The nodeCode scenario

Typescript for ABL developers

Introduction

- Me, myself and I
- Build.One
- What do I want to get out of this presentation ?

A journey in 5 parts

- TypeScript Unveiled: A Quick Overview
- The Case for TypeScript: Benefits and Opportunities
- Drawing Parallels: ABL and TypeScript Compared
- My Typescript Voyage
- Getting Started with TypeScript: Essential First Steps

Why a Typescript and ABL OO presentation ?

- Typescript classes very similar to ABL OO classes
- Helps with transition to javascript
- Opens new possibilities for the cool ABL OO developers
- Why not ?

Quick Poll - Who's into OO

- Never!
- Not yet
- "Researching"
- Sometimes. When I'm forced to.
- Mostly
- I'm a 4GL OO wizard. My OO foo is legendary.

Why OO is advantageous (ABL & Typescript)

- Type-safe for properties / methods
  - run “GetSalesValeus” in this-procedure
- Inheritance
  - One level deep only. Change my mind
- Code Reusability
  - Reusable components and classes reduce redundancy and promote consistency
- Enhanced Maintainability
  - OO principles contribute to more maintainable and scalable codebases.

Hang on a minute

- Thought this was a nodecode scenario
    - Where’s the node ?

Part I: A Typescript history (#0 of 3)

- Introduced by Microsoft in 2012
  - TypeScript was created to enhance JavaScript's capabilities.
- Designed by Anders Hejlsberg
  - The chief architect behind C# aimed to provide strong typing in JavaScript.
- Open-source release
  - An open-source project, allowing community contributions.
- Targeting large applications
  - Initially aimed at improving the maintainability of large-scale applications.

Part I: A Typescript history (#1 of 3)

BUILD.ONE

- First stable version (1.0)
  - Released in 2014
- Integration with JavaScript
  - TypeScript is a superset of JavaScript, making it easy to adopt existing JavaScript code.
- Adoption by major frameworks
  - Libraries like Angular embraced TypeScript, boosting its popularity.
- TypeScript 2.0 (2016)
  - Introduced features like null checking, type inference, and better support for generics.

Part I: A Typescript history (#2 of 3)

BUILD.ONE

- Expanded tooling support
  - Enhanced integration with IDEs (e.g., Visual Studio, VS Code)
- Growing community and ecosystem
  - Rapidly expanding community contributed to libraries and frameworks supporting TypeScript.
- TypeScript 3.0 and beyond (2018)
  - features like conditional types and improved type checking.
- Strong focus on developer experience
  - Continuous enhancements to type definitions and error messaging.

Part I: A Typescript history (#3 of 3)

- TypeScript in the age of React
  - Became a preferred choice for modern frontend development alongside React
- Continuous updates
  - Regular releases and updates to improve performance and add new features.
- Vision for the future
  - TypeScript aims to remain a cornerstone for scalable, maintainable JavaScript development.

Part I: A History of ABL (#1 of 10)

- Yeah, just kidding.

Part II: The Case for TypeScript

- Enhanced Developer Productivity
- Improved Code Quality
- Seamless Integration with JavaScript
- Strong Community and Ecosystem
- Statistics

Part II: Enhanced Developer Productivity

- **Static Typing**: Catch errors during development instead of runtime, leading to quicker debugging.
- **Intelligent Code Completion**: Improved auto-completion and suggestions in IDEs streamline coding.
- **Refactoring Support**: Easier to make changes across large codebases with confidence.
- **Better Documentation**: Type annotations serve as self-documenting code, improving clarity.

Part II: Improved Code Quality

- **Type Safety**: Reduces the likelihood of bugs by enforcing data types.
- **Clearer APIs**: Function signatures and interfaces define clear contracts, making collaboration easier.
- **Maintainability**: Strong typing leads to more maintainable code over time, especially in large teams.
- **Enhanced Testing**: Types provide a foundation for more effective unit tests and validations.

Part II: Seamless Integration with JavaScript

- **Superset of JavaScript**: All existing JavaScript code is valid TypeScript, easing the transition.
- **Gradual Adoption**: Teams can adopt TypeScript incrementally in existing projects.
- **Rich Ecosystem**: Compatible with popular libraries and frameworks, ensuring versatility.
- **JavaScript Compatibility**: Outputs clean, standard JavaScript, ensuring compatibility with all browsers.
- **Defect Density:** Code written in TypeScript can have 15% fewer bugs on average compared to plain JavaScript

Part II: Strong Community and Ecosystem

- **Vibrant Community**: A large and active community contributes to resources, libraries, and support.
- **Regular Updates**: Frequent releases add new features and improvements based on community feedback.
- **Wide Adoption**: Many major companies and frameworks utilize TypeScript, indicating its reliability.
- **Educational Resources**: Abundant tutorials, documentation, and courses available to facilitate learning.

Part II: Statistics

- **Most popular programming language**: TypeScript ranked 5th in the Stack Overflow 2023 Developer Survey
- **Rapid growth**: Usage grew from 21% in 2019 to over 34% in 2023
- **GitHub repository activity**: As of 2023, TypeScript has over 95,000 stars on GitHub, making it one of the top open-source projects.
- **NPM downloads**: TypeScript sees over 90 million downloads per week on NPM, reflecting its wide adoption across the JavaScript ecosystem.

Part III: Drawing Parallels

- Naming conventions
- File Locations
- Compiling / Building
- Classes Compared

Part III: Naming Conventions

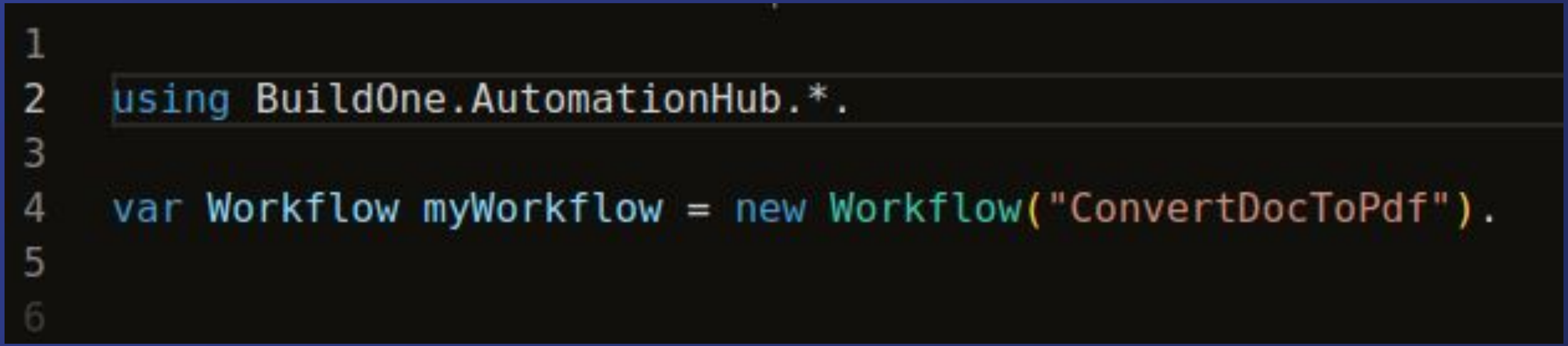
- ABL forces the source code filename to be the same as the classname
- Typescript class name can be anything
- Convention is class names are PascalCase
- Multiple classes per source code file

Part III: Where is the code stored - ABL

- Each class is defined in a single .cls file
  - Compiles to a .r file

- The Folder structure is the “namespace”
  - **BuildOne/AutomationHub/Workflow.cls** creates the class package **BuildOne.AutomationHub.Workflow**

