Use State => save Data in State

Import

```jsx
import { useState } from 'react';

function App() {
  const [count, setCount] = useState(0)

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

Destruct

Init

Use Effect

- runs every update
- second arg is an array of states if its stat then its only return when a specific state changed

old way (class based component)

```jsx
componentDidMount() {
  // initialized
}

componentDidUpdate() {
  // state updated
}

componentWillUnmount() {
  // destroyed
}
```

Start

update

End

empty array means no update

```jsx
useEffect(
  () => {
    fetch('foo').then(() => setLoaded(true) )
  },
  [count]
)
```

RUN when count changes

```jsx
useEffect(() => {
  alert('hello side effect!')

  return () => alert('goodbye component!')
})
```

RUN before component is removed from UI

use context

```
const moods = {
  happy: '😀',
  sad: '😭'
}
const MoodContext = createContext(moods);
```

```
function App(props) {

  return (
    <MoodContext.Provider value={moods.happy}>

        <MoodEmoji />

    </MoodContext.Provider>
  );
}
```

```
function MoodEmoji() {
  const mood = useContext(MoodContext);

  return <p>{ mood }</p>
}
```

consume value from nearest parent provider

use ref

```
function App() {

  const count = useRef(0);

  return (
      <button onClick={() => count.current++ }>
        {count.current}
      </button>
  );
}
```

mutable value
does **NOT** re-render UI

use Reducer

init as second argument

```
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return state + 1;
    case 'decrement':
      return state - 1;
    default:
      throw new Error();
  }
}
```

```
function App() {
  const [state, dispatch] = useReducer(reducer, 0);
  return (
    <>
      Count: {state}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

use Memo

```
function App() {

  const [count, setCount] = useState(60);


  const expensiveCount = useMemo()


  return <></>;
}
```

CAUTION
use only as needed
for expensive calculations

```
function App() {

  const [count, setCount] = useState(60);

  const expensiveCount = useMemo(() => {
    return count ** 2;
  }, [count])


  return <></>;
}
```

use callback

```
function App() {

  const [count, setCount] = useState(60);

  const showCount = useCallback(() => {
    alert(`Count ${count}`)
  }, [count])


  return <> <SomeChild handler={showCount} /> </>;
}
```

use Layout Effect

```
function App() {

  const myBtn = useRef(null);

  useLayoutEffect(() => {
    const rect = myBtn.current.getBoundingClientRect();

    console.log(box.height);

  })


  return <><button ref={ref}></button></>
}
```

Use Debug Value

→ add Debug to React Dev tools on Custom hooks