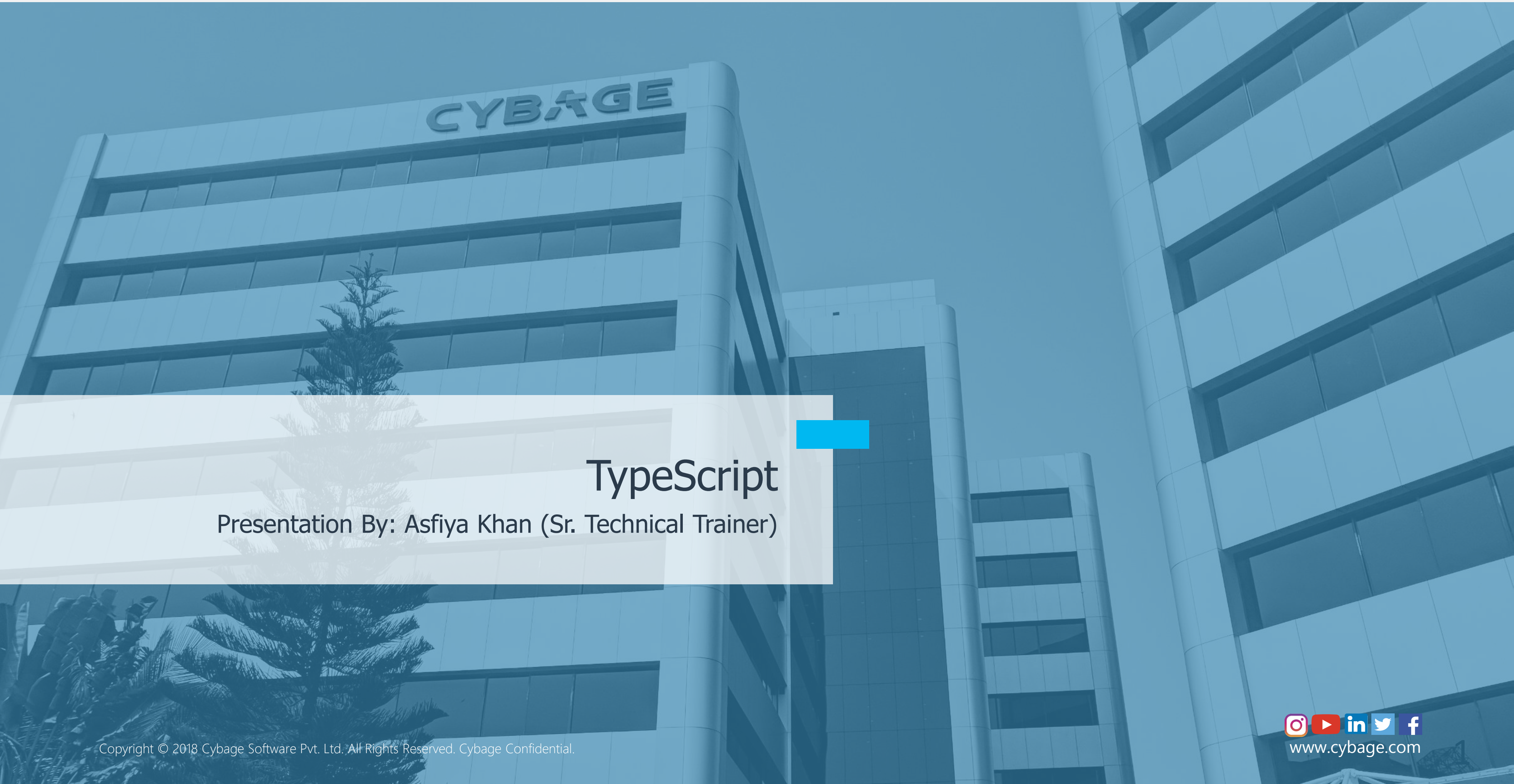


TypeScript

Presentation By: Asfiya Khan (Sr. Technical Trainer)

www.cybage.com

Document History

Version No.	Authored/ Modified by	Remarks/ Change History	Date <dd- mon-yy >
1.0	Asfiya Khan	First version of TypeScript	24 July 2019

Course Structure

Target audience	Trainee,SE,SSE
Level	1,2,3
Pre-requisites	Javascript
Training methods	Presentation , Demos, Hands-on
Evaluation	Multiple Choice Question

Agenda

- Key concepts of Typescript
- Why to write Typescript
- Some of features Typescript offers
- Tools to be used
- Some other frameworks
- Alternative to typescript

CYBAGE

Why use Typescript ?

