TypeScript

Full stack development using TypeScript

OpenSlava 16

CHAD MOTT

Manager @ Emerging Technology

3 years with Accenture

Product owner for NodeJS Architecture within ET LWA

AWS Certified Solutions Architect, MTA Tech Arch certified, MCP JS/HTML5

New TypeScript advocate and enthusiast

Lives in St. Louis

Chad.Mott@accenture.com

DANIEL DEREVJANIK

Application Development Associate

1.5 years with Accenture

Working with emerging technologies

Focusing on functional programming

Lives in Bratislava

Daniel.Derevjanik@accenture.com

OTHER CONTRIBUTORS

Patrick Opie
Seattle Based ET Analyst
Patrick.opie@accenture.com

Tom Manion
Chicago based ET Consultant
Thomas.w.manion@accenture.com

Session Objectives

By the end of this session, my hope is that you will:

- Be comfortable with the basics of TypeScript
- Be able to implement TS on your projects
- Know why to large-scale JavaScript development is easier with TypeScript

Session Agenda

- Give an Overview of TypeScript – What it is, how to use it, general overview
- Create a Full Stack application using Type Script in 3 Contexts (this is the fun part)
  - Create a ”Dynamic DNS” clone using AWS Lambda
  - Create a home automation API using NodeJS
  - Create a home automation App using Angular 2
- Review: What have we learned and TypeScript next steps
  - Learning resources
  - What’s coming up in TS 2.1

Housekeeping

- This Deck, along with all material, can be found in this repo https://github.com/chadmott/OpenSlava-exercises
- Prerequisites
  - A ”linux like” environment – Mac, CYGWIN/GitBash on Windows
    - We don’t have time to troubleshoot node issues on your PC across the many different types of shells
  - TypeScript installed
    - npm install -g typescript
  - An AWS “Free Tier” account
    - Not required, as we can run the example locally, but I will walk through how to “wire this up” and having the account will be helpful for you

TypeScript Overview

TypeScript – What is it?

TypeScript

- First released in 2012 by Anders Hejlsberg at Microsoft
- Designed to overcome the limitations of JavaScript for large-scale applications.
- Superset of JavaScript, any existing JavaScript programs are also valid TypeScript programs.
- Allows inferred typing on all function calls, verified at compile-time.
- Next Gen ES features transcompiled to today’s browser support.

ECMAScript 6

To understand **TypeScript**, we need to understand how it relates to the current standard of JavaScript:
ECMAScript 6

JavaScript Keywords (ECMAScript 5)

- break
- case
- catch
- continue
- debugger
- default
- delete
- else

- finally
- for
- function
- if
- in
- instanceof
- new
- return

- switch
- this
- throw
- try
- typeof
- var
- while
- with

ECMAScript 6 Keywords

- break
- case
- class
- catch
- const
- continue
- debugger
- default
- delete
- else
- export

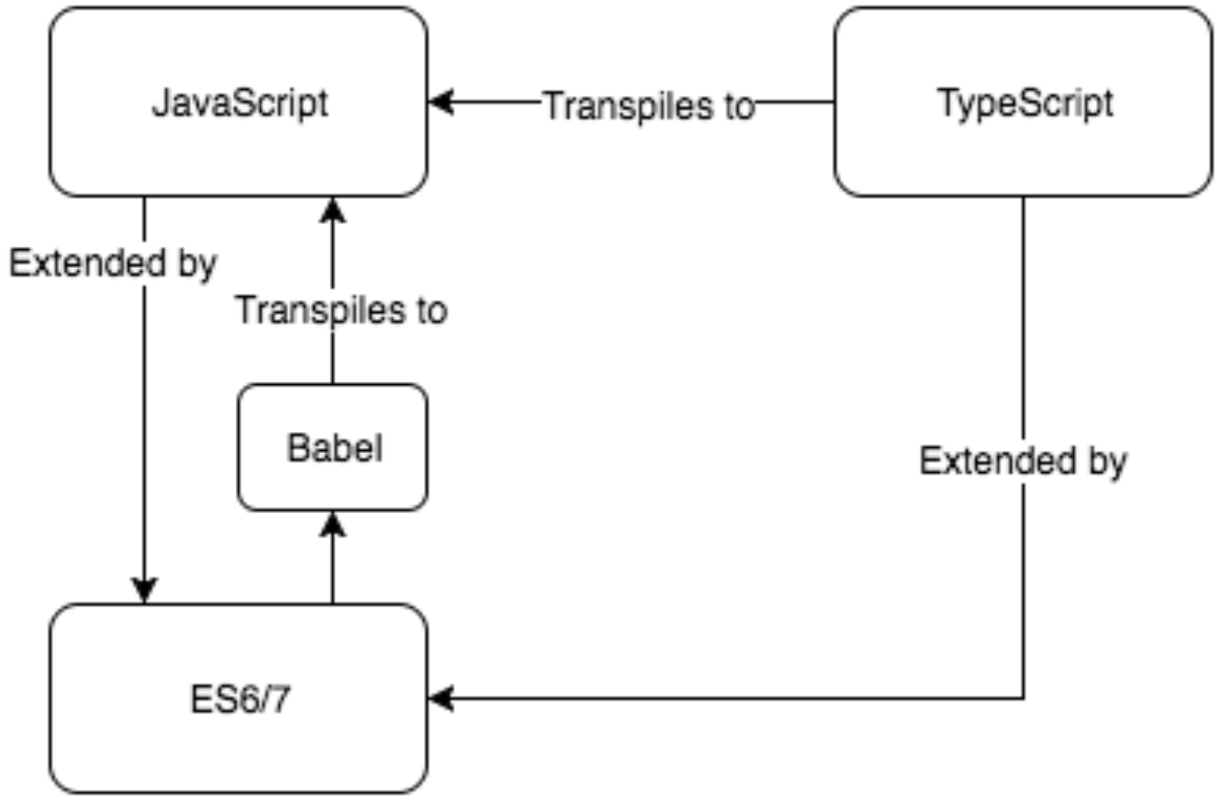
- extends
- finally
- for
- function
- if
- import
- in
- instanceof
- let
- new
- return

- super
- switch
- this
- throw
- try
- typeof
- var
- while
- with
- yield

ECMAScript 6 Features

- Arrow Functions
- Classes
- Enhanced object literals
- Template strings
- Destructuring
- default + spread + rest
- let + const
- iterators + for… of
- Generators
- Unicode
- Modules
- Module Loaders
- map + set
- Proxies
- Symbols
- Subclassable built-ins
- Promises & Streams
- new object APIs
- Binary and Octal Literals
- Reflect API
- Tail calls

