Q1. What is Redux, and why is it used in React applications? Explain the core concepts of actions, reducers, and the store.

Ans: Redux is a state management library for JavaScript applications, commonly used with React.
It helps manage the global state of an application in a predictable and centralized way.

- ➢ Why Redux is Used in React Applications

In large React apps:

- Data needs to be shared between many components
- Props drilling (passing data parent → child → child) becomes complex
- State management becomes hard to track and debug

Redux solves these problems by:

- Providing a single source of truth
- Making state changes predictable
- Improving debugging and maintainability

Core Concepts of Redux

1. Store

The store is the central container that holds the entire state of the application.

- There is only one store in a Redux app
- It stores the current state
- It allows state to be updated via reducers

2. Actions

An action is a plain JavaScript object that describes what happened in the app.

- Actions must have a type
- They do not change state directly
- They only describe the intent

Example:

```
const incrementAction = {
  type: "INCREMENT"
};
```

3. Reducers

A reducer is a pure function that decides how the state should change based on an action.

- Takes current state and action
- Returns a new state
- Does not modify the original state

Example:

```
function counterReducer(state = { count: 0 }, action) {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    default:
      return state;
  }
}
```

Q2. How does Recoil simplify state management in React compared to Redux?

Ans: Recoil is a modern state management library created by Meta (Facebook) that is designed specifically for React.
It simplifies state management by making it more component-friendly, flexible, and less verbose than Redux.

Key Differences: Recoil vs Redux

1. No Centralized Store Required

- Redux:
  Uses a single global store for the entire application.

- Recoil:
  Uses atoms (small pieces of state) that can exist independently.

2. Less Boilerplate Code

- Redux requires:
  - Actions
  - Action creators
  - Reducers
  - Store setup
- Recoil requires:
  - Atoms
  - Selectors (optional)

3. Component-Level State Access

- Redux:
  Components subscribe to the store and select data.
- Recoil:
  Components directly read and write atoms using hooks.

Example:

```
const count = useRecoilValue(counterAtom);
```

4. Better Performance

- Redux:
  A state update may cause many components to re-render.
- Recoil:
  Only components using the specific atom re-render.

5. Built-in Support for Derived State

- Redux:
  Derived state requires selectors and extra logic.
- Recoil:
  Uses selectors that automatically update when dependencies change.

6. Async State Made Easy

- Redux:
  Needs middleware like Thunk or Saga.
- Recoil:
  Handles async state natively using selectors.