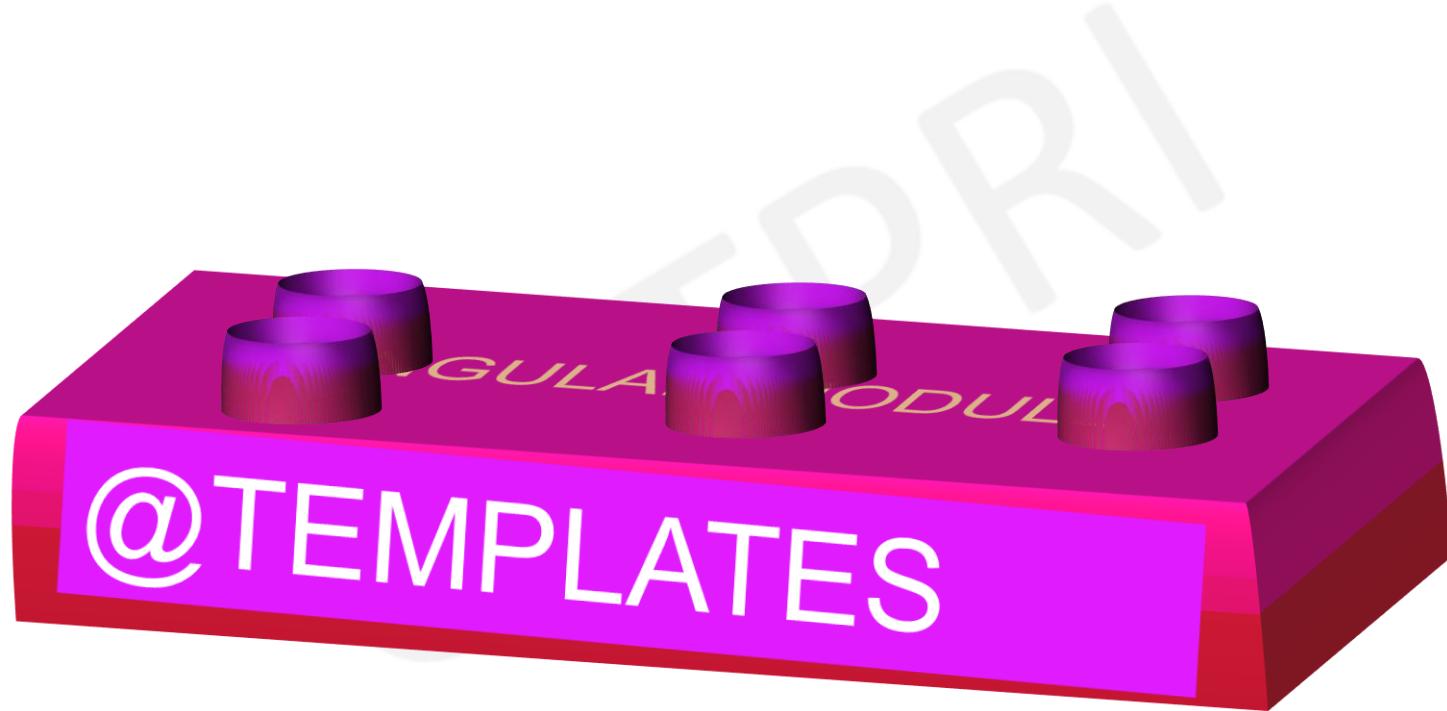


Event Binding: <button (click) = 'onClick()' >



Two-way Binding: <input [(())]=‘empName’/>







© TPRI





# Angular CLI





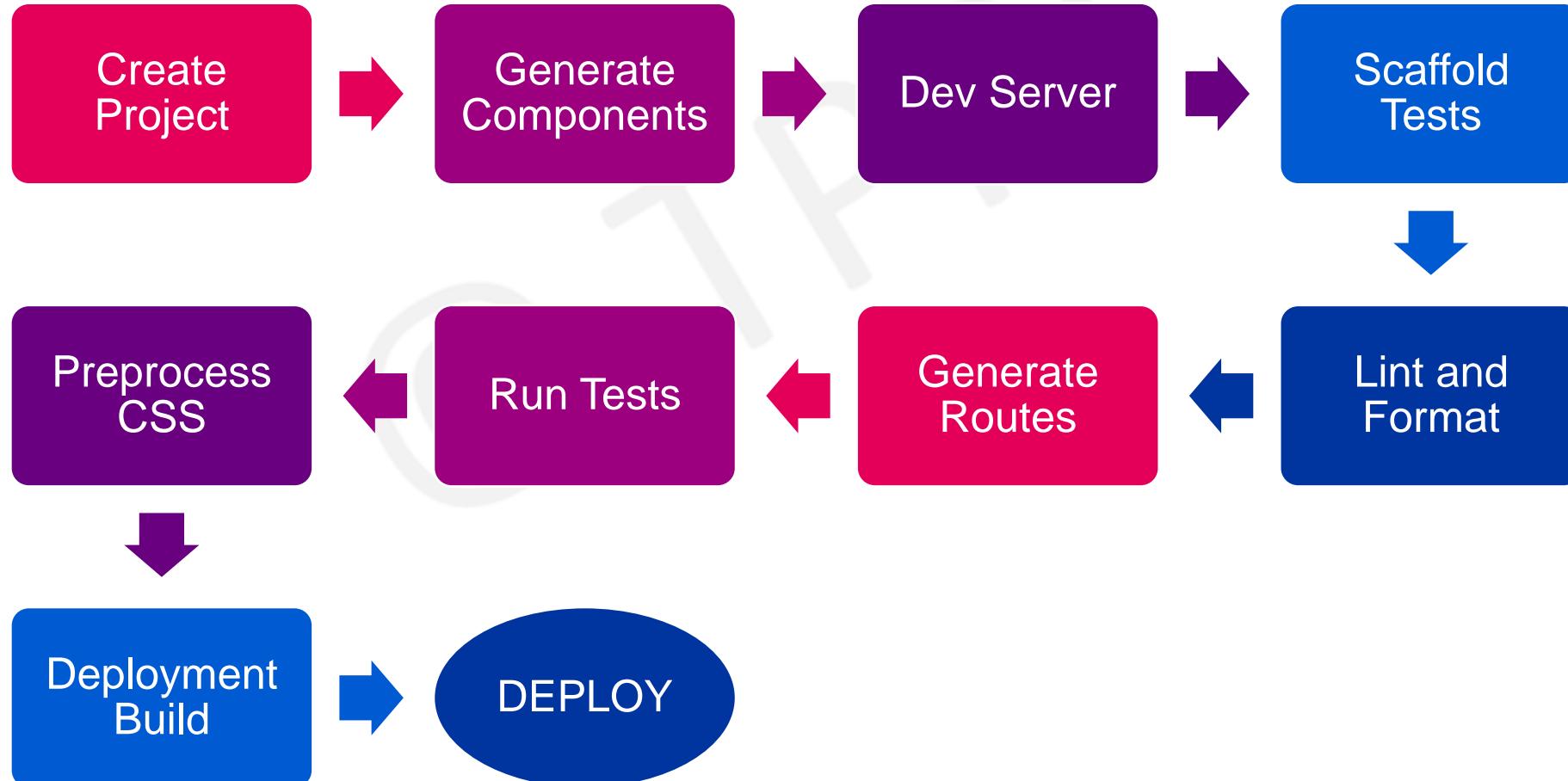
# Angular 2 Setup

- Creating our app files
- Configuring TypeScript and Typings
- Adding script tags for
  - Angular 2
  - Rx.js
  - System.js
- Configuring System.js
- Creating our Angular 2 component
- Bootstrapping our Angular 2 application