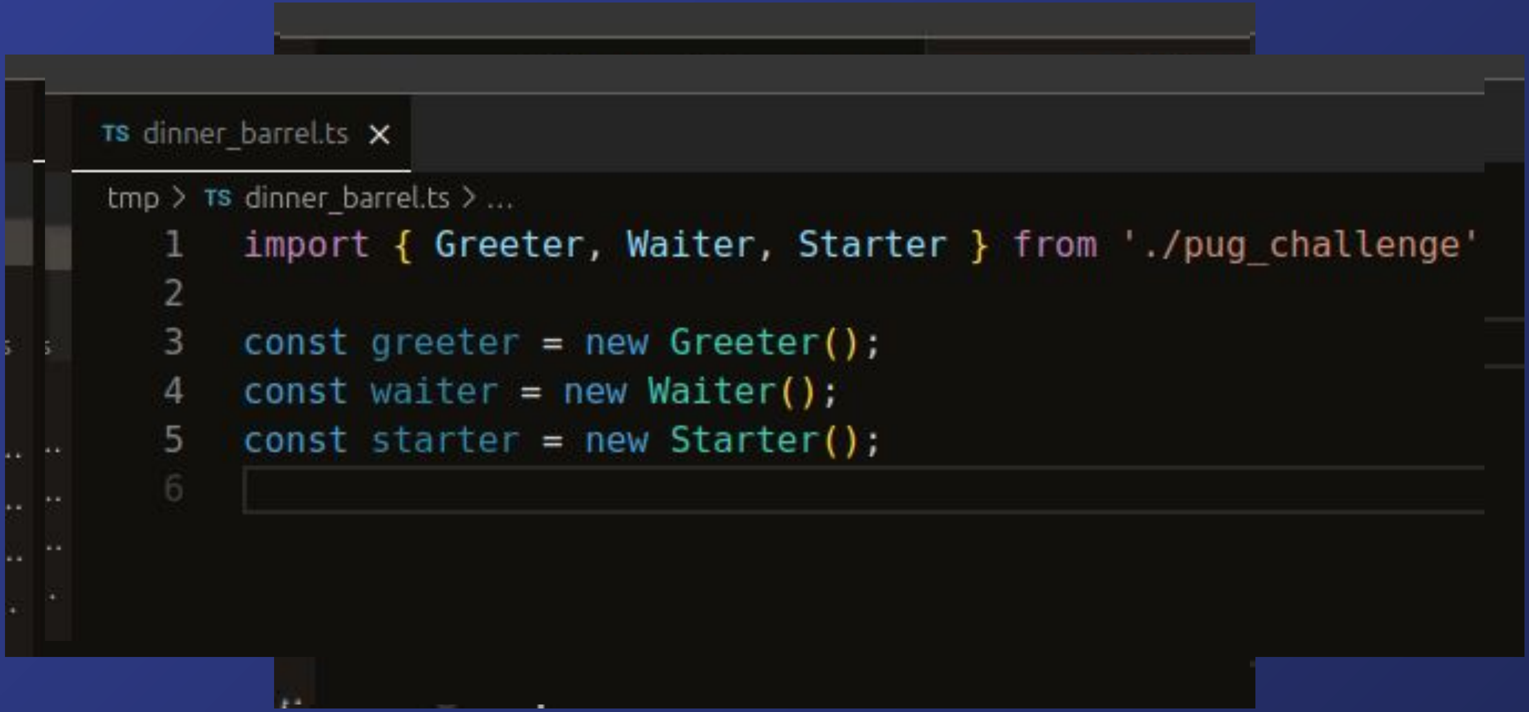
- The Using statement allows for shorthand references



```

using BuildOne.AutomationHub.*.

var Workflow myWorkflow = new Workflow("ConvertDocToPdf").


```

Part III: Where is the code stored - Typescript

- One or more classes can be defined in a single file
- Classes are exported for use by other code
- Barrel files can be created to re-export



```typescript
import { Greeter, Waiter, Starter } from './pug_challenge'

const greeter = new Greeter();
const waiter = new Waiter();
const starter = new Starter();

```

Part III: Compiling code

- ABL
  - Compiles using Progress compiler
  - One cls => one .r
  - Can bundle all .r into a procedure library (.pl)

- Typescript
  - “Compiles” with tsc / babel
  - Produces a js file for each ts
  - When building for production, all .js files can be bundled into a single .js
  - Tree-Shaking

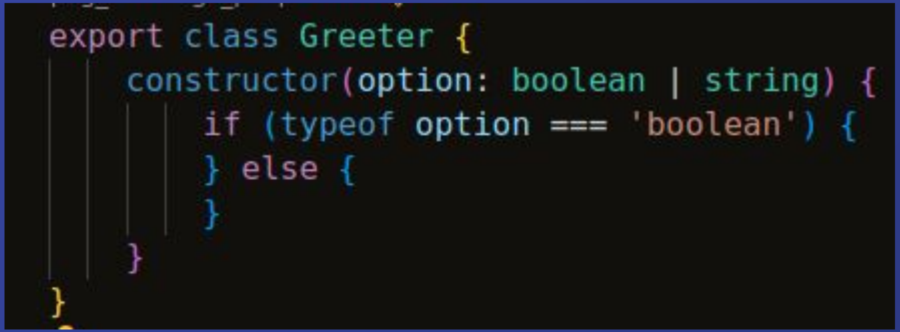
Part III: Classes Compared

- Constructors
- Destructors
- Methods
- Properties
- Inheritance
- Abstracts
- Statics

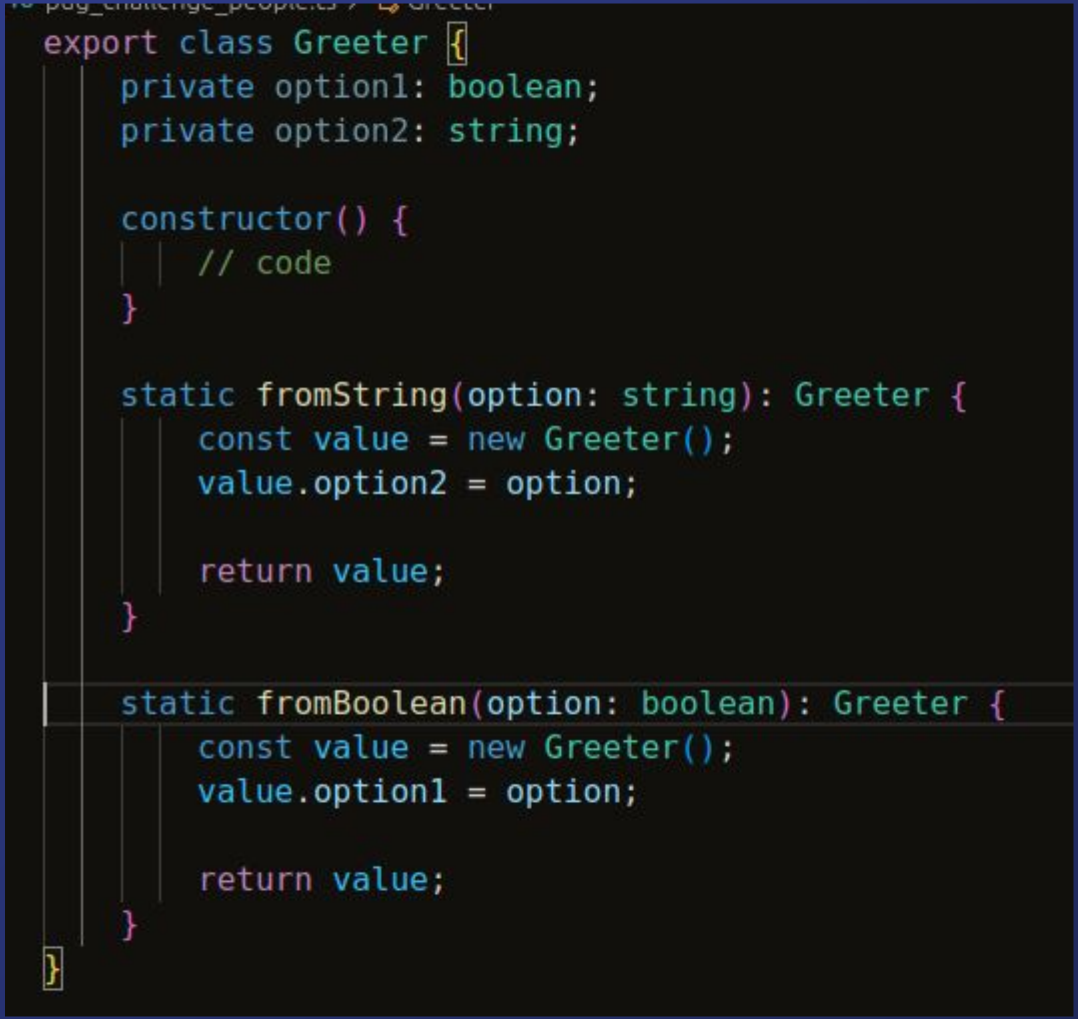
Part III: Constructors

- Typescript has a single constructor only.
- Overloading is not possible, but some workarounds
  - Union parameters
  - Factory methods

