✅ Mastering TypeScript – Complete Guide + Roadmap (Final Version)

🧭 Part 1: TypeScript Roadmap – Learn Step by Step

🔑 1. Prerequisites

Before learning TypeScript, make sure you're comfortable with:

- JavaScript fundamentals (ES6+)
- Functions, Arrays, Objects, Promises
- Modules, Classes, and Inheritance

🔑 2. Getting Started

- Install: `npm install -g typescript`
- Initialize project: `tsc --init`
- Compile TS to JS: `tsc file.ts`
- Use with Node: `node file.js`

🔑 3. Learn Basic Types

- `string`, `number`, `boolean`, `null`, `undefined`
- Special: `any`, `unknown`, `void`, `never`
- Type inference & explicit annotations

🔑 4. Functions in TypeScript

- Function return & parameter types
- Optional/default/rest parameters
- Arrow functions with types

🔑 5. Working with Objects

- Object type declarations
- Arrays and Tuples
- Enums (basic and advanced)
- Literal types
- Union and Intersection types

🔑 6. Type Guards & Narrowing

- `typeof`, `instanceof`, `in`
- Discriminated unions
- Custom type guards

---

7. Interfaces & Types

- Interfaces and type aliases
- Optional & readonly properties
- Extending interfaces
- `type` vs `interface`

---

8. Classes & OOP

- Classes, Constructors, Inheritance
- Access modifiers: `public`, `private`, `protected`, `readonly`
- Implements, Abstract classes, Static methods

---

9. Generics

- Generic functions, interfaces, and classes
- Constraints using `extends`
- Default generics

---

10. Utility Types

- `Partial`, `Required`, `Readonly`, `Pick`, `Omit`, `Record`
- `Exclude`, `Extract`, `NonNullable`, `ReturnType`, `Parameters`

---

11. Advanced Types

- Mapped types
- Conditional types
- Template literal types
- `keyof`, `typeof`, `infer`, `in`

---

12. Modules & Code Organization

- ES modules: `import` / `export`
- Type-only imports
- Declaration files (`.d.ts`)
- Type-safe folder structure
- Module augmentation (advanced)

---

13. Real-World Usage

- React: JSX, props, hooks typing
- Node.js & Express: Request/Response types
- Working with APIs: Axios, fetch, and runtime validation (Zod/Yup)

🔑 14. Testing

- Jest/Vitest setup with TS
- Typing test cases
- Mocking with types

🔑 15. Decorators (Advanced/Optional)

- Class, method, and property decorators
- Enable in `tsconfig.json`:
  `"experimentalDecorators": true`

🔑 16. Build & Deploy

- Configure `tsconfig.json` for builds
- Bundle with `vite`, `webpack`, `tsup`, `esbuild`
- Emit declarations for libraries

🔑 17. Namespaces (Legacy but sometimes useful)

- Organize code internally before ES modules became popular
- Syntax: `namespace MyNamespace { ... }`

🔑 18. Type Assertions & Non-null Assertion

- Cast type: `let x = someValue as string;`
- Non-null assertion: `someValue!`

🧠 Outcome:

After this roadmap, you'll be able to:

- ✅ Build fully typed applications
- ✅ Prevent bugs before they happen
- ✅ Work efficiently in large teams
- ✅ Master both frontend and backend TypeScript

📚 Part 2: All TypeScript Concepts – Detailed Reference

1. Basic Types

```
let name: string = "John";
let age: number = 25;
let isActive: boolean = true;
let anything: any = "Can be anything";
let unknownVar: unknown = 42;
```

2. Type Inference vs Annotations

```
let count = 5;           // inferred as number
let price: number = 99.99;  // explicit annotation
```

3. Functions

```
function greet(name: string): string {
  return `Hello, ${name}`;
}

function log(msg: string): void {
  console.log(msg);
}

// Optional parameter
function greetOptional(name?: string): string {
  return `Hello, ${name ?? "Guest"}`;
}

// Default parameter
function greetDefault(name = "Guest"): string {
  return `Hello, ${name}`;
}

// Rest parameters
function sum(...nums: number[]): number {
  return nums.reduce((total, num) => total + num, 0);
}
```

4. Arrays & Tuples

```
let arr: number[] = [1, 2, 3];
let tuple: [string, number] = ["Age", 30];
```

5. Enums

```
enum Role {
  Admin,
  User,
  Guest
}

// String enums
enum Status {
  Active = "active",
  Inactive = "inactive"
}

// Heterogeneous enums (less common)
enum Mixed {
  No = 0,
  Yes = "YES"
}
```