# Workflow Automation: Angular CLI





# Angular-cli: NEW

- Installation
  - Npm install -g angular-cli
- Generating and serving an Angular 2 project via a development server
  - ng new PROJECT\_NAME
  - cd PROJECT\_NAME

- Ng NEW OPTIONS

- --dry-run : Only output the files created and operation performed. It does not actually create the project.
- --verbose : show more information
- --skip-npm: don't run any npm commands like installing dependencies
- --skip-git : don't create a new git repo for this project
- --directory : specify the directory to create this project in





# Angular-cli: Initialize

- Initialize a new application
- `ng init PROJECT_NAME`
- NG INIT OPTIONS
  - `--dry-run` : only output the files created and operations performed.  
It doesn't actually create the project.
  - `--verbose` : show more information
  - `--skip-npm` : don't run any npm command like installing dependencies
  - `--name` : Name the new project you are creating





# Angular-cli: Serve

- Serve our application to browser
- Ng serve
  - Allows us to serve our application to the browser.
  - Built with BrowserSync: Reload on saves
  - Automatically routed for us
  - <http://localhost:4200>
  - Simplicity and ease of mind
- Ng serve -port 4201 -live-reload-port 49513





# Angular-cli:Generate

- ng generate or ng g : **to generate angular components**
  - The **ng generate command can generate for us**
    - **A new component**
    - **A new directive**
    - **A new route**
    - **A new pipe**
    - **A new service**
    - **A new class**
    - **A new interface**
    - **A new enum**

## • NG GENERATE OPTIONS

- --flat : don't create the code in it's own directory, just add all files to the current directory
- --route=<route> : specify the parent route. Only used for generating components and routes
- --skip-router-generation: Don't create the route config. Only used for generating routes.
- --default : the generated route should be a default route
- --lazy: specify if route is lazy – default true





# Angular-cli:Generate

Scaffold	Usage
Component	Ng g component my-new-component
Directive	Ng g directive my-new-directive
Pipe	Ng g pipe my-new-pipe
Service	Ng g service my-new-service
Class	Ng g class my-new-class
Interface	Ng g interface my-new-interface
Enum	Ng g enum my-new-enum





# Angular-cli: Build

- `ng build` : build artifacts will be stored in the `dist/` directory
- A build can specify both a build target ( development or production) and an environment file to be used with that build.
- `/src/app/environments/environment.ts`
- `Ng build -target = production - environment=prod`
- `Ng build --prod -env=prod`
- `Ng build -prod`
- `Ng build -target=development - environment = dev`
- `Ng build -dev -e=dev`
- `Ng build -dev`
- `Ng build`





# Angular-cli: Bundling

- Builds created with the -prod flag via ng build -prod or ng serve -prod
- Bundle all dependencies into a single file, a
- Unit Test
  - Ng test – tests will execute after a build is executed via karma. And it will automatically watch your files for changes.
- E2E test
  - Ng e2e - before we run the test make sure we are serving the app via ng serve.
  - E2e tests are run via protractor





# Angular-cli: Git Deploy

- Deploying apps via github
- Ng github-pages: deploy -message "Optional commit message"
  - Creates GitHub repo for the current project if one doesn't exist
  - Rebuild the app in production mode at the current HEAD
  - Creates a local gh-pages branch if one doesn't exist
  - Moves your app to the gh-pages branches and creates a commit
  - Edit the base tag in index.html to support github pages
- Autocompletion commands
  - ng completion >> ~/.bashrc





# Angular-cli: update

- to update angular-cli to a new version. We must update both the global packages and project's local package.
- Npm uninstall -g angular-cli
- Npm cache clean
- Npm install -g angular-cli@latest
- Local project package
  - rm -rf node\_modules dist tmp
  - npm install -save -dev angular-cli@latest
- Ng init





# Angular-cli: miscellaneous

- ng get : get values for project
- ng set : sets values for project
- ng lint : run codalyzer to analyze code
- ng format
- ng doc
- ng version : get the version of the CLI





# A2.0 Forms

Section IX





# Forms

- ngModel
- ngSubmit
- FormBuilder
- Validation





# Angular Pipes

Section X





Filter/Pipe Name	Angular 1.x	Angular 2
currency	✓	✓
date	✓	✓
uppercase	✓	✓
json	✓	✓
limitTo	✓	✓
lowercase	✓	✓
number	✓	
orderBy	✓	
filter	✓	
async		✓
decimal		✓
percent		✓





# Pipes

- What are pipes?
- Built-in pipes
- Custom pipes
- Async Pipes

- Filters are now called as Pipes in angular 2.
- Pipes allow us to transform data for display in a template
  - Transform bound properties before display
  - Built-in pipes
    - Date
    - Number, decimal, percent, currency
    - Json, slice





# Pipe Examples

```
 {{ product.productCode | lowercase }}  
  
<img [src]='product.imageUrl'  
      [title]='product.productName | uppercase'>  
  
 {{ product.price | currency | lowercase }}  
  
 {{ product.price | currency:'USD':true:'1.2-2' }}
```





# Pipes

- A pipe takes in data as input and transforms it to a desired output
- We use them in our templates with interpolation
- Include parameters to a pipe by separating them with a colon
- Pipes are chain-able





# Built-in Pipes

- Format a value in a Template

```
<p><span>{{emp.name | uppercase}}</span></p>
<p><span>{{emp.name | lowercase}}</span></p>
//Built in pipes for dates
//date[:format]
<p><span>{{emp.startDate | date:'medium'}}</span></p>
<p><span>{{emp.startDate | date:'yMMMd'}}</span></p>
//numeric pipes
<p><span>{{emp.salary | currency}}</span></p>
<p><span>{{emp.taxrate | percent:'1.1-1'}}</span></p>
<p><span>{{emp.yearofExp | number:'1.1-3'}}</span></p>
```





