Week 1

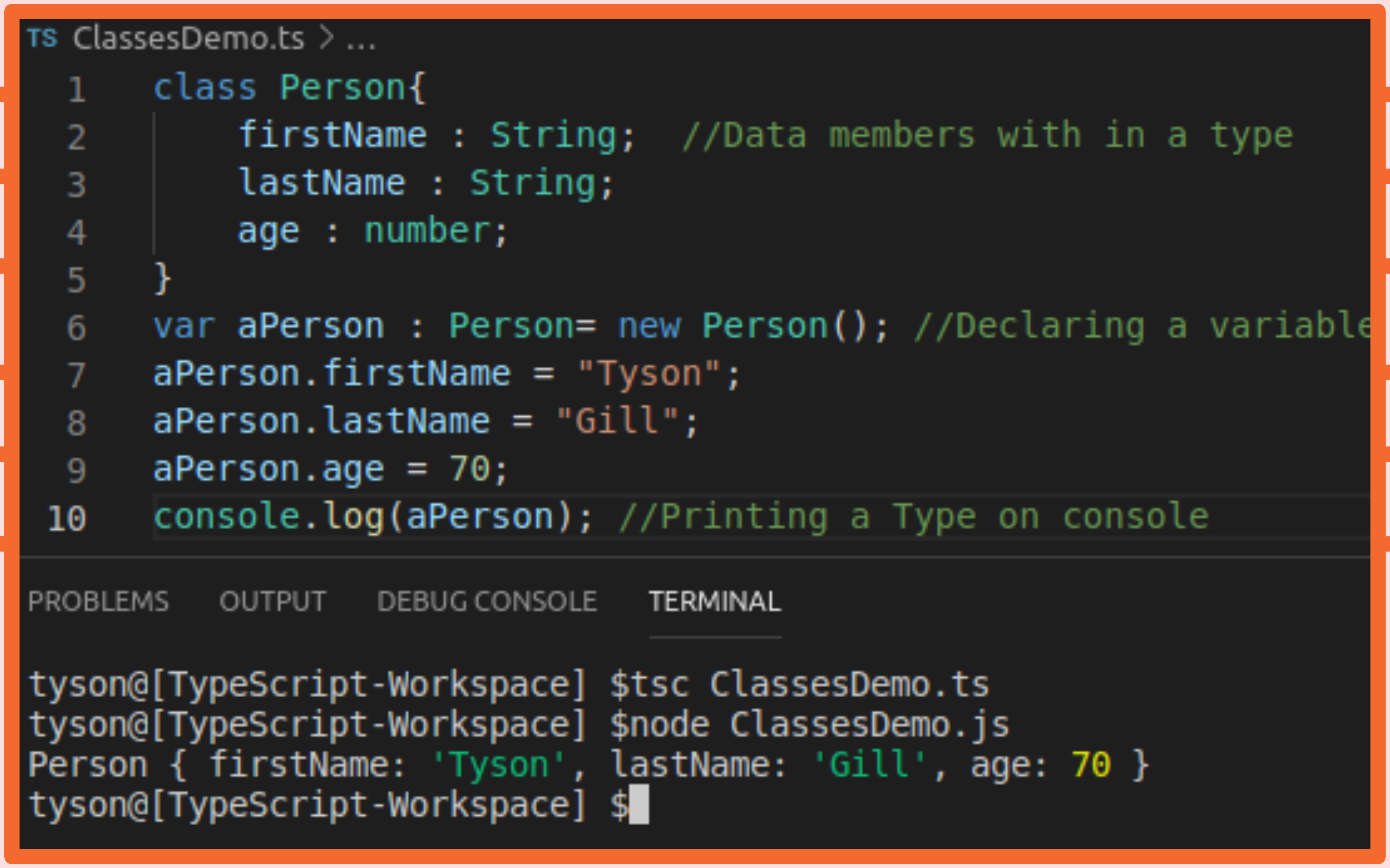
Typescript

What is typescript?

