Theory Assignment

Module : 9

• Question 1: What are components in React? Explain the difference between functional ?

Answer : In React, components are independent, reusable "building blocks" that define parts of your user interface. They accept inputs called props and encapsulate both structure (via JSX) and behavior. These components can manage their own state and be composed to build complex UIs react.dev+15legacy.reactjs.org+15youtube.com+15.

⚙️ **Functional Components**

- **Implemented as plain JavaScript functions (including modern arrow functions) that accept props and return JSX .**
- **With React 16.8+, Hooks allow functional components to manage state (useState), run effects (useEffect), and more—features once exclusive to class components reddit.com+5en.wikipedia.org+5dhiwise.com+5.**
- **Pros:**
    - **Shorter and cleaner syntax—no this, no constructors reactnative.dev+15dhiwise.com+15twilio.com+15.**

    - **Easier to test and reason about.**
    - **Slightly better performance and encourages modern functional programming practices youtube.com+1twilio.com+1medium.com+9dhiwise.com+9en.wikipedia.org+9.**

- **Ideal for: Most new development, especially presentational or simpler components uxpin.com+14dhiwise.com+14geeksforgeeks.org+14.**

Example:

jsx

CopyEdit

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

🏛️ **Class Components**

- **Defined as ES6 classes extending React.Component, with a required render() method returning JSX reddit.com+15legacy.reactjs.org+15w3schools.com+15.**
- **State is managed via this.state, updated with this.setState(), and lifecycle stages are handled through**

methods like componentDidMount, componentDidUpdate, and componentWillUnmount .

- **Pros:**
  - **Offer explicit lifecycle control and can implement error boundaries (only available in class components currently) dhiwise.com+1medium.com+1.**
- **Cons:**
  - **More verbose, boilerplate-heavy, and require this bindings .**
  - **Hook-based libraries often won't integrate natively.**
  - **Recognized as legacy; React recommends functional components for new code twilio.com+7en.wikipedia.org+7reddit.com+7react.dev+9react.dev+9reddit.com+9.**

Example:

jsx

CopyEdit

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

}

⚖️ **Comparison Summary**

Feature	Functional Components	Class Components
Syntax & Boilerplate	Simple functions, minimal code legacy.reactjs.org+5medium.com+5stackoverflow.com+5	Requires classes, constructors, render()
State Management	useState, useReducer (via Hooks)	this.state, this.setState()
Lifecycle Handling	useEffect covers mount/update/unmount	componentDidMount, componentDidUpdate, etc.
this Keyword	No need for this	Requires this, often needs binding

Feature	Functional Components	Class Components
Performance	Lightweight and optimized	Slightly heavier due to class instances
Use Cases	Recommended for most use cases	Legacy codebases, error boundaries

Question 2: How do you pass data to a component using props?

1. Answer: **Pass data in the parent component:**
   You include the child component in JSX and pass values as attributes.

jsx

CopyEdit

```
function ParentComponent() {
  return (
    <ChildComponent name="Alice" age={25} />
  );
}
```

2. **Receive props in the child component:**
   The child component can access the passed data using the props object.

jsx

CopyEdit

```
function ChildComponent(props) {
  return (
    <div>
      <p>Name: {props.name}</p>
      <p>Age: {props.age}</p>
    </div>
  );
}
```

✅ **Using Destructuring (Recommended for cleaner code):**

jsx

CopyEdit

```
function ChildComponent({ name, age }) {
  return (
    <div>
      <p>Name: {name}</p>
```

```
      <p>Age: {age}</p>
    </div>
  );
}
```

Question 3: What is the role of render() in class components?

- Answer : It's the **only required** method in a class-based component—without it, the component won't work en.wikipedia.org+9legacy.reactjs.org+9dhiwise.com+9.
- When called, it examines the component's this.props and this.state, then returns what should be displayed:
    - JSX/React elements (e.g., , other components)
    - Strings or numbers (rendered as text nodes)
    - Fragments or arrays (for grouping multiple items)
    - Portals (for rendering outside the hierarchy)
    - null, undefined, or false (renders nothing) legacy.reactjs.org.

🚫 Purity & Side-Effects

- render() must be a **pure function**: it shouldn't modify state, cause side effects, or interact with the browser directly—it should only compute and return UI based on its inputs .
- Any side effects (like HTTP requests or subscriptions) belong in lifecycle methods such as componentDidMount, componentDidUpdate, etc. en.wikipedia.org+10legacy.reactjs.org+10react.dev+10.

