Typescript

```
npm install -g typescript
```

saludar.ts

```
function saludar(persona) {
    return "Hola, " + persona;
}

let user = “Pablo López";

document.body.innerHTML = saludar(user);
```

```
tsc saludar.ts
```

saludar.ts

```
function saludar(persona: string) {
    return "Hola, " + persona;
}

let user = “Pablo López";

document.body.innerHTML = saludar(user);
```

POO

Interfaces

```
interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person: Person) {
    return "Hello, " + person.firstName + " " +
person.lastName;
}

let user = { firstName: "Jane", lastName: "User" };

document.body.innerHTML = greeter(user);
```

Clases

```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string,
public lastName: string) {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person : Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = new Student("Jane", "M.", "User");

document.body.innerHTML = greeter(user);
```

```
tsc saludar.ts
```

saludar.html

```
<!DOCTYPE html>
<html>
    <head><title>Primer día de curso Angular</title></head>
    <body>
        <script src="saludar.js"></script>
    </body>
</html>
```

Javascript → Typescript

TIPOS BÁSICOS

Boolean

```
let isDone: boolean = false;
```

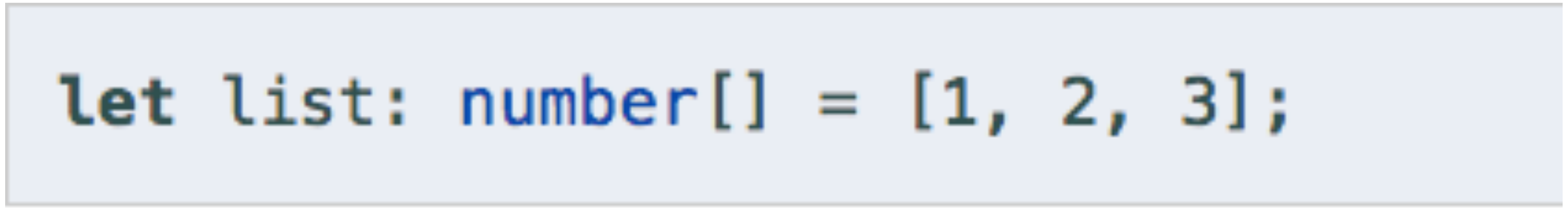
Number

```
let decimal: number = 6;
let hex: number = 0xf00d;
let binary: number = 0b1010;
let octal: number = 0o744;
```

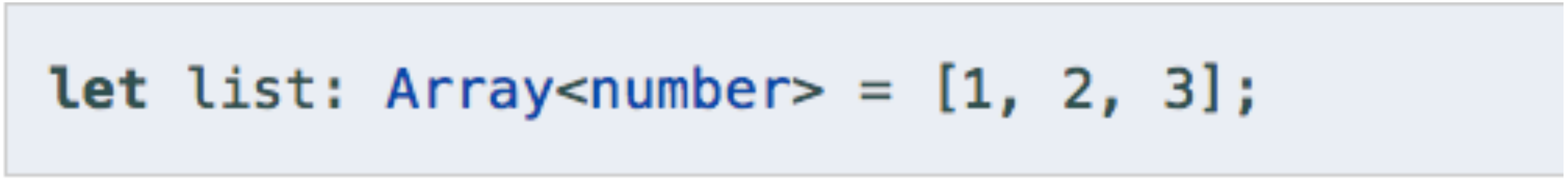
String

```
let color: string = "blue";
color = 'red';
```

