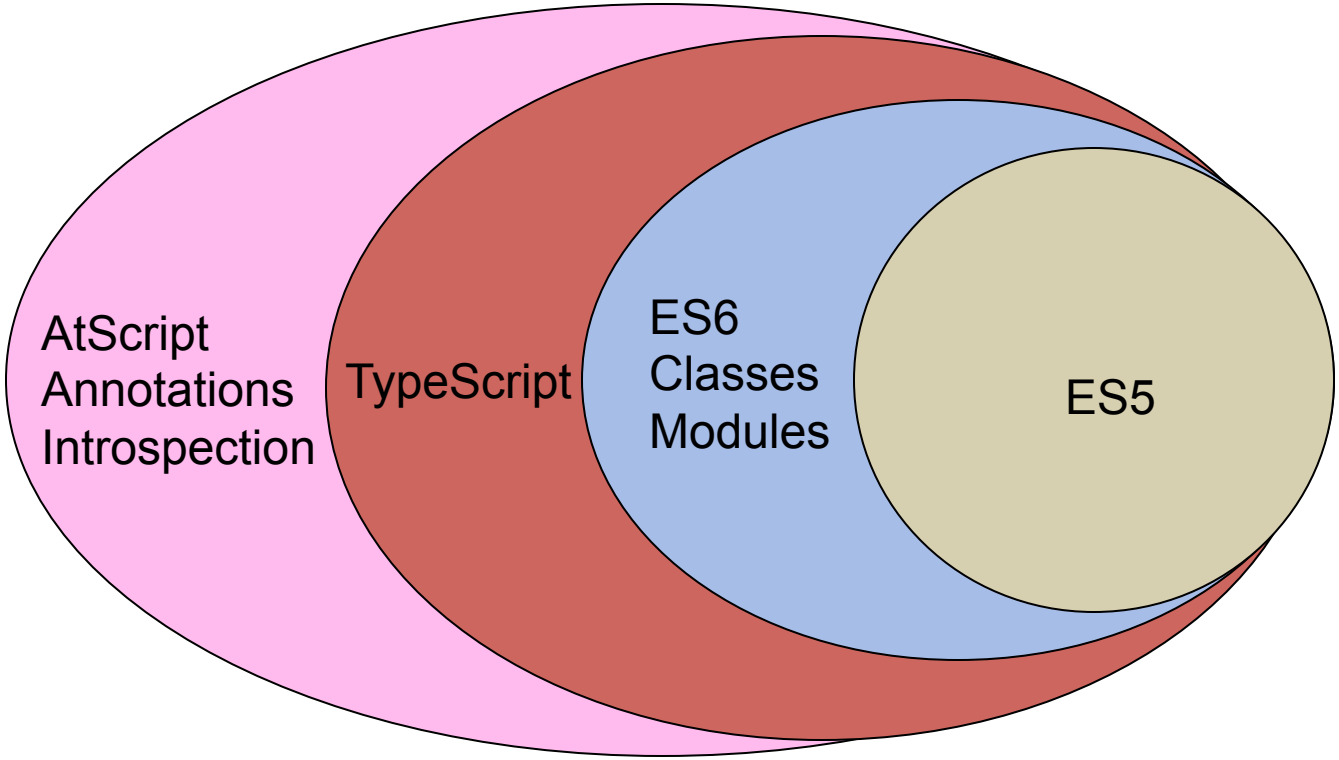
Introduce TypeScript

Ionic1++ != Ionic2

John Pearson

2015 Microsoft and Google Standardize on Typescript



TypeScript

Types plus much more

TypeScript transpiles to JavaScript

tsc filename.ts --target ES5

ES6 + ES7 target ES5 supports current browsers

TypeScript ~= ES6+

MSOpenTech => Microsoft Open Technology Programs Office

Relationship with Google, support for Angular 2

TypeScript and Developers

- Why do I care? Show todoitem.ts example()
- Intellisense & find errors while I am typing.
- Why you should care?
- Another skill to differentiate you in the market.
- (Ionic2, Angular2) => { $ }

TypeScript pros

- Clea[n|r]er grammar
- Getting type intent, correct definition, not any
- Express designs better, support larger teams
- Common vocabulary for humans and tools
- Editors like Visual Studio *, Sublime …
- Testing tools like Karma, e2e ...

