



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

Class based components

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

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

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

Functional components

```
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>
  );
}
```

▼ Functional

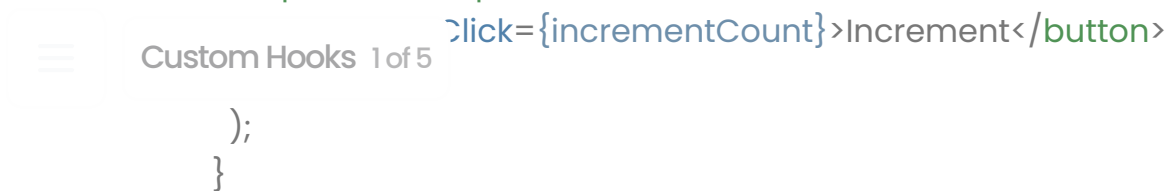
```
import React, { useState } from 'react';
```

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

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



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

```

Functional components

```

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

```

```
import React { useState, useEffect } from 'react';  
  
Custom Hooks 1 of 5  
  
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 subscription)  
  }  
  
  render() {  
    // Render UI  
  }  
}
```

▼ Functional solution

```
import React, { useEffect, useState } from 'react'  
import './App.css'  
  
function App() {  
  const [render, setRender] = useState(true);  
  
  useEffect(() => {  
    setInterval(() => {
```

```
}, []);
```



Custom Hooks 1 of 5

```
<>
  {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're 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.



Custom Hooks 1 of 5

t are custom hooks

Hooks that you create yourself, so other people can use them are called custom hooks.

A **custom hook** is effectively a function, but with the following properties –

1. Uses another hook internally (useState, useEffect, another custom hook)
2. Starts with **use**

A few good examples of this can be

1. Data fetching hooks
2. Browser functionality related hooks – **useOnlineStatus** , **useWindowSize**, **useMousePosition**
3. Performance/Timer based – **useInterval**, **useDebounce**



fetching hooks

Data fetching hooks can be used to encapsulate all the logic to fetch the data from your backend

For example, look at the following code-

```
import { useEffect, useState } from 'react'
import axios from 'axios'

function App() {
  const [todos, setTodos] = useState([])

  useEffect(() => {
    axios.get("https://sum-server.100xdevs.com/todos")
      .then(res => {
        setTodos(res.data.todos);
      })
  }, [])

  return (
    <>
      {todos.map(todo => <Track todo={todo} />)}
    </>
  )
}

function Track({ todo }) {
  return <div>
    {todo.title}
    <br />
    {todo.description}
  </div>
}

export default App
```



Step 1 – Converting the **data fetching** bit to a custom hook

```
import { useEffect, useState } from 'react'
import axios from 'axios'

function useTodos() {
  const [todos, setTodos] = useState([])

  useEffect(() => {
    axios.get("https://sum-server.100xdevs.com/todos")
      .then(res => {
        setTodos(res.data.todos);
      })
  }, [])

  return todos;
}

function App() {
  const todos = useTodos();

  return (
    <>
      {todos.map(todo => <Track todo={todo} />)}
    </>
  )
}

function Track({ todo }) {
  return <div>
    {todo.title}
    <br />
    {todo.description}
  </div>
}
```



Step 2 – Cleaning the hook to include a **loading** parameter

What if you want to show a loader when the data is not yet fetched from the backend?

```
import { useEffect, useState } from 'react'
import axios from 'axios'

function useTodos() {
  const [loading, setLoading] = useState(true);
  const [todos, setTodos] = useState([])

  useEffect(() => {
    axios.get("https://sum-server.100xdevs.com/todos")
      .then(res => {
        setTodos(res.data.todos);
        setLoading(false);
      })
  }, [])

  return {
    todos: todos,
    loading: loading
  };
}

function App() {
  const { todos, loading } = useTodos();

  if (loading) {
    return <div>
      Loading...
    </div>
  }

  return <div>
```




```

    </>
    <div>
      {todos.map(todo => <Track todo={todo} />)}
    </div>
  )
}

function Track({ todo }) {
  return <div>
    {todo.title}
    <br />
    {todo.description}
  </div>
}

export default App

```

Step 3 – Auto refreshing hook

What if you want to keep polling the backend every n seconds?

n needs to be passed in as an input to the hook

```

import { useEffect, useState } from 'react'
import axios from 'axios'

function useTodos(n) {
  const [loading, setLoading] = useState(true);
  const [todos, setTodos] = useState([])

  function getData() {
    axios.get("https://sum-server.100xdevs.com/todos")
      .then(res => {
        setTodos(res.data.todos);
        setLoading(false);
      })
  }

  useEffect(() => {
    setInterval(() => {
      getData();
    }, n);
  }, [n]);
}

```

```
}, [n])
```



Custom Hooks 1 of 5

```
  todos: todos,
  loading: loading
};
}
```

```
function App() {
  const { todos, loading } = useTodos(5);

  if (loading) {
    return <div>
      Loading...
    </div>
  }

  return (
    <>
      {todos.map(todo => <Track todo={todo} />)}
    </>
  )
}
```

```
function Track({ todo }) {
  return <div>
    {todo.title}
    <br />
    {todo.description}
  </div>
}
```

```
export default App
```

