TypeScript Basics for JavaScript Developers

1. How to Declare Types

You can tell TypeScript what kind of data a variable holds.

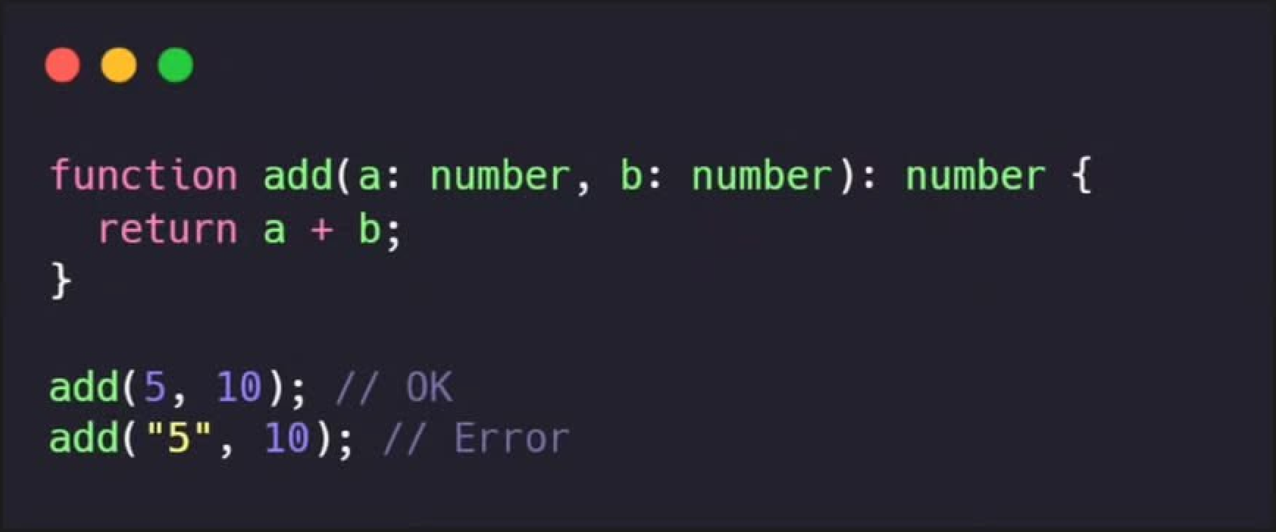
```typescript
let name: string = "Oliver"; // Only strings allowed
let age: number = 30;        // Only numbers allowed
let isHappy: boolean = true; // Only true or false
```

If you don't add a type, TypeScript guesses it from the value.

```typescript
let city = "London"; // TypeScript thinks this is a string
```

2. Type Annotations in Functions

You can say what types go in and out of a function.



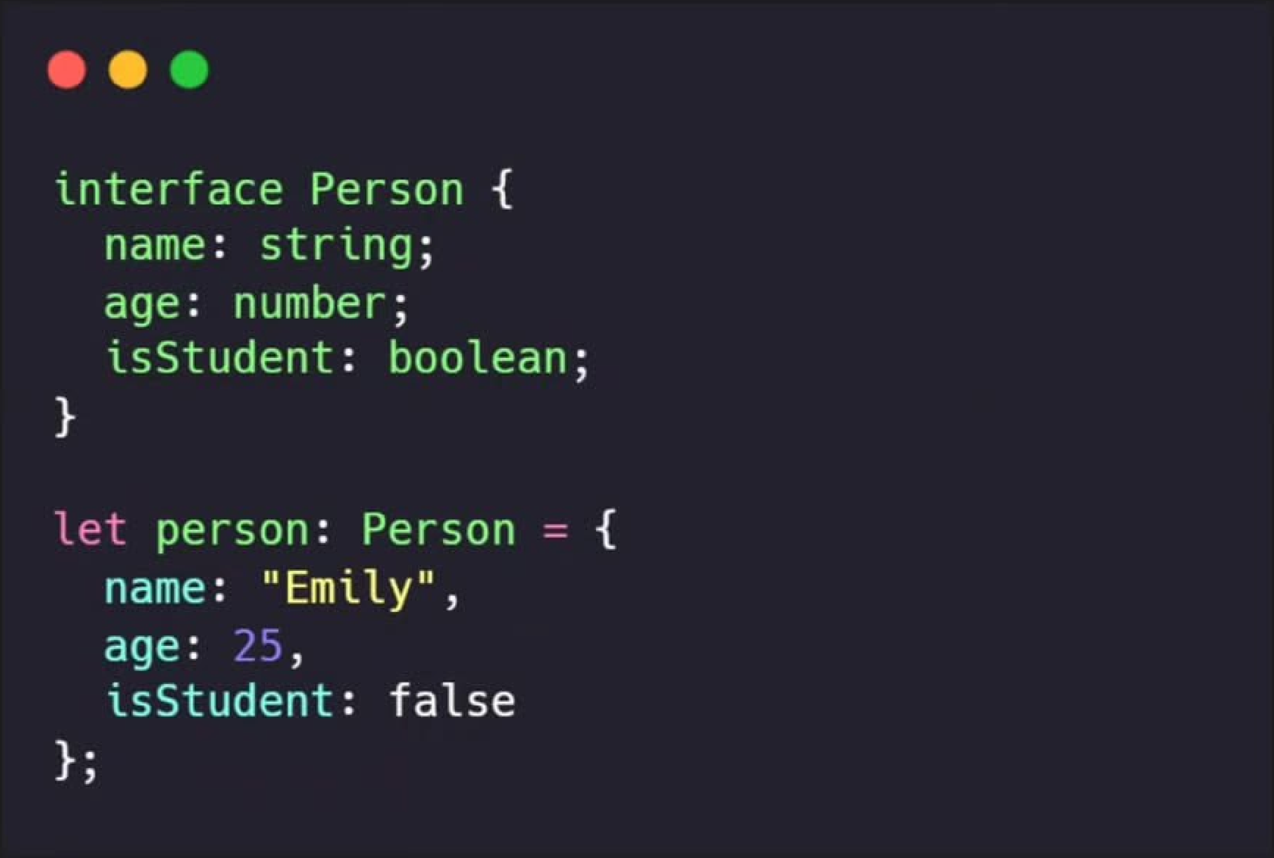
```
function add(a: number, b: number): number {
  return a + b;
}

add(5, 10); // OK
add("5", 10); // Error
```

