**TYPESCRIPT NOTES BEGINNER TO ADVANCED**

**HEADING**

**EXAMPLE**

**Beginner Level**

1. **Basic Types**
   * Learn the core types: string, number, boolean, array, object, tuple, and enum.
   * Example:

Ts

let age: number = 25;

let name: string = "John";

let isStudent: boolean = true;

1. **Type Inference**
   * TypeScript can infer types automatically based on the value assigned.
   * Example:

ts

let count = 10; // TypeScript infers this as number

1. **Functions with Typed Parameters and Return Types**
   * Define types for function parameters and return values.
   * Example:

ts

function add(x: number, y: number): number {

return x + y;

}

1. **Interfaces**
   * Define the shape of an object using interface.
   * Example:

Ts

interface Person {

name: string;

age: number;

}

let john: Person = { name: "John", age: 30 };

1. **Union Types**
   * Allow variables to hold multiple types using the | operator.
   * Example:

ts

let id: number | string;

id = 10;

id = "ABC";

**Intermediate Level**

1. **Optional Properties**
   * Use ? to mark properties as optional.
   * Example:

ts

interface Car {

brand: string;

model?: string;

}

1. **Readonly Properties**
   * Use readonly to prevent modification of properties after initialization.
   * Example:

ts

interface Book {

readonly title: string;

}

1. **Type Aliases**
   * Define custom types using type.
   * Example:

ts

type ID = number | string;

let userId: ID;

1. **Intersection Types**
   * Combine multiple types into one using &.
   * Example:

ts

interface A { a: string; }

interface B { b: number; }

let ab: A & B = { a: "Hello", b: 42 };

1. **Classes**
   * Use TypeScript with OOP concepts like classes, inheritance, and access modifiers (private, public, protected).
   * Example:

ts

class Animal {

private name: string;

constructor(name: string) {

this.name = name;

}

speak() {

return `${this.name} makes a noise.`;

}

}

1. **Generics**
   * Create reusable components with type safety using generics.
   * Example:

ts

Copy code

function identity<T>(arg: T): T {

return arg;

}

**Advanced Level**

1. **Advanced Types: Partial, Required, Pick, Omit**
   * Use utility types to transform existing types.
   * Example:

tS

interface User { name: string; age: number; email: string; }

type PartialUser = Partial<User>;

1. **Mapped Types**
   * Transform types by iterating over keys of an object type.
   * Example:

ts

type ReadonlyUser<T> = { readonly [K in keyof T]: T[K]; };

1. **Conditional Types**
   * Use types that depend on a condition.
   * Example:

ts

type IsNumber<T> = T extends number ? true : false;

1. **Type Guards**
   * Use runtime checks to narrow down types.
   * Example:

ts

function isString(value: any): value is string {

return typeof value === "string";

}

1. **Discriminated Unions**
   * Use a common property to distinguish between union types.
   * Example:

ts

interface Square { kind: "square"; size: number; }

interface Circle { kind: "circle"; radius: number; }

type Shape = Square | Circle;

1. **Advanced Generics (Constraints, Default Types, etc.)**
   * Impose constraints on generic types and set default values.
   * Example:

ts

function merge<T extends object, U extends object>(obj1: T, obj2: U): T & U {

return { ...obj1, ...obj2 };

}

1. **Decorators (Experimental)**
   * Use decorators for metaprogramming.
   * Example:

ts

function Log(target: any, propertyKey: string) {

console.log(`Property ${propertyKey} has been accessed`);

}

class MyClass {

@Log

myMethod() {}

}

1. **Module Augmentation**
   * Extend modules by adding new declarations.
   * Example:

ts

declare module 'my-module' {

interface MyModule {

newProperty: string;

}

}

1. **Declaration Merging**
   * Combine declarations with the same name.
   * Example:

ts

interface User { name: string; }

interface User { age: number; }

let user: User = { name: "John", age: 30 };

**Best Practices:**

* **Strict Mode**: Always use TypeScript’s strict mode for better type safety (strict: true in tsconfig.json).
* **Avoid any**: Minimize the use of any, which can defeat the purpose of TypeScript’s type-checking.
* **Use Type Inference**: Let TypeScript infer types when it can, but always annotate function signatures.

4o