Introduction to TypeScript

By Guillaume Gérard

What is it?

- Born in 2012 by Anders Hejlsberg (Turbo Pascal, J++, C#/.NET and TypeScript…)
- Made by Microsoft
- Code transpiler like Babel
- Add some POO languages (Java, C#) concepts over JavaScript
- Add reliability to your code (type checking at compile time)
- Help to split code with namespaces and modules
- Embed modern JavaScript featured : let, const, arrow function, template string, classes…
- Last version: TypeScript 3.2

Get TypeScript

Node.js	Main editors	And More...
`npm install -g typescript`	• Atom	• Eclipse
Init config	• Visual Studio Code	• WebStorm
`tsc --init`	• Sublime Text	• Visual Studio 2017
Compile	• Vim	• Visual Studio 2015
`tsc demo.ts`	• Emacs	

Source: https://www.typescriptlang.org/index.html#download-links

Types

- boolean
- number
- string
- Array: []
- Tuple: [string, number]
- enum {}
- void
- object
- any
- Null
- undefined
- never
- Type assertion/cast: <string>var

```typescript
let isDone: boolean = false;
let decimal: number = 6;
let color: string = "blue";
let list: number[] = [1, 2, 3];
let x: [string, number] = ["hello", 10];
console.log(x[0].substr(1)); // OK
console.log(x[1].substr(1)); // Error, 'number' does not have 'substr'

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

let notSure: any = 4;
notSure = "maybe a string instead"; // OK
notSure = false; // OK

function warnUser(): void {
  console.log("This is my warning message");
}

function error(message: string): never {
  throw new Error(message);
}

let create = function(o: object | null): void {
  console.log(`create with ${o}`);
}
create({ prop: 0 }); // OK
create(null); // OK
create(42); // Error
create("string"); // Error
create(false); // Error
create(undefined); // Error

let someValue: any = "this is a string";
let strLength: number = (<string>someValue).length;
let strLength2: number = (someValue as string).length;
```

