Instructor Notes:

Add instructor notes here.

Instructor Notes:

Add instructor notes here.

Lesson Objectives

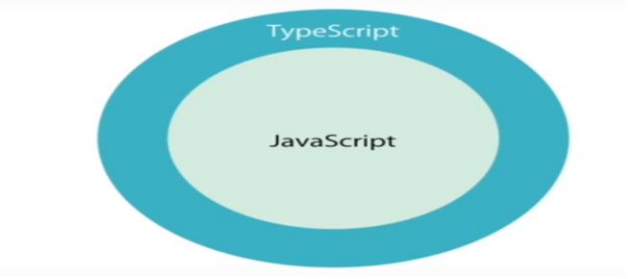
- Introduction to Typescript
- JavaScript & Typescript
- The type system-Variable, Array
- Defining class and interface
- Arrow Functions
- Template Strings
- Defining a module
- Importing a module
- Generics

Instructor Notes:

Add instructor notes here.

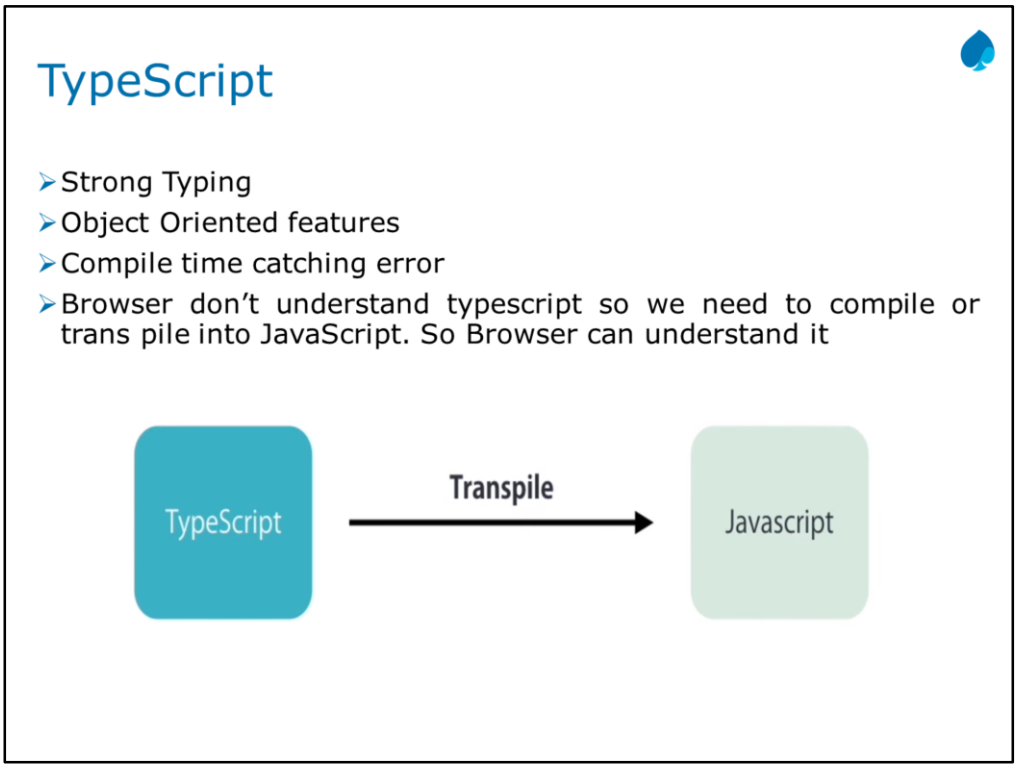
TypeScript

- **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

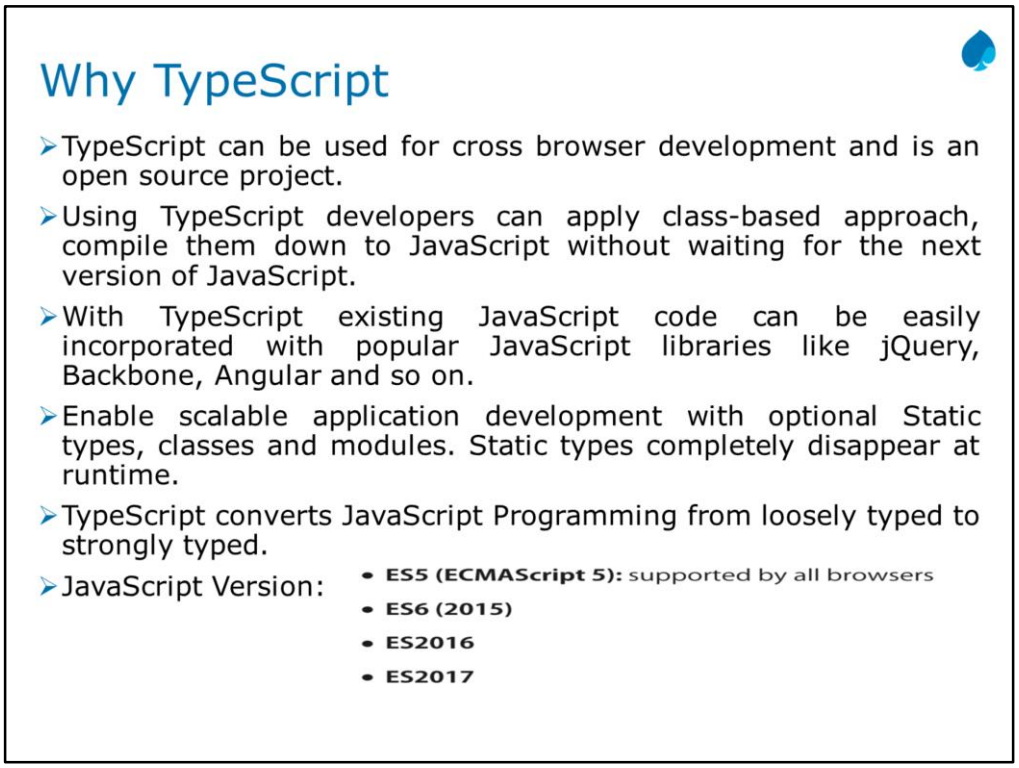
Instructor Notes:



Compilation − JavaScript is an interpreted language. Hence, it needs to be run to test that it is valid. It means you write all the codes just to find no output, in case there is an error. Hence, you have to spend hours trying to find bugs in the code. The TypeScript transpiler provides the error-checking feature. TypeScript will compile the code and generate compilation errors, if it finds some sort of syntax errors. This helps to highlight errors before the script is run.
Strong Static Typing − JavaScript is not strongly typed. TypeScript comes with an optional static typing and type inference system through the TLS (TypeScript Language Service). The type of a variable, declared with no type, may be inferred by the TLS based on its value.
TypeScript **supports type definitions** for existing JavaScript libraries. TypeScript Definition file (with **.d.ts** extension) provides definition for external JavaScript libraries. Hence, TypeScript code can contain these libraries.
TypeScript **supports Object Oriented Programming** concepts like classes, interfaces, inheritance, etc.
At its heart, TypeScript has the following three components −
Language − It comprises of the syntax, keywords, and type annotations.
The TypeScript Compiler − The TypeScript compiler (tsc) converts the instructions written in TypeScript to its JavaScript equivalent.
The TypeScript Language Service − The "Language Service" exposes an additional layer around the core compiler pipeline that are editor-like applications. The language service supports the common set of a typical editor operations like statement completions, signature help, code formatting and outlining, colorization, etc.

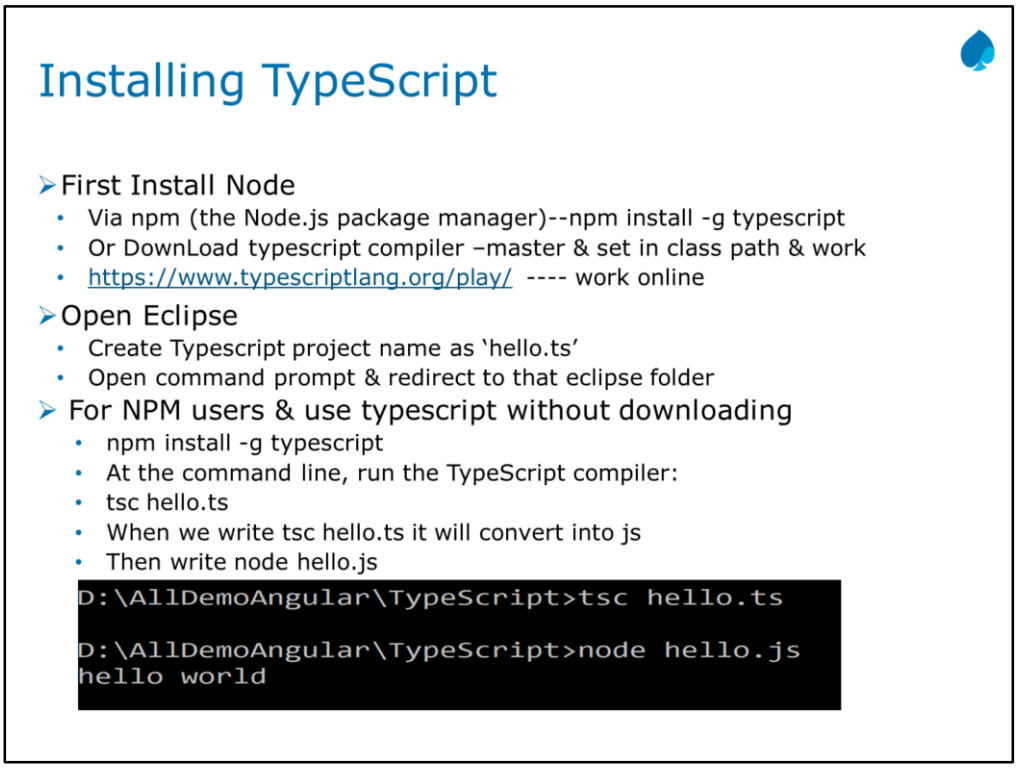
Instructor Notes:

Add instructor notes here.