3. Interfaces and Types

Use interface or type to describe objects.

Interface Example



```typescript
interface Person {
  name: string;
  age: number;
  isStudent: boolean;
}

let person: Person = {
  name: "Emily",
  age: 25,
  isStudent: false
};
```

Type Example

```typescript
type Car = {
  brand: string;
  year: number;
};

let myCar: Car = {
  brand: "Ford",
  year: 2020
};
```

4. Optional and Read-Only Properties

- Optional (?): Property might not exist.
- Read-only: Cannot change the value.

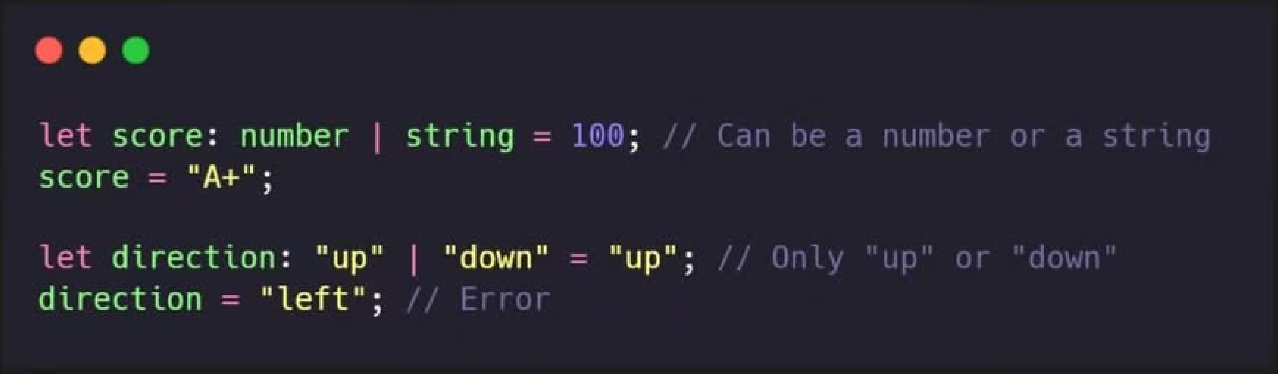
```typescript
interface House {
  address: string;
  owner?: string; // Optional
  readonly builtYear: number; // Read-only
}

let myHouse: House = {
  address: "221B Baker Street",
  builtYear: 1887
};

myHouse.address = "10 Downing Street"; // OK
myHouse.builtYear = 1900; // Error
```

5. Union and Literal Types

- Union: Variable can have multiple types.
- Literal: Only specific values allowed.



```typescript
let score: number | string = 100; // Can be a number or a string
score = "A+";

let direction: "up" | "down" = "up"; // Only "up" or "down"
direction = "left"; // Error
```

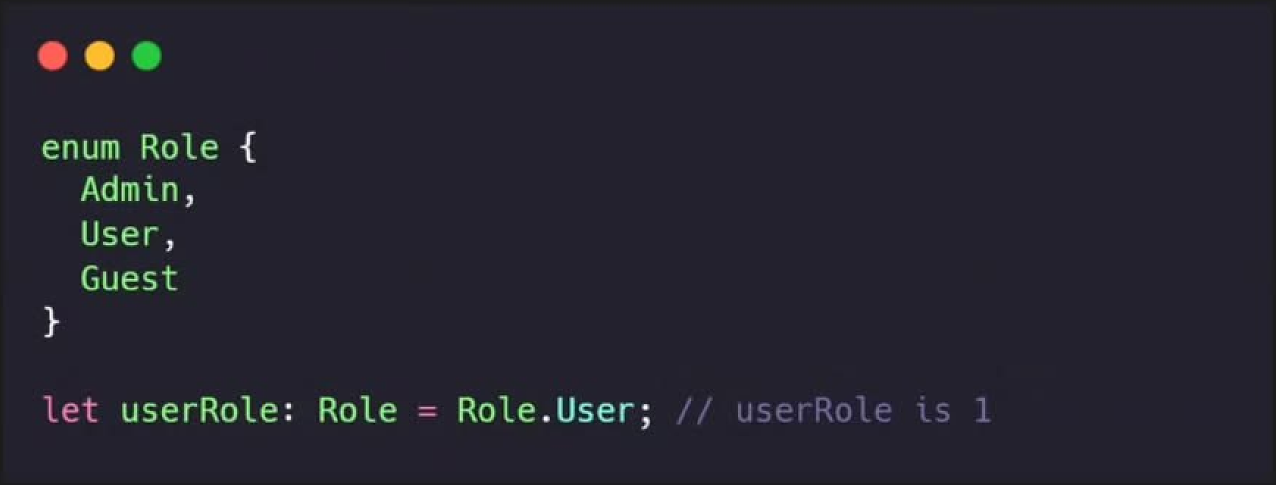
6. Arrays and Tuples

- Arrays: All elements of the same type.
- Tuples: Fixed number of elements with specific types.

```
let fruits: string[] = ["apple", "banana", "cherry"];
let coordinates: [number, number] = [51.5, -0.1];
```

7. Enums

Enums give names to numbers or strings.

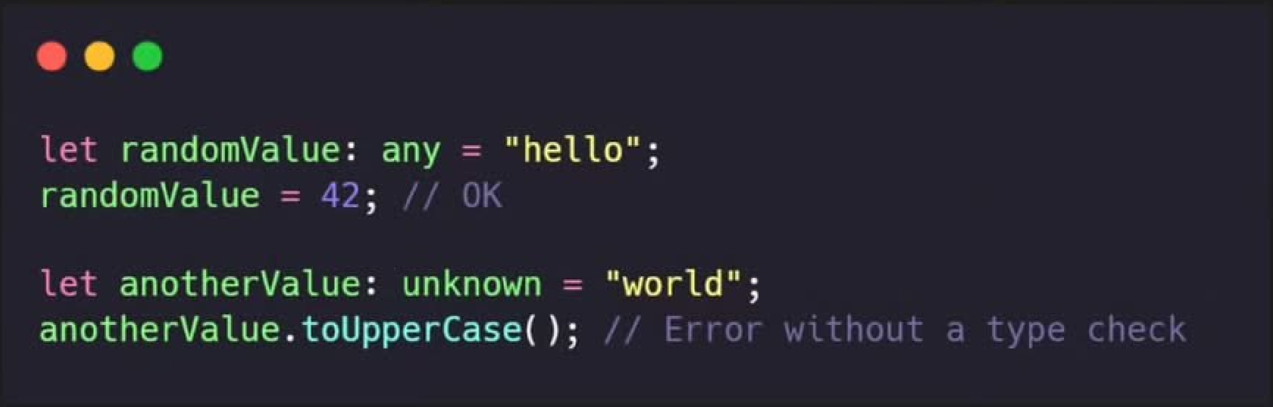


```
enum Role {
  Admin,
  User,
  Guest
}

let userRole: Role = Role.User; // userRole is 1
```

8. Any and Unknown

- any: Any type (try to avoid).
- unknown: Like any, but safer.



```
let randomValue: any = "hello";
randomValue = 42; // OK

let anotherValue: unknown = "world";
anotherValue.toUpperCase(); // Error without a type check
```

9. Type Assertions

Tell TypeScript you know the type.

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

10. Classes

TypeScript improves classes with visibility modifiers (public, private, protected).

```typescript
class Animal {
  private name: string;

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

  speak() {
    console.log(`${this.name} makes a noise.`);
  }
}

let dog = new Animal("Max");
dog.speak(); // OK
dog.name = "Buddy"; // Error
```

11. Generics

Make reusable components with types.

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

identity<string>("hello");
identity<number>(123);
```

12. Modules

Use export and import to organize code.

File: mathUtils.ts

```
export function multiply(a: number, b: number):
number {
  return a * b;
}
```

File: main.ts

```
import { multiply } from "./mathUtils";
console.log(multiply(3, 4));
```