Part III: Constructor workaround



```
export class Greeter {
    constructor(option: boolean | string) {
        if (typeof option === 'boolean') {
        } else {
        }
    }
}
```



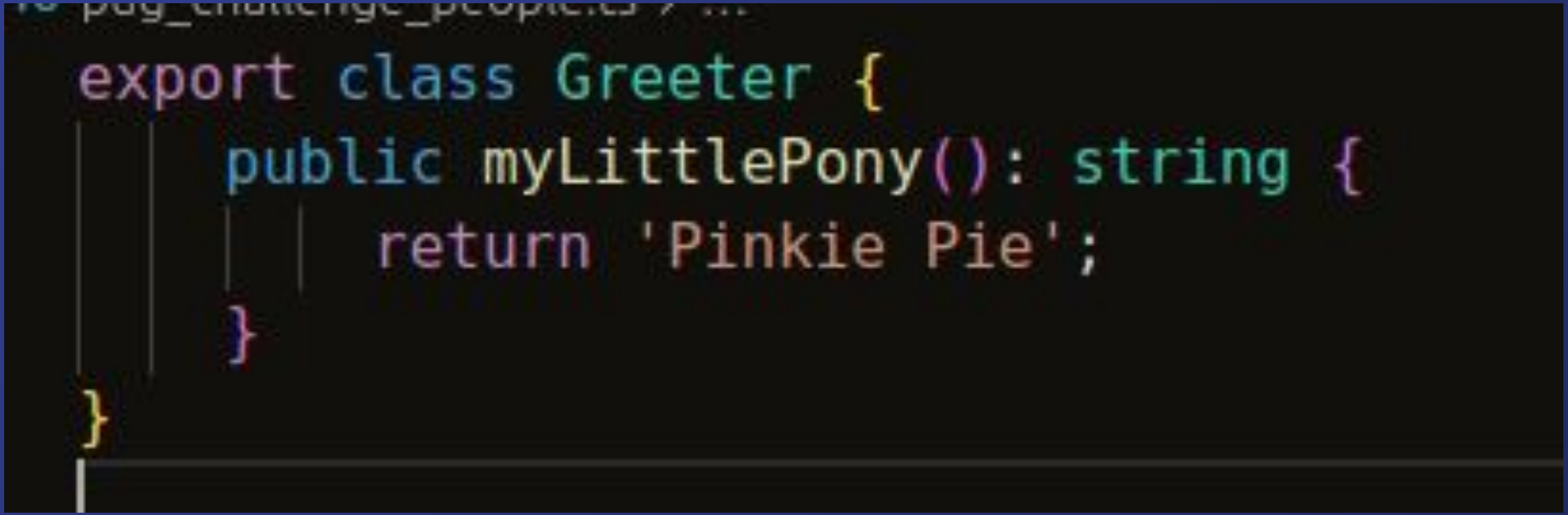
```
export class Greeter {
    private option1: boolean;
    private option2: string;

    constructor() {
        // code
    }

    static fromString(option: string): Greeter {
        const value = new Greeter();
        value.option2 = option;

        return value;
    }

    static fromBoolean(option: boolean): Greeter {
        const value = new Greeter();
        value.option1 = option;

        return value;
    }
}
```

Part III: Destructors

- No such thing
- Nothing to see here. Move on.
- No, there’s not a finally either
- Well, maybe there is a destructor-type thing
  - Even the docs say “avoid where possible” so let’s not talk about that child

Part III: Methods

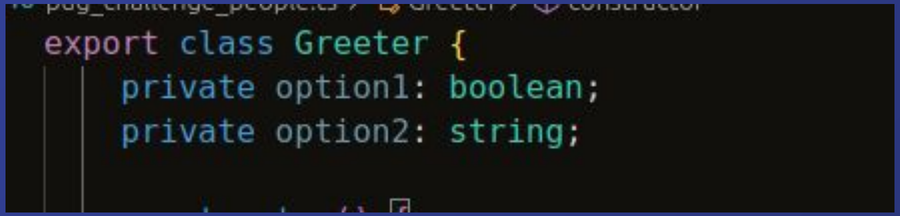
- Similar to the ABL
- Private, public, protected, static, abstract
- No overloading



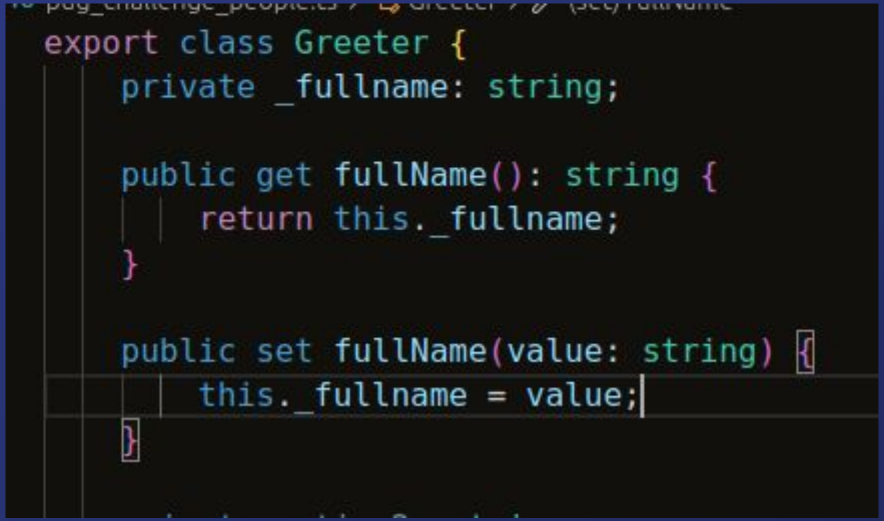
```
export class Greeter {
    public myLittlePony(): string {
        return 'Pinkie Pie';
    }
}
```

Part III: Properties

- Can be defined simply as a var on the class
- Also has getters and setters
- Public, private, static, abstract



```
export class Greeter {
    private option1: boolean;
    private option2: string;
```



