Type Alias in TypeScript

1. Problem Statement

You're building a **Warehouse Inventory System** that must track products, orders, and storage locations. You need clear, reusable type definitions so the code is maintainable and self-documenting. Without aliases, you'd repeat complex union or object types everywhere, leading to errors and duplication.

---

2. Learning Objectives

By the end of this tutorial, you will be able to:

- Define **type aliases** for primitives, unions, tuples, objects, and functions.
- Use **generic** aliases to model containers of varied data.
- Simplify function signatures with alias types.
- Improve code readability and maintainability with custom type names.

---

3. Concept Introduction with Analogy

Analogy: The Shipping Label Templates
In a warehouse, each package gets a **shipping label**. Rather than scribbling all fields each time, you use a **template**:

1. **Simple Labels** for tracking numbers (primitive alias).
2. **Status Tags** that accept "pending," "shipped," or "returned" (union alias).
3. **Coordinate Stickers** listing aisle and shelf numbers (tuple alias).
4. **Product Cards** with ID, name, and location (object alias).
5. **Action Forms** that log events or errors (function alias).
6. **Container Wraps** that hold any item type (generic alias).

Type aliases in TypeScript are like these templates-concise, reusable definitions for complex types.

---

4. Technical Deep Dive

- **Syntax:** `type AliasName = ExistingType;`
- **Primitives:** alias `number` or `string` to meaningful names.
- **Union Types:** combine multiple primitives into one named type.
- **Tuples:** fixed-length arrays with specified element types.
- **Objects:** shape definitions for repeated object structures.
- **Function Types:** define parameter and return value signatures.

- **Generics:** parameterize aliases for any data type.

---

5. Step-by-Step Code Walkthrough

```ts
// 1. Primitive Alias
type ProductID = number;
let widgetId: ProductID = 42;

// 2. Union Alias
type OrderStatus = "pending" | "shipped" | "returned";
let status: OrderStatus = "shipped";

// 3. Tuple Alias
type Coordinate = [aisle: number, shelf: number];
let loc: Coordinate = [3, 14];

// 4. Object Alias
type Product = {
  id: ProductID;
  name: string;
  location: Coordinate;
  price: number;
};
let product: Product = {
  id: widgetId,
  name: "Widget",
  location: loc,
  price: 19.99
};

// 5. Function Type Alias
type Logger = (message: string) => void;
const consoleLogger: Logger = msg => console.log(`[LOG] ${msg}`);

// 6. Generic Alias
type Container<T> = { value: T; timestamp: Date };
let productContainer: Container<Product> = {
  value: product,
  timestamp: new Date()
};
let idContainer: Container<ProductID> = {
  value: 7,
  timestamp: new Date()
};
```

---

6. Interactive Challenge / Mini-Project

Your Turn!

1. **Define** a `CustomerID` alias for `string`.
2. **Create** a `Customer` object alias with `id: CustomerID`, `name: string`, and optional `email?: string`.
3. **Implement** a `processOrder` function type alias that accepts `orderId: number` and a callback `(status: OrderStatus) => void`.
4. **Use** the `Container<T>` generic to wrap a `Customer` object.

---

7. Solution & Deep Dive Explanation

```ts
// 1. Primitive Alias for CustomerID
type CustomerID = string;

// 2. Object Alias for Customer
type Customer = {
  id: CustomerID;
  name: string;
  email?: string;
};
let customer: Customer = { id: "C123", name: "Alice" };

// 3. Function Type Alias for processOrder
type OrderCallback = (status: OrderStatus) => void;
const processOrder: OrderCallback = status => {
  consoleLogger(`Order is now ${status}`);
};
processOrder("pending");

// 4. Wrap Customer in Container<T>
let customerContainer: Container<Customer> = {
  value: customer,
  timestamp: new Date()
};
console.log(customerContainer);
```

Explanation:

- **`CustomerID`** makes it clear that this `string` is a customer identifier.
- **`Customer`** alias reuses `CustomerID` and marks `email` optional with `?`.
- **`OrderCallback`** defines the exact signature for order-status handlers.
- **`Container<Customer>`** uses the generic alias to wrap a `Customer` with metadata.

---

8. Quick Recap & Key Takeaways

- **Type Aliases** = named templates for any TypeScript type.
- Cover **primitives**, **unions**, **tuples**, **objects**, **functions**, and **generics**.
- Improve code **readability**, **reuse**, and **consistency**.
- Keep aliases **focused**, **meaningful**, and **well-named**.