

Instructor Notes:

Add instructor notes here.

Angular 2.0

Lesson 01 : Introduction
to TypeScript



Instructor Notes:

Add instructor notes here.

Lesson Objectives

- Introduction to Typescript
- JavaScript & Typescript
- The type system-Variable, Array
- Defining class and interface
- Arrow Functions
- Template Strings
- Defining a module
- Importing a module
- Generics

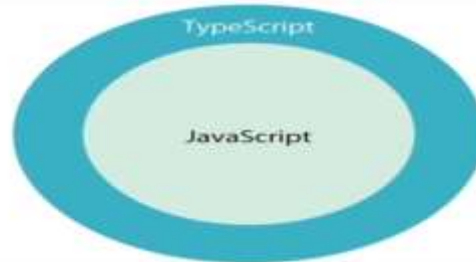


Instructor Notes:

Add instructor notes here.

TypeScript

- **TypeScript** is an open-source programming language developed and maintained by Microsoft.
- It is superset of JavaScript.
- It is a strict syntactical superset of JavaScript, and adds optional static typing to the language.
- Anders Hejlsberg, lead architect of C# and creator of Delphi & Pascal, has worked on the development of TypeScript.
- TypeScript may be used to develop JavaScript applications for client-side or server-side -Node.js execution.



TypeScript is designed for development of large applications and transpile to JavaScript. As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs.

TypeScript supports definition files that can contain type information of existing JavaScript libraries, much like C++ header files can describe the structure of existing object files. This enables other programs to use the values defined in the files as if they were statically typed TypeScript entities. There are third-party header files for popular libraries such as jQuery, MongoDB, and D3.js. TypeScript headers for the Node.js basic modules are also available, allowing development of Node.js programs within TypeScript.

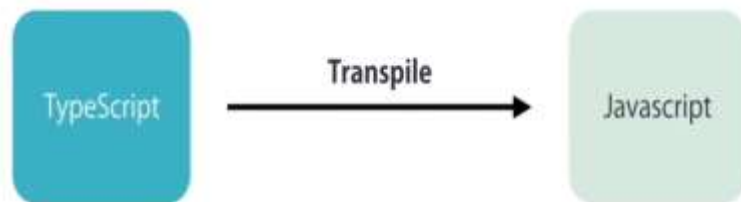
The TypeScript compiler is itself written in TypeScript and compiled to JavaScript. It is licensed under the Apache 2 License.

TypeScript is included as a first-class programming language in Microsoft Visual Studio 2013 Update 2 and later, beside C# and other Microsoft languages. An official extension allows Visual Studio 2012 to support TypeScript as well.

Instructor Notes:

TypeScript

- Strong Typing
- Object Oriented features
- Compile time catching error
- Browser don't understand typescript so we need to compile or transpile into JavaScript. So Browser can understand it



- TypeScript is a free and open source programming language developed and maintained by Microsoft.
- TypeScript is a typed superset of JavaScript.
- TypeScript is designed for development of large applications and compiles to JavaScript.
- TypeScript offers classes, modules, and interfaces to help you build robust components.
- TypeScript syntax includes several proposed features of ECMAScript 6 (ES6)
- TypeScript aligns more closely with ES6 standards and some new features of its own.
- TypeScript isn't a completely new language, it's a superset of ES6.
- ES6 code is nothing but a compilable TypeScript code.
- One of the main goals for the upcoming TypeScript 2.0 release is to fully support the ECMAScript 6 standard.

Instructor Notes:

Add instructor notes here.

Why TypeScript

- TypeScript can be used for cross browser development and is an open source project.
- Using TypeScript developers can apply class-based approach, compile them down to JavaScript without waiting for the next version of JavaScript.
- With TypeScript existing JavaScript code can be easily incorporated with popular JavaScript libraries like jQuery, Backbone, Angular and so on.
- Enable scalable application development with optional Static types, classes and modules. Static types completely disappear at runtime.
- TypeScript converts JavaScript Programming from loosely typed to strongly typed.
- JavaScript Version:
 - ES5 (ECMAScript 5): supported by all browsers
 - ES6 (2015)
 - ES2016
 - ES2017

Syntax defines a set of rules for writing programs. Every language specification defines its own syntax. A TypeScript program is composed of –

Modules
Functions
Variables
Statements and Expressions
Comments

Instructor Notes:

Getting Started with TypeScript



- Visual Studio includes TypeScript in the box, starting with Visual Studio 2013 Update 2 and in Visual Studio 2015.
- Text editors like WebStorm, Atom, SublimeText, Eclipse and Brackets(freeware) can be used.
- TypeScript compiler can be installed for Node Environment by installing it as a node package.
 - `npm install -g typescript` (installing typescript as a global module)
 - `tsc <filename.ts>` (To compile TypeScript from command line).
- In the webpages, we need to refer JavaScript code file (*.js) and not the TypeScript code file (*.ts)
- Microsoft provides an online editor, where we can compile TypeScript Code to Native JavaScript.
 - Link : <http://www.typescriptlang.org/Playground>

ES6 output mode

In previous TypeScript releases, '--target' command line option that allows us to choose between ES3 and ES5 output modes in TypeScript 1.4 a new 'ES6' option to the targets command line option:

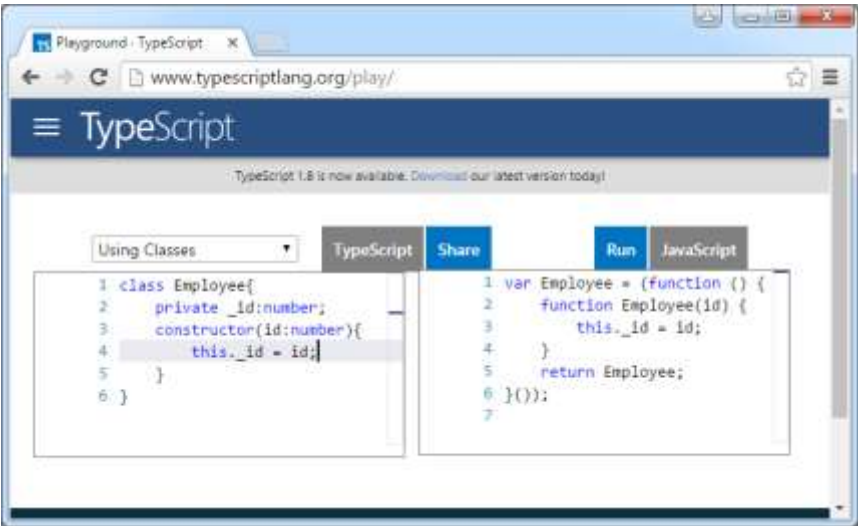
```
> tsc --target ES6 myfile.ts
```

This option will be required for ES6 features that cannot be output to ES3 or ES5, just as ES5-only features, such as getters and setters, have required the ES5 target option.

Instructor Notes:

Add instructor notes here.

TypeScript Playground

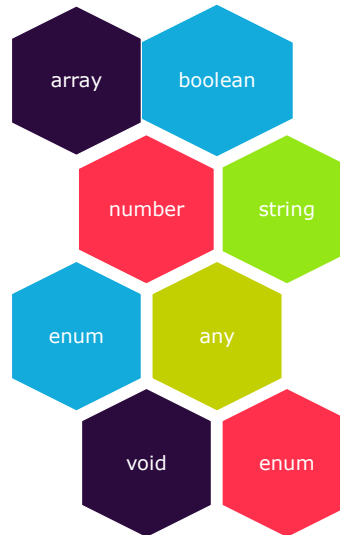


Instructor Notes:

Add instructor notes here.

Basic Types

➤ TypeScript supports the following Basic Types



```
/*number type*/  
var age:number = 33;
```

```
/*string type*/  
var name:string = "Karthik";
```

```
/*boolean type*/  
var maritalStatus:boolean = true;
```

```
/*number Array*/  
var numbers:number[] =[1,2,3];
```

```
/*number Array - Generic*/  
var numberList:Array<number> =[1,2,3];
```

```
/*Enum*/  
enum Directions {NORTH, EAST, WEST, SOUTH};  
var myDirection:Directions = Directions.EAST;
```

```
/*any type*/  
var test:any = "Karthik"
```

```
/*void*/
```

```
function nothing(): void {  
    alert("Function doesn't return a value");  
}
```


Instructor Notes:

Difference between let & var

➤ Using var

```

01. function doGet() {
02.   for (var i = 0; i < 5; i++) {
03.     document.write(i + "<br/>");
04.   }
05.   document.write("Finally : " + i);
06. }
07.
08. doGet();

```

```

0
1
2
3
4
Finally : 5

```

- Now if you use 'let' instead of 'var' it will give compilation error at line number 05, because scope is limited
- So in typescript we have to use 'let' instead of 'var'

Var

The JavaScript variables statement is used to declare a variable and, optionally, we can initialize the value of that variable.

