Advance React Training – Day 3 Assignment

1. What is useEffect? What are the different behaviors of useEffect? What is a dependency array?
   - ⇨ useEffect is a React hook that manages side effects, such as data fetching, setting up subscriptions, or manually changing the DOM, which are tasks that are not performed during the rendering of a component.
     Different Behaviors of useEffect:

   a) Default Execution: When useEffect is used with only a callback function, it executes after every render of the component, ensuring the side effects run whenever the component updates.
   b) Initial Render Only: By passing an empty dependency array as the second argument, useEffect will only execute once, immediately after the initial render. This is useful for initialization logic that should only run once.
   c) Conditional Execution: When provided with a dependency array containing specific values, useEffect will execute only when any of those values change. This allows you to control precisely when the side effects should run based on changes in your component's state or props.
   d) Cleanup Function: useEffect can return a cleanup function that is invoked before the component unmounts or before the effect is re-executed. This is essential for cleaning up resources like subscriptions, timers, or event listeners to prevent memory leaks.

   Dependency Array: The dependency array is a list of values that useEffect monitors. When any value in this array changes, the effect runs again. If the array is empty, the effect runs only once after the initial render, ideal for one-time setup tasks.

2. What is useRef and When to Use It?
   - ⇨ useRef is a React hook that creates a mutable object which persists across component re-renders without causing re-renders when its properties are updated. It is most commonly used to access and interact with DOM elements directly, as well as to store mutable values that should not trigger re-renders, such as a reference to a timer.

3. How to Reuse Hook Logic in React?

⇨ To reuse logic within hooks, you can create custom hooks. Custom hooks are functions that leverage existing React hooks to encapsulate common logic, such as data fetching, form handling, or other reusable behaviors. This promotes cleaner, more modular code by abstracting repetitive logic into standalone functions, which can then be easily shared across multiple components. Custom hooks can also utilize other custom hooks or built-in hooks to build complex functionality in a more organized manner.