TypeScript

The Basics

COS216
AVINASH SINGH
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF PRETORIA

TYPESCRIPT - OVERVIEW

- Open-source, released October 2012, latest version 5.0.4
- Developed and Maintained by Microsoft
- Strict Superset of JavaScript
- Designed for large scale systems
- Mostly found in Angular products
- Needs NodeJS to run or can be converted to JavaScript

WHY TYPESCRIPT?

- It is quite similar to JS
- Provide an optional type system for JavaScript
- Provide planned features from future JavaScript editions to current JavaScript engines
- Types have proven ability to enhance code quality and understandability

- To install typescript run the command:

```
npm install –g typescript
```

- Now you will get the **tsc** command which is the typescript compiler

```
tsc  --version
```

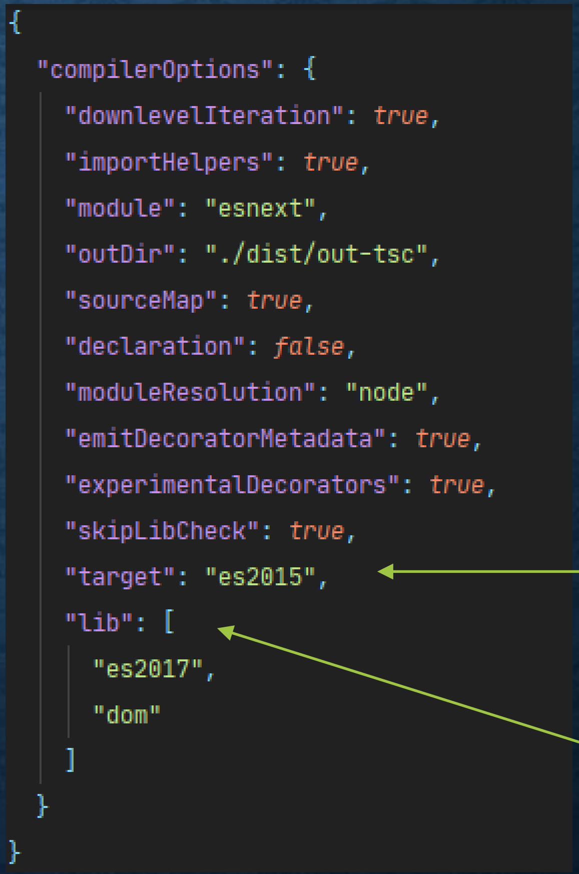
TYPESCRIPT

- A **type** declaration statement specifies the type, length, and attributes of objects and functions. You can assign initial values to objects.
- Typescript cannot be executed in its raw format and has a file extension (.ts, .tsx)
- It has to be converted to Vanilla JavaScript in order to execute (transpiled)

  To convert ts to js simple run -> tsx index.ts and that will be compiled to ES6 JS into a index.js file

TYPESCRIPT

- TS can be configured per project.
- These configurations are usually saved in ***tsconf.json***



```json
{
  "compilerOptions": {
    "downlevelIteration": true,
    "importHelpers": true,
    "module": "esnext",
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "skipLibCheck": true,
    "target": "es2015",
    "lib": [
      "es2017",
      "dom"
    ]
  }
}
```