Array

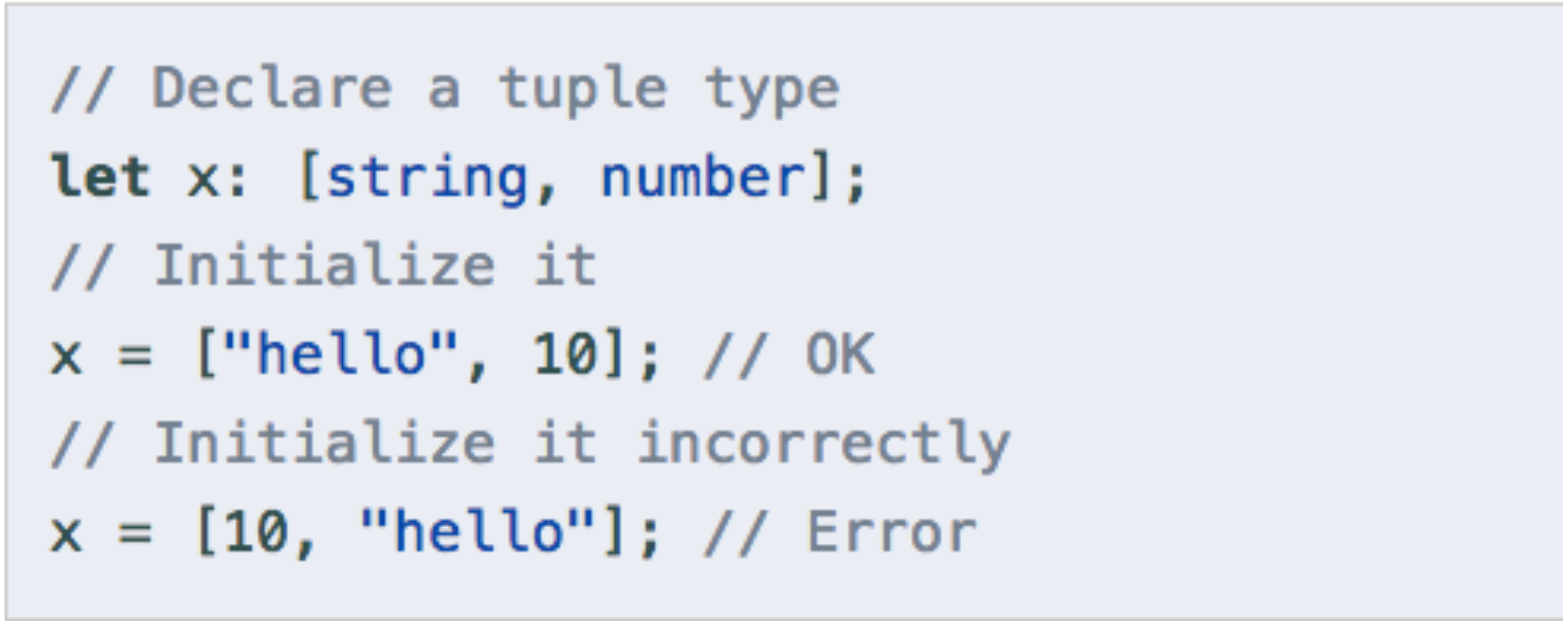


```
let list: number[] = [1, 2, 3];
```



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

Tupla



```
// Declare a tuple type
let x: [string, number];
// Initialize it
x = ["hello", 10]; // OK
// Initialize it incorrectly
x = [10, "hello"]; // Error
```

Enum

```
enum Color {Red, Green, Blue}
let c: Color = Color.Green;
```

```
enum Color {Red = 1, Green = 2, Blue = 4}
let c: Color = Color.Green;
```

```
enum Color {Red = 1, Green, Blue}
let colorName: string = Color[2];

alert(colorName); // Displays 'Green' as it's value is 2 above
```

Any

```
let notSure: any = 4;
notSure = "maybe a string instead";
notSure = false; // okay, definitely a boolean
```

```
let list: any[] = [1, true, "free"];

list[1] = 100;
```

Void

```
function warnUser(): void {
    alert("This is my warning message");
}
```

null & defined

```
// Not much else we can assign to these variables!
let u: undefined = undefined;
let n: null = null;
```