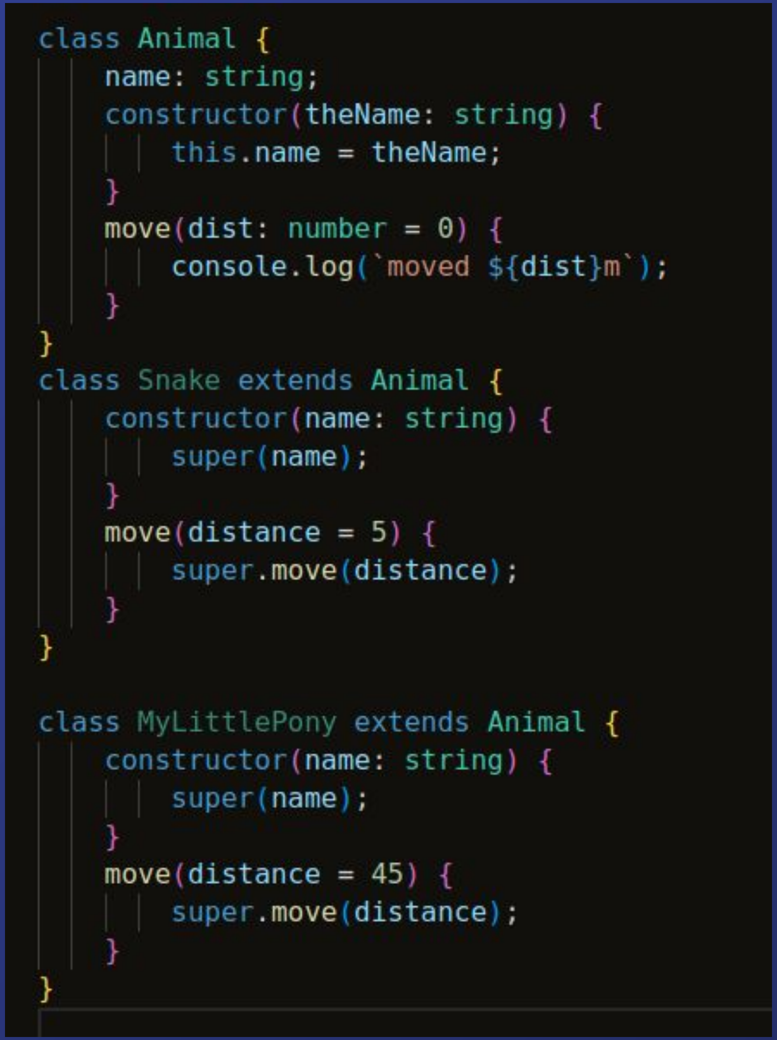
```
export class Greeter {
    private _fullname: string;

    public get fullName(): string {
        return this._fullname;
    }

    public set fullName(value: string) {
        this._fullname = value;
    }
```

Part III: Inheritance

- Like the ABL, Typescript can extend (inherit) other classes

- Multiple levels
  - One level deep only. Change my mind
  - “As a general guideline, more than 2 or 3 levels of inheritance is often seen as a red flag in software design.”

Part III: Inheritance example

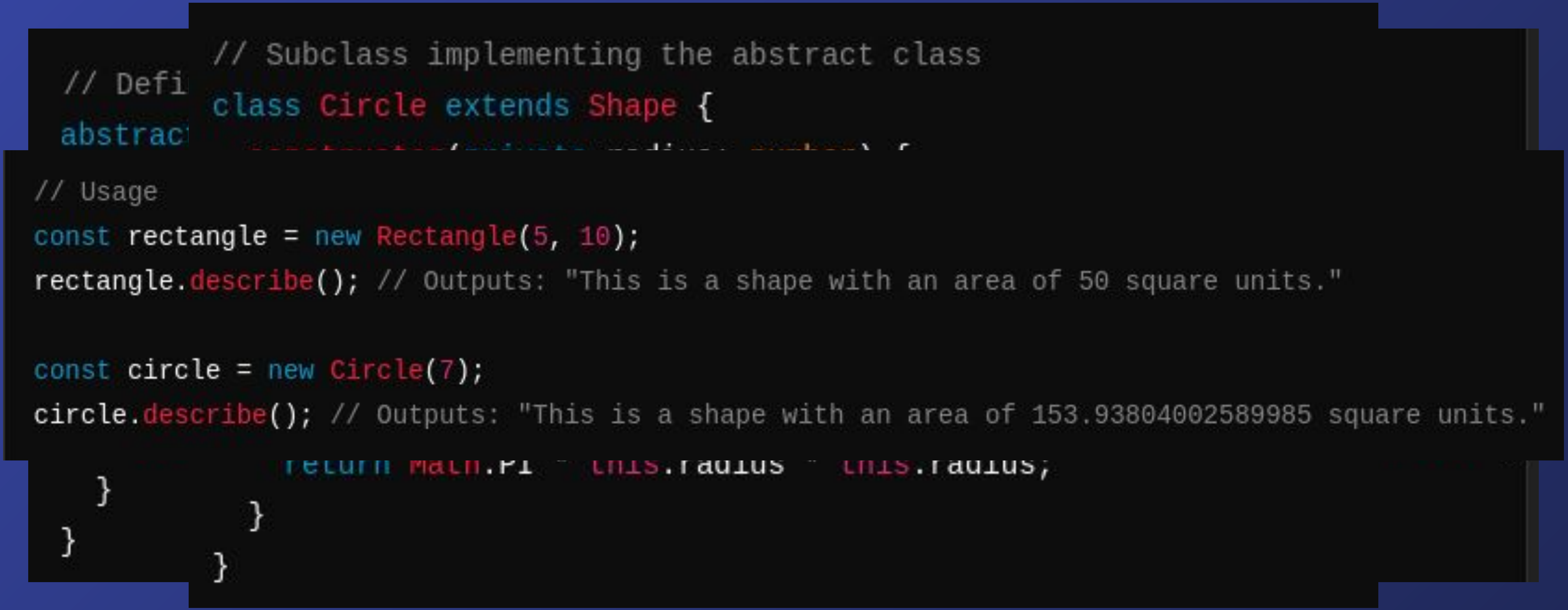


```
class Animal {
    name: string;
    constructor(theName: string) {
        this.name = theName;
    }
    move(dist: number = 0) {
        console.log(`moved ${dist}m`);
    }
}
class Snake extends Animal {
    constructor(name: string) {
        super(name);
    }
    move(distance = 5) {
        super.move(distance);
    }
}

class MyLittlePony extends Animal {
    constructor(name: string) {
        super(name);
    }
    move(distance = 45) {
        super.move(distance);
    }
}
```

Part III: Abstracts

- Similar to the ABL
- Abstract class cannot be instantiated
- Needs to be inherited
- All properties and methods are available to subClass

Part III: Abstract Classes



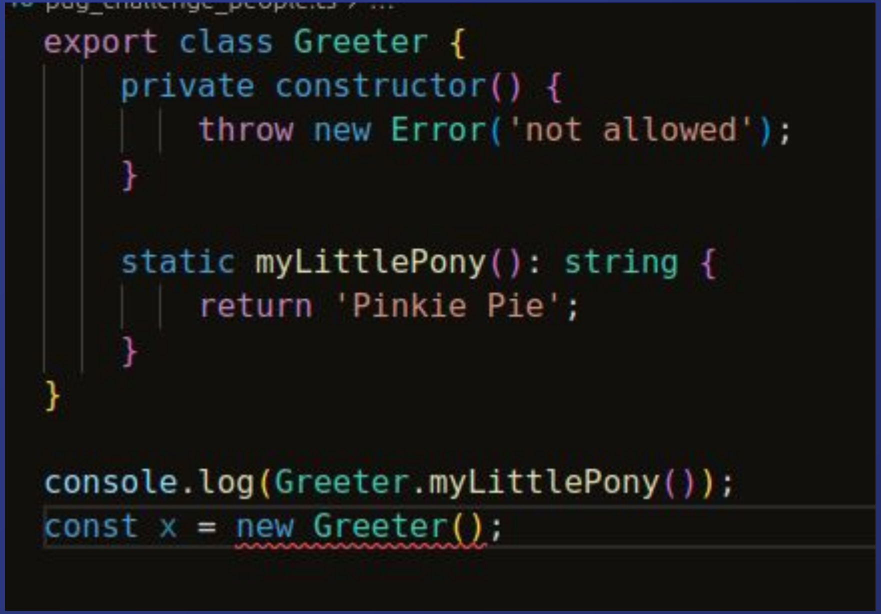
```
// Defi
abstrac

  }
}
```

```
// Subclass implementing the abstract class
class Circle extends Shape {
  constructor(private radius: number) {

    return Math.PI * this.radius * this.radius;
  }
}
```

```
// Usage
const rectangle = new Rectangle(5, 10);
rectangle.describe(); // Outputs: "This is a shape with an area of 50 square units."

const circle = new Circle(7);
circle.describe(); // Outputs: "This is a shape with an area of 153.93804002589985 square units."
```

