Deloitte.

TypeScript

Deloitte Technology Academy (DTA)

Agenda

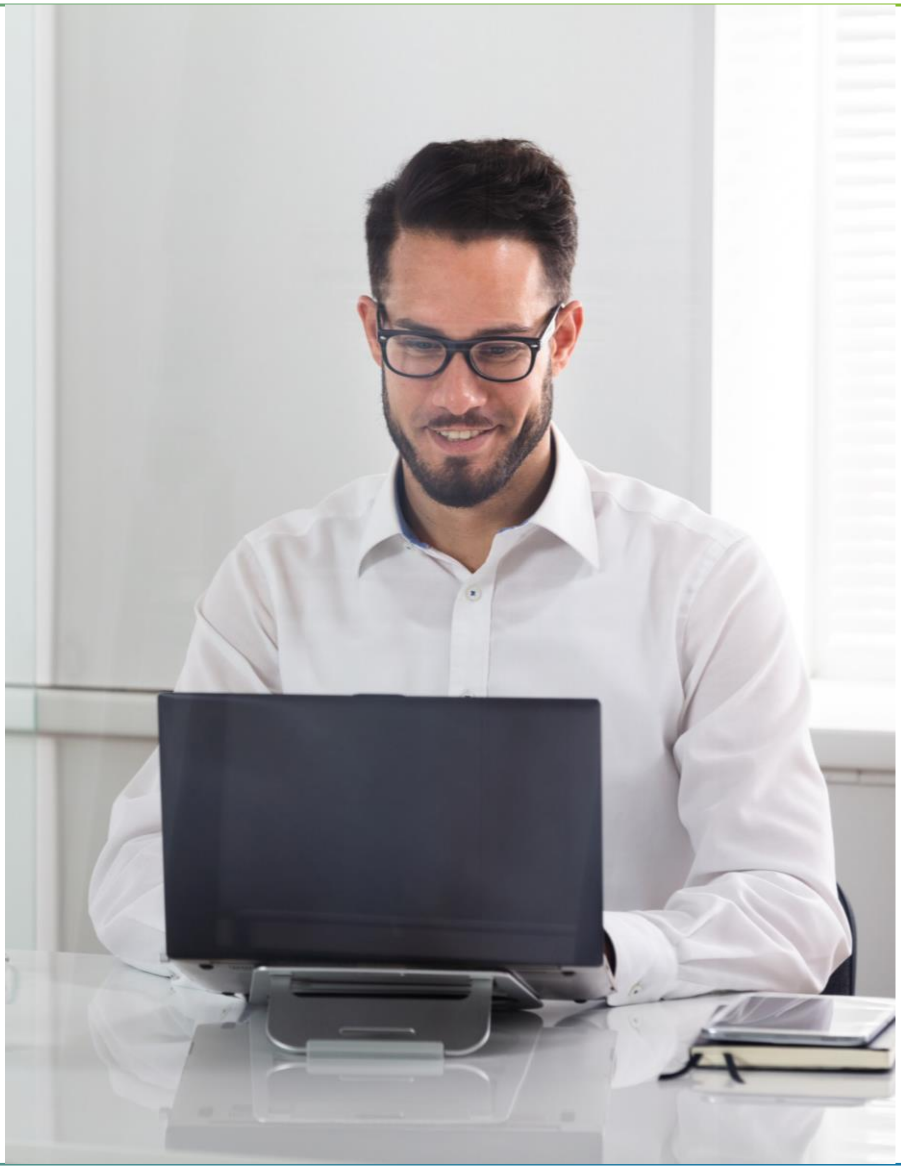
Topic	Description	Duration
Overview of Typescript	TypeScript Syntax, The Type System, Types in TypeScript, Setting Up Context, Decorators, TypeScript Configuration, TypeScript Transpilation, Generics, Working with Modules, Parameter and Return Value Types, Class Constructors, and Interfaces and Abstract Classes	4 hours

Learning Objectives

By the end of this session, you will be able to:

- Explain TypeScript and its syntax
- Describe type systems, decorators, and generics
- Explain parameters and constructors
- Describe interfaces and abstract classes in TypeScript

Overview of Typescript



Overview of Typescript

- Microsoft develops and maintains Typescript which is an object-oriented, open-source language.
- Extends JavaScript by adding data types, classes, and other object-oriented features with type-checking
- Plain JavaScript is compiled in this typed superset of JavaScript.

