Interfaces and Type Aliases

Part 5

TS

A Beginner's Guide for JavaScript Developers

Nadhem JBELI

/in/nadhemjbeli

1

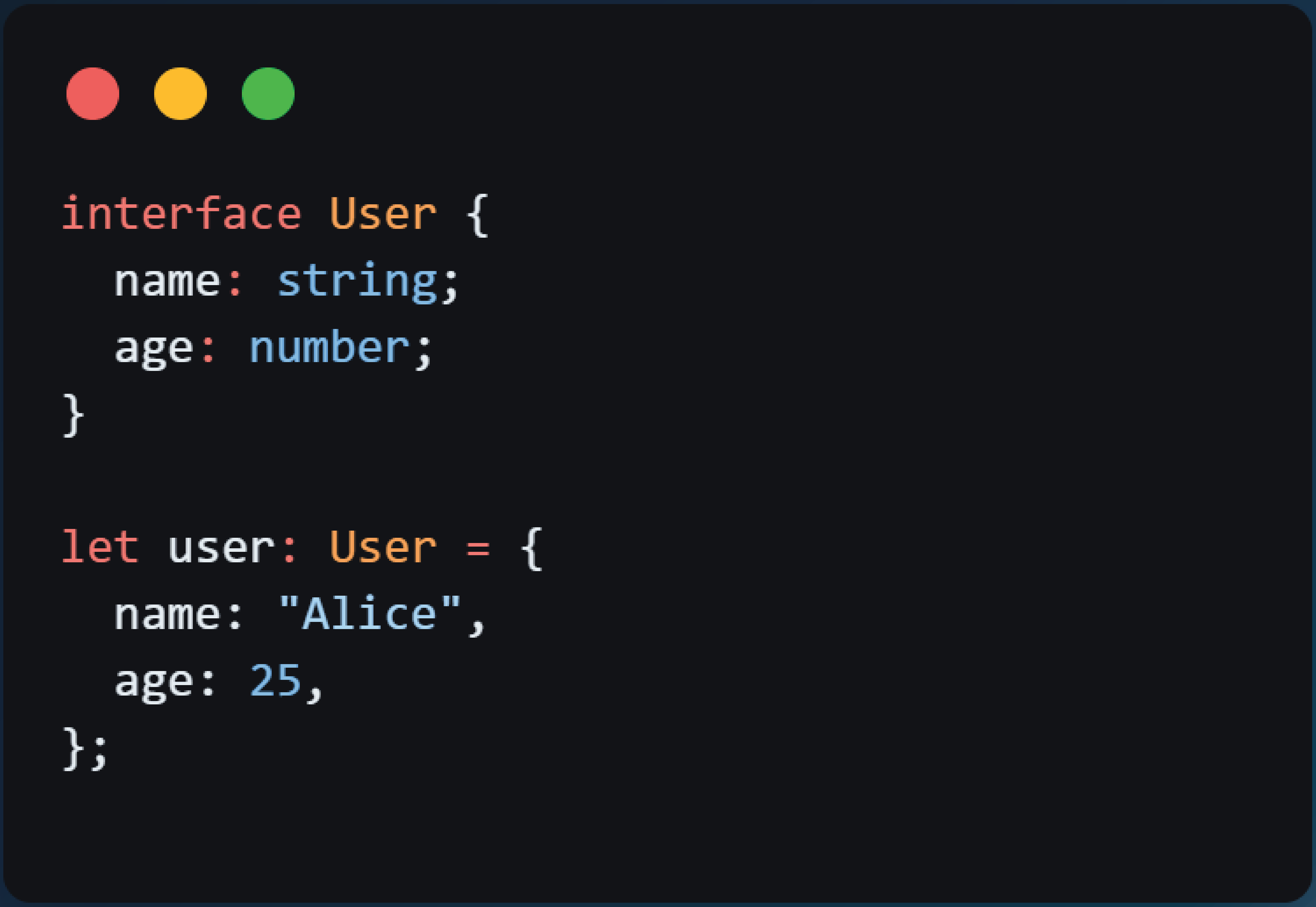
What You’ll Learn

- What are interfaces and type aliases?
- Differences between interfaces and type aliases.
- Practical use cases for each.

2

What are Interfaces?

- Interfaces define the structure of an object.
- They are used to enforce a specific shape for objects.

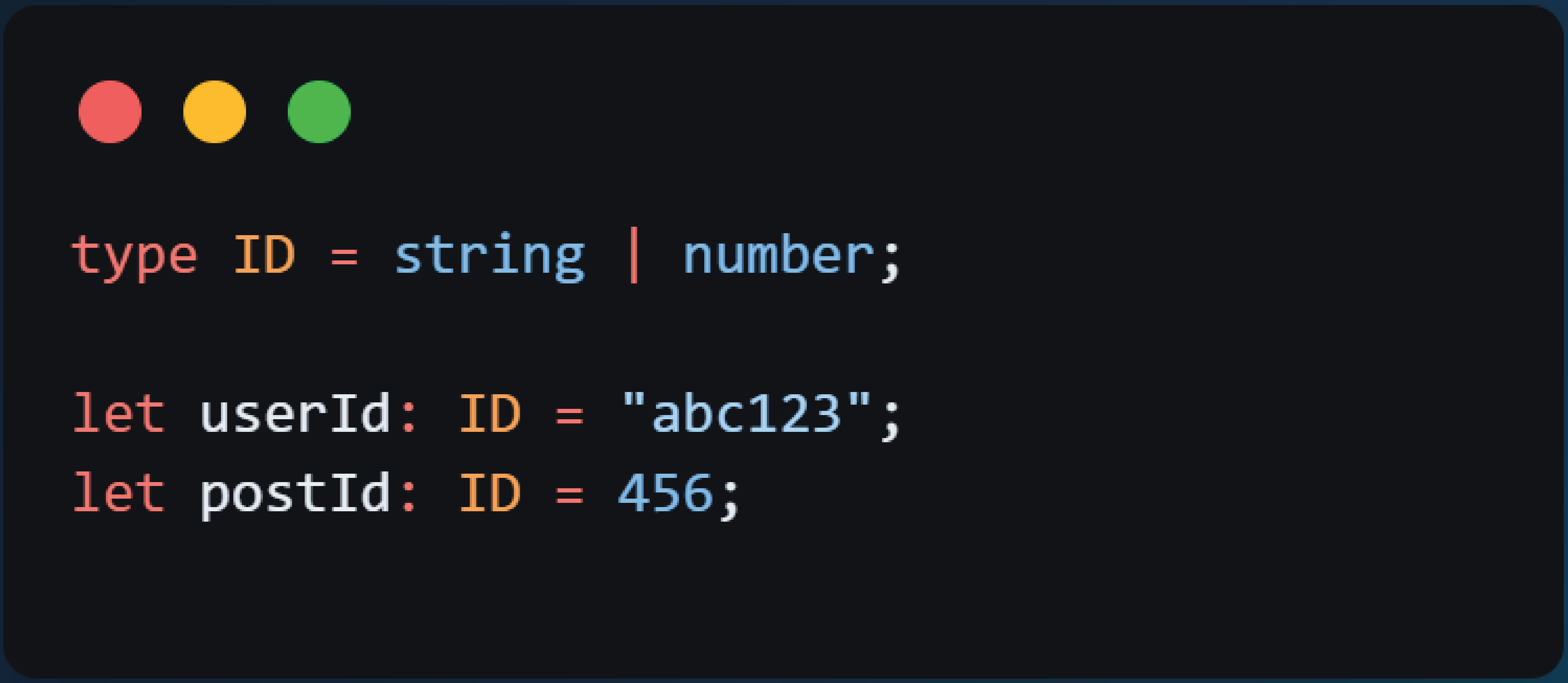


```
interface User {
  name: string;
  age: number;
}

let user: User = {
  name: "Alice",
  age: 25,
};
```

3

What are Type Aliases?

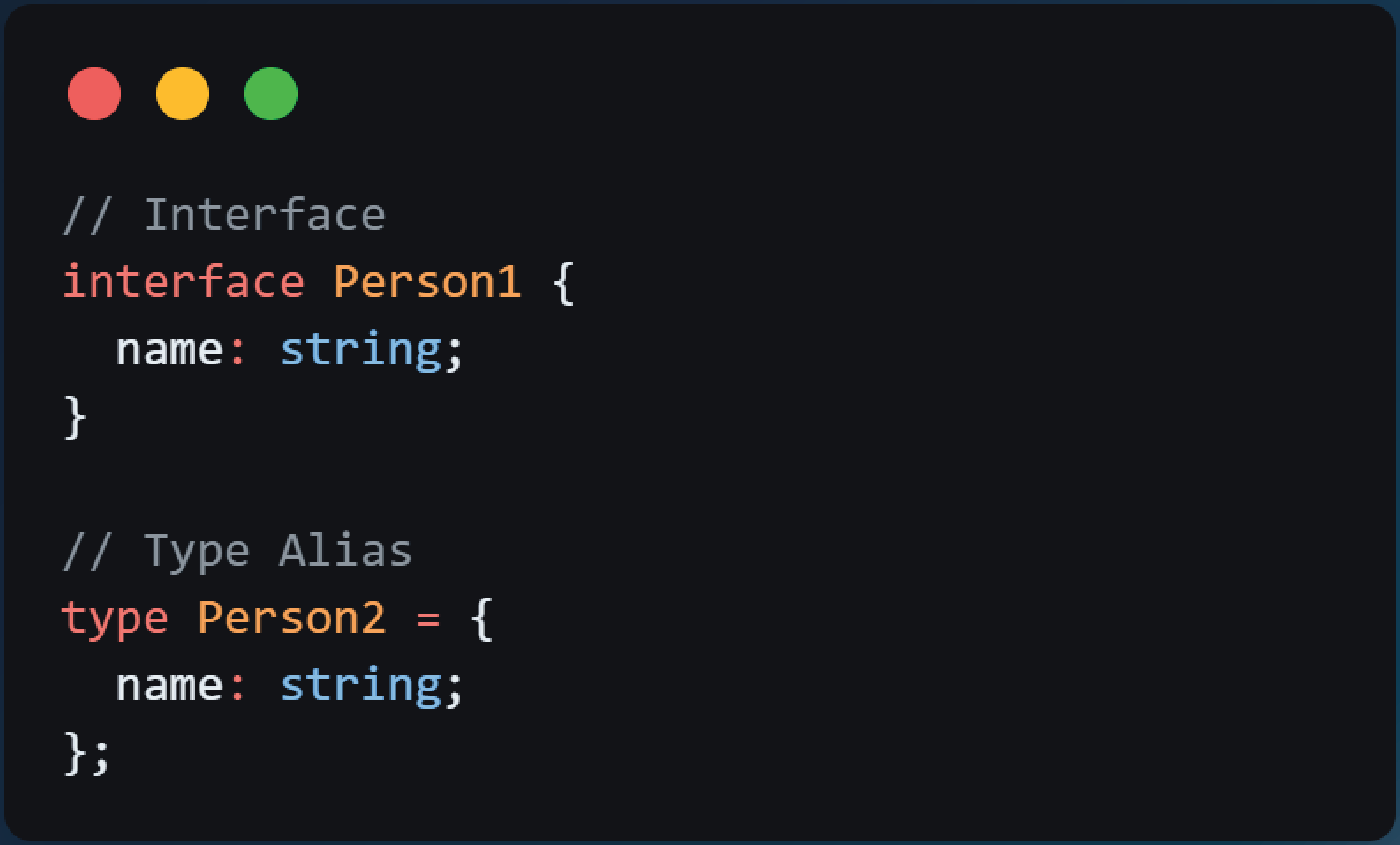
- Type aliases allow you to create custom types.
- They can represent primitive types, unions, tuples, and more.



```
type ID = string | number;

let userId: ID = "abc123";
let postId: ID = 456;
```

4 Interfaces vs Type Aliases

- **Interfaces**
  - Used for object shapes.
  - Can be extended or implemented.
- **Type Aliases**
  - More flexible (can represent any type).
  - Cannot be extended or implemented.

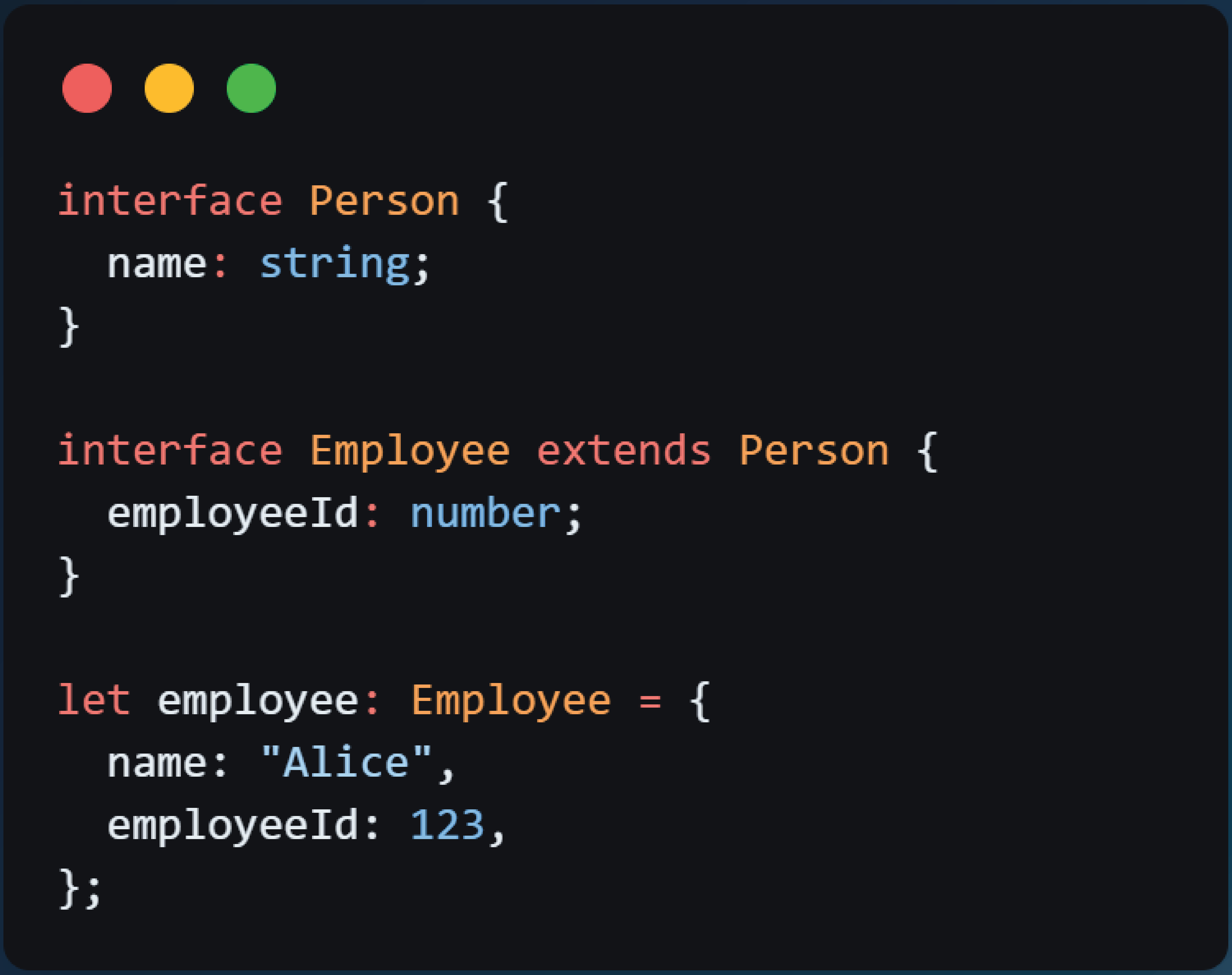


```typescript
// Interface
interface Person1 {
  name: string;
}

// Type Alias
type Person2 = {
  name: string;
};
```

5

Extending Interfaces

- Interfaces can extend other interfaces.
- Useful for creating reusable and modular types.

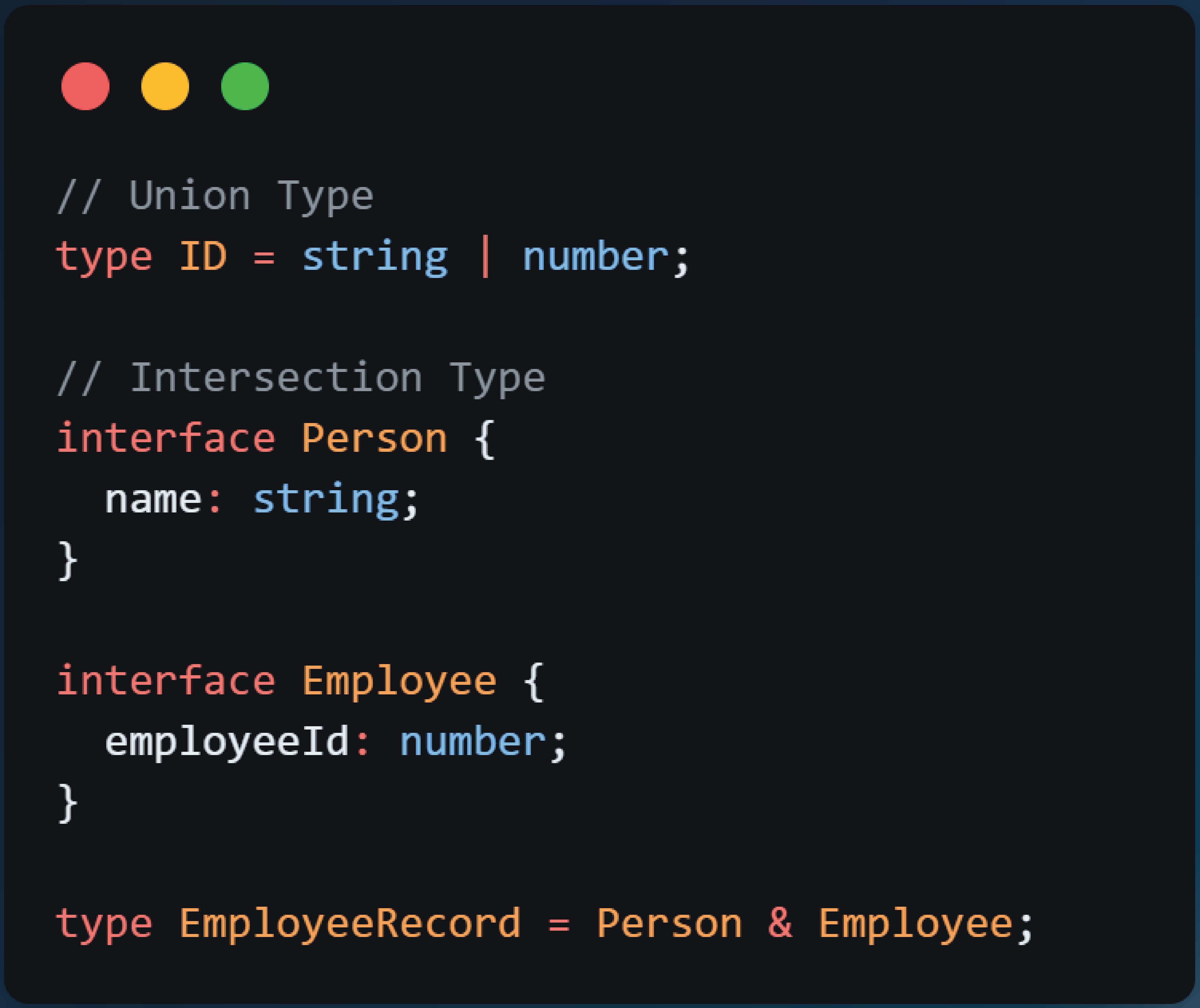


```
interface Person {
  name: string;
}

interface Employee extends Person {
  employeeId: number;
}

let employee: Employee = {
  name: "Alice",
  employeeId: 123,
};
```

6

Union and Intersection

- **Union Types:** Combine multiple types using |
- **Intersection Types:** Combine multiple types using &

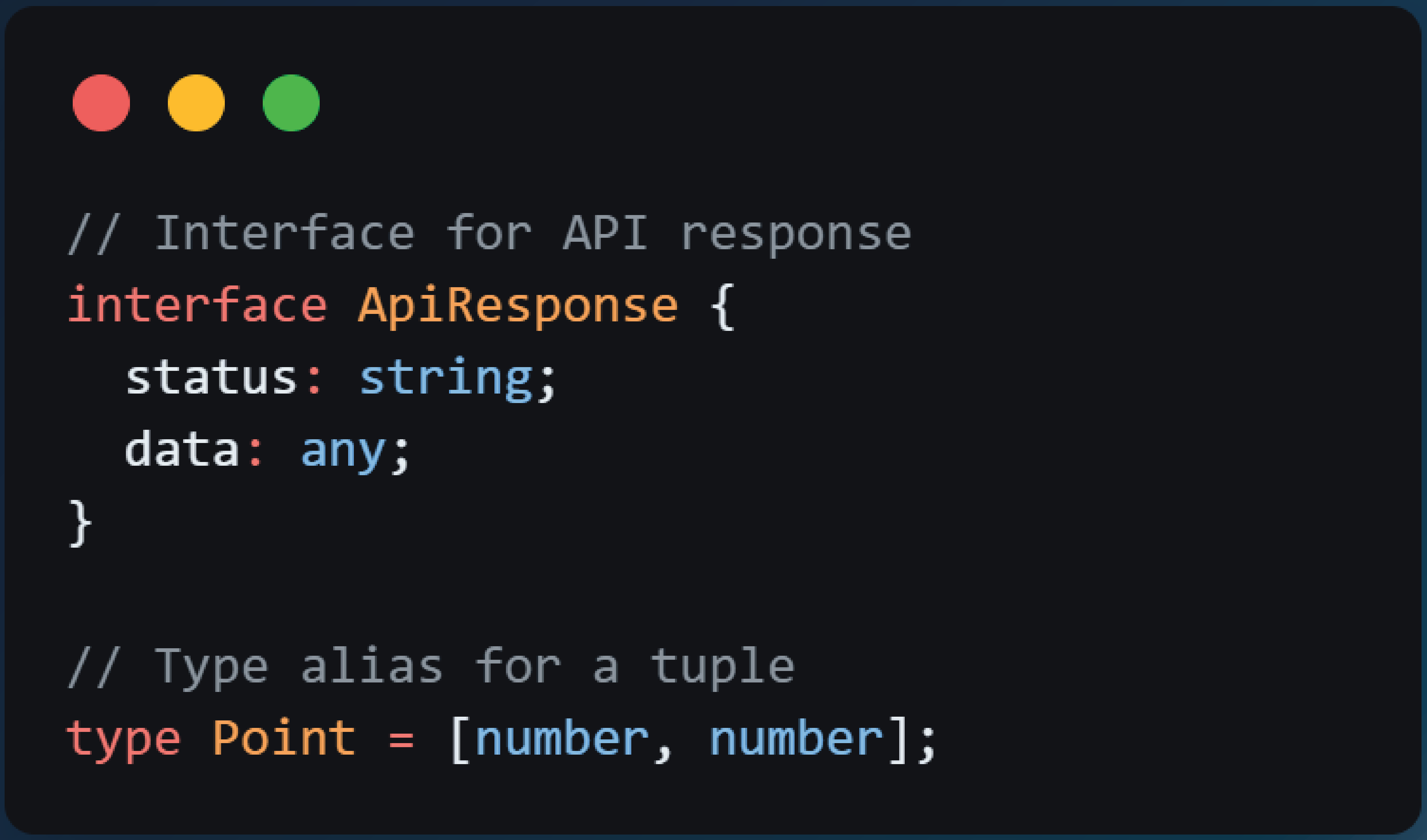


```
// Union Type
type ID = string | number;

// Intersection Type
interface Person {
  name: string;
}

interface Employee {
  employeeId: number;
}

type EmployeeRecord = Person & Employee;
```

7

Practical Use Cases

- **Interfaces**
  - Define object shapes (e.g., API responses).
  - Extend existing types.
- **Type Aliases**
  - Create reusable types for primitives, unions, or tuples.
  - Simplify complex type definitions.



```typescript
// Interface for API response
interface ApiResponse {
  status: string;
  data: any;
}

// Type alias for a tuple
type Point = [number, number];
```

8

Summary

- Interfaces define object shapes and can be extended.
- Type aliases are flexible and can represent any type.
- Use interfaces for objects and type aliases for unions, primitives, or tuples.

Ready to dive deeper? Stay tuned for "Classes and Object-Oriented Programming"!