1. React Hooks

Hooks were added to React in version 16.8.

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

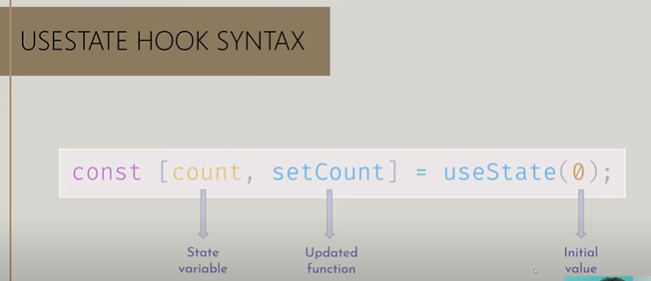
What is a Hook?

Hooks allow us to "hook" into React features such as state and lifecycle methods.

# **React useState Hook**

The React useState Hook allows us to track state in a function component.

State generally refers to data or properties that need to be tracking in an application.



**Method #1 - Loop using the map function**

The first method to loop through an array of objects involves using the JavaScript *map* function with a callback that returns the React component.

Each component you generate from the array of objects must have a unique key. This key is mandatory when you are working with lists in React. It helps spot changes inside the list.

*Here is how to use the map function to loop through an array in React:*

javascriptimport React from "react";

import ReactDOM from "react-dom";

const animals = [

{

type: "cat",

age: 3

},

{

type: "dog",

age: 2

}

];

const App = () => (

<>

{animals.map((animal, index) => (

<div key={index}>

<div>Type: {animal.type}</div>

<div>Age: {animal.age}</div>

</div>

))}

</>

);

ReactDOM.render(<App />, document.getElementById("container"))

# **React useEffect Hooks**

The useEffect Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

useEffect accepts two arguments. The second argument is optional.

useEffect(<function>, <dependency>)

Use setTimeout() to count 1 second after initial render:

import { useState, useEffect } from "react";

import ReactDOM from "react-dom/client";

function Timer() {

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

useEffect(() => {

setTimeout(() => {

setCount((count) => count + 1);

}, 1000);

});

return <h1>I've rendered {count} times!</h1>;

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Timer />);

But wait!! It keeps counting even though it should only count once!

useEffect runs on every render. That means that when the count changes, a render happens, which then triggers another effect.

This is not what we want. There are several ways to control when side effects run.

We should always include the second parameter which accepts an array. We can optionally pass dependencies to useEffect in this array.