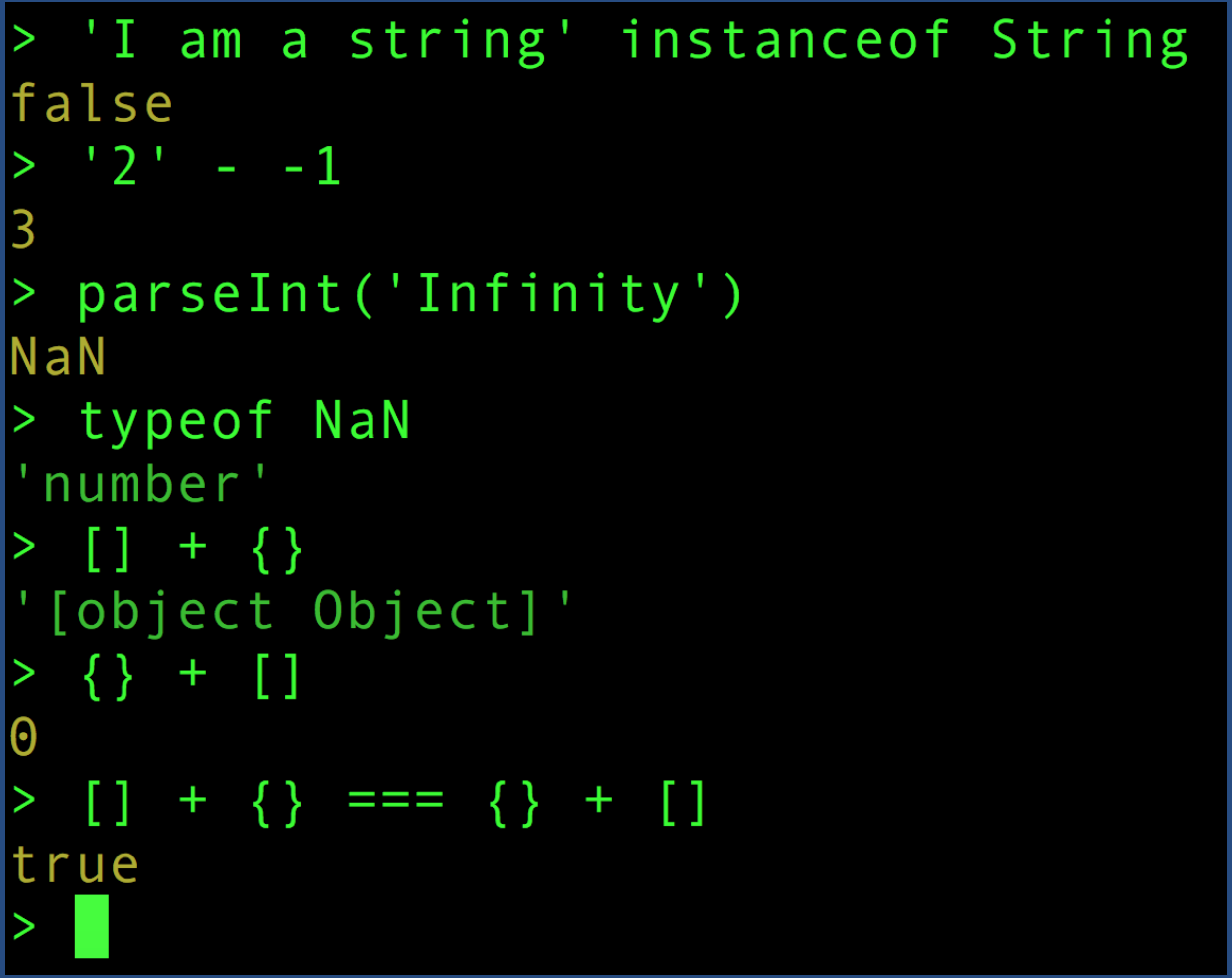
Overview of Languages and Transpilers



JavaScript Idiosyncrasies

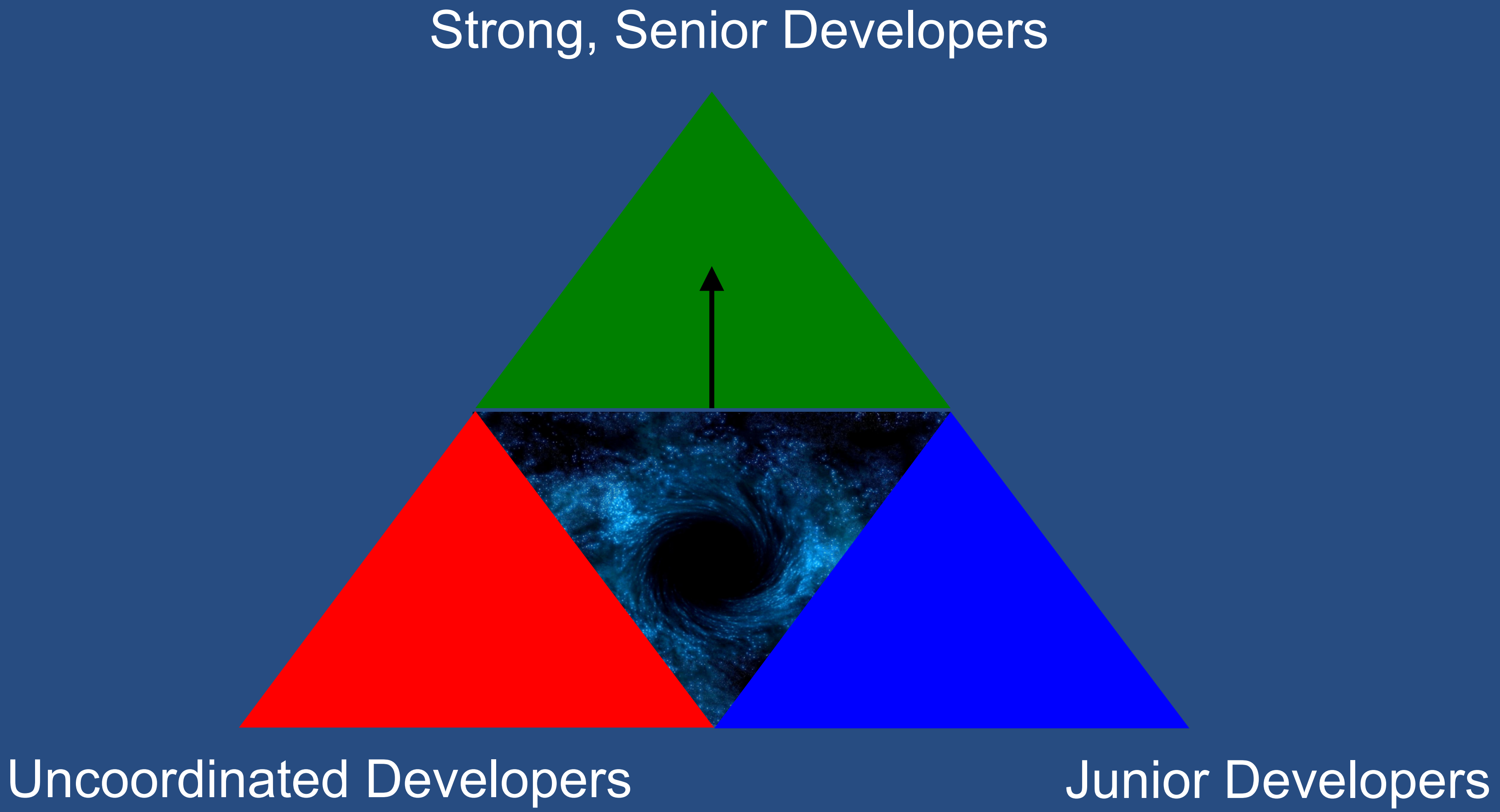
dotJS 2012 - Brian Leroux - WTFJS

https://www.youtube.com/watch?v=et8xNAc2ic8

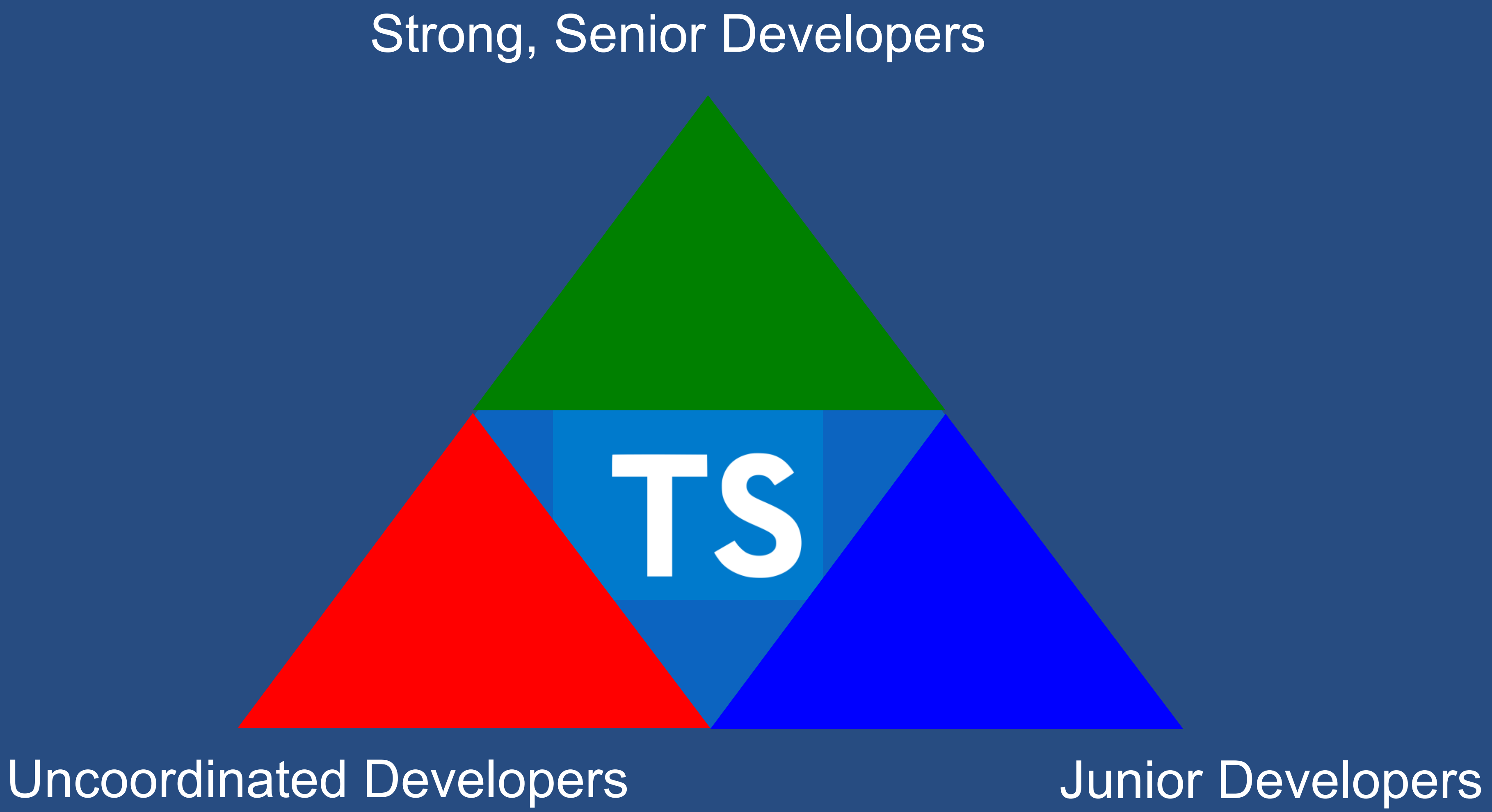


```
> 'I am a string' instanceof String
false
> '2' - -1
3
> parseInt('Infinity')
NaN
> typeof NaN
'number'
> [] + {}
'[object Object]'
> {} + []
0
> [] + {} === {} + []
true
> 
```

