Notes

The **useState** hook is a way to manage state in functional components in React. It takes an initial value as an argument and returns an array with two elements: the first element is the current state value, and the second element is a function to update the state.

Hooks also come with a set of rules, that you need to follow while using them. This applies to all React hooks, including the **useState** hook that you just learned.

* You can only call hooks at the top level of your component or your own hooks.
* You cannot call hooks inside loops or conditions.
* You can only call hooks from React functions, and not regular JavaScript functions.

In addition to the **useState** hook, there are other hooks that come in handy such as **useContext**, **useMemo**, **useRef**, etc. When you need to share logic and reuse the same logic across several components, you can extract the logic into a custom hook. Custom hooks offer flexibility and can be used for a wide range of use-cases such as form handling, animation, timers, and many more.

**The useRef hook**

We use the **useRef** hook to access a child element directly.

When you invoke the **useRef** hook, it will return a **ref** object. The **ref** object has a property named **current**.

function TextInputWithFocusButton() {

  const inputEl = useRef(null);

  const onButtonClick = () => {

    // `current` points to the mounted text input element

    inputEl.current.focus();

  };

  return (

    <>

      <input ref={inputEl} type="text" />

      <button onClick={onButtonClick}>Focus the input</button>

    </>

  );

}

Using the ref attribute on the input element, I can then access the current value and invoke the focus() method on it, thereby focusing the input field.

There are situations where accessing the DOM directly is needed, and this is where the useRef hook comes into play.

The distinction between stateful and

stateless components is that a stateful component holds

states as internal data and

its state changes based on

the way that the app is built;

often as a result of user actions.

A stateless component however,

doesn't store states and

any changes must be inherited through props.

 although

a stateless component cant directly pass state,

it can still trigger actions that will

update the state of other components.