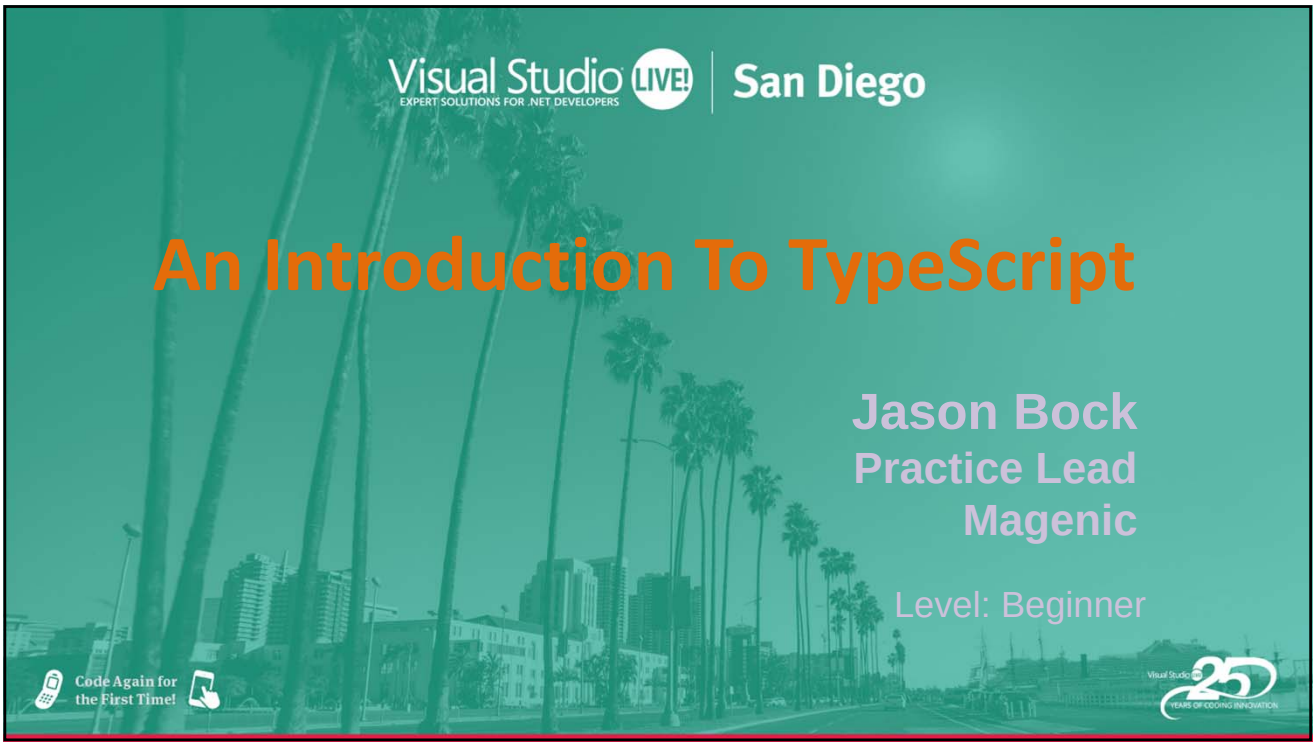


Personal

- http://www.magenic.com
- http://www.jasonbock.net
- https://www.twitter.com/jasonbock
- https://www.github.com/jasonbock
- jasonb@magenic.com

Downloads

https://github.com/jasonbock/IntroToTypeScript

https://github.com/JasonBock/Presentations/blob/master/An%20Introduction%20to%20TypeScript.pptx

