

**Introduction to TypeScript**

**Course Number:** SCRPT-194
**Duration:** 2 days

**Overview**

This Introduction to TypeScript training course teaches attendees the fundamentals of the TypeScript language. Discover the powerful and expressive type system that TypeScript provides as well as TypeScript tooling (including the type checker, editor plugins, compiler, and more).

**Prerequisites**

* Proficiency/experience in at least one modern object-oriented programming language, preferably a statically-typed language (e.g., C# or Java)
* Previous experience with JavaScript either as a frontend web developer or backend Node.js developer

**Materials**

All TypeScript training attendees receive comprehensive courseware.

**Software Needed on Each Student PC**

* Google Chrome and/or Firefox
* Other modern browsers as desired
* IDE/development environment of your choice
* Other free software and lab files that Accelebrate would specify

**Objectives**

* Write and deploy type-safe server and client-side applications using TypeScript
* Interface with existing TypeScript libraries
* Use TypeScript’s powerful type system to improve existing JavaScript code and increase type safety
* Understand warnings and errors reported by the TypeScript compiler/type-checker (tsc) and take action to resolve them
* Go beyond basic type annotations, instead learning to represent real-world application state in a type-safe way

**Outline**

* Introduction
* Basic Types and Variable Declarations
* Primitives and Non-Primitives
* Typing Function Arguments and Return Types
* Basic Type Inference
* Generics and Type Parameters
* Any
* Unions
* More Advanced Generics
* Any vs. Unknown
* Type Guards and Narrowing
* Interfaces vs. Type Aliases
* Intersection Types
* Modules and Exporting Type Aliases
* Ways to Break Type Guarantees
	+ any
	+ foo as Bar
	+ Function or Object or object in certain cases
	+ Non-null assertion operator foo!.bar
* Using as to tell TypeScript “trust me”
* Discriminated unions (tagged unions)
* Creating Custom Type Guards
* Literal Types
* Enums and Const Enums
* Building a Validator using Custom Type Guards
* Using Runtypes or Zod to Validate Data
* Declaration Files and DefinitelyTyped Repository of Community Types
* Recursive Types
* Typing JSON
* Utility Types
* Generics with Constraints
* Advanced Types (typeof, keyof)
* Making a Property Read-only
* Conditional Types
* Making a Type or Interface Read-Only (the hard way)
* Runtime Manifestation (type position vs. value position)
* Mapped Types (using a key in to modify an object type or interface)
* ReadonlyArray (and the read-only modifier for object properties)
* Declaration Merging
* Adding Types to Existing Code
* Declaration Files
	+ Shipped with the package from the library author
	+ Decl Files written by the community
	+ Write your own and put it in global.d.ts
	+ If it is your own library, then you can add JSDoc annotations, then you don’t need a d.ts file
	+ If you have a JS file with JSDoc annotations then you can auto-generate the d.ts for export (such as if you’re a library author)
* Conclusion