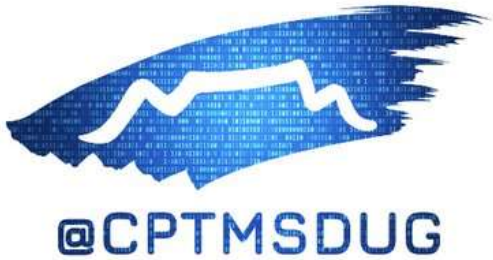
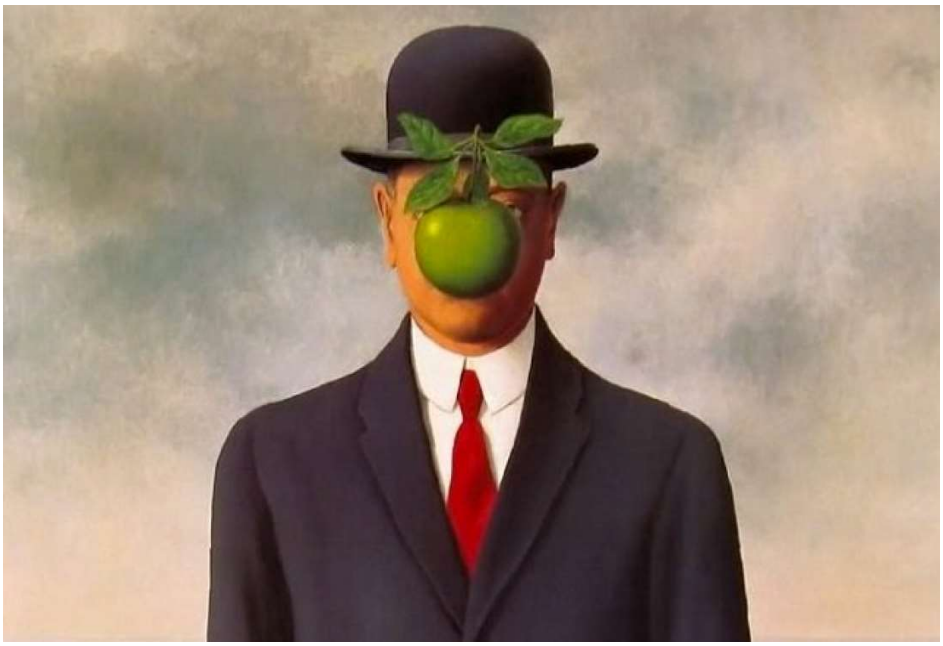


TYPESCRIPT BASICS

CPTMSDUG - September 2018

HELLO!



WHO AM I?

- Turbo Pascal
- Delphi
- J++
- C#
- TypeScript

JS

JS IN 3 MINS

JS is a high-level, interpreted programming language. It's dynamic, weakly typed, prototype-based and multi-paradigm

JavaScript is an event-driven and supports BOTH functional AND imperative (including object-oriented and prototype-based) programming styles

Single threaded event loop architecture with background workers

JAVASCRIPT RUNTIME

Single threaded event driven (loop) architecture with background workers (concurrent)

loupe

Save + Run

```
setTimeout(function one () {
    console.log("ONE!")
}, 10000);

setTimeout(function two () {
    console.log("TWO!")
}, 5000);

console.log("THREE!");
```

Call Stack

Web Apis

Callback Queue

loupe

Save + Run

```
setTimeout(function one () {
    console.log("ONE!")
}, 10000);

setTimeout(function two () {
    console.log("TWO!")
}, 5000);

console.log("THREE!");
```

Call Stack

Web Apis

Callback Queue

JAVASCRIPT TYPES

Boolean
Number
String
Null
Undefined
Symbol

Object { }

Arrays are objects

FUNCTIONS() are OBJECTS

TS

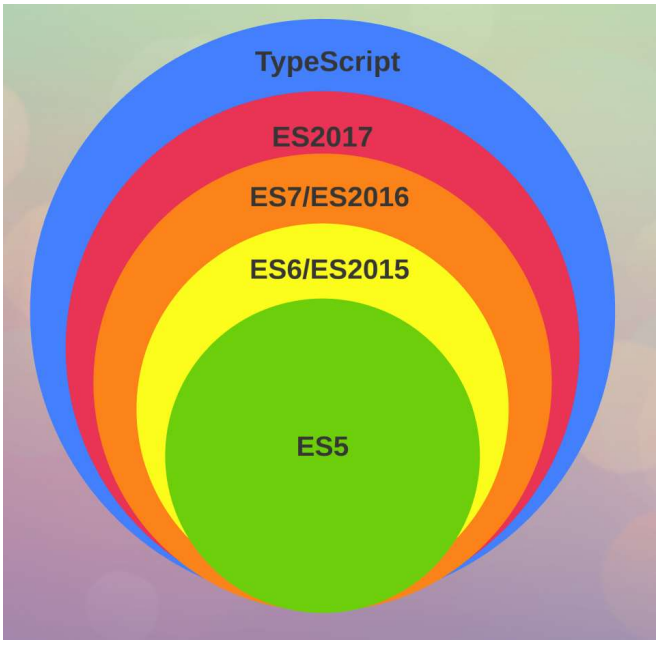
What is TS?

