TypeScript 101

Overview

Compile

Install

```
npm install -g typescript
```

Compile

```
tsc xxx.ts
```

Type annotations

With type annotations we can implicitly typecheck parameters. In that case the parameter is also required.

```
function iTakeAString(name: string){}
```

Using interfaces as types

```
interface Rectangle {
    x: number;
    y: number;
}

function area(room : Rectangle){}
```

ES6 like classes

```
class Address {
    description : string;
    constructor(public number, public street, public city) {
        this.description = number + " " + street + ", " + city;
    }
}

var home = new Address("11", "Foch avenue", "Paris");
```

Detailed

Types

```
var imABool: boolean = true;
var imANumber: number = 42;
var imAString: string = 'Hi';
var imAnArrayOfNumber: number[] = [1, 2];
var imAnArrayOfNumber: Array<number> = [1, 2];
```

Enums

```
enum Light {Green, Yellow, Red};
var current: Light = Light.Red;
```

Special types

Any

```
var myList:any[] = ['test', true, 18];
```

Void

```
function echoSemething(): void {}
```

Interfaces

Interfaces are complex types. You can either create an interface or directly use the objectf as a type.

Anonymous interface

```
function grabTheName(item: {name: string}) {}
```

Declared interface

```
interface Person {
    name: string;
}

function grabTheName(item: Person) {}
```

Optional property

```
interface Person {
    name?: string;
}
```

Functions

```
interface calculateArea {
    (x: number, y: number): number;
}
```

Implementing an interface

```
interface simpleAreaItf {
    x: number;
    y: number;
    getArea() : number;
}

class Square implements simpleAreaItf  {
    x: number;
    x: number;
    getArea ...
    constructor ...
}
```

Extending an interface

```
interface Living {
    height: number;
}

interface Person extends Living {
    name: string;
}
```

Classes

Inheritance

```
class Genius extends Person {
    constructor(name: string) { super(name); }
    ...
}
```

Public/private

Properties are public by default.

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

Customised property I/O

```
class Person {
    private _name: string;

    get name(): string {
        return this._name;
    }

    set _name(newName: string) {
        if (...) {
            this._name = newName;
        }
    }
}
```

Static property

```
class Location {
    static latBounds = {
        min: 0,
        max: 90
    };
    ...
}
```

Modules

A modules is the TypeScript transcription of namespaces. It's mainly use for complex applications.

Use export pour create an object reachable from outside.

```
module Xx {
    export interface Zz {}
    export class Yy implements Zz {}
}

var n : Xx.Zz = new Xx.Yy();
```

Use 'reference' to import splitted files

```
/// <reference path="xxx.ts" />
```

Functions

Typing

```
function add(x: number, y: number): number {
    return x+y;
}
```

Optional and by default parameter

```
function area(x?: string, y = 3) {...}
```

Rest parameters

```
function globalName(firstName: string, ...otherNames: string[]) {}

var name = buildName("name1", "name2", "name3", "name3", ...);
```

Generics

```
function identity<T>(arg: T): T {
    return arg;
}
```