▼ Final solution

```
import { useEffect, useState } from 'react'
import axios from 'axios'

function useTodos(n) {
  const [loading, setLoading] = useState(true);

  const [todos, setTodos] = useState([])
```



Custom Hooks 1 of 5

```

    > {
      = setInterval(() => {
        axios.get("https://sum-server.100xdevs.com/todos")
          .then(res => {
            setTodos(res.data.todos);
            setLoading(false);
          })
      }, n * 1000)

    axios.get("https://sum-server.100xdevs.com/todos")
      .then(res => {
        setTodos(res.data.todos);
        setLoading(false);
      })

    return () => {
      clearInterval(value)
    }
  }, [n])

  return {todos, loading};
}

function App() {
  const {todos, loading} = useTodos(10);

  if (loading) {
    return <div> loading... </div>
  }

  return (
    <>
      {todos.map(todo => <Track todo={todo} />)}
    </>
  )
}

function Track({ todo }) {
  return <div>
    {todo.title}
    {todo.description}
  </div>
}

```

</div>



Custom Hooks 1 of 5

export default App

swr – React Hooks for Data Fetching

swr is a popular React library that creates a lot of these hooks for you, and you can use it directly.

For example –

```
import useSWR from 'swr'

// const fetcher = (url) => fetch(url).then((res) => res.json());
const fetcher = async function(url) {
  const data = await fetch(url);
  const json = await data.json();
  return json;
};

function Profile() {
  const { data, error, isLoading } = useSWR('https://sum-server.100xdevs.com/todos');

  if (error) return <div>failed to load</div>
  if (isLoading) return <div>loading...</div>
  return <div>hello, you have {data.todos.length} todos!</div>
}
```

<https://swr.vercel.app/>

4 - Browser functionality related hooks

Custom Hooks 1 of 5

1. **uselsOnline** hook

Create a hook that returns true or false based on whether the user is currently online

You are given that -

1. `window.navigator.onLine` returns **true** or **false** based on whether the user is online
2. You can attach the following event listeners to listen to whether the user is online or not

```

window.addEventListener('online', () => console.log('Became online'))
window.addEventListener('offline', () => console.log('Became offline'))

```

▼ Solution

```

import { useEffect, useState } from 'react'

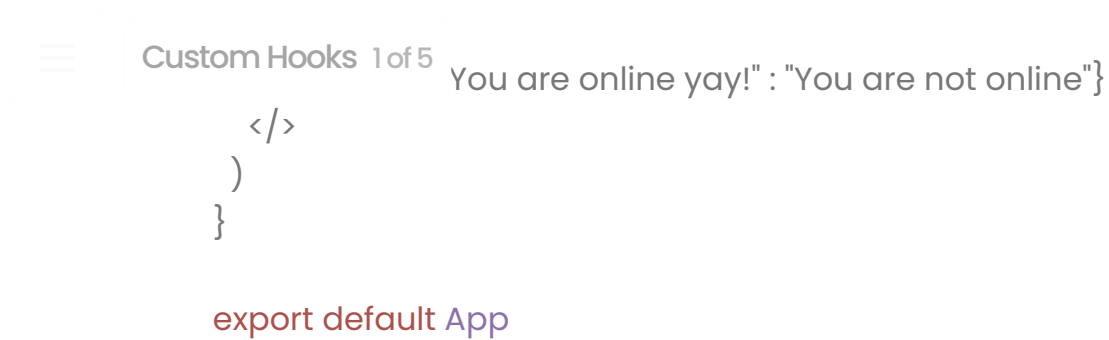
function uselsOnline() {
  const [isOnline, setIsOnline] = useState(window.navigator.onLine)

  useEffect(() => {
    window.addEventListener('online', () => setIsOnline(true));
    window.addEventListener('offline', () => setIsOnline(false));
  }, [])

  return isOnline;
}

export default uselsOnline;

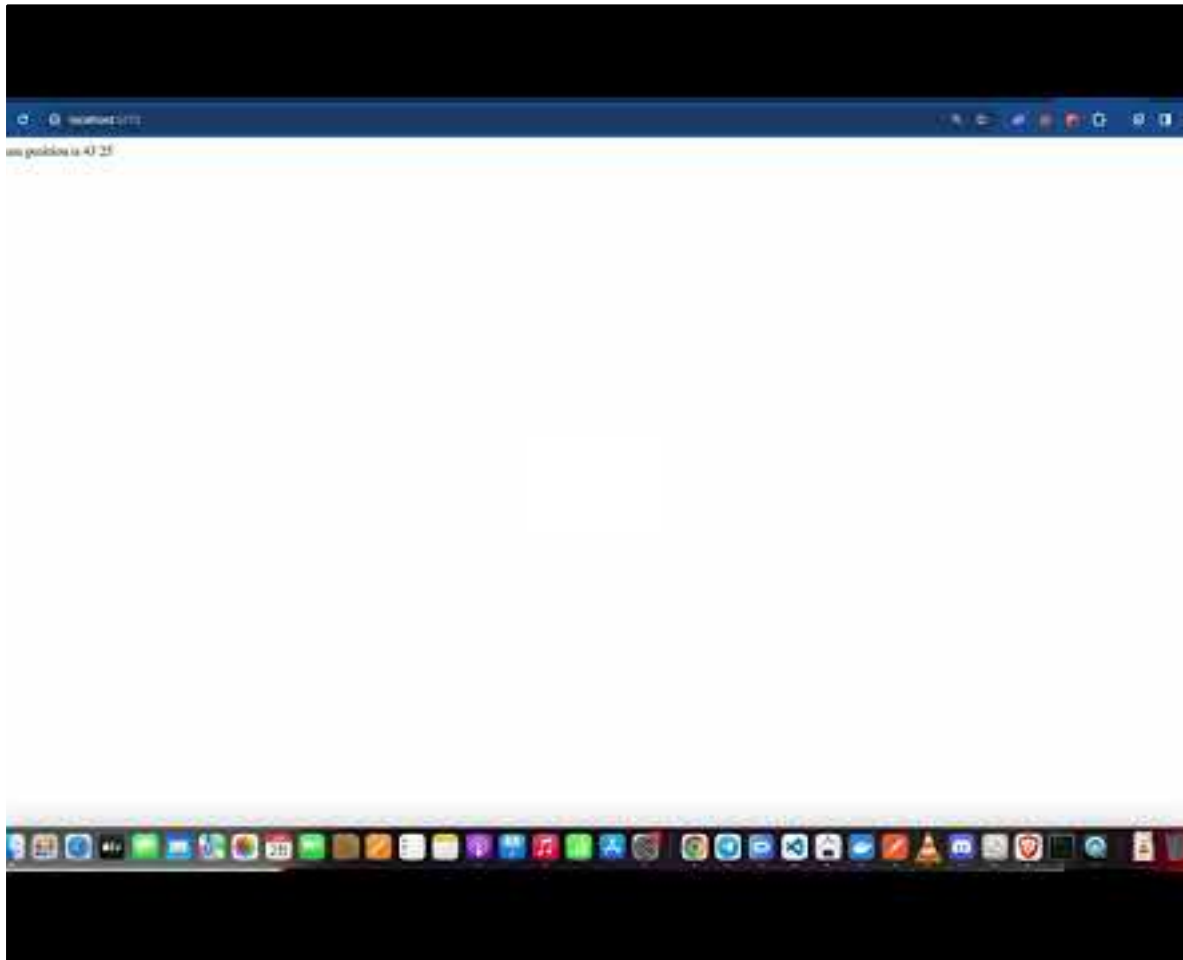
```



