TypeScript Project Guide

given in details topic wise typescript notes, in future I doing projrct in company, please given quick example with details notes

Here's a **detailed, topic-wise TypeScript notes** guide with **quick examples**, ideal for future use in **real-world company projects**. Each topic includes a concise explanation and example code for quick understanding and reference.

🟦 1. What is TypeScript?

- TypeScript is a **typed superset of JavaScript** that compiles to plain JavaScript.
- Adds **static typing**, **interfaces**, **classes**, **enums**, etc.

```ts
let message: string = "Hello, TypeScript";
console.log(message);
```

🟩 2. Basic Types

Type	Example
string	`let name: string = "Supriya";`
number	`let age: number = 26;`
boolean	`let isActive: boolean = true;`
any	`let value: any = "could be any"`
unknown	`let input: unknown = 5;`
void	`function log(): void {}`
null	`let x: null = null;`
undefined	`let y: undefined = undefined;`

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

🟨 3. Arrays and Tuples

```ts
let numbers: number[] = [1, 2, 3];
let names: string[] = ["A", "B", "C"];

// Tuple
let person: [string, number] = ["Supriya", 26];
```

🟧 4. Enums

```ts
enum Role {
  ADMIN,
  USER,
  GUEST
}

let userRole: Role = Role.ADMIN;
```

🟪 5. Interfaces

Used to define the shape of objects (like contracts).

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

const user: User = {
  id: 1,
  name: "Supriya"
};
```

⬛ 6. Type Aliases

```ts
type ID = number | string;

let userId: ID = "abc123";
```

🟫 7. Functions with Types

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

```
// Arrow function
const subtract = (a: number, b: number): number => a - b;
```

🔵 8. Union & Intersection Types

```ts
// Union
let data: string | number;
data = 5;
data = "five";

// Intersection
interface A { a: number }
interface B { b: number }

type AB = A & B;
const obj: AB = { a: 1, b: 2 };
```

🟠 9. Generics

Make code reusable and type-safe.

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

let output1 = identity<string>("hello");
let output2 = identity<number>(100);
```

🟢 10. Classes

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

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

const dog = new Animal("Dog");
dog.move(10);
```

🔴 11. Access Modifiers

- `public`: accessible anywhere
- `private`: accessible inside the class
- `protected`: accessible in class and subclasses

```ts
class Car {
  private engine: string;

  constructor(engine: string) {
    this.engine = engine;
  }

  start(): void {
    console.log("Starting engine: " + this.engine);
  }
}
```

🟤 12. Type Assertion

```ts
let someValue: any = "This is a string";

let strLength: number = (someValue as string).length;
// or: let strLength: number = (<string>someValue).length;
```

⚪ 13. Literal Types

```ts
let direction: "up" | "down";
direction = "up"; // valid
// direction = "left"; // ❌ Error
```

🔷 14. Null, Undefined & Optional Chaining

```ts
let user: { name: string; age?: number } = { name: "Supriya" };

console.log(user.age?.toString()); // undefined
```

🔶 15. Readonly Properties

```ts
interface Point {
  readonly x: number;
  readonly y: number;
}

const pt: Point = { x: 10, y: 20 };
// pt.x = 5; ❌ Error
```

🟣 16. Modules (Import / Export)

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

// main.ts
import { sum } from './math';
console.log(sum(5, 10));
```

🟥 17. Utility Types

Utility	Description
`Partial<T>`	Makes all properties optional
`Required<T>`	Makes all properties required
`Readonly<T>`	Makes all properties readonly
`Record<K, T>`	Constructs an object type
`Pick<T, K>`	Selects a set of properties
`Omit<T, K>`	Removes a set of properties

```ts
interface User {
  id: number;
  name: string;
  email: string;
}

type PartialUser = Partial<User>;
type ReadonlyUser = Readonly<User>;
```

🔵 18. Type Narrowing

```ts
function printId(id: number | string) {
  if (typeof id === "string") {
    console.log(id.toUpperCase());
  } else {
    console.log(id.toFixed(2));
  }
}
```

🧩 19. Advanced: Custom Types for APIs

```ts
interface APIResponse<T> {
  status: number;
  data: T;
}

const response: APIResponse<{ name: string }> = {
  status: 200,
  data: { name: "Supriya" }
};
```

🛠️ 20. TypeScript with React (Quick Example)

```tsx
interface Props {
  title: string;
}

const Header: React.FC<Props> = ({ title }) => {
  return <h1>{title}</h1>;
};
```

✅ Bonus: Compile TypeScript to JS

```bash
tsc index.ts
```

🚀 Summary Table

Topic	Use Case Example
Basic Types	`` `let name: string = "Supriya"` ``
Functions	`` `function add(a: number, b: number): number` ``
Interfaces	For object shapes
Classes	OOP support
Generics	Reusable components
Modules	Large projects / real company codebases
Utility Types	`` `Partial` ``, `` `Readonly` ``, etc.
React + TS	`` `React.FC<Props>` ``

If you want a **PDF version** of these notes or a **TypeScript cheat sheet for company use**, just let me know — I'll generate it for you.

Would you like that?

it is full details yes or no

✅ **Yes**, the notes I provided cover all **important foundational and intermediate TypeScript topics** you need for:

- Company projects 👨‍💻
- Real-world React + TypeScript apps ⚛️
- Backend with Node.js + TypeScript 🧠
- Clean, maintainable, and type-safe code 📦

📌 Covered:

- ✅ All basic and advanced types
- ✅ Functions, arrays, tuples, enums
- ✅ Interfaces vs Type aliases
- ✅ Access modifiers (public, private, protected)
- ✅ Classes, generics, utility types
- ✅ Modules (import/export)
- ✅ Type narrowing, assertion, literal types
- ✅ React + TypeScript example
- ✅ Bonus: Compile command

🧩 Not Yet Covered (Optional Advanced Topics – Tell me if you want them):

If you're building more complex projects, you might also need:

1. ✅ **TypeScript with Express.js**
2. ✅ **Decorators** (used in NestJS)
3. ✅ **Type Guards & Discriminated Unions**
4. ✅ **Advanced Generic Constraints**
5. ✅ **Working with Third-party Type Definitions**
6. ✅ **Type Inference Deep Dive**
7. ✅ **TypeScript config (`tsconfig.json`) explained**
8. ✅ **Monorepo / Modular structures in large projects**

📄 Want this as a downloadable PDF or Notion-style doc?

Let me know — I'll create and share a copy for your future reference.

Is this conversation helpful so far?