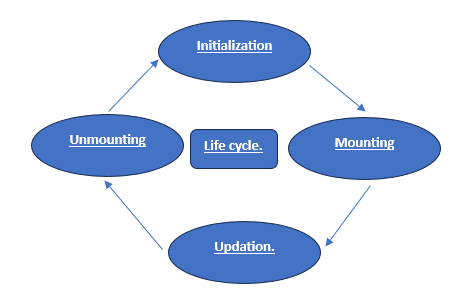
**Module – 4 Lists and Hooks**

**Explain Life cycle in Class Component and functional component with Hooks**



**Class Components**

In class components, React provides several special methods called lifecycle methods that are called at different stages:

**1. Mounting (when the component is added to the DOM)**

- constructor(): Sets up initial state and other initializations.

- componentDidMount(): Runs after the component is first added to the DOM. Good for fetching data.

**2. Updating (when the component’s state or props change)**

- shouldComponentUpdate(): Decides if the component should re-render or not.

- componentDidUpdate(): Runs after the component updates in the DOM. Good for updating the DOM based on new data.

**3. Unmounting (when the component is removed from the DOM)**

- componentWillUnmount(): Runs right before the component is removed. Good for cleanup like removing timers.

**Functional Components with Hooks**

Functional components use hooks to manage state and side effects:

1. useState: Manages state in a functional component.

```javascript

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

```

2. useEffect: Manages side effects like fetching data, directly manipulating the DOM, or setting up subscriptions.

- To run code when the component mounts:

```javascript

useEffect(() => {

console.log('Component did mount');

}, []); // Empty array means this runs only once after the first render.

```

- To run code when the component updates:

```javascript

useEffect(() => {

console.log('Component did update');

}); // No second argument means this runs after every render.

```

- To clean up before the component unmounts:

```javascript

useEffect(() => {

return () => {

console.log('Component will unmount');

};

}, []); // Empty array means this clean-up runs only once before the component unmounts.

```

**Simple Comparison**

- Class components use lifecycle methods like `componentDidMount` and `componentWillUnmount`.

- Functional components use hooks like `useEffect` to handle what those lifecycle methods do.

Functional components with hooks are generally easier to read and write, and they allow you to manage state and side effects more directly within the component's function.

