React Hooks Cheat Sheet

1. useState

Allows you to add state to a functional component. It returns a stateful value and a function to update it. Each call to setState causes the component to re-render with the new state.

Example:

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

<button onClick={() => setCount(count + 1)}>Increment</button>
```

2. useEffect

Performs side effects in function components. Runs after the render and can be used for data fetching, DOM updates, subscriptions, etc. The dependency array controls when it runs.

Example:

```
useEffect(() => {
  console.log("Component mounted or count changed");
}, [count]);
```

3. useRef

Returns a mutable ref object whose .current property is initialized to the passed argument. Useful to persist values across renders without causing re-renders.

Example:

```
const inputRef = useRef(null);

<input ref={inputRef} />
<button onClick={() => inputRef.current.focus()}>Focus</button>
```

4. useMemo

Memoizes the result of an expensive computation and only recomputes when dependencies change. Helps optimize performance by avoiding unnecessary recalculations.

Example:

```
const expensiveValue = useMemo(() => {
  return computeHeavyValue(num);
}, [num]);
```

5. useCallback

Returns a memoized callback function that only changes if dependencies change. Useful for passing stable

functions to child components.

Example:

```
const handleClick = useCallback(() => {
  console.log("Clicked");
}, []);
```

6. useContext

Allows you to consume a React context directly in a functional component without needing a <Context.Consumer>.

Example:

```
const value = useContext(MyContext);
```

7. useReducer

An alternative to useState for managing complex state logic. It accepts a reducer function and an initial state, and returns the current state paired with a dispatch method.

Example:

```
const [state, dispatch] = useReducer(reducer, initialState);

function reducer(state, action) {
  switch(action.type) {
    case "increment": return { count: state.count + 1 };
    default: return state;
  }
}
```

8. useLayoutEffect

Similar to useEffect, but it fires synchronously after all DOM mutations. Use it when you need to read layout from the DOM and re-render synchronously.

Example:

```
useLayoutEffect(() => {
  console.log("DOM updated");
}, []);
```

9. useImperativeHandle

Customizes the instance value exposed when using ref. Typically used with forwardRef to expose specific methods or properties.

Example:

```
useImperativeHandle(ref, () => ({
  focus: () => inputRef.current.focus()
}));
```

10. useDebugValue

Used to display a label for custom hooks in React DevTools. It doesn't affect app behavior.

Example:

```
useDebugValue(user ? "Logged In" : "Logged Out");
```

11. Custom Hooks

Custom hooks are JavaScript functions whose names start with 'use'. They can call other hooks and allow you to extract component logic into reusable functions.

Example:

```
function useCounter(initialValue) {
  const [count, setCount] = useState(initialValue);
  const increment = () => setCount(c => c + 1);
  return { count, increment };
}
```