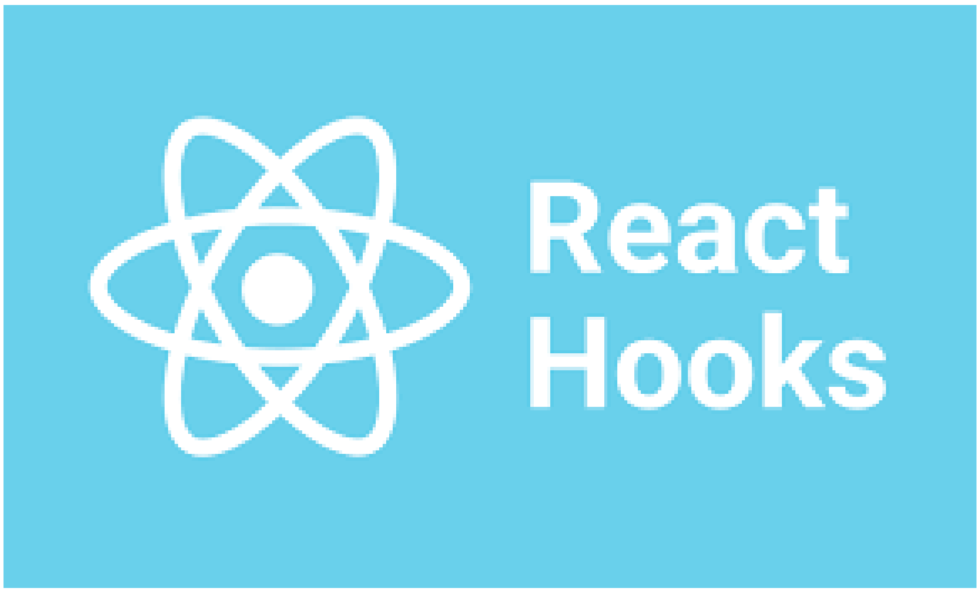
React States



What is a State?

The `state` object is where you store property values that belong to the component.

When the `state` object changes, the component re-renders.

Hooks are a set of predefined functions that let you use state and other React features without writing a Function .
Hooks are a new addition in React 16.8.

What is UseState?

UseState is one of the built-in react hooks available in 0.16.7 version.

During the initial render, the returned state (state) is the same as the value passed as the first argument (initialState).

What is SetState ?

The setState function is used to update the state. It accepts a new state value and enqueues a re-render of the component.

What Are React-state Hooks ?

The hook in React is a built-in function .that allows a functional component to have state and to update that state when necessary. In React, state is an internal data of a component that can be modified and used to manage the state of the component.

The State hook is a special function that allows you to add state to a functional component. It is a part of the React Hooks API, which was introduced in React 16.8 to allow developers to use state and other features previously only available in class components, in functional components as well.

The syntax of the State hook is as follows:

const [state, setState] = useState(initialState);

Here, **useState** is the State hook function, **state** is the current state value, **setState** is a function used to update the state, and **initialState** is the initial state value. The **useState** function returns an array with two elements - the current state value and a function to update the state.

Parameters

- **initialState**: The value you want the state to be initially. It can be a value of any type, but there is a special behaviour for functions. This argument is ignored after the initial render.

    - If you pass a function as initialState, it will be treated as an initializer function. It should be pure, should take no arguments, and should return a value of any type. React will call your initializer function when initialising the component, and store its return value as the initial state.

Where to Use States 👍

React states can be used in various scenarios where the internal data of a component needs to change over time, in response to user interactions or changes in data. Here are some common use cases for React states:

1. User input: React states can be used to store and update user input, such as form data or search queries.
2. Component behaviour: React states can be used to manage the behaviour of a component, such as toggling between different modes or changing the visibility of elements.
3. API data: React states can be used to store and update data from an API, such as a list of items or a user profile.
4. Authentication: React states can be used to manage the state of a user's authentication status, such as whether they are logged in or logged out.
5. Animation: React states can be used to create animations by updating the position, size, or opacity of elements in response to user interactions or other events.

How to Use States :

To use states in React, you need to follow these steps:

1. Import the **useState** hook from the React library:

import React, { useState } from 'react';

2. Declare the state variable using the **useState** hook, along with its initial value:

const [state, setState] = useState(initialValue);

Here, state **is the current value of the state variable,** setState **is a function to update the state, and** initialValue **is the initial value of the state variable.**

3. Use the state variable in your component:

In this example, the state variable is state, and it is displayed in the component using JSX syntax. The setState function is called when the button is clicked, and it updates the state by incrementing it by one.