IT'S DEVELOPMENT TOOLING.

TRANSPILER

OPTIONAL TYPE CHECKING AT COMPILE TIME.

JS === TS, but TS !== JS



TC
39

https://github.com/tc39

TRANSPILER EXAMPLE

ASYNC

```
//ASYNC AWAIT
const timeout = millisecs => new Promise(res =>

(async function () {
    console.log(new Date());
    await timeout(2000);
    console.log(new Date());
})();
```

FEATURES!

TYPE ANNOTATIONS

```
function TriteTaxExample(amount) {
    return amount / 100 * 114;
}

console.log(TriteTaxExample(5000));
```

5700

TYPE ANNOTATIONS

```
function TriteTaxExample(amount) {
    return amount / 100 * 114;
}

console.log(TriteTaxExample("YOLO"));
```

NaN

TYPE ANNOTATIONS

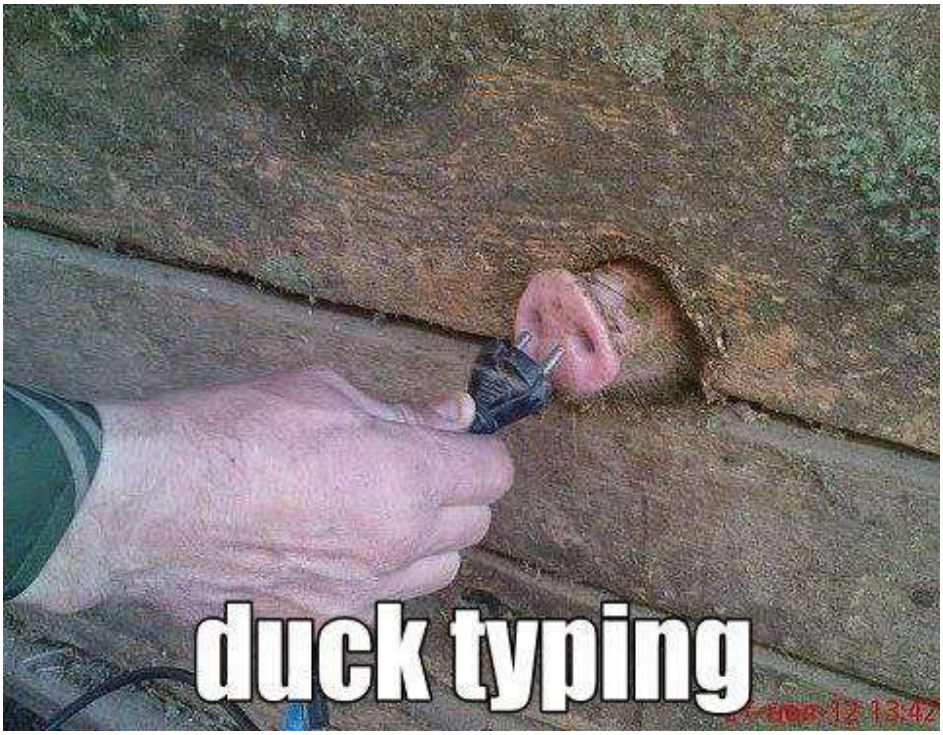
```
function TriteTaxExample(amount: number) {
    return amount / 100 * 114;
}

console.log(TriteTaxExample("YOLO"));
```

```
C:\code\cptmsug_tstalk\codesamples>tsc ta.ts
ta.ts:5:29 - error TS2345: Argument of type '"YOLO"' is not assignable to parameter of type 'number'.

5 console.log(TriteTaxExample("YOLO"));
                              ~~~~~~
```