# Aysnc Pipe

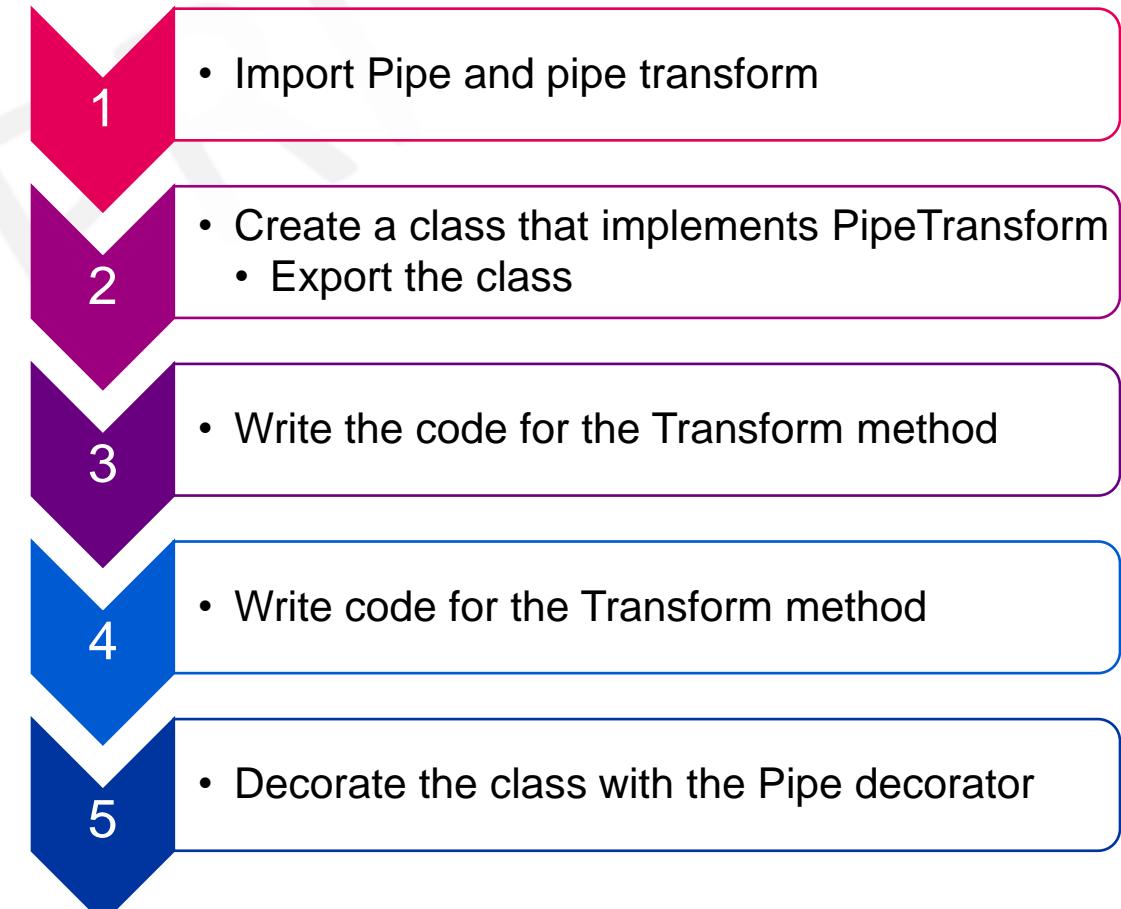
- Subscribes to a Promise or an Observable ,returning the latest value emitted.
- Resolves async data (observables/promises) directly in the template
- Skips the process of having to manually subscribe to async methods in the component and then setting those values for the template to bind to





# Custom Pipes

- Value to transform and Optional arguments.
- Import the pipe decorator and PipeTransform interface
- Import{Pipe,PipeTransform } from 'angular2/core'





# Using a Custom Pipe

1. Use the pipe in the template
  - a. Pipe character
  - b. Pipe name
  - c. Pipe arguments  
(separated with colons)
2. Import the custom pipe
3. Add the pipes property to the component's metadata





# Challenges

- Use two or more built-in pipes to transform data in the template
- Create a custom pipe that filters an array of strings based on a particular letter
- Create an asynchronous method or attribute on the component and bind to it in the template



# A2.0 CSS Styling

Section 11



# Handling Unique Component Styles

- Templates sometimes require unique styles
- Crude approach : inline styles directly into HTML template in component
- Define styles using external stylesheet that the container component would load.
- Alternative : Encapsulating Component Styles





# Encapsulating Component Styles

## styles

```
@Component({  
  selector: 'pm-products',  
  templateUrl: 'app/products/product-list.component.html',  
  styles: ['thead {color: #337AB7;}'])})
```

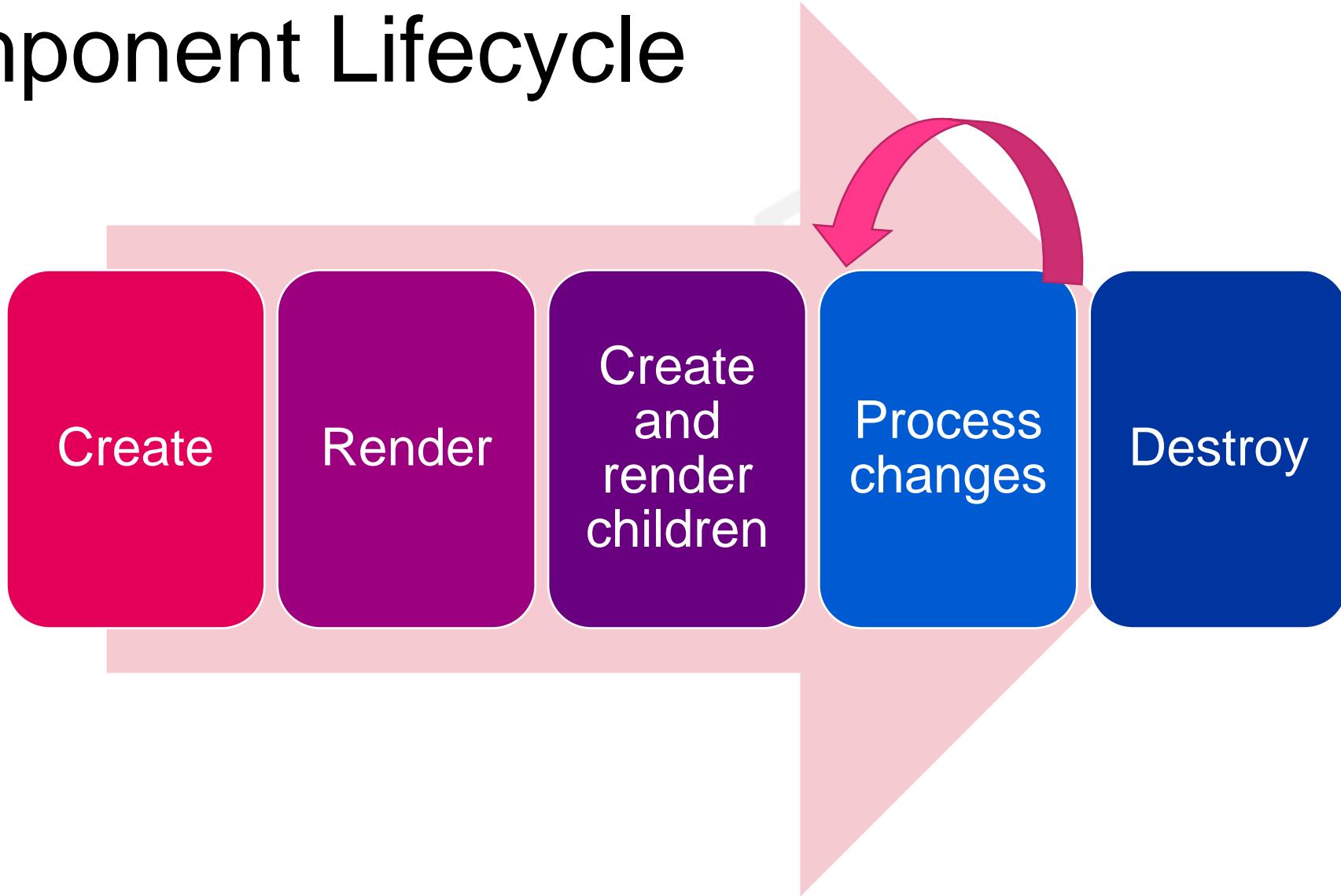
## styleUrls

```
@Component({  
  selector: 'pm-products',  
  templateUrl: 'app/products/product-list.component.html',  
  styleUrls: ['app/products/product-list.component.css']})
```





