TypeScript

The How and the Why

Jonathan "J." Tower

Hi, I'm J.

Jonathan "J." Tower
Principal Consultant & Partner
Trailhead Technology Partners

TRAILHEAD
TECHNOLOGY PARTNERS

trailheadtechnology.com

- Microsoft MVP in ASP.NET
- Telerik/Progress Developer Expert
- Organizer of Beer City Code

- jtower@trailheadtechnology.com
- trailheadtechnology.com/blog
- jtowermi

Summary

- The Why
  - The Basics
  - When to use it
  - When not to use it
  - Advantages/Disadvantages
- The How
  - Getting started
  - Language features
  - Demos

The Why

Basics, When to Use, Advantages/Disadvantages

TypeScript Basics

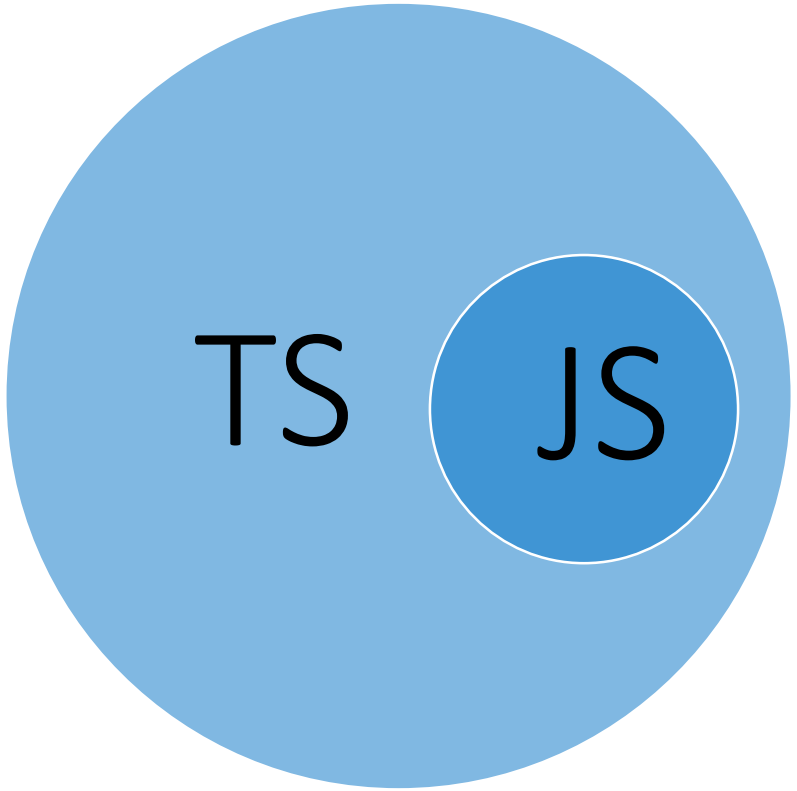
Type·Script *n.*

\tīp-skript\

1. JavaScript that scales.
2. Something someone said I should like.
3. Something someone said I shouldn’t like.

Let’s try to delve deeper than #2 and 3…

TypeScript: A Superset of JavaScript



- All valid JavaScript is also valid TypeScript
- TypeScript only *adds* to JavaScript