Strongly typed programming language

Features from future JavaScript

Built on JavaScript

Better tooling

An optional type system for JavaScript

</>

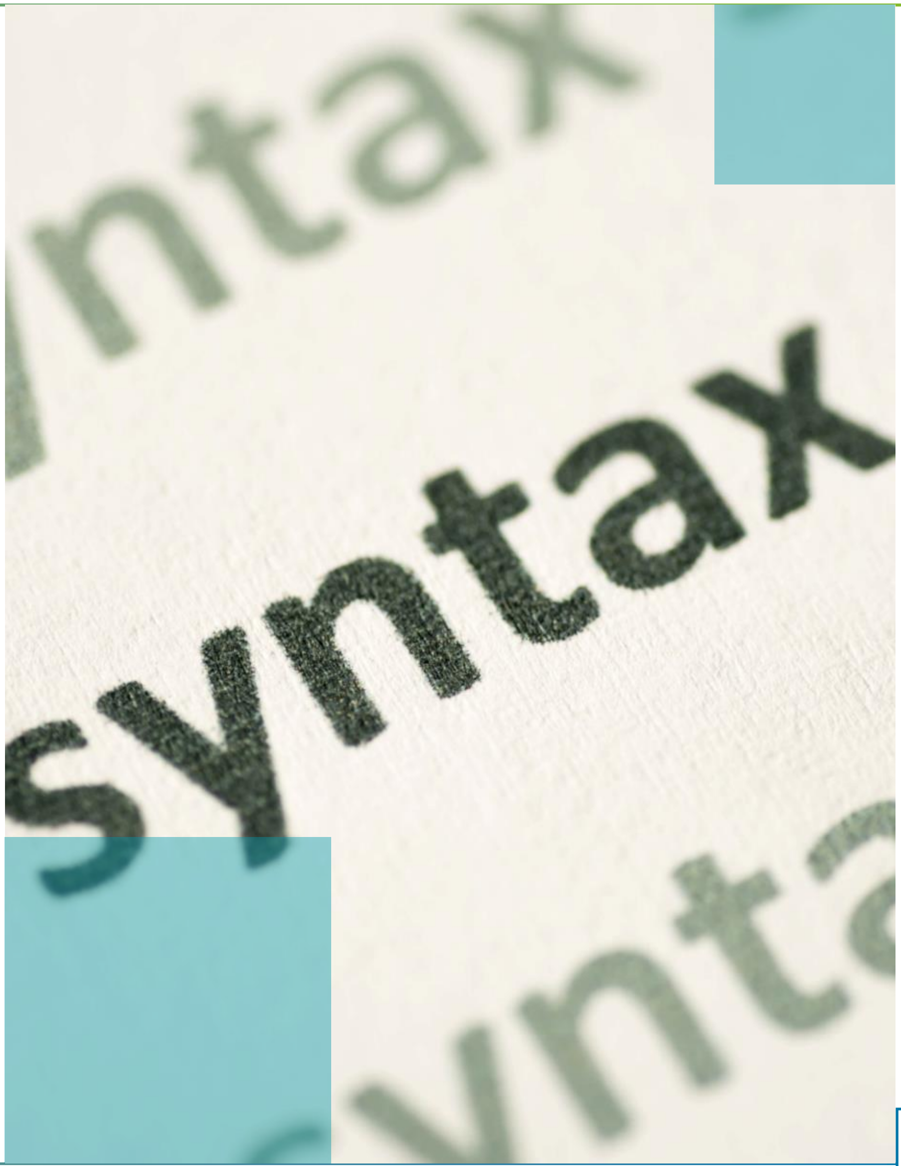
TypeScript Syntax

A TypeScript program is composed of the following:

- **Modules**
- **Functions**
- **Variables**
- **Statements and Expressions**
- **Comments**

TypeScript Syntax (Cont.)

