JS and TypeScript deep dive

What is TypeScript

TypeScritp is a typed superset of JavaScript that compiles to plain JavaScript.

Installation

```
npm install -g typescript
touch index.ts
tsc index.ts
```

You should have a file called `index.js` in your directory.

For comfort, from now own run

```
tsc index.ts && node index.js
```

Basic Types

TypeScript has basic types.

```
let decimal: number = 6;
let amItrue: boolean = true;
let myName: string = "";
```

Notice that number isn't Number

```
let name:string;
name = 10; // this should fail.
```

Arrays

You can define typed arrays.

```
let someNumbers: number[] = [1,2,3]
```

or

```
let someNumbers: Array<number> = [1,2,3]
```

Enum

And enum is a way of giving more friendly names to sets of numeric values.

```
enum Difficulty {Easy, Normal, Hard};
let gameplay : Difficulty = Difficulty.Easy;

console.log(gameplay); // 0
console.log(Difficulty[0]); // easy
```

Any

The type any is similar to the type `Object` in Java, but not to the type
`Object` in JS.

```
let notSure: any = 4;
notSure.ifItExists();
notSure.toFixed();

let prettySure: Object = 4;
prettySure.toFixed(); // Error
```

Functions

In JS:

```
const add = (a, b) => a + b;
```

In TS

```
const add = (a: number, b: number): number => a + b;
```

Function as Types

A variable can be typed as a very specific function with typed parameters.

```
let add: (a: number, b: number) => number;

add = (a: number, b: number): number => return a + b;

console.log(2, 3); // 5
```

Optional Parameters

A flexible tool in JS are default parameters as well as optional ones.

```
const happyBirthday = (name: string, age?: number): void => {
  if (!age) {
    console.log(`Happy Birthday, ${name}`);
  } else {
    console.log(`Happy ${age} Birthday, ${name}`);
  }
};

happyBirthday("Jesus", 33); // Happy 33 Birthday, Jesus
happyBirthday("Mark"); // Happy Birthday, Mark
```

Classes

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

class Rhino extends Animal {
    constructor() { super("Rhino"); }
}

class Employee {
    private name: string;
    constructor(theName: string) { this.name = theName; }
}

let animal = new Animal("Goat");
let rhino = new Rhino();

animal = rhino;
```

Private Methods

```
class Duck {
    private name: string;
    constructor(theName: string) {
      this.name = theName;
    }

    cuack():void {
      console.log(
        `${this.name} says: I am a humanoid. I do not bow nor quack!`
      );
    }
}

let duck = new Duck("Donald");
duck.cuack();
// Donald says: I am a humanoid. I do not bow nor quack!
```

Interfaces

One of TypeScript’s core principles is that type-checking focuses on the shape that values have.

This is sometimes called “duck typing” or “structural subtyping

Describing Objects

```
interface Config {
  environment: string,
  pwd: string,
  timeout: number,
  port?: number,
}
```

You could also add

```
[ propName : string]: any
```

for extra options.

Describing Functions

```
interface Search {
    (source: string, subString: string): boolean;
}

let finder = (
  subString: string,
  content: string,
  algorithm: Search
): boolean => algorithm(subString, content);
```

Some high-end JavaScript

Spread Syntax with Arrays

Spread syntax allows an iterable such as an array expression or string to be expanded.

```
const salad: string[] = ['lettuce', 'tomato', 'onions'];
const burger: string[] = ['meat', 'bread', ...salad];
console.log(burger);
// [ 'meat', 'bread', 'lettuce', 'tomato', 'onions' ]
```

Spread Syntax with Objects

```
const config: object = {env: 'prod', path: '/'};
const extendedConfig: object = {port: 9000, ...config};

console.log(extendedConfig);
// { port: 9000, env: 'prod', path: '/' }
```

This is the best thing since sliced 🍞 and will be super useful as we move forward.

Destructuing Assignment

Is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

```
// from
const port: number = config.port
const env: string = config.env

//to
const {port, env} = config
```

Destructuring Assignment with Arrays

```
const [meat, bread, ...vegan] = [
  'meat',
  'bread',
  'lettuce',
  'tomato',
  'onions'
];
console.log(meat); // meat
console.log(bread); // bread
console.log(vegan); // [ 'lettuce', 'tomato', 'onions' ]
```

Exports

Any declaration can be exported by adding the export keyword.

```
export const numberRegexp = /^[0-9]+$/;

export class ZipCodeValidator {
    isAcceptable(s: string) {
        return s.length === 5 && numberRegexp.test(s);
    }
}
// exports {numberRegexp: /^[0-9]+$/, ZipCodeValidator: class ZipCodeValidator}
```

Named Exports

You don't have to be constrained by your implementation's naming conventions.

```
const numberRegexp = /^[0-9]+$/;

class ZipCodeValidator {
    isAcceptable(s: string): boolean => s.length === 5 && numberRegexp.test(s)
}

export {ZipCodeValidator as Validator}

// this will export {Validator: class ZipCodeValidator}
```

Default Exports

You don't have to be constrained by your implementation's naming conventions.

```
const numberRegexp = /^[0-9]+$/;

export default class ZipCodeValidator {
    isAcceptable(s: string) {
        return s.length === 5 && numberRegexp.test(s);
    }
}

// this will export the class ZipCodeValidator
```

Importing Modules

Import a single export from a module

```
import { ZipCodeValidator } from './validators';
```

Importing Modules

Import a single export from a module

```
import { ZipCodeValidator } from './validators';
```

Imports can also be renamed

```
import { ZipCodeValidator as Validator } from './validators';
```

Importing Modules

Import a single export from a module

```
import { ZipCodeValidator } from './validators';
```

Imports can also be renamed

```
import { ZipCodeValidator as Validator } from './validators';
```

Import the entire module into a single variable

```
import Validator from './validators';
```

Importing Modules

Import a single export from a module

```
import { ZipCodeValidator } from './validators';
```

Imports can also be renamed

```
import { ZipCodeValidator as Validator } from './validators';
```

Import the entire module into a single variable

```
import Validator from './validators';
```

FIN / Questions?