Version of JS to transpile to

Set typings for certain environments

TYPESCRIPT - PRIMITIVES

String *'string'*

Number *785*

Boolean *true*

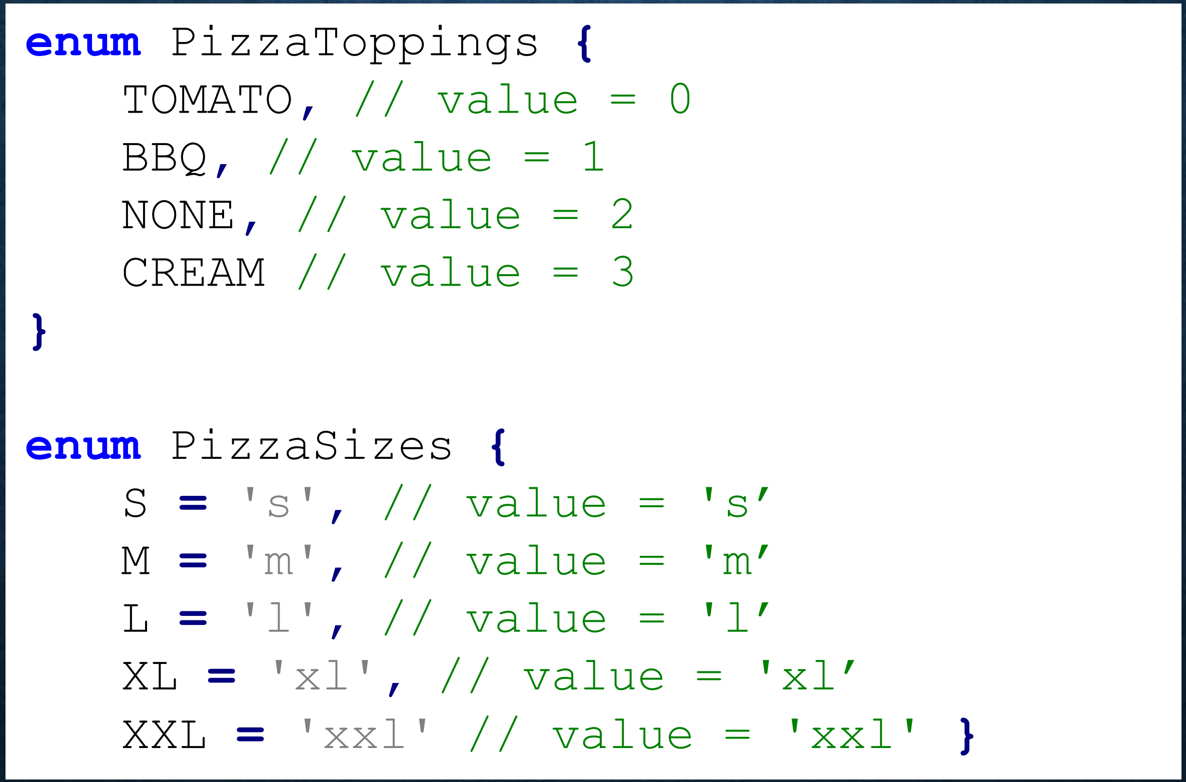
Undefined *undefined*

Null *null*

Object *{}*

Function *function fake() {…}*

TYPESCRIPT - ENUM

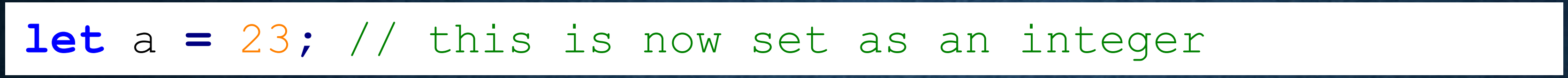


```
enum PizzaToppings {
    TOMATO, // value = 0
    BBQ, // value = 1
    NONE, // value = 2
    CREAM // value = 3
}

enum PizzaSizes {
    S = 's', // value = 's’
    M = 'm', // value = 'm’
    L = 'l', // value = 'l’
    XL = 'xl', // value = 'xl’
    XXL = 'xxl' // value = 'xxl' }
```

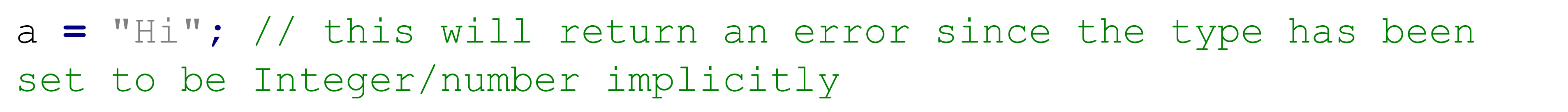
TYPESCRIPT – VARIABLE DECLARATIONS

- ***You can declare variable types implicitly and explicitly***
- ***Implicitly:***



```
let a = 23; // this is now set as an integer
```

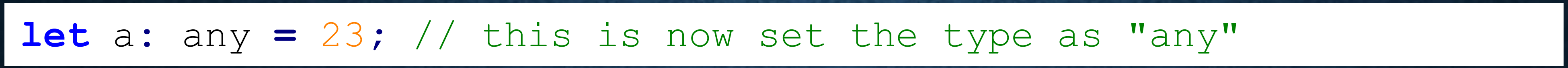
This means that a datatype will be set to integer and so the type can’t be changed



```
a = "Hi"; // this will return an error since the type has been
set to be Integer/number implicitly
```

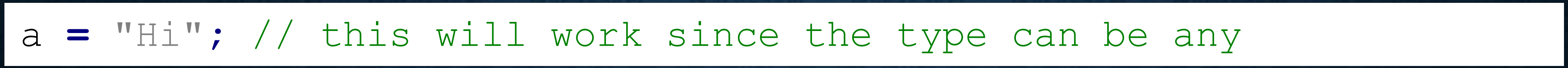
TYPESCRIPT – VARIABLE DECLARATIONS

- ***You can declare variable types implicitly and explicitly***
- ***Explicitly:***