# Component Lifecycle





# A2.0 Dependency Injection

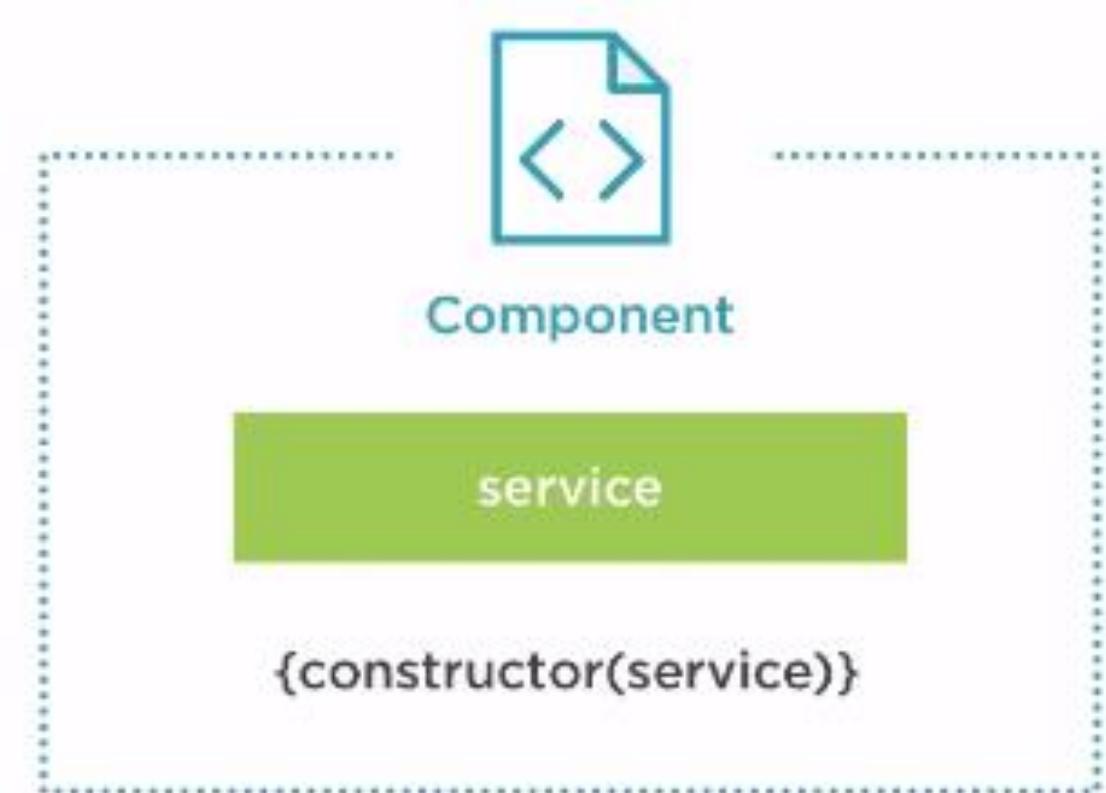
Section XII





# Dependency Injection

- It is how we provide an instance of a class to another Angular feature
- Services in Angular 2 are injected into Component's Constructor.
- In angular 2 we register a service using the Providers in the @Component decorator.
- Using a constructor to inject a service into another object.
- **Register the service with the injector at the parent that contains all components that require the service.**





# A 2.0 HTTP, Promises and Observables. Server Communication

Section XIII





# Server Communication

- The HTTP Module
- Methods
- Observable.toPromise
- Error Handling
- Header

© TPRI





# The HTTP Module Methods

- **Request:** performs any type of http request
- **Get:** performs a request with GET http method
- **Post:** performs a request with POST http method
- **PUT:** performs a request with PUT http method
- **DELETE:** performs a request with DELETE http method
- **PATCH:** performs a request with PATCH http method
- **HEAD:** performs a request with HEAD http method





# Observable

<https://tc39.github.io/proposal-observable/>

- A lazy event stream which can emit zero or more events
- Composed of subjects and observers
- A subject performs some logic and notifies the observer at the appropriate times.

Observables	Promise
Observables handle multiple values over time	Promises are only called once and will return a single value
Observables are cancellable	Promises are not cancellable





# Observable vs Promise

- Observables are lazy- they do not run unless subscribed to while promises run no matter what
- Observables can define both setup and teardown aspects of asynchronous behaviour
- Observables are cancellable
- Observables can be retired, while a called must have access to the original function that returned the promise in order to retry.





# Observable.subscribe

- We finalize an observable stream by subscribing to it
- The subscribe method accepts three event handlers
  - onNext is called when new data arrives
  - onError is called when an error is thrown
  - onComplete is called when the stream is completed





# Observable.toPromise

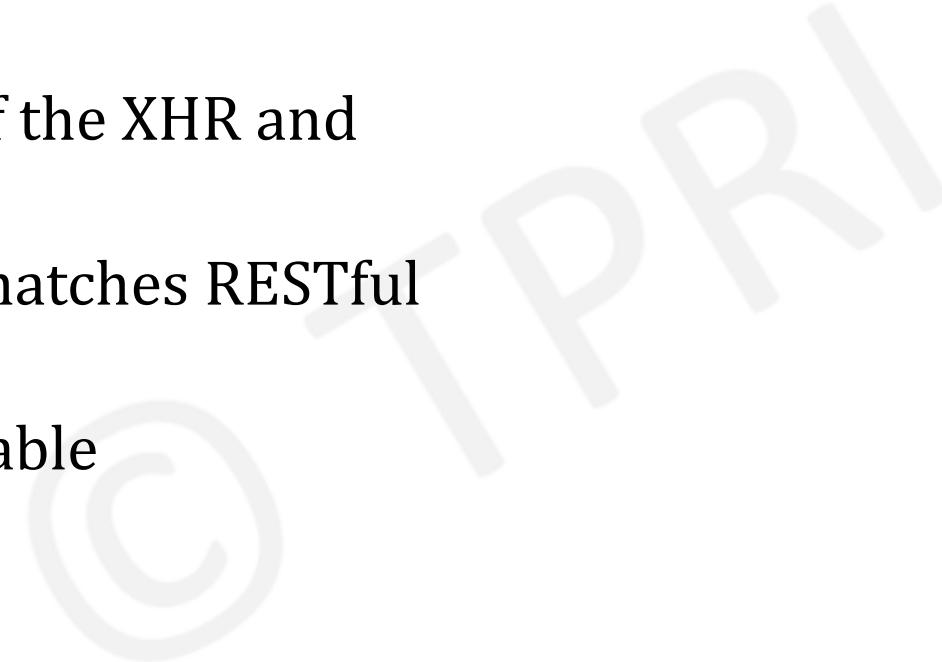
- Diving into observable can be intimidating
- We can chain any HTTP method ( or any observable for that matter) with **toPromise**
- Then we can use **.then** and **.catch** to resolve the promise as always





# The HTTP Module

- simplifies usage of the XHR and JSONP APIs
- API conveniently matches RESTful verbs
- Returns an observable