🔄 Component Lifecycle Integration

- Called during both **mounting** and **updating** phases:
  1. **Mount**: constructor() → ... → render() → componentDidMount()
  2. **Update**: props/state changes → [optional shouldComponentUpdate()] → render() → componentDidUpdate() stackoverflow.com+6legacy.reactjs.org+6react.dev+6.
- Note: If shouldComponentUpdate() returns false, React **skips** render() and later lifecycle methods legacy.reactjs.org+1react.dev+1.

🔍 Why Class Needs render(), But Functional Doesn't

- A **functional component** *is* the render()—it's literally just a function that returns UI.
- A **class component**, on the other hand, uses the class structure for encapsulating state, context, and lifecycle methods—so React mandates that the **UI logic** resides in a method named render() react.dev+9stackoverflow.com+9w3schools.com+9.

Props & State

Question 1: What are props in React.js? How are props different from state?

Answer : What are Props in React.js?

Props (short for properties) are used in React to pass data from one component to another, typically from a parent component to a child component.

They allow you to customize components and make them dynamic and reusable.

- **Example:**

jsx

CopyEdit

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

<Welcome name="Alice" />
```

In the example above, "Alice" is passed as a prop to the Welcome component.

✅ Difference Between Props and State

Feature	Props	State
Definition	Props are used to pass data to components	State is used to manage internal data of a component
Mutability	Immutable (read-only)	Mutable (can be updated with setState or useState)
Usage	Passed from parent to child	Managed within the component itself
Control	Controlled by parent component	Controlled by the component itself

Feature	Props	State
Purpose	For configuration	For handling dynamic behavior

Question 2: Explain the concept of state in React and how it is used to manage component data?

- **Answer : State is a component-specific data structure used to track variables that can change between renders—like form inputs, toggles, counters, etc. When state changes, React automatically re-renders that component to reflect the new data dhiwise.com+9react.dev+9medium.com+9.**
- **Unlike local variables, state is persisted across renders and updates trigger UI updates react.dev.**

⚙️ **Using State in Functional Components**

- **Use the useState Hook to declare state:**

jsx

CopyEdit

import { useState } from 'react';

```
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <button onClick={() => setCount(count + 1)}>
      Count: {count}
    </button>
  );
}
```

- How it works:
  - useState(initialValue) returns [value, setterFunction].
  - Calling the setter (e.g., setCount) updates the state and triggers a re-render with the new value w3schools.com+1legacy.reactjs.org+1react.dev.
- You can have multiple state variables in one component—each declared separately crsinfosolutions.com+9react.dev+9w3schools.com+9.

🏛️ State in Class Components

- Declare state in the constructor:

jsx

CopyEdit

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return <button onClick={this.increment}>Count: {this.state.count}</button>;
  }
}
```

- **Use this.setState() to update state. React re-renders the component after each update** [dhiwise.com+8legacy.reactjs.org+8medium.com+8](#).

✨ Why State Matters

1. **Persistent Memory Across Renders**
   State lives across re-renders, so it can store values like user input, timer ticks, etc. .
2. **Triggers UI Updates**
   Changing state signals React to re-render, ensuring the displayed UI stays in sync with data crsinfosolutions.com+9react.dev+9w3schools.com+9.
3. **Supports Interactive UIs**
   Enables dynamic behaviors—clicks, typing, toggles, async processes—all reflected through state changes medium.com.

🔁 Data Flow and State Management

- Local state is private to a component.
- To share state between components:
  - Use props to pass data down.
  - Use callbacks to send events or changes upward.
  - Or lift state up to a common parent for shared control

Question 3: Why is this.setState() used in class components, and how does it work?

Answer : In React class components, this.setState() is used to update the component's state and trigger a re-render. Directly modifying this.state does not update the UI or trigger a re-render — React will ignore changes unless setState() is used.

✅ **How does this.setState() work?**

- **this.setState() merges the new state with the current state.**
- **It tells React to re-render the component with the updated data.**
- **It is asynchronous, meaning updates may not happen immediately.**

🔹 **Syntax:**

jsx

CopyEdit

```
this.setState({ key: newValue });
```

Example:

jsx

CopyEdit

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increase</button>
      </div>
    );
```

```
  }
}
```

✅ **Functional form of setState() (when next state depends on previous):**

```
jsx
CopyEdit
this.setState((prevState) => ({
  count: prevState.count + 1
}));
```