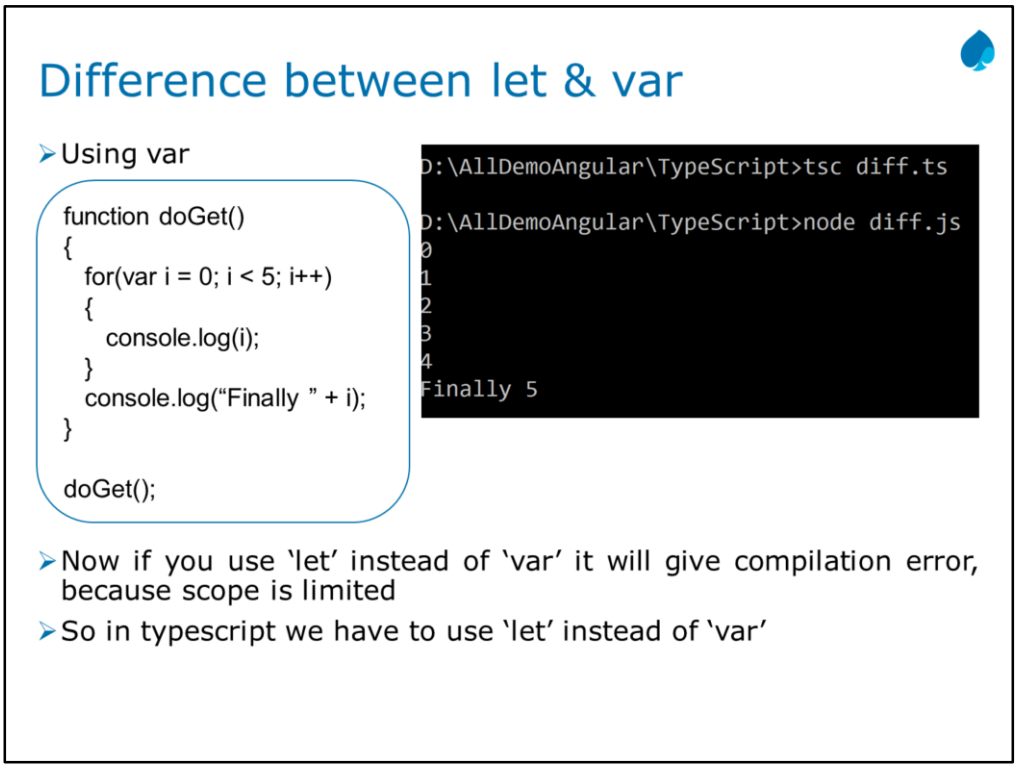
Syntax defines a set of rules for writing programs. Every language specification defines its own syntax. A TypeScript program is composed of −
Modules
Functions
Variables
Statements and Expressions
Comments

Instructor Notes:



The TypeScript Compiler
The TypeScript compiler is itself a **.ts** file compiled down to JavaScript (.js) file. The TSC (TypeScript Compiler) is a source-to-source compiler (transcompiler / transpiler).
The TSC generates a JavaScript version of the **.ts** file passed to it. In other words, the TSC produces an equivalent JavaScript source code from the Typescript file given as an input to it. This process is termed as transpilation.
However, the compiler rejects any raw JavaScript file passed to it. The compiler deals with only **.ts** or **.d.ts** files.
Installing Node.js
Node.js is an open source, cross-platform runtime environment for server-side JavaScript. Node.js is required to run JavaScript without a browser support. It uses Google V8 JavaScript engine to execute code. You may download Node.js source code or a pre-built installer for your platform. Node is available here
− *https://nodejs.org/en/download*

Instructor Notes:



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
}
```

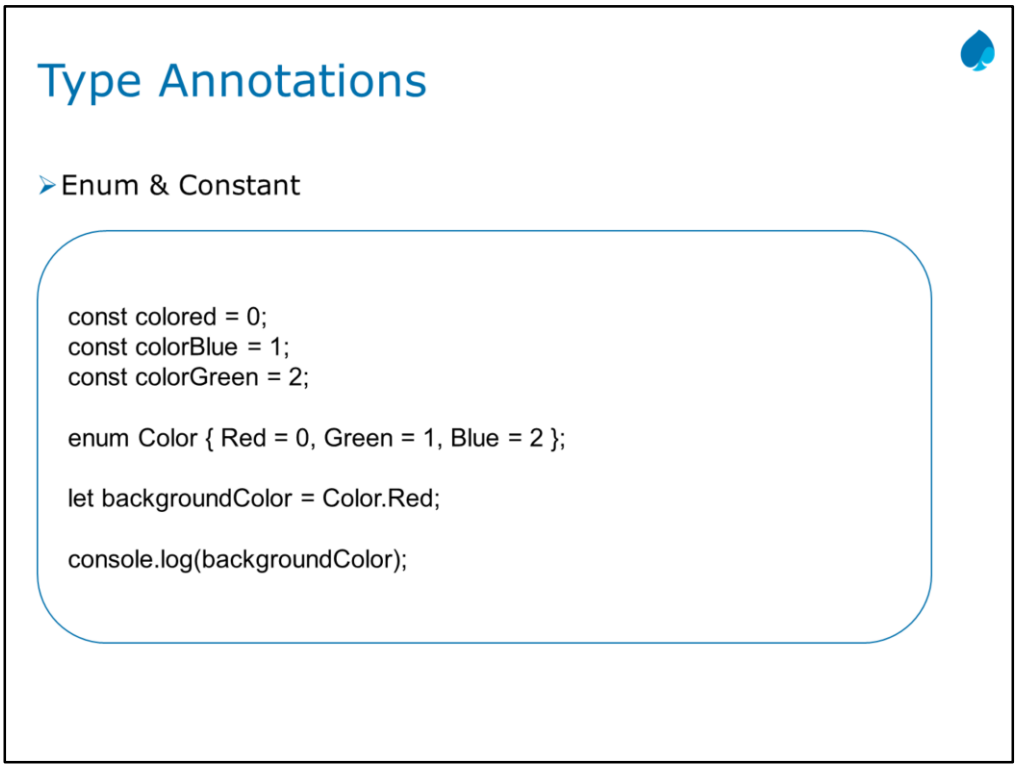
Instructor Notes:

Type Annotations

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

```
let empId: 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:



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.substring(2,3);


let str2;