Overview

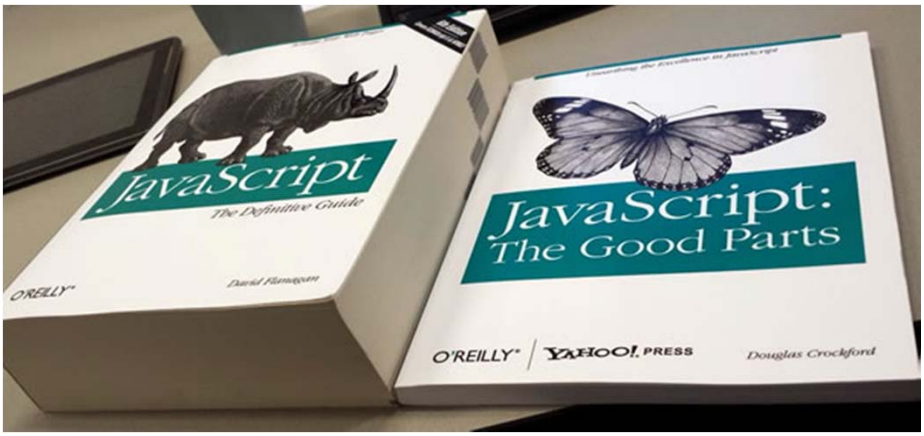
- Why TypeScript?
- Language Features
- Usage
- Future

Remember…
https://github.com/jasonbock/IntroToTypeScript

https://github.com/JasonBock/Presentations/blob/master/An%20Introduction%20to%20TypeScript.pptx

Why TypeScript?

Why TypeScript?



http://www.globalnerdy.com/wordpress/wp-content/uploads/2014/08/javascript-and-the-good-parts.jpgc

Why TypeScript?

```
console.log('4' - 4);
console.log('4' + 4);
```

```
0
44
```

Why TypeScript?

Why is the method called `includes` and not `contains`?

The latter was the initial choice, but that broke code on the web (MooTools adds this method to `Array.prototype`).

http://www.2ality.com/2016/02/array-prototype-includes.html

Why TypeScript?

TypeScript

"TypeScript is a typed superset of JavaScript that compiles to plain JavaScript."

http://www.typescriptlang.org/

An Introduction To TypeScript

DEMO: USING THE PLAYGROUND

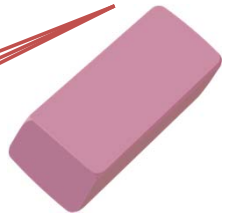
Features

Add.ts

```
function add(x: number, y: number): number {
  return x + y;
}
```

Add.js

```
function add(x, y) {
    return x + y;
}
```



https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Pink-eraser.svg/570px-Pink-eraser.svg.png

An Introduction To TypeScript

DEMO: USING VS CODE FOR TYPESCRIPT FEATURES

Features

- Variables
  - Straightforward – “var answer = 44;”
  - Can be explicit about type – “var answer : number = 44;”
  - Basic types: boolean, number, string
  - Arrays
    - var answers : number[] = [44, 42]; (can be inferred as well)
    - Array<> can be used (generics will be covered later)
  - Enums
    - enum Answers { Correct = 44, Wrong = 42 }
    - If no value is specified, number starts at 0
  - Any and void
    - Any lets anything go (like dynamic in C#)
    - void – used for methods that return nothing (not required)

Features

- Basic language features
  - Template strings: `The answer is {this._value}` (note the tick mark, it’s required)
  - Ternary: this._value = value == null ? 44 : value;
  - let and const
    - let allows for scoped variables
    - const allows for constant values
    - Object.freeze() should be used for immutable values
  - Try…catch…finally is supported

Features

- Methods
  - Can take any number of arguments, and can also specify types if desired
  - Return type can be deferred from “return” statements
  - Anonymous methods can be declared
  - Arguments
    - Methods must be called with correct number of parameters
    - Optional arguments can be declared with “?” after the name
    - Or, use a default argument value (makes it optional)
    - “Rest” parameters, use “…” to grab multiple optional arguments