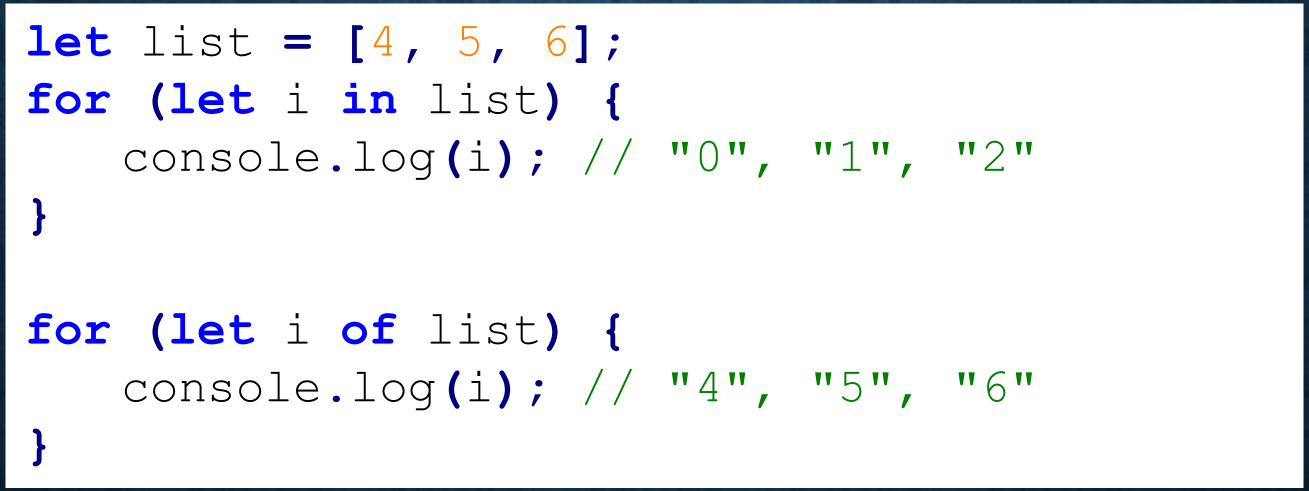
```
let a: any = 23; // this is now set the type as "any"
```

This means that a datatype will be set to be any and so the type can be changed dynamically



```
a = "Hi"; // this will work since the type can be any
```

TYPESCRIPT – LOOPS



```
let list = [4, 5, 6];
for (let i in list) {
    console.log(i); // "0", "1", "2"
}

for (let i of list) {
    console.log(i); // "4", "5", "6"
}
```

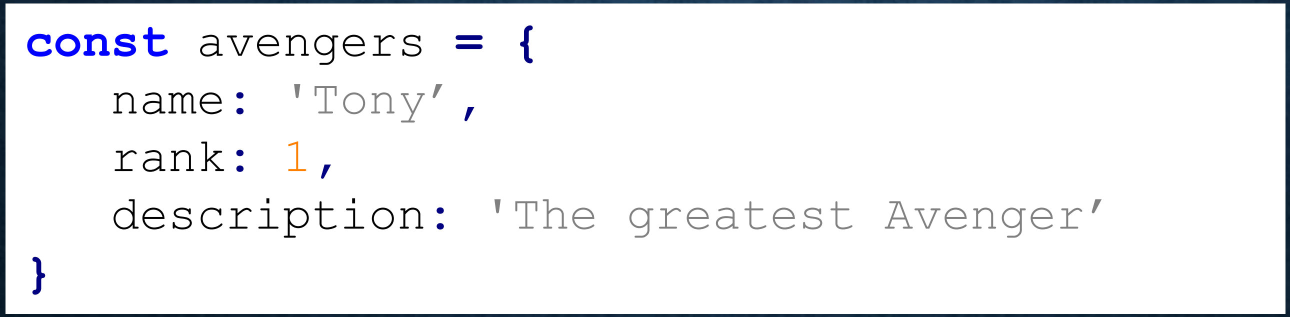
TYPESCRIPT – TYPES

- ***You can create your own custom types***

```
type Style = 'italic' | 'bold' | 'normal’;
let font: Style;
font = 'something'; // this will return an error because it can only be
                    // 'italic' or 'bold' or 'normal'
```

TYPESCRIPT – OBJECTS

- ***How to create a Object***



```
const avengers = {
   name: 'Tony’,
   rank: 1,
   description: 'The greatest Avenger’
}
```

TYPESCRIPT – INTERFACES

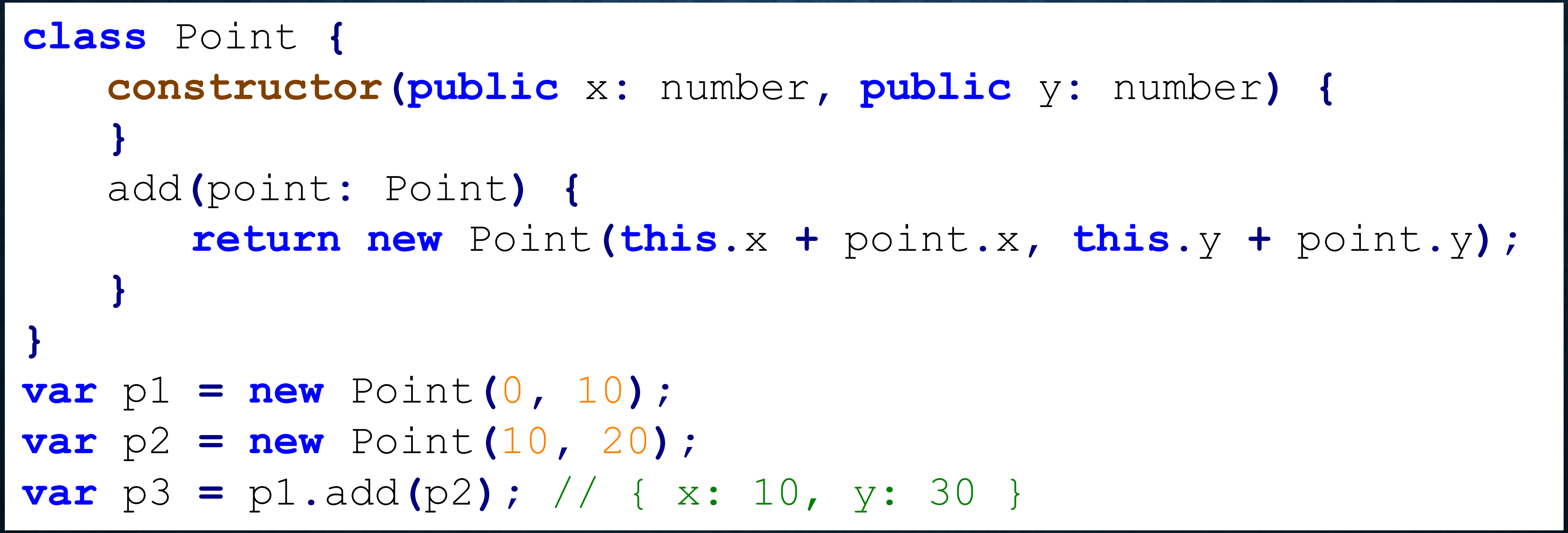
- ***However, that was not typed (i.e. there were no declarations for variable data types), How do we fix this? With interfaces ☺***

```
interface Avenger {
    name: String,
    rank: number,
    description: String
}
```

```
const avenger: Avenger = {
    name: 'Tony’,
    rank: 1,
    description: 'The greatest Avenger’
}
```

TYPESCRIPT – CLASSES

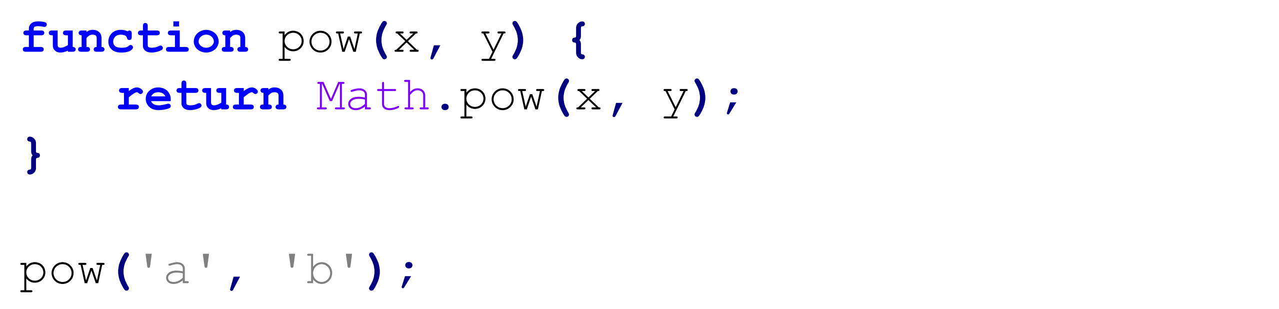
- ***What about Classes? Just like all OOP languages Typescript also allows Classes.***
- ***It was created to simplify OOP in JS since JS wasn’t built for OOP in its design.***



```
class Point {
    constructor(public x: number, public y: number) {
    }
    add(point: Point) {
        return new Point(this.x + point.x, this.y + point.y);
    }
}
var p1 = new Point(0, 10);
var p2 = new Point(10, 20);
var p3 = p1.add(p2); // { x: 10, y: 30 }
```

TYPESCRIPT – FUNCTIONS

- *JS doesn’t have typings like the function below, what would happen if the function was called like that?*

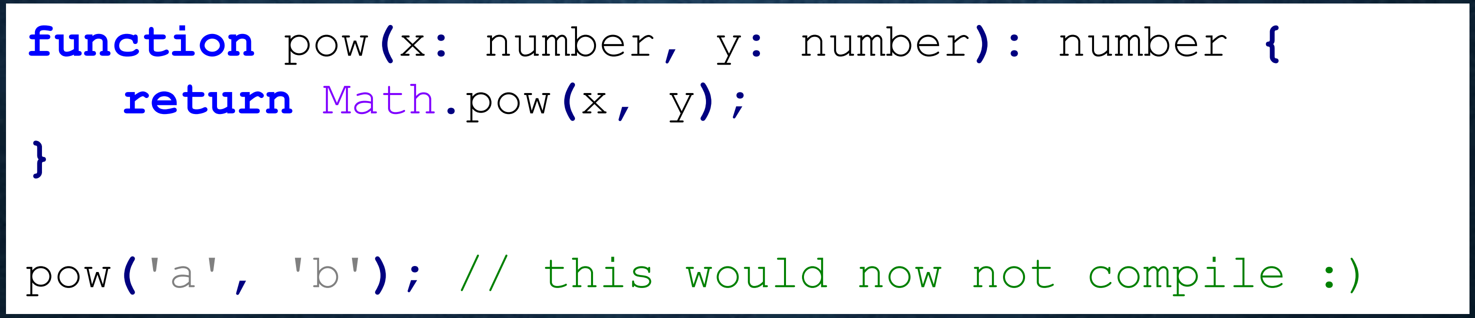


```
function pow(x, y) {
    return Math.pow(x, y);
}

pow('a', 'b');
```