6. Union & Intersection

```
let id: string | number;  // Union type: can be string or number

type Admin = { role: string };
type Employee = { department: string };
type Manager = Admin & Employee;  // Intersection type: must have both Admin and
Employee properties
```

7. Interfaces & Types

```
interface User {
  name: string;
  age: number;
  readonly id: string;
  isAdmin?: boolean;  // optional property
}

type Product = {
  name: string;
  price: number;
}
```

8. Classes

```ts
class Animal {
  constructor(public name: string) {}

  move(distance: number): void {
    console.log(`${this.name} moved ${distance}m.`);
  }
}

class Dog extends Animal {
  bark() {
    console.log("Woof!");
  }
}
```

9. Access Modifiers

- **public**: default, accessible anywhere
- **private**: accessible only inside class
- **protected**: accessible in class + subclasses
- **readonly**: cannot be changed after initialization

10. Generics

```ts
function identity<T>(value: T): T {
  return value;
}

const result = identity<number>(42);

// Generic classes:
class Box<T> {
  contents: T;

  constructor(value: T) {
    this.contents = value;
  }
}
```

11. Utility Types

Utility	Purpose
`Partial<T>`	All properties optional
`Required<T>`	All properties required
`Readonly<T>`	Make properties immutable
`Pick<T, K>`	Pick some keys from type

Utility	Purpose
`Omit<T, K>`	Omit some keys from type
`Record<K, T>`	Map keys to values
`Exclude<T, U>`	Exclude types from union
`Extract<T, U>`	Extract types from union
`NonNullable<T>`	Remove null and undefined
`ReturnType<T>`	Get function return type
`Parameters<T>`	Get function parameter types

```
interface User {
  id: number;
  name?: string;
  age?: number;
}

type PartialUser = Partial<User>;  // All properties optional
```

◈ 12. Advanced Types

Mapped Types

```
type ReadonlyUser = {
  [K in keyof User]: User[K];
};
```

Conditional Types

```
type Message<T> = T extends string ? string : never;
```

Template Literal Types

```
type Lang = "en" | "fr";
type Messages = `message_${Lang}`;
```

◈ 13. Type Guards

```ts
function isString(value: unknown): value is string {
  return typeof value === "string";
}
```

14. Declaration Files (.d.ts)

```ts
declare var GLOBAL_VERSION: string;
```

- Used to add types to existing JavaScript libraries.

15. Type Operators

- `keyof`: Get keys of a type
- `typeof`: Get type from a variable
- `in`: Used for mapping over keys in mapped types
- `infer`: Extract inner types in conditional types

16. Modules

```ts
// math.ts
export function add(a: number, b: number): number {
  return a + b;
}

// main.ts
import { add } from './math';
```

17. Module Augmentation (Advanced)

```ts
// Extending existing module declarations
declare module 'express' {
  interface Request {
    user?: string;
  }
}
```

18. Namespaces (Legacy)

```ts
namespace Utility {
  export function log(msg: string) {
    console.log(msg);
  }
```

```
}

Utility.log("Hello");
```

19. Type Assertions & Non-null Assertion

```
let someValue: unknown = "this is a string";
let strLength: number = (someValue as string).length;

// Non-null assertion
let elem = document.getElementById("id")!;
```

20. TypeScript with React

```
interface Props {
  name: string;
}

const Welcome: React.FC<Props> = ({ name }) => <h1>Hello, {name}</h1>;

// useState with type annotation
const [count, setCount] = useState<number>(0);
```

21. TypeScript with Node.js

```
import express, { Request, Response } from "express";

const app = express();

app.get("/", (req: Request, res: Response) => {
  res.send("Hello TS");
});
```

22. Decorators (Experimental)

```
function Logger(constructor: Function) {
  console.log("Logging...");
}

@Logger
class MyService {}
```

🔷 23. tsconfig.json Highlights

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "strict": true,
    "esModuleInterop": true,
    "outDir": "./dist",
    "rootDir": "./src",
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  }
}
```

🔷 24. Runtime Validation (Bonus)

Use libraries like Zod or Yup for schema validation and to generate TypeScript types.

🎯 Summary

This guide contains everything from beginner to advanced TypeScript concepts, practical usage patterns, and best practices.

Feel free to expand and customize this document as you grow your skills.

🚀 Happy TypeScripting!