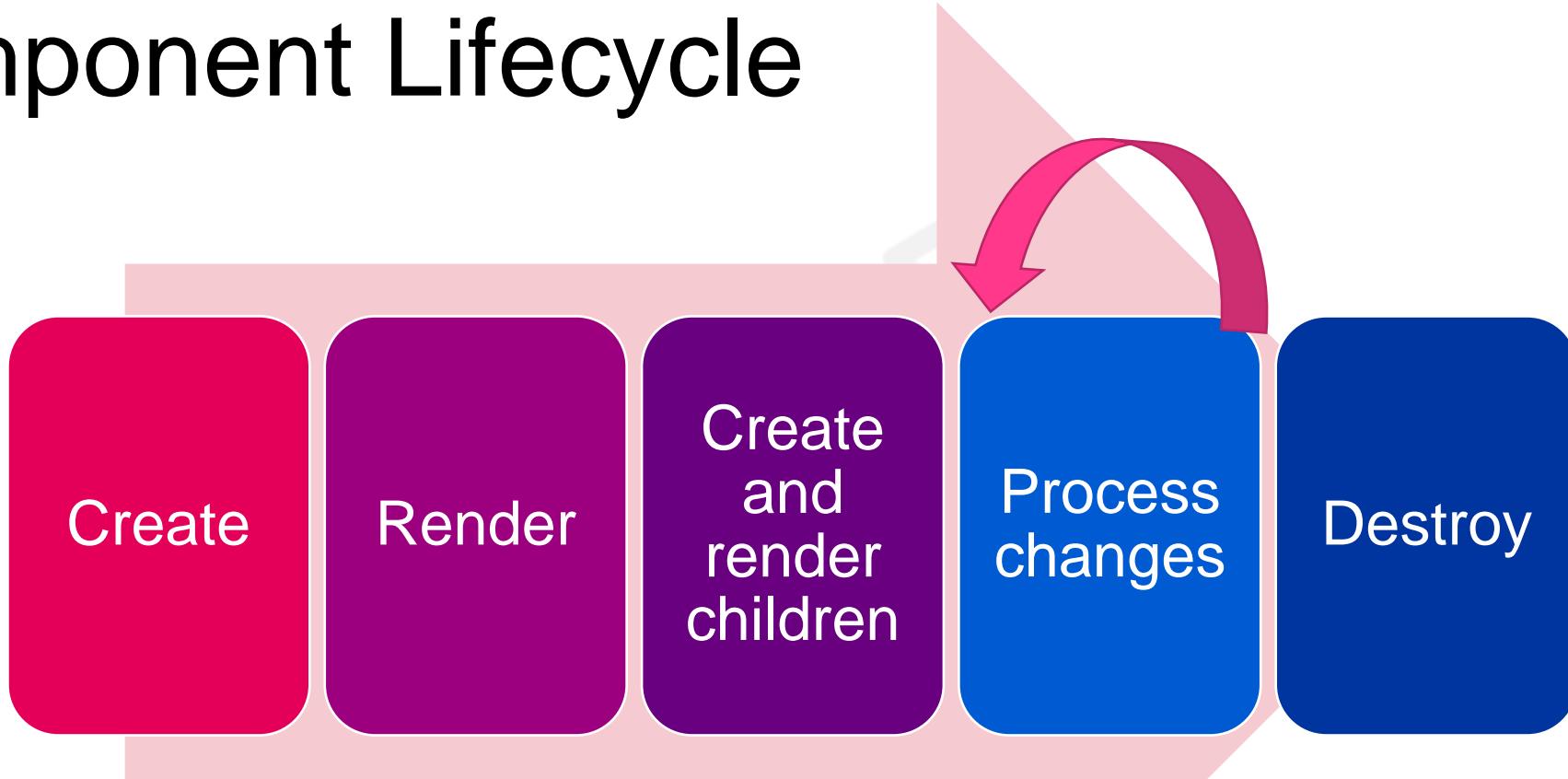
# Component Lifecycle Hooks

Section XIV





# Component Lifecycle



**Lifecycle Hooks allow us to tap into specific moments in the application lifecycle to perform logic**





# Component LifeCycle

1. **OnChanges** – runs first and when a data bound input property value changes
  2. **OnInit** – after the first OnChanges
  3. **DoCheck**- during every Angular change detection cycle
  4. **AfterViewInit**- after init of the component's views and child views
  5. **On Destroy**-Just before Angular destroys the component
1. Import the lifecycle hook interface
  2. Implement the lifecycle hook interface
  3. Write code for the hook method





# Lifecycle Hooks

- Angular calls lifecycle hook methods on directives, components as it **creates, changes and destroys** them.

## Creates:

- **OnInit**
- **AfterContentInit**
- **AfterViewInit**

## Changes:

- **DoCheck**
- **OnChanges**
- **AfterContentChecked**
- **AfterViewChecked**

## Destroys:

- **OnDestroy**





# SPA

Section IV

© TPRI





# Traditional apps



**SERVER**

Request/Response for All  
rendered Content and  
assets

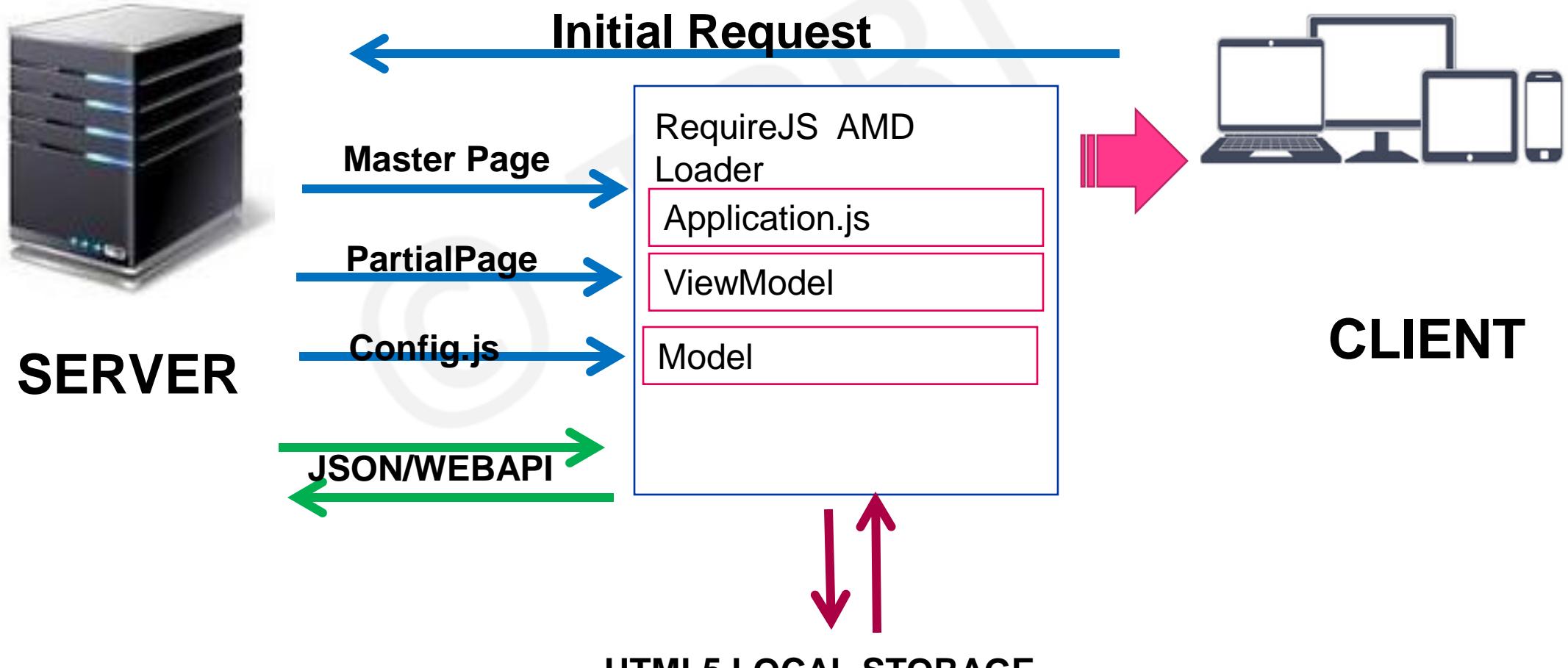


**CLIENT**





# Single Page Application (SPA)





# SPA

- Code Reusability
- Templating
- Interactivity
- Speed
- Bundling and Minification
- Better User Experience
- Hybrid Application
- State based routing
- Challenges include
  - Browser History and Back Navigation
  - Deep Linking
  - SEO
  - Cross-Browser Support





# Histories

A **History** is a user's request to the server in a specific time line, which is recorded by the browser.

