**What is Strong Typing?**

* In TypeScript (used in Angular), **we tell what type of data a variable can hold**.
* Example: If we say a variable is a **number**, it cannot hold a **string**.

👉 This avoids mistakes before running the program (compiler shows error).

let age: number = 20; // correct

age = "twenty"; // ❌ Error: "twenty" is a string, not a number

### What is an Interface?

* **Interface = shape/blueprint of an object.**
* It tells **what properties and methods an object must have**.

// Define interface

interface Student {

rollNo: number;

name: string;

course: string;

}

// Use interface

let s1: Student = {

rollNo: 1,

name: "Apsara",

course: "Angular"

};

If we miss a property or put the wrong type, Angular will give an **error immediately**.

### Why Strong Typing & Interfaces are Good?

1. **Catch errors early** → Mistakes are found before running.  
   (e.g., putting a string instead of a number).
2. **Improves code quality** → Code looks neat and clean.
3. **Easy to understand** → Anyone can quickly see what type of data is used.
4. **Better productivity** → Less debugging, less tension.
5. **IDE support** → While typing, VS Code will show **suggestions and autocompletion**.

// student.model.ts

export interface Student {

id: number;

name: string;

course: string;

}

// student.component.ts

import { Component } from '@angular/core';

import { Student } from './student.model';

@Component({

selector: 'app-student',

template: `

<h2>Student List</h2>

<ul>

<li \*ngFor="let s of students">

{{s.id}} - {{s.name}} ({{s.course}})

</li>

</ul>

`

})

export class StudentComponent {

students: Student[] = [

{ id: 1, name: "Arun", course: "Angular" },

{ id: 2, name: "Divya", course: "React" },

{ id: 3, name: "Ravi", course: "Vue" }

];

}

Benefits here:

* If we miss course, Angular will **show error immediately**.
* Helps us avoid runtime mistakes.

Interfaces in Angular

Blueprint for Data

// Define interface

interface Person {

name: string;

age: number;

}

// Use interface

let p1: Person = {

name: "Rahul",

age: 25

};

If we forget age or put a string instead of number, we get an error.

let p2: Person = {

name: "Rahul",

age: "twenty-five" // ❌ Error: should be number

};

### Contract for Components

* Interfaces can also be used as **contracts** → if a class implements an interface, it **must** provide the required methods.

// Define interface

interface Logger {

log(message: string): void;

}

// Class must follow the contract

class ConsoleLogger implements Logger {

log(message: string): void {

console.log("LOG:", message);

}

}

// Usage

let logger = new ConsoleLogger();

logger.log("Angular is awesome!");

Rule: If a class **implements** an interface, it must write all the methods inside.

# **Styling in Angular**

In Angular, we can style components in **4 ways**:

### 1. ****Inline Styles****

* Add style **directly inside the HTML tag** using style="".
* Good for **small, quick changes**.
* ❌ Not good for big projects → becomes messy.

<p style="color: red; font-size: 20px;">

This is inline styled text.

</p>

### 2. ****Component Styles****

* Write CSS **inside the component TypeScript file** (in styles: []).
* Styles apply **only to that component** (scoped).
* More **organized** than inline styles.

import { Component } from '@angular/core';

@Component({

selector: 'app-message',

template: `<p>Hello Students!</p>`,

styles: [`

p {

color: blue;

font-weight: bold;

}

`]

})

export class MessageComponent {}

### 3. ****External Stylesheets****

* Create a **separate CSS file** (e.g., message.component.css).
* Mention it in the component using styleUrls.
* Best for **bigger styling**, easy to manage.

@Component({

selector: 'app-message',

template: `<p>Hello Students!</p>`,

styleUrls: ['./message.component.css']

})

export class MessageComponent {}

message.component.css

p {

color: green;

font-size: 18px;

}

### 4. ****Global Styles****

* Write CSS in the **global styles.css file** (in Angular project root).
* Affects the **whole application**.