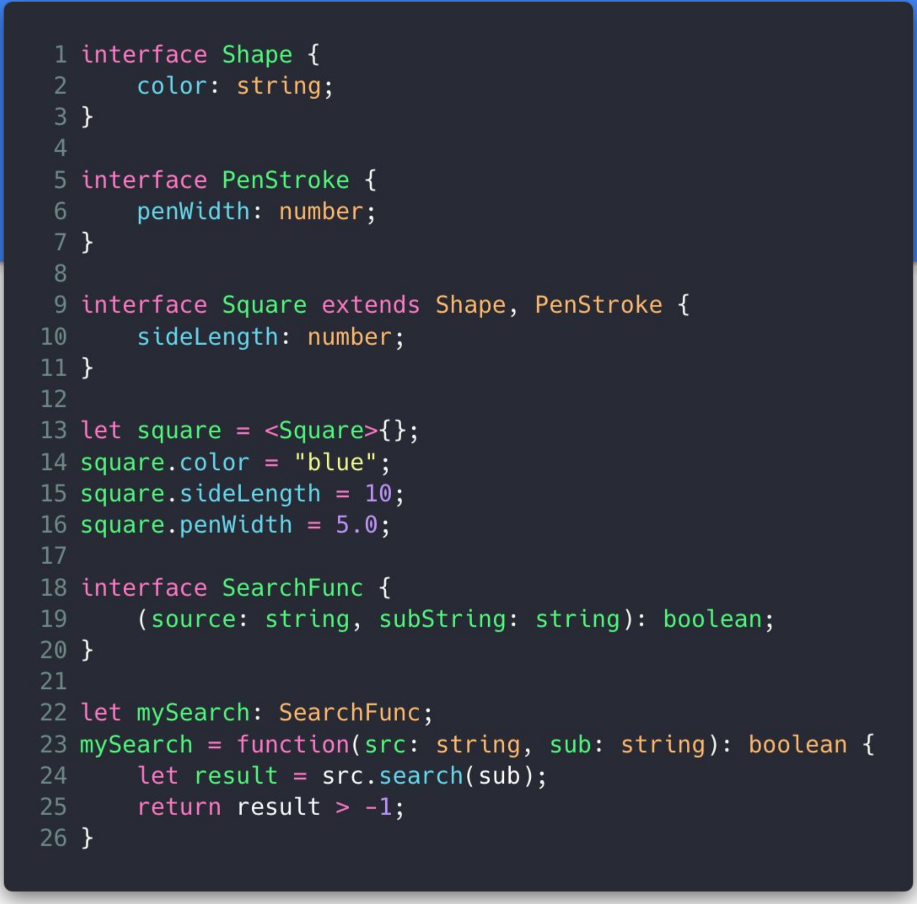
Classes

- Classes (from ES6) are Types
- Inheritance
- `public`, `private`, `protected`, `readonly` modifiers
- Accessors: `get`, `set`
- `static`: fixed values shared over instance
- Abstract class (interface with code)

```typescript
abstract class Animal {
  static planet: string = "earth";
  readonly name: string;
  private _type: string | undefined;
  abstract makeSound(): void;
  public constructor(pName: string) {
    this.name = pName;
    this._type = undefined;
  }
  get type(): string | undefined {
    return this._type;
  }
  set type(pType: string | undefined) {
    this._type = pType;
  }
  move(): void {
    console.log(`roaming the ${Animal.planet}...`);
  }
}

class Lion extends Animal {
  makeSound() {
    console.log(`${this.name} says "ROAR!"`);
  }
}

let lion = new Lion("Léo");
lion.makeSound();
```

Interfaces

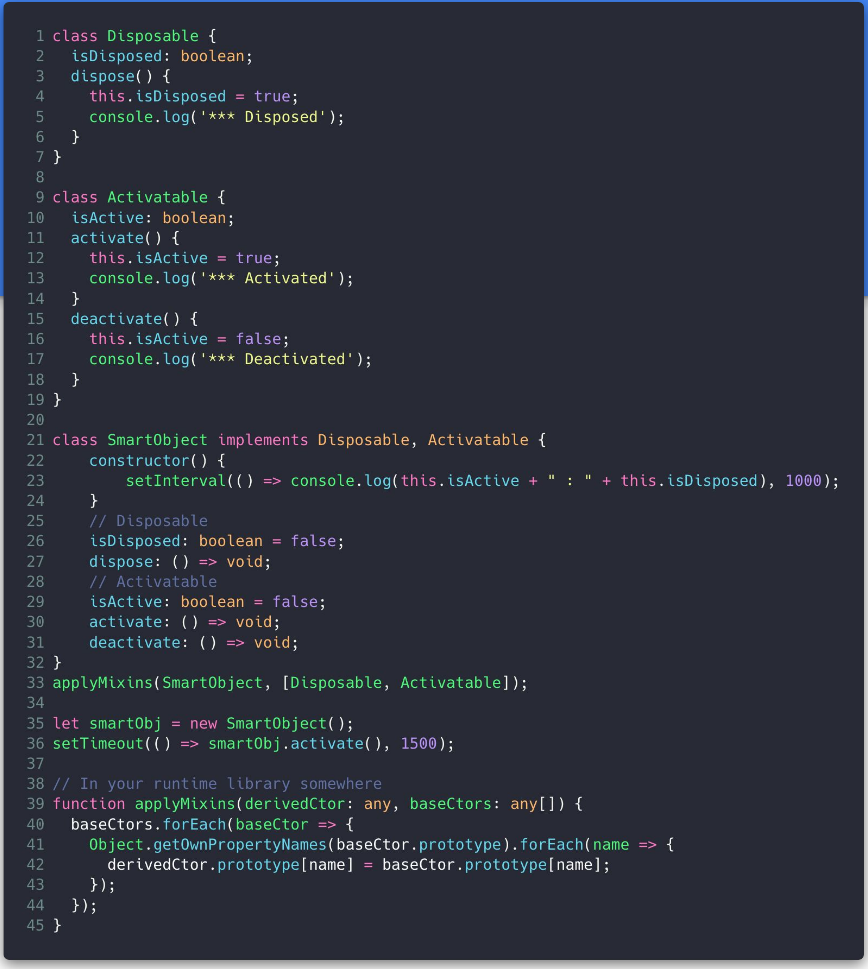
- Like contracts
- Types
- Inheritance (multiple)
- Classes Types
- Functions Types



```typescript
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;

interface SearchFunc {
    (source: string, subString: string): boolean;
}

let mySearch: SearchFunc;
mySearch = function(src: string, sub: string): boolean {
    let result = src.search(sub);
    return result > -1;
}
```