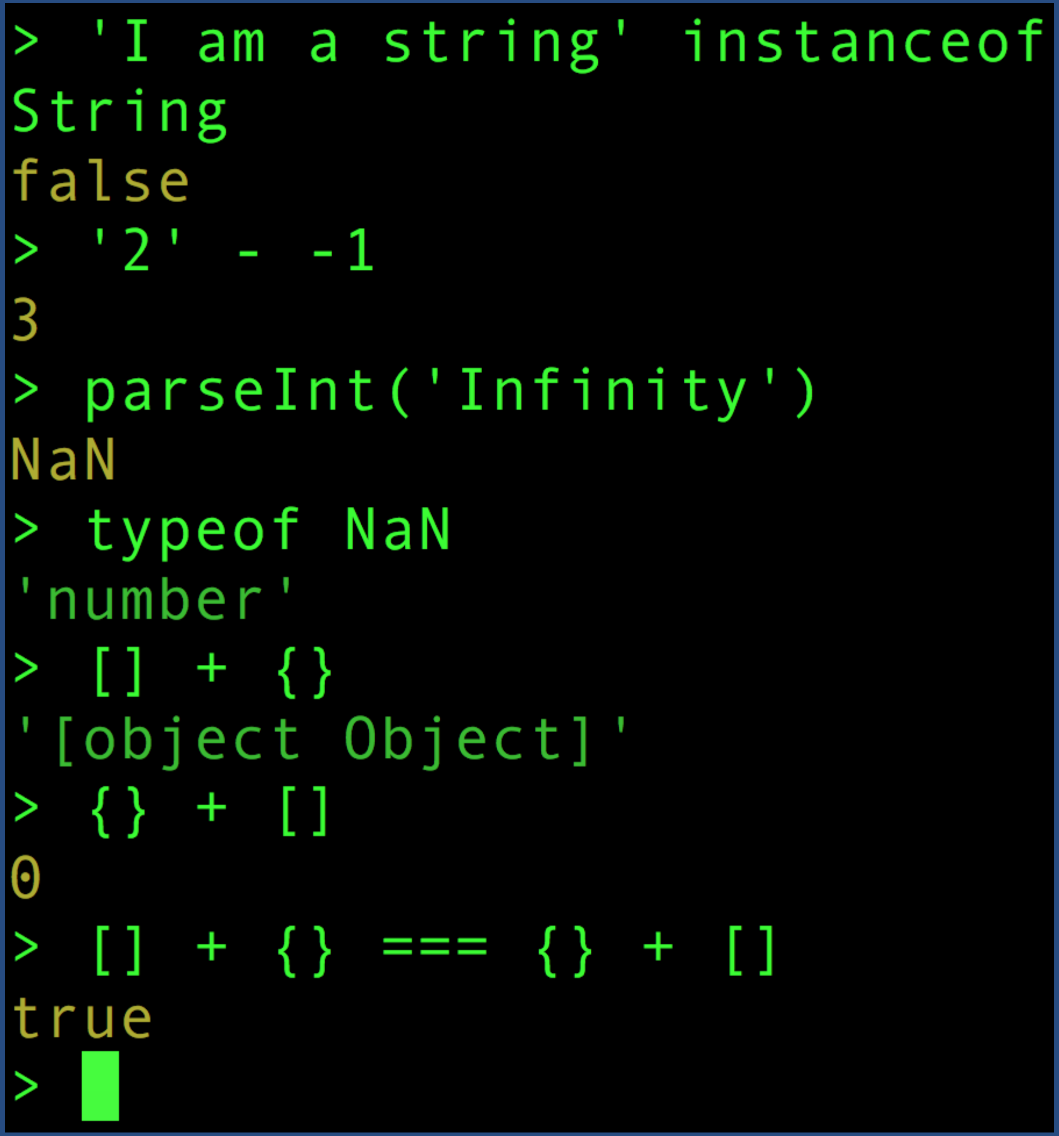
Large-scale JavaScript Application – Developer Skill Types



Speaking One Language



TypeScript “fixing” JavaScript?



```
> 'I am a string' instanceof
String
false
> '2' - -1
3
> parseInt('Infinity')
NaN
> typeof NaN
'number'
> [] + {}
'[object Object]'
> {} + []
0
> [] + {} === {} + []
true
> 
```

- Type often known.
- Compile (Type) Error
- Type Error
- Still exists.
- Compile (Type) Error
- Compile (Type) Error
- Compile (Type) Error

Benefits of Typing

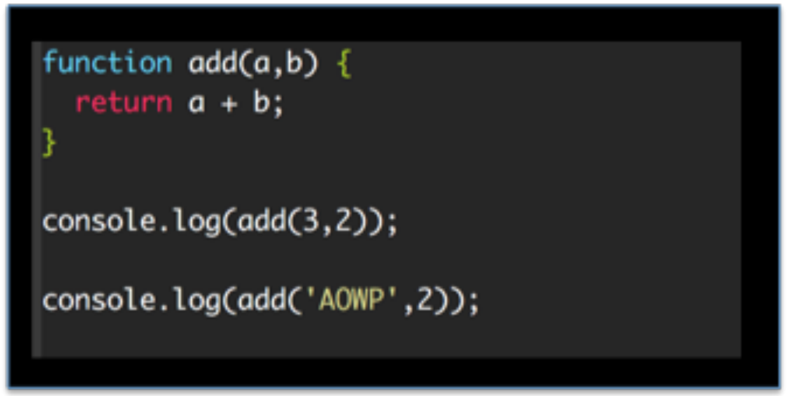
- can make code more readable
- can make code easier to analyze
- can allow for reliable refactoring
- can allow for generally better IDE support
- can catch errors early(before runtime)

TypeScript Features

- Type Annotations
- Public/Private/Protected
- Compile-time type checking
- Type inference
- Tuples
- Read only properties
- Enums
- Mixin
- Generics
- Optional properties
- Interfaces
- ‘Any’ support

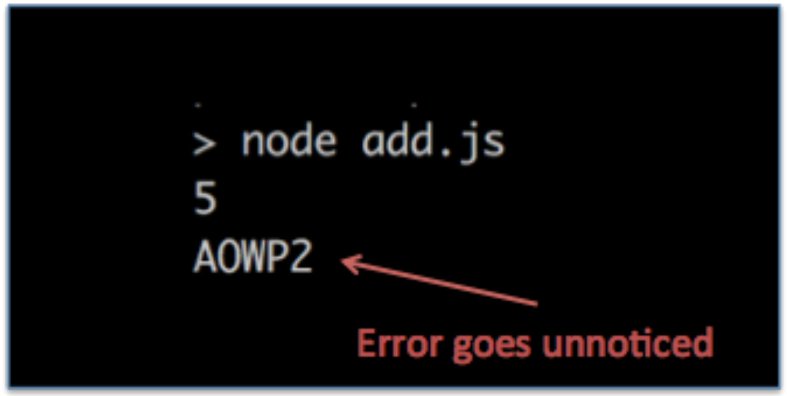
Compile Time Checking

JavaScript Code



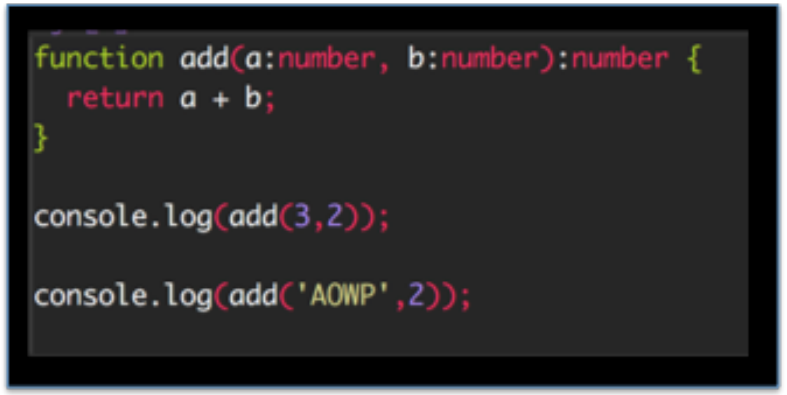
```
function add(a,b) {
  return a + b;
}

console.log(add(3,2));

console.log(add('AOWP',2));
```

JavaScript Output



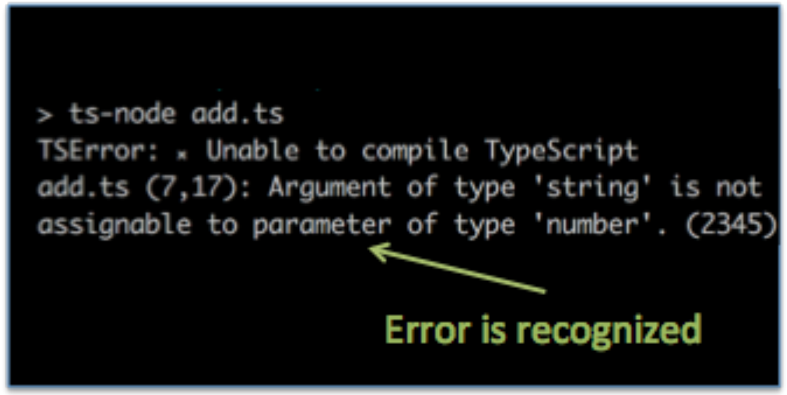
```
> node add.js
5
AOWP2
```

TypeScript Code



```
function add(a:number, b:number):number {
  return a + b;
}

console.log(add(3,2));

console.log(add('AOWP',2));
```

TypeScript Output



```
> ts-node add.ts
TSError: × Unable to compile TypeScript
add.ts (7,17): Argument of type 'string' is not
assignable to parameter of type 'number'. (2345)
```

Details on TypeScript types

Similar to JavaScript

- Boolean:
  - var alive:boolean = true;
- Number:
  - var age:number = 89;
  - var hex:number = 0xf00d;
- String:
  - var name:string = ‘RanchDressing.com’;
- Array:
  - var pets:Array<string> = [‘I’, ‘do’, ‘not’, ‘own’, ‘pets’];
- Object:
  - var car:Car = {name: ‘BMW’, year: ‘2012’, stolen: true}
- Class

Details on TypeScript types

Introducing some new terms

- Tuple:
  - let x: [string, number];
  - x = [“age”, 10]; // Okay
  - x = [10, “age”] // Error
- Enum:
  - enum Color {Red, Green, Blue};
  - var c: Color = Color.Green;
- Void: function hello(): void { console.log(‘...’);}
- Interface:
  - Interface Person {name:string, age:number};
  - var bob:Person = {name: ‘Bob’, age: 75}; // Ok
  - var bob:Person = {name: ‘Bob’, age: ‘75’} // Error
- Any:
  - var myFeelings: any = ‘Happy’;
  - myFeelings = [‘Happy’];
  - myFeelings.pop();
  - myFeelings.push(‘Sad’, ‘Angry’, ‘Depressed’);
  - myFeelings = false; // You now have no feelings

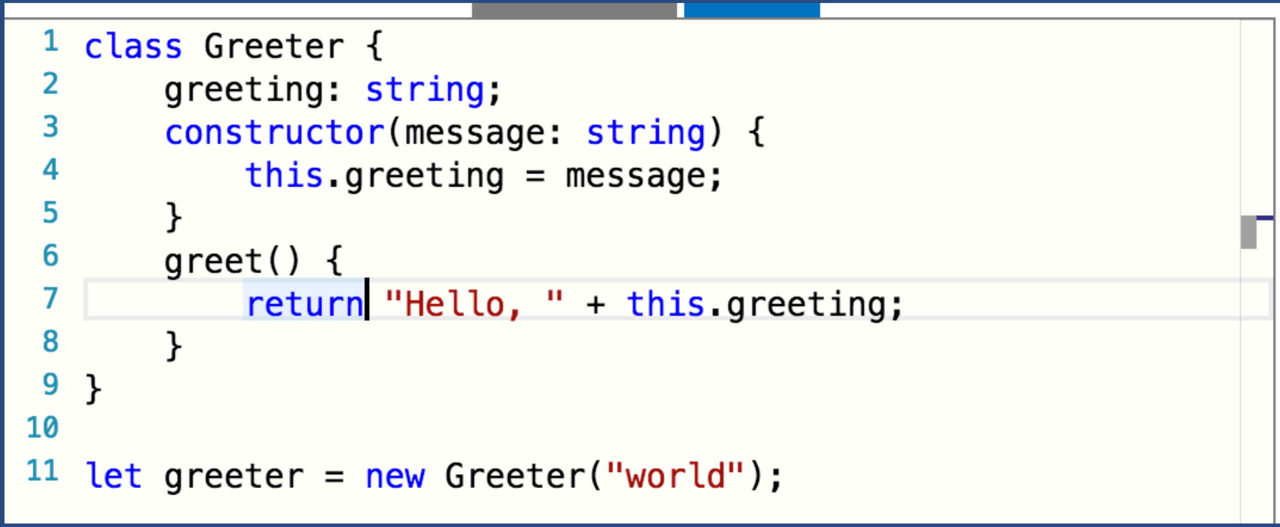
Functions

We get arrows, return types, required & optional parameters and all kinds of goodies.