- You can add type declarations to variables, function parameters, and function return types.
- The type is written following a colon after the variable name, for example: var num: number = 5;
- Where possible, the compiler will check the types during compilation and report type errors.
- Basic types
  - number (both integers and floating point numbers)
  - string
  - boolean
  - array—The array's element types can be specified. Array types can be defined in two equivalent ways: `Array<T>` and `T[]`. For example:
    - `number[]` - array of numbers
    - `array<string>` - array of strings

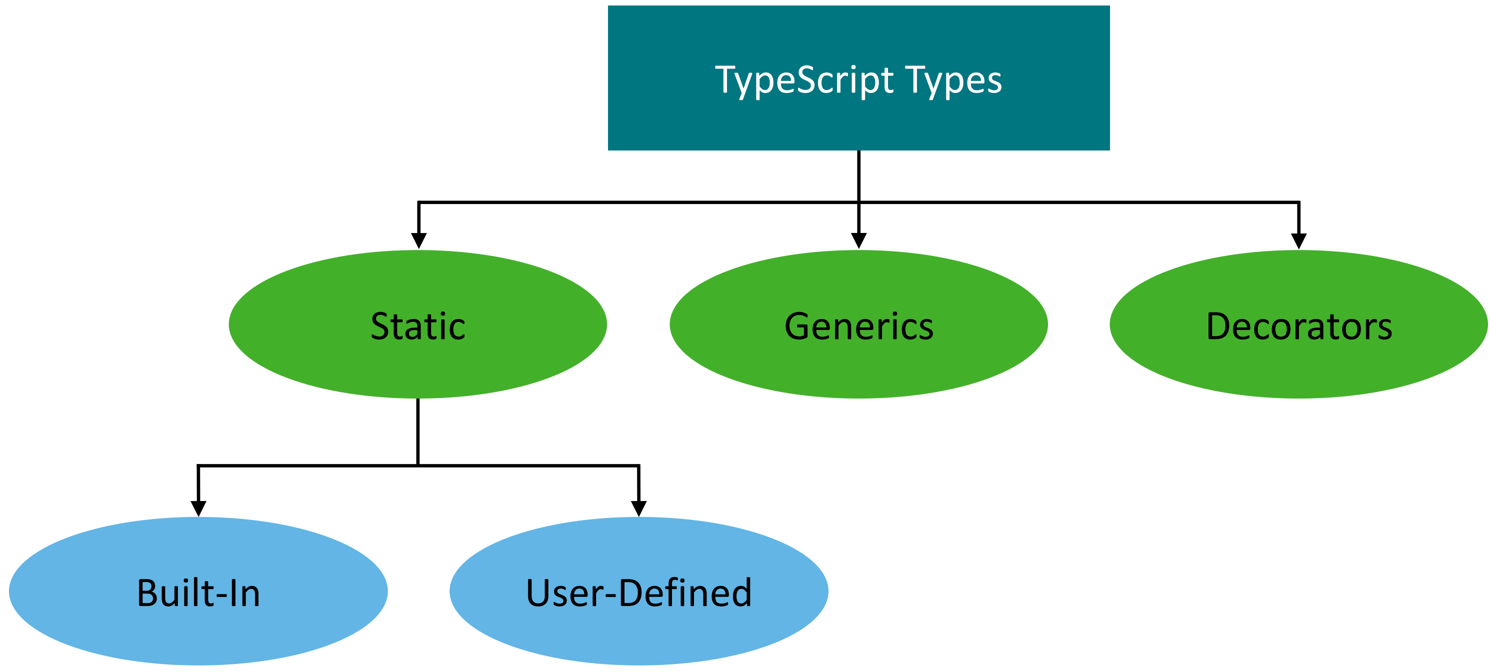


The Type System

- Represents the different types of values supported by the language
- Before the program can store or manipulate the supplied values, they are checked for validity to ensure the code behaves appropriately.
- Allows for richer code hinting, and automated documentation too
- The compiler checks all variables and expressions against their types and removes all type information when it converts the code into valid JavaScript

Types in TypeScript

An optional type system is provided by TypeScript for data types. We can classify the TypeScript data type as given below.