Typescript is a strongly typed programming language built on top of Javascript

- Type definitions
- Scalability
- Already very similar to typescript

Dynamic vs Strong Types



ClassesDemo.ts > ...

```
class Person{
    firstName : String;  //Data members with in a type
    lastName : String;
    age : number;
}
var aPerson : Person= new Person(); //Declaring a variable
aPerson.firstName = "Tyson";
aPerson.lastName = "Gill";
aPerson.age = 70;
console.log(aPerson); //Printing a Type on console
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
tyson@[TypeScript-Workspace] $tsc ClassesDemo.ts
tyson@[TypeScript-Workspace] $node ClassesDemo.js
Person { firstName: 'Tyson', lastName: 'Gill', age: 70 }
tyson@[TypeScript-Workspace] $
```

Dynamic

All data types are treated as interchangeable. This can make it easy to quickly make small programs, but can lead to hard-to-find bugs

e.g.- let bug = 10 + '1' // '101'

Strong

In a strongly typed language, the type of a construct does not change – an int is always an int, and trying to use it as a string will result in an error.

Using Typescript

Typescript is a strongly typed programming language built on top of Javascript

After installing Node.js

```
> npm install -g typescript

> tsc [./path/to/your/typescript/file]
```

Primitive Types

Typescript has a lot of the same basic types that you might find in other languages you might have used like c++, c#, or java

Number

A number can be an integer or a float

```
let x = 10; const y = 3.14159;
```

String

A string can be just one character, or entire sentences and paragraphs

```
let x = ‘hello, world’
```

Bool

A Boolean can be either true or false

```
let isItRaining = false;
```

Collection Types

Typescript has a lot of the same basic types that you might find in other languages you might have used like c++, c#, or java

Array

An array is a list of values

```
let arr: Number[] = [1,2,3]
```

Object

An object is a collection of key-value pairs

```
let obj = { a: 10, b: 'hi'}
```

Tuple

A fixed-length array with specific types at each index

```
let tup: [number,string] = [10,'abc']
```

Functions

In typescript, we can annotate what types of data are supposed to fit into our functions and what kinds of data are meant to be returned

```
function sum(a: number, b: number): number {
    return a + b;
}
sum(10,20); // 30
sum(10,’hi’); // error, mismatched types
```