DECLARACION VARIABLES

var

```
var a = 10;
```

let

```
let hello = "Hello!";
```

const

```
const numLivesForCat = 9;
```

POO

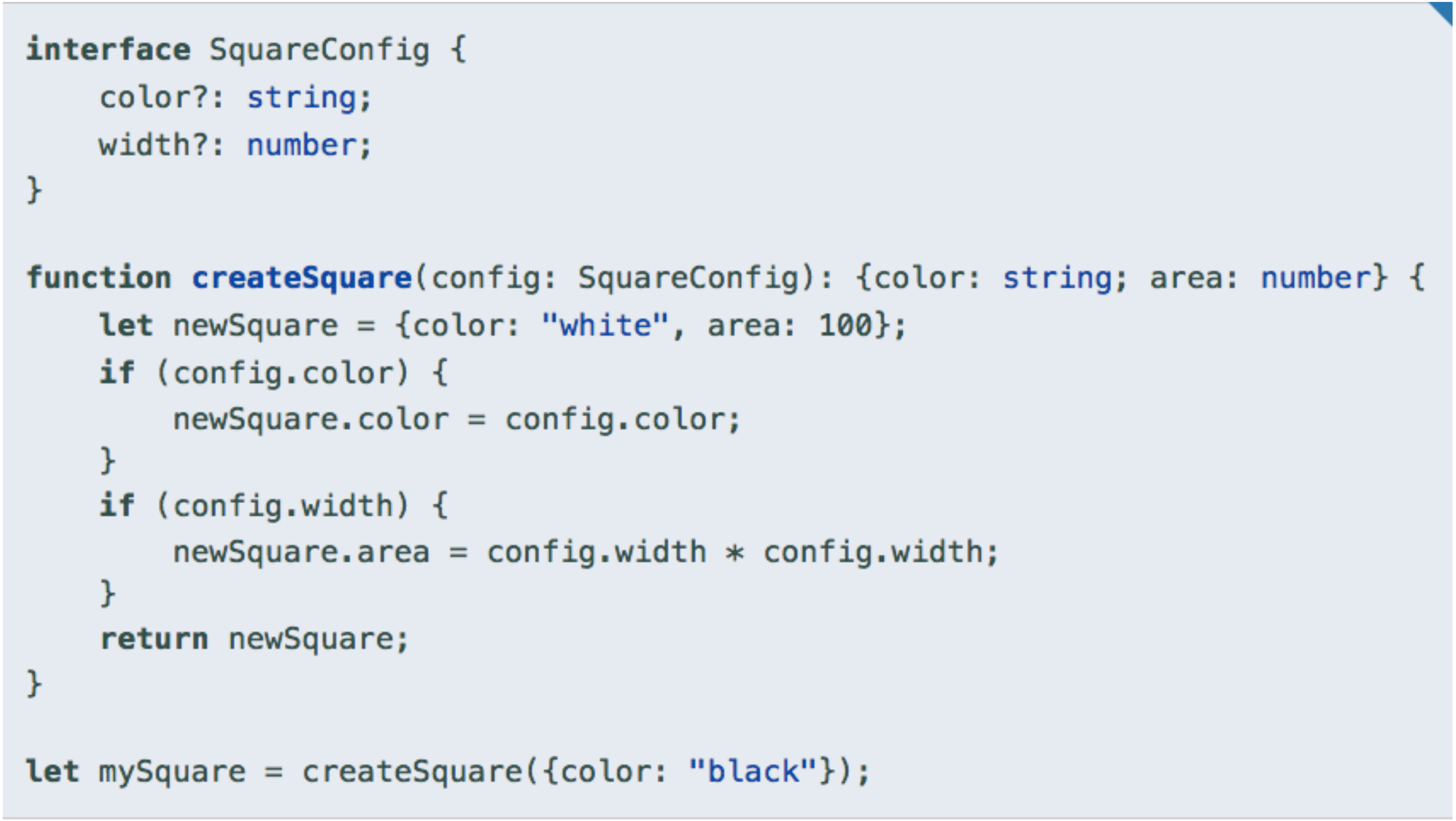
Interfaces

1

```
function printLabel(labelledObj: { label: string }) {
    console.log(labelledObj.label);
}

let myObj = {size: 10, label: "Size 10 Object"};
printLabel(myObj);
```

