What is React.js?

React.js is an open-source JavaScript library developed by **Facebook (Meta)** for building **fast, interactive, and reusable** user interfaces, mainly for **single-page applications (SPAs)**.

□ Core Concepts

1. Components

React apps are made up of **components** — small, reusable pieces of UI.

There are two main types:

```
// Functional Component
function Welcome() {
  return <h1>Hello, World!</h1>;
}

// Class Component
class Welcome extends React.Component {
  render() {
    return <h1>Hello, World!</h1>;
  }
}
```

---

2. JSX (JavaScript XML)

JSX lets you write HTML-like syntax inside JavaScript.

```
const element = <h1>Hello, {userName}!</h1>;
```

JSX is compiled to JavaScript:

```
React.createElement('h1', null, 'Hello, ' + userName + '!');
```

---

3. Props (Properties)

Props let you pass data **from parent to child components**.

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Usage
<Greeting name="Alice" />
```

---

4. State

State represents **data that can change over time** inside a component.

```
import { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </>
  );
}
```

---

5. Events

React handles events in a camelCase style.

```
<button onClick={handleClick}>Click Me</button>
```

---

6. Conditional Rendering

Render elements based on conditions:

```
{isLoggedIn ? <Dashboard /> : <Login />}
```

---

7. Lists and Keys

Render lists using the `map()` method.
Always include a **unique `key`**.

```
const todos = ['Learn React', 'Build App', 'Deploy'];

<ul>
  {todos.map((todo, index) => (
    <li key={index}>{todo}</li>
  ))}
</ul>
```

---

8. useEffect Hook

Used for side effects like fetching data, timers, etc.

```
import { useEffect } from "react";

useEffect(() => {
  console.log("Component mounted");
}, []); // [] ensures it runs once
```

---

9. React Router (for navigation)

Used for page navigation in SPAs.

```
npm install react-router-dom
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> | <Link to="/about">About</Link>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
```

---

10. Creating a React App

You can quickly start a React project using:

```
npx create-react-app my-app
cd my-app
npm start
```

The two main types of React components:

1. **Functional components** (modern and preferred)
2. **Class components** (older but still used in legacy code)

We'll separate each component into its own file inside a `src/components/` folder.

Folder Structure

```
my-react-app/
└── src/
    ├── components/
    │   ├── FunctionalWelcome.js
    │   └── ClassWelcome.js
    ├── App.js
    ├── index.js
    └── index.css
```

Step-by-Step Code

1. `src/components/FunctionalWelcome.js`

```
import React from "react";

function FunctionalWelcome() {
  return <h2 style={{ color: "blue" }}>Hello from Functional
Component!</h2>;
}

export default FunctionalWelcome;
```

---

2. `src/components/ClassWelcome.js`

```
import React from "react";

class ClassWelcome extends React.Component {
  render() {
    return <h2 style={{ color: "green" }}>Hello from Class Component!</h2>;
  }
}

export default ClassWelcome;
```

---

3 `src/App.js`

```
import React from "react";
import FunctionalWelcome from "./components/FunctionalWelcome";
import ClassWelcome from "./components/ClassWelcome";

function App() {
  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>React Components Example</h1>
      <p>React apps are made up of components — small, reusable pieces of
UI.</p>

      <div style={{ marginTop: "30px" }}>
        <FunctionalWelcome />
        <ClassWelcome />
      </div>
    </div>
  );
}

export default App;
```

---

4 `src/index.js`

(Usually already created by Create React App — just ensure it looks like this)

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);
```

▶ Run It

1. Create the app:

   ```
   npx create-react-app my-react-app
   cd my-react-app
   ```

2. Create the `components` folder and add the two component files.
3. Replace your `App.js` and `index.js` with the code above.
4. Start your app:

   ```
   npm start
   ```

5. Visit http://localhost:3000 — you'll see both the **Functional** and **Class** components rendered.

Here's a full working React example **of your `Counter` component — including how it fits inside a complete React app.**

```
npx create-react-app my-app
cd my-app
npm start
```

📁 Folder Structure (basic React app)

```
my-app/
├── src/
│   ├── App.js
│   ├── Counter.js
│   └── index.js
├── package.json
└── ...
```

📄 src/Counter.js

```
import { useState } from "react";

function Counter() {
  // Declare state variable 'count' and a function 'setCount' to update it
  const [count, setCount] = useState(0);

  return (
    <>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </>
  );
}

export default Counter;
```