- JavaScript can feel messy !

Why use Typescript ?

- We want maintainable code

JavaScript code encapsulation

Function Spaghetti Code

Ravioli Code
(JavaScript Pattern)

JavaScript Dynamic Types

Javascript Provides dynamic type system

The Good:

–Variable can hold any object

–Types determined on the fly

–Implicit type coercion (eg. String to number)

The Bad:

–Difficult to ensure proper types are passed without tests

–Not all developer use ===

–Enterprise –scale apps can have 1000 of lines of code to maintain

Migrating from Server-side to Client-side

- Migrating from server-side apps to client side apps can be challenging.

What are alternatives?

- Several Typescript alternative exists:
  - –Write pure JavaScript
  - –Apply JavaScript patterns
  - –Coffeescript
  - –Dart

Typescript Features

- Typescript is a typed superset of JavaScript that complies to plain JavaScript.

https://www.typescriptlang.org/

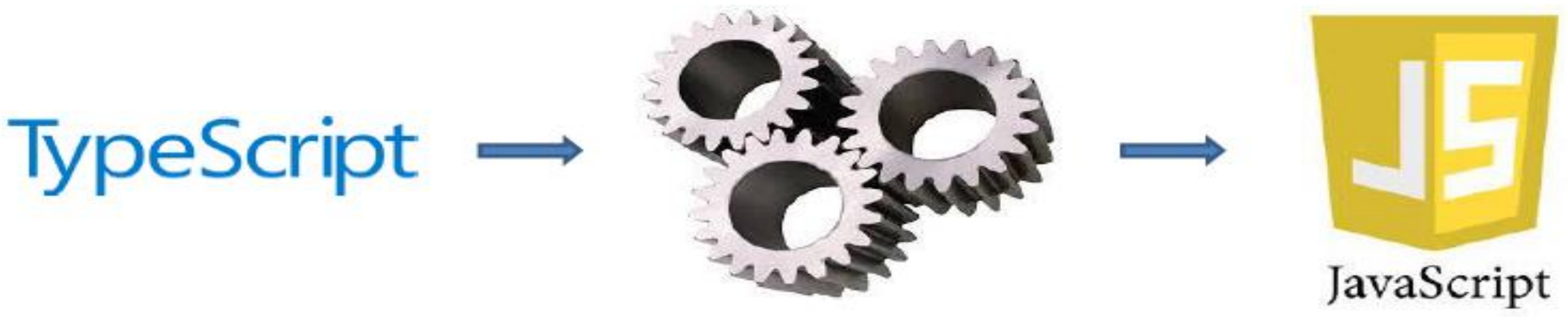
Typescript Features

- Any Browser
- Any Host
- Any OS
- Open Source
- Tool Support

Typescript Features

- Support standard Javascript code
- Provides static typing
- Encapsulati on through classes and modules
- Support for constructors, properties functions
- Define interfaces
- => Function support (lambdas)
- Intellisense and syntax checking