TypeScript cons

- Time
- You must use a transpiler
  - Command line
  - Grunt
  - Gulp
  - Webpack

Give Credit

- Anders Hejlsburg: father of TypeScript

https://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript

- Josh Morony: Ionic2 JavaScript Todo app

http://www.joshmorony.com/build-a-todo-app-from-scratch-with-ionic-2-video-tutorial/

Support stable environment[s]+

- Version of node, STOP!
- Node Version Manager (NVM)
- nvm install v0.12.7
- nvm list
- nvm use v0.12.7
- nvm version
- npm install –g typescript // Note: typescript@next, 1.8 beta

https://blogs.msdn.microsoft.com/typescript/2016/01/28/announcing-typescript-1-8-beta/

TypeScript Basic Types

- var isDone: boolean = false;
- var height: number = 6;
- var name: string = “bob”; //Note: ‘bob’
- var list: number[] = [1, 2, 3];

TypeScript Basic Types

- var list: Array<number> = [1, 2, 3];
- enum Color { Red = 1, Green, Blue };
- var list: any[] = [1, true, “free”];
- function warnUser() : void { …}

Error Types

- Error
- EvalError
- InternalError
- RangeError
- ReferenceError

- SyntaxError
- TypeError
- URIError
- Error type signature

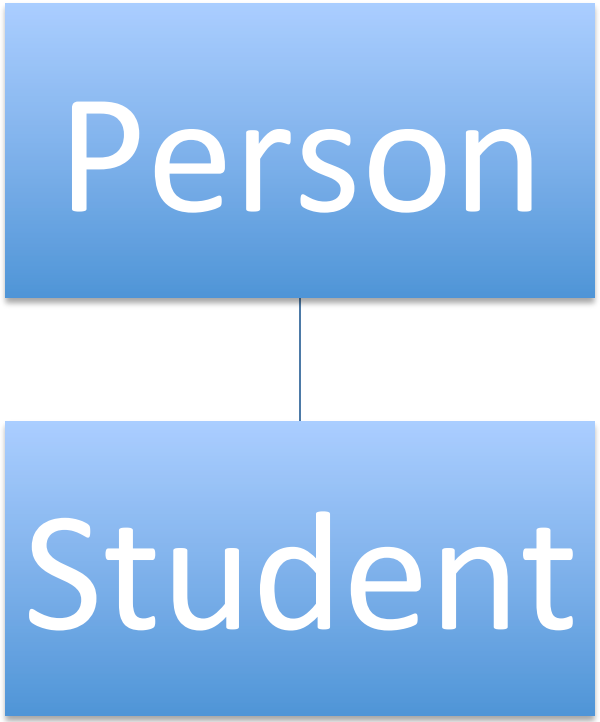
throw new Error(

[message[, filename[, lineNumber]]])

TypeScript Basic Features

- Interfaces : allow loose to tighter cohesion between objects, you are in the drivers seat
- Classes : allow common Object Oriented Style
- Generics : allow lightweight, reusable containers without runtime overhead
- Modules : allow human organization of code reuse while avoiding collisions

~ Object Oriented JavaScript



Classes are nice

- Creating new instances, multiple times
- Inheritance
- Needing singletons
- Discrete properties and methods
- You need to be aware of “this” keyword

“this” Variable and Scope

- Why do I need “this” keyword
  - new creates an instance variable
  - Execution Context, 2 objects
  - Passing control, object1 → object2
  - Context, context, context

Function parameters

Optional parameter

constructor(firstname : string, lastname : string, anyStudentData? : any[])

Default parameter

constructor(firstname : string, lastname : string , anyStudentData : any[] = [“default”])

Rest parameters

constructor(firstname : string, lastname : string, …anyStudentData : any[])