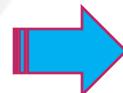
## Different flavors of History are available

**browserHistory**



<http://www.sycliq.com/company/careers>

**hashHistory**



<http://www.sycliq.com/#/company/careers>

**createMemoryHistory**



Custom history API similar to history.js





# Hash Location

- Creates ugly urls with # in them
- Works in all browsers
- Not compatible with server-rendering.
- [www.syqliq.com/#products](http://www.syqliq.com/#products)

# History Location

- Clean URLs
  - IE10+
  - Works for server-rendering (for isomorphic/universal javascript)
  - [www.syqliq.com/products](http://www.syqliq.com/products)
- `back()`  
`forward()`  
`go(index)`  
`pushState(stateObject, title, url)`  
`replaceState(stateObject, title, url)`





# History API

Methods
Window.history.back()
Window.history.forward()
Window.history.go(-1)
Window.history.length
Window.history.pushState()
Window.history.onpopstate()
Window.history.replaceState()

```
> window.history
< ▼ History ⓘ
  length: 1
  scrollRestoration: "auto"
  state: null
  ▼ __proto__: History
    ► back: back()
    ► constructor: History()
    ► forward: forward()
    ► go: go()
    length: (...)

    ► get length: ()
    ► pushState: pushState()
    ► replaceState: replaceState()
      scrollRestoration: (...)

    ► get scrollRestoration: ()
    ► set scrollRestoration: ()
      state: (...)

    ► get state: ()
      Symbol(Symbol.toStringTag): "History"
    ► __proto__: Object
```



# A2.0 Routing

Section XV

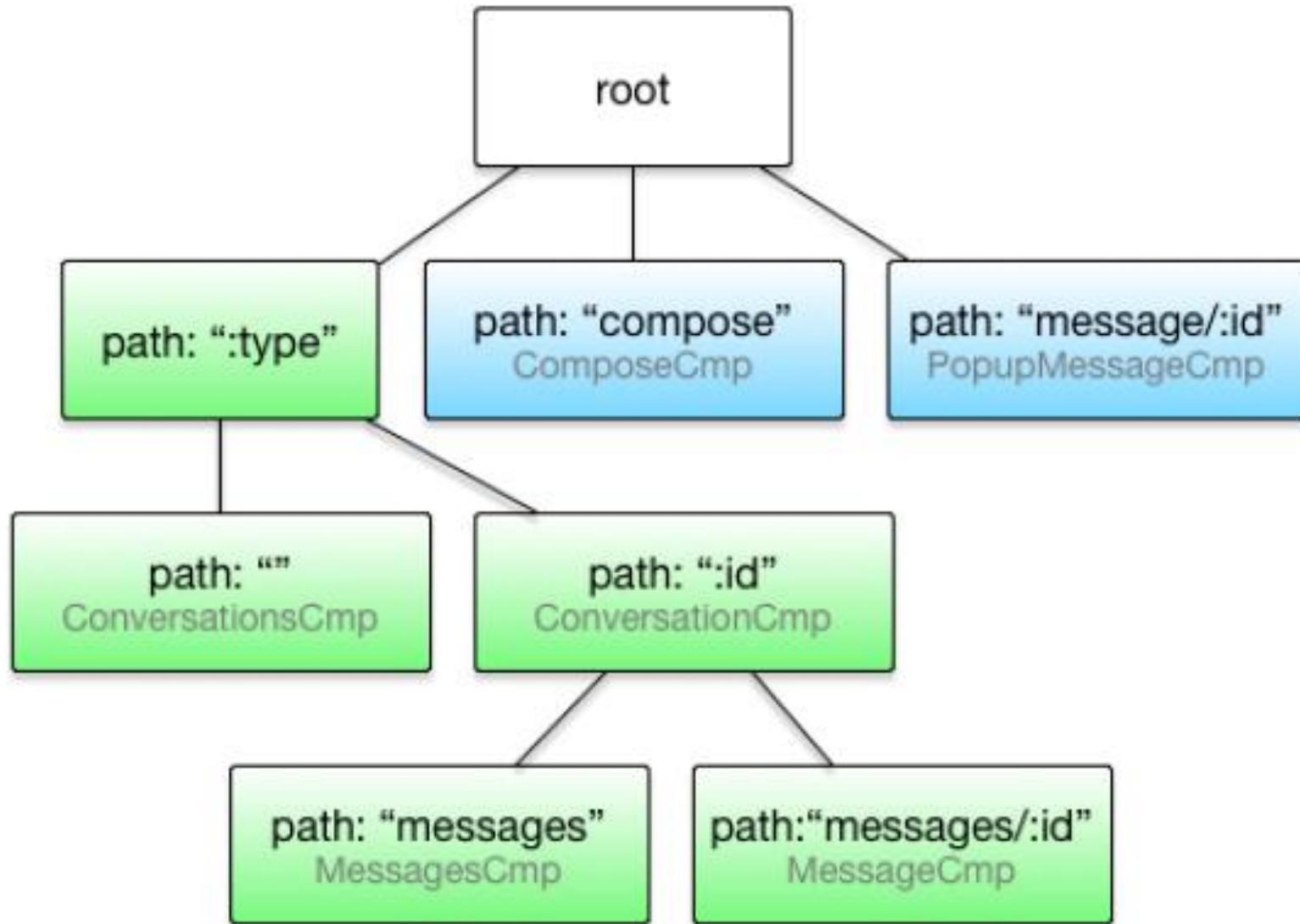


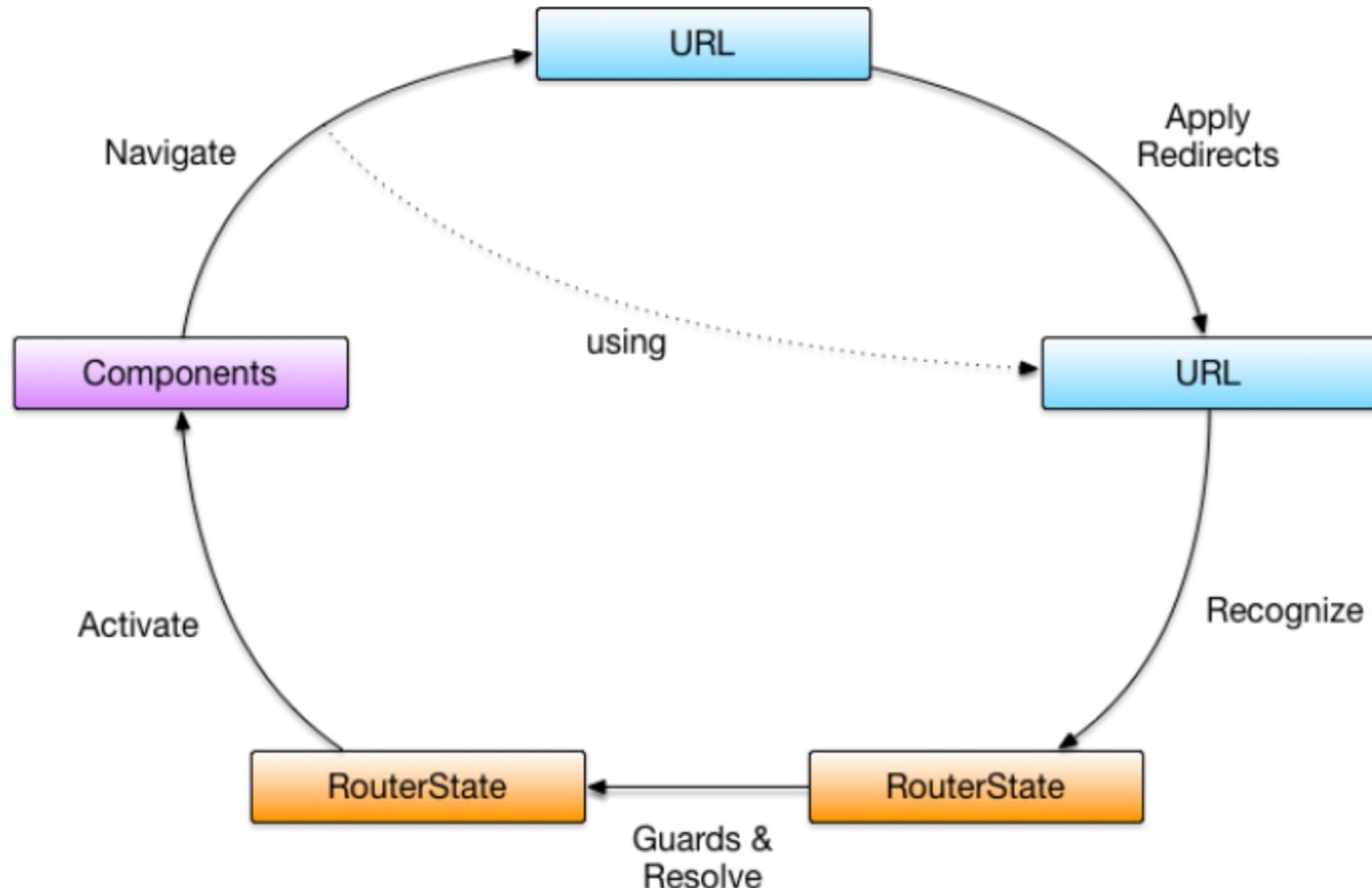
## Routing :

```
@RouteConfig({...})  
<router-outlet>  
[routerLink]="linkParameters"  
RouteParams  
Router
```