Mixins

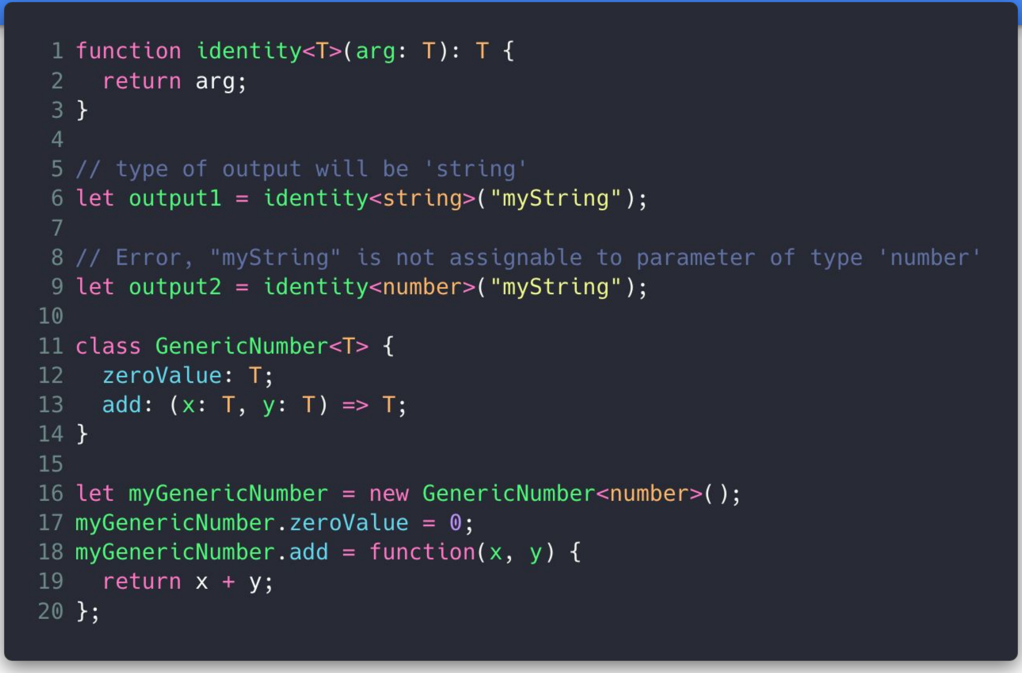
- Reuse components
- `implements` keyword: treats the classes as interfaces, and only uses the types behind Disposable and Activatable rather than the implementation
- `applyMixins` helper: run through the properties of each of the mixins and copy them over to the target, filling out the stand-in properties with their implementations



```
class Disposable {
  isDisposed: boolean;
  dispose() {
    this.isDisposed = true;
    console.log('*** Disposed');
  }
}

class Activatable {
  isActive: boolean;
  activate() {
    this.isActive = true;
    console.log('*** Activated');
  }
  deactivate() {
    this.isActive = false;
    console.log('*** Deactivated');
  }
}

class SmartObject implements Disposable, Activatable {
    constructor() {
        setInterval(() => console.log(this.isActive + " : " + this.isDisposed), 1000);
    }
    // Disposable
    isDisposed: boolean = false;
    dispose: () => void;
    // Activatable
    isActive: boolean = false;
    activate: () => void;
    deactivate: () => void;
}
applyMixins(SmartObject, [Disposable, Activatable]);

let smartObj = new SmartObject();
setTimeout(() => smartObj.activate(), 1500);

// In your runtime library somewhere
function applyMixins(derivedCtor: any, baseCtors: any[]) {
  baseCtors.forEach(baseCtor => {
    Object.getOwnPropertyNames(baseCtor.prototype).forEach(name => {
      derivedCtor.prototype[name] = baseCtor.prototype[name];
    });
  });
}
```