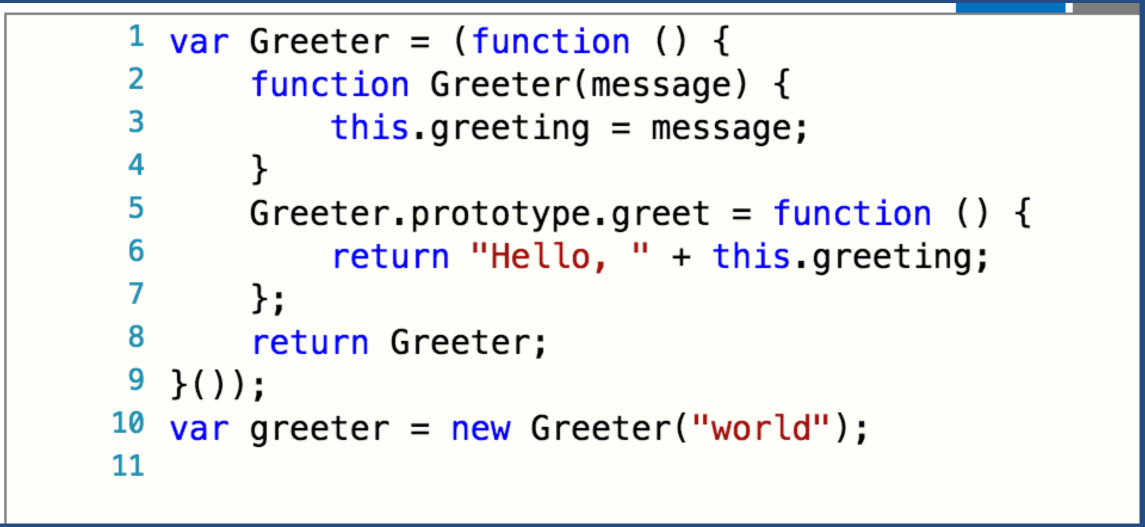
TypeScript	**JavaScript (ES5)**
Types	No Types
Arrow Functions	ES6 Only
Function Types	No Function types
Required & Optional Function Params	All params optional
Default Params	ES6 Only
Rest parameters (...params)	ES6 Only
Overloaded functions	No overloaded functions
Destructuring Assignment	No destructuring

Classes

Part of ES6, usable today with TS



```typescript
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");
```



```javascript
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
var greeter = new Greeter("world");
```

Class Annotations

Part of ES6, usable today with TS

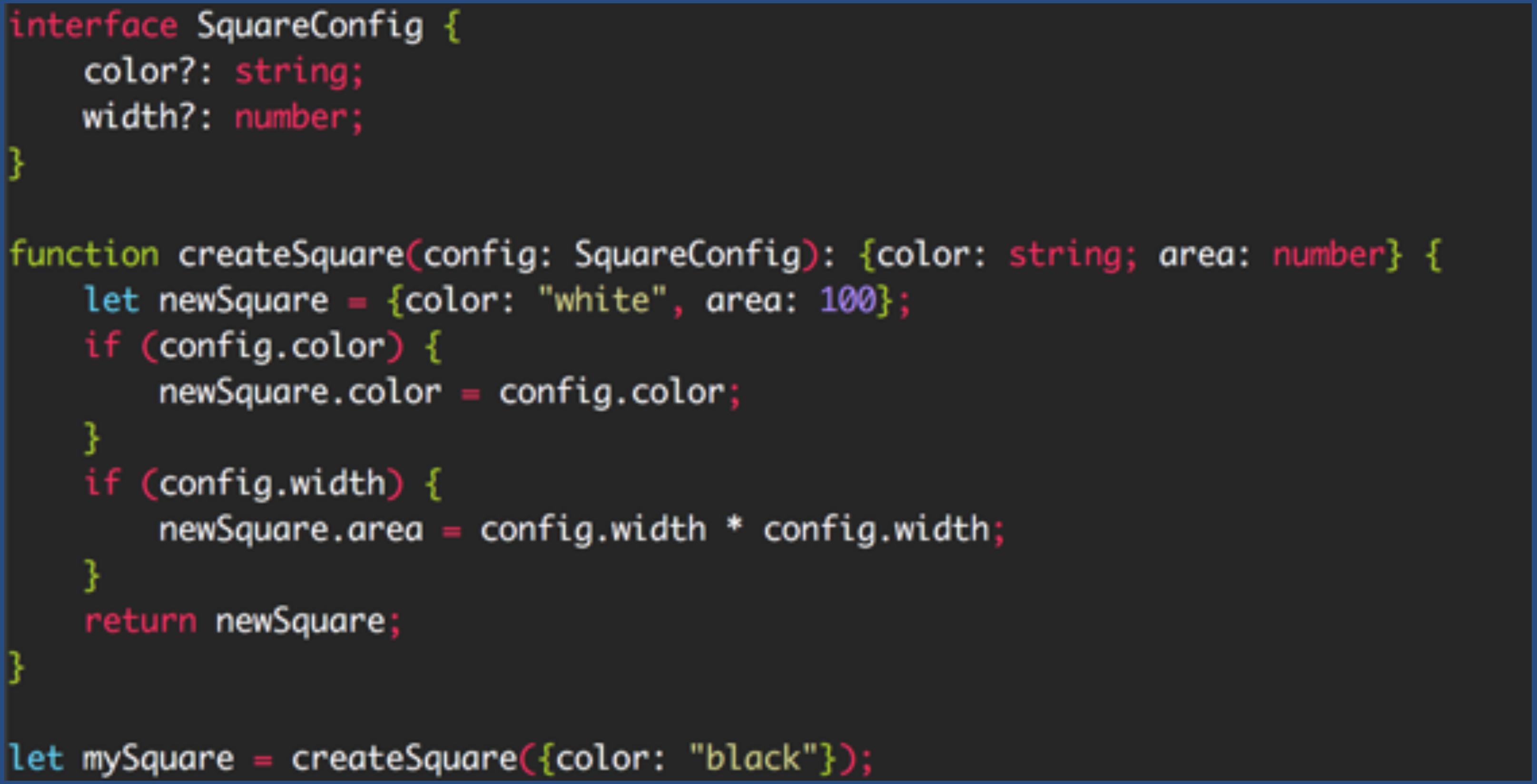
```
function sealed(constructor: Function) {
    Object.seal(constructor);
    Object.seal(constructor.prototype);
}
```

```
@sealed
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
```

A Class Decorator is declared just before a class declaration. The class decorator is applied to the constructor of the class and can be used to observe, modify, or replace a class definition.

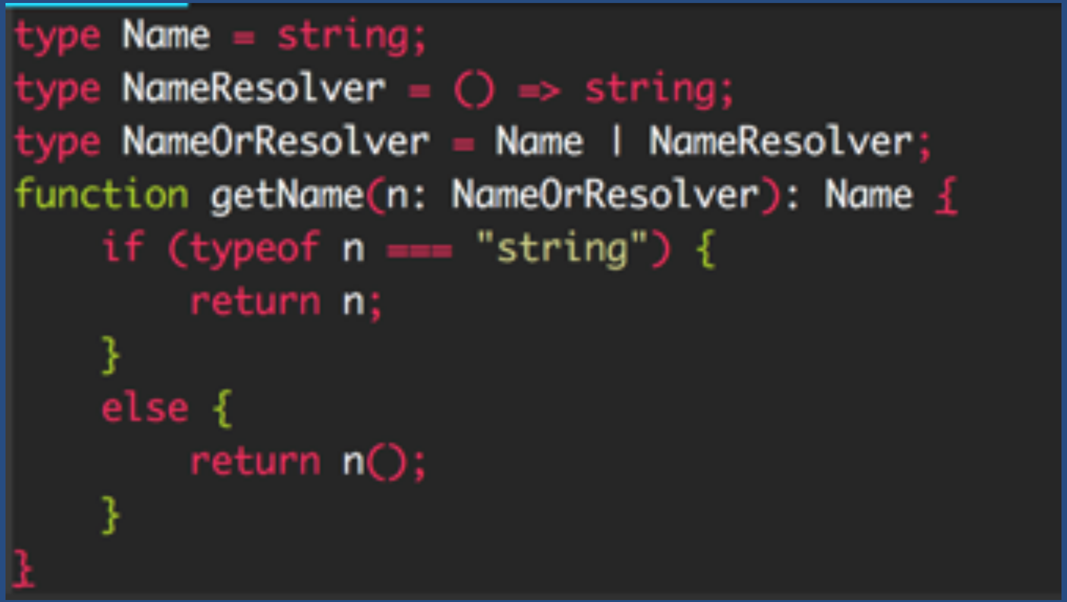
Advanced Typings

Interfaces



```
interface SquareConfig {
    color?: string;
    width?: number;
}

function createSquare(config: SquareConfig): {color: string; area: number} {
    let newSquare = {color: "white", area: 100};
    if (config.color) {
        newSquare.color = config.color;
    }
    if (config.width) {
        newSquare.area = config.width * config.width;
    }
    return newSquare;
}

let mySquare = createSquare({color: "black"});
```