TYPESCRIPT – FUNCTIONS

- ***This can be fixed using Typescript***



```
function pow(x: number, y: number): number {
    return Math.pow(x, y);
}

pow('a', 'b'); // this would now not compile :)
```

TYPESCRIPT – ARRAYS

- ***In JS arrays can have dynamic types that change based on the data stored***
- ***This issue can be solved with TS by defining arrays to be of a certain type***

```
const arr = []; // by default this has a type “any”
const arr: number = [];
arr.push("COS"); // this will give an error
arr.push(216);
arr.push(true); // this will give an error
```

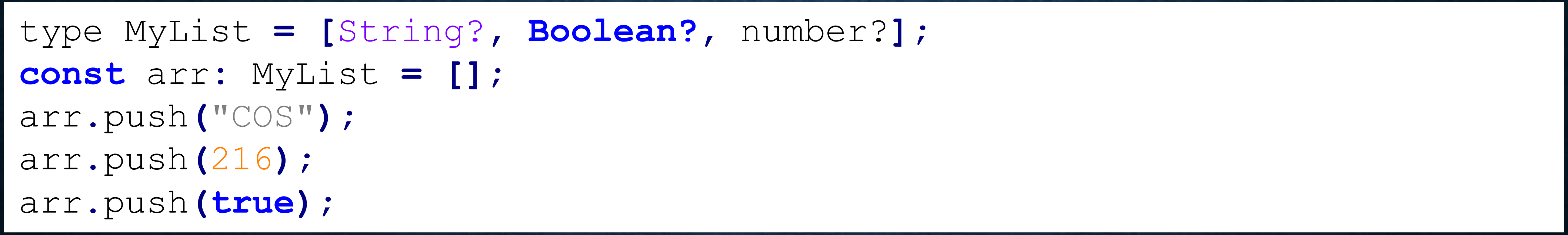
TYPESCRIPT – TUPLES

- ***Tuples like python can have mixed data types, with a fixed length array***

```
type MyList = [String, boolean, number];
const arr: MyList = []; // however this declaration will complain, why?
arr.push("COS");
arr.push(216);
arr.push(true);
```

TYPESCRIPT – TUPLES

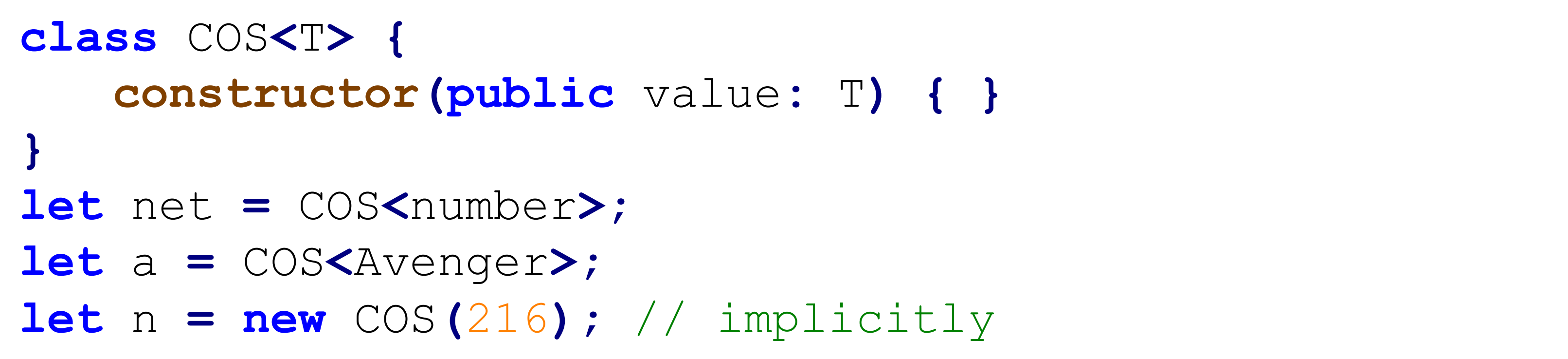
- *How do we solve this issue? With the magic "?", in Typescript if a parameter is "optional" then you can specify it by adding a question mark to the end of the type*



```
type MyList = [String?, Boolean?, number?];
const arr: MyList = [];
arr.push("COS");
arr.push(216);
arr.push(true);
```

TYPESCRIPT – GENERICS?