Part III: Statics

- Properties and methods can be
- Typescript class itself cannot be static
  - You can simulate a static class by creating a private constructor



```
export class Greeter {
    private constructor() {
        throw new Error('not allowed');
    }

    static myLittlePony(): string {
        return 'Pinkie Pie';
    }
}

console.log(Greeter.myLittlePony());
const x = new Greeter();
```

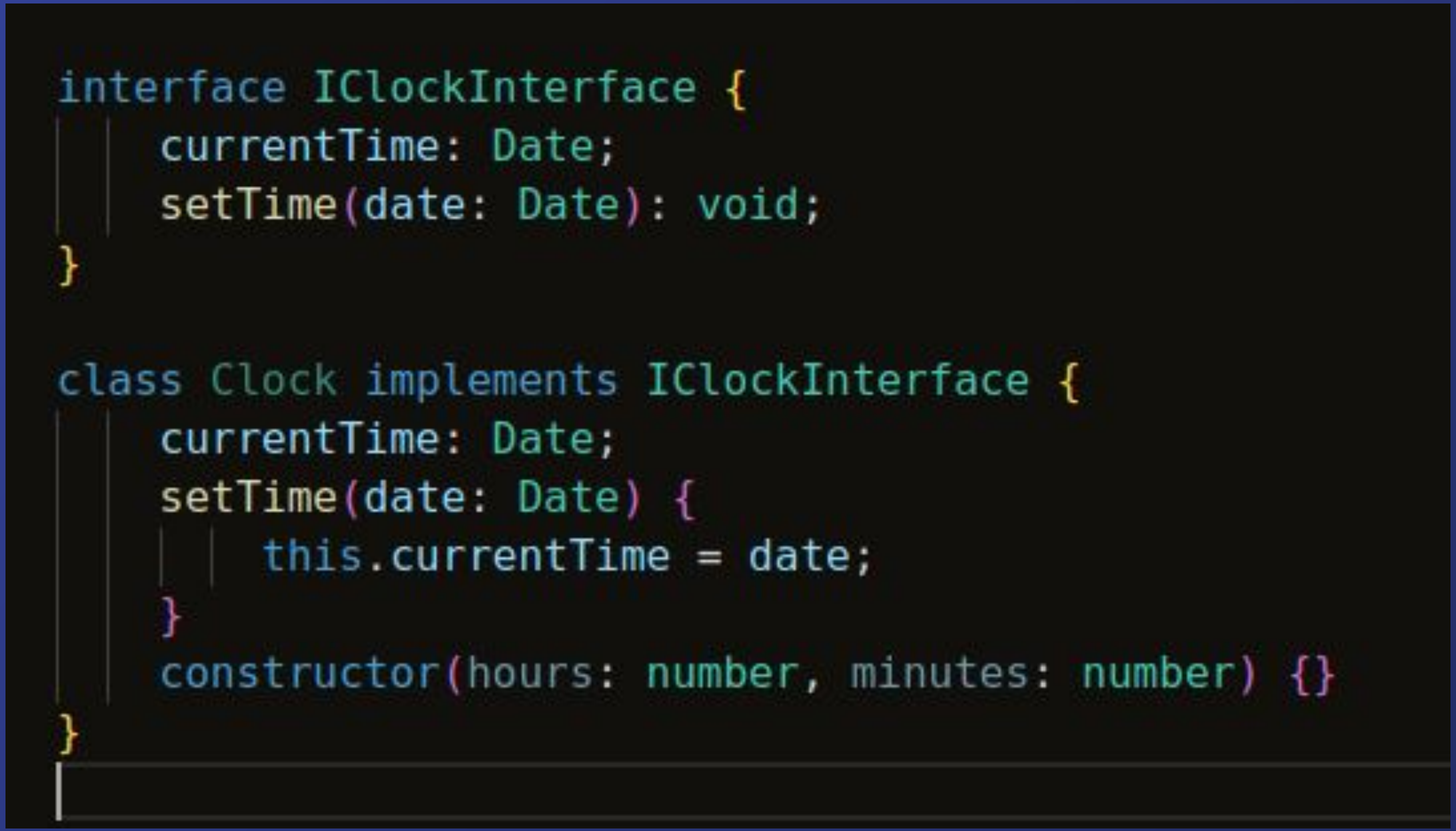
Part III: Class events

- There aren’t any
- Can use standard emitters which are a pub / sub
- Similar to SUBSCRIBE TO "MyEvent" ANYWHERE.

Part III: Interfaces

- Similar to the ABL
- Specify a set of properties and methods that a class must implement
- Build several classes that conform to a standard API
- Each class must implement all properties and methods in the interface
- Naming Convention is that Interfaces begin with an “I”

Part III: Interface example



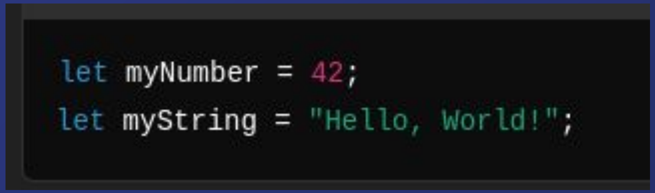
```
interface IClockInterface {
    currentTime: Date;
    setTime(date: Date): void;
}

class Clock implements IClockInterface {
    currentTime: Date;
    setTime(date: Date) {
        this.currentTime = date;
    }
    constructor(hours: number, minutes: number) {}
}
```

Part IV: My Typescript voyage

- Lessons learnt moving to Typescript

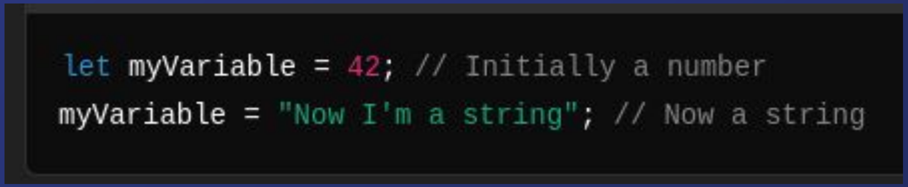
Part IV: Javascript vs ABL

- Implicit typing
  - vars are declared without specifying a type



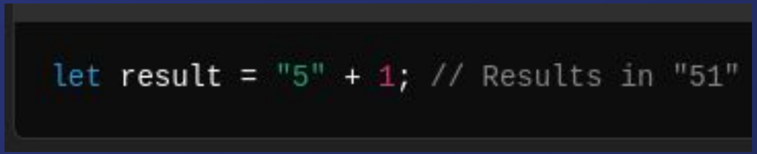
```
let myNumber = 42;
let myString = "Hello, World!";
```

- Dynamic typing
  - the type of a variable can change at runtime.