Type Alias



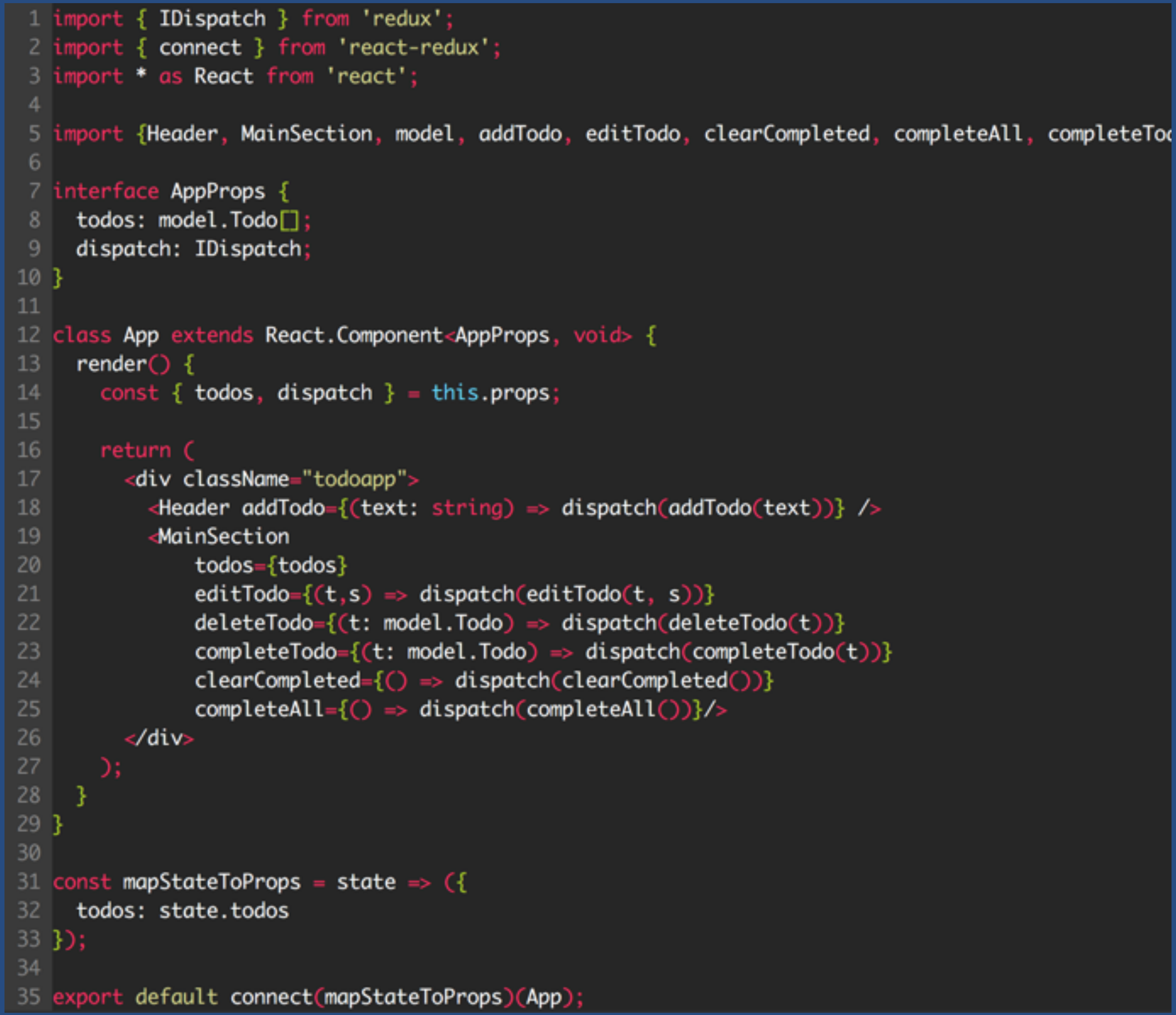
```
type Name = string;
type NameResolver = () => string;
type NameOrResolver = Name | NameResolver;
function getName(n: NameOrResolver): Name {
    if (typeof n === "string") {
        return n;
    }
    else {
        return n();
    }
}
```

TypeScript Examples Across the Stack

TypeScript - FullStack

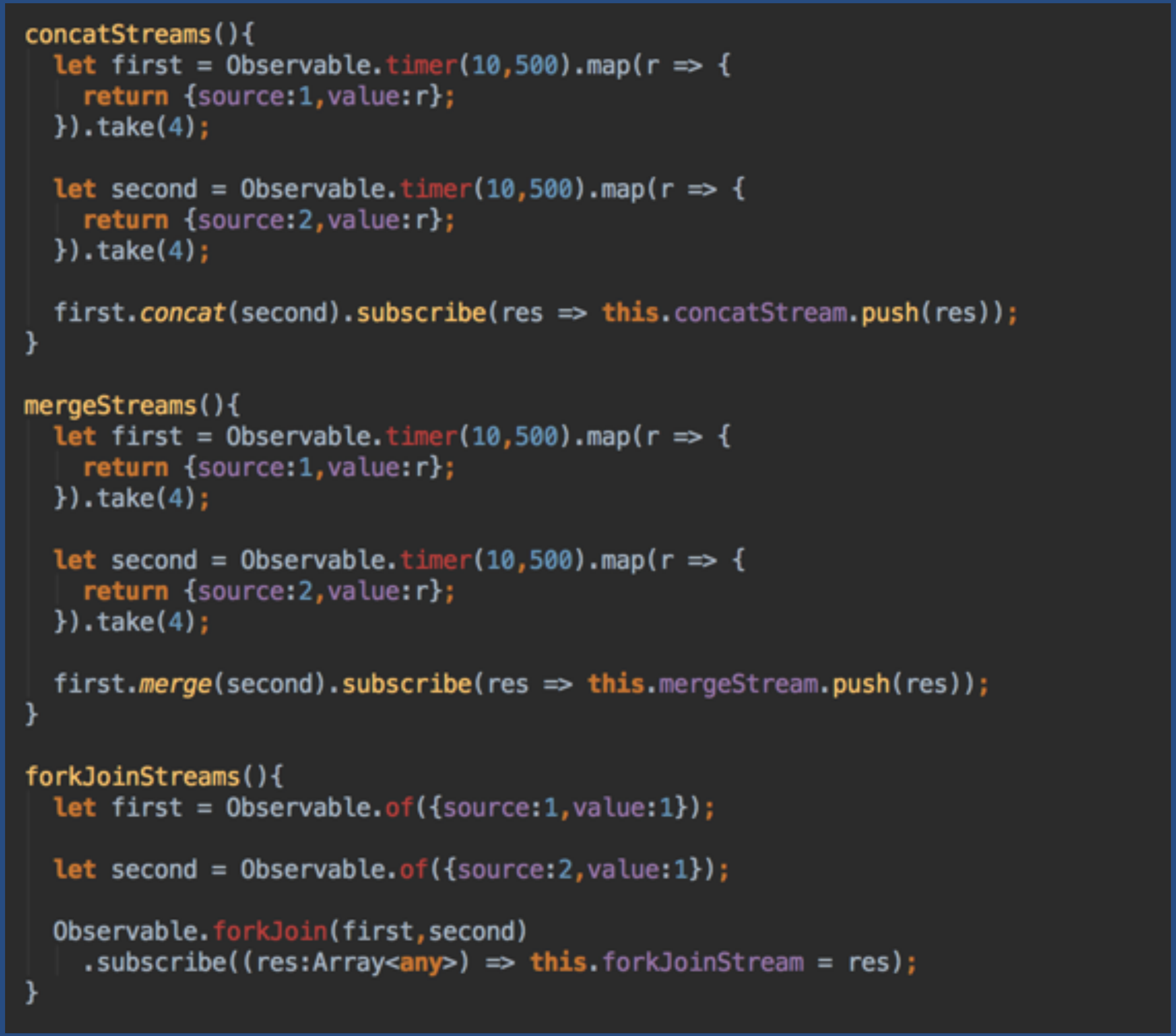
Use all your favorite node frameworks with TypeScript