2. **useMousePointer** hook

Create a hook that returns you the current mouse pointer position.

The final react app that uses it looks like this



```
window.addEventListener('mousemove', handleMouseMove);
```

Custom Hooks 1 of 5

will trigger the `handleMouseMove` function anytime the mouse pointer is moved.

▼ Solution

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

const useMousePointer = () => {
  const [position, setPosition] = useState({ x: 0, y: 0 });

  const handleMouseMove = (e) => {
    setPosition({ x: e.clientX, y: e.clientY });
  };

  useEffect(() => {
    window.addEventListener('mousemove', handleMouseMove);
    return () => {
      window.removeEventListener('mousemove', handleMouseMove);
    };
  }, []);

  return position;
};

function App() {
  const mousePointer = useMousePointer();

  return (
    <>
    Your mouse position is {mousePointer.x} {mousePointer.y}
    </>
  )
}

export default App
```



5 – Performance/Timer based

1. `useInterval`

Create a hook that runs a certain callback function every n seconds.

You have to implement `useInterval` which is being used in the code below –

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

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

  useInterval(() => {
    setCount(c => c + 1);
  }, 1000)

  return (
    <>
      Timer is at {count}
    </>
  )
}
```



Final app should look like this

Custom Hooks 1 of 5

▼ Solution

```
const useInterval = (callback, delay) => {  
  useEffect(() => {  
    const intervalId = setInterval(callback, delay);  
  
    return () => clearInterval(intervalId);  
  }, [callback, delay]);  
};
```

2. useDebounce

Create a hook that debounces a value given

1. The value that needs to be debounced
2. The interval at which the value should be debounced.

```
import React, { useState } from 'react';  
import useDebounce from './useDebounce';  
  
const SearchBar = () => {  
  const [inputValue, setInputValue] = useState("");  
  const debouncedValue = useDebounce(inputValue, 500); // 500
```

ur component logic, e.g., trigger

```
return (
```



Custom Hooks 1 of 5

```
    value={inputValue}
    onChange={(e) => setInputValue(e.target.value)}
    placeholder="Search..."
  />
);
};
```

```
export default SearchBar;
```

▼ Solution

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

const useDebounce = (value, delay) => {
  // State to store the debounced value
  const [debouncedValue, setDebouncedValue] = useState(val

  useEffect(() => {
    // Set up a timer to update the debounced value after the sp
    const timerId = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);

    // Clean up the timer if the value changes before the delay h
    return () => clearTimeout(timerId);
  }, [value, delay]);

  return debouncedValue;
};
```

