**1. Understanding the Need for Styling React Components**

Styling is essential in React applications for the same reasons it is in any web application—it improves the visual appearance, enhances user experience, and communicates brand or functional design.

**Why Styling React Components is Needed:**

* To make the UI look clean, modern, and user-friendly.
* To apply consistent design patterns across the application.
* To visually separate and organize different parts of the app.
* To handle dynamic visual feedback (e.g., hover, active, responsive design).
* React promotes **component-based architecture**, so each component should manage its **own styling** to remain modular and reusable.

**2. Working with CSS Module and Inline Styles**

React offers multiple ways to style components. Two common approaches are **CSS Modules** and **Inline Styles**.

**A. CSS Modules**

**What is a CSS Module?**  
A CSS Module is a CSS file where class names and animations are scoped locally by default. This means styles written in one component won’t affect others unintentionally.

**How to use CSS Modules:**

1. Create a file with .module.css extension.

/\* Button.module.css \*/

.primary {

background-color: blue;

color: white;

padding: 10px;

}

1. Import and use it inside your React component:

import styles from './Button.module.css';

function Button() {

return <button className={styles.primary}>Click Me</button>;

}

**Benefits:**

* Style encapsulation and no global conflicts.
* Clear mapping between styles and components.
* Easier to maintain in large apps.

**B. Inline Styles**

**What are Inline Styles?**  
Inline styles in React are written as JavaScript objects and applied directly to elements using the style attribute.

**Example:**

function Header() {

const headerStyle = {

color: 'white',

backgroundColor: 'black',

padding: '15px',

textAlign: 'center'

};

return <h1 style={headerStyle}>Welcome</h1>;

}

**Key Points:**

* Style property names use **camelCase** (e.g., backgroundColor).
* Values are usually strings (e.g., '15px', 'blue').
* Suitable for dynamic styles and one-off customizations.

**Limitations:**

* No support for pseudo-classes (:hover, :active) or media queries.
* Not ideal for complex styling or reusable styles.