# REACT HOOKS

## React Hooks :overview

## Introduction to React Hooks

The React `16.8` update was a major revolution in the way that React treats and interacts with components and this all thanks to hooks.

Hooks is making big waves in the React community because it reduces the complexity of state management.
In this skill, we are going to see:

- What are React Hooks?
- What is useState hook?
- What is useEffect hook?
- What is useRef hook?
- How to make your own custom hooks?

## useState

While describing the use of state in the previous Super Skill, we have seen that the state can only be updated using the `this.setState` method.
The state is characterized by two major things: a **value** (aka `getter`) and a method to set this value (aka `setter`).

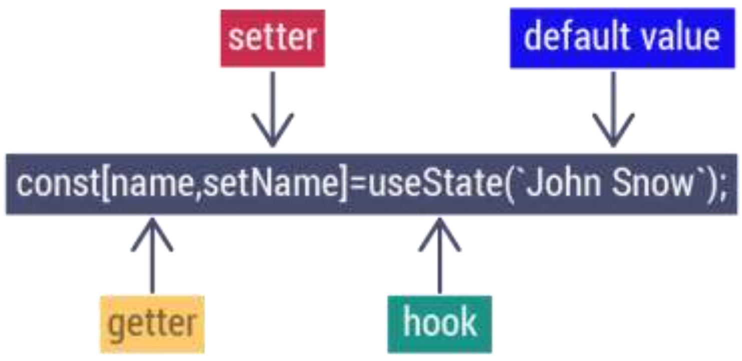
The same logic still stands with the `useState()` method.

## Declaring a state variable:

First of all, we need to learn how to declare a state variable in our component:

1. Import the hook useState.
2. Declare your state variable in an array (the getter and the setter)
3. Initialize your state variable with useState( ).
   This example will help you understand more:

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



## Accessing The State

Now that we have already declared our component state, we need to be able to access the value of that state.
Like any other variable in JavaScript, if we want the value of a state, we invoke the state’s name like it's shown in the example below:

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



const MyFunctionComponent = props => {

 // setting the state hooks

 const [name, setName] = useState("Arya Stark");

 return (

   <div>

     {/* here we use the getter to get the state value */}

     <p>hello my name is {name}</p>

   </div>

 );

};
```

## Set the state's value:

To change the state's value, we have to use the setter that we implemented in the state declaration. In this example, the setName method acts as the setter (or the modifier):

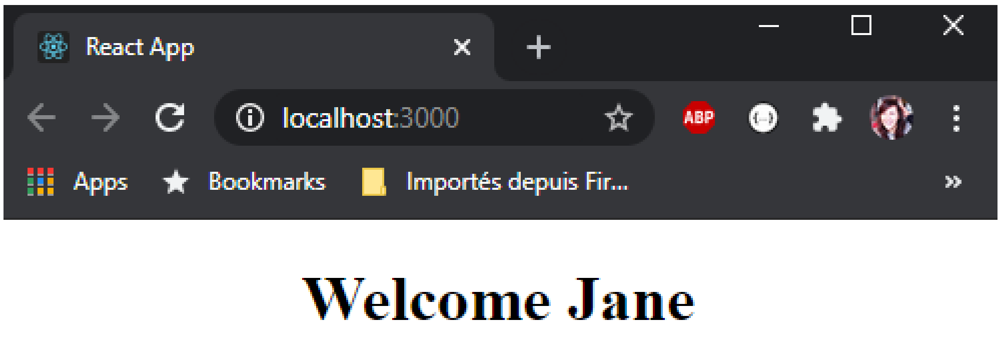
```
const MyFunctionComponent = props => {

 // declaring the state hooks

 const [name, setName] = useState("Arya Stark");

 // here we use the setter to change the content of the name state

 const handleClick = () => setName('Tyron Lanyster')

 return (

   <div>

     {/* here we use the getter to get the state value */}

     <p>hello my name is {name}</p>

     <button onClick={handleClick}>Click if you want to give the crown to Tyron Lanyster</button>

   </div>

 );

};
```

## Keypoints to keep in mind:

Now that we know what the useState() Hook does and once we our component initial state, we need to be able to access the value of that state.
Like any other variable in JavaScript, if we want the value of a state, all we have to do is invoke the state’s name, like shown in the example below:

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


function Welcome() {

const [name, setName] = useState(“Jane”)

    return (

<div>

<h1> Welcome {name} </h1>

</div>

  )

}


export default Welcome;
```

Output:



Welcome Jane

In this case, counter is :

```
const [counter,setCounter] = useState(0);
```

getter

setter

What is the initial value of the state?

```
const [counter,setCounter] = useState(5)
```

2

5

1

## What is the initial value of the state?

```
const [counter,setCounter] = useState(5)
```

2

5

1

## the setter is:

A function that changes the value of the state variable.

A function that calls the state variable inside the component.

A function that sets the state's value by default.

# useEffect

The `useEffect()` method takes:
A function as the first parameter that describes the effect we want to apply. Then, it takes an array as a second parameter that will determine when this effect should appear or disappear.

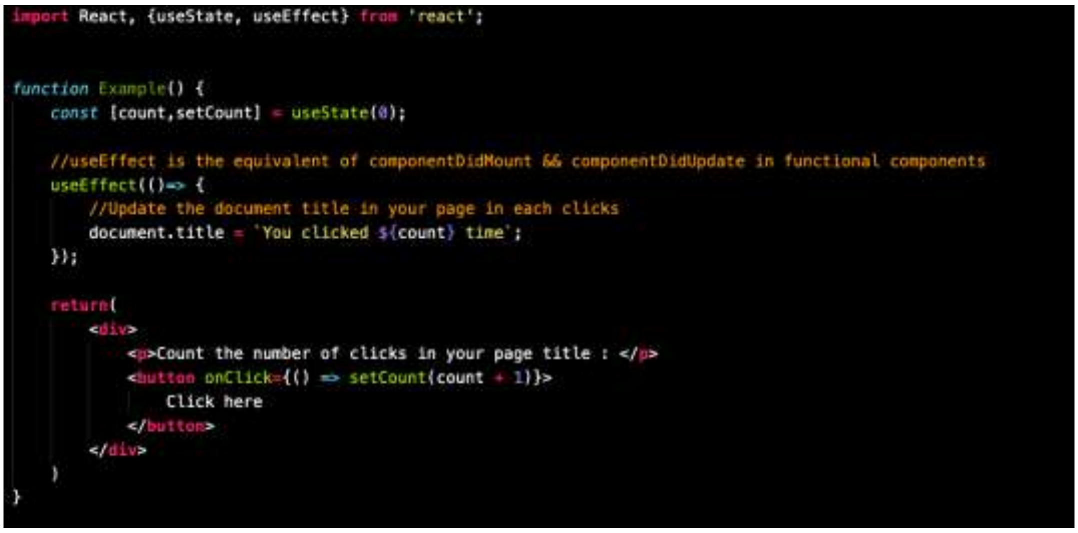
So keep in mind that to start up, we need to use an empty array `[ ]` as a second parameter. However, this parameter remains optional.
`[]` means the effect doesn’t use any value that participates in React's data flow and that's the reason why it is considered safe to apply it once.
It is also considered a common source of bugs when the value is actually being used.

This code will better illustrate what we have mentioned:

```
// setting the useEffect to trigger while the component is been rendering

useEffect(()=>{

 alert(`hello this a message`)

},[])
```

# useEffect

The `useEffect()` method allows component functions to manage side effects.
It plays the same role as `componentDidMount()`, `componentDidUpdate()`, and `componentWillUnmount()` in React classes only it is implemented through a single API.



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


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

    //useEffect is the equivalent of componentDidMount && componentDidUpdate in functional components
    useEffect(()=> {
        //Update the document title in your page in each clicks
        document.title = `You clicked ${count} time`;
    });

    return(
        <div>
            <p>Count the number of clicks in your page title : </p>
            <button onClick={() => setCount(count + 1)}>
                Click here
            </button>
        </div>
    )
}
```

[https://www.youtube.com/watch?v=wbzMK4tNBao&list=PL-w_yrNy8uTaWNAYCFoyW0C0zPm4S9b3v](https://www.youtube.com/watch?v=wbzMK4tNBao&list=PL-w_yrNy8uTaWNAYCFoyW0C0zPm4S9b3v)

## When does the component execute the useEffect method?

With every state modification.

Only when it's first rendered.

When this component is destroyed.

## Will it work if we remove the dependency array [ ] argument from the useEffect function?

True

False

## In order to use a spinner, what should we add as a secondary parameter inside the useEffect method?

[ ]

{ }

()

## Common mistakes

## useRef()

The `useRef()` Hook is a function that returns a mutable reference object whose current property is initialized to the passed argument (initialValue). The returned object will persist for the component's entire lifetime.

This example may help you:

```
function App() {

 let [name, setName] = useState("Ned stark");

 // we declare the input inside the variable

 let nameRef = useRef();

 // we are referring to input to change the value

 const submitButton = () => {

   setName(nameRef.current.value);

 };


 return (

   <div className="App">

     <p>{name}</p>

     <h1>Who is your favorite Games of throne character</h1>


     <div>

       <input

         placehoder="enter your preferred GOT character..."

         ref={nameRef}

         type="text"

       />

       <button type="button" onClick={submitButton}>

         Submit

       </button>

     </div>

   </div>

 );

}
```

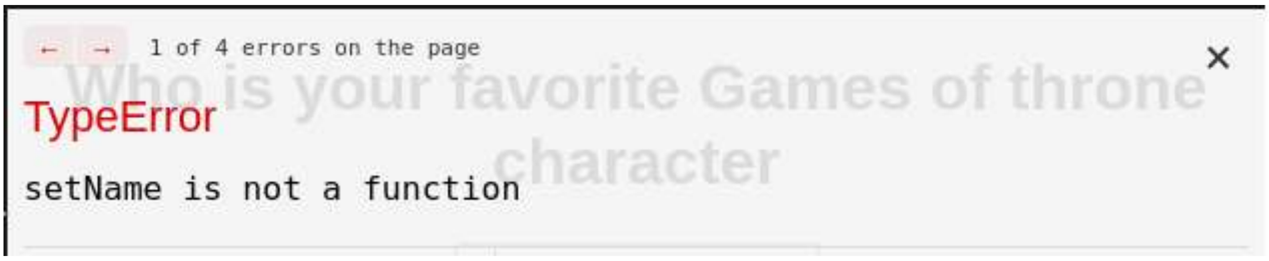
In this function we are recovering the input value using the useRef method instead of onChange event. That may be a potential bug event if works for now. A ref created with **useRef** will be created only when the component has been mounted. Refs can be used for accessing DOM nodes or React elements, and for keeping mutable variables.

## Destructuring using object:

Don't forget: hooks use arrays to store data.
As we have previously mentioned, Hooks are applied in an array:

```
const { state, setState} = useState("intial state")
```

If we perform destructuring using the curly brackets, we'll receive this error:



```
1 of 4 errors on the page
TypeError
setName is not a function
```

Reminder that this code is incorrect. We need to use brackets and not curly braces.

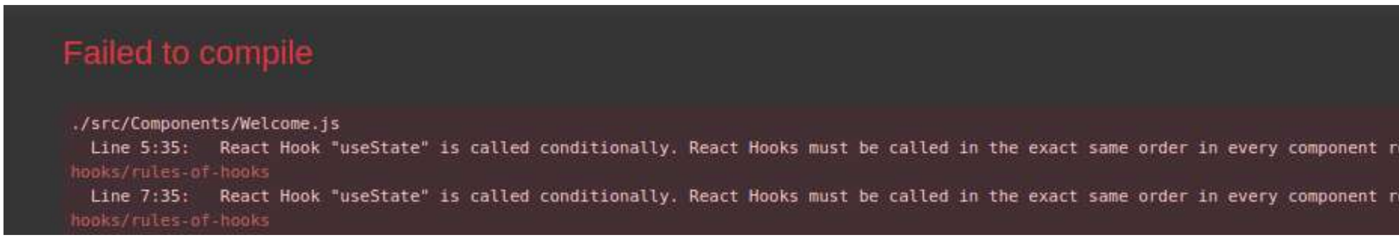
```
const [state, setState] = useState("intialState)
```

## Hooks inside a condition:

We should avoid creating the state hook in a condition because that will violate the hook rules. Once you break these rules, this error will occur:

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


const Welcome = props =>{

 if(props.name==='ned stark'){

   const [bgColor,setBgColor ]= useState('white')

 }

 else{

   const [bgColor, setBgColor] = useState('black')

 }


 return (

   <h1 style={{backgroundColor:{bgColor}}} >{props.name}</h1>

 )

}


export default Welcome
```

Output



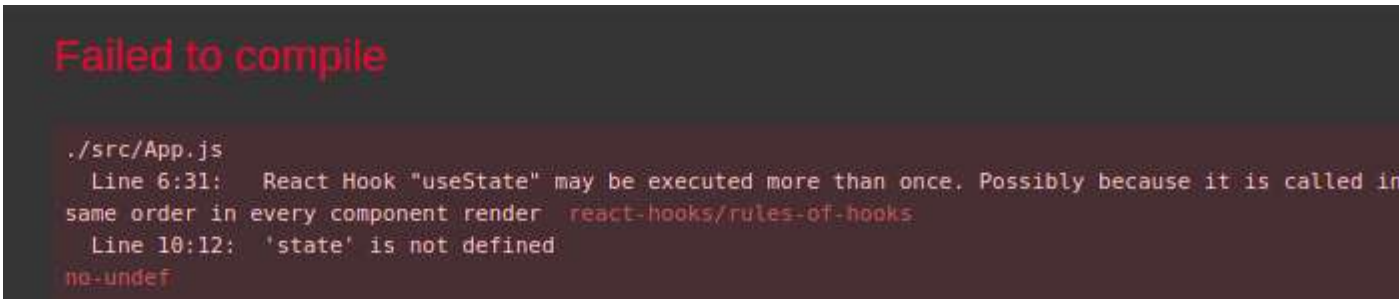
```
Failed to compile

./src/Components/Welcome.js
  Line 5:35:   React Hook "useState" is called conditionally. React Hooks must be called in the exact same order in every component r
hooks/rules-of-hooks
  Line 7:35:   React Hook "useState" is called conditionally. React Hooks must be called in the exact same order in every component r
hooks/rules-of-hooks
```

## Hooks inside a loop:

Another common mistake is to use the hook inside loops. Here is an example to illustrate the error.

```
function App() {

 for (let i = 1; i < 5; i++) {

   const [state, setstate] = useState(i);

 }

 return (

   <div>

     <h1>{state}</h1>

   </div>

 );

}



export default App;
```

Output



```
Failed to compile

./src/App.js
  Line 6:31:   React Hook "useState" may be executed more than once. Possibly because it is called in
same order in every component render  react-hooks/rules-of-hooks
  Line 10:12:  'state' is not defined
no-undef
```

