🌟 JSX, React Components, Compilation Flow, and Variations

💡 JSX: JavaScript XML

☑ What is JSX?

JSX allows writing **HTML-like code inside JavaScript**. It's syntactic sugar for `React.createElement()`.

```
const heading = <h1>Hello, JSX!</h1>;
```

> JSX is not HTML – it's closer to XML and gets compiled to `React.createElement` under the hood.

⚡ JSX Compilation Flow

```
const heading = <h1>React</h1>;
```

Becomes:

```
React.createElement("h1", null, "React")
```

Then:

```
// React Element (Plain JS Object)
{
  type: "h1",
  props: {
    children: "React"
  }
}
```

Then:

Rendered by React DOM to actual HTML:

```
<h1>React</h1>
```

> JSX ➡️ `React.createElement` ➡️ React Element ➡️ JS Object ➡️ HTML (UI)

🚀 Packages Involved

Tool	Role
`babel`	Compiles JSX to JS (React.createElement)
`@babel/preset-react`	JSX transformer config
`react`	Core React library (manages components & elements)
`react-dom`	Renders elements to the DOM

🔢 Types of React Components

1. Functional Component (Stateless)

```
function Header() {
  return <h1>Hello</h1>;
}
```

2. Arrow Function Component

```
const Header = () => <h1>Hello</h1>;
```

3. Component without Return

```
const Header = () => (
  <h1>Hello</h1>
);
```

4. Component with multiple lines

```
const Header = () => {
  return (
    <>
      <h1>Hello</h1>
      <p>Welcome</p>
    </>
  );
}
```

> JSX multiline code must be wrapped in `()` if using `return`, or use `<>...</>` (Fragments)

📃 JSX Rules

- Only one parent element
- Use `className` instead of `class`
- Expressions must be inside `{}`
- Boolean, ternary, and functions allowed in `{}`

```
const name = "Darshan";
const greet = () => "Hello!";

return <h1>{greet()}, {name}</h1>
```

🔍 JSX vs JavaScript

JSX	JavaScript
`<h1>Hello</h1>`	`React.createElement("h1", null, "Hello")`

📖 Naming Conventions

Type	Convention	Example
Components	PascalCase	`MyComponent`
Variables, functions	camelCase	`myFunction`

> All components must start with a **Capital Letter** in JSX or React treats them as HTML tags.

👉 Ways to Write JSX

1. Single-line JSX

```
return <h1>Hello</h1>;
```

2. Multi-line JSX

```
return (
  <div>
    <h1>Hello</h1>
    <p>Welcome</p>
  </div>
);
```

🔗 Expressions in JSX

```
return <h2>{1 + 2}</h2>
```

> Anything valid in JS expression context can go inside `{}`

📅 Component Inside Component

```
const Title = () => <h1>Title</h1>;

const App = () => (
  <div>
    <Title />
    <Title></Title>
    {Title()}
  </div>
);
```

> All 3 work:

- `<Title />` – JSX style
- `<Title></Title>` – long form
- `{Title()}` – function call (not recommended in large apps)

📋 React Element Inside Component

```
const element = <h1>Hello</h1>;

const App = () => (
  <div>
    {element}
  </div>
);
```

📋 Component Inside React Element

```
const App = () => <Title />;
```

📋 Element Inside Element

```
const element = <h1>{<span>Nested</span>}</h1>;
```

⚠ Cross-site Scripting (XSS)

JSX escapes by default:

```
const userInput = "<script>alert('Hacked')</script>";
return <p>{userInput}</p>; // Prints as string, not executed
```

> JSX uses **dangerouslySetInnerHTML** if raw HTML is absolutely needed (use with care):

```
<p dangerouslySetInnerHTML={{ __html: userInput }} />
```

📊 Variations of Functional Component

☑ Arrow Function + `return`

```
const App = () => {
  return <h1>Hello</h1>;
};
```

☑ Arrow Function without `return` (implicit return)

```
const App = () => <h1>Hello</h1>;
```

☑ Function Declaration

```
function App() {
  return <h1>Hello</h1>;
}
```

☑ Multiple Elements using React Fragments

```
const App = () => (
  <>
    <h1>Header</h1>
    <p>Welcome!</p>
  </>
);
```

🔄 Execution Order Recap:

```
const Title = () => <h1>Title</h1>;

const App = () => (
  <div>
    <Title />                // Component
    {<h2>JSX Element</h2>}  // Element inside Component
    {Title()}                // JSX via function
  </div>
);
```

> JSX compiles all of them to `React.createElement` => React Element => Render to DOM
