**Class Component Life Cycle**

Class components have a number of built-in methods or lifecycle hooks that allow you to control the rendering and updating of your components. Here is an overview of the most commonly used lifecycle methods:

constructor(): This is called when the component is first created. It is used to set the initial state and bind event handlers.

render(): This is the only required method in a class component. It returns the JSX that will be rendered on the screen.

componentDidMount(): This is called after the component has been rendered for the first time. It is often used to fetch data from an API or set up event listeners.

shouldComponentUpdate(): This method is called before the component updates. It allows you to control whether or not the component should re-render based on changes to its props or state.

componentDidUpdate(): This method is called after the component has updated. It is often used to update the DOM or fetch new data based on changes to the props or state.

componentWillUnmount(): This method is called just before the component is removed from the DOM. It is used to clean up any resources used by the component, such as event listeners.

Functional Component Life Cycle with Hooks

Functional components with hooks allow you to use state and other React features without writing a class. They also have their own set of lifecycle hooks, which are implemented using the useEffect() hook. Here is an overview of the most commonly used hooks:

useState(): This hook allows you to add state to your functional component. It returns an array with two values: the current state value and a function to update it.

useEffect(): This hook allows you to perform side effects in your functional component. It takes a function as its first argument, which is called after every render. You can also pass a second argument to control when the effect is called.

useContext(): This hook allows you to access data that is stored in a context object. It takes a context object as its argument and returns the current value of that context.

useReducer(): This hook allows you to manage complex state in your functional component. It takes a reducer function and an initial state value as its arguments, and returns the current state value and a dispatch function to update it.

useCallback(): This hook allows you to memoize a function so that it is only re-created when its dependencies change.

useMemo(): This hook allows you to memoize a value so that it is only re-calculated when its dependencies change.

useRef(): This hook allows you to create a mutable reference that persists across renders. It returns a ref object that has a .current property, which can be used to store and update the value of the reference.