Component:

A component is a reusable, self-contained piece of UI that manages its own structure, style, and behaviour. Think of it like a building block for creating user interfaces.

Types of Components

There are **two main types**:

1. **Functional Components** (most common in modern React)

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

2. **Class Components** (older style, still used in some codebases)

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Composing Components

Components can refer to other components in their output.

Header.js

```
import './Header.css';

function Header() {
    return (
        <>
            <h1>ECOM App</h1>
            <hr></hr>
        </>
    )
}

export default Header;
```

Footer.js

```
function Footer() {
    const year = new Date().getFullYear();

    return (
        <>
            <hr></hr>
            <p>&copy; {year} Xyz Pvt Ltd. All rights reserved.</p>
        </>
    )
}
export default Footer;
```

Main.js:

```
function Main() {

    return(
        <>
          <h3>This is Main component</h3>
        </>
    )
}
export default Main;
```

App.js

```
import './App.css';
import Header from './components/Header';
import Main from './components/Main';
import Footer from './components/Footer';

function App() {

  return (
    <div className="App">

      <Header />
      <Main />
      <Footer />

    </div>
  );
}

export default App;
```

Props in React:

In React, props (short for properties) are a way of passing data from a parent component to a child component.

They are read-only — meaning the child component cannot modify the props it receives. Props help make components reusable and dynamic.

Main.js

```
import Welcome from "./Welcome";

function Main() {

    const name ="Raj Kumar";
    const name2 = "Karan";

    return(
        <>
            <Welcome user={name}/>
            <Welcome user={name2}/>
        </>
    )
}
export default Main;
```

Welcome.js

```
function Welcome(props) {

    return(
        <>
        <h2>Welcome, {props.user}</h2>
        </>
    );
}
```

or

```
function Welcome({ user }) {

    return (
        <>
            <h2>Welcome, {user}</h2>
        </>
    );
}
export default Welcome;
```

State and Lifecycle

Stateful Components:

- **State: A Component's Memory**

```
import React from 'react';

class Counter extends React.Component {
    constructor() {
        super();
        this.state = {
            counter: 0
        }
        console.log("constructor called");
    }
    componentDidMount() {
        console.log("component did mount called");
    }
    increment = () => {
        this.setState({
            counter: this.state.counter + 1
        })
    }
    decrement = () => {
        this.setState({
            counter: this.state.counter - 1
        })
    }
    render() {
        console.log("render method called");
        return (
            <>
                <h3>Counter: {this.state.counter}</h3>
                <button onClick={this.increment}>+</button>
                <button onClick={this.decrement}>-</button>
            </>
        )
    }
}
export default Counter;
```

The Component Lifecycle

Each component has several "lifecycle methods" that you can override to run code at particular times in the process.

Mounting

These methods are called in the following order when an instance of a component is being created and inserted into the DOM:

- **constructor()**
- **render()**
- **componentDidMount()**

Updating

An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered:

- **render()**
- **componentDidUpdate()**

Unmounting

This method is called when a component is being removed from the DOM:

- **componentWillUnmount()**

Handling Events

React lets you add *event handlers* to your JSX. Event handlers are your own functions that will be triggered in response to interactions like clicking, hovering, focusing form inputs, and so on.

Adding event handlers

To add an event handler, you will first define a function and then pass it as a prop to the appropriate JSX tag. For example, here is a button that doesn't do anything yet:

```
export default function Button() {
  function handleClick() {
    alert('You clicked me!');
  }

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

Alternatively, you can define an event handler inline in the JSX:

```
<button onClick={() => {
  alert('You clicked me!');
}}>
```

Functions passed to event handlers must be passed, not called. For example:

passing a function (correct)	calling a function (incorrect)
`<button onClick={handleClick}>`	`<button onClick={handleClick()}>`

When you write code inline, the same pitfall presents itself in a different way:

passing a function (correct)	calling a function (incorrect)
`<button onClick={() => alert('...')}>`	`<button onClick={alert('...')}>`

Passing event handlers as props:

Example 1: A Button That Notifies the Parent

Parent.js (Parent Component)

```
import ChildButton from "./ChildButton";

function Parent() {

    const handleButtonClick = () => {
        alert("Button was clicked in the child component!");
    }

    return(
        <>
            <h3>Passing Event Handler Example</h3>
            <ChildButton onBtnClick={handleButtonClick}/>

        </>
    )
}
export default Parent;
```

ChildButton.js (Child Component)

```
function ChildButton({onBtnClick}) {

    return(
        <>
            <button onClick={onBtnClick}>Click</button>
        </>
    )
}
export default ChildButton;
```

Example 2: To-Do List Item Completion Toggle

Each child item has a checkbox. When toggled, it tells the parent to update the item's status.

TodoList.js (Parent Component)

```
import { useState } from "react";
import TodoItem from "./TodoItem";

function TodoList() {
    const [todos, setTodos] = useState([
        { id: 1, text: 'Learn React', completed: false },
        { id: 2, text: 'Write Code', completed: false },
    ]);
    const toggleTodo = (id) => {
        setTodos((prevTodos) =>
            prevTodos.map(todo =>
                todo.id === id ? { ...todo, completed: !todo.completed } : todo
            )
        );
    };
    return (
        <div>
            <h2>To-Do List</h2>
            {todos.map((todo) => (
                <TodoItem key={todo.id} todo={todo} onToggle={toggleTodo} />
            ))}
        </div>
    );
}

export default TodoList;
```

TodoItem.js (Child Component)