Example: var a =10;

Variable declarations are processed before the execution of the code.

The scope of a JavaScript variable declared with var is its current execution context.

The scope of a JavaScript variable declared outside the function is global.

```

function nodeSimplified(){
var a =10;
console.log(a); // output 10
if(true){
var a=20;
console.log(a); // output 20
}
console.log(a); // output 20
}

```

let

The **let** statement declares a local variable in a block scope. It is similar to **var**, in that we can optionally initialize the variable.

Example: let a =10;

The let statement allows you to create a variable with the scope limited to the block on which it is used.

It is similar to the variable we declare in other languages like Java, .NET, etc.

```

function nodeSimplified(){
let a =10;
console.log(a); // output 10
if(true){
let a=20;
console.log(a); // output 20
}
console.log(a); // output 10
}

```

Instructor Notes:

Type Annotations

- Type annotations in TypeScript are lightweight ways to record the intended contract of the function or variable.

```
let empld: number;           --- number
let empName: string;        --- string
let empFeedback: boolean;   --- Boolean
let anyType: any;           --- any
let myArray: number[] = [1, 2, 3]; --- number
let anyArrayType: any[] = [1, 'Rahul', false, true]; --- any array
```

Instructor Notes:

Type Annotations



➤ Enum & Constant

```
const colored = 0;  
const colorBlue = 1;  
const colorGreen = 2;  
  
enum Color { Red = 0, Green = 1, Blue = 2 };  
  
let backgroundColor = Color.Red;  
  
document.write(backgroundColor.toString());
```

Instructor Notes:

Type Assertion in TypeScript

- TypeScript allows changing a variable from one type to another.
- TypeScript refers to this process as *Type Assertion*.
- The syntax is to put the target type between `< >` symbols and place it in front of the variable or expression.

```
let str: string;
```

```
str = "Angular JS";
```

```
let str2;
```

```
str2 = str.substring(2,3);
```

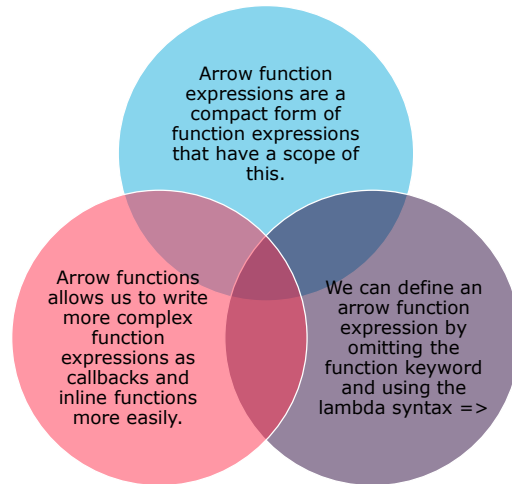
```
(<string>str2).length;
```

```
(str2 as string).length;
```

Assertion in
TypeScript

Instructor Notes:

Add instructor notes here.

Arrow Functions

```
var addNumbers = (firstNumber:number, secondNumber:number) => firstNumber + secondNumber;
addNumbers(5, 6);
```

/*Without arrow function*/

```
class Test{
  private _name = "Karthik";
  printName() {
    setTimeout((function () {
      //this refers window Object not the Test Object
      alert(this._name);
    }),1000);
  }
}
```

```
var testObj: Test = new Test();
testObj.printName();
```

/*Using arrow function*/

```
class Test{
  private _name = "Shilpa";
  printName() {
    setTimeout(() => {
      alert(this._name);
    },3000);
  }
}
```

```
var testObj: Test = new Test();
testObj.printName();
```

Instructor Notes:

Arrow Functions

```
let log = function(message)
{
  console.log('Welcome to Arrow');
}

//Arrow function equivalent to above function
let doLog = (message) => console.log(message);

//Arrow function equivalent to no parameter function
let withoutparameter = () => console.log();
```

Instructor Notes:

Add instructor notes here.

