Theory Assignment

Hooks(useState, useEffect, useReducer, useMemo, useRef, useCallback)

Question 1: What are React Hooks? How do useState() and useEffect() work?

Hooks are functions that let you use React features (like state and lifecycle methods) in **functional components**.

- **useState()**
  → Adds state to a functional component.
  Example:

js

CopyEdit

```
const [count, setCount] = useState(0);
```

  - count is the state value.
  - setCount is used to update it.

- **useEffect()**
  → Runs code when the component mounts, updates, or unmounts.
- **Example:**

```
useEffect(() => {
  console.log("Component Loaded");
}, []);
```

  - Empty [] runs it **only once** (on mount).

Question 2: What problems did hooks solve? Why are they important?

Hooks solved these problems:

- ✓ No need for class components to use state or lifecycle.
- ✓ Easier code sharing with custom hooks.
- ✓ Cleaner and shorter code.
  - ❖ Hooks made **functional components more powerful** and reduced complexity.

Question 3: What is useReducer? How do we use it in React?

useReducer() is like useState(), but better for **complex state logic** (e.g., with multiple values or actions).

Example:

```
const reducer = (state, action) => {

  if (action.type === "increment") return { count: state.count + 1 };

  return state;

};

const [state, dispatch] = useReducer(reducer, { count: 0 });

<button onClick={() => dispatch({ type: "increment" })}>Add</button>
```

Question 4: What is the purpose of useCallback() and useMemo()?

Both are **performance optimization hooks**.

- **useCallback()**: Memoizes a function to prevent re-creating it on every render.
- **useMemo()**: Memoizes a **calculated value** to prevent recalculating every time.

Question 5: Difference between useCallback() and useMemo()?

Feature	**useCallback()**	**useMemo()**
Returns	A memoized **function**	A memoized **value**
Use when	Function is passed as prop	Expensive calculation (like filtering)
Example	useCallback(() => {}, [deps])	useMemo(() => result, [deps])

Question 6: What is useRef()? How does it work?

useRef() gives you a **mutable reference** that doesn’t trigger re-renders. Often used for:

- Accessing DOM elements directly.
- Storing values across renders without re-rendering.

Example:

```
const inputRef = useRef();
const focusInput = () => {
  inputRef.current.focus();
}

<input ref={inputRef} />
<button onClick={focusInput}>Focus</button>
```