📄 src/App.js

```
import React from "react";
import Counter from "./Counter";

function App() {
  return (
```

```
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>Simple React Counter</h1>
      <Counter />
    </div>
  );
}

export default App;
```

---

src/index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

// Create root and render the App component
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

To Run This:

1. Create a new React app (if you haven't already):
2. `npx create-react-app my-app`
3. Navigate into it:
4. `cd my-app`
5. Replace the contents of the `src` folder with the code above.
6. Start the app:
7. `npm start`
8. Open your browser at **http://localhost:3000**

Output:

You'll see:

```
Simple React Counter
You clicked 0 times
[Click me]
```

Each time you click the button, the number increases by 1 — all powered by **React's useState hook**

or with **Vite (faster alternative)**:

```
npm create vite@latest my-app -- --template react
cd my-app
npm install
npm run dev
```

Let's turn your `Greeting` component into a fully working React app so you can run it and see how `props` work in React.

What This Example Does:

- Defines a simple **Greeting** component.
- Passes a `name` prop from a parent (`App`) component.
- Renders a personalized greeting like:
  "Hello, Alice!"

Folder Structure

```
my-app/
├── src/
│   ├── App.js
│   ├── Greeting.js
│   └── index.js
```

---

src/Greeting.js

```
// Greeting.js
import React from "react";

function Greeting(props) {
  return <h3>Hello, {props.name}!</h3>;
}

export default Greeting;
```

---

src/App.js

```
// App.js
import React from "react";
import Greeting from "./Greeting";

function App() {
  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h2>React Props Example</h2>


      <Greeting name="Alice" />
      <Greeting name="Bob" />
      <Greeting name="Charlie" />
    </div>
  );
}

export default App;
```

---

src/index.js

```
// index.js
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

// Create React root and render App
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

□ To Run This:

1. Create a new React app (if not done yet):

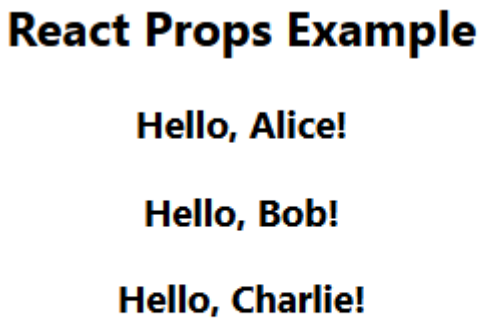
   ```
   npx create-react-app my-app
   cd my-app
   ```

2. Replace the contents of the `src` folder with the files above.
3. Run your app:

   ```
   npm start
   ```

4. Open **http://localhost:3000** in your browser.

□ Output:



□ Key Takeaways

- Props allow data to flow **from parent → child**.
- They make components **reusable** with different data.
- Example:
- ```
  <Greeting name="Alice" />
  <Greeting name="Bob" />
  ```
  ```

Here's a full working React example showing how to use that `<button>` with a `handleClick` function.

□ Full Working Example (React)App.js:-

```
import React from "react";
import ReactDOM from "react-dom/client";

function App() {
  // Define the click handler
  const handleClick = () => {
    alert("Button clicked!");
  };

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>Hello, React!</h1>
      <button onClick={handleClick}>Click Me</button>
    </div>
  );
}

export default App;
```

□ How to Run It

1. **Create a React project** (if you don't have one yet):

   ```
   npx create-react-app my-app
   cd my-app
   ```

2. **Replace the contents of** `src/App.js` with the code above.
3. **Start the app**:

   ```
   npm start
   ```

4. Open http://localhost:3000 — you'll see a "Click Me" button that shows an alert when pressed.

very common React pattern: **conditional rendering** using the ternary operator (`condition ? A : B`).

Here's a **full, working React example** demonstrating how to render either a `Dashboard` or a `Login` component depending on whether the user is logged in.

