Introduction to React

What is React?

React is an open-source **JavaScript library** for building dynamic user interfaces. Maintained by Facebook and the developer community, it enables efficient updates without reloading the entire page.

React uses a **virtual DOM** to optimize rendering and improve performance.

Applications are built using reusable components, which can manage internal state and respond dynamically to user interactions.

Why Use React?

Efficient Virtual DOM – Updates only necessary parts for better performance.

Component-Based Architecture – Enables reusable and modular UI development.

Better State Management – Automatically updates UI when data changes.

Scalability – Ideal for large applications like e-commerce and dashboards.

Single Page Applications (SPAs) – Dynamic content updates without reloading.

React Native – Allows mobile app development using React.

React Example

Below is an example demonstrating how to render a simple 'Hello, World!' message using React:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>React Example</title>
  <script src="https://fb.me/react-15.2.1.js"></script>
  <script src="https://fb.me/react-dom-15.2.1.js"></script>
</head>
<body>
  <div id="root"></div>
  <script src="./index.js" type="text/javascript"></script>
</body>
</html>
```

index.js

```
let root = document.querySelector("#root");
var rElement = React.createElement('h1', null, 'Hello, world!');
ReactDOM.render(rElement, root);
```

Advantages of React

Feature	Vanilla JavaScript (Method 1)	React (Method 2)
Method	Direct DOM Manipulation	Virtual DOM
Performance	Slower (Updates entire DOM)	Faster (Only updates necessary parts)
Code Complexity	More manual work needed	Cleaner, reusable components
State Management	Manual DOM updates	React handles updates automatically
Scalability	Difficult for large apps	Ideal for large apps

React Components

Introduction

A React component is the building block of a React application. Components allow developers to divide the UI into independent, reusable pieces and handle each part separately.

React components are JavaScript functions or classes that return React elements (JSX), which define the structure of the UI.

Types of React Components

1. Functional Components (Stateless)

- These are simple JavaScript functions.
- They return JSX (HTML-like syntax) or React elements.
- Best for UI rendering and presentational logic.

Example: Functional Component

```
function Welcome() {
  return <h1>Hello, React!</h1>;
}
export default Welcome;
```

2. Class Components (Stateful)

- Before React 16.8, class components were widely used to manage state.
- They extend React.Component and contain a render() method.
- They can have lifecycle methods.

Example: Class Component

```
import React from "react";

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

JSX in React Components

React components use JSX (JavaScript XML), which looks like HTML but is actually JavaScript. JSX allows us to write UI elements inside JavaScript.

Example:

```
function Welcome() {
  return (
    <div>
      <h1>Welcome to React</h1>
      <p>This is a functional component.</p>
    </div>
  );
}
```

Props (Properties) in Components

Props (short for 'properties') are a mechanism in React for passing data from one component to another, typically from a parent component to a child component. They allow components to be dynamic and reusable by enabling the transfer of different values without modifying the component itself.

Key Characteristics of Props

• Read-Only: Props are immutable, meaning they cannot be modified inside the child component that receives them. This ensures one-way data flow, keeping the application predictable.

• Passed from Parent to Child: A parent component sends props to a child component as attributes in JSX. The child component accesses these values using props.propertyName.

• Enhances Reusability: By passing different values through props, the same component can be reused in multiple places with different data.

• Supports Functional and Class Components: Props work with both functional components (using props as a function parameter) and class components (accessed via this.props).

Example: Using Props in a Functional Component

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

function App() {
  return <Greeting name="YourName" />;
}
```

Explanation:
- The App component passes a name prop with the value ‘YourName’ to the Greeting component.
- The Greeting component receives props and displays 'Hello, YourName!'.

Default Props

React allows setting default props to ensure a component has a fallback value if no prop is provided.

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

Welcome.defaultProps = {
  user: "Guest",
};
```

If no user prop is passed, the output will be 'Welcome, Guest!'.

Props vs. State

Feature	Props	State
Mutability	Immutable	Mutable
Ownership	Passed from parent to child	Managed within the component
Usage	Used for passing data	Used for handling component behavior

In React, **state** is a built-in object used to store data **inside a component**. It allows components to remember values and update the UI when the data changes.

---

1. Why Use State?

State makes components **dynamic and interactive**. Instead of hardcoding values, React can update UI **automatically** when state changes.

React Hooks (State in Functional Components)

React Hooks are **special functions** that let you use state and other React features in functional components **without writing a class**.

Example: State in a Functional Component using Hooks

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

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

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}
export default Counter;
```

Advantages of React Components

Feature	Explanation
Reusability	Components can be reused in multiple places.
Performance	React's Virtual DOM makes updates faster.
Modularity	UI is split into small, manageable pieces.
Easier Debugging	Each component works independently.

Understanding useState in React

useState is a React Hook that allows functional components to manage state. Before Hooks, state management was only possible in class components using this.state.

- **Why is useState Important?**
  - Enables stateful logic in functional components.
  - Updates component state dynamically, triggering re-renders.

How to Use useState?

The useState Hook returns an array with two values:

1. The current state value.
2. A function to update the state.

Example

const [state, setState] = useState(initialValue);

- state → Holds the current state value.
- setState → Updates the state and triggers a re-render.
- initialValue → The starting value of the state.

What If we do not use useState()?

Feature	Vanilla JavaScript	React (useState)
State Management	Manually update and track state	Automatically manages state updates
DOM Manipulation	Directly modify the DOM (e.g., document.getElementById().innerText = ...)	Uses **Virtual DOM**, optimizing rendering
Performance	Frequent DOM updates can be **slow**	**Efficient updates**, only re-renders necessary parts
Reactivity	No built-in reactivity; updates must be manually handled	**Auto re-renders** UI when state changes
Code Complexity	More boilerplate, harder to manage in large projects	Cleaner, modular, and **scalable**
Component Reusability	Harder to reuse dynamic elements	**Reusable components** with isolated state