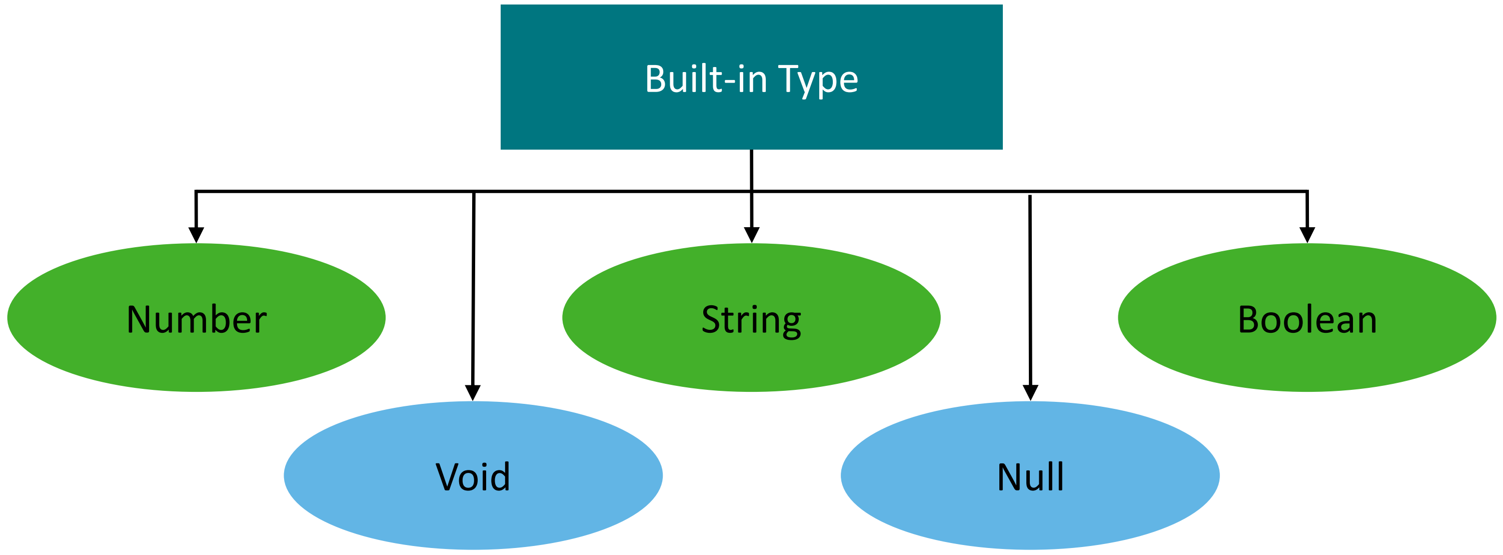
Types in TypeScript (Cont.)

Static Types

"Without running a program" or "at compile time" mean static types in context of type systems.

Built-in or Primitive Type

This TypeScript has five built-in data types as given below.



Types in TypeScript (Cont.)

The Any Type:

In TypeScript any data type is the supertype of all types. It denotes a dynamic type. Using any type is equivalent to opting 'out of type' checking for a variable.

Null and Undefined	Generic	Decorators
• The data type of a variable cannot be referenced by null and undefined, but it can be assigned as values to a variable. • The null and undefined are not the same. A variable that has no object or value assigned to it when initialized is undefined while null is defined as a variable that has been set to an object whose value is undefined.	Generic is used to create a component that can work with a variety of data types not just a single one. A way to create reusable components is allowed.	A decorator is a special data type that can be attached to a class declaration, accessor, property, method, and parameter. It provides classes and functions a way to add both annotations and a meta-programing syntax. It is used with"@" symbol.

Setting Up Context

- To create a TypeScript project with create-react-app, you need to add the flag—template typescript, otherwise, the app will only support JavaScript.
- Sharing and managing states across your components without passing down props is provided by React Context.
- Only the components who need to consume the data will be provided with it by the context. An example to demonstrate Context with React is shown here.

