Multiple States

1. We want to gather all the values for all inputs and then combine them into an object when the form overall is submitted.
2. To make sure that we store the value and that it survives if that function would be re-executed and the component would be re-evaluated, we will use **useState**.
3. we can import the useState hook from React.

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

4. then use useState , initially that's an empty string because initially when this component is rendered for the first time, nothing was entered. so **useState("")**.
5. after that use destructuring to get our two elements, which is the currently enteredTitle(entitledTitle). Maybe we a function(setChangedTitle) for updating the State. **const [entitledTitle, setChangedTitle]**

```
const ExpenseForm = (props) => {
  const [entitledTitle, setChangedTitle] = useState("");
```

6. then call the setChangedTitle function inside handler to sotre the value.

```
const titleChangeHandler = (event) => {
    setChangedTitle(event.target.value);
}
```

> Note: We can have multiple States, multiple States slices or State pieces per component. And all of these States inside of one at the same component will then all to be totally separated from each other.

Other ways of handing data - Updating State that Depends on the Previous State

1. Using One state Instead.

- by calling used state only once and by passing in an object as a value.

```
const [userInput,setUserInput] = useState({
      entitledTitle:'',
      entitledAmount:'',
      entitledDate:''
  });
```

- and then using below method we can set indiviual state.

```
setUserInput({
        ...userInput,
        entitledTitle: event.target.value
    });
```

- but above method is not feasible to set bcz Reacts schedules state updates. So, if you schedule a lot of state updates at the same time, you could be depending on an outdated or incorrect state snapshot if you use this approach.

> Note : If your state update depends on the previous state then use the below return syntax.

```
setUserInput((prevState) =>{
        return {...prevState, entitledTitle: event.target.value};
    });
```

- If you use this approach, React will guarantee that the state snapshot it gives you here in this inner function, will always be the latest state snapshot, keeping all scheduled state updates in mind. So this is the safer way to ensure that you always operate on the latest state snapshot. So you should use this function syntax here whenever your state update depends on the previous state.