Features

- Classes and interfaces
  - Interfaces define structural contracts (“implements”)
  - Classes define structural contracts and behavior (“extends”)
  - Visibility
    - Public (anyone), private (only that class), protected (that class and descendants)
  - Properties
    - Use “get” and “set”
    - Visibility must be same (can’t mix)
    - Can use “readonly” fields

Features

- Classes and interfaces (con't)
  - Overloading methods
    - It's possible, but really awkward
    - Just better off declaring a method with a different name
  - Mixins
    - "implement" multiple classes
    - Really you're implementing the structural contract of a class

Features

- Modules
  - Analogous to namespaces in C#
  - Only way to declare a const within some "scope"
  - Internal
    - Just use "module". Other classes within that TS file will see members within the module IF they use "export"
    - Use triple-slash reference to help the compiler and tooling
  - External
    - Use the import keyword to "require" the module
    - You can use require(), but "import * as someName from "…"" is a cleaner way

Features

- Types and Generics
  - Type guards – this or that or something else -> A | B | C
  - Type assertions – “duck typing”. This isn’t casting, you’re just telling the compiler you know more.
  - Generics
    - You can use generic parameters in TypeScript to create reusable, safe code
    - I.e. you don’t have to use “object” or “any”; you can be specific with the type used
    - E.g. Array<string> or Array<Person>
    - Constrained - <T extends Person>
    - “new”

Features

- Iterators
  - For statements…(for i = 0;…)
  - For-in statements….but be careful what you’re iterating, it’s the keys
  - For-of statements…works like “foreach” in C#

Features

- Asynchronous code
  - JS traditionally used callbacks
  - Then we moved to promises
  - Now TS (and JS) has async and await

Features

- .d.ts files
  - If you're doing everything in TS (ExternalModules example), everything….works
  - If you're using a JS library like moment.js (http://momentjs.com/), you need the .d.ts file for TS to get type information
    - npm install typings –global
    - typings install moment
  - If you want to generate a .d.ts
    - From TS files, use "declaration" compiler switch
    - From C#, consider TypeLite (http://type.litesolutions.net/)
  - More info: http://definitelytyped.org/

Features

- Decorators
  - Origination was arguably with AtScript
  - Annotations, or decorations, were put into TypeScript
  - https://github.com/Microsoft/TypeScript/issues/1557
  - Can decorate properties, classes, methods and parameters
  - Is somewhat advanced, but it's worth diving into
    - Interception capabilities
    - Powerful extension mechanisms (e.g. dependency injection)
    - Metaprogramming
    - Serialization

Usage

The generated file looks like this:

src/app/app-routing.module.ts (generated)

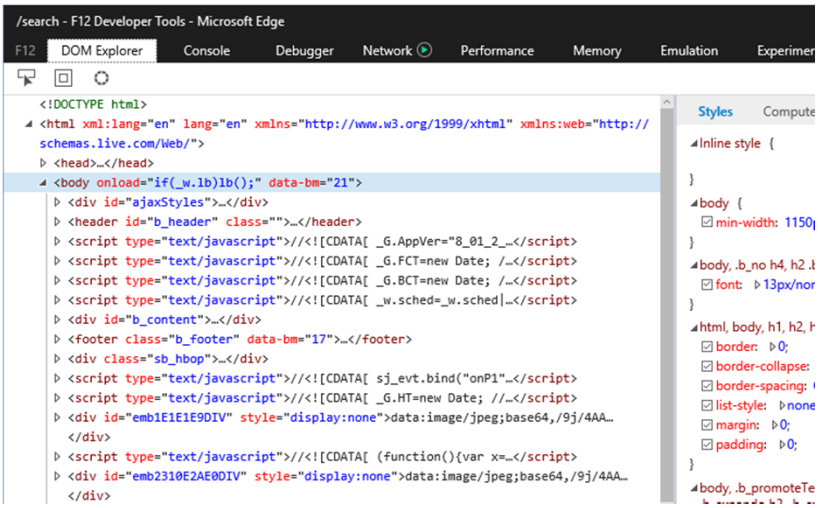
```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: []
})
export class AppRoutingModule { }
```