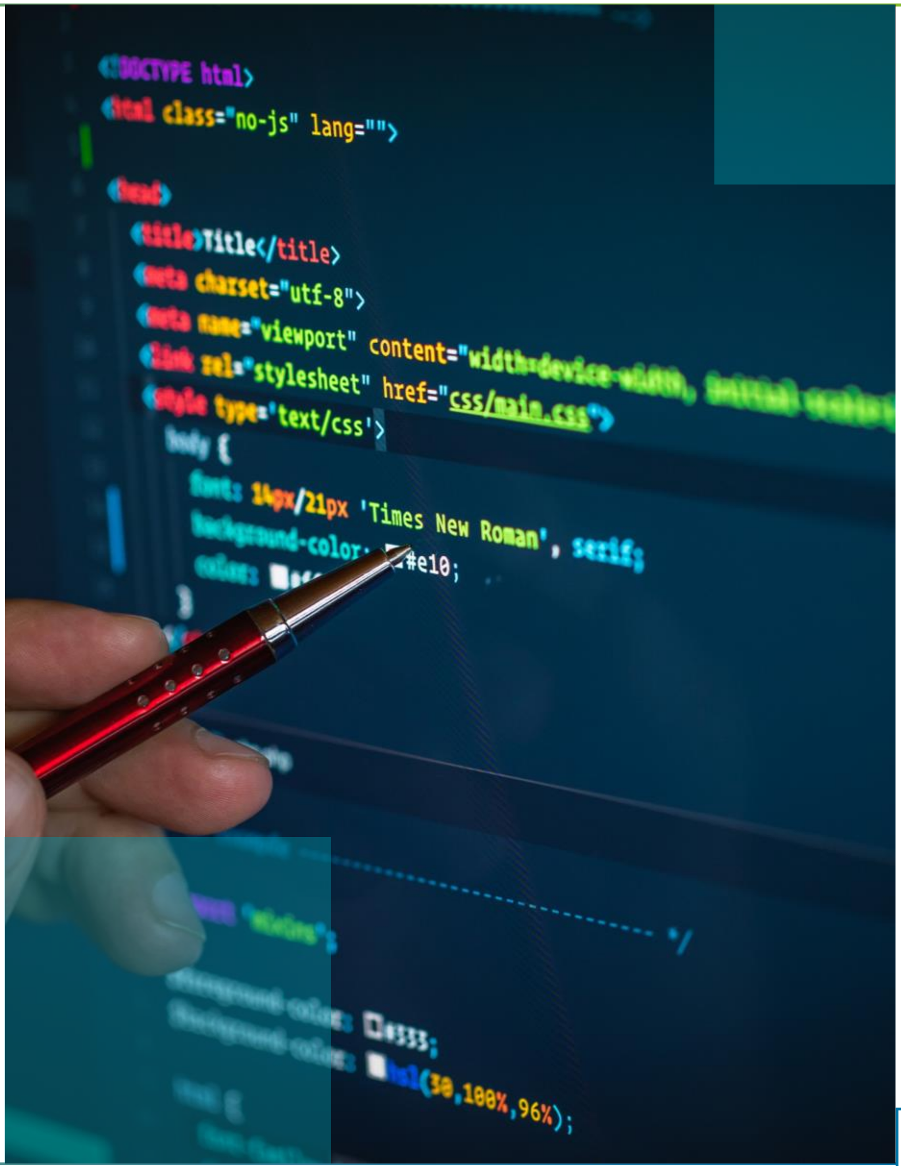
```
import { createContext } from "react";
interface AppContextInterface {
  name: string;
  author: string;
  url: string; }
const AppCtx = createContext<AppContextInterface | null>(null);
// Provider in your app
const sampleAppContext: AppContextInterface = {
  name: "Using React Context in a Typescript App",
  author: "Deloitte",
  url: "http://www.deloitte.com", };
export const App = () => (
  <AppCtx.Provider
value={sampleAppContext}>...</AppCtx.Provider> );
// Consume in your app
import { useContext } from "react";
export const PostInfo = () => {
  const appContext = useContext(AppCtx);
  return (
    <div>
      Name: {appContext.name}, Author: {appContext.author},
Url:{" "}
      {appContext.url}
    </div>
  );
};
```

