Web Programming 04/11

* React.js (Part II)
* In-class practice

Refresh…

- Use pure function or class to define React element prototype (i.e. component)…

```
> function Welcome(props) {
    return <h1>Hello, {props.name}</h1>; }  // Why no "this"?
> import React, { Component } from 'react';
  class Welcome extends Component {
    render() {
      return <h1>Hello, {this.props.name}</h1>; }
}
```

1. Remember, JS uses prototype-based object construction.
   ==> You don't need to "define" properties of a class before using them.
   ==> When they are used, they are defined.
2. All React components must act like pure functions with respect to their props
   ==> Can't assign to props.

- 依 component 產生 element
  - ```
    const element = <Welcome name="Ric" />;
    ReactDOM.render(element,
      document.getElementById('root'));
    ```
    ```

HTML vs. React / JSX

```
<head></head>
<body>
  <div id='root'>
    <table border='2px'>
      <caption> Ric's score </caption>
      <thead>
        <tr><th>Subject</th><th>Score</th></tr>
      </thead>
      <tbody>
        <tr><th>Math</th> <th>100</th> </tr>
        <tr><th>Chinese</th> <th> 87</th></tr>
      </tbody>
    </table>
  </div>
</body>
```

Ric´s score

Subject	Score
Math	100
Chinese	87

HTML vs. React / JSX

```
class Subject extends Component {…}
class SubjectList extends Component {…}
class Caption extends Component {…}
class ScoreCard extends Component {
  render() {
    return (
      <table border='2px'>
        <Caption name={this.props.scoreCard.name} />
        <thead><tr><th>Subject</th><th>Score</th></tr>
        </thead>
        <SubjectList
          subjectList={this.props.scoreCard.records} />
      </table>
    );
  }
}
const schoolRecord = {…}
ReactDOM.render(
  <ScoreCard scoreCard={schoolRecord} />,
  document.getElementById('root')
);
```

Ric's score

Subject	Score
Math	100
Chinese	87

Test Yourself #1

- ```
  const schoolRecord = ….;
  ReactDOM.render(
    <ScoreCard scoreCard=schoolRecord />,
    document.getElementById('root')
  );
  ```
  ==> Error! Why?
  - Syntax error: JSX value should be either an expression or a quoted JSX text
- So this is not OK either…
  ```
  ReactDOM.render(
    <ScoreCard score=100 />,
    document.getElementById('root')
  }
  ```
  ```

Test Yourself #2

- How about this?

```
class Welcome extends React.Component {
  render() {
    const n = 'Hello, ' + this.props.name;
    return <h1>n</h1>;
  }
}
```

  - Display “n” instead of “Hello Ric”…

- String or not string?

```
class ScoreCard extends Component {
  render() {
    return (
      …
      <Caption name={this.props.scoreCard.name} />
      <thead><tr><th>Subject</th><th>Score</th></tr>
      …
    );
  }
}
```

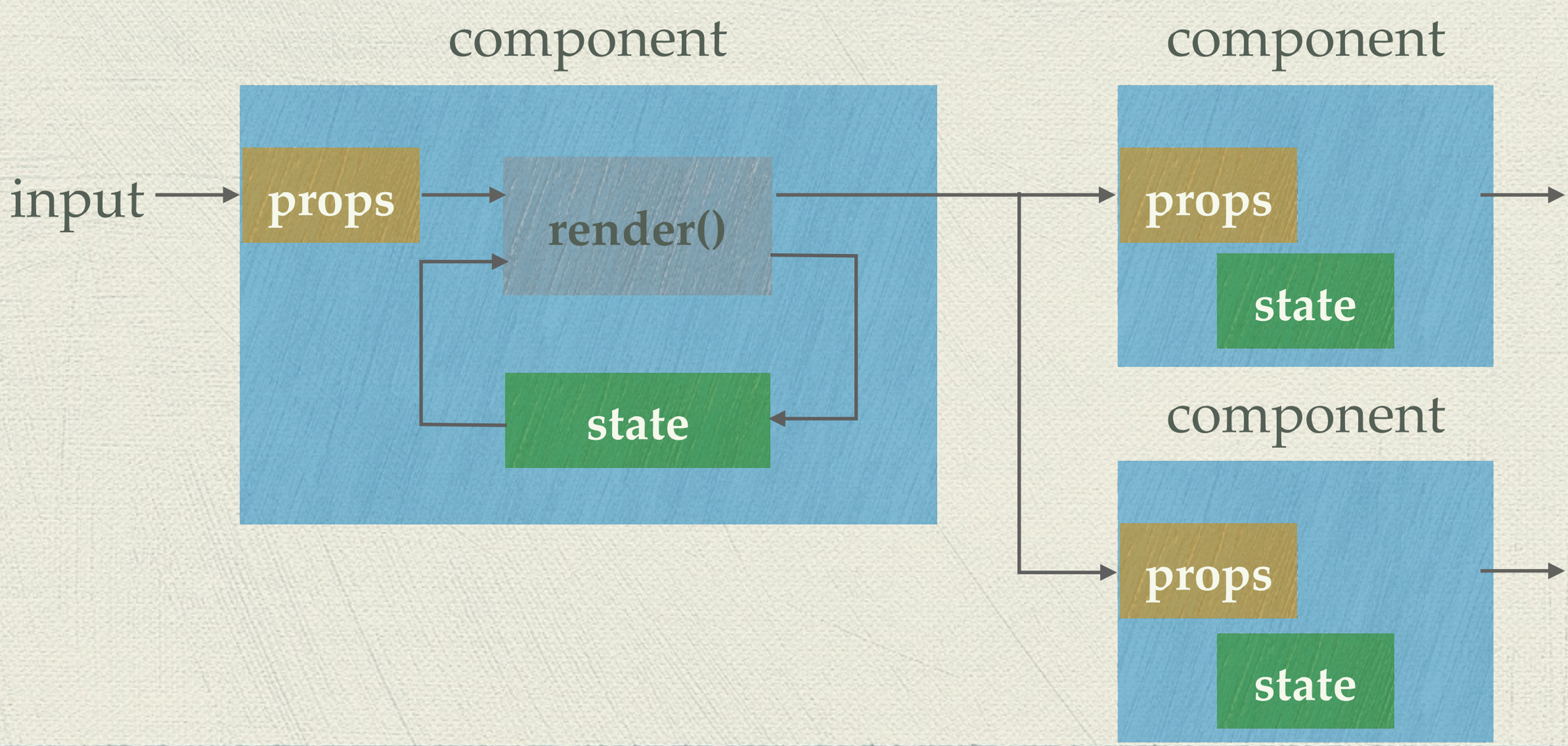
Test Yourself #3

- Or this?

```
class Welcome extends Component {
  render() {
    const n = 'Hello, ' + this.props.name;
    return (
      <h1>{n}</h1>
      <p>Nice to meet you!</p>
    );
  }
}
```

- Syntax error: Adjacent JSX elements must be wrapped in an enclosing tag

所以如果想要在 component 裡頭 “記住” 一些資訊(e.g.按讚數量,目前計算結果...)，用來增加網頁的互動，則需要用 “state”



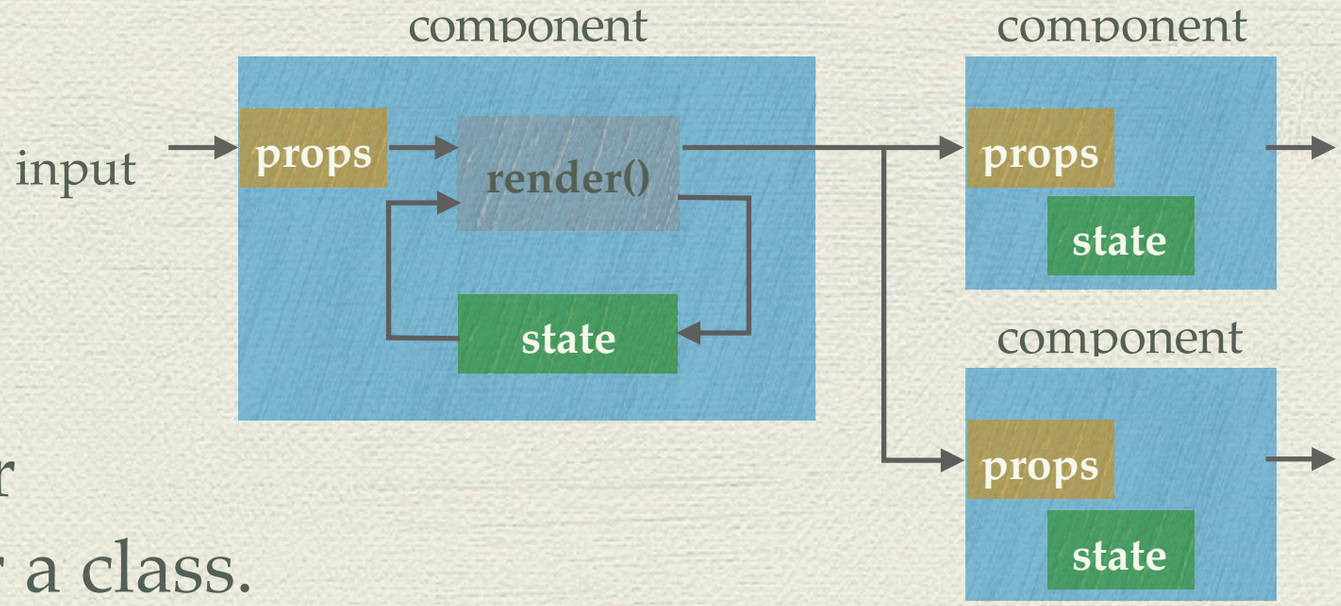
Using React state

```
class StatefulButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,  // This is to initialize state
      Record: [{ subject: 'Math', score: 100 },
               { subject: 'Chinese', score: 87 }
              ]
    };  // Define state as an JS object
  }
  render() {
…   return (<p>{this.state.count}</p>);
  }
}
```