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

Assertion in TypeScript

Instructor Notes:

Arrow Functions

- => is a and also called a Arrow function

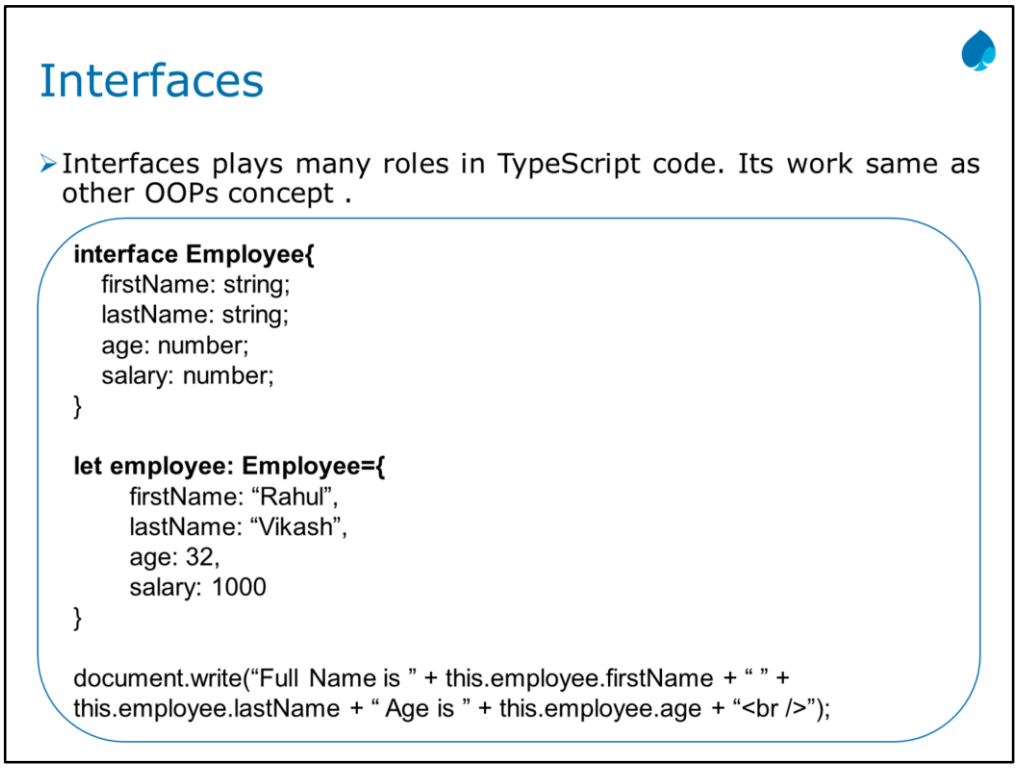
```
let log = function(message)
{
      console.log('Welcome to Arrow');
}

//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:

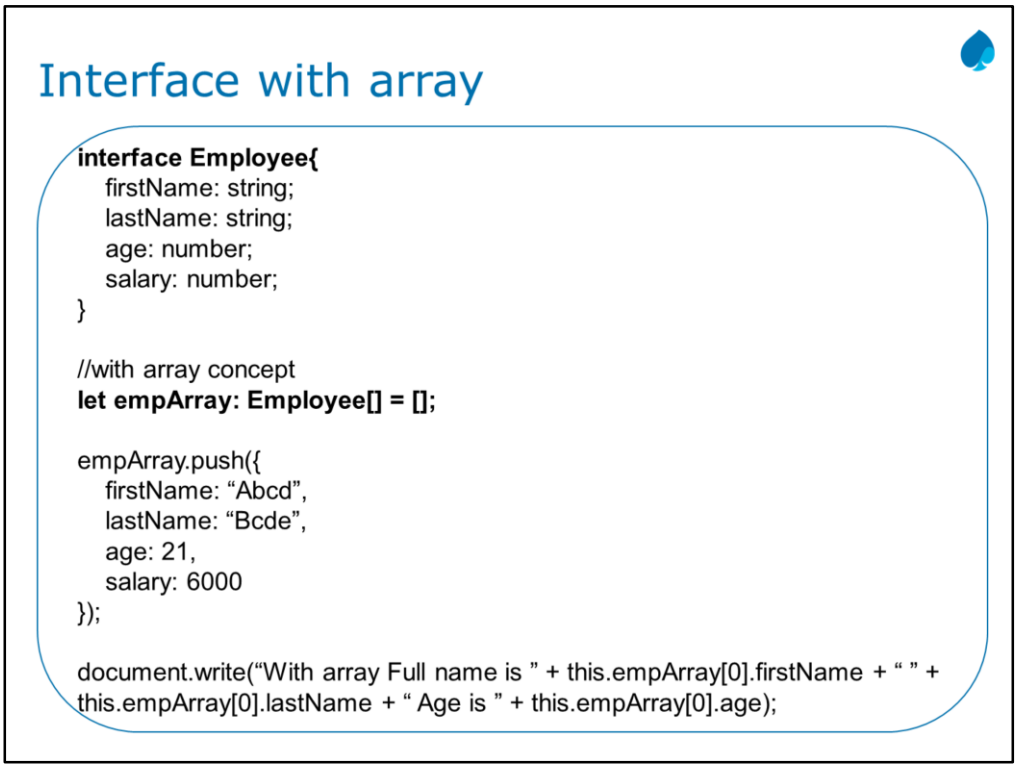
Add instructor notes here.



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 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:

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:

Optional, Default

- ? Is know as optional parameter

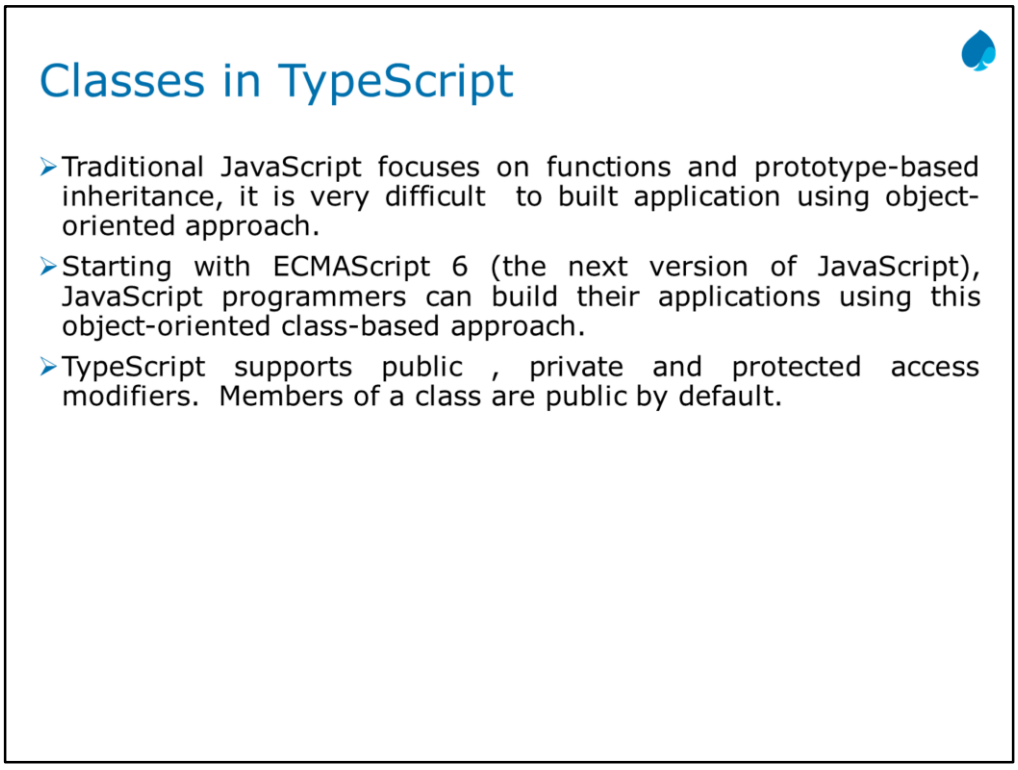
```
//Optional parameter----? for optional & Default parameter
function doGet(one: number, two = 5, three?: number): void{
    //alert("hii");
    document.write(one.toString());
    document.write(two.toString());
    document.write(three.toString());
}