□ Full Working Example (React)App.js file code:-

```
import React, { useState } from "react";
```

```
import ReactDOM from "react-dom/client";
// Dashboard component
function Dashboard({ onLogout }) {
 return (
 <div style={styles.container}>
 <h2>Welcome to your Dashboard!</h2>
 <button onClick={onLogout}>Logout</button>
 </div>
 );
}
// Login component
function Login({ onLogin }) {
 return (
 <div style={styles.container}>
 <h2>Please Log In</h2>
 <button onClick={onLogin}>Login</button>
 </div>
 );
}
// Main App component
function App() {
 const [isLoggedIn, setIsLoggedIn] = useState(false);
 // Event handlers
 const handleLogin = () => setIsLoggedIn(true);
 const handleLogout = () => setIsLoggedIn(false);
 return (
 <div>
 <h1>Conditional Rendering Example</h1>
 {/* Conditionally render based on state */}
 {isLoggedIn ? (
 <Dashboard onLogout={handleLogout} />
 ) : (
 <Login onLogin={handleLogin} />
 )}
 </div>
 );
}
// Simple styles
const styles = {
 container: {
 textAlign: "center",
 marginTop: "50px",
 },
};
export default App;
```

□ How to Run It

1. **Create a new React app** if you don't have one:

   ```
   npx create-react-app conditional-rendering
   cd conditional-rendering
   ```

2. **Replace `src/App.js`** with the code above.
3. Run:

   ```
   npm start
   ```

4. Go to http://localhost:3000 —
   you'll see:
   - A **Login** screen initially
   - After clicking "Login," it switches to the **Dashboard**

   - Clicking "Logout" switches it back.

React list rendering example using `Array.map()` to display items dynamically.

Here's a **full, working React example** that demonstrates how to render that list inside a component.

□ Full Working Example (React)App.js file code:-

```
import React from "react";
import ReactDOM from "react-dom/client";

function App() {
  // Define the array of todos
  const todos = ["Learn React", "Build App", "Deploy"];

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>My Todo List</h1>

      {/* Render the list dynamically */}
      <ul style={{ listStyleType: "none", padding: 0 }}>
        {todos.map((todo, index) => (
          <li
            key={index}
            style={{
              backgroundColor: "#f0f0f0",
              margin: "8px auto",
              width: "200px",
              padding: "10px",
              borderRadius: "5px",
            }}
          >
            {todo}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

□ How to Run It

1. **Create a React project** (if you don't have one yet):

   ```
   npx create-react-app todo-list
   cd todo-list
   ```

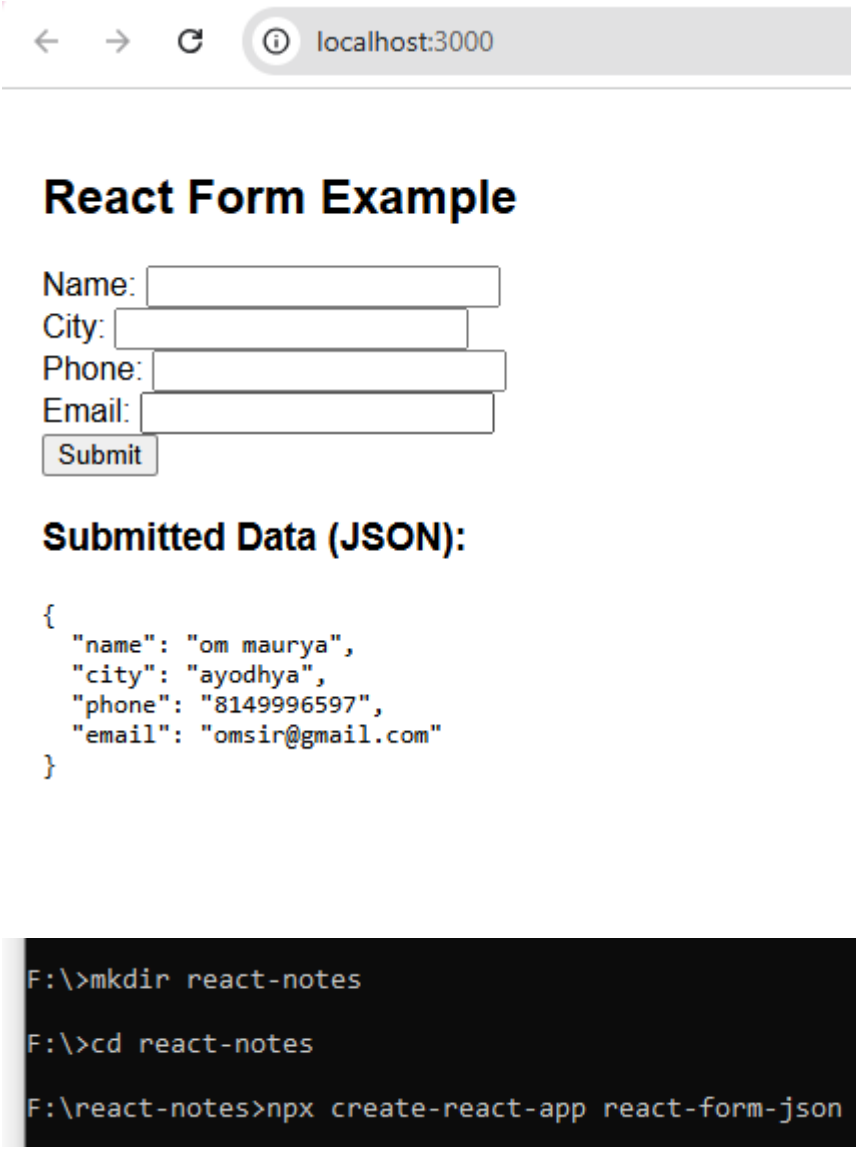
2. **Replace the contents of `src/App.js`** with the code above.
3. Run:

   ```
   npm start
   ```

4. Visit http://localhost:3000 — you'll see a simple styled **Todo List** with:

```
Learn React
Build App
Deploy
```

React Form Example:-




```
F:\>mkdir react-notes

F:\>cd react-notes

F:\react-notes>npx create-react-app react-form-json
```

src/App.js file code:-

```jsx
import React, { useState } from "react";

function App() {
  const [formdata, setformdata] = useState({
    name: "",
    city: "",
    phone: "",
    email: "",
  });

  const [submittedData, setSubmittedData] = useState(null);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setformdata({ ...formdata, [name]: value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    setSubmittedData(formdata); // Save submitted data
    setformdata({ name: "", city: "", phone: "", email: "" }); // Reset form
  };

  return (
    <div style={{ padding: "20px", fontFamily: "Arial" }}>
      <h2>React Form Example</h2>
      <form onSubmit={handleSubmit}>
        <div>
          <label>Name: </label>
          <input
            type="text"
            name="name"
            value={formdata.name}
            onChange={handleChange}
            required
          />
        </div>
        <div>
          <label>City: </label>
          <input
            type="text"
            name="city"
            value={formdata.city}
            onChange={handleChange}
            required
          />
        </div>
        <div>
```

```
          <label>Phone: </label>
          <input
            type="tel"
            name="phone"
            value={formdata.phone}
            onChange={handleChange}
            required
          />
        </div>
        <div>
          <label>Email: </label>
          <input
            type="email"
            name="email"
            value={formdata.email}
            onChange={handleChange}
            required
          />
        </div>
        <button type="submit">Submit</button>
      </form>

      {submittedData && (
        <div style={{ marginTop: "20px" }}>
          <h3>Submitted Data (JSON):</h3>
          <pre>{JSON.stringify(submittedData, null, 2)}</pre>
        </div>
      )}
    </div>
  );
}

export default App;
```

index.js file code:-

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

you can also write in another way example2 **App.js** file code:-

```
import React, { useState } from "react";

function App() {
  const [name, setName] = useState("");
  const [city, setCity] = useState("");
  const [phone, setPhone] = useState("");
  const [email, setEmail] = useState("");

  const [submittedData, setSubmittedData] = useState(null);

  const handleSubmit = (e) => {
    e.preventDefault();
    const data = { name, city, phone, email };
    setSubmittedData(data);

    // Reset form
    setName("");
    setCity("");
    setPhone("");
    setEmail("");
  };
```

```jsx
return (
  <div style={{ padding: "20px", fontFamily: "Arial" }}>
    <h2>Form with Separate States</h2>
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Name"
        value={name}
        onChange={(e) => setName(e.target.value)}
        required
      />
      <br />
      <input
        type="text"
        placeholder="City"
        value={city}
        onChange={(e) => setCity(e.target.value)}
        required
      />
      <br />
      <input
        type="tel"
        placeholder="Phone"
        value={phone}
        onChange={(e) => setPhone(e.target.value)}
        required
      />
      <br />
      <input
        type="email"
        placeholder="Email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        required
      />
      <br />
      <button type="submit">Submit</button>
    </form>

    {submittedData && (
      <div style={{ marginTop: "20px" }}>
        <h3>Submitted Data (JSON):</h3>
        <pre>{JSON.stringify(submittedData, null, 2)}</pre>
      </div>
    )}
  </div>
);
```

```
}

export default App;
```

Key Points:-

- Both examples display the submitted form data as **pretty JSON** using `JSON.stringify(submittedData, null, 2)` and `<pre>`.
- **Example 1** is better for **larger forms**.
- **Example 2** is simpler for **small forms**.