TypeScript: Transpiled

Browsers don’t run typescript

.ts becomes .js at compile-time

Transpiles to

ES3

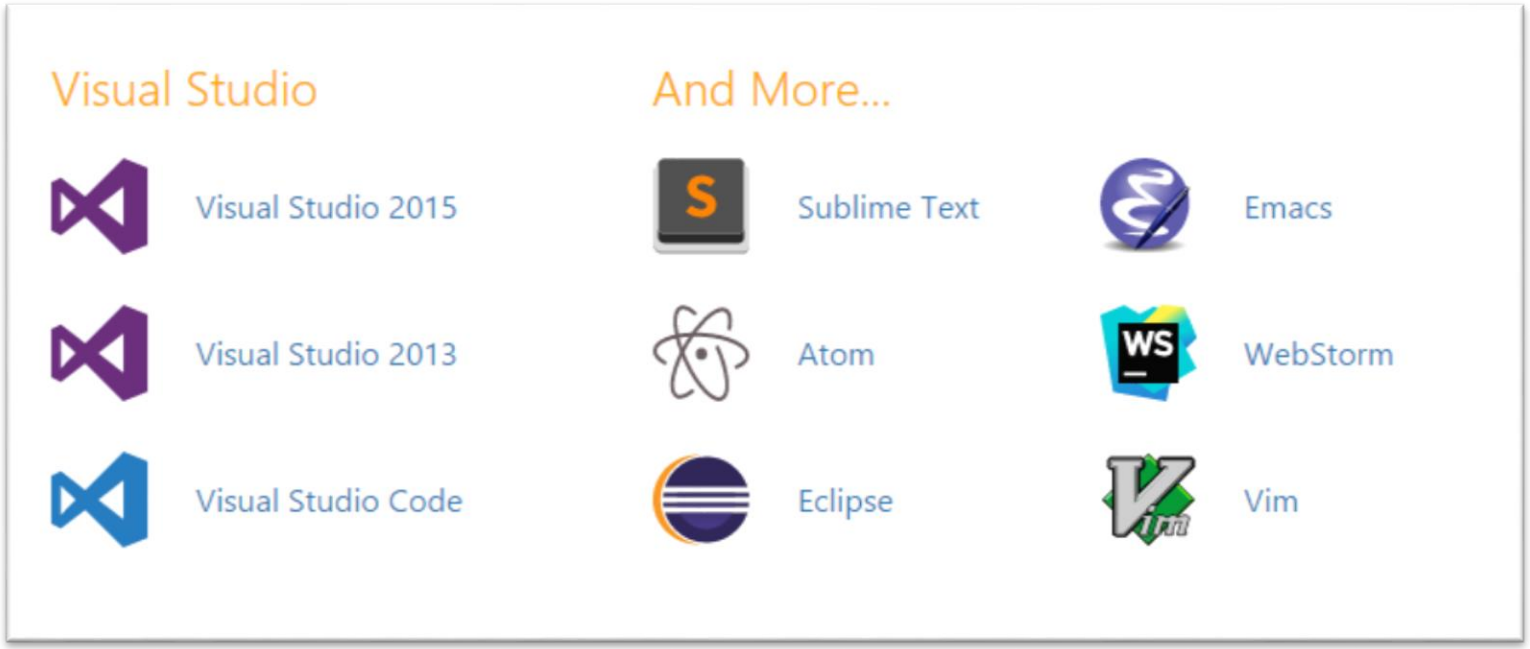
ES5

ES6 (ES2015)