TypeScript Compiler



tsc first.ts

CYBAGE

TypeScript

JavaScript

Encapsulation

Static typing

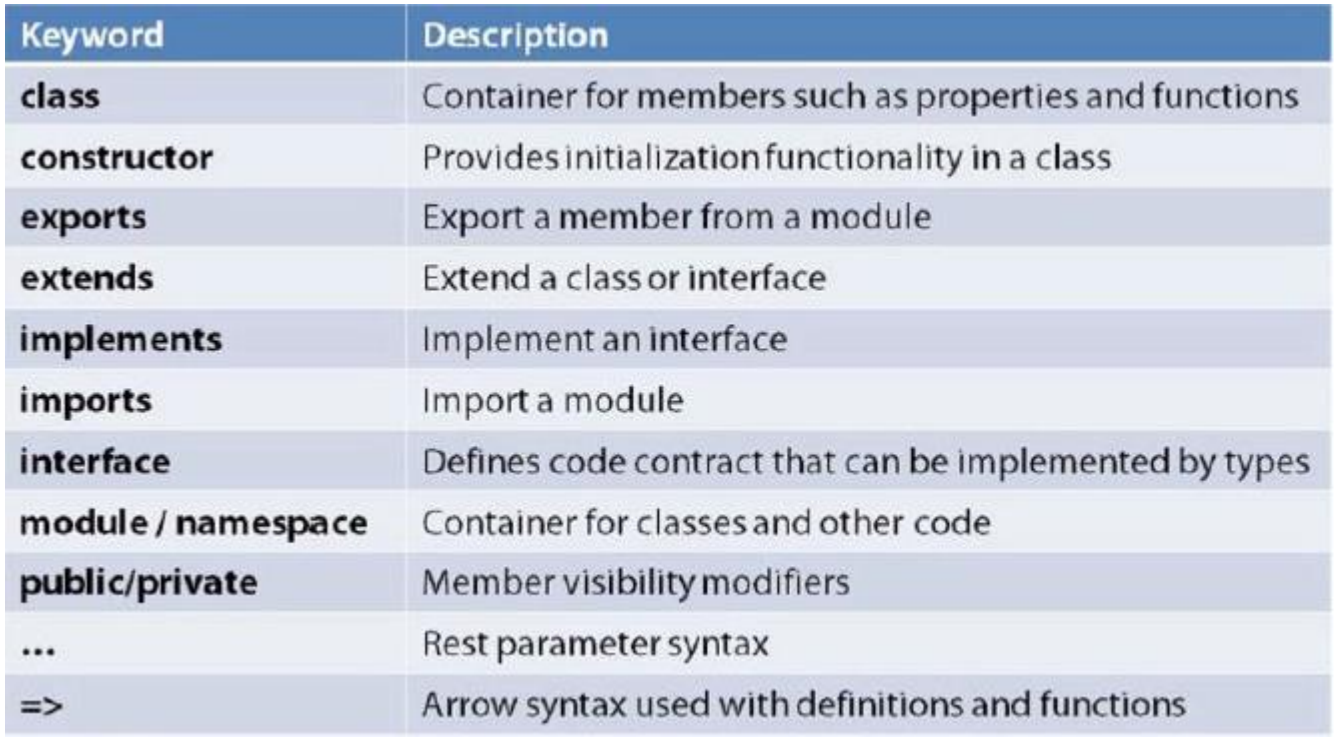
```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
```

```
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
```