Classes in TypeScript

- Traditional JavaScript focuses on functions and prototype-based inheritance, it is very difficult to build application using object-oriented approach.
- Starting with ECMAScript 6 (the next version of JavaScript), JavaScript programmers can build their applications using this object-oriented class-based approach.
- TypeScript supports public, private and protected access modifiers. Members of a class are public by default.

```
class Employee{  
    private _id:number;  
    constructor(id:number){  
        this._id = id;  
    }  
}
```

Code Snippet

Creating classes

Use the class keyword to declare a class in TypeScript. The syntax for the same is given below –

```
class class_name { //class scope }
```

The class keyword is followed by the class name. The rules for identifiers must be considered while naming a class.

A class definition can include the following –

Fields – A field is any variable declared in a class. Fields represent data pertaining to objects

Constructors – Responsible for allocating memory for the objects of the class

Functions – Functions represent actions an object can take. They are also at times referred to as methods

A constructor is a special function of the class that is responsible for initializing the variables of the class. TypeScript defines a constructor using the constructor keyword. A constructor is a function and hence can be parameterized.

The **this** keyword refers to the current instance of the class. Here, the parameter name and the name of the class's field are the same. Hence to avoid ambiguity, the class's field is prefixed with the **this** keyword.

Instructor Notes:

Add instructor notes here.

Classes in TypeScript (Contd...)

```
class Employee {
  empId: number;
  empName: string;
  empsalary: number;

  static emppf: number = 12;
  static company: string = 'CAPGEMINI';
}

let emp = new Employee();
emp.empId = 1001;
emp.empName = "Vikash";
emp.empsalary = 1111;
document.write("ID is " + emp.empId + " Name is " + emp.empName + "
company " + Employee.company);
```

Code Snippet

Creating classes

Use the class keyword to declare a class in TypeScript. The syntax for the same is given below –

```
class class_name { //class scope }
```

The class keyword is followed by the class name. The rules for identifiers must be considered while naming a class.

A class definition can include the following –

Fields – A field is any variable declared in a class. Fields represent data pertaining to objects

Constructors – Responsible for allocating memory for the objects of the class

Functions – Functions represent actions an object can take. They are also at times referred to as methods

A constructor is a special function of the class that is responsible for initializing the variables of the class. TypeScript defines a constructor using the constructor keyword. A constructor is a function and hence can be parameterized.

The **this** keyword refers to the current instance of the class. Here, the parameter name and the name of the class's field are the same. Hence to avoid ambiguity, the class's field is prefixed with the **this** keyword.

Instructor Notes:

Constructor -Typescript

```
class EmployeeOne {  
  empld: number;  
  empName: string;  
  
  constructor(id: number, name: string) {  
    this.empld = id;  
    this.empName = name;  
  }  
  
  doGet(): void {  
    document.write(this.empld + " " + this.empName);  
  }  
}  
  
let empOne = new EmployeeOne(1001, "Abcd");  
empOne.doGet();
```

Data hiding

1 public

A public data member has universal accessibility. Data members in a class are public by default.

2. private

Private data members are accessible only within the class that defines these members. If an external class member tries to access a private member, the compiler throws an error.

3. protected

A protected data member is accessible by the members within the same class as that of the former and also by the members of the child classes.

Instructor Notes:

Add instructor notes here.

Accessors

TypeScript supports getters/setters as a way of intercepting accesses to a member of an object.



It gives way of having finer-grained control over how a member is accessed on each object.

```
class Foo {
  private _name: string;
  get name():string{
    return this._name;
  }
  set name(name){
    this._name=name;
  }
}

var fooObj = new Foo();
fooObj.name = "Karthik";
fooObj.name;
```

Creating Properties in JavaScript

```
> var foo = {
  fooProperty: 'Foo String'
}
> var baz = Object.create(foo,{
  bazProperty:{
    value:'Baz String',
    writable:true,
    configurable:true,
    enumerable:true
  }
});
> baz
Object {bazProperty: "Baz String", fooProperty: "Foo String"}
> baz.bazProperty = "Changed String"
Changed String
> for(key in baz){
  console.log(key);
}
bazProperty
fooProperty
> delete baz.bazProperty
true
> baz
Object {fooProperty: "Foo String"}
```

Instructor Notes:

Static Property

- In TypeScript we can also create static members of a class, those that are visible on the class itself rather than on the instances.

```
class Foo {
  static staticVariable:string;
  instanceVariable:string
  constructor(instanceVariable:string){
    this.instanceVariable = instanceVariable;
  }
  static staticMethod(){
    return Foo.staticVariable;
  }
}

var fooObj = new Foo("Instance");
console.log(fooObj.instanceVariable);
Foo.staticVariable = "Static"
Foo.staticMethod();
```

The static Keyword

The static keyword can be applied to the data members of a class. A static variable retains its values till the program finishes execution. Static members are referenced by the class name.