//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.

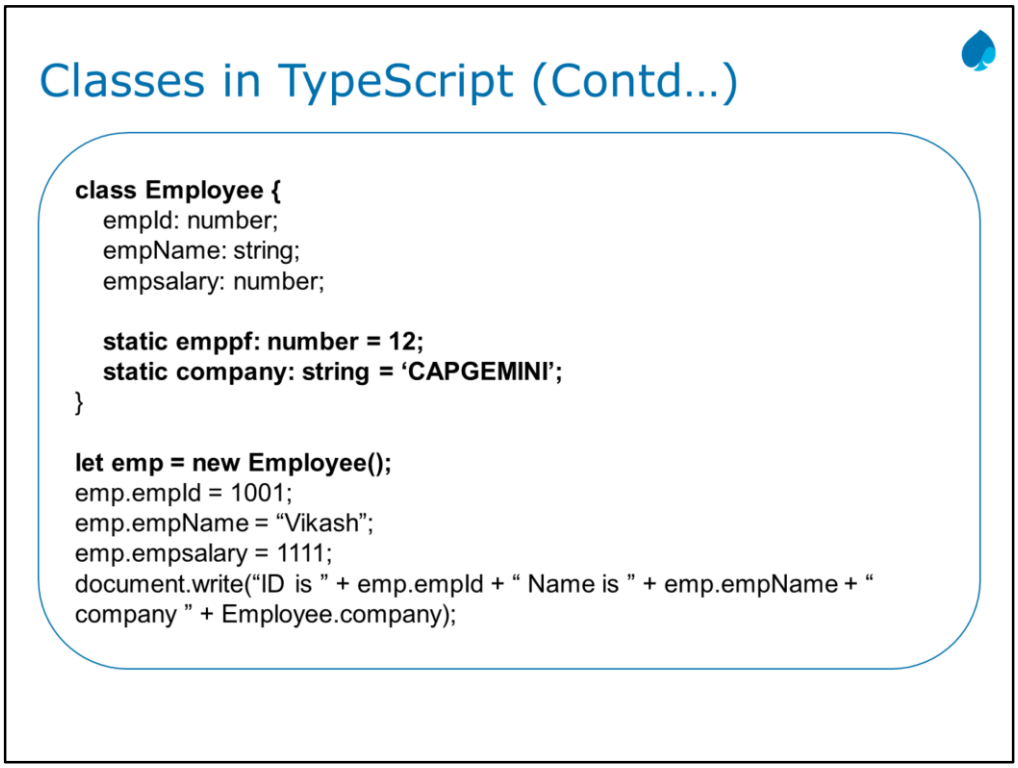


Code Snippet

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.



