Agenda:

- What is Ambient Declaration?
- How to integrate JavaScript functions, variables to TypeScript

What is Ambient Declaration?

Ambient Declaration is the process of providing declarations for user JavaScript as well as for JavaScript third party libraries.

Third party libraries can be either **JQuery, AngularJS, Node JS, Require JS, Polymer JS** etc.

- User **need not rewrite** the third party bundles to make use of them in typescript with type-safe.
- Provides **Intellisense** for third party bundles. This is really relaxing that when we press **(.)** it shows all members
- All the ambient declarations will be written in a declaration file with an extension **(.d.ts)**, where **'d'** denotes the declaration.

  Ex: sample.js, sample.d.ts
- We can tell the typescript that the code we are trying to reuse is existed at some other place using **declare** keyword.
- It's not mandatory that declarations must be in a declaration file with extension **(.d.ts)** it can be either in **(.ts / .d.ts)**, but it's **recommended** to separate the declarations in a separate file.
- If a file has the extension .d.ts then each **root level** definition must have the **declare** keyword prefixed to it, so that the programmer can make sure that declared items will exist at run time.

How to integrate JavaScript functions, variables to TypeScript

Step1: Create a javascript file

MyLibrary.js:

```
var Profile = {
   emp: (function () {
      function emp(name, sal) {
         this.EmpName = name;
         this.EmpSalary = sal;
      }
      return emp;
   })(),

   sayHello: function (message) {
      return "Hello " + message;
   }
};
```

Here we have implemented two functions **sayHello** and **emp** under **Profile.** Remember that it's recommended to separate the implementation into (.js) and declaration into (.d.ts).

Step2: Create typescript file with extension **(.d.ts)**

MyLibrary.d.ts:

```
declare module MProfile {
   export interface IEmployee {
      EmpName: string;
      EmpSalary: number;
      new (name: string, sal: number): IEmployee;
   }
   export interface Main {
      emp: IEmployee;
      sayHello(msg:string): string;
   }
}
//after declaring module root level interface is declared as Profile. Where this declaration variable must be same as the root variable in your library.
// in this example "Profile" is the root variable in library.js, so we are declaring that root variable in this declaration file with the same name given.
declare var Profile: MProfile.Main;
```

Step3: Create your client typescript file with extension **(.ts)** and give the reference of declaration file.

Main.ts:

```
/// <reference path="mylibrary.d.ts" />
var msg = Profile.sayHello("Deccansoft");
console.log(msg);
console.log("List of Employees");
var emp1 = new Profile.emp("Phani", 15000);
console.log("Name= " + emp1.EmpName + ",Salary= " + emp1.EmpSalary);
var emp2 = new Profile.emp("Charan", 20000);
console.log("Name= " + emp2.EmpName + ",Salary= " + emp2.EmpSalary);
```

Step4: Create an html file and refer both the script files in the same order as given.

Sample.html:

```
<!DOCTYPE html>
<html>
<body>
```

```
<script src="mylibrary.js"></script>
<script src="main.js"></script>
</body>
</html>
```

Output: