What Are Hooks?

React Hook is a feature that allows to use state and other React features without writing a class component.

Why Use Hooks?

Hooks improve code organization as it's easier to write separate components and still able to fetch data.

Reusability, as you can create custom hooks to encapsulate common logic and reuse it across multiple components to avoid redundancy.

Hooks can make your code easier to understand by making the flow of data more explicit.

Example of Default Hooks in React

useState: This hook allows you to add state to your functional component.

useEffect: This hook allows you to perform side effects in your functional components, such as fetching data or setting up subscriptions.

useContext: This hook allows you to access the value of a context provided by a Context.Provider.

useReducer: This hook is a more powerful alternative to useState for managing complex state.

useRef: This hook provides a way to access DOM elements or create mutable references in your functional components.

How it's used :

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

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

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
```

```
  );
}
```

Thus no need to make classes and you can use functions.

It's possible to create custom hooks too for whatever logic you need.

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

function useFetch(url) {
const [data, setData] = useState(null);
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);

useEffect(() => {
const fetchData = async () => {
setLoading(true);

try {
const response = await fetch(url);
const json = await response.json();
setData(json);
setError(null);
} catch (error) {
setError(error);
} finally {
setLoading(false);
}
};


fetchData();
```