TypeScript Syntax, Keywords and code Hierarchy

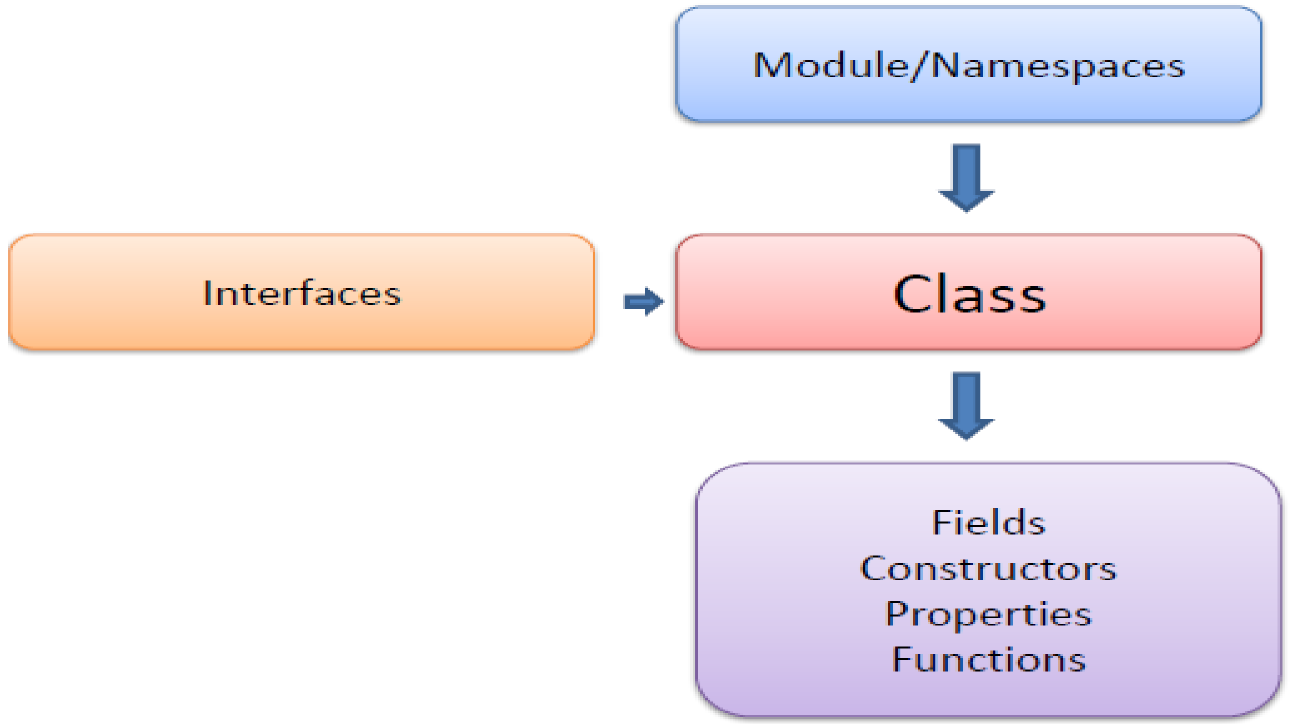
- **Typescript is superset of Javascript**
  - Follow the same syntax rules:
  - {} bracket defines code blocks
  - Semi-colons end code expressions
- **JavsScript keywords:**
  - For
  - If
  - More…

Important Keywords and Operators



Keyword	Description
class	Container for members such as properties and functions
constructor	Provides initialization functionality in a class
exports	Export a member from a module
extends	Extend a class or interface
implements	Implement an interface
imports	Import a module
interface	Defines code contract that can be implemented by types
module / namespace	Container for classes and other code
public/private	Member visibility modifiers
...	Rest parameter syntax
=>	Arrow syntax used with definitions and functions

Code Hierarchy



Tool/Framework Support

- Node.js
- Sublime
- Emacs
- Vi
- Visual Studio
- TypeScript Plyground

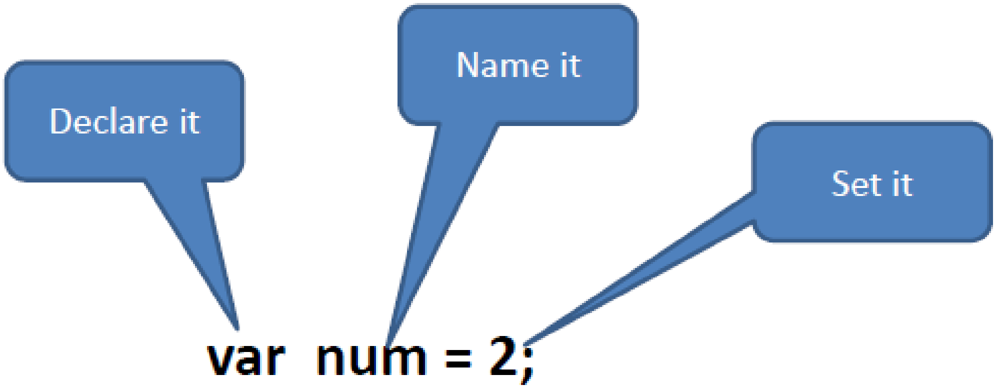
VSCode Features

- Free open source code editor
- Minimum memory footprints
- In build Git supports
- Numerous plug-ins

TypeScript types

- **Number**: the “number” is a primitive number type in TypeScript. There is no different type for float or double in TypeScript.
- **Boolean**: The “boolean” type represents true or false condition .
- **String**: The “string” represent sequence of characters similar to C#
- **Null**: The “null” is a special type which assigns null value to a variable .
- **Undefined**: The “undefined” is also a special type and can be assigned to any variable .