Redux-React with TS



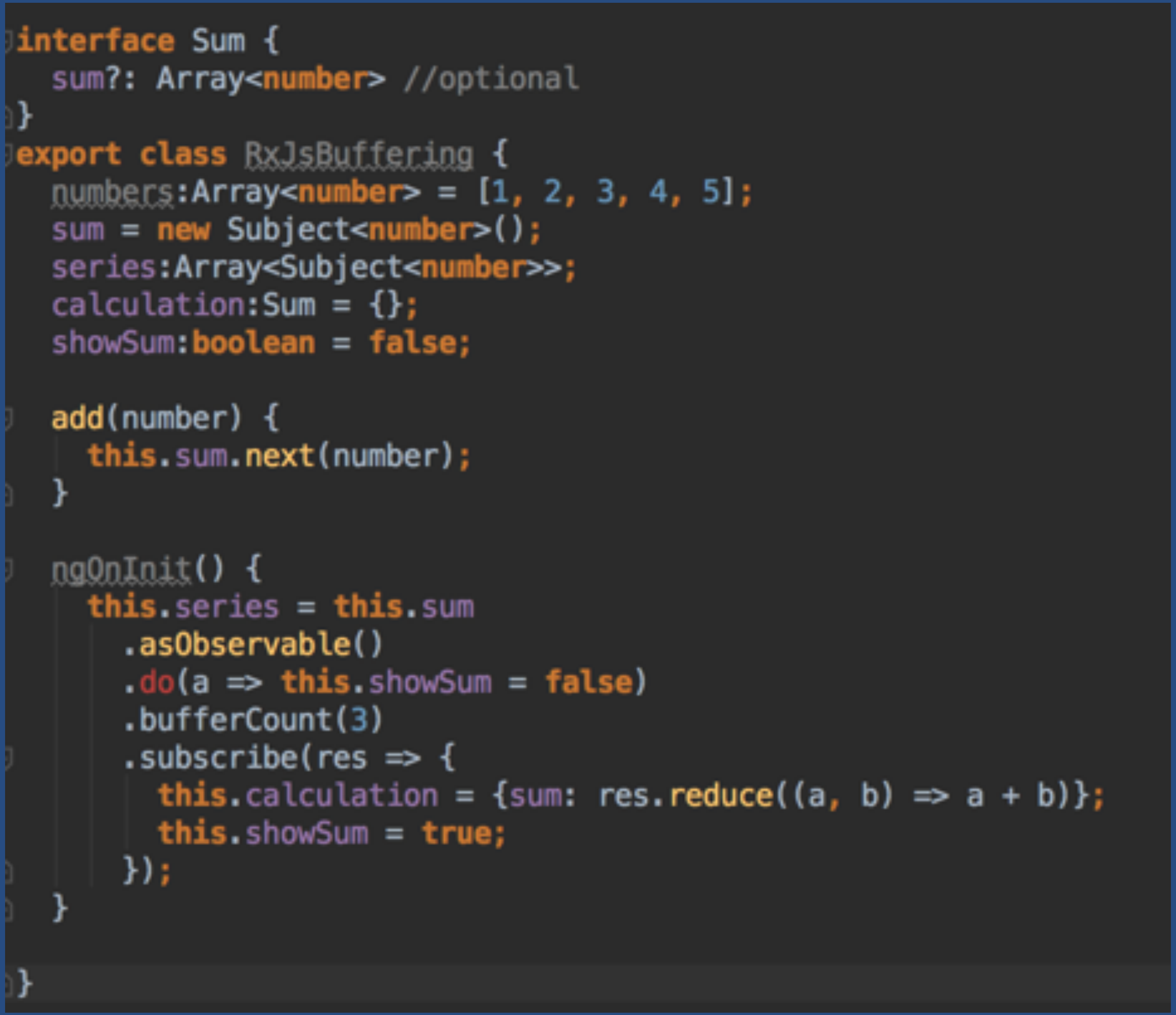
```
import { IDispatch } from 'redux';
import { connect } from 'react-redux';
import * as React from 'react';

import {Header, MainSection, model, addTodo, editTodo, clearCompleted, completeAll, completeToc

interface AppProps {
  todos: model.Todo[];
  dispatch: IDispatch;
}

class App extends React.Component<AppProps, void> {
  render() {
    const { todos, dispatch } = this.props;

    return (
      <div className="todoapp">
        <Header addTodo={(text: string) => dispatch(addTodo(text))} />
        <MainSection
            todos={todos}
            editTodo={(t,s) => dispatch(editTodo(t, s))}
            deleteTodo={(t: model.Todo) => dispatch(deleteTodo(t))}
            completeTodo={(t: model.Todo) => dispatch(completeTodo(t))}
            clearCompleted={() => dispatch(clearCompleted())}
            completeAll={() => dispatch(completeAll())}/>
      </div>
    );
  }
}

const mapStateToProps = state => ({
  todos: state.todos
});

export default connect(mapStateToProps)(App);
```

Angular 2: RxJS Streams



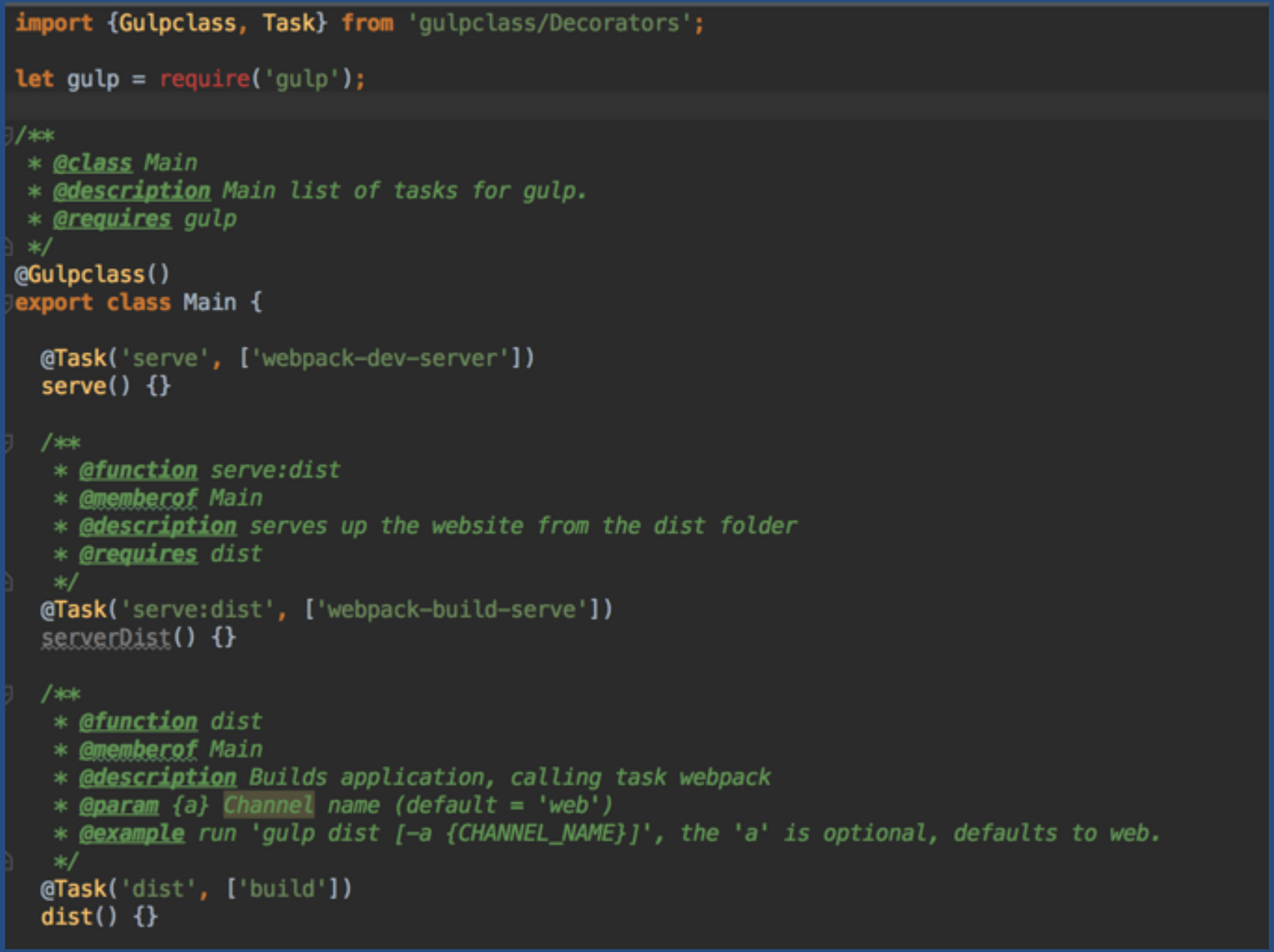
```
concatStreams(){
  let first = Observable.timer(10,500).map(r => {
    return {source:1,value:r};
  }).take(4);

  let second = Observable.timer(10,500).map(r => {
    return {source:2,value:r};
  }).take(4);

  first.concat(second).subscribe(res => this.concatStream.push(res));
}

mergeStreams(){
  let first = Observable.timer(10,500).map(r => {
    return {source:1,value:r};
  }).take(4);

  let second = Observable.timer(10,500).map(r => {
    return {source:2,value:r};
  }).take(4);

  first.merge(second).subscribe(res => this.mergeStream.push(res));
}

forkJoinStreams(){
  let first = Observable.of({source:1,value:1});

  let second = Observable.of({source:2,value:1});

  Observable.forkJoin(first,second)
    .subscribe((res:Array<any>) => this.forkJoinStream = res);
}
```

Angular 2: RxJS Buffering



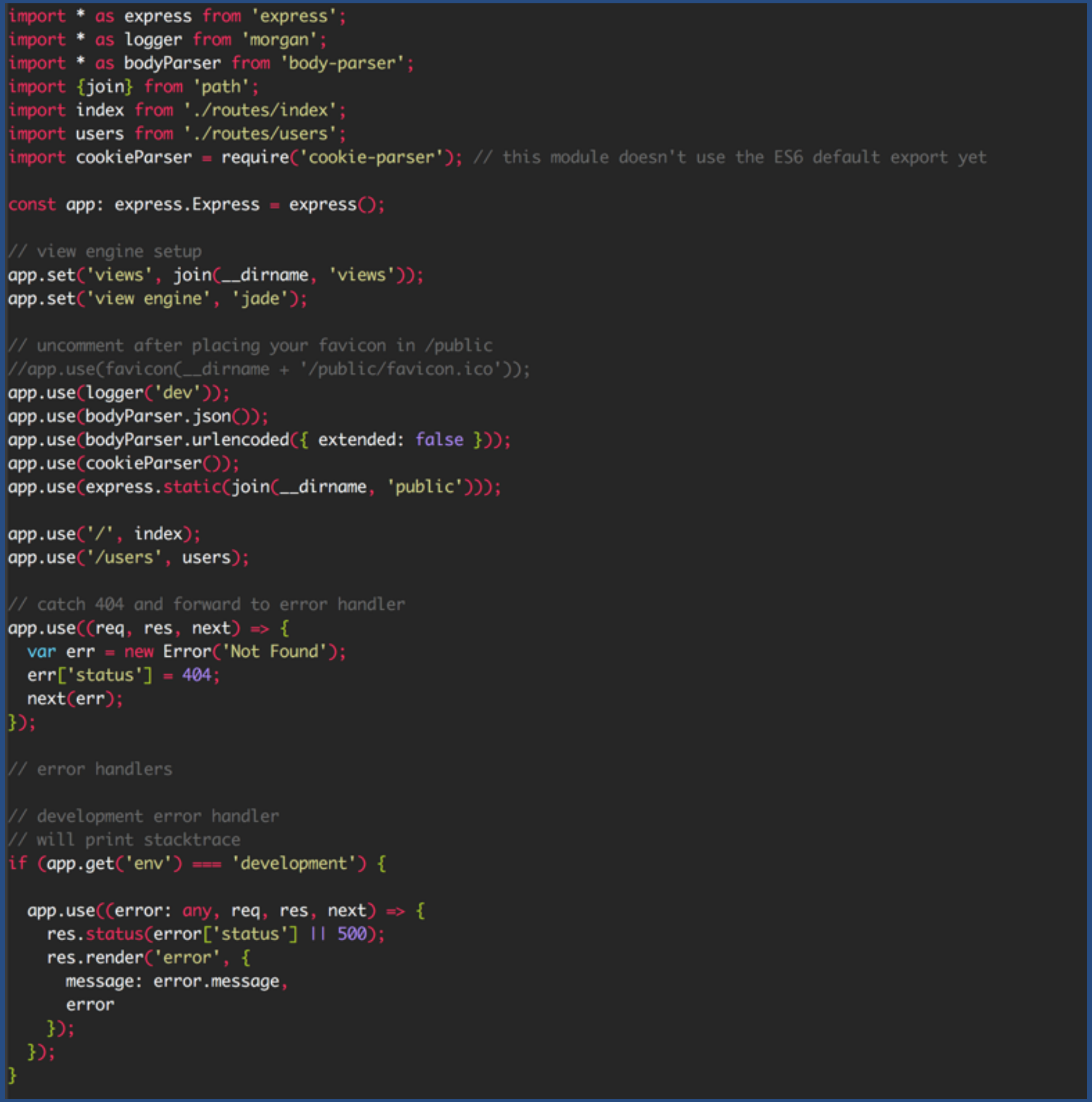
```
interface Sum {
  sum?: Array<number> //optional
}
export class RxJsBuffering {
  numbers:Array<number> = [1, 2, 3, 4, 5];
  sum = new Subject<number>();
  series:Array<Subject<number>>;
  calculation:Sum = {};
  showSum:boolean = false;

  add(number) {
    this.sum.next(number);
  }

  ngOnInit() {
    this.series = this.sum
      .asObservable()
      .do(a => this.showSum = false)
      .bufferCount(3)
      .subscribe(res => {
        this.calculation = {sum: res.reduce((a, b) => a + b)};
        this.showSum = true;
      });
  }

}
```