Instructor Notes:

Static Property (Contd...)

```
class EmployeeOne {  
    empId: number;  
    empName: string;  
    static numberOfEmployee: number = 0;  
  
    constructor(id: number, name: string) {  
        this.empId=id;  
        this.empName=name;  
        EmployeeOne.numberOfEmployee++;  
    }  
  
    doGet(): void{  
        document.write(this.empId+" "+this.empName);  
    }  
    static getNumber(): number{  
        return EmployeeOne.numberOfEmployee;  
    }  
}  
  
let empOne=new EmployeeOne(1001, "Abcd");  
empOne.doGet();  
EmployeeOne.getNumber();
```

The static Keyword

The static keyword can be applied to the data members of a class. A static variable retains its values till the program finishes execution. Static members are referenced by the class name.

Instructor Notes:

Function:

➤ Creating Functions

```
//2 parameter with number as return type
function getsum(numOne: number, numTwo: number): number{
    return numOne + numTwo;
}
```

```
let add = getsum(10,6);
document.write("Sum is " + add + "<br />");
```

```
//any number of data--know as rest parameter
function sumAll(...num: number[]){
    let sum: number = 0;
    for (let data of num) {
        sum = sum + data;
        document.write("Addition of number " + data + "<br />");
    }
    document.write("Sum is " + sum + "<br />");
}

sumAll(6, 7, 8, 9);
```

Optional Parameters

Optional parameters can be used when arguments need not be compulsorily passed for a function's execution. A parameter can be marked optional by appending a question mark to its name. The optional parameter should be set as the last argument in a function. The syntax to declare a function with optional parameter is as given below –

```
function function_name (param1[:type], param2[:type], param3[:type])
```

Rest Parameters

Rest parameters are similar to variable arguments in Java. Rest parameters don't restrict the number of values that you can pass to a function. However, the values passed must all be of the same type. In other words, rest parameters act as placeholders for multiple arguments of the same type.

To declare a rest parameter, the parameter name is prefixed with three periods. Any nonrest parameter should come before the rest parameter.

Instructor Notes:

Add instructor notes here.

Optional, Default & Rest Parameters

- In JavaScript, every parameter is considered optional, we can get this functionality in TypeScript by using the '?' beside parameters we want optional. Optional parameters must follow required parameters.
 - `function test(firstName:string, lastName?:string){ }`
- In TypeScript, we can also set up a value that an optional parameter will have if the user does not provide one. These are called default parameters.
 - `function test(firstName:string, lastName = "Gupta"){ }`
- To pass multiple parameters as a group or when we don't know how many parameters a function take we can use parameter followed by (...).
 - `function test(firstName:string,...others:string[]){ }`

Code Snippet

```
function buildName(firstName: string, middleName?: string, lastName = "M",
...others:string[]) {

    if (middleName !== undefined)
        console.log("%s %s %s", firstName, middleName, lastName);
    else
        console.log("%s %s", firstName, lastName);

    if(others.length > 0){
        others.forEach(function(item){
            console.log(item);
        });
    }
}

buildName("Karthik");
buildName("Ashik", "Mohammed", "Ibrahim");
buildName("Ashik", "Mohammed", "Ibrahim", "Test1", "Test2");
```

Instructor Notes:

Optional, Default & Rest Parameters

➤ ? Is known as optional parameter

```
//Optional parameter----? for optional & Default parameter
function doGet(one: number, two = 5, three?: number): void{
  //alert("hii");
  document.write(one.toString());
  document.write(two.toString());
  document.write(three.toString());
}

//doGet(10);
doGet(10);
```

Default Parameters

Function parameters can also be assigned values by default. However, such parameters can also be explicitly passed values.

Instructor Notes:

Add instructor notes here.

Template Strings

- In ES6 new template strings were introduced.
- The two salient features of template strings are
 - Variables within strings (without being forced to concatenate with +)
 - Multi-line strings (using backticks `)
 - TypeScript now supports ES6 template strings. These are an easy way to embed arbitrary expressions in strings:

```
var name = "TypeScript";  
var greeting = `Hello, ${name}! Your name has ${name.length} characters`;
```

- When compiling to pre-ES6 targets, the string is decomposed:

```
var name = "TypeScript!";  
var greeting = "Hello, " + name + " ! Your name has " + name.length + " characters";
```

Code Snippet

Instructor Notes:

Add instructor notes here.