You generally don't declare components in a routing module so you can delete the `@NgModule declaration`

https://angular.io/

Usage



https://dev.windows.com/en-us/microsoft-edge/platform/documentation/f12-devtools-guide/

Usage

Friends of TypeScript

https://www.typescriptlang.org/community/friends.html

Future

ES2019

JS

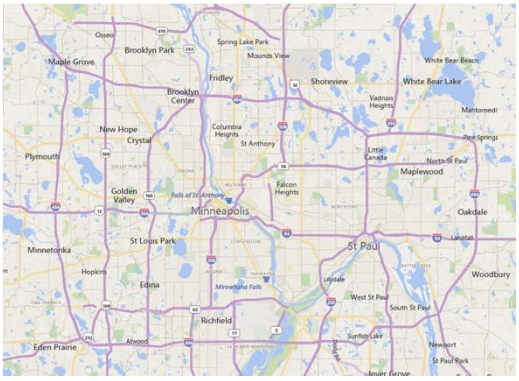
Async Functions

Shared Memory

Atomics

https://tc39.github.io/ecma262/

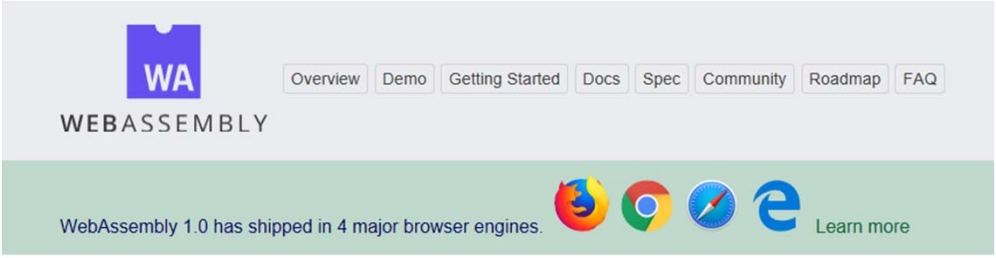
Future



- 3.1
  - Mapped tuple types
  - Support for BigInt
  - Named type arguments & partial type argument inference
  - Property assignments on function declarations
  - Refactorings to...
    - Rename files from import/export paths
- Future
  - Variadic types
  - Investigate nominal typing support
  - Flattening declarations
  - Implement ES Decorator proposal
  - Implement ES Private Fields
  - Investigate Ambient, Deprecated, and Conditional decorators
  - Investigate error messages in haiku or iambic pentameter
  - Decorators for function expressions/arrow functions