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 {
    empId: number;
    empName: string;

    constructor(id: number, name: string) {
        this.empId = id;
        this.empName = name;
    }

    doGet(): void{
        document.write(this.empId + " " + 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.

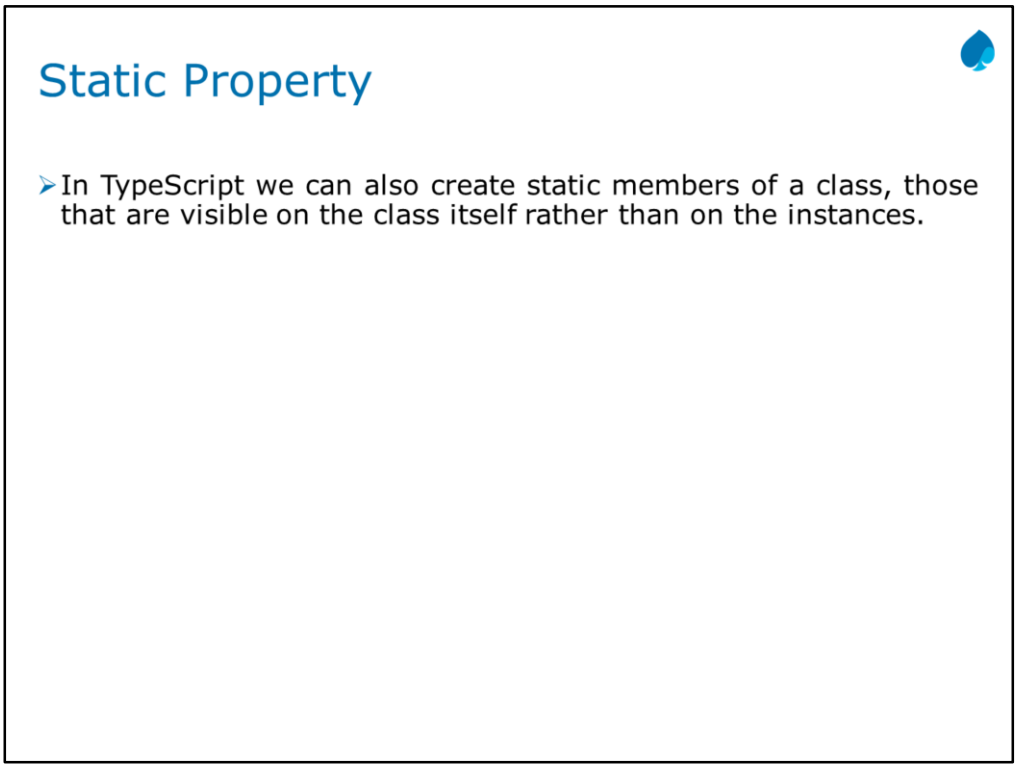
2.**private**

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**

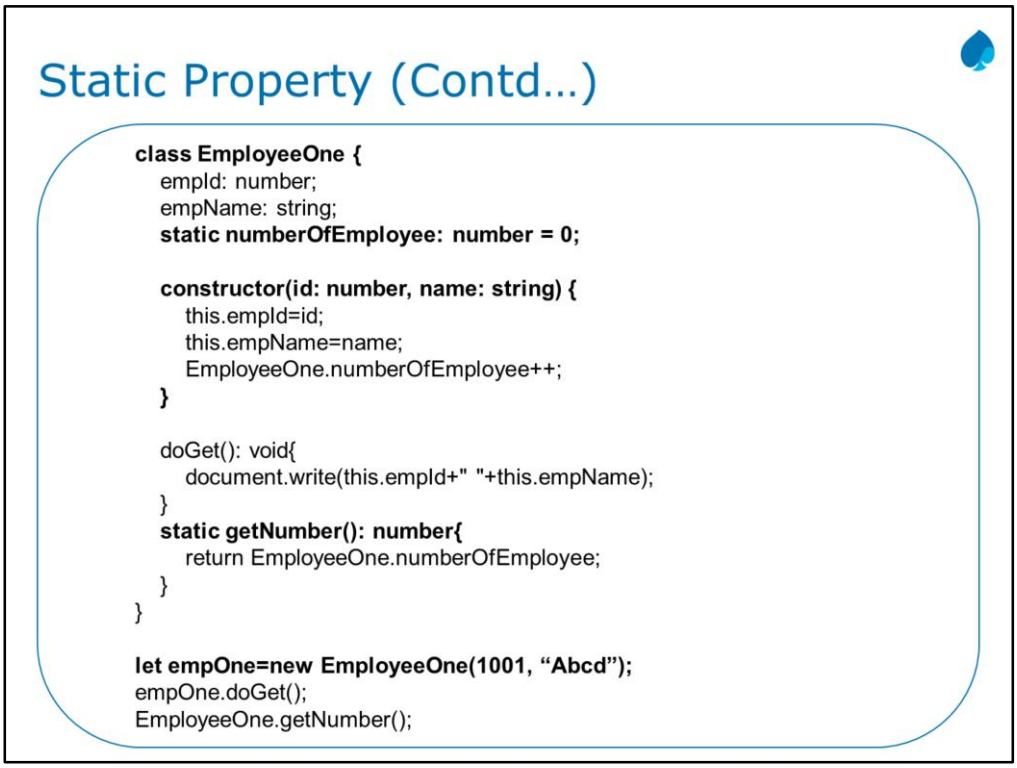
A protected data member is accessible by the members within the same class as that of the former and also by the members of the child classes.

Instructor Notes:



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.