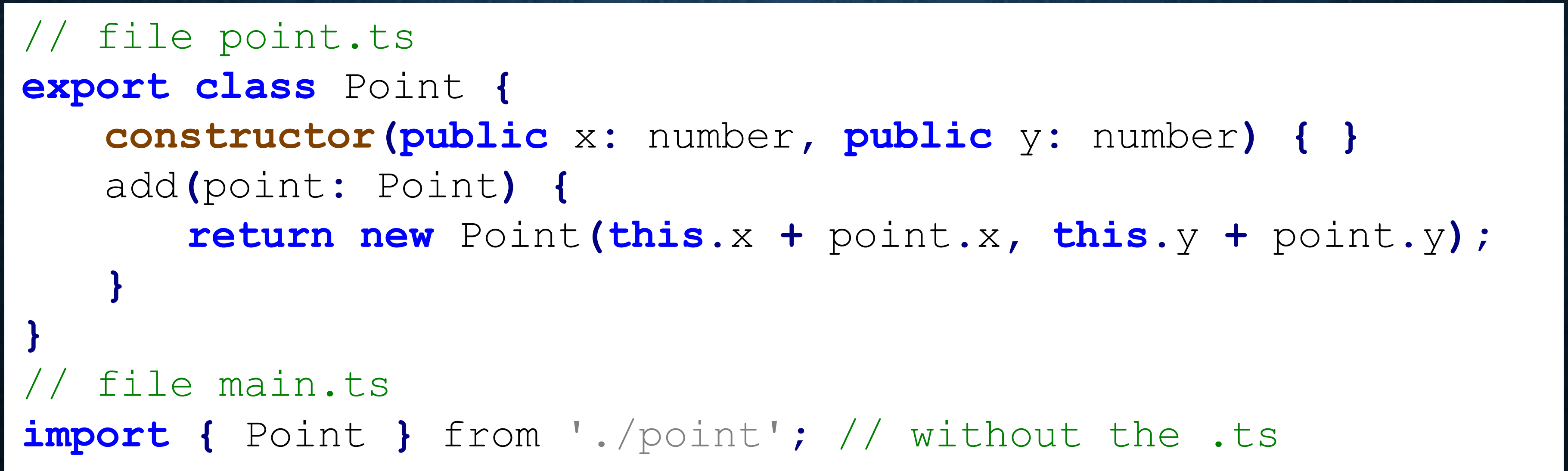
- ***Like C++ templates from COS110, Typescript also provides a developer with the ability to provide templates, however it is tagged as generics (since it applies to objects)***



```
class COS<T> {
    constructor(public value: T) { }
}
let net = COS<number>;
let a = COS<Avenger>;
let n = new COS(216); // implicitly
```

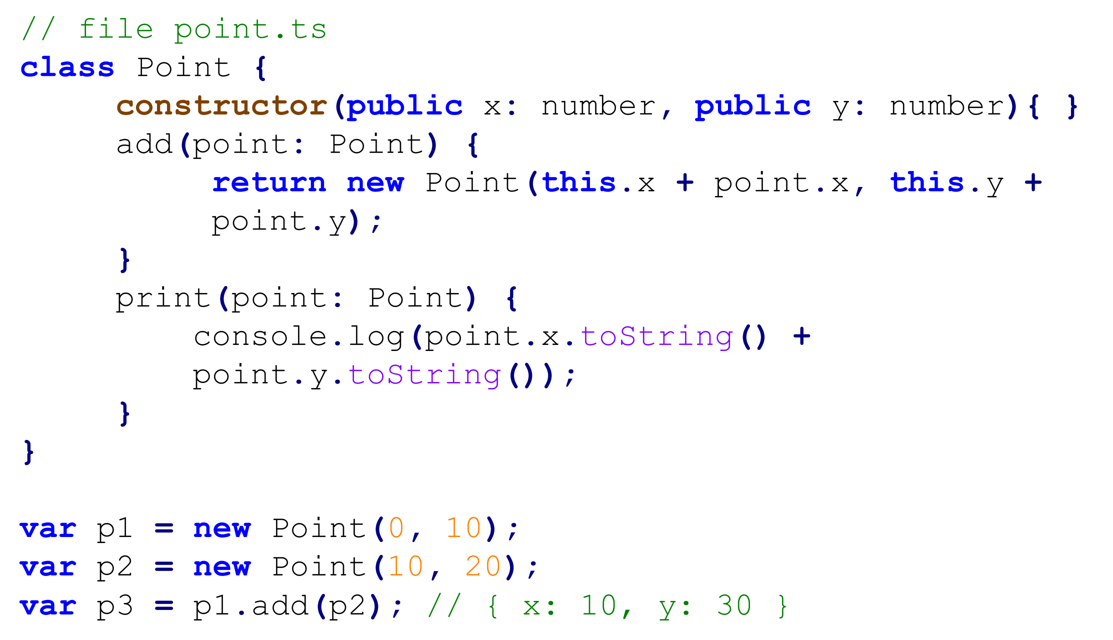
TYPESCRIPT – EXPORT AND IMPORT

- ***Remember from the NodeJS lectures there was an export keyword? This is also available in Typescript.***