https://github.com/Microsoft/TypeScript/wiki/Roadmap

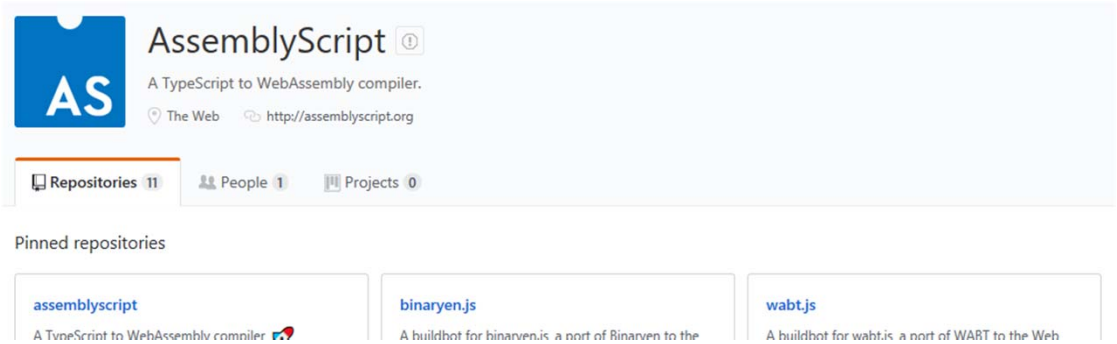
Future



WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

https://webassembly.org/

Future



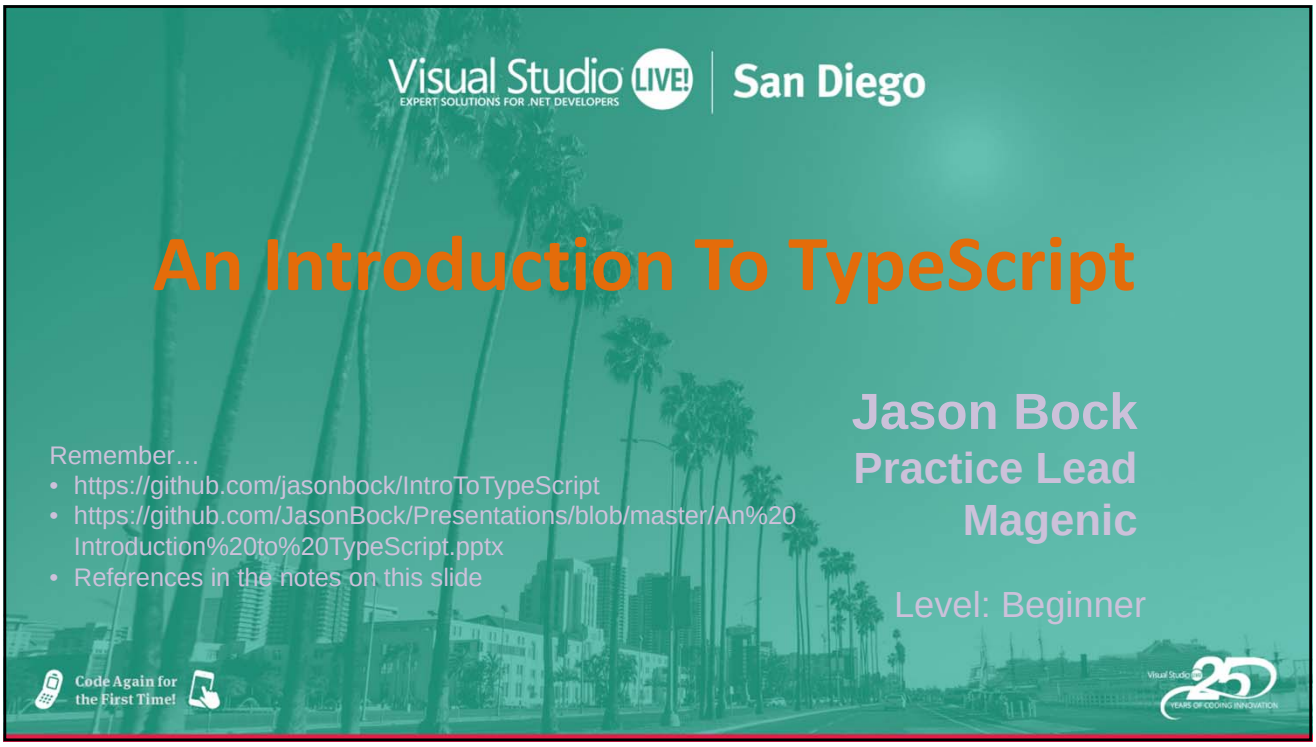
https://github.com/AssemblyScript

An Introduction To TypeScript

DEMO: PLAYING WITH ASSEMBLYSCRIPT

Conclusion





Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS
San Diego
An Introduction To TypeScript
Jason Bock
Practice Lead
Magenic
Level: Beginner
Remember…
• https://github.com/jasonbock/IntroToTypeScript
• https://github.com/JasonBock/Presentations/blob/master/An%20Introduction%20to%20TypeScript.pptx
• References in the notes on this slide
Code Again for the First Time!