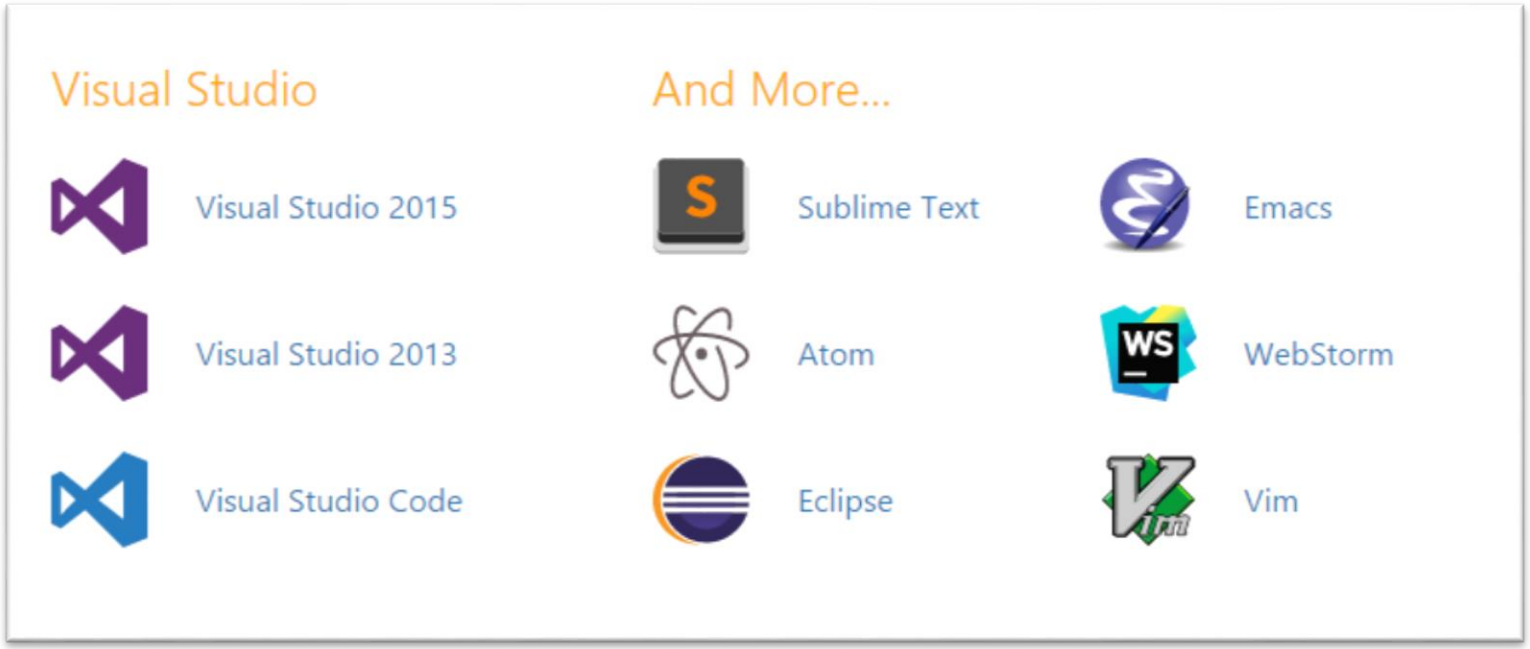
Even ES2016 and ES2017!

Runs in Node or Browser, no runtime needed

TypeScript: Not Just for MS Devs

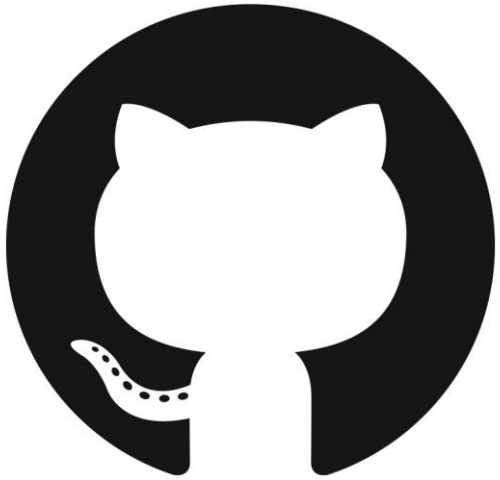


TypeScript: Not Just for MS Devs



Chosen by the Angular2 team (at Google!)

TypeScript: Open Source



Hosted

License

TypeScript: Statically-Typed

Static – type checking is performed during compile-time

Dynamic – type checking is performed at run-time only

TypeScript: Statically-Typed

Static – type checking is performed during compile-time

Dynamic – type checking is performed at run-time only

JavaScript = Dynamic / TypeScript = Static

TypeScript: Addition of OO to JS

Keywords like **class**, **interface**, **extends** and **module** are available in TypeScript

TypeScript: ES6 Support

Arrow functions

Classes

Template strings

Modules

let

const

Much more…

TypeScript: Packaging Support

```
> tsc main.ts --outFile everything.js
```

TypeScript: Similar to Back-End Code

Benefits of many back-end languages to the front-end

(Assuming you're using a statically typed back-end language such as Java, Scala, C#, etc)

Superset of JavaScript

Superset of JavaScript

Smooth learning curve for JavaScript devs

Why Not Just Use JavaScript?

