TypeScript Primer

About me

Senior Software Developer at eMoney Advisor

Developing web applications since 2004

Currently exploring reactivity and Svelte

https://deldreth.com

https://github.com/deldreth

...or being a web developer needed to be more complicated

... or how to impress and alienate your developer friends and coworkers

... or infuriate your coworkers with a single dependency

All jokes aside!

TypeScript is...

Superset of ECMAScript 2015

Transcompiled to JavaScript

Created by Microsoft

```
enum Color {
 Green,
 Blue
}

interface Fish {
 name: string;
 color: Color;
}

enum BettaVariety {
 VeilTail,
 Delta
}

interface Betta extends Fish {
 variety: BettaVariety;
}
```

Static Typing

Type annotations

No types? Regular dynamic typing is used

Enforces type consistency

```
const isValid: boolean = false;

const isNumeric: number = 100;

let isString: string = "Great string!";

isString = 3; // Nope!
```

Why?

Specific errors caught early

Ease the mental burden

Self documenting API

Shipped across platforms

```
interface User {
 name: string;
 age: string;
}

type NameAge = [string, number];

function* tuples(users: User[]): IterableIterator<NameAge>
{
 for (const user of users) {
   yield [user.name, user.age];
 }
}
```

Why not?

Up front investment

Constraints placed on application

Types may not exist for third party utilities

```
import React from "react";

interface WithLoadingProps {
 loading: boolean;
}

const withLoading = <P extends object>(
 Component: React.ComponentType<P>
): React.FC<P & WithLoadingProps> => ({
 loading,
 ...props
}: WithLoadingProps) => {
 if (loading) {
   return <LoadSpinner />;
 }

 return <Component {...props as P} />;
};
```

Ok, what about?

You still have to write tests

Current JavaScript may not be 100% compilable

Difficult to go back

```
// valid in JavaScript, but will throw
// Property 'bar' does not
// exist on type '{ foobar: boolean; }'
const foobar = { foo: true };
foobar.bar = false;
```

Basic Types

number, boolean, and string

Arrays, Tuples, and Enums

any, void, never, null, undefined, and object

```
const isValid: boolean = false;

const isNumeric: number = 100;

const isString: string = "Great string!";

const numberList: number[] = [1, 2, 3, 4];
const stringList: string[] = ["1", "2", "3", "4"];
const boolList: boolean[] = [true, false, true, false];

const nameAgeList: [string, number][] = [
  ["John", 23], ["Jane", 28]
];

enum Size {
 Small,
 Medium,
 Large
}
const shirtSize: Size = Size.Medium;
```

Interfaces

Shape of values

Contracts structure of data

Optional & readonly properties

OOP concepts

```
enum Color {
 Gold
}

interface Animal {
 color: Color;
}

interface Fish extends Animal {
 swim: () => boolean;
}

class Koi implements Fish {
 color: Color;

 constructor(color: Color) {
   this.color = color;
 }

 swim() {
   return true;
 }
}
```

Generics

Placeholder types

Allows for extensible typing

```
interface GenericClass<T> {
 add: (a: T, b: T) => T;
}

class TestString implements GenericClass<string> {
 add(a: string, b: string) {
   return a + b;
 }
}

class TestNumber implements GenericClass<number> {
 add(a: number, b: number) {
   return a + b;
 }
}
```

Generics

Apply constraints to parameters

```
interface Person {
 age: number;
}

function getAgePasses<T extends Person>(person: T): number {
 return person.age;
}

function getAgeFailure<T>(person: T): number {
 // Property 'age' does not exist on type 'T'
 return person.age;
}
```

Inference

Type inferred at initialization

Inferred types maintain typing

```
// inferred type number
let three = 3;
// Type 'false' is not assignable
// to type 'number'
three = false;

// inferred type (string | number)[]
let nameAgeHeight = ["John", 23, 128];

nameAgeHeight = ["Jane", "Doe", 323];

// Type 'false' is not assignable
// to type 'string | number'
nameAgeHeight = ["Sally", false];
```

Intersect, |

Type to be one type or another

Set equivalent to **or**

```
enum Cls { Aves, Mammalia }

interface Animal {
 classification: Cls;
}

interface Bat extends Animal {
 classification: Cls.Mammalia;
}

interface Bird extends Animal {
 classification: Cls.Aves;
}

type FlyingAnimal = Bat | Bird;

// darkAnimal is of type Bat
const darkAnimal: FlyingAnimal = {
 classification: Cls.Mammalia
};
```

Union, &

Types made of multiple types

Set equivalent to **and**

Extends

```
enum Color { Gold }

interface Animal { color: Color; }

interface CanSwim {
 swim: () => boolean;
}

type Fish = Animal & CanSwim;

class Koi implements Fish {
 color: Color;

 constructor(color: Color) {
   this.color = color;
 }

 swim() { return true; }
}
```

Use TypeScript today!

Visual Studio Code

// @ts-check

jsconfig.json

```
// @ts-check
let thing = 1234;
thing = "asdf";

/**
* Format a number
* @param num {number} Some decimal number
* @return {string}
*/
function format(num) {
 return num.toFixed(2);
}

format("1234");

{
 "compilerOptions": {
   "checkJs": true
 }
}
```

Questions