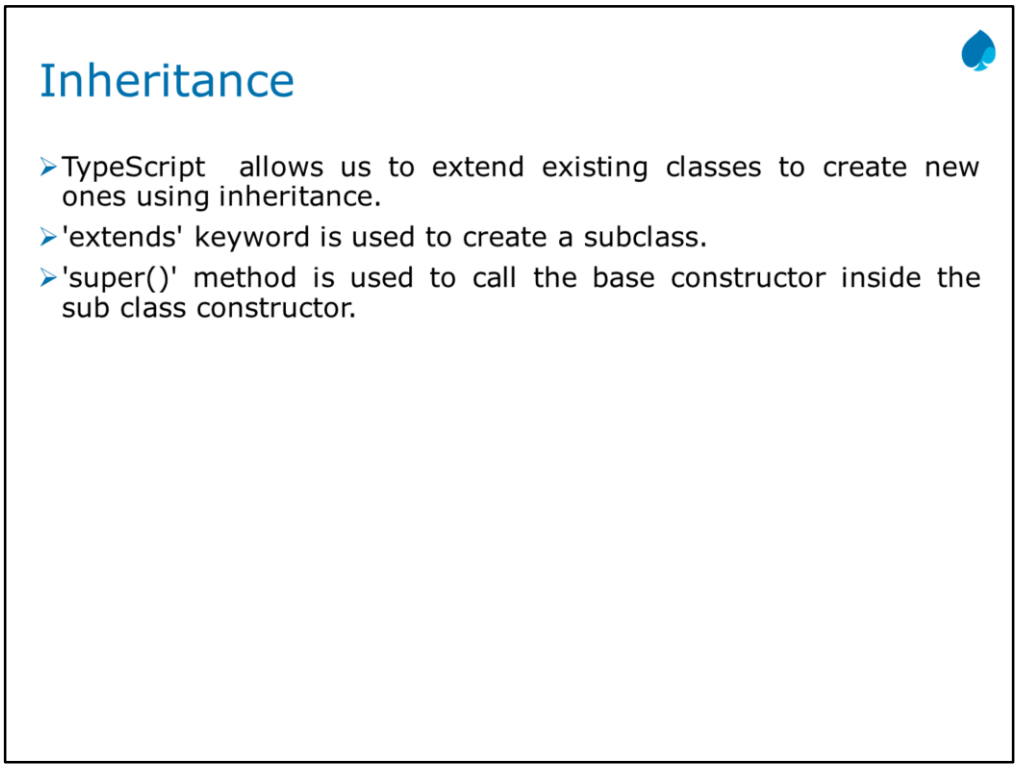
Instructor Notes:



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:

Add instructor notes here.



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("Galloping...");
      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.

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”;
```

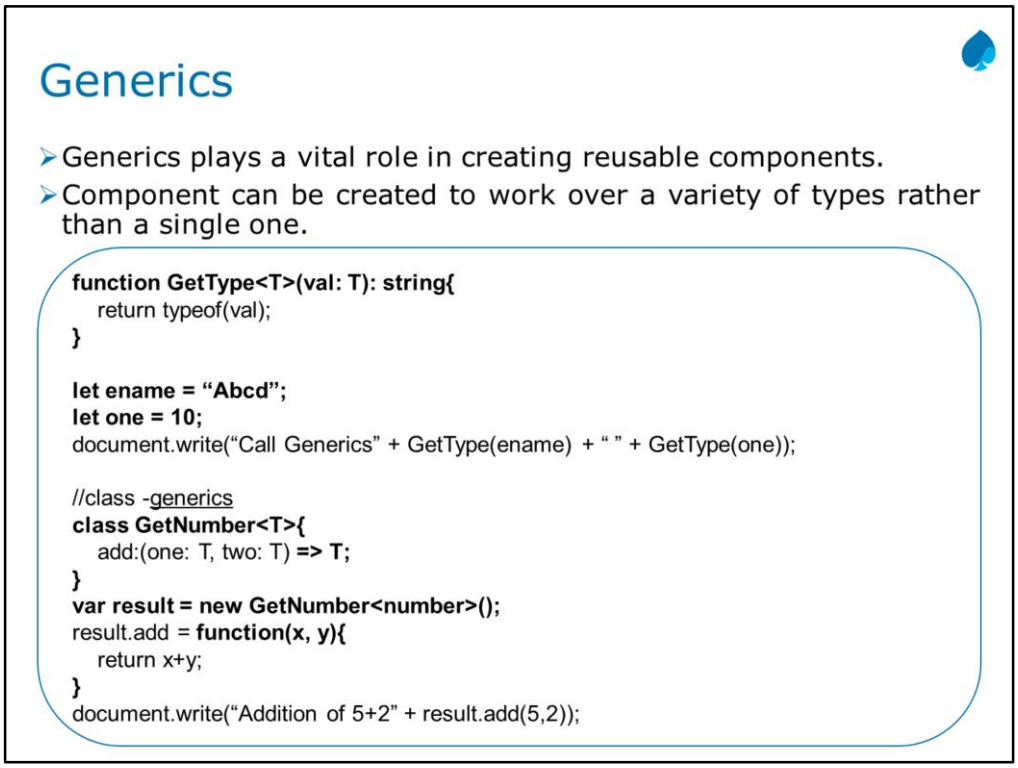
```
D:\AllDemoAngular\TypeScriptModule>tsc Demotemplatestring.ts

