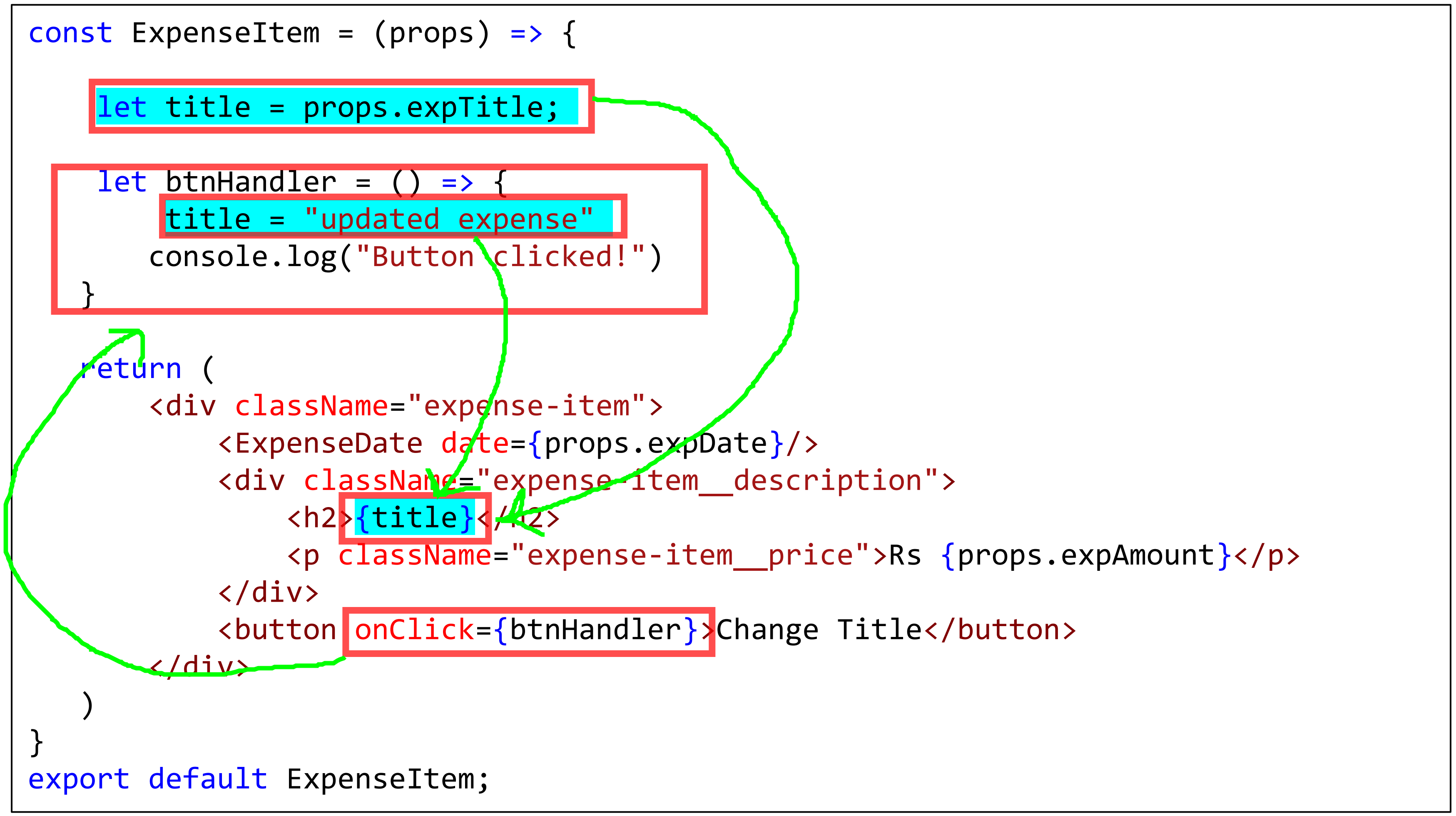
STATEFUL COMPONENTS

React State

- The state is a built-in React object that is used to contain data or information about the component.
- A component's state can change over time; whenever it changes, the component re-renders.
  - The change in state can happen as a response to user action or system-generated events and these changes determine the behavior of the component and how it will render.
- A component with state is known as stateful component.