2

```
interface LabelledValue {
    label: string;
}

function printLabel(labelledObj: LabelledValue) {
    console.log(labelledObj.label);
}

let myObj = {size: 10, label: "Size 10 Object"};
printLabel(myObj);
```



```
interface SquareConfig {
    color?: string;
    width?: number;
}

function createSquare(config: SquareConfig): {color: string; area: number} {
    let newSquare = {color: "white", area: 100};
    if (config.color) {
        newSquare.color = config.color;
    }
    if (config.width) {
        newSquare.area = config.width * config.width;
    }
    return newSquare;
}

let mySquare = createSquare({color: "black"});
```

```
interface ClockInterface {
    currentTime: Date;
    setTime(d: Date);
}

class Clock implements ClockInterface {
    currentTime: Date;
    setTime(d: Date) {
        this.currentTime = d;
    }
    constructor(h: number, m: number) { }
}
```

```typescript
interface Shape {
    color: string;
}

interface Square extends Shape {
    sideLength: number;
}

let square = <Square>{};
square.color = "blue";
square.sideLength = 10;
```

```
interface Shape {
    color: string;
}

interface PenStroke {
    penWidth: number;
}

interface Square extends Shape, PenStroke {
    sideLength: number;
}

let square = <Square>{};
square.color = "blue";
square.sideLength = 10;
square.penWidth = 5.0;
```

Clases

```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");
```

```
class Animal {
    move(distanceInMeters: number = 0) {
        console.log(`Animal moved ${distanceInMeters}m.`);
    }
}

class Dog extends Animal {
    bark() {
        console.log('Woof! Woof!');
    }
}

const dog = new Dog();
dog.bark();
dog.move(10);
dog.bark();
```

```
class Animal {
    name: string;
    constructor(theName: string) { this.name = theName; }
    move(distanceInMeters: number = 0) {
        console.log(`${this.name} moved ${distanceInMeters}m.`);
    }
}

class Snake extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 5) {
        console.log("Slithering...");
        super.move(distanceInMeters);
    }
}

class Horse extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 45) {
        console.log("Galloping...");
        super.move(distanceInMeters);
    }
}
```

```typescript
let sam = new Snake("Sammy the Python");
let tom: Animal = new Horse("Tommy the Palomino");

sam.move();
tom.move(34);
```