Decorators

- Are an experimental TypeScript feature that gives us a clean syntax for metaprogramming with classes, class methods, properties, and method parameters
- Are syntax for calling a function on the thing you are decorating

Generics

- A placeholder type used to enforce a type-level constraint in multiple places
- Also known as a polymorphic type parameter
- The generic types declared within the triangle brackets: `<T>`
- Constraining the generic types is done with the extends keyword: `<T extends Car>`



TypeScript Transpilation

- Transpiling means converting one higher-level language to another higher-level language. TypeScript is a high-level language but after it is transpiled, it will turn into JavaScript (another high-level language).
- Part of this transpiling process is called type erasure.
- Type erasure is quite simple when all the types get removed from the TypeScript code as it is transpiled to JavaScript.

TypeScript code that looks like this:

```
let name: string = 'abc';
```

Eventually gets compiled/transpiled to this:

```
let name = 'abc'
```

Working With Modules

- The default global scope of the code can be prevented through TypeScript provided modules and namespaces, which can also be utilized to maintain and organize a large code base.
- A local scope in the file can be created by using modules. So, all variables, classes, and functions, etc. and not accessible outside of the module once they are declared in a module.
- Using the keyword ‘export’ can create a module and the keyword 'import' can allow a module to be used in another module.

Parameter Types

In TypeScript, if a user does not provide an assigned value for a parameter, we can provide one, or if the user passes ‘undefined’ in its place.

Rest Parameters:

- To easily accommodate ‘n’ number of parameters, rest parameters were introduced by TypeScript.
- Rest parameters can be used when the number of parameters that a function will receive can vary or are unknown. The "arguments" variable can achieve this in JavaScript.
- However, an ellipsis (...) denotes the use of rest paramenter in TypeScript.
- The rest parameter can be passed zero or more arguments.
- The rest parameter name that we provide will allow the compiler to create an array of arguments.

Example:

```
let Greet = (greeting: string, ...names:
string[]) => {
    return greeting + " " + names.join(", ")
+ "!";
}


Greet("Hello", "abc", "xyz"); // returns
"Hello abc, xyz!"


Greet("Hello");// returns "Hello !"
```

Return Value Types

- A function’s type has the same two parts—type of the arguments and the return type.
- We write out the parameter types giving each parameter a name and a type.
- Regardless of what the parameters in a function type are named, they will be considered valid types for the function if the parameter types line up.
- The return type is the second part, which is made clear by using an arrow (=>) between the parameters and the return type.

Example:

```
let myAdd = function (x: number, y: number):
number {
  return x + y;
};
```

Class Constructors

- Parameters can be added with type annotations, default values, and overloads, very similar to functions.
- There are just a few differences between class constructor signatures and function signatures
- Cannot have type parameters—these belong on the outer class declaration
- Cannot have return type annotations—the class instance type is what’s always returned