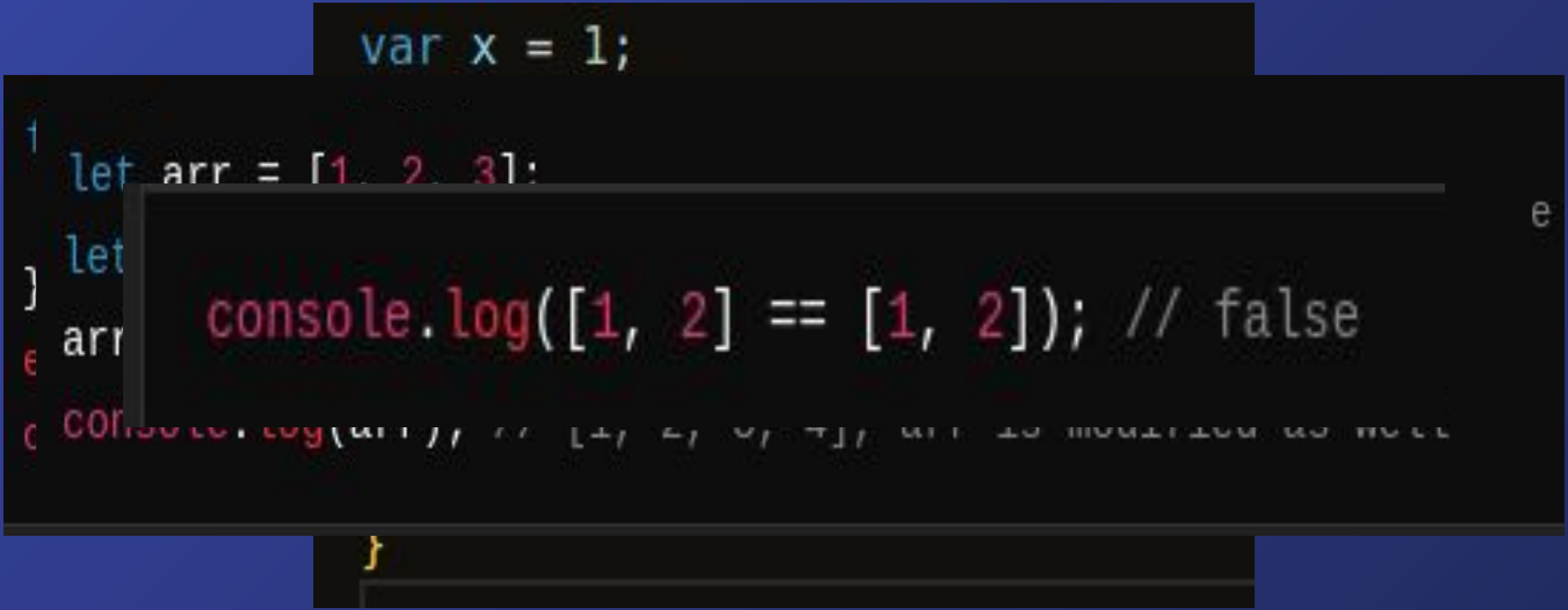
```
 let myVariable = 42; // Initially a number
myVariable = "Now I'm a string"; // Now a string
```

- Weak Typing
  - performs implicit type conversions (type coercion).



```
let result = "5" + 1; // Results in "51"
```

Part IV : some Gotchas



```
var x = 1;
```

```
f
  let arr = [1, 2, 3];
  let
}
  arr
e
c console.log(arr); // [1, 2, 3, 4]; arr is modified as well
```

```
console.log([1, 2] == [1, 2]); // false
```

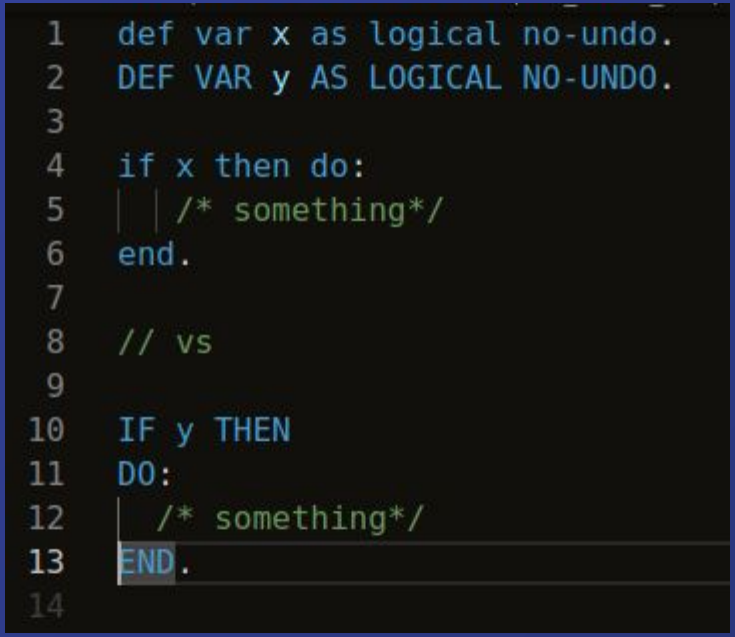
```
}
```

Part V: Disclaimer / Warnings

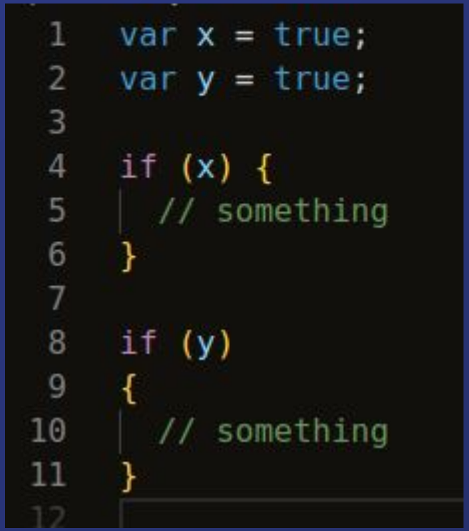
- Yes, yes. Disclaimers must be at the top of the presentation deck
    - This *is* about Typescript ;)
- You will not be a TS wizard at the end of the presentation
- Code formatting in slides is not … optimal ..

Part V: Code formatting

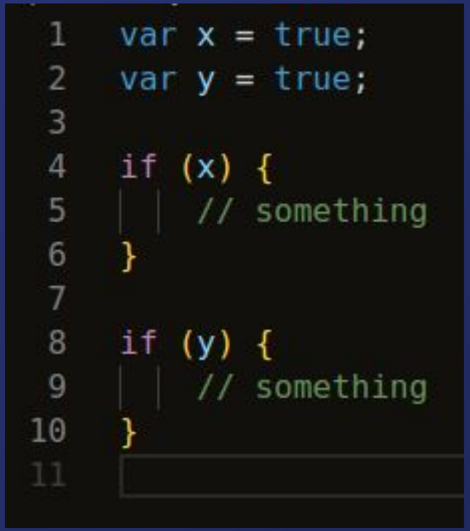
- Everyone is going to be offended
  - Tough, live it it
- There are two types of people
  - Programmers will know



```
def var x as logical no-undo.
DEF VAR y AS LOGICAL NO-UNDO.

if x then do:
    /* something*/
end.

// vs

IF y THEN
DO:
  /* something*/
END.