Routing allows our application to navigate between different components, passing parameters where needed









# Routing

- Component Router
- Navigating Routes
- Route Parameters
- Query Parameters
- Child Routes

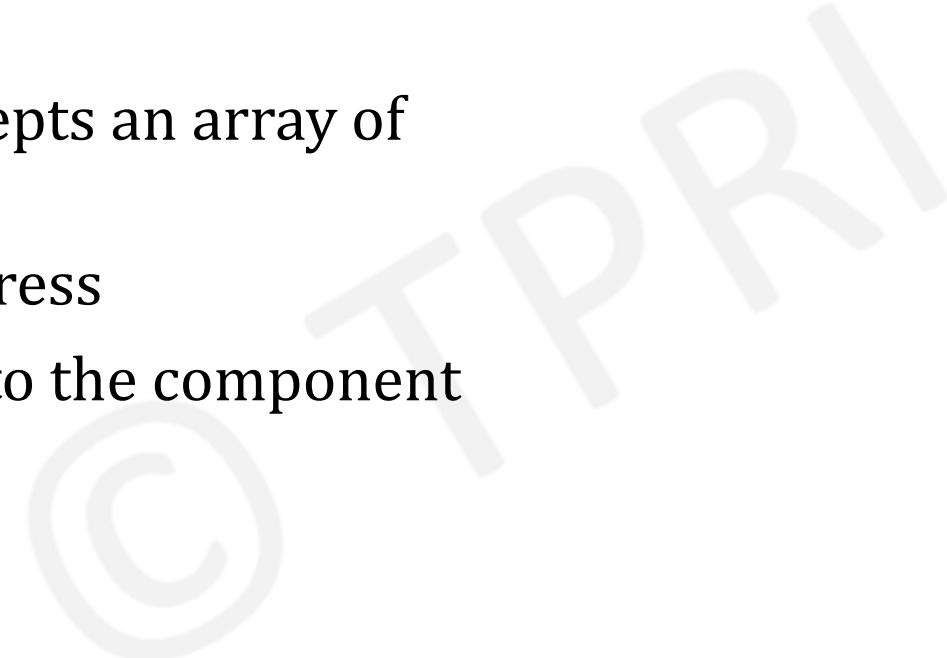
- **Routing Steps**
  1. Configure a route for each component
  2. Define options/actions
  3. Tie a route to each option/action
  4. Activate the route based on user's action
  5. Activating a route displays the component's view





# @RouteConfig

- @RouteConfig accepts an array of route definitions
- Path maps the address
- Component maps to the component and its Template





# Component Router

- Import ROUTE\_PROVIDERS, ROUTE\_DIRECTIVES and the RouteConfig decorator
- Set a base href in the head tag of your HTML
  - <base href="/">
- Configuration is handled via a decorator function (generally placed next to a component) by passing in an array of route definition objects
- Use the router-outlet directive to tell angular where you want a route to put its template <router-outlet></router-outlet>

```
@RouteConfig([
  {path: '/home', name: 'Home', component: HomeComponent, useAsDefault:true},
  {path: '/about', name: 'About', component: AboutComponent},
  {path: '/experiments', name: 'Experiments', component: ExperimentsComponent},
])
export class AppComponent {}
```





# Navigating Routes

- Add a **routerLink** attribute directive to an anchor tag
- Bind it to a template expression that returns an array of route link parameters
  - `<a[routerLink] = "['Users']">Users<a>`
- Navigate imperatively by importing **Router**, injecting it, and then calling `.navigate()` from within a component method
  - We pass the same array of parameters as we would to the routerLink directive  
`this._router.navigate(['Users']);`





# RouterLink

```
<div id="menu">
<a [routerLink]="'/Home'" class="btn">Home<a>
<a [routerLink]="'/About'" class="btn">About<a>
<a [routerLink]="'/Experiments'" class="btn">Experiments<a>
</div>
```





# Router.navigate

```
export class App {  
  constructor(private _router:Router) {}  
  navigate(route){  
    this.router.navigate(['/${route}']);  
  }  
}
```





# Query Parameters

© TPRI





# RouteParams

© TPRI





# Child Routes

© TPRI





# Challenges

- Create a route to the widgets feature
- Use routeLink to navigate to the widgets feature
- Create a method in the items component that imperatively navigates to that route
- Add both route parameters and query parameters to the widgets route





# Directives

Section XVI





Angular 1.x has around 43 Built-in Directives, while Angular 2 Template concepts remove 40+ Angular 1.X Built-in Directives.

`[]` is a property directive  
`()` is a event directive





# Directive?

- A directive is responsible for modifying a dynamic template
- A component is a specific kind of directive with a template
- A directive is a component without a template





© TPRI





# Directives

- What is a Directive?
- Attribute Directives
- Structural Directives
- Custom Directives
- Accessing the DOM





# Structural Directives

- Indicated by the \* prefix
- Changes the structure

© TPRI





# Services

Section XVII

© TPRI





## A2: Service

- Unlike in AngularJS 1.x, we do not have any
  - Provider
  - Constant
  - Value
  - Factory
  - Service
  - Decorator
- Angular 2 : Service
  - A class with a focused purpose
  - Used for features that
    - Are independent from any particular component
    - Provide shared data or logic across components
    - Encapsulate external interactions.





# Services

- A service provides anything our application needs. It often shares data or functions between other angular features.