Generics

- Make reusable code with generic type definition (example <T>)
- There is also Type which handle generic like `Array<T>`
- You can create your own class/interface to handle generic

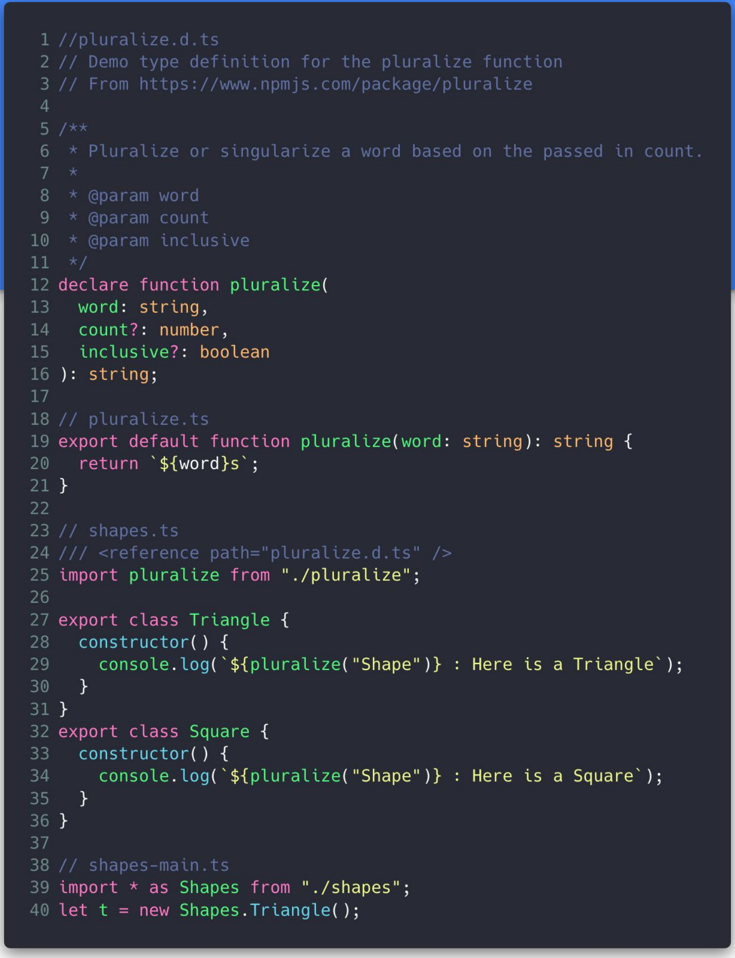


```typescript
function identity<T>(arg: T): T {
  return arg;
}

// type of output will be 'string'
let output1 = identity<string>("myString");

// Error, "myString" is not assignable to parameter of type 'number'
let output2 = identity<number>("myString");

class GenericNumber<T> {
  zeroValue: T;
  add: (x: T, y: T) => T;
}

let myGenericNumber = new GenericNumber<number>();
myGenericNumber.zeroValue = 0;
myGenericNumber.add = function(x, y) {
  return x + y;
};
```

Namespaces and Modules

- ES6 import and export
- AMD and CommonJS Compatibility
- Namespaces are named objects in the global namespace

- Modules contains both code and declaration
- Modules are used to create npm TS package
- https://github.com/DefinitelyTyped/DefinitelyTyped

