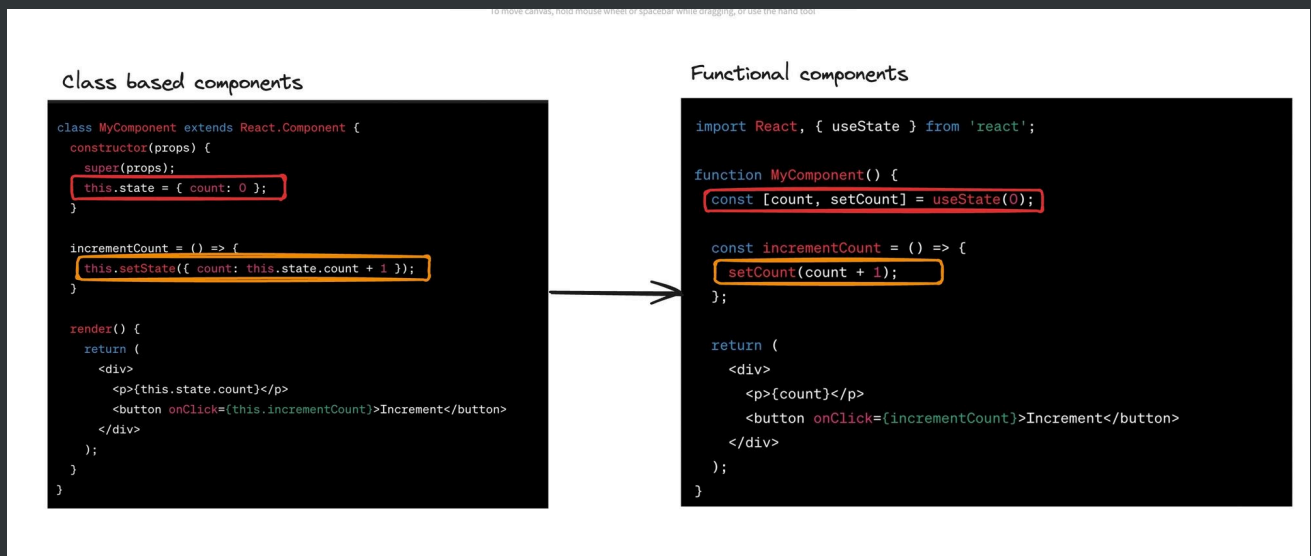


1 - What are hooks

What are hooks

Hooks are a feature introduced in **React 16.8** that allow you to use state and other React features without writing a class. They are functions that let you "hook into" React state and lifecycle features from function components.

State



▼ Functional

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

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

  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>{count}</p>
      <button onClick={incrementCount}>Increment</button>
    </div>
  );
}
```

▼ Class Based

```
class MyComponent extends React.Component {
  constructor(props) {
```

```

    super(props);
    this.state = { count: 0 };
  }

  incrementCount = () => {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    return (
      <div>
        <p>{this.state.count}</p>
        <button onClick={this.incrementCount}>Increment</button>
      </div>
    );
  }
}

```

Lifecycle events

Class based components

```

1
2 class MyComponent extends React.Component {
3   componentDidMount() {
4     // Perform setup or data fetching here
5   }
6
7   componentWillUnmount() {
8     // Clean up (e.g., remove event listeners or cancel subscriptions)
9   }
10
11   render() {
12     // Render UI
13   }
14 }

```

Functional components

```

1
2 import React, { useState, useEffect } from 'react';
3
4 function MyComponent() {
5   useEffect(() => {
6     // Perform setup or data fetching here
7   });
8
9   return () => {
10    // Cleanup code (similar to componentWillUnmount)
11  };
12
13  // Render UI
14 }
15

```

▼ Functional

```

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

function MyComponent() {
  useEffect(() => {
    // Perform setup or data fetching here

    return () => {
      // Cleanup code (similar to componentWillUnmount)
    };
  }, []);

  // Render UI
}

```

▼ Class based

```

class MyComponent extends React.Component {
  componentDidMount() {
    // Perform setup or data fetching here
  }
}

```

```

    }

    componentWillUnmount() {
      // Clean up (e.g., remove event listeners or cancel subscriptions)
    }

    render() {
      // Render UI
    }
  }
}

```

▼ Functional solution

```

import React, { useEffect, useState } from 'react'
import './App.css'

function App() {
  const [render, setRender] = useState(true);

  useEffect(() => {
    setInterval(() => {
      setRender(r => !r);
    }, 5000)
  }, []);

  return (
    <>
      {render ? <MyComponent /> : <div></div>}
    </>
  )
}

function MyComponent() {
  useEffect(() => {
    console.error("component mounted");

    return () => {
      console.log("component unmounted");
    };
  }, []);

