To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Advanced TypeScript

Class Duration

21 hours of live training delivered over 3 days.

Student Prerequisites

- Completion of TypeScript Essentials or equivalent professional TypeScript experience
- Comfortable writing typed functions, interfaces, classes, and basic generics
- Working knowledge of modern JavaScript (ES2020+)
- Familiarity with npm-based project tooling

Target Audience

This course is designed for working TypeScript developers who want to move beyond everyday application typing into full command of the type system. It suits library authors, platform and tooling engineers, tech leads who review type-heavy code, and senior application developers responsible for shared types, build configuration, and compiler performance across large codebases.

Description

Advanced TypeScript takes experienced TypeScript developers deep into the type system and the compiler itself. The course begins with generics mastery (constraints, defaults, inference behavior, and `const` type parameters), then builds up the full type-level programming toolkit: conditional types with `infer`, mapped types with key remapping, template literal types, discriminated unions with exhaustiveness checking, type predicates, assertion functions, and variance. Throughout, the emphasis is on practical patterns: designing APIs whose types guide callers toward correct usage and catch real bugs at compile time. The second half turns to TypeScript at scale: authoring and publishing typed libraries with well-crafted declaration files for npm and JSR, configuring `tsconfig` with confidence including project references for monorepos, and diagnosing slow compilations with the compiler's performance tooling. The course covers the TypeScript 7 native compiler, currently in beta as a Go-based port of the compiler and language

service that delivers dramatically faster builds and editor responsiveness, including what changes for existing projects and how to evaluate it today. Students also learn type-level testing with expect-type and Vitest, and develop judgment about when type-level programming stops paying for itself.

Learning Outcomes

- Design generic functions and types using constraints, defaults, and `const` type parameters with predictable inference.
- Build conditional types that use `infer` to extract and transform types, and create mapped types with key remapping to derive new shapes from existing ones.
- Apply template literal types to model string-based APIs such as routes and event names.
- Model domain logic with discriminated unions, enforce exhaustiveness at compile time, and write type predicates and assertion functions that narrow types safely at runtime boundaries.
- Explain covariance and contravariance and predict how TypeScript checks function and container assignability.
- Author declaration files, publish well-typed libraries to npm and JSR, and configure `tsconfig` deliberately, including strictness flags, module settings, and project references.
- Diagnose and fix slow type checking using compiler performance diagnostics, and evaluate the TypeScript 7 native compiler beta to plan a migration path for builds and editor tooling.
- Test types themselves using expect-type assertions in a Vitest suite, and recognize when type-level complexity exceeds its value and choose simpler designs.

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

- Node.js 24 LTS (or current Node.js 22+)
- Visual Studio Code or another TypeScript-aware editor
- Permission to install npm packages and editor extensions
- Git for version control

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- A free GitHub account for lab repositories

Training Topics

Generics Mastery

- Generic functions, classes, and interfaces revisited
- Constraints with `extends` and constraint chaining
- Generic parameter defaults
- How inference selects type arguments
- `const` type parameters and literal inference
- Overloads versus generics: choosing the right tool

Conditional Types and infer

- Conditional type syntax and evaluation
- Distributive conditional types over unions
- Extracting types with `infer`
- Recursive conditional types
- Built-in utility types reimplemented from scratch
- Practical patterns: unwrapping promises, function parameters, and return types

Mapped Types and Key Remapping

- Mapped type fundamentals and modifiers (`readonly`, `?`)
- Key remapping with `as`
- Filtering keys with `never`
- Combining mapped and conditional types
- Deriving DTO, patch, and form types from domain models

Template Literal Types

- Template literal type syntax
- Intrinsic string manipulation types (`Uppercase`, `Capitalize`, and friends)
- Pattern matching strings with `infer`
- Typed event emitters and route parameters
- Limits and compile-cost considerations

Discriminated Unions and Exhaustiveness

- Designing discriminants and tagged unions
- Narrowing with `switch` and control-flow analysis
- Exhaustiveness checking with `never`

- Unions versus class hierarchies
- Modeling state machines with unions

Type Predicates and Assertion Functions

- User-defined type guards with `is`
- Assertion functions with `asserts`
- Inferred type predicates
- Validating unknown data at runtime boundaries
- Combining guards with discriminated unions

Variance and Assignability

- Structural assignability rules
- Covariance and contravariance in plain terms
- Function parameter bivariance and `strictFunctionTypes`
- `in`/`out` variance annotations
- Why mutable containers are invariant

Declaration Files and Publishing Typed Libraries

- Anatomy of `.d.ts` files
- Generating declarations with `declaration` and `declarationMap`
- Packaging for npm: `exports`, `types`, and dual ESM/CJS concerns
- Publishing to JSR with source-level types
- Versioning types without breaking consumers

tsconfig Deep Dive and Project References

- Strictness flags and what each actually checks
- `module`, `moduleResolution`, and `target` in modern projects
- TypeScript 6 defaults and what changed
- Project references and incremental builds
- Structuring a monorepo with composite projects

Compiler Performance Diagnostics

- `--extendedDiagnostics` and `--generateTrace`
- Finding expensive types and instantiation explosions
- `skipLibCheck`, isolated declarations, and build hygiene
- Editor responsiveness and language-server tuning

The TypeScript 7 Native Compiler

- Project Corsa: the Go-based port of the compiler and language service

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- Current status: the TypeScript 7 beta and the `@typescript/native-preview` package
- What stays the same: language behavior aligned with TypeScript 6
- What changes: build pipelines, APIs, and editor/LSP speedups
- Trying `tsgo` on an existing codebase and planning migration

Type-Level Testing

- Why types need tests
- expect-type assertions
- Type testing in a Vitest 4 suite
- Guarding public API types in CI

When Type-Level Programming Goes Too Far

- Compile-time cost versus correctness payoff
- Readability, onboarding, and error-message quality
- Refactoring clever types toward simpler designs