- State allows us to create components that are dynamic and interactive.
  - State is private, it must not be manipulated from the outside.
  - Also, it is important to know when to use 'state', it is generally used with data that is bound to change.

Component without state



```
const ExpenseItem = (props) => {

    let title = props.expTitle;

    let btnHandler = () => {
        title = "updated expense"
        console.log("Button clicked!")
    }

    return (
        <div className="expense-item">
            <ExpenseDate date={props.expDate}/>
            <div className="expense-item__description">
                <h2>{title}</h2>
                <p className="expense-item__price">Rs {props.expAmount}</p>
            </div>
            <button onClick={btnHandler}>Change Title</button>
        </div>
    )
}
export default ExpenseItem;
```

React Hooks

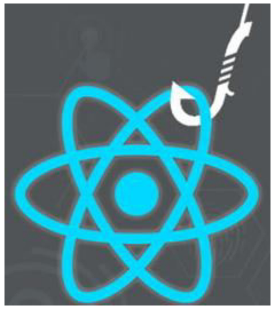
- Hooks allow us to "hook" into React features such as state and lifecycle methods
  - React Hooks are special functions provided by React to handle a specific functionality inside a React functional component.
  - Eg React provides *useState()* function to manage state in a functional component.
  - When a React functional component uses React Hooks, React Hooks attach itself into the component and provides additional functionality.

- You must import Hooks from react
  - Eg : import React, { useState } from "react"; Here - useState is a Hook to keep track of the application state.

- There are some rules for hooks:
  - Hooks can only be called inside React function components.
  - Hooks can only be called at the top level of a component.
  - Hooks cannot be conditional
  - Hooks will not work in React class components.
  - If you have stateful logic that needs to be reused in several components, you can build your own custom Hooks

Working with “state” in functional component

- The React useState Hook allows us to track state in a function component.
- To use the useState Hook, we first need to import it into our component.
  - import { useState } from "react";
  - We initialize our state by calling useState in our function component.



```
import React, {useState} from 'react';

const UseStateComponent = () => {
    useState(); //hooks go here
}
```

  - useState accepts an initial state and returns two values:

1. The current state.
2. A function that updates the state.

  - Eg:

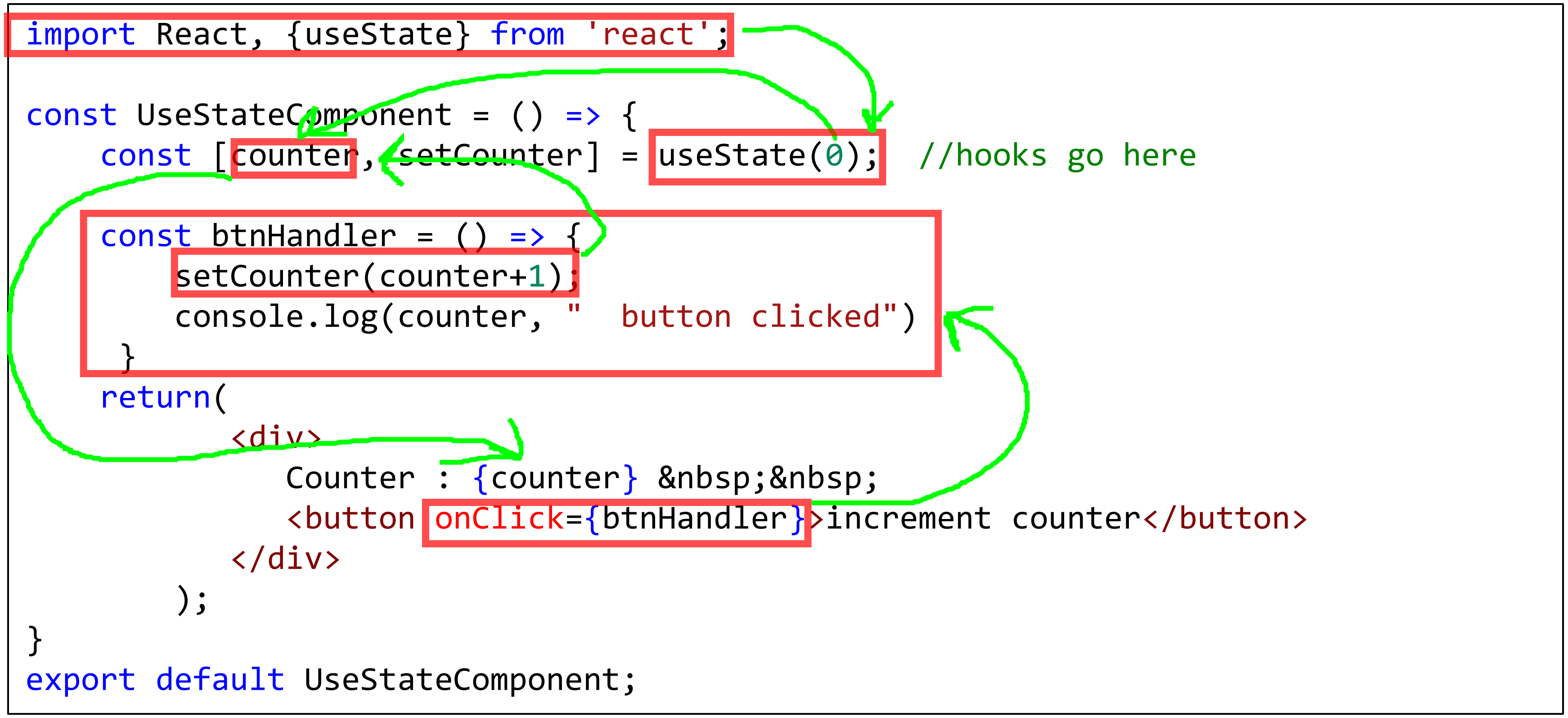
function to update state

```
function FavoriteColor() {
    const [color, setColor] = useState("");
}
```