Transpiled languages can add features

Transpiled languages are free to innovate

Transpiled languages can catch compile-time bugs

When To Use TypeScript

And When Not To

Use TypeScript if…

You have a large codebase

Your team's developers are already accustom to statically-typed languages

TypeScript can serve as a replacement for Babel

Your library or framework recommends TypeScript

You really feel the need for speed

DON’T Use TypeScript if…

You want to avoid the extra transpilation tax

You have a pure-JS team

You want to maximize your team’s agility

The How

Getting Started, Languages Features, Live Demo

Getting Started

Getting Started

Visual Studio

OR

```
> npm install -g typescript
> tsc mycode.ts
```

Getting Started

DEMO

Language Features

Type System

Type	TypeScript Type
Object	any
Void	void
Null and Undefined	null and undefined
Functions that don’t return	never
Boolean	boolean
Integer, Float, Long, Double, etc	number
Strings and Chars	string
Arrays	someType[] Array<someType>
Tuple	[string, number]
Enum	Color {Red, Green, Blue}

Optionally Typed

You don’t have to specify types for properties, variable, parameters, etc.

```
function divideByTwo(x) { return x / 2 }

function divideByTwo(x: number):number { return x / 2 }
```

Classes

Uses ES6 Syntax

Can Implement Interfaces

Inheritance

Instance members

Static members

Single constructor

Default and optional parameters

let vs var

```
// block scope, explicit type
let x: string = "hello, world";

// block scope, implicit type
let x = "hello, world";

// hoisted to function scope
var x = "hello, world";
```

const

```
const maxNumberOfTries = 9;

const config = {
    maxTries: 1
};

config = { };

config.maxTries = 2;
```

Interfaces

```
interface Border {
    color: string;
    width: number;
}

class Box implements Border {
    public color: string;
    public width: number;
    public getCss(): string {
        return `border-color:{color}; border-width:{width}px`;
    }
}
```

Access Modifiers and Properties

```
class Animal {
    private name: string;
    public constructor(theName: string) {
        this.name = theName;
    }
    public move(meters: number) {
        console.log(`${this.name} moved ${meters}m.`);
    }
}
```

Constructors

```
class Animal {
    public constructor(theName: string) {
        this.name = theName;
    }
}

class Tiger extends Animal {
    constructor(name: string) { super(name); }
}
```

Static and Instance Members

```
class Grid {
    static origin = {x: 0, y: 0};
    distanceFromOrigin(point: {x: number; y: number;}) {
        let xDist = (point.x - Grid.origin.x);
        let yDist = (point.y - Grid.origin.y);
        return Math.sqrt(xDist * xDist + yDist * yDist);
    }
}
```

Abstract Classes

```
abstract class Animal {
    abstract makeSound(): void;
    move(): void {
        console.log("roaming the earth...");
    }
}
```

Function Overloading

```
class Foo {
    myMethod(a: string);
    myMethod(a: number);
    myMethod(a: number, b: string);
    myMethod(a: any, b?: string) {
        alert(a.toString());
    }
}
```

Inheritance

```
class Parent {
    constructor(){}
    Foo(value : any){}
}
class Child extends parent {
    constructor() { super(); }
    Foo(value : string ) : void;
    Foo(value : number) : void;
}
```

Inheritance

```
class Parent {
    constructor(){}
    Foo(value : any){}
}
class Child extends parent {
    constructor() { super(); }
    Foo(value : any ){} //actual function implementation
}
```

Enums

```
enum Direction {
    Up = 1,
    Down,
    Left,
    Right
}
```

```
enum FileAccess {
  None,
  Read    = 1 << 1,
  Write   = 1 << 2,
  ReadWrite  = Read | Write
}
```

Arrow Functions (Lambdas)

```
export class Sample {
    search(complete: (count: number) => void) {
        console.log('Searching...');
        complete(5)
    }
}

let s = new Sample();
s.search((v) => { console.log(`Found ${v}`) });
```

Modules

Uses ECMAScript2015 modules syntax

Definition is “exported”

Dependencies are “imported”

Imported modules loaded with a module loader
- CommonJS (for Node)
- require.js (for web)

Modules

DEMO

Summary

- The Why
  - The Basics
  - When to use it
  - When not to use it
  - Advantages/Disadvantages
- The How
  - Getting started
  - Language features
  - Demos

Thanks! Questions?

Jonathan "J." Tower
Principal Consultant & Partner
Trailhead Technology Partners

TRAILHEAD
TECHNOLOGY PARTNERS

trailheadtechnology.com

- Microsoft MVP in ASP.NET
- Telerik/Progress Developer Expert
- Organizer of Beer City Code

- jtower@trailheadtechnology.com
- trailheadtechnology.com/blog
- jtowermi