destroyallsoftware.com/talks/wa


duck typing

CLASSES

Classes in TS (or > JS ES6) are object factories.

```typescript
class Person {
    private _name: string;
    public get name(): string {
        return this._name;
    }
    public set name(v: string) {
        this._name = v;
    }
    constructor(name: string) {
        this._name = name;
    }
}

const logName = (p: Person) => console.log(p.name);
const p = new Person("Brett");
p.name = "Mutated Brett";
logName(p);
```

```
class Person {
    constructor(public name: string) {
    }
}

const logName = (p: Person) => console.log(p.name);
const p = new Person("Brett");
p.name = "Mutated Brett";
logName(p);
```

```typescript
class Person {
    constructor(public readonly name: string) {
    }
}

const logName = (p: Person) => console.log(p.name);
const p = new Person("Brett");
p.name = "Mutated Brett";
logName(p);
logName({name: "Totally Fine Duck Typing"});
logName({name: 123456789});
logName({name: "Totally Fine", ExtraStuff: "Not fine"});
```

```
32  class Person<T> {
33      constructor(public readonly name: T) {
34      }
35  }
36
37  const logNameNoType = (p: Person) => console.log(p.name);
38  const logName = (p: Person<number>) => console.log(p.name);
39  const p = new Person<number>("Brett");
40  logName(p);
41  logName({name: "Not the right shape duck"});
42  logName({name: 123456789}); //We take numbers now.
43
```

[ts] Generic type 'Person<T>' requires 1 type argument(s).

class Person<T>

INTERFACES

Only exists within the context of TypeScript used solely for Type Checking

```
interface IThing {
    someString: string;
    someNumber: number;
    optionalBool?: boolean;
}

function func(m:IThing){}

func({someNumber:1}) `***ERROR***`
func({someNumber:1, someString: "2"})
func({someNumber:1, someString: "2", optionalBool: true
func({someNumber:1, someString: "2", randomJunk: {}})
```

```
interface IThing {
    someString: string;
    someNumber: number;
    optionalBool?: boolean;
}
//extends
interface IMoreThing extends IThing {
    randomJunk: boolean;
}
function func(m:IMoreThing){}

func({someNumber:1, someString: "2"}) `***ERROR***`
func({someNumber:1, someString: "2", randomJunk: true})
func({someNumber:1, someString: "2", optionalBool: true
```

```
//generics
interface IGenericThing<T, K> {
    someString: string;
    someGeneric: T
    OptionalGeneric?: K
}


function func(m:IGenericThing<Boolean, Number>){}

func({someString:"YOLO", someGeneric: true})
func({someString:"YOLO", someGeneric: "true"}) `***ERRO
func({someString:"YOLO", someGeneric: true, OptionalGen
```

TYPE | GUARDS

```
function guarded(arg: Number | String) {

}
guarded(1);
guarded("1");

        [ts] Argument of type 'true' is not assignable to parameter of
         type 'String | Number'.
guarded(true);
```

ENUMS

```typescript
enum Things {
    Boat,
    Shoe,
    Banana,
    Whatever = 99
}

function DoThing(thing: Things){
    switch (thing){
        case Things.Banana: console.log('Yellow'); break;
        case Things.Shoe: console.log('Laces!'); break;
        default: throw new Error(`DIDN'T DEAL WITH THING!`)
    }
}

DoThing("BANANA?");
DoThing(Things.Banana);
```

```
[ts] Argument of type '"BANANA?"' is not assignable to paramete
r of type 'Things'.
```

DISCRIMINATED UNIONS & LITERAL TYPES

```
type myThings = "Ducky" | "Squirrel" | "Pancake"

function DoThing(thing: myThings){
    switch (thing) { //INFERS my TYPE HERE
        case "Pancake": console.log('Yellow'); break;
        case "Squirrel": console.log('Furry'); break;
    }
}

DoThing("Pancake");
DoThing("DUCKY"); `***ERROR***`
```

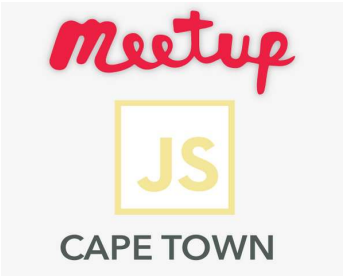
```
type myThings = "Ducky" | "Squirrel" | "Pancake"

function DoThing(thing: myThings){
    switch (thing) { //INFERS my TYPE HERE
        case "Pancake": console.log('Yellow'); break;
        case [ts] Type '"DUCKY"' is not comparable to type 'myThings'.
        case "DUCKY" : console.log('Quack'); break;
    }
}
```

THANKS

Complaints to @WrathZA

HTTPS://GITHUB.COM/BASARAT/TYPESCRIPT-BOOK



Next Meetup: September 12th