D:\AllDemoAngular\TypeScriptModule>node Demotemplatestring.js
Hello, TypeScript! Your name has 10 characters
```

Code Snippet

Instructor Notes:

Add instructor notes here.



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.

Modules

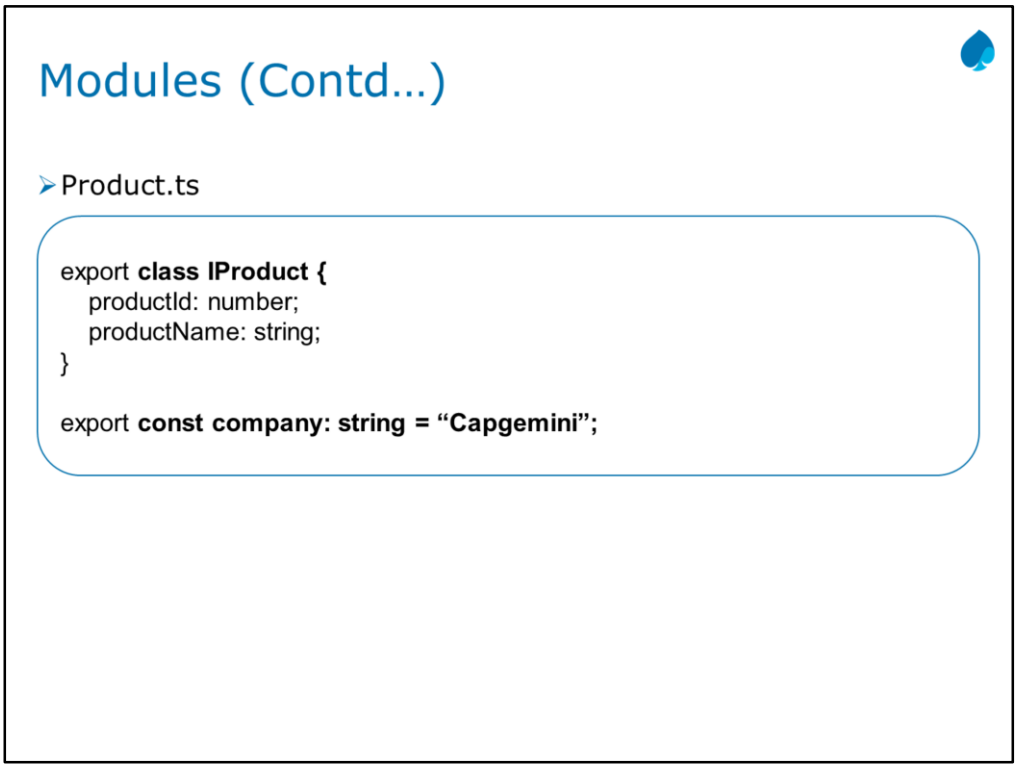
- Starting with the ECMAscript 2015,javascript has a concept of modules Typescript shares this concept.
- Modules are executed within their own scope, not in the global scope; this means that variables, functions, classes etc. declared un a module are not visible outside the module unless they are explicitly 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 doing with these below syntax

```
export { StudentInfo }
```

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

Instructor Notes:



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

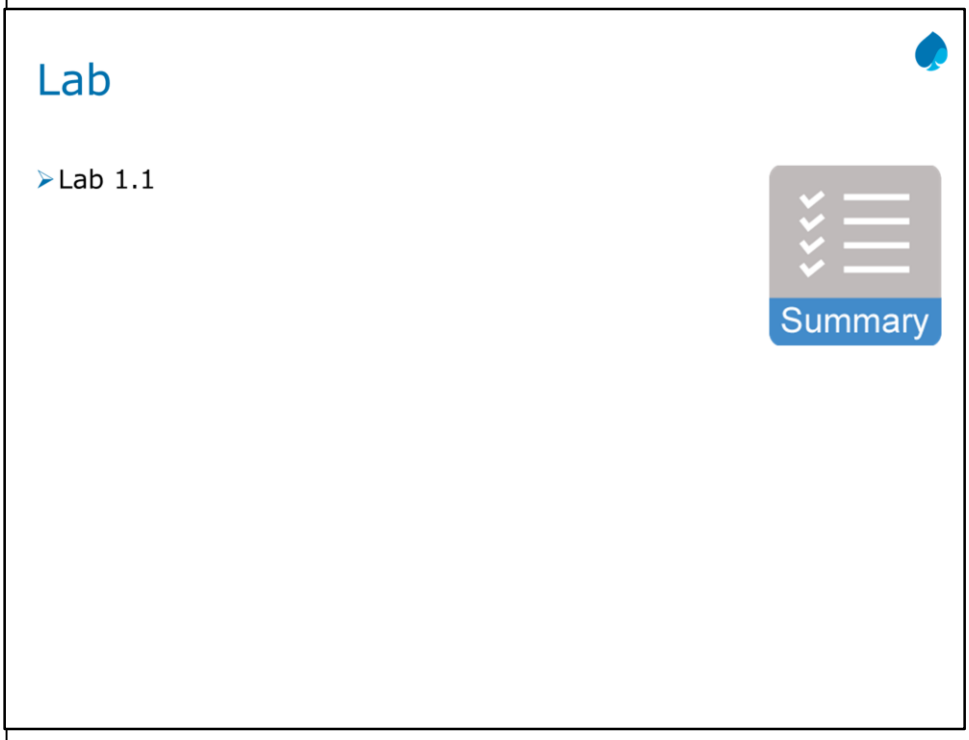
Instructor Notes:

Demo

- TypeScript Demo
- Typescript Module Demo

Instructor Notes:

Add instructor notes here.



Add the notes here.