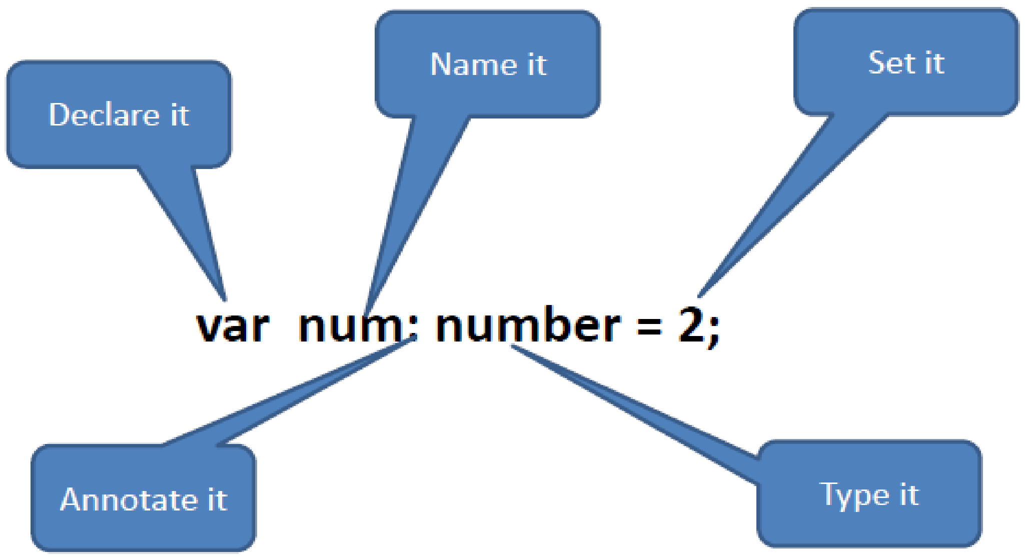
Type Inference



```
var  num = 2;
```

This typescript example is javascript

Type Annotations



Initializing the variable is optional. The initialized variable must match the annotated type

Annotations and Inference

```
var  any1;
```

Type could be any type(any) any type is base type of object. It could be string , int etc.

```
var  num1: number;
```

Type Annotation

```
var  num2: number = 2;
```

Type annotation setting the value

```
var  num3 = 3;
```

Type Inference (number)

```
var  num4 = num3 + 100;
```

Type Inference (number)

```
var  num4 = num3 + 'abc';
```

Type Inference (string)

```
var  nothappy : number = num1 + 'abc';
```

Error !

CYBAGE

Typescript Arrays

This example defines an array that can only hold Customer objects using [] syntax

```
var  custs: Customr[];

var  custs: Array<Customer>;
```