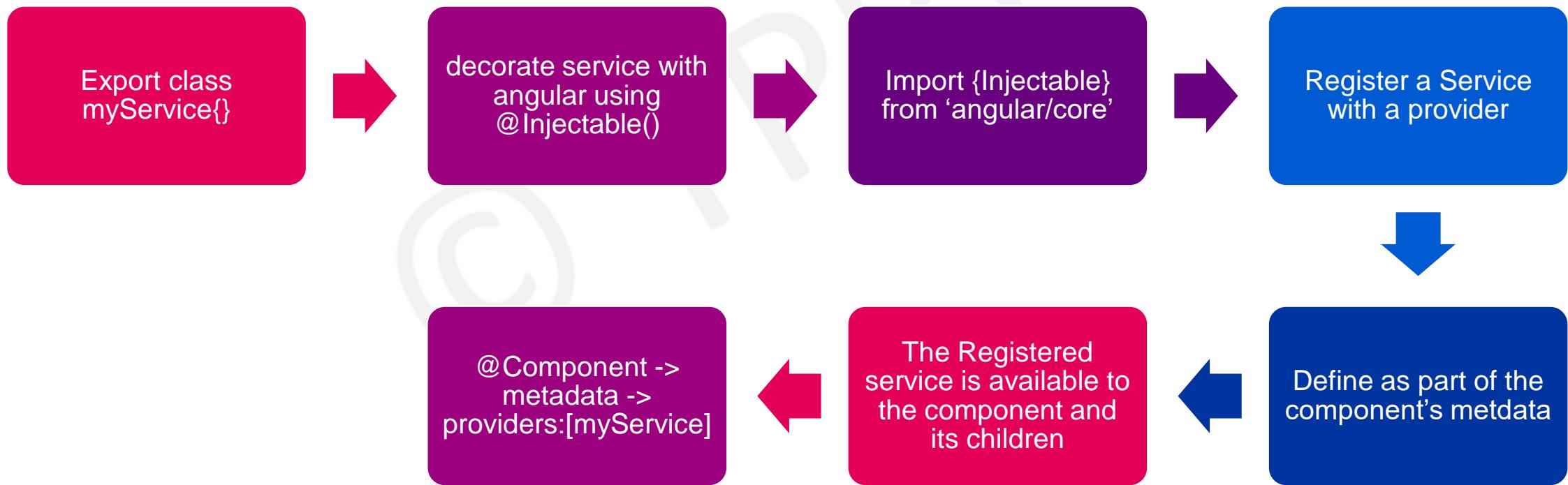
- Example
  - Data logger service
  - Exception handler service
  - Message service

```
//employee.service.ts
import { Injectable } from '@angular/core'
@Injectable()
export class EmployeeService{
  getEmployees () {
    return [
      new Employee(1, 'Syed Awase Khirni', 782728),
      new Employee(1, 'Syed Ameese Sadath', 1232138),
      new Employee(1, 'Syed Azeez', 721282728),
      new Employee(1, 'Syed Rayyan', 32234728)
    ];
  }
}
```





# How does SERVICE work in A2?





<http://rxmarbles.com/>



<http://reactivex.io/rxjs/>

SYED AWASE





# Object.observe()

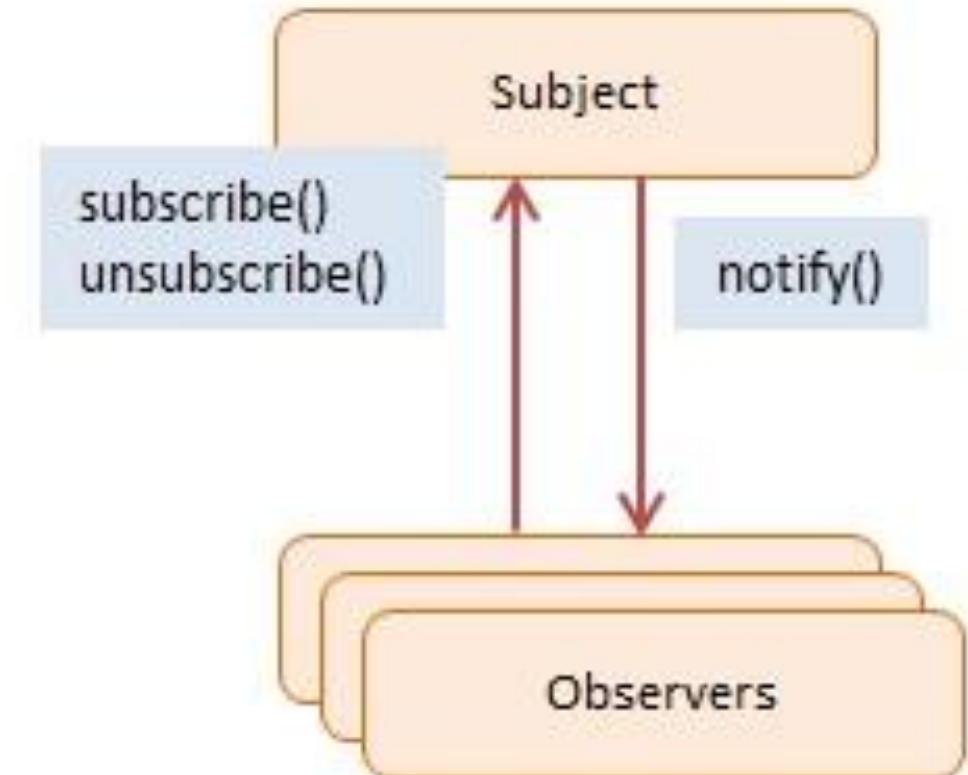
- introduced in ES5, with chrome 36.
- A method for asynchronously observing changes to JavaScript Objects, without the need for a separate library.
- Allows an observer to receive a time-ordered sequence of change records which describe the set of changes which took place to a set of observed objects.
- With Object.observe() – we can implement two-way data-binding without the need for a framework.
- We want to observe()
  - Changes to raw JavaScript Objects
  - When properties get added, changed, deleted
  - When arrays have elements spliced in and out of them
  - Changes to the prototype of the Object





# Observer Pattern/ Pub-Sub Pattern

- It offers a subscription model in which objects subscribe to an event and get notified when the event occurs.
- Event handlers are functions that will be notified when a certain event fires.





# Observer Pattern

- Subject
  - Maintains list of observers
  - Any number of Observer objects may observe a subject
  - Implements an interface that lets observer objects subscribe or unsubscribe
  - Sends a notification to its observers when its state changes.
- Observers
  - Has a function signature that can be invoked when the **Subject** changes (i.e. event occurs)





# Observer Pattern (Pros and Cons)

- Pros

- Very loose coupling between objects
- The ability to broadcast changes and updates

- Cons

- Potentially unexpected updates and sequencing issues.





# Reactive Extensions (Rx)

- A library for composing asynchronous and event-based programs using **observable sequences and LINQ-style query operators**.
- Data sequences can take many forms such as
  - Stream of data from a file
  - Webservice(req,resp)
  - System notifications
  - Series of events (user inputs)
- Reactive Extensions represent all these data sequences as **observable sequences**.
- An application can subscribe to these observable sequences to receive asynchronous notifications as new data arrives.





# Reactive Extensions (Rx)

```
//ES6 via npm
npm install rxjs-es
//commonJS via npm
npm install rxjs
//commonJS with TypeScript
typings install es6-shim --ambient
//All module types (CJS/ES6/AMD/TypeScript)
npm install @reactivex/rxjs
//CDN
https://cdnjs.cloudflare.com/ajax/libs/rxjs/4.1.0/rx.all.js
```





```
console.clear();
var source= ['71238','13212','Syed Awase','Syed Ameese','Syed Azeez','123141223231','true'];

var result = source
    .map(x=>parseInt(x))
    .filter(x=> !isNaN(x)).reduce((x,y)=>x+y);

console.log(result);
```

ES6/JavaScript



# Observables and Reactive Extensions

Section XVIII



# Observables and Reactive Extensions

- Observables are an array whose items arrive asynchronously over time. They help us manage asynchronous data and is a proposed feature for ES 2016 (next version of JS).

<http://rxmarbles.com>

- Angular uses Reactive Extensions (RxJS)- a third party library





# Promise

- Promises returns a single value
- Promises are not cancellable

# Observable

- Observables work with multiple values over time.
- Observables are cancellable
- Observables support map, filter, reduce and similar operators.