```

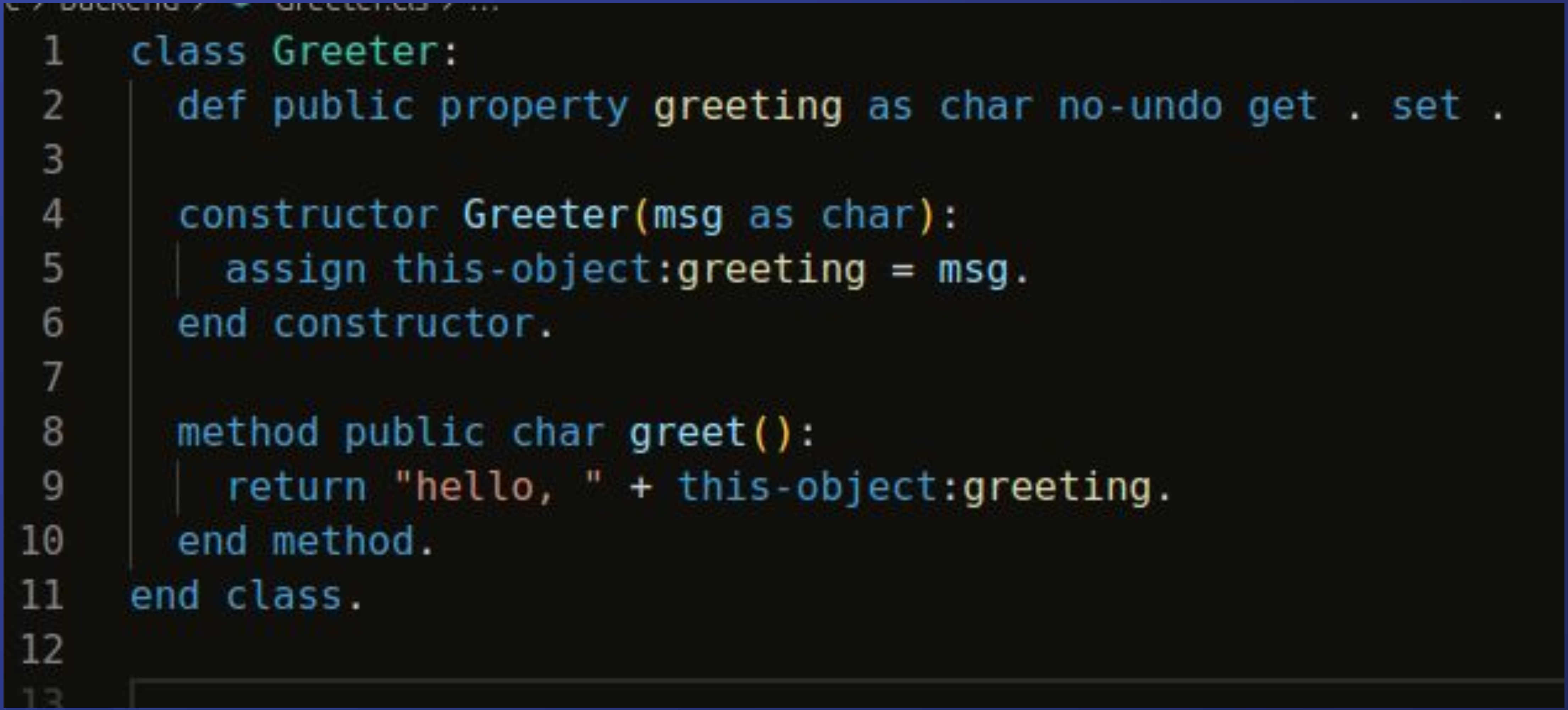


```
var x = true;
var y = true;

if (x) {
  // something
}

if (y)
{
  // something
}
```



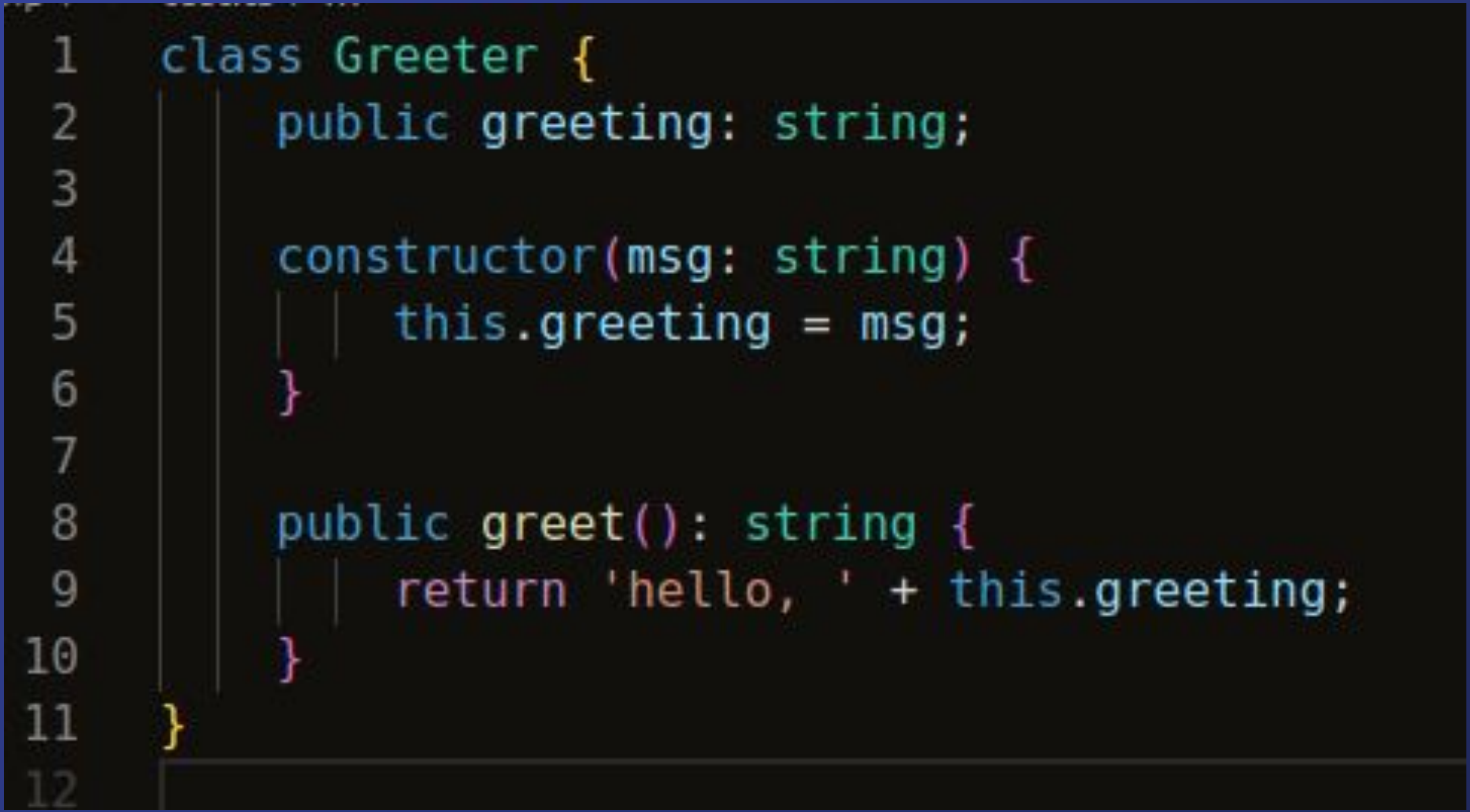
```
var x = true;
var y = true;

if (x) {
    // something
}

if (y) {
    // something
}

```

Part V: Show me the code (ABL) !



```
class Greeter:
  def public property greeting as char no-undo get . set .

  constructor Greeter(msg as char):
    assign this-object:greeting = msg.
  end constructor.

  method public char greet():
    return "hello, " + this-object:greeting.
  end method.
end class.

```

Part V: Show me the code (Typescript) !