You can also use multiple state variables in a single component by calling the useState hook multiple times:

const [name, setName] = useState('John');
const [age, setAge] = useState(30);

In this example, there are two state variables, name and age, and each has its own corresponding setState function.

Adding state to a component

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

function MyComponent() {
  const [age, setAge] = useState(42);
  const [name, setName] = useState('Taylor');
```

`useState` returns an array with exactly two items:

1. The current state of this state variable, initially set to the initial state you provided.
2. The `set` function that lets you change it to any other value in response to interaction.

To update what's on the screen, call the `set` function with some next state:

```
function handleClick() {
  setName('Robin');
}
```

React will store the next state, render your component again with the new values, and update the UI.

Some Examples in the form of Code :

Example 1)

In this example, the `count` state variable holds a number. Clicking the button increments it

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

export default function Counter() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <button onClick={handleClick}>
      You pressed me {count} times
    </button>
  );
}
```

Example 2)

In this example, the `text` state variable holds a string. When you type, `handleChange` reads the latest input value from the browser input DOM element, and calls `setText` to update the state. This allows you to display the current `text` below.

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

export default function MyInput() {
```

```
const [text, setText] = useState('hello');

function handleChange(e) {
  setText(e.target.value);
}

return (
  <>
    <input value={text} onChange={handleChange} />
    <p>You typed: {text}</p>
    <button onClick={() => setText('hello')}>
      Reset
    </button>
  </>
);
}
```

Updating state based on the previous state

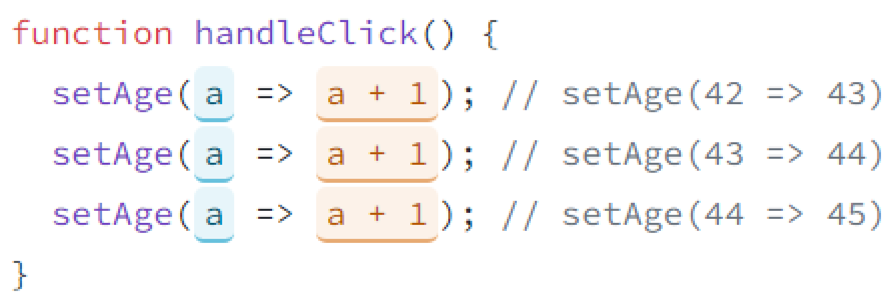
Suppose the `age` is `42`. This handler calls `setAge(age + 1)` three times:

```
function handleClick() {
 setAge(age + 1); // setAge(42 + 1)
 setAge(age + 1); // setAge(42 + 1)
 setAge(age + 1); // setAge(42 + 1)
}
```

Note :👍

However, after one click, `age` will only be `43` rather than `45`! This is because calling the `set` function , the `age` state variable in the already running code. So each `setAge(age + 1)` call becomes `setAge(43)`.

To solve this problem, **you may pass an *updater function*** to `setAge` instead of the next state:



```
function handleClick() {
  setAge(a => a + 1); // setAge(42 => 43)
  setAge(a => a + 1); // setAge(43 => 44)
  setAge(a => a + 1); // setAge(44 => 45)
}
```

Here, `a => a + 1` is your updater function. It takes the pending state and calculates the next state from it.

React puts your updater functions in a [queue.](queue) Then, during the next render, it will call them in the same order:

1. `a => a + 1` will receive `42` as the pending state and return `43` as the next state.
2. `a => a + 1` will receive `43` as the pending state and return `44` as the next state.
3. `a => a + 1` will receive `44` as the pending state and return `45` as the next state.

There are no other queued updates, so React will store `45` as the current state in the end.

By convention, it's common to name the pending state argument for the first letter of the state variable name, like `a` for `age`. However, you may also call it like `prevAge` or something else that you find clearer.

Example of passing an updater function :

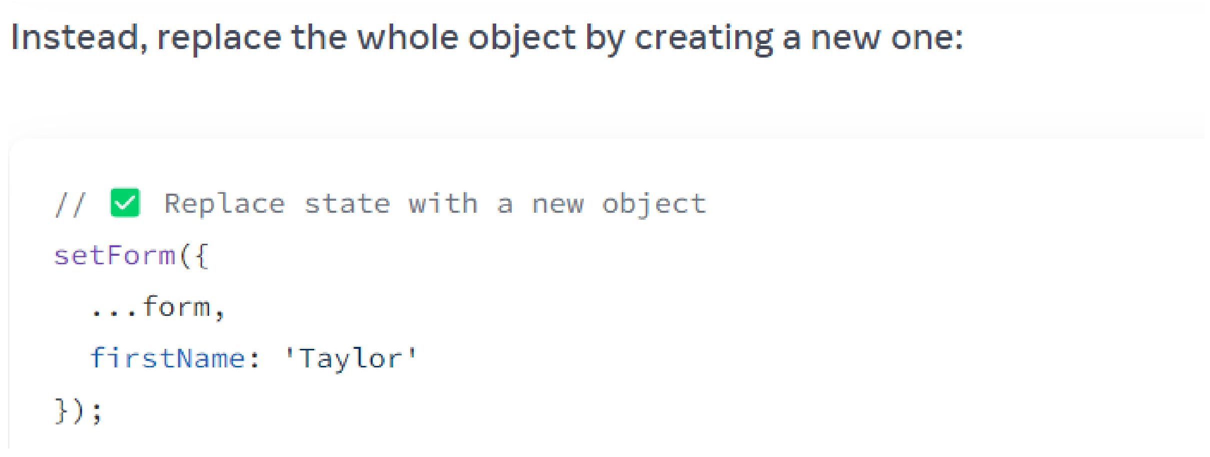
This example passes the updater function, so the “+3” button works.

```
import { useState } from 'react';
export default function Counter() {
  const [age, setAge] = useState(42);
  function increment() {
    setAge(a => a + 3);
  }
  return (
    <>
      <h1>Your age: {age}</h1>
      <button onClick={() => {
        increment();
        increment();
        increment();
      }}>+3</button>
      <button onClick={() => {
        increment();
      }}>+1</button>
    </>
  );
}
```

Updating objects and arrays in state

You can put objects and arrays into state. In React, state is considered read-only, so **you should *replace* it rather than *mutate* your existing objects**. For example, if you have a `form` object in state, don't update it like this:

```
// 🚩 Don't mutate an object in state like this:
form.firstName = 'Taylor';
```



Instead, replace the whole object by creating a new one:

```
// ✅ Replace state with a new object
setForm({
  ...form,
  firstName: 'Taylor'
});
```

Example

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

export default function Form() {
  const [form, setForm] = useState({
    firstName: 'Barbara',
    lastName: 'Hepworth',
    email: 'bhepworth@sculpture.com',
  });

  return (
    <>
      <label>
        First name:
        <input
          value={form.firstName}
          onChange={e => {
            setForm({
              ...form,
              firstName: e.target.value
            });
          }}
        />
      </label>
      <label>
        Last name:
        <input
          value={form.lastName}
          onChange={e => {
            setForm({
              ...form,
              lastName: e.target.value
            });
          }}
        />
      </label>
      <label>
        Email:
        <input
          value={form.email}
          onChange={e => {
            setForm({
              ...form,
              email: e.target.value
            });
          }}
        />
      </label>
      <p>
        {form.firstName}{' '}
        {form.lastName}{' '}
        ({form.email})
      </p>
    </>
  );
}
```

Avoiding recreating the initial state

React saves the initial state once and ignores it on the next renders.

```
function TodoList() {
  const [todos, setTodos] = useState(createInitialTodos());
  // ...
```

Although the result of `createInitialTodos()` is only used for the initial render, you're still calling this function on every render. This can be wasteful if it’s creating large arrays or performing expensive calculations.

To solve this, you may pass it as an *initializer* function to `useState` instead:

```
function TodoList() {
  const [todos, setTodos] = useState(createInitialTodos);
  // ...
```

Notice that you’re passing `createInitialTodos`, which is the *function itself*, and not `createInitialTodos()`, which is the result of calling it. If you pass a function to `useState`, React will only call it during initialization.

SOME INTERVIEW QUESTIONS 👍

1. What is React state and how is it used in a component?

2. What is the difference between state and props in React?

3. How do you initialize state in a React component?

4. How do you update state in a React component?

5. Can you provide an example of how to use setState to update state in a React component?