initial state

  - The first value, color, is our current state.
  - The second value, setColor, is the fuction that is used to update our state.
  - Lastly, we set the initial state to an empty string: useState("")

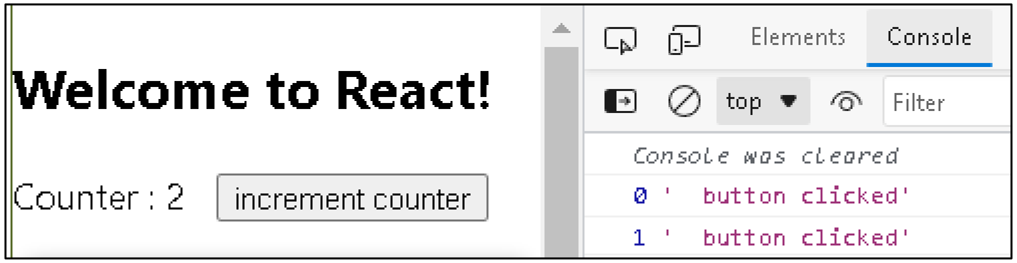
Working with “state” in functional component



```
import React, {useState} from 'react';

const UseStateComponent = () => {
    const [counter, setCounter] = useState(0);  //hooks go here

    const btnHandler = () => {
        setCounter(counter+1);
        console.log(counter, "  button clicked")
    }
    return(
        <div>
            Counter : {counter} &nbsp;&nbsp;
            <button onClick={btnHandler}>increment counter</button>
        </div>
    );
}
export default UseStateComponent;
```



Welcome to React!

Counter : 2

```
Console was cleared
0 '  button clicked'
1 '  button clicked'
```

Working with “state” in functional component

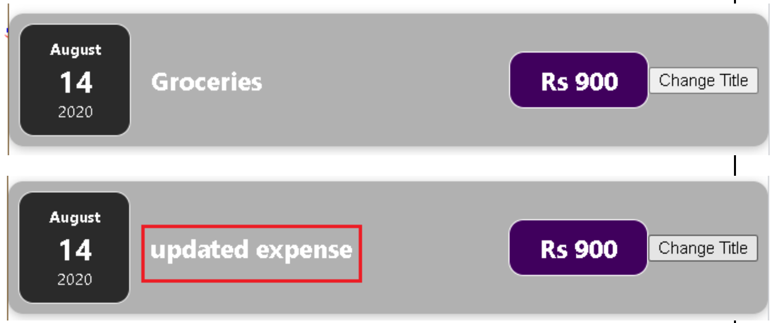
```
import React, {useState} from 'react'

const ExpenseItem = (props) => {

    const [title, setTitle] = useState(props.expTitle);

    let btnHandler = () => {
        setTitle("updated expense")
       console.log("Button clicked!")
    }

    return (
       <div className="expense-item">
           <ExpenseDate date={props.expDate}/>
           <div className="expense-item__description">
               <h2>{title}</h2>
               <p className="expense-item__price">Rs {props.expAmount}</p>
           </div>
           <button onClick={btnHandler}>Change Title</button>
       </div>
    )
}
export default ExpenseItem;
```