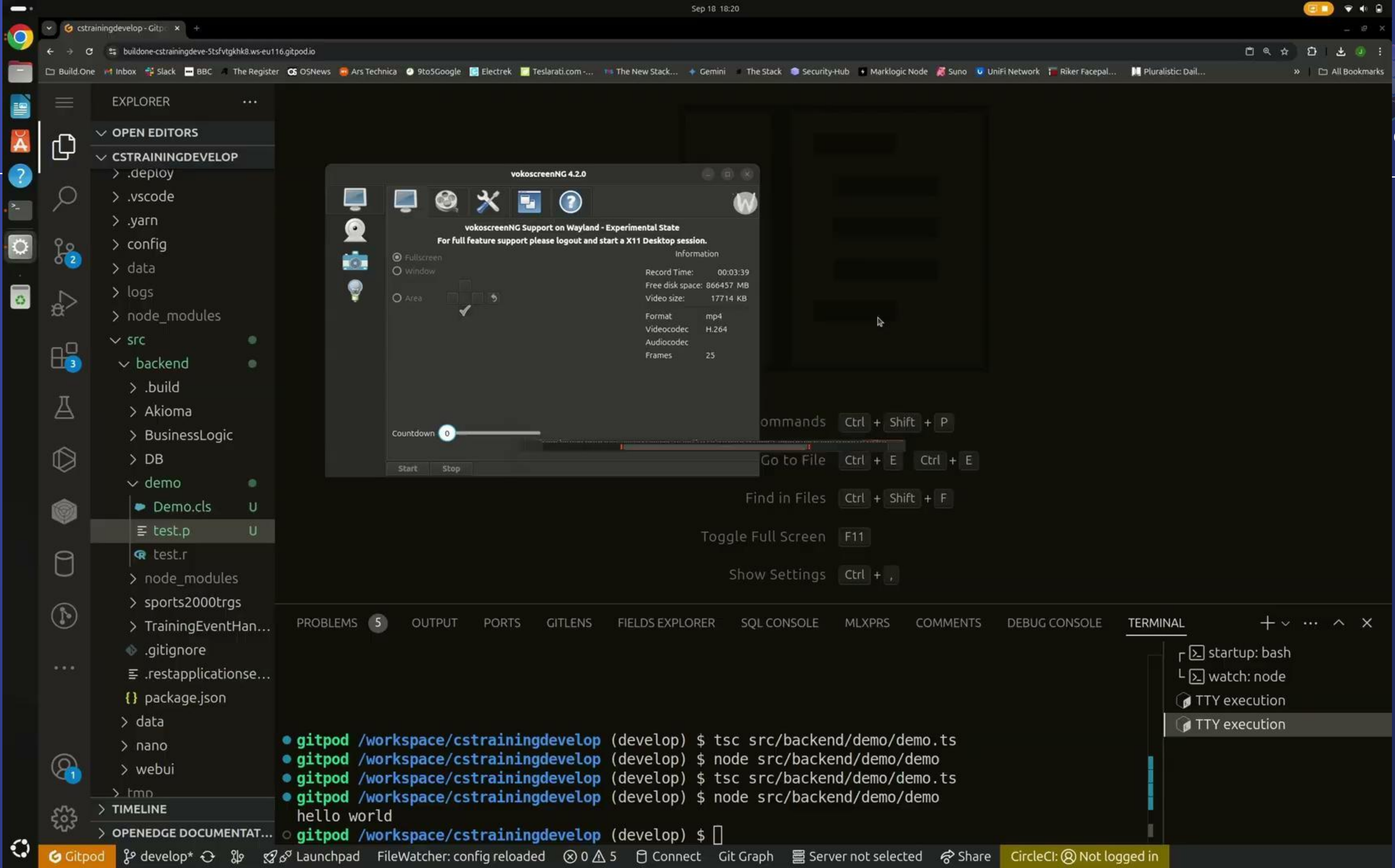
```typescript
class Greeter {
    public greeting: string;

    constructor(msg: string) {
        this.greeting = msg;
    }

    public greet(): string {
        return 'hello, ' + this.greeting;
    }
}

```

Part V: Installing node and typescript

#1 the right way

#2 the wrong way

Part V: converting ABL



vokoscreenNG Support on Wayland - Experimental State
For full feature support please logout and start a X11 Desktop session.

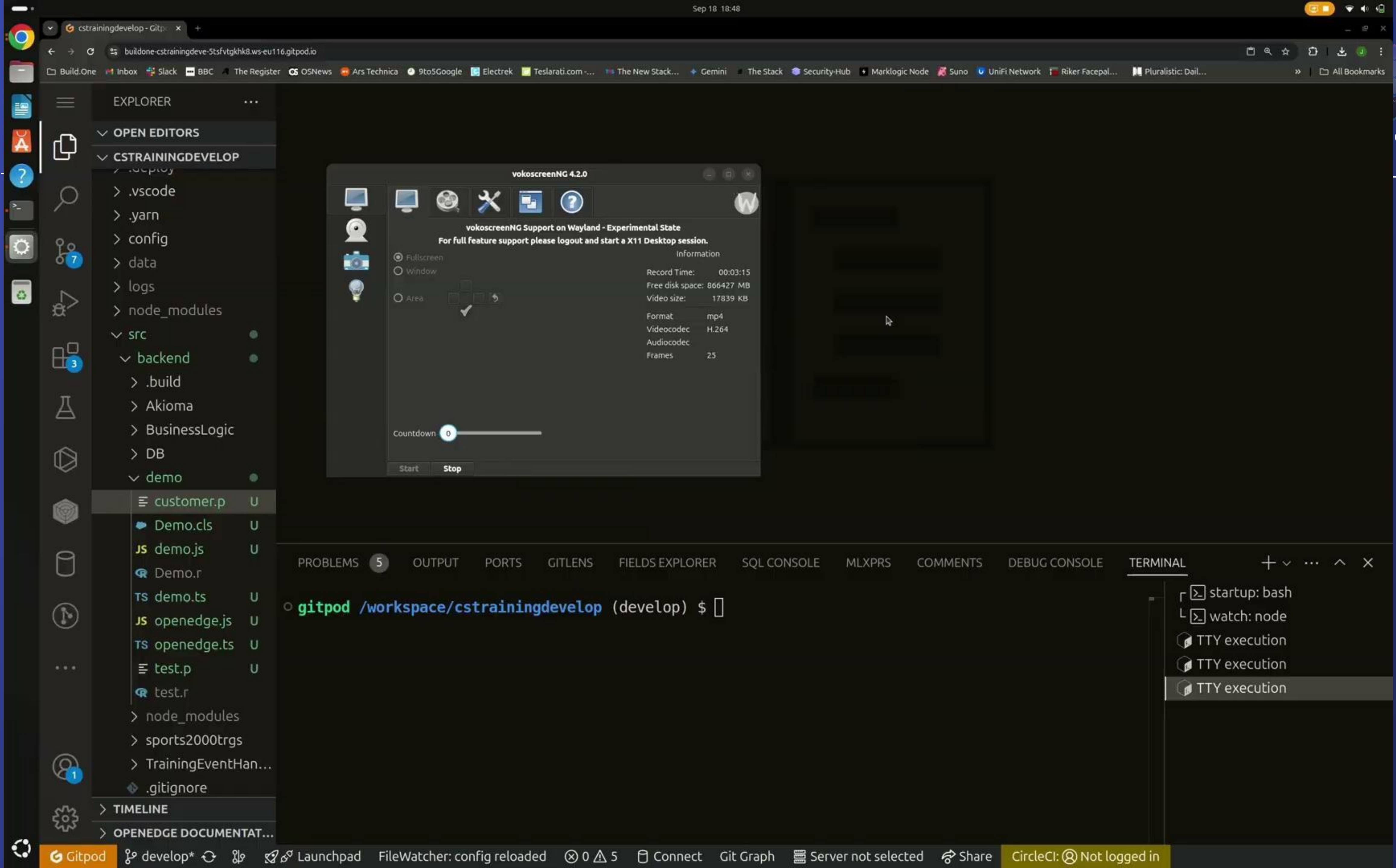
Information

Record Time: 00:03:39
Free disk space: 866457 MB
Video size: 17714 KB

Format mp4
Videocodec H.264
Audiocodec
Frames 25

```
● gitpod /workspace/cstrainingdevelop (develop) $ tsc src/backend/demo/demo.ts
● gitpod /workspace/cstrainingdevelop (develop) $ node src/backend/demo/demo
● gitpod /workspace/cstrainingdevelop (develop) $ tsc src/backend/demo/demo.ts
● gitpod /workspace/cstrainingdevelop (develop) $ node src/backend/demo/demo
hello world
○ gitpod /workspace/cstrainingdevelop (develop) $ 
```

Part V: calling ABL



vokoscreenNG Support on Wayland - Experimental State
For full feature support please logout and start a X11 Desktop session.

Information

Record Time: 00:03:15
Free disk space: 866427 MB
Video size: 17839 KB

Format mp4
Videocodec H.264
Audiocodec
Frames 25

```
gitpod /workspace/cstrainingdevelop (develop) $ 
```

Wrapup

In this presentation we have been on a journey

- TypeScript Unveiled: A Quick Overview
- The Case for TypeScript: Benefits and Opportunities
- Drawing Parallels: ABL and TypeScript Compared
- My Typescript Voyage
- Getting Started with TypeScript: Essential First Steps

Wrapup

Questions

BUILD.ONE