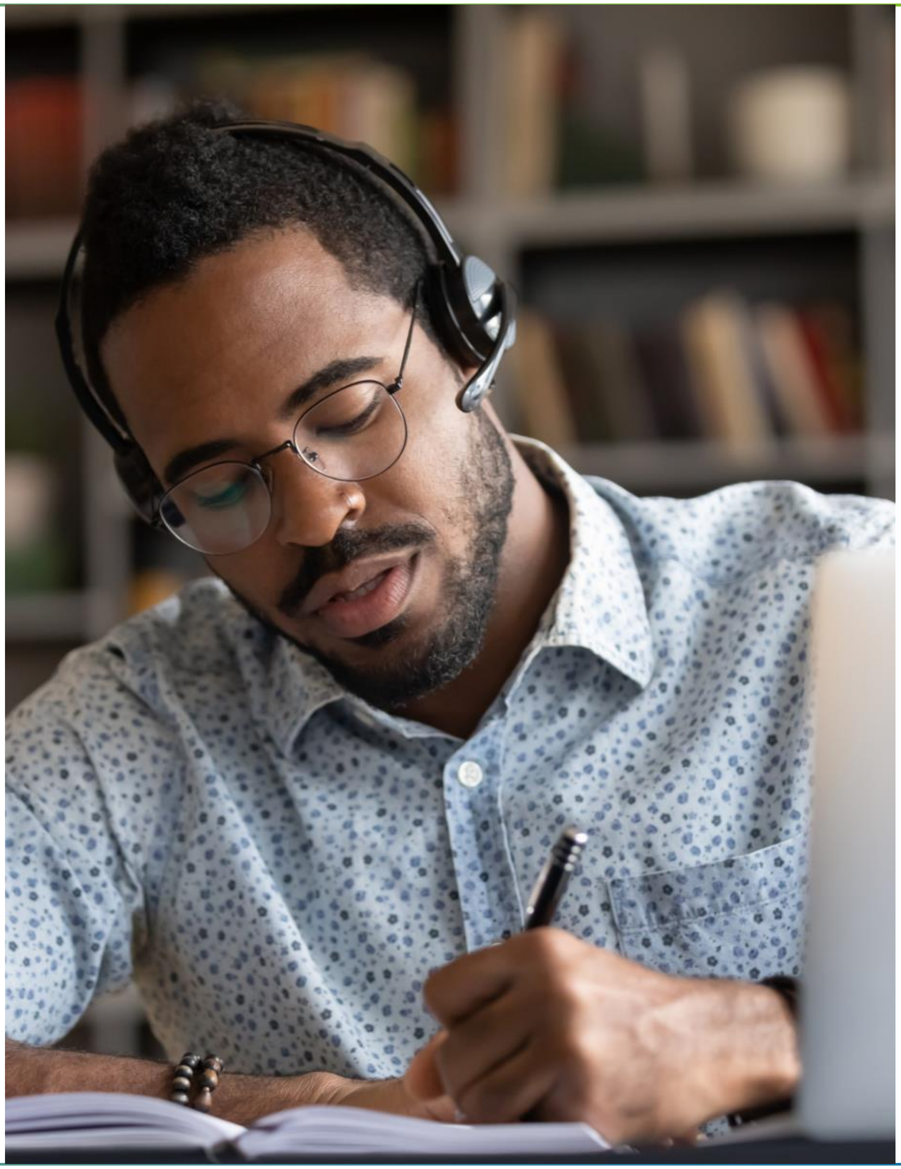
```
class Point {
  x: number;
  y: number;
  // Normal signature with defaults
  constructor(x = 0, y = 0) {
    this.x = x;
    this.y = y;
  }
}
```

Interfaces and Abstract Classes

A TypeScript abstract class is a class that may have some unimplemented methods. These methods are called abstract methods. An instance of an abstract class cannot be created.

Interface	Abstract Class
• All members are abstract. • Interfaces support multiple inheritances. • TypeScript interface has zero JavaScript code which means it is only available in TypeScript and does not produce any code in compiled JavaScript files. • Interfaces can be implemented by any class making them generic in nature. For instance, IClone interface can be implemented by any class such as business objects, or database classes.	• Some members are fully implemented and others are abstract. • Abstract class does not support multiple inheritances. • Abstract class compile to JavaScript functions. • Abstract classes are related. For example, if ViewModelBase is abstract, then we know this class will only inherit by ViewModels.

Hands-On Labs



Hands-On Activity 1

Activity Details	
Problem Statement	Write a function that returns the string "Hello, World!".

Summary

The key learning points of the module.

- By using type-checking to add data types, classes, and other object-oriented features TypeScript is able to extend JavaScript.
- The type is written after a colon following the variable name, like this: `var num: number = 5;`
- Before supplied values are stored or manipulated by the program, they are checked for validity by the type system.
- Decorators are syntax for calling a function on the thing you are decorating
- The generic types declared within the triangle brackets: `<T>`
- Transpiling means converting one higher-level language to another higher-level language. TypeScript is a high-level language but after it is transpiled, it will turn into JavaScript

Thank You

Deloitte.