You can also declare an array as an Array of some specific data type

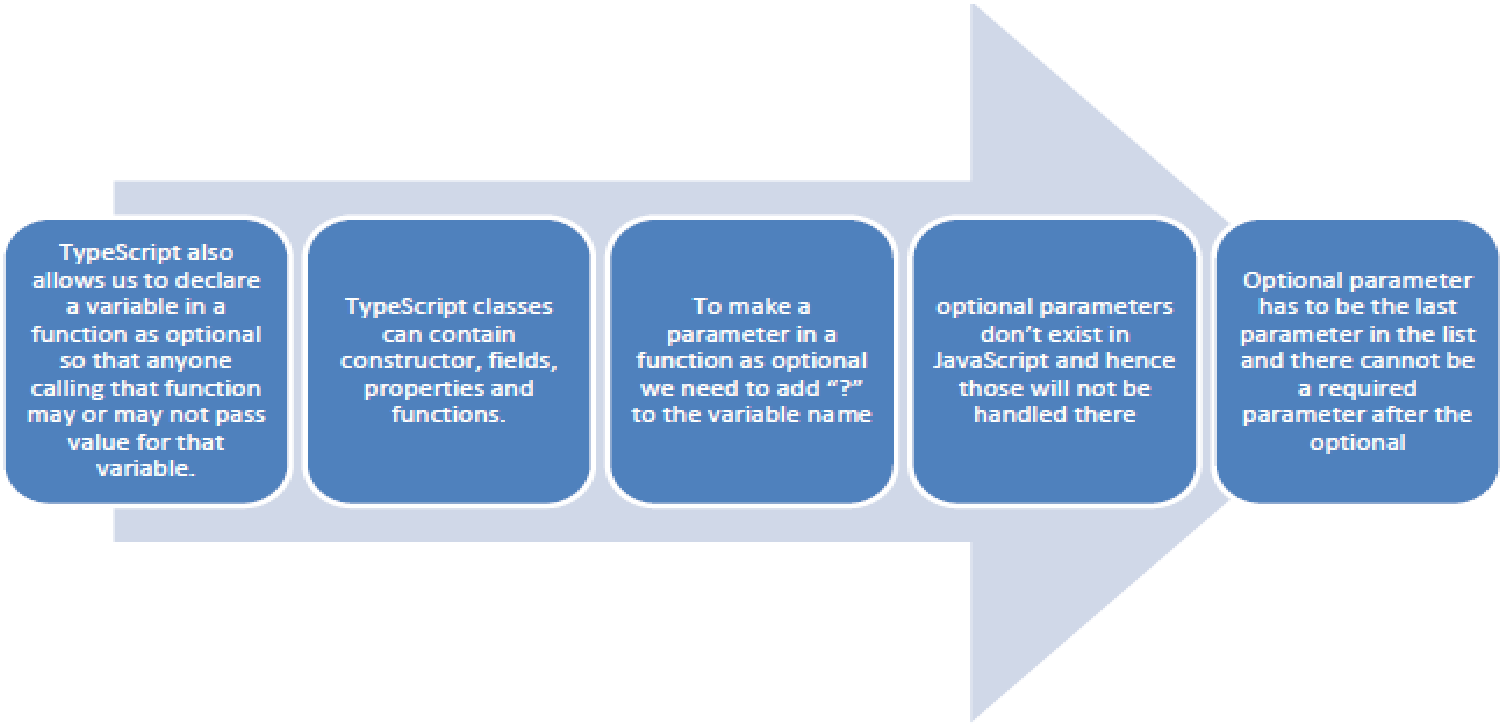
```
custs = [];
```

This code provides a very basic initialization

```
custs = [new Customer("A123"),
new Customer("B456")];
```

This example initializes my array with two Customer objects using an array literal

Optional Type


TypeScript also allows us to declare a variable in a function as optional so that anyone calling that function may or may not pass value for that variable.