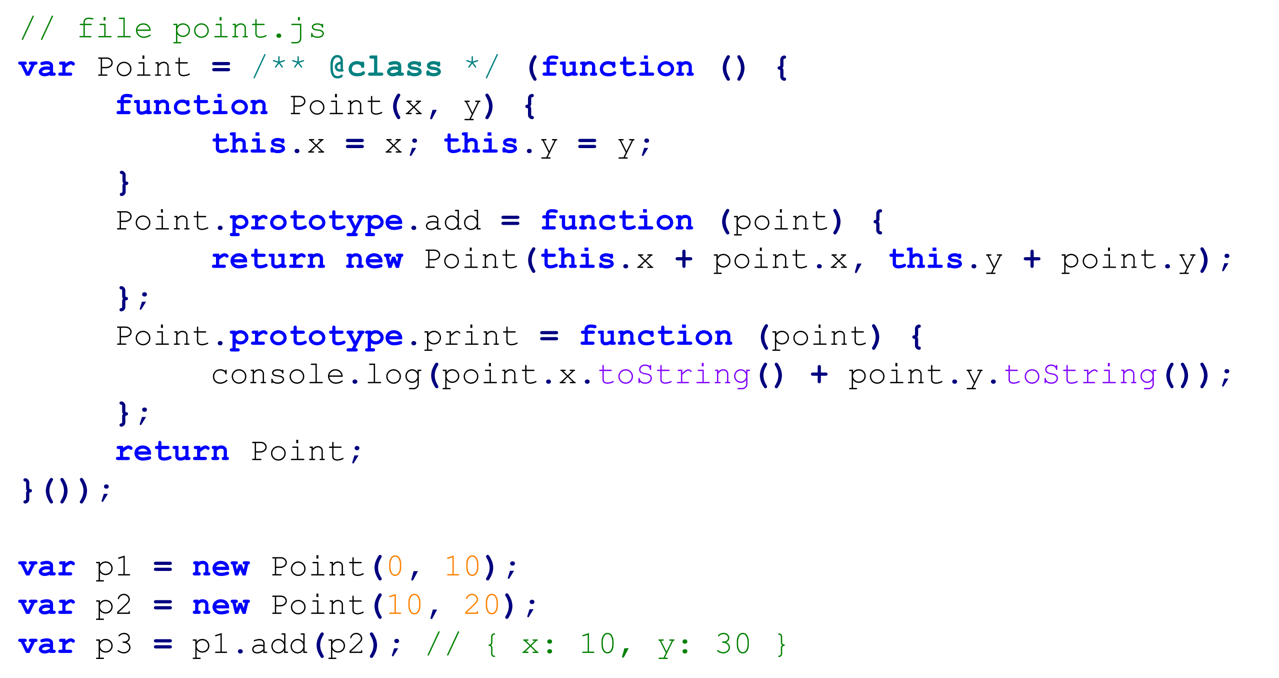
```
// file point.ts
export class Point {
    constructor(public x: number, public y: number) { }
    add(point: Point) {
        return new Point(this.x + point.x, this.y + point.y);
    }
}
// file main.ts
import { Point } from './point'; // without the .ts
```

TYPESCRIPT – THE DIFFERENCE?



```
// file point.ts
class Point {
     constructor(public x: number, public y: number){ }
     add(point: Point) {
          return new Point(this.x + point.x, this.y +
          point.y);
     }
     print(point: Point) {
         console.log(point.x.toString() +
         point.y.toString());
     }
}

var p1 = new Point(0, 10);
var p2 = new Point(10, 20);
var p3 = p1.add(p2); // { x: 10, y: 30 }
```