Function Overloads

```
var suits = ["hearts", "spades", "clubs", "diamonds"];

function pickCard(x: {suit: string; card: number; }[]): number;
function pickCard(x: number): {suit: string; card: number; };
function pickCard(x): any {
    // Check to see if we're working with an object/array
    // if so, they gave us the deck and we'll pick the card
    if (typeof x == "object") {
        var pickedCard = Math.floor(Math.random() * x.length);
        return pickedCard;
    }
    // Otherwise just let them pick the card
    else if (typeof x == "number") {
        var pickedSuit = Math.floor(x / 13);
        return { suit: suits[pickedSuit], card: x % 13 };
    }
}
```

```
var myDeck = [{ suit: "diamonds", card: 2 },
{ suit: "spades", card: 10 }, { suit: "hearts",
card: 4 }];
var pickedCard1 = myDeck[pickCard(myDeck)];
alert("card: " + pickedCard1.card + " of " +
pickedCard1.suit);

var pickedCard2 = pickCard(15);
alert("card: " + pickedCard2.card + " of " +
pickedCard2.suit);
```

Inference / duck typing

Student

- firstname
- lastname
- mySearch

Person

- firstname
- lastname
- mySearch

What is Code Refactoring

- Code refactoring is the process of restructuring existing computer code – changing the factoring – without changing its external behavior
- Advantages include improved code readability and reduced complexity
- Improve extensibility

Keyword Implements is structural

Student implements

- firstname //I have to
- lastname //I have to
- mySearch //I have to

Person

- firstname
- lastname
- mySearch

Keyword Extends

Student extends Person

- extend firstname
- extend lastname
- Your choice mySearch, super

- firstname
- lastname
- mySearch

JavaScript Debugging

- Module Pattern / Anonymous Closures
  => implied global import, var module =
- var Person = (function () {return Person; })();
- Chrome.View.Developer & source map
- Setting Breakpoints and logging
- document.location.reload()

Modules

- Decouple
- Reusable
- Synchronous verses Asynchronous
- Identifies dependencies
- Asynchronous Module Definition => Browser
- CommonJS => NodeJS

Modules

AMD

- Browser
- Asynchronous
- Zero File I/O
- Objects, Functions, Constructor, Strings, JSON and other types of modules

CommonJS

- Server
- Synchronous
- I/O, File System, Promises and more
- Supports unwrapped
- Only supports objects as modules

Refactor to AMD

CDN or local copy of require.js

<script src=”require.js" data-main=”LogicForLaunch"></script>

tsc --sourcemap --module amd app.ts

tsc; tsconfig.json => { "compilerOptions": {

"module": "amd”, "sourceMap": true },

"files": ["app.ts”] }

Lambda expression ()=>{}

- Show list.ts additem() pop
- Review the returnToList(item:Ijson)

Lambda expression ()=>{}

JavaScript

```
Greeter.prototype.start =
 function () {
    var _this = this;
    this.timerToken =
setInterval(function () {
return _this.span.innerHTML = new
Date().toUTCString(); }, 500);
       };
```

TypeScript

```
start()
    {
 // context is taken care of
this.timerToken =
 setInterval(
() => this.span.innerHTML =
new Date().toUTCString(),500);
}
```

Lambda expression

- “this” is evaluated in the context of the method definition
- “this” is not evaluated in the context of the method execution

Generic Types and Constraints

```
import Person = require("person")

greeter(person : Person.Person) : string {
    return "Hello, " + person.firstname + " " +
person.lastname + " any " +
((person.someData == null) ? "no student
data" : person.someData[0]) ;
    }
```

```
NOTE: remove Person
interface tripleCheck {
        firstname: string;
        lastname: string;
        someData: any[];
    }
greeter<T extends tripleCheck>(arg: T): string {
    return "Hello, " + arg.firstname + " " +
arg.lastname + " any " + ((arg.someData ==
null) ? "no student data" : arg.someData[0]) ;;
    }
```