```
function TodoItem({ todo, onToggle }) {
    return (
        <div>
            <input
                type="checkbox"
                checked={todo.completed}
                onChange={() => onToggle(todo.id)}
            />
            <span style={{ textDecoration: todo.completed ? 'line-through' : 'none' }}>
                {todo.text}
            </span>
        </div>
    );
}
export default TodoItem;
```

Conditional Rendering

Your components will often need to display different things depending on different conditions. In React, you can conditionally render JSX using JavaScript syntax like if statements, &&, and ? : operators.

Conditionally returning JSX

```
import GuestGreetings from "./GuestGreetings";
import UserGreetings from "./UserGreetings";

function Greetings() {

    const isLoggedIn = false;
    const userName = "Raj Kumar";

    if(isLoggedIn) {
        return <UserGreetings uname={userName}/>
    }
    return <GuestGreetings/>
}

export default Greetings;
```

```
function UserGreetings(props) {
    return (
        <>
            <h2>Welcome {props.uname}</h2>
        </>
    )
}

export default UserGreetings;

function GuestGreetings() {

    return(
        <>
        <h3>Hello Guest!</h3>
        </>
    )
}

export default GuestGreetings;
```

Logical AND operator (&&)

```
function Mailbox() {
    const unreadMsgs = ["msg-1", "msg-2", "msg-3"];
    return (
        <>
            <h1>Inbox</h1>
            {
                unreadMsgs.length > 0 &&
                <h3>You have {unreadMsgs.length} unread messages in your mail box</h3>
            }
        </>
    )
}
export default Mailbox;
```

Conditional (ternary) operator (? :)

```
function Order() {
    const paymentStatus = "success";

    return (
        <>
            {
                paymentStatus === "failed" ?
                    <div>
                        <h2>Order placed successfuly</h2>
                        <h3>Order Details</h3>
                        <p>Order No: 12345</p>
                        <p>Order Amount: $567</p>
                    </div>

                    : <h2>Order Failed</h2>
            }
        </>
    )
}

export default Order;
```

Lists and Keys:

We can use the JavaScript array methods to manipulate an array of data.

use filter() and map() with React to filter and transform your array of data into an array of components.

Rendering data from arrays of strings:

```
function Items() {
    const fruits = ['orange', 'apple', 'mango', 'banana'];
    return (
        <>
            <h3>Fruits</h3>
            {
                fruits.map((f,index) =>
                    <p key={index}>
                    {index+1} : {f}
                    </p>
                )
            }
        </>
    )
}
export default Items;
```

Rendering data from arrays of Objects:

```
function DisplayAllProducts() {
    const products = [
        {
            id: 101,
            name: "IPhonex",
            price: 89000
        },
        {
            id: 102,
            name: "SamsungM30",
            price: 65000
        }
    ];
```

```
    return (
        <>
            <h2>All Products</h2>
            <table border={1}>
                <thead>
                    <tr>
                        <th>Id</th>
                        <th>Name</th>
                        <th>Price</th>
                    </tr>
                </thead>
                <tbody>
                    {
                        products.map(p =>
                            <tr key={p.id}>
                                <td>{p.id}</td>
                                <td>{p.name}</td>
                                <td>{p.price}</td>
                            </tr>
                        )
                    }
                </tbody>
            </table>
        </>
    )
}
export default DisplayAllProducts;
```

Filtering arrays of items

Let's say you want a way to only show people whose profession is 'chemist'. You can use JavaScript's filter() method to return just those people.

```
function ListDemo() {

    const people = [{
        id: 0,
        name: 'Creola Katherine Johnson',
        profession: 'mathematician',
    }, {
        id: 1,
        name: 'Mario José Molina-Pasquel Henríquez',
        profession: 'chemist',
    }, {
        id: 2,
        name: 'Mohammad Abdus Salam',
        profession: 'physicist',
    }, {
        id: 3,
        name: 'Percy Lavon Julian',
        profession: 'chemist',
    }, {
        id: 4,
        name: 'Subrahmanyan Chandrasekhar',
        profession: 'astrophysicist',
    }];
```

```
    const chemists = people.filter(person =>
        person.profession === 'chemist'
    );

    const listItems = chemists.map(person =>
        <li>
            <p>
                <b>{person.name}:</b>{' ' + person.profession}
            </p>
        </li>
    );

    return (
        <>
            <ul>{listItems}</ul>
        </>
    )
}
export default ListDemo;
```

Hooks:

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

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

function Counter2() {
    const[counter,setCounter] = useState(0);

    const increment = () => {
        setCounter(counter+1);
    }
    const decrement = () => {
        setCounter(counter-1);
    }

    return(
        <>
            <h2>Counter (using Hooks) : {counter} </h2>
            <button onClick={increment}>+</button>
            <button onClick={decrement}>-</button>
        </>
    )
}
export default Counter2;
```

Forms:

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input.

In React, mutable state is typically kept in the state property of components, and only updated with setState().

Controlled vs Uncontrolled Components:

- **Controlled Components** (React way): Form elements whose value is controlled by React state.
- **Uncontrolled Components** (HTML way): Form elements manage their own state internally.

You typically manage form input values with **React state**.

Updates are handled via the onChange event and setState() or useState() in function components.

The form elements become **controlled components**, meaning their value is tied to the React state.

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

function LoginForm() {

    const [email, setEmail] = useState('');
    const [userpwd, setUserpwd] = useState('');

    const doLogin = () => {
        if (userpwd === '12345') {
            alert("Login Success with email " + email);
        }
        else {
            alert("Invalid credentials");
        }
    }

    return (
        <>
            <div>
                <label>Email</label>
                <input type="text" name="email" value={email}
                    onChange={event => setEmail(event.target.value)} />
            </div>
            <div>
                <label>Password</label>
                <input type="password" name="userpwd" value={userpwd}
                    onChange={event => setUserpwd(event.target.value)} />
            </div>
            <button onClick={doLogin}>Login</button>
        </>
    )
}
export default LoginForm;
```