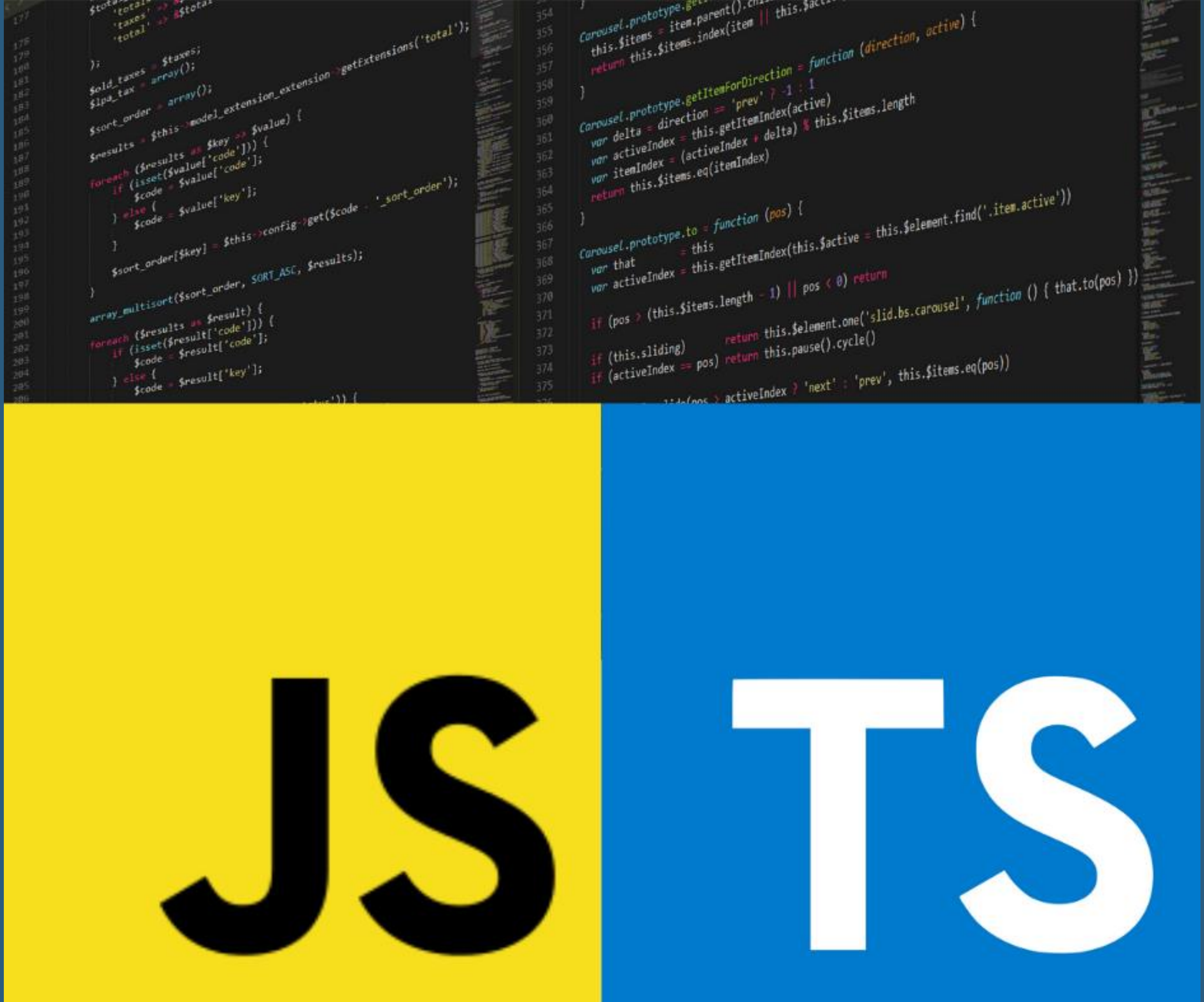
```
// file point.js
var Point = /** @class */ (function () {
     function Point(x, y) {
          this.x = x; this.y = y;
     }
     Point.prototype.add = function (point) {
          return new Point(this.x + point.x, this.y + point.y);
     };
     Point.prototype.print = function (point) {
          console.log(point.x.toString() + point.y.toString());
     };
     return Point;
}());

var p1 = new Point(0, 10);
var p2 = new Point(10, 20);
var p3 = p1.add(p2); // { x: 10, y: 30 }
```

TYPESCRIPT

Some more reading for you:

https://basarat.gitbook.io/typescript/



```
    'taxes' => $
    'total' => &$total
);

$old_taxes = $taxes;
$lpa_tax = array();

$sort_order = array();

$results = $this->model_extension_extension->getExtensions('total');

foreach ($results as $key => $value) {
    if (isset($value['code'])) {
        $code = $value['code'];
    } else {
        $code = $value['key'];
    }

    $sort_order[$key] = $this->config->get($code . '_sort_order');
}

array_multisort($sort_order, SORT_ASC, $results);

foreach ($results as $result) {
    if (isset($result['code'])) {
        $code = $result['code'];
    } else {
        $code = $result['key'];
```

```
Carousel.prototype.get
  this.$items = item.parent().ch
  return this.$items.index(item || this.$act
}

Carousel.prototype.getItemForDirection = function (direction, active) {
  var delta = direction == 'prev' ? -1 : 1
  var activeIndex = this.getItemIndex(active)
  var itemIndex = (activeIndex + delta) % this.$items.length
  return this.$items.eq(itemIndex)
}

Carousel.prototype.to = function (pos) {
  var that        = this
  var activeIndex = this.getItemIndex(this.$active = this.$element.find('.item.active'))

  if (pos > (this.$items.length - 1) || pos < 0) return

  if (this.sliding)       return this.$element.one('slid.bs.carousel', function () { that.to(pos) })
  if (activeIndex == pos) return this.pause().cycle()

                          activeIndex ? 'next' : 'prev', this.$items.eq(pos))
```