**React hooks**

React hooks are functions to provide a more concise and readable way to manage state and side effects in functional components. React hooks are provide a alternative to write class based components and offers a alternative approach to state management and life cycle methods.

React hook are introduced in React16.8 version. First released in October 2018.

**Pros of react hooks**

**Simpler and cleaner code – Improve code maintainability**

React hooks eliminate the need for class components and make the code more concise and easier to read. This can lead to improved code maintainability.

**Reusability – Code reuse across differen t components**

We can encapsulate and reuse stateful logic creating custom hooks. This promotes code reuse across different components.

**Functional components**

React hooks allow us to use functional components for both stateful and stateless logic, reducing the cognitive overhead of dealing with class component lifecycle methods.

**Easier testing**

Functional components with hooks are easier to test since we can test individual functions and logic in isolation.

**No more wrapper hell**

With class components, we might have to use higher-order components or render props to share state and logic. React eliminate the need for such wrapper , making our codebase more straightforward.

**Improved performance**

React hooks help optimize our components by allowing us to use memorization techniques like ‘useMemo’ and ‘useCallback’ to prevent unnecessary re-renders.

**Better tooling**

React development tools have improved support for hooks, making it easier to inspect and debug our components.

**Cons of react hooks**

**Learning curve**

Learning to use hooks effectively can be challenging, especially for developers who are already familiar with class components. It may take some tim to get used to the new syntax and concept.

**Breaking changes**

Introduction of hooks was a significant change in react, and it required developer to update their existing codebase. This could be disruptive, especially for large projects.

**Backwards compatibility**

React hooks are only available in react 16.8and later, which means older react codebases using class components can’t easily migrate to hooks without significant refactoring.

**Less prescriptive**

While the flexibility of hooks is a pro, it can also be a con. Some developer may struggle with deciding where to put certain logic or state, leading to inconsistencies in code structure.

**Potential for performance pitfalls**

Although hooks can improve performance when used correctly, misuse or overuse of hooks can lead to performance issues, especially when not optimizing expensive operations with ‘useMemo’ or ‘useCallback’

**Tooling challenges**

While react development tools have improved for hooks there may still be some challenges with debugging and inspecting components, especially when dealing with custom hooks

1. useState()
2. useEffect()
3. useMemo()
4. useCallback()
5. useRef()
6. useContext()
7. useReducer()
8. useLayoutEffect()
9. useDebugValue()