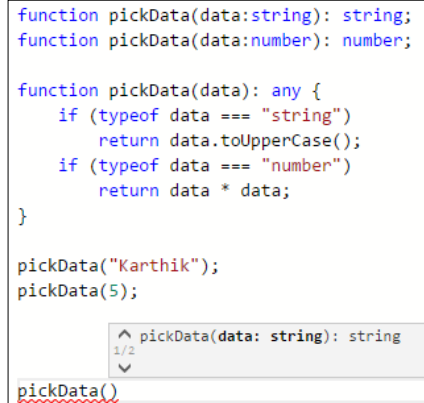
Function Overload

- JavaScript is inherently a very dynamic language.
- JavaScript functions can be overloaded and return different types of objects based on the arguments passed in.

```
function pickData(data:string): string;
function pickData(data:number): number;

function pickData(data): any {
  if (typeof data === "string")
    return data.toUpperCase();
  if (typeof data === "number")
    return data * data;
}

pickData("Karthik");
pickData(5);
```



Instructor Notes:

Add instructor notes here.

Generics

- Generics plays a vital role in creating reusable components.
- Component can be created to work over a variety of types rather than a single one.

```
function GetType<T>(val: T): string{
    return typeof(val);
}

let ename = "Abcd";
let one = 10;
document.write("Call Generics" + GetType(ename) + " " + GetType(one));

//class -generics
class GetNumber<T>{
    add:(one: T, two: T) => T;
}
var result = new GetNumber<number>();
result.add = function(x, y){
    return x+y;
}
document.write("Addition of 5+2" + result.add(5,2));
```

TypeScript 2.3 implemented *generic parameter defaults* which allow you to specify default types for type parameters in a generic type.

```
type Constructor<T> = new (...args: any[]) => T; type Constructable = Constructor<{}>;
```

Instructor Notes:

Add instructor notes here.

Inheritance

- TypeScript allows us to extend existing classes to create new ones using inheritance.
- 'extends' keyword is used to create a subclass.
- 'super()' method is used to call the base constructor inside the sub class constructor.

```
class Foo {  
  constructor() {  
    console.log("Foo constructor Invoked");  
  }  
}  
  
class Baz extends Foo {  
  constructor() {  
    super();  
    console.log("Baz constructor Invoked");  
  }  
}  
  
new Baz();
```

Code Snippet

Class Inheritance

TypeScript supports the concept of Inheritance. Inheritance is the ability of a program to create new classes from an existing class. The class that is extended to create newer classes is called the parent class/super class. The newly created classes are called the child/sub classes.

Inheritance can be classified as –

Single – Every class can at the most extend from one parent class

Multiple – A class can inherit from multiple classes. TypeScript doesn't support multiple inheritance.

Multi-level – The following example shows how multi-level inheritance works.

Instructor Notes:

Add instructor notes here.

Inheritance (Contd...)

```
class Animal {  
  constructor(public name: string) { }  
  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("Gallopig...");  
    super.move(distanceInMeters);  
  }  
}
```

Code Snippet

Class Inheritance

TypeScript supports the concept of Inheritance. Inheritance is the ability of a program to create new classes from an existing class. The class that is extended to create newer classes is called the parent class/super class. The newly created classes are called the child/sub classes.

Inheritance can be classified as –

Single – Every class can at the most extend from one parent class

Multiple – A class can inherit from multiple classes. TypeScript doesn't support multiple inheritance.

Multi-level – The following example shows how multi-level inheritance works.

Instructor Notes:

Add instructor notes here.

Interfaces

➤ Interfaces plays many roles in TypeScript code. Its work same as other OOPs concept.

