Hooks

Hooks are functions that let you “hook into” React state and lifecycle features from function components.

Hooks allow you to use React without classes. It means you can use state and other React features without writing a class.

React provides a few built-in Hooks like useState, useEffect etc

Hooks are a new addition in React 16.8.

When use Hooks

If you write a function component and realize you need to add some state to it.

Rules of Hooks

- Only call Hooks at the top level – We should not call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function.
- Only call Hooks from React functions – We should not call Hooks from regular JavaScript functions. Instead, call Hooks from React function components or call Hooks from custom Hooks
- React relies on the order in which Hooks are called.
- Hooks don’t work inside classes.

Declaring State

useState ( ) - useState is a Hook that allows you add React state to function components. We call it inside a function component to add some local state to it.

useState returns a pair - the current state value and a function that lets you update it.

React will preserve this state between re-renders.

You can call this function from an event handler or somewhere else.

Ex:-

import React, { useState } from 'react';

useState(“Rahul”);

const nameStateVariable = useState(“Rahul”);

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

importing useState Hook from React

Declaring State Variable

Declaring State

const [name, setName] = useState("Rahul"); ← Declaring State Variable

When we declare a state variable with useState, it returns a pair - an array with two items. So, by writing square bracket we are doing Array Destructuring.

const nameStateVariable = useState("Rahul");

- The first item is the current value.
- The second is a function that lets us update it.

```
const name = nameStateVariable[0];          // First item of Array
const setName = nameStateVariable[1];       // Second item of Array
```

Note - you can call useState as many times as you want.

```
function App () {
const nameStateVariable = useState(“Rahul”);
const [name, setName] = useState(“Rahul”);
const [roll, setRoll] = useState(101);
const [subject, setSubject] = useState( [  {sub: “Math”}  ] );
}
```

Accessing State

In a function, we can use state variable directly.

Ex: -

<h1> Your Name is {name} </h1>

Updating State

Ex:-

setName(“GeekyShows”)

Effect Hooks

The Effect Hook lets you perform side effects in function components. Data fetching, setting up a subscription, and manually changing the DOM in React components are all examples of side effects.

useEffect ( )

useEffect is a hook for encapsulating code that has 'side effects,' if you're familiar with React class lifecycle methods, you can think of useEffect Hook as componentDidMount, componentDidUpdate, and componentWillUnmount combined.

Ex:-

```
import React, { useState, useEffect } from 'react';
useEffect(Function)
useEffect(Function, Array)
```

importing useEffect Hook from React

- The function passed to useEffect will run after the render is committed to the screen.
- Second argument to useEffect that is the array of values that the effect depends on.

Note - you can call useEffect as many times as you want.

useEffect ( )

```
useEffect(() => {
    console.log(“Hello useEffect”);
});

useEffect(() => {
    console.log(“Hello useEffect”);
}, [count]);
```

What does useEffect do ?

By using this Hook, you tell React that your component needs to do something after render. React will remember the function you passed and call it later after performing the DOM updates. In this effect, we set the document title, we could also perform data fetching or call some other imperative API.

Why is useEffect called inside a Component

Placing useEffect inside the component lets us access the state variable or any props right from the effect.

Does useEffect run after every Render

Yes! By default, it runs both after the first render and after every update.

Custom Hook

A custom Hook is a JavaScript function, when we want to share logic between two JavaScript functions, we extract it to a third function.

Building your own Hooks lets you extract component logic into reusable functions.

You can write custom Hooks that cover a wide range of use cases like form handling, animation, declarative subscriptions, timers, and many more.

Creating Custom Hook

A custom Hook is a JavaScript function whose name starts with ”use” and that may call other Hooks.

Ex:-

```
function useSomething( ) {
     return
};
```

Using Custom Hook

A custom Hook is a JavaScript function whose name starts with ”use” and that may call other Hooks.

Ex:-

const data = useSomething( );