Enums

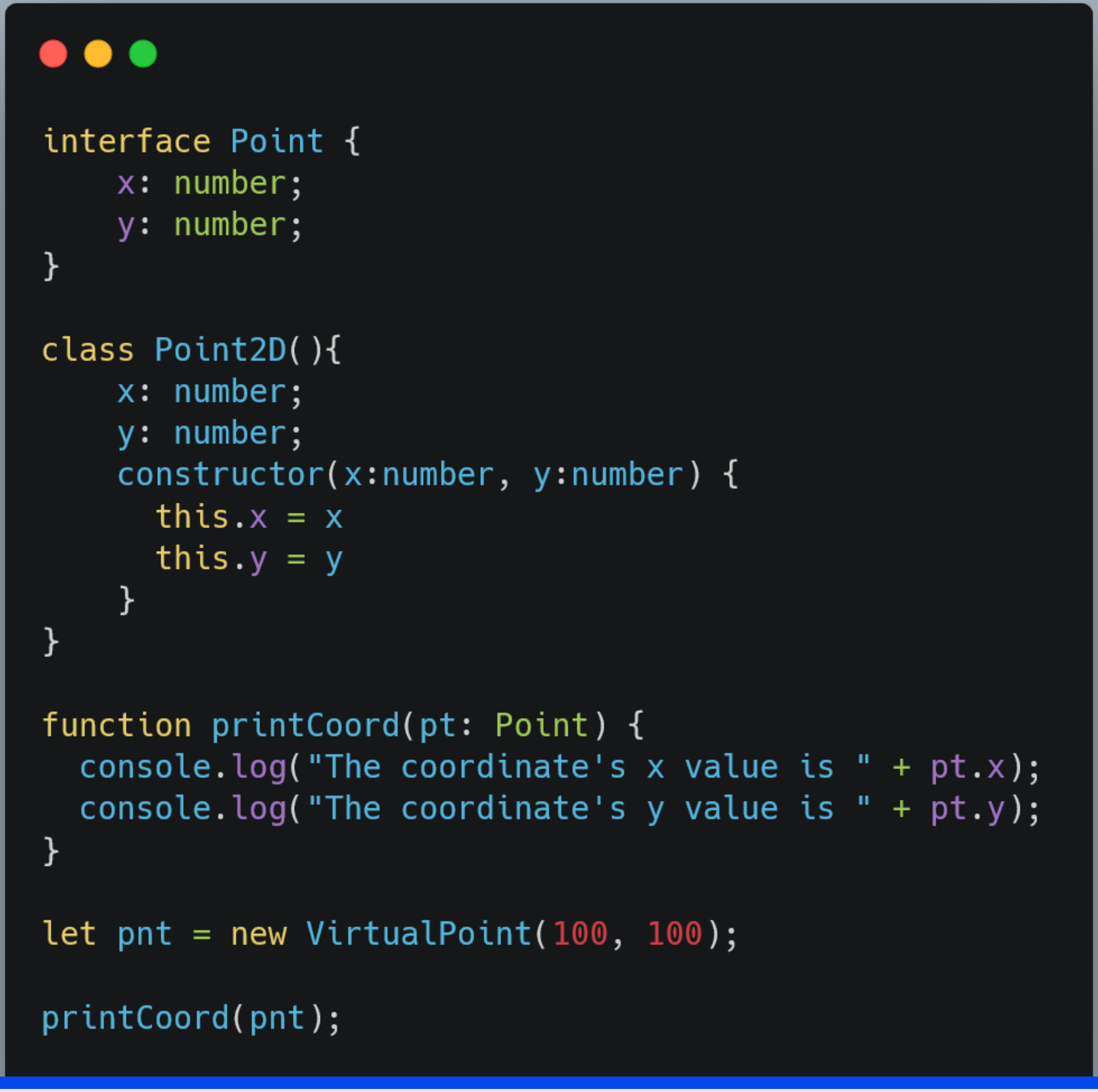
Enums are one of the few features TypeScript has which is not a type-level extension of JavaScript.

Enums allow us to define a set of named constants. Using enums can make it easier to document intent, or create a set of distinct cases. TypeScript provides both numeric and string-based enums.

```
enum Coin {
          Penny = 0,
          Nickle = 0.05,
          Dime = 0.1,
          Quarter = 0.25
}
enum Direction {
          Up,
          Down,
          Left,
          Right
}
```

Objects

Interfaces



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

class Point2D(){
    x: number;
    y: number;
    constructor(x:number, y:number) {
      this.x = x
      this.y = y
    }
}

function printCoord(pt: Point) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
}

let pnt = new VirtualPoint(100, 100);

printCoord(pnt);
```

Interfaces

Interface declarations are a way of naming object types