  return <div>
    From inside my component
  </div>
}

export default App

```

Until now we've seen some commonly used hooks in React-

1. useState
2. useEffect

3. useMemo

4. useCallback

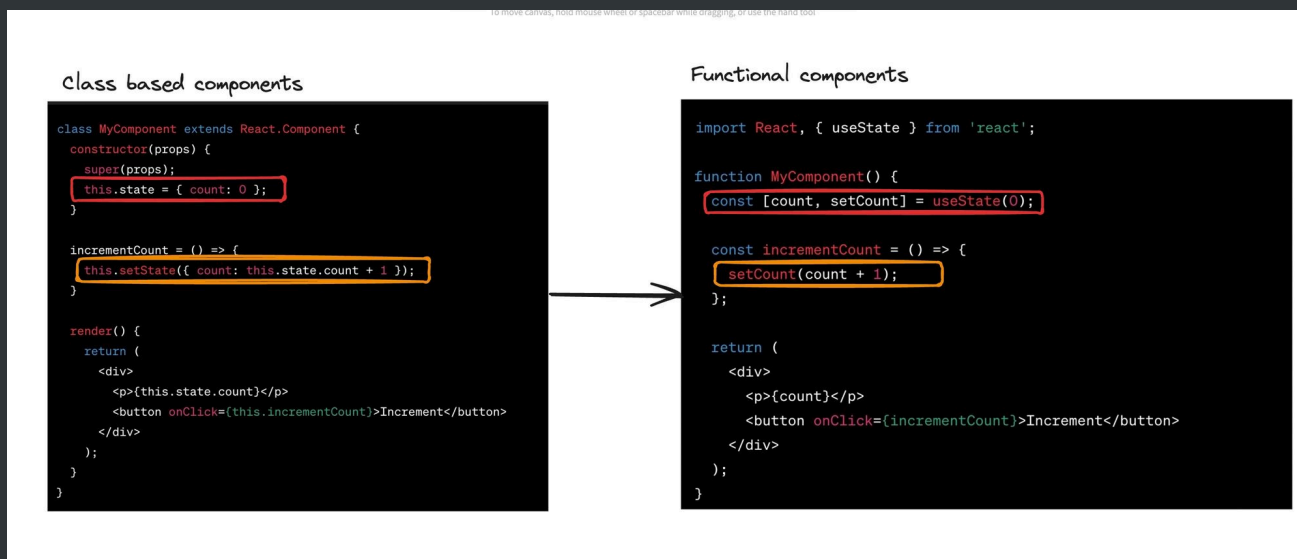
These hooks are provided to you by the **React** library.

1 - What are hooks

What are hooks

Hooks are a feature introduced in **React 16.8** that allow you to use state and other React features without writing a class. They are functions that let you "hook into" React state and lifecycle features from function components.

State



▼ Functional

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

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

  const incrementCount = () => {
    setCount(count + 1);
  };
}
```

```

    return (
      <div>
        <p>{count}</p>
        <button onClick={incrementCount}>Increment</button>
      </div>
    );
  }
}

```

▼ Class Based

```

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  incrementCount = () => {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    return (
      <div>
        <p>{this.state.count}</p>
        <button onClick={this.incrementCount}>Increment</button>
      </div>
    );
  }
}

```

Lifecycle events

Class based components

```

1  class MyComponent extends React.Component {
2
3    componentDidMount() {
4      // Perform setup or data fetching here
5    }
6
7    componentWillUnmount() {
8      // Clean up (e.g., remove event listeners or cancel subscriptions)
9    }
10
11    render() {
12      // Render UI
13    }
14  }

```

Functional components

```

1  import React, { useState, useEffect } from 'react';
2
3
4  function MyComponent() {
5    useEffect(() => {
6      // Perform setup or data fetching here
7    });
8    return () => {
9      // Cleanup code (similar to componentWillUnmount)
10    };
11  }, []);
12
13  // Render UI
14
15

```

▼ Functional

```

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

function MyComponent() {
  useEffect(() => {
    // Perform setup or data fetching here

  }, []);

  return () => {

```

```

        // Cleanup code (similar to componentWillUnmount)
    };
}, []);

// Render UI
}

```

▼ Class based

```

class MyComponent extends React.Component {
  componentDidMount() {
    // Perform setup or data fetching here
  }

  componentWillUnmount() {
    // Clean up (e.g., remove event listeners or cancel subscriptions)
  }

  render() {
    // Render UI
  }
}

```

▼ Functional solution

```

import React, { useEffect, useState } from 'react'
import './App.css'

function App() {
  const [render, setRender] = useState(true);

  useEffect(() => {
    setInterval(() => {
      setRender(r => !r);
    }, 5000)
  }, []);

  return (
    <>
      {render ? <MyComponent /> : <div></div>}
    </>
  )
}

function MyComponent() {
  useEffect(() => {
    console.error("component mounted");

    return () => {
      console.log("component unmounted");
    };
  }, []);
}

```

```
    return <div>
      From inside my component
    </div>
  }

  export default App
```

Until now we've seen some commonly used hooks in React-

1. useState
2. useEffect
3. useMemo
4. useCallback

These hooks are provided to you by the **React** library.