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.

Import useState

To use the useState Hook, we first need to import it into our component.

Example:

At the top of your component, import the useState Hook.

import { useState } from "react";

Notice that we are destructuring useState from react as it is a named export.

Initialize useState

We initialize our state by calling useState in our function component.

useState accepts an initial state and returns two values:

* The current state.
* A function that updates the state.

Example:

Initialize state at the top of the function component.

import { useState } from "react";

function FavoriteColor() {

const [color, setColor] = useState("Red");

}

Notice that again, we are destructuring the returned values from useState.

The first value, color, is our current state.

The second value, setColor, is the function that is used to update our state.

These names are variables that can be named anything you would like.

Lastly, we set the initial state to an empty string: useState("")

Read State

We can now include our state anywhere in our component.

Example:

Use the state variable in the rendered component.

import { useState } from "react";

import ReactDOM from "react-dom/client";

function FavoriteColor() {

const [color, setColor] = useState("red");

return <h1>My favorite color is {color}!</h1>

}

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

root.render(<FavoriteColor />);

Update State

To update our state, we use our state updater function.

We should never directly update state. Ex: color = "red" is not allowed.

Example:

Use a button to update the state:

import { useState } from "react";

import ReactDOM from "react-dom/client";

function FavoriteColor() {

const [color, setColor] = useState("red");

return (

<>

<h1>My favorite color is {color}!</h1>

<button

type="button"

onClick={() => setColor("blue")}

>Blue</button>

</>

)

}

What Can State Hold

The useState Hook can be used to keep track of strings, numbers, booleans, arrays, objects, and any combination of these!

We could create multiple state Hooks to track individual values.

Example:

Create multiple state Hooks:

import { useState } from "react";

import ReactDOM from "react-dom/client";

function Car() {

const [brand, setBrand] = useState("Ford");

const [model, setModel] = useState("Mustang");

const [year, setYear] = useState("1964");

const [color, setColor] = useState("red");

return (

<>

<h1>My {brand}</h1>

<p>

It is a {color} {model} from {year}.

</p>

</>

)

}

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

root.render(<Car />);

Or, we can just use one state and include an object instead!

Example:

Create a single Hook that holds an object:

import { useState } from "react";

import ReactDOM from "react-dom/client";

function Car() {

const [car, setCar] = useState({

brand: "Ford",

model: "Mustang",

year: "1964",

color: "red"

});

return (

<>

<h1>My {car.brand}</h1>

<p>

It is a {car.color} {car.model} from {car.year}.

</p>

</>

)

}

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

root.render(<Car />);

Since we are now tracking a single object, we need to reference that object and then the property of that object when rendering the component. (Ex: car.brand)

Updating Objects and Arrays in State

When state is updated, the entire state gets overwritten.

What if we only want to update the color of our car?

If we only called setCar({color: "blue"}), this would remove the brand, model, and year from our state.

We can use the JavaScript spread operator to help us.

Example:

Use the JavaScript spread operator to update only the color of the car:

import { useState } from "react";

import ReactDOM from "react-dom/client";

function Car() {

const [car, setCar] = useState({

brand: "Ford",

model: "Mustang",

year: "1964",

color: "red"

});

const updateColor = () => {

setCar(previousState => {

return { ...previousState, color: "blue" }

});

}

return (

<>

<h1>My {car.brand}</h1>

<p>

It is a {car.color} {car.model} from {car.year}.

</p>

<button

type="button"

onClick={updateColor}

>Blue</button>

</>

)

}

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

root.render(<Car />);

Because we need the current value of state, we pass a function into our setCar function. This function receives the previous value.

We then return an object, spreading the previousState and overwriting only the color.