- Describing an Object : If function take lot of members but you're likely to pass a few of those, we can describe an object using interface with optional parameters
- Describing an Indexable Object : Using TypeScript interfaces we can represent the expected type of an indexing operation.
- Ensuring Class Instance Shape: Interface can be used in the same way which we use in traditional OOP languages like C# and JAVA.

| | | |
|---|---|---|
| <pre>interface testable{ name:string; } function test(args:testable){ console.log(args.name); }</pre> | <pre>interface CarPart { [name: string]: string; } class Test{ part:CarPart={}; } var testObj = new Test(); testObj["frame"] = "Frame";</pre> | <pre>interface Greetable { greet(message: string): void; language: string; } class Greet implements Greetable{ language = 'English'; greet(message: string) { console.log(message); } }</pre> |
|---|---|---|

Code Snippet

Declaring Interfaces

The interface keyword is used to declare an interface. Here is the syntax to declare an interface –

Syntax

```
interface interface_name { }
```

Instructor Notes:

Interfaces

```
interface Employee{  
  firstName: string;  
  lastName: string;  
  age: number;  
  salary: number;  
}
```

```
let employee: Employee={  
  firstName: "Rahul",  
  lastName: "Vikash",  
  age: 32,  
  salary: 1000  
}
```

```
document.write("Full Name is " + this.employee.firstName + " " +  
this.employee.lastName + " Age is " + this.employee.age + "<br />");
```

Instructor Notes:

Interface with array

```
interface Employee{
  firstName: string;
  lastName: string;
  age: number;
  salary: number;
}

//with array concept
let empArray: Employee[] = [];

empArray.push({
  firstName: "Abcd",
  lastName: "Bcde",
  age: 21,
  salary: 6000
});

document.write("With array Full name is " + this.empArray[0].firstName + " " +
this.empArray[0].lastName + " Age is " + this.empArray[0].age);
```

Interfaces and Arrays

Interface can define both the kind of key an array uses and the type of entry it contains. Index can be of type string or type number.

Instructor Notes:

Add instructor notes here.

Modules

- A module is the overall container which is used to structure and organize code.
- It avoids collision of the methods/variables with other global APIs.
- TypeScript support modules to organize the code.
- 'export' keyword is used before the classes/ inner module to make it visible outside the module.
- We can refer the multi file internal module using the following line and compile multiple internal file into a single JS file using typescript compiler
 - `/// <reference path="<typescriptfilename>" />`
- External modules can be imported using 'import' keyword
 - `import module1 = require('./Module1');`

To Compile the Internal Module

- `tsc --out Module.js Test-Module-Internal.ts`

To Compile the External Module

- `tsc --module commonjs Test-Module-External.ts`

Instructor Notes:

Add instructor notes here.

Modules (Contd...)

- Starting with the ECMAScript 2015, JavaScript has a concept of modules. TypeScript shares this concept.
- Modules are executed within their own scope, not in the global scope; this means that variables, functions, classes etc. declared in a module are not visible outside the module unless they are explicitly exported using one of the export forms.
- To consume a variable, function, class, interface etc. exported from a different module.

Exporting and importing from modules are done with the following syntax:

```
export { StudentInfo }
```

```
import { StudentInfo } from './IStudentInfo';
```

Instructor Notes:

Modules (Contd...)

➤ Product.ts

```
export class IProduct {  
  productId: number;  
  productName: string;  
}  
  
export const company: string = "Capgemini";
```

Instructor Notes:

Modules (Contd...)

```
import {IProduct} from "./Product";
import {company} from "./Product";
//Declare Product
let prod: IProduct={
    productId:1001,
    productName:"iPhone"
}
let productArray: IProduct[]={
    {productId: 1002, productName: "LG"},
    {productId: 1003, productName: "CoolPad"},
    {productId: 1004, productName: "Mi"} ];
console.log(prod.productId);
console.log(prod.productName);

for (let pro of productArray) {
    console.log(prod.productId);
    console.log(prod.productName);
}

console.log(company);
```

Instructor Notes:

Modules (Contd...)

```
D:\AllDemoAngular\TypeScriptModule>tsc ProductUsingModule.ts
D:\AllDemoAngular\TypeScriptModule>node ProductUsingModule.js
1001
iPhone
1001
iPhone
1001
iPhone
1001
iPhone
capgemini
```

Instructor Notes:

Add instructor notes here.

Summary

- TypeScript is an open source project maintained by Microsoft.
- TypeScript generates plain JavaScript code which can be used with any browser.
- TypeScript offers many features of object oriented programming languages such as classes, interfaces, inheritance, overloading and modules, some of which are proposed features of ECMA Script 6.
- TypeScript is a promising language that can certainly help in writing neat code and organize JavaScript code making it more maintainable and extensible.
- Angular 2 is built in typescript



Instructor Notes:

Demo

- TypeScript Demo
- Typescript Module Demo



Instructor Notes:

Add instructor notes here.

Lab

➤ Lab 1.1



Add the notes here.