```
interface Point {
    x: number;
    y: number;
}
function printCoord(pt: Point) {
  console.log(`x:${pt.x}, y:${pt.y}`)
}
printCoord({x: 100, y: 100});
```

Classes

Objects made from classes can be interoperable with interfaces if they share the same shape of data

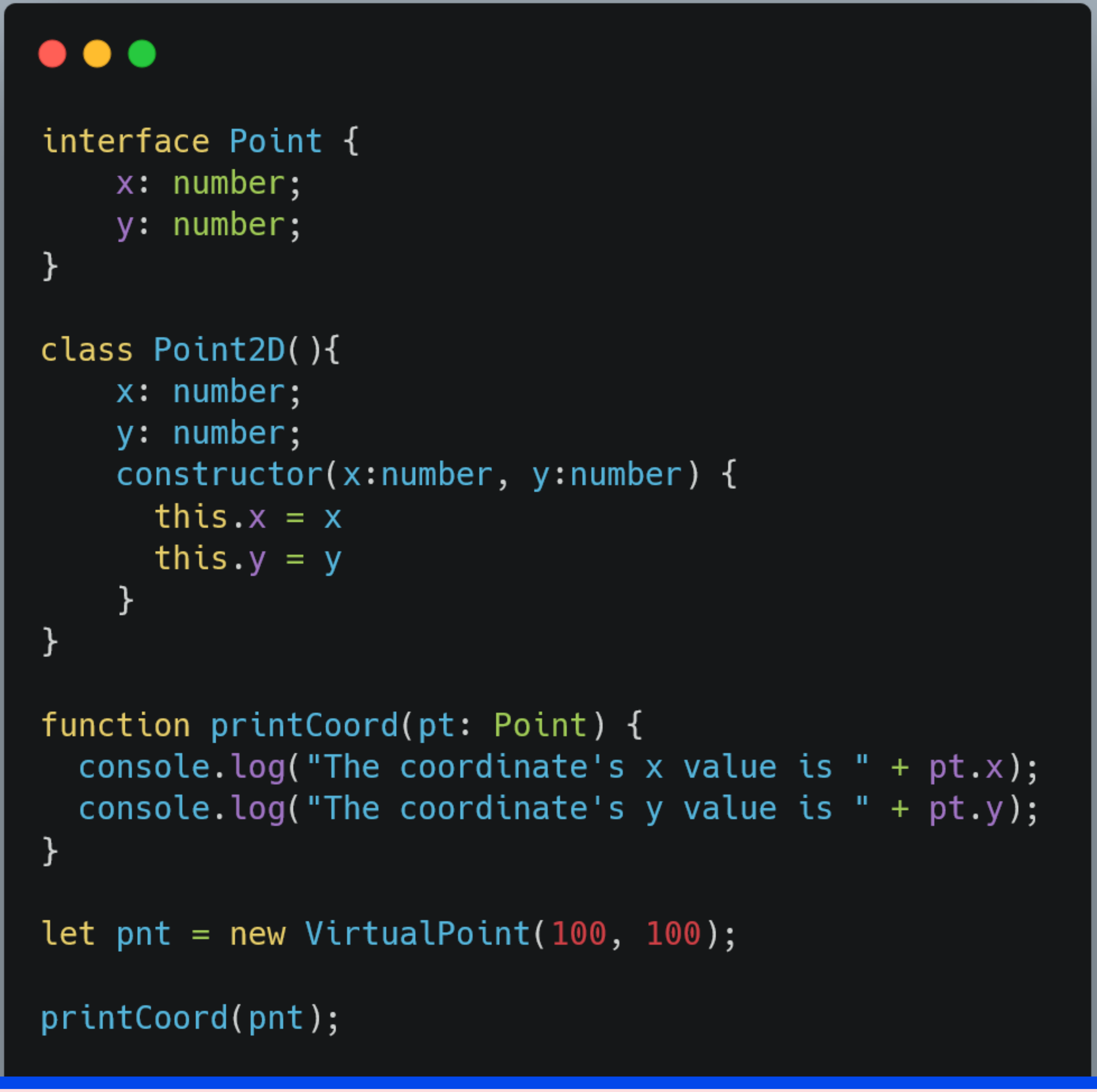
```
class VirtualPoint {
     x: number;
     y: number;
     constructor(x: number, y: number){
          this.x = x
          this.y = y
     }
}
Let pnt = new Point2D(100,100);
printCoord(pnt);
```

Unions and Intersections