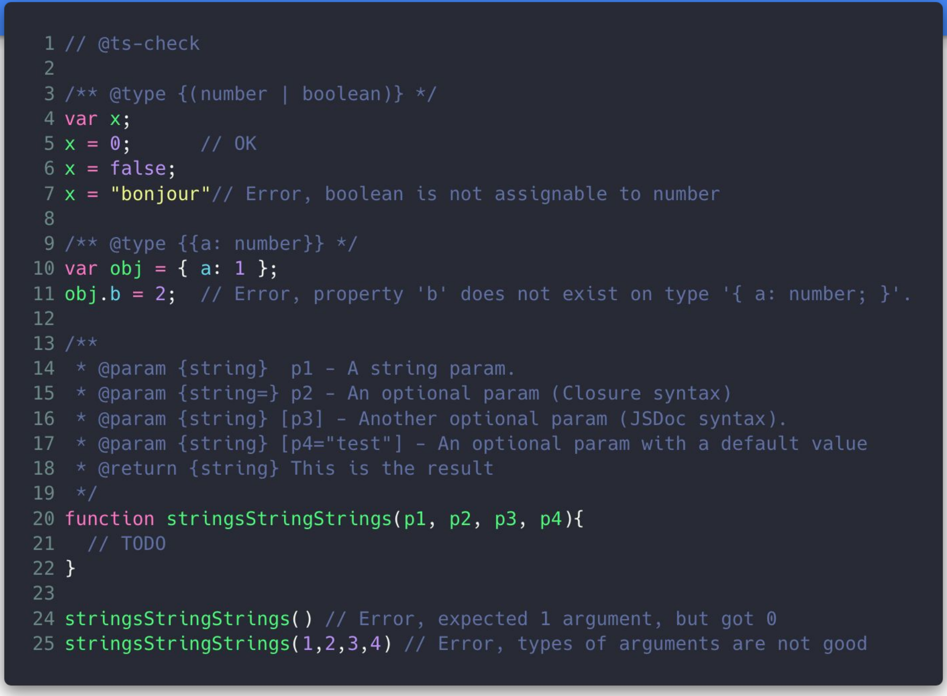


```
//pluralize.d.ts
// Demo type definition for the pluralize function
// From https://www.npmjs.com/package/pluralize

/**
 * Pluralize or singularize a word based on the passed in count.
 *
 * @param word
 * @param count
 * @param inclusive
 */
declare function pluralize(
  word: string,
  count?: number,
  inclusive?: boolean
): string;

// pluralize.ts
export default function pluralize(word: string): string {
  return `${word}s`;
}

// shapes.ts
/// <reference path="pluralize.d.ts" />
import pluralize from "./pluralize";

export class Triangle {
  constructor() {
    console.log(`${pluralize("Shape")} : Here is a Triangle`);
  }
}
export class Square {
  constructor() {
    console.log(`${pluralize("Shape")} : Here is a Square`);
  }
}

// shapes-main.ts
import * as Shapes from "./shapes";
let t = new Shapes.Triangle();
```

Type Checking JavaScript Files

- Supports JSDoc in JS file
- Ignore one line `// @ts-ignore`
- Ignore a file `// @ts-nocheck`
- Supported keywords:

  @type
  @param (or @arg or @argument)
  @returns (or @return)
  @typedef
  @callback
  @template
  @class (or @constructor)
  @this
  @extends (or @augments)
  @enum



```javascript
// @ts-check

/** @type {(number | boolean)} */
var x;
x = 0;       // OK
x = false;
x = "bonjour"// Error, boolean is not assignable to number

/** @type {{a: number}} */
var obj = { a: 1 };
obj.b = 2;  // Error, property 'b' does not exist on type '{ a: number; }'.

/**
 * @param {string}  p1 - A string param.
 * @param {string=} p2 - An optional param (Closure syntax)
 * @param {string} [p3] - Another optional param (JSDoc syntax).
 * @param {string} [p4="test"] - An optional param with a default value
 * @return {string} This is the result
 */
function stringsStringStrings(p1, p2, p3, p4){
  // TODO
}

stringsStringStrings() // Error, expected 1 argument, but got 0
stringsStringStrings(1,2,3,4) // Error, types of arguments are not good
```

Resources

- https://github.com/GreatWizard/typescript-introduction
- https://www.typescriptlang.org/docs/home.html
- https://github.com/Microsoft/TypeScript-Handbook
- https://blog.mariusschulz.com/series/typescript-evolution