Gulp with TS



```
import {Gulpclass, Task} from 'gulpclass/Decorators';

let gulp = require('gulp');

/**
 * @class Main
 * @description Main list of tasks for gulp.
 * @requires gulp
 */
@Gulpclass()
export class Main {

  @Task('serve', ['webpack-dev-server'])
  serve() {}

  /**
   * @function serve:dist
   * @memberof Main
   * @description serves up the website from the dist folder
   * @requires dist
   */
  @Task('serve:dist', ['webpack-build-serve'])
  serverDist() {}

  /**
   * @function dist
   * @memberof Main
   * @description Builds application, calling task webpack
   * @param {a} Channel name (default = 'web')
   * @example run 'gulp dist [-a {CHANNEL_NAME}]', the 'a' is optional, defaults to web.
   */
  @Task('dist', ['build'])
  dist() {}
```

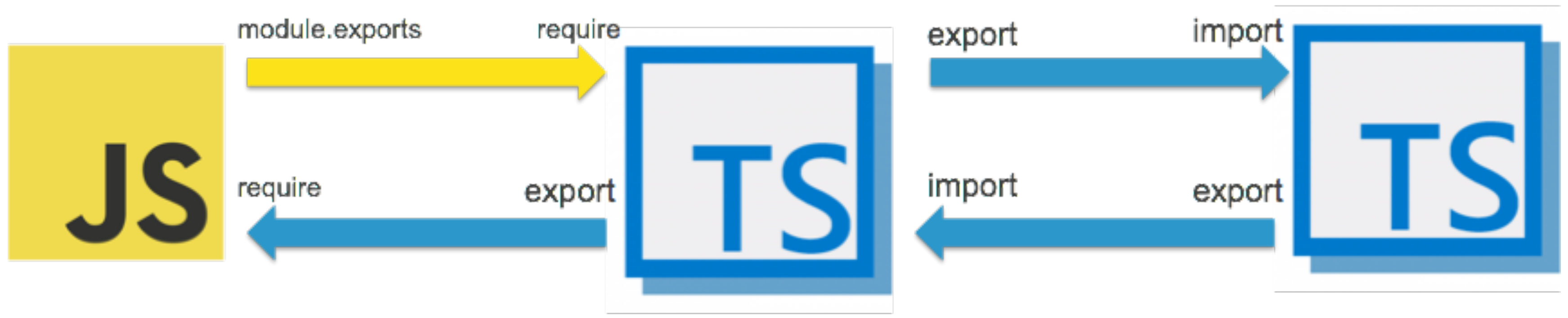
Node-Express



```
import * as express from 'express';
import * as logger from 'morgan';
import * as bodyParser from 'body-parser';
import {join} from 'path';
import index from './routes/index';
import users from './routes/users';
import cookieParser = require('cookie-parser'); // this module doesn't use the ES6 default export yet

const app: express.Express = express();

// view engine setup
app.set('views', join(__dirname, 'views'));
app.set('view engine', 'jade');

// uncomment after placing your favicon in /public
//app.use(favicon(__dirname + '/public/favicon.ico'));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(join(__dirname, 'public')));

app.use('/', index);
app.use('/users', users);

// catch 404 and forward to error handler
app.use((req, res, next) => {
  var err = new Error('Not Found');
  err['status'] = 404;
  next(err);
});

// error handlers

// development error handler
// will print stacktrace
if (app.get('env') === 'development') {

  app.use((error: any, req, res, next) => {
    res.status(error['status'] || 500);
    res.render('error', {
      message: error.message,
      error
    });
  });
}
```

Using JavaScript with TS

- TypeScript can directly required JavaScript and vice versa.
- This enables a seamless transition
- Typings for external libraries are available



Typings

- TypeScript doesn't know the types of the parameters and signatures of 3rd party libraries
- To get the full value of TS, you should install the Typings Definition file for it
  - Originally, this was called "tsd"
  - Then a new player emerged, called simply "typings"
  - As of TS 2.0, you use *neither* of these. You simply install with NPM
    - npm install --save @types/lodash

  - When looking at StackOverflow or elsewhere, ignore tsd/typings if using TypeScript 2.0
- You can make your own typings file
  - These are just interfaces!
  - Typings files are *excellent* sources of documentation

Node for TypeScript

- Common Options for running node with TypeScript:
- TSC watch: Out of the box watch ability with TypeScript. Generate JS code when TS files are modified.
  - Execute node foo.js as normal
- TS-Node: TypeScript execution environment and REPL for node.
- NPM as a build pipeline
  - Watch for changes, compile, reload server
  - More setup required here, but there are bootstrappers available

DECIDING BETWEEN ES6 AND TYPESCRIPT

ES6

- Prefer Dynamic typing *and know why*
- Small applications
- Small teams
- Advanced Developers

TypeScript

- When the Target system interprets JavaScript :)
- Large Project
- Prefer compile time checking
- Developers background is in Java or C#
- Need developers from various backgrounds to align to one standard.
- When using Angular 2

Use JavaScript (ES5) over ES6/TS?

At this stage we recommend all projects use ES6 + Transcompiler or TypeScript.

TypeScript vs Flow

TypeScript is not the only choice

TypeScript

- A compiler
- Gives you ES6 + some 7/8 today
- Better Tooling Support
- OK at inferring types
- Has Non-nullable types
- Works with React (with TSX)
- Works with Angular 2

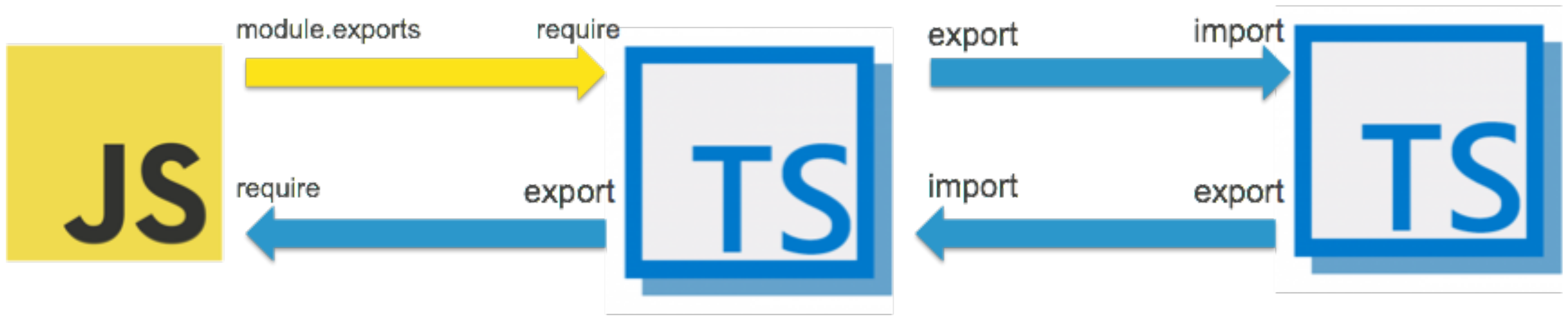
Flow

- A checker
- Uses ES6
- Tool support *getting better*
- Great at inferring types
- Has Non-nullable types
- Works with React (with TSX)
- Not Angular 2 optimized

These two projects have influenced each other greatly. If you learn one, it is not much harder to learn the other one.