```
interface Cat {
          name: string
          speak: Meow
}
interface Dog {
          name: string
          speak: Bark
}
type CatOrDog = cat & dog;
type CatAndDog = cat | dog;
```

Objects

Interfaces



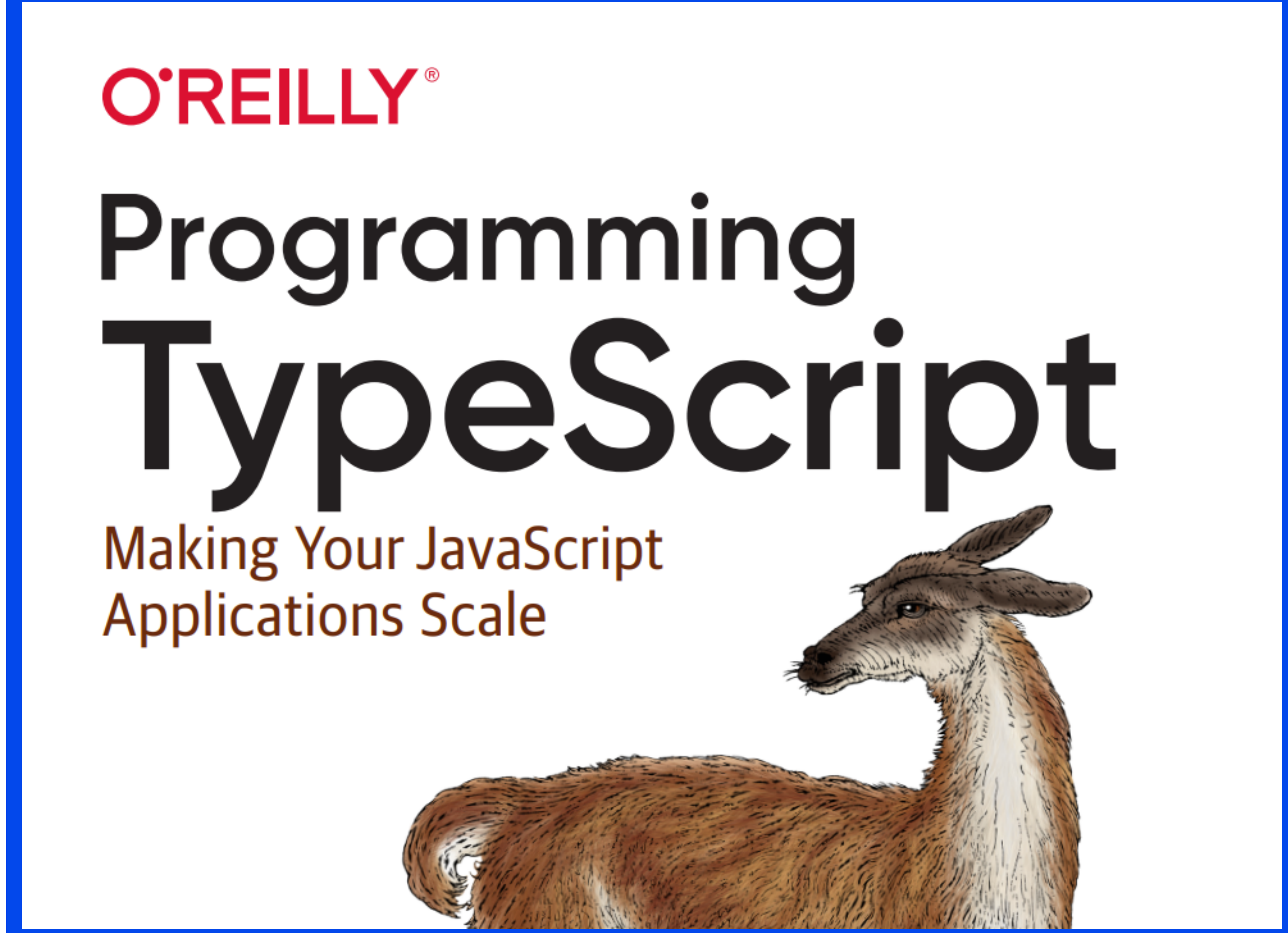
```
interface Point {
    x: number;
    y: number;
}

class Point2D(){
    x: number;
    y: number;
    constructor(x:number, y:number) {
      this.x = x
      this.y = y
    }
}

function printCoord(pt: Point) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
}

let pnt = new VirtualPoint(100, 100);

printCoord(pnt);
```



O'REILLY

Programming TypeScript

Making Your JavaScript Applications Scale

Homework for next week

Readings

Read Chapters 1-3 of Programming Typescript

Optional readings (Links on Brightspace):

- Eloquent JS, chapters 1-6
- Video Typescript Basics
- Ladybug Podcast on Typescript