TypeScript Quick Bootcamp — 1 day learning + ½ day practical (12 hours total)

Goal: Give freshers a compact, practical crash-course in TypeScript so they can start using it confidently in real projects.

Format: 1 learning day (8 hours) + ½ practical day (4 hours). Every section includes *what to learn*, *why*, short examples, and actionable exercises.

Summary / Timeline

- **Day 1 (8 hrs)** — TypeScript fundamentals → types → functions → OOP → advanced types & tooling
- **Half Day (4 hrs)** — Practical: small connected project (convert or build + compile + lint + run)

DAY 1 — TypeScript Fundamentals (8 hours)

0. Warm-up: Why TypeScript? (15 min)

- Adds **static typing** to JS → catches many bugs at compile time.
- Better **editor/autocomplete** & maintainability.
- Works with existing JS (gradual adoption).

---

1. Basic Types (45 min)

What to learn

- Primitives: `string`, `number`, `boolean`, `bigint`, `symbol`, `null`, `undefined`
- `any` vs `unknown` vs `never` vs `void`

- Arrays: `string[]` or `Array<string>`
- Tuples: `[string, number]`
- Objects & type inference

Examples

```
let name: string = "Alice";
let age: number = 30;
let tags: string[] = ["ui", "backend"];
let coord: [number, number] = [40.7128, -74.0060];
let maybe: undefined | string = undefined;
```

Why: Know these to type variables correctly and avoid runtime surprises.

---

2. Object Types, Interfaces & Type Aliases (45 min)

What to learn

- `interface` vs `type`
- optional properties `?`
- readonly properties

Examples

```
interface User {
  readonly id: string;
  name: string;
  email?: string;
}
type Point = { x: number; y: number };
```

Why: Define clear contracts for objects used across code.

---

3. Functions & Signatures (45 min)

What to learn

- typed params & return types
- optional/default params
- rest & spread
- function types (callbacks)

Examples

```
function add(a: number, b: number): number { return a + b; }
const greet = (name: string = "Guest"): string => `Hi ${name}`;
type Mapper = (s: string) => string;
```

Why: Keep APIs explicit; help with refactoring.

---

4. Union, Intersection & Literal Types (30 min)

What to learn

- `union` types (`string | number`)
- `intersection` (`A & B`)
- literal types (`'small' | 'medium' | 'large'`)

Examples

```
type ID = string | number;
type Admin = User & { role: 'admin' };
type Size = 'S' | 'M' | 'L';
```

Why: Model flexible inputs precisely.

---

5. Enums & Const Assertions (20 min)

What to learn

- Numeric enums, string enums
- `as const` for literal inference

Examples

```
enum Role { User = "USER", Admin = "ADMIN" }
const COLORS = ['red','blue'] as const; // readonly tuple
```

Why: For expressive domain values (but prefer union string types often).

---

6. Generics (60 min)

What to learn

- Generic functions & interfaces
- Generic constraints
- When to use generics vs `any`

Examples

```
function identity<T>(arg: T): T { return arg; }
interface ApiResponse<T> { data: T; error?: string; }
function pluck<T, K extends keyof T>(obj: T, key: K) { return
obj[key]; }
```

Why: Write reusable, strongly-typed utilities (collections, API clients).

---

7. Utility Types (30 min)

Common built-ins

- `Partial<T>`, `Required<T>`, `Readonly<T>`, `Pick<T, K>`, `Omit<T, K>`, `Record<K, T>`, `ReturnType<T>`, `Parameters<T>`

Example

```
type EditableUser = Partial<User>;
type UserRecord = Record<string, User>;
```

Why: Save time for common type transformations.

---

8. Advanced Types — Conditional & Mapped Types (30 min)

What to learn

- `type Foo<T> = T extends X ? A : B`
- Mapped types to transform properties

Why: For library-level typing and complex transformations.

---

9. Classes, Inheritance & Access Modifiers (40 min)

What to learn

- `class`, `constructor`, `public/private/protected`, `readonly`, `static`
- `implements` & `extends`

Examples

```
class Person {
  constructor(public name: string) {}
  greet() { return `Hi ${this.name}`; }
}
class Employee extends Person { constructor(name:string, public
id:number){ super(name) } }
```

Why: Useful for OOP patterns in apps (services, models).

---

10. Modules & Namespaces (20 min)

What to learn

- `export` / `export default` / `import { } from './x'`
- default vs named exports
- module resolution basics

Why: Architect code across files.

---

11. Interop with JavaScript & Declaration Files (20 min)

What to learn

- `@types/` from DefinitelyTyped
- `declare module 'lib'` or `.d.ts` files
- using `require` with `esModuleInterop`

Why: Real-world projects mix JS libs with TS.