Mixins Implement

Firstname

- firstname

Class Lastname

- Lastname

Class MySearch

- mySearch

Mixin example

Create stand-in properties

```
import FirstName = require("FirstName");
import LastName = require("LastName");
import MySearch = require("MySearch");
import SomeData = require("SomeData");

export class MakeMeAPerson implements FirstName.FirstName
, LastName.LastName
, MySearch.MySearch
, SomeData.SomeData {

    constructor(myfirstname: string, mylastname: string, mysomedata:
any[]){
        this.firstname = myfirstname;
        this.lastname = mylastname;
        this.someData = mysomedata;
    }
```

Mixin signature then apply

```
    firstname;
    lastname;
    mySearch: () => boolean;
    someData;
}
applyMixins(MakeMeAPerson, [FirstName.FirstName,
LastName.LastName, MySearch.MySearch
, SomeData.SomeData] )
function applyMixins(derivedCtor: any, baseCtors: any[]) {
    baseCtors.forEach(baseCtor => {
Object.getOwnPropertyNames(baseCtor.prototype).forEach(
name => {
     derivedCtor.prototype[name] = baseCtor.prototype[name];
        })
    });
}
```

Mixin

```
import Person = require("makemeaperson");
…
var mixedUpPerson = new Person.MakeMeAPerson("Barbara", "Pearson", ["mydata"]);

this.greetPerson.innerHTML = this.greeter(mixedUpPerson);

…
```

Decorators

- Decorators are new to TypeScript, but TypeScript already handles very similar generation.

```
class student extends person
var __extends = this.__extends || function (d, b) {
    for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
    function __() { this.constructor = d; }
    __.prototype = b.prototype;
    d.prototype = new __();
};
```

Decorators

1. declare type ClassDecorator = <TFunction extends Function>(target: TFunction) => TFunction | void;
2. declare type PropertyDecorator = (target: Object, propertyKey: string | symbol) => void;
3. declare type MethodDecorator = <T>(target: Object, propertyKey: string | symbol, descriptor: TypedPropertyDescriptor<T>) => TypedPropertyDescriptor<T> | void;
4. declare type ParameterDecorator = (target: Object, propertyKey: string | symbol, parameterIndex: number) => void;

Ionic 2

- npm install –g ionic@beta cordova typescript
- ionic start ionic2-todo-ts blank --v2 --ts
- cd ionic2-todo-ts
- ionic serve -b
- Note: I renamed description to summary, if you ever port to iOS description equals toString

Ionic 2 Decorators

- @App
  - App is an Ionic decorator that bootstraps an application. It can be passed a number of arguments, that act as global config variables for the app
- @Page
  - The Page decorator indicates that the decorated class is an Ionic navigation component, meaning it can be navigated to using a NavController.

Ionic 2

- @App
  - node_modules\ionic-framework\decorators\app.*
- Navigation Controller (Tab,Nav)
  - node_modules\ionic-framework\components\nav*.*
- @Page
  - node_modules\ionic-framework\decorators\page.*

Ionic 2 @App

- @App declares this is the root component
- Inject in the constructor(platform: Platform)
- The Platform service provides information about the platform that the app is running on (e.g. width, height, landscape, portrait etc.).

Ionic 2 @App

```
constructor(platform: Platform) {
this.platform = platform;
this.initializeApp();
this.root = ListPage;
}
```

Ionic 2 @App

```
initializeApp() { // Lambda expression ()=>{}
    this.platform.ready().then(() => {
      console.log('MyApp.initializeApp Platform ready');
    });// Cordova Ready
  }
```

Ionic 2 TypeScript Walkthrough

- todoItem.ts: example(), native string, overload, getter, setter, implements, private, public
- Ijson.ts: interface
- evenbetterTodoItem.ts: extends, super
- App.ts: decorator, types, lambda, root
- ListPage.ts: dataservice promise, lambda, new clases, additem (pop lambda), listitem
- Add-Item.ts: constructor, init overloads, pop
- Item-detail.ts: events, ionic serve c, debug

Ionic 2 TypeScript Summary

- (Ionic2, Angular2) => { $ } //more money

- More TypeScript examples

https://github.com/Microsoft/TypeScriptSamples.git