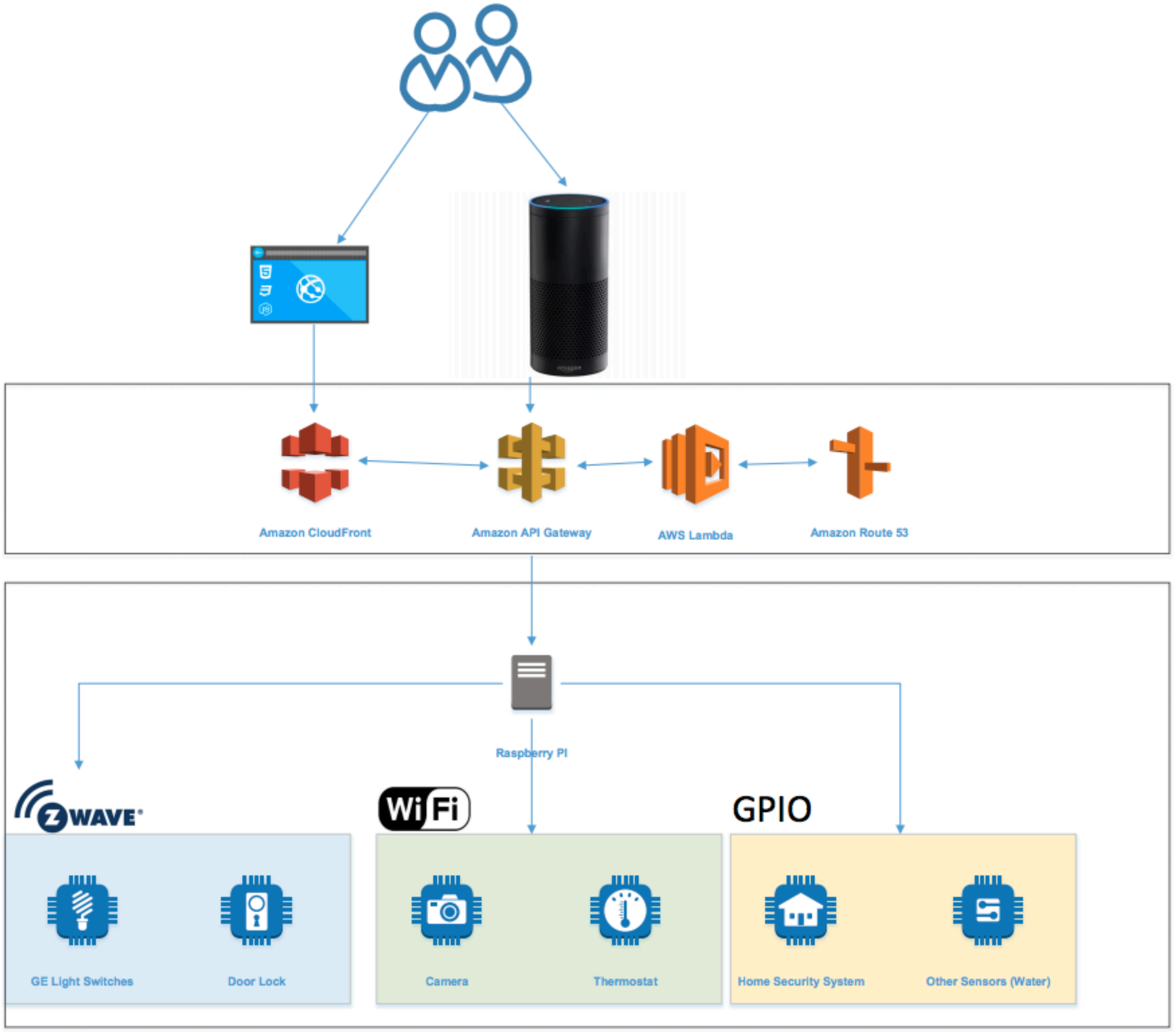
MIGRATING FROM ES3/5

- TypeScript can directly required JavaScript and vice versa.
- This enables a seamless transition.
- To “Type” a JavaScript file, create a .d.ts file with the same name.
  - i.e. app.js can have typings defined with app.js.d.ts



Hands on Overview

Diagram



Use Case

Homeowner and Guest can control lighting, temperature, and monitor home status via an app. For convenience, users can also ask Alexa to do the same tasks.

AWS Architecture

App runs on CloudFront/S3, and sends commands to API gateway, which proxies to the Raspberry pi. The pi auto-updates its public IP address via a custom lambda function.

Home Architecture

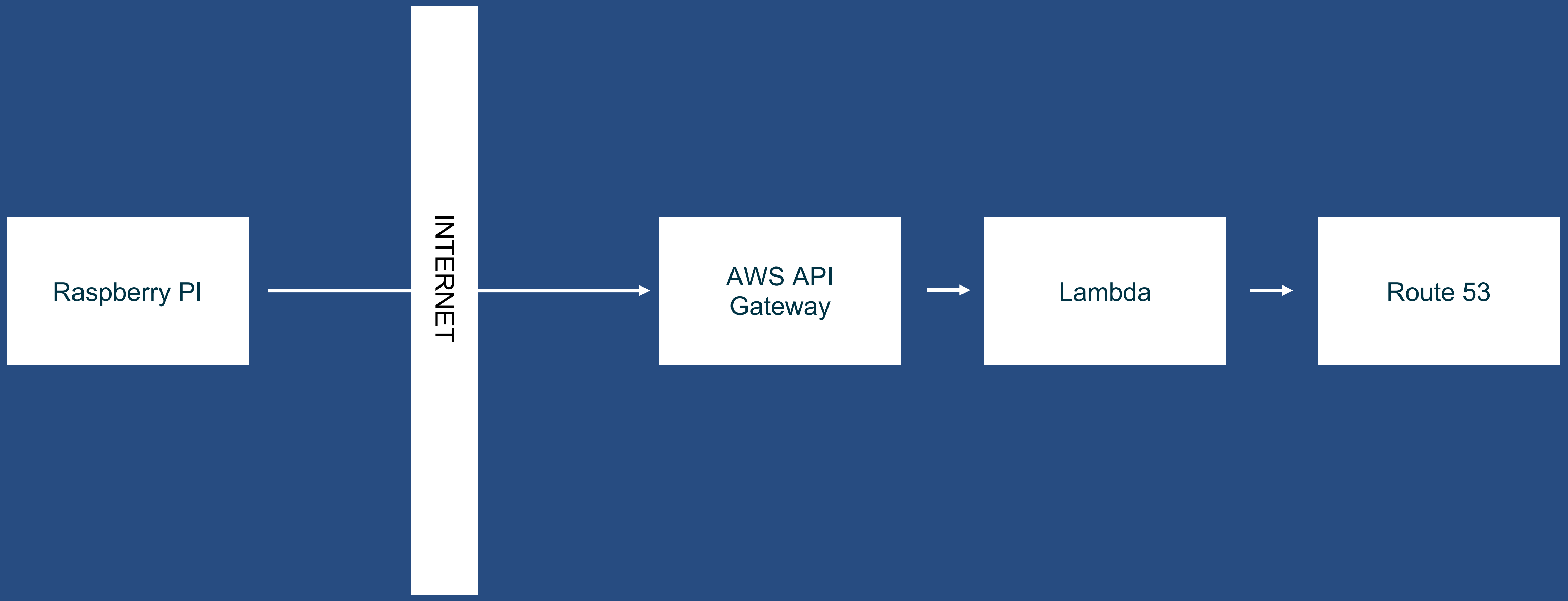
Light Switches are GE, and operate on the ZWAVE protocol. Raspberry pi has a zWave USB stick. Additionally, the pi connects to WiFi devices and GPIO devices via hardwire.

Hands on Part 1

Dynamic DNS Lambda

DynamicDNS Clone

My raspberry PI is behind normal internet, which changes its public IP from time to time. I am too cheap to pay for DynamicDNS



Lambda

- Lambda is commonly referred to as a "serverless" platform for event handlers
  - Its not serverless. There is a server... you just don't have to worry about it.
    - In practice, it is a container, with specific limitations
- Lambda just runs functions. It does not store data
  - You can run node.js, Python, or java *it does not run TypeScript!*
  - It's great for short lived requests, not great as a web server
- Lambda is Cheap, but not free
  - About 20 cents per million requests, and you are charged for compute time
  - Can be challenging to model, so you should use it for short-run requests
  - Details: https://www.trek10.com/blog/lambda-cost/
- Common use cases
  - Do "something" when "something else" happens – something else can be DB update,S3 upload, SQS trigger
  - When I upload image, create thumbnail – make this a microservice using lambda

Demo

- We will build the Lambda function locally using TypeScript
- Test it locally using node-lambda
- Build it with TSC
- Package it up for deployment & deploy it
- Setup the API to be able to hit the lambda from the web (optional step)

Hands on Part 2

Home Automation API in Node

Demo

- We will build an API using express and TypeScript
- Setup a reusable Node.JS build pipeline
- Show how we can Unit Test our API

Hands on Part 3

Home Automation UI in Angular 2

Demo

- We will build an Angular 2 application using angular CLI
- We will wire it up to our API
- We will showcase a reusable UI layer build pipeline
- *We will NOT teach you how to use Angular 2 in depth*

End to End

- I will show you the working demo, end to end, and answer any questions about the demo

TypeScript Review

What we did Today

- Discussed TypeScript, what it is, why you should use it, and how to implement it
- Created an AWS Lambda function in TS, Deployed it, and ran it in JS
- Created an API using TS and Node
- Created a UI application using Angular 2 and TS

Online Resources to Build TypeScript Skills

- EdX – Good Course but it is already a bit out of date
  - https://courses.edx.org/courses/course-v1:Microsoft+DEV201x+1T2016/info
- TypeScript Website
  - https://www.typescriptlang.org/docs/tutorial.html <- good documentation here
  - https://www.typescriptlang.org/play/index.html <- TypeScript Playground
- Angular 2 Website
  - Angular 2 is optimized for TypeScript, lots of tutorials will be in TS
  - https://angular.io/docs/ts/latest/quickstart.html <- Easy/quick intro
  - https://angular.io/docs/ts/latest/tutorial/ <- More in-depth tutorial
- Pluralsight
  - https://www.pluralsight.com/courses/typescript-practical-start <-Quick Intro
  - ¨https://www.pluralsight.com/courses/typescript-in-depth <- Good Course
- Books
  - No good TS books out yet that I can suggest... The language is evolving too fast
    - Online book https://basarat.gitbooks.io/typescript/content/docs/template-strings.html
  - I will recommend the excellent *You don't Know JS ES6 & Beyond*
    - https://github.com/getify/You-Dont-Know-JS/tree/master/es6%20%26%20beyond
- Join the TypeScript Yammer group within Accenture

Thank You

Thank you for your attention and participation. If you have further questions, or if we may be of assistance, please do not hesitate to reach out!

CHAD MOTT

Chad.Mott@accenture.com

DANIEL DEREVJANIK

Daniel.Derevjanik@accenture.com