---

12. tsconfig & Compiler Options (20 min)

What to learn

- `strict`, `noImplicitAny`, `strictNullChecks`, `esModuleInterop`, `skipLibCheck`
- `rootDir`, `outDir`, `paths` (module aliases)

Why: Configure for safety and developer ergonomics.

---

13. Tooling: ts-node, ts-node-dev, Babel, ESLint (20 min)

What to learn

- `ts-node-dev` for local dev
- Build step: `tsc`
- Linting: `eslint` + `@typescript-eslint/parser` + plugin
- Format: Prettier

Why: Developer experience & CI setup.

---

14. Best Practices & Common Gotchas (20 min)

- Prefer `unknown` over `any`; narrow types via guards.
- Avoid implicit `any` (`noImplicitAny`).
- Use `strictNullChecks`.

- Prefer interfaces for public APIs; type aliases for unions.
- Beware of structural typing quirks.
- Keep code small, modular and add types gradually.

---

15. Mini Quiz & Recap (10–15 min)

- Quick 6-question quiz (type examples, fix a typed bug).

Half Day — Practical (4 hours): Connected Project + Exercises

We'll build a **small, connected project** that demonstrates real-world TS usage and ties multiple concepts together.

Project name: `contacts-manager` (CLI + simple module)
Goal: Build a typed contact manager library + small CLI to read/write JSON.

Project outline (steps + estimated time)

Setup (10 minutes)

- `npm init -y`
- `npm i -D typescript ts-node-dev @types/node`
- `npx tsc --init`
- Create `src/` and `src/index.ts`

Task 1 — Define Domain Types (30 min)

Create `src/types.ts`

```
export interface Contact {
  id: string;
  name: string;
  email?: string;
  phones: string[];
  tags?: string[];
}
```

Why: Practice interfaces, optional props, arrays.

Task 2 — Implement a ContactStore using Map + Generics (45 min)

Create `src/store.ts`

```
import { Contact } from './types';
export class ContactStore {
  private store = new Map<string, Contact>();

  add(contact: Contact) { this.store.set(contact.id, contact); }
  get(id: string) { return this.store.get(id); }
  list(): Contact[] { return Array.from(this.store.values()); }
  remove(id: string) { return this.store.delete(id); }
}
```

Why: Use Map for O(1) lookups; implement class with typed methods.

Task 3 — File I/O with Types & Safe Parsing (30 min)

Create `src/io.ts`

- Read / write `contacts.json` in project root using `fs/promises`.
- When parsing JSON, use type guard to ensure each entry is Contact (basic checks).

Example:

```
import fs from 'fs/promises';
import { Contact } from './types';
export async function loadContacts(path: string): Promise<Contact[]> {
  const raw = await fs.readFile(path, 'utf-8');
  const parsed = JSON.parse(raw);
  // simple guard
  if (!Array.isArray(parsed)) throw new Error("Invalid format");
  return parsed as Contact[];
}
```

Why: Practice async/await + typings for I/O.

Task 4 — CLI Script to Add/List/Remove (45 min)

Create `src/cli.ts` or extend `src/index.ts`:

- Accept command line args (`process.argv`) typed.
- Commands:
  - `npm run dev -- add --name "Alice" --email "a@x.com"`
  - `npm run dev -- list`
  - `npm run dev -- remove --id 12345`
- Use helper functions typed with parameter and return types.

Why: Practice function types, parsing CLI input, and assembling modules.

Task 5 — Compile, Run & Lint (20 min)

- `npm run dev` to run with `ts-node-dev`.
- `npm run build` (tsc) → `node dist/index.js` to run compiled output.

Add ESLint + Prettier if time:

```
npm i -D eslint @typescript-eslint/parser
@typescript-eslint/eslint-plugin prettier
```

Basic `.eslintrc` with plugin.

Deliverables

- `src/types.ts`, `src/store.ts`, `src/io.ts`, `src/index.ts` (or `cli.ts`), `contacts.json` sample
- README with commands to run

Acceptance Criteria

- TypeScript compiles without `--noEmit` errors under `strict` mode.
- CLI `list` prints a typed list.
- `add` writes to `contacts.json`.
- Runtime errors handled gracefully (try/catch).

Common Pitfalls & Tips

- **Avoid `any`**: use `unknown` then narrow.
- **Enable `strict`** in `tsconfig` early — fixes many issues.
- **Use `@types/`** for library typings (`npm i -D @types/lodash` etc).
- **Prefer union of string literals** to enums for small sets (`type Role = 'user'|'admin'`).
- **Use `as const`** for literal arrays when you need tuple types.

References (starter reading)

- W3Schools — TypeScript Tutorial (quick examples & basic topics)
- TypeScript Handbook — Official (deeper reference for advanced types)
  (Encourage trainees to read both: W3Schools for quick hits, Handbook for in-depth.)