Modificadores de acceso

PUBLIC

PROTECTED

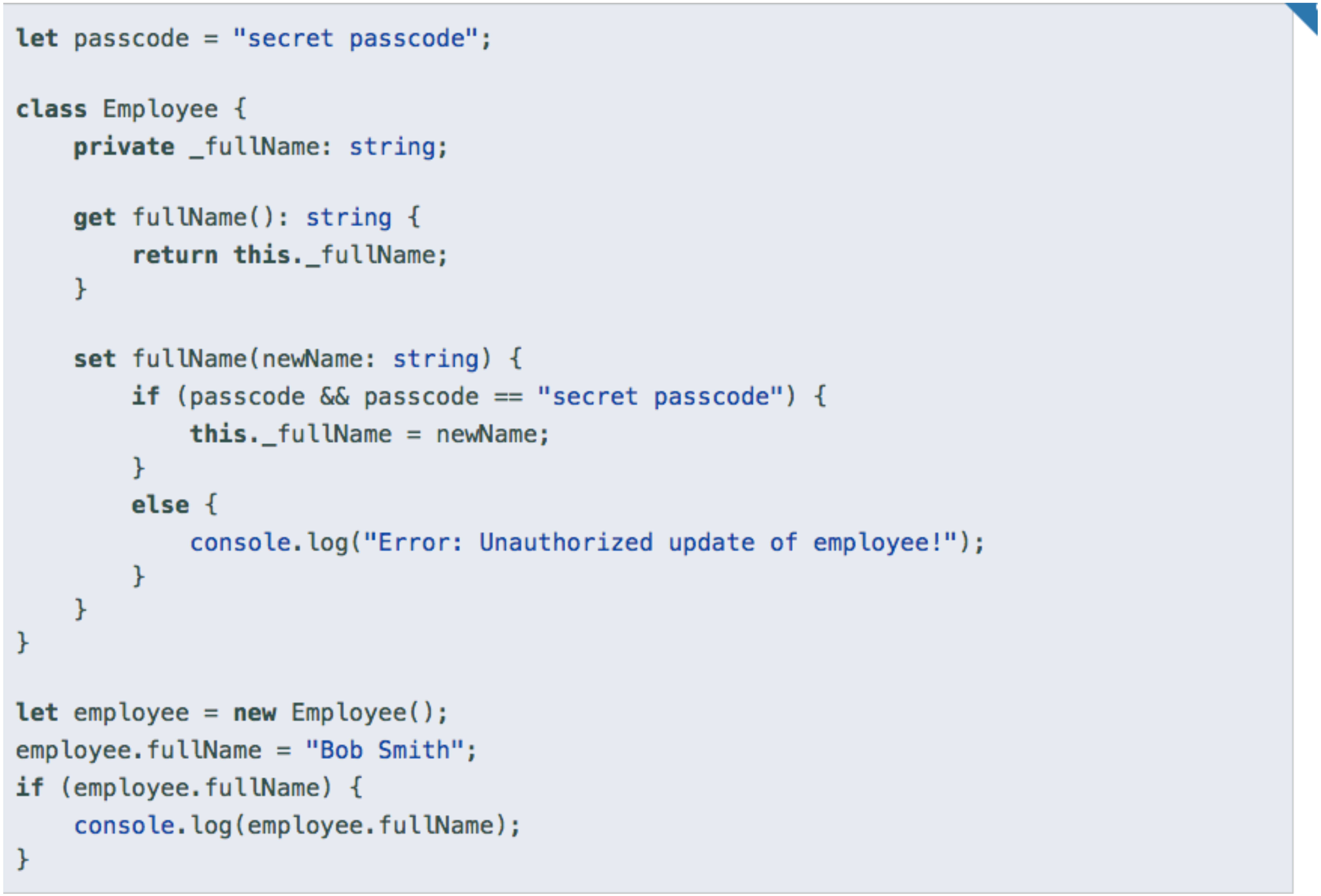
PRIVATE

```
class Animal {
    public name: string;
    public constructor(theName: string) { this.name = theName; }
    public move(distanceInMeters: number) {
        console.log(`${this.name} moved ${distanceInMeters}m.`);
    }
}
```

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

new Animal("Cat").name; // Error: 'name' is private;
```

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

class Employee extends Person {
    private department: string;

    constructor(name: string, department: string) {
        super(name);
        this.department = department;
    }

    public getElevatorPitch() {
        return `Hello, my name is ${this.name} and I work in ${this.department}.`;
    }
}

let howard = new Employee("Howard", "Sales");
console.log(howard.getElevatorPitch());
console.log(howard.name); // error
```

Getters / Setters



```
let passcode = "secret passcode";

class Employee {
    private _fullName: string;

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

    set fullName(newName: string) {
        if (passcode && passcode == "secret passcode") {
            this._fullName = newName;
        }
        else {
            console.log("Error: Unauthorized update of employee!");
        }
    }
}

let employee = new Employee();
employee.fullName = "Bob Smith";
if (employee.fullName) {
    console.log(employee.fullName);
}
```

Clases Abstractas

```
abstract class Department {

    constructor(public name: string) {
    }

    printName(): void {
        console.log("Department name: " + this.name);
    }

    abstract printMeeting(): void; // must be implemented in derived classes
}
```

```
class AccountingDepartment extends Department {

    constructor() {
        super("Accounting and Auditing"); // constructors in derived classes must call
 super()
    }

    printMeeting(): void {
        console.log("The Accounting Department meets each Monday at 10am.");
    }

    generateReports(): void {
        console.log("Generating accounting reports...");
    }
}
```

```
let department: Department; // ok to create a reference to an abstract type
department = new Department(); // error: cannot create an instance of an abstract clas
s
department = new AccountingDepartment(); // ok to create and assign a non-abstract sub
class
department.printName();
department.printMeeting();
department.generateReports(); // error: method doesn't exist on declared abstract type
```

Iteradores

```
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"
}
```

Módulos

ZipCodeValidator.ts

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

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

```
import { ZipCodeValidator } from "./ZipCodeValidator";

let myValidator = new ZipCodeValidator();
```