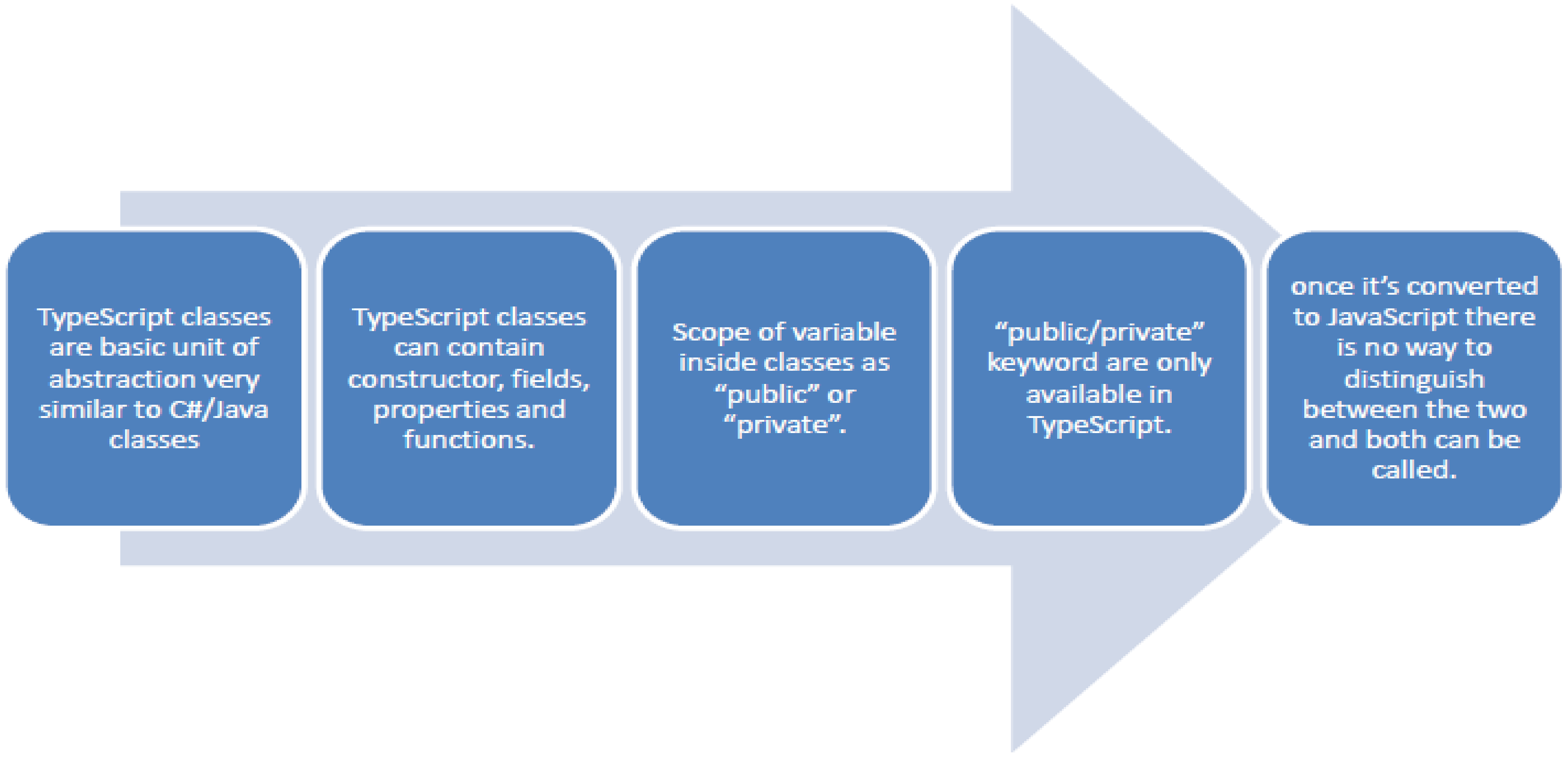
TypeScript classes can contain constructor, fields, properties and functions.

To make a parameter in a function as optional we need to add "?" to the variable name

optional parameters don't exist in JavaScript and hence those will not be handled there

Optional parameter has to be the last parameter in the list and there cannot be a required parameter after the optional

Classes



TypeScript classes are basic unit of abstraction very similar to C#/Java classes

TypeScript classes can contain constructor, fields, properties and functions.

Scope of variable inside classes as “public” or “private”.

“public/private” keyword are only available in TypeScript.

once it’s converted to JavaScript there is no way to distinguish between the two and both can be called.

CYBAGE

Inheritance

Having classes and interface means TypeScript also support inheritance which is a very powerful feature and aligns writing client side code to the way we write C# code

Using inheritance we can extend classes, implement and extend interfaces and write code which very closes recognizes with OOPs

In TypeScript when we extend a base class in child class we use keyword “super” to call the constructor of base class or even the public methods of the base class.

To extend a class in TypeScript we use “extend” keyword after the class name and the followed by the class through which we need to extend

We can also inherit interfaces on other interfaces.

Interfaces

TypeScript offers support for Interfaces to use them as a contract for classes similar to C#

Declare an interface, using keyword “interface” followed by the interface name.

Important thing to know about interfaces is that when compiled in JavaScript, interface code is ignored and there is no corresponding JavaScript generated

Classes implement interfaces using keyword “implement” followed by interface name.

As in C#, classes can implement multiple interfaces and TypeScript does a design time check to make sure that the class is implementing all the methods of that interface.

Any questions?

Thank You!

www.cybage.com