props and state

- props and state are CORE concepts of React.
  - Actually, only changes in props and/ or state trigger React to re-render your components and potentially update the DOM in the browser
- Props : allow you to pass data from a parent (wrapping) component to a child component.
  - Eg : AllPosts Component : “title” is the custom property (prop) set up on the custom Post component.
  - Post Component: receives the props argument. React will pass one argument to your component function; an object, which contains all properties you set up on <Post ... /> .
  - {props.title} then dynamically outputs the title property of the props object - which is available since we set the title property inside AllPosts component

```
//AllPosts
const posts = () => {
    return (
        <div>
            <Post title="My first Post" />
            <Post title="My second Post" />
        </div>
    );
}
```

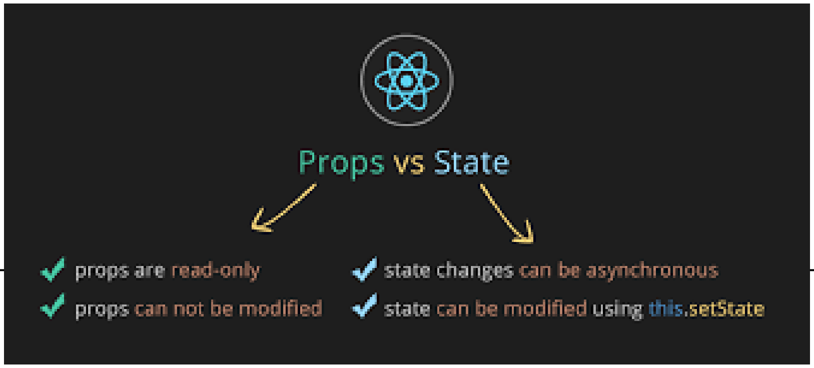
```
//Post
const post = (props) => {
    return (
        <div>
            <h1>{props.title}</h1>
        </div>
    );
}
```

props and state

- State : While props allow you to pass data down the component tree (and hence trigger an UI update), state is used to change the component’s, well, state from within.
  - Changes to state also trigger an UI update.
  - Example: NewPost Component: this component contains state . Only class-based components can define and use state . You can of course pass the state down to functional components, but these then can't directly edit it.

```
class NewPost extends Component { // state can only be accessed in class-based components!
  state = {
      counter: 1
  };

  render () { // Needs to be implemented in class-based components! Return some JSX!
      return (
          <div>{this.state.counter}</div>
      );
  }
}
```



- ✓ props are read-only
- ✓ props can not be modified
- ✓ state changes can be asynchronous
- ✓ state can be modified using this.setState

props and state

- Props are immutable. cannot be modified
- They should not be updated by the component to which they are passed.
- They are owned by the component which passes them to some other component.

- State is something internal and private to the component.
- State can and will change depending on the interactions with the outer world.
- State should store as simple data as possible, such as whether an input checkbox is checked or not or a CSS class that hides or displays the component

Simple example : props + state

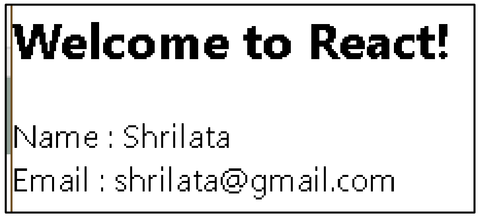
```
import React, {useState} from 'react'
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
   const [uname, setUname] = useState('Shrilata')
   const [email, setEmail] = useState('shrilata@gmail.com')

   return(
       <ChildComponent uname={uname} email={email} />
   );
}
export default ParentComponent;
```

```
function App() {
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <ParentComponent />
...
```

```
const ChildComponent = (props) => {
    return(
        <div>
            <div>Name : {props.uname}</div>
            <div>Email : {props.email}</div>
        </div>
    );
}
export default ChildComponent;
```



Welcome to React!

Name : Shrilata
Email : shrilata@gmail.com