## Nesting before using the state:

We can't nest inside a function before using our state.

```
function App({ date }) {

 function updateCount(byValue) {

   const [currentDate, setCurrentDate] = useState(new Date());

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

   setCount(count + byValue);
   setCurrentDate(new Date());

 }


 function formatDate() {

   const hour = currentDate.getHours();

   const minute = currentDate.getMinutes();

   const second = currentDate.getSeconds();


   return `${hour}:${minute}:${second}`;

 }


 const prettyDate = formatDate();


 return (

   <div className="App">

     <h2>

       You clicked {count} times, last time at {prettyDate}!

     </h2>


     <button onClick={() => updateCount(-1)}>Decrement</button>

     <button onClick={() => updateCount(1)}>Increment</button>

   </div>

 );

}
```

## useState inside an effect:

Now, we will go through some points where we invoke the useEffect.
**Important notice:** The combined use of both the useState and useEffect generates an infinite loop.
So, don't call a useState inside a useEffect.

## Let's Sum Up!

To summarize, we can only call Hooks at the top level.
We should be aware of these rules:

- Don't declare hooks in if statements.
- Don't declare hooks in loops.
- Don't declare hooks in nested function.
- Always make sure to use the brackets instead of the curly brackets.

### We can't apply useEffect

inside a useState.

in a (if/else statement) condition.

in a sub-funciton.

All answers are correct.

### Is the following code correct?

```
const {name, setName}= useState()
```

True

False

### We can invoke a useState() in a useEffect() hook.

True

False

## Creacting custom Hooks

## Custom Hooks.

A custom Hook is a JavaScript function whose name begins with "use". It provides flexible logic sharing.
We can call other Hooks in our custom Hooks.
Before we start, We need to be able to build our own Hooks in a personalized way and it will still ensure the functional reusability.
The following example of users and posts will help us better understand custom Hooks' practicality.

## Custom hooks

Custom hooks are simply a wrapper function surrounding our existing hooks.
That's all there is to it! The only catch is: in order for React to recognize a function as a custom hook, its name should always start with use so that you can tell at a glance that the rules of Hooks apply to it.

## Create a custom Hook

We will create a **useInterval** Hook and then we will use it.

```
function useInterval (timeout, getValue) {

   const [value, setValue] = useState(getValue);

    useEffect(() => {

     const intervalID = setInterval(

       () => setValue(getValue()),

       timeout

     );

    return function () {

     clearInterval(intervalId);

   }

 }, []);

  return value;

 }

  const get CurrentDate = () => new Date();

  function App() {

   const date = useInterval(1000, getCurrentDate);

    return <p>Nous sommes le {date.toLocaleString("fr-FR")}</p>;

 }
```

# Recap

In this video you will find an overview about React Hooks

[https://www.youtube.com/watch?v=cxQZaYkBLRg&list=PL-w_yrNy8uTaWNAYCFoyW0C0zPm4S9b3v](https://www.youtube.com/watch?v=cxQZaYkBLRg&list=PL-w_yrNy8uTaWNAYCFoyW0C0zPm4S9b3v)

A custom Hook is a JavaScript function.

True

False

We can invoke a custom Hook in a conditional statement.

True

False

A custom Hook is a function that provides flexible code.

True

False

## Conclusion:

### What you should know by now :

## Tips:

- Avoid declaring `useState`, `useEffect` or custom Hooks:
    - In a loop.
    - In a condition.
    - In a nested function.
- Don't forget array destructuring.
- You should use custom Hooks with components that have the same functionality.
- Avoid applying useRef to DOM elementS.

## Let’s recap:

Hooks is a new feature that has been introduced to React in order to make dynamic data manipulation easier.
useState is essential for selecting a default state value then updating it.

Time to put into practice what we have been learning!

---

### We prepared a cheatsheet with everything you learned in this chapter:

## ☐ GMC Hooks Cheat Sheet.

**Objective**

In this checkpoint, we are going to create a movie app where you can present your favorite movies or TV shows. We will also be able to use the react hooks.

**Instructions**

- Create the following components:
    - MovieCard
    - MovieList
    - Filter ( title, rate)
- Every movie should have the following attributes: title, description, posterURL, rating
- The user should be:
    - Able to add a new movie.
    - Filter the movies with title/rating.