Take a look again at this example…

```
class Clock extends Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()}; // This is just to initialize
  }
  componentDidMount() {
    this.timerID = setInterval(() => this.tick(), 1000);
  }                                  // This defines a functional object
  componentWillUnmount() { clearInterval(this.timerID); }
  tick() { this.setState({date: new Date()});} // Use setState()
  render() {                                   // to assign state
    return (                                   // Otherwise, won't
      <div>                                    // be re-rendered.
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
);}}
ReactDOM.render(<Clock />, document.getElementById('root'));
```

- Neither parent nor child components can know if a certain component is **stateful** or **stateless**, and they shouldn't care whether it is defined as a function or a class.



- This is why state is often called **local** or **encapsulated**. It is not accessible to any component other than the one that owns and sets it.
- A component may choose to pass its state down as props to its child components:

```
<h2>It is {this.state.date.toLocaleTimeString()}.</h2>
```

- This is commonly called a "**top-down**" or "unidirectional" data flow. Any state is always owned by some specific component, and any data or UI derived from that state can only affect components "below" them in the tree.

Ref: https://reactjs.org/docs/state-and-lifecycle.html

React Event Handling

- 兩件事 —
  1. 在用 JSX render React component 的時候把 event handler bind 到 React component 的 event property 上
     - `<button onClick={this.decNum}>{'-'}</button>`
     - `<button mouseOver={doSomething}>touch me</button>`

     ==> React event properties are camelCase
  2. If the handler is a function in the same class, you need to bind “this” to this event handler (下兩列種做法皆可)
     - ```
       class Counter extends React.Component {
         constructor(props) {
           …
           this.incNum = this.incNum.bind(this);
         }
         incNum() { … }
         render() { … }
       }
       ```
     - `<button onClick={(e)=>this.decNum(e)}>{'-'}</button>` // e 可省
       ```

About “bind()”…

- `funcObj.bind(x);`
  - bind(x) returns a reference to the caller “funcObj” and bind funcObj’s **this** to bind()’s parameter (i.e. x)
  - **`var foo = { x: 3 }`**

```
var bar = function(){
    console.log(this.x);
}

bar(); // undefined

var boundFunc = bar.bind(foo);

boundFunc(); // 3
```

Callback function

- bind() is especially useful for **callback** function
- A callback function is a function which —
  - passed as an argument to another function, and,
  - is invoked after the caller finish its task.
- ```
  downloadPhoto('http://url.com/xyz.gif',
    callBack); // Assume this take a while
  function callBack(error, photo) {
    … // error handling
    console.log('Download finished', photo)
  }
  console.log('Download started')
  ```
- Note: JavaScript is asynchronous. In the previous example, while downloadPhoto() has been executed, the message "Download started" will be logged, and after download finishes, the callback function will then be called to handle error or inform the download completion.
  ```

Callback function and bind()

- In JavaScript, class methods are not bound by default.
- Look at this example —

```
var myObj = {
    x: 3,
    callBack: function () { console.log(x); },
    doSomething: function(cb) {
      console.log("Do something…" + x);
      cb();
    },
    render: function () {
      doSomething(callBack);
    }
};
myObj.render();
```

- Anything wrong?
  > doSomething is not defined. ==> add “this”

Callback function and bind()

- Revise the example —
  - ```
    var myObj = {
      x: 3,
      callBack: function () { console.log("cb " + this.x); },
      doSomething: function(cb) {
        console.log("Do something…" + this.x);
        cb();
      },
      render: function () {
        this.doSomething(this.callBack);
      }
    };
    myObj.render();
    ```
- Do something…3
  cb undefined <== why?
- "callBack()" is called, but **this** is NOT **this**.
    ```

Callback function and bind()

- Revise again —

```
var myObj = {
    x: 3,
    callBack: function () { console.log("cb " + this.x); },
    doSomething: function(cb) {
      console.log("Do something…" + this.x);
      cb();
    },
    render: function () {
      var that = this;
      this.doSomething(that.callBack);
    }
};
myObj.render();
```

- Do something…3
  cb undefined <== still don't work…
- **this** of doSomething Does NOT bind to **this** of callBack.

Callback function and bind()

- Try again —
  - ```
    var myObj = {
      x: 3,
      callBack: function () { console.log("cb " + this.x); },
      doSomething: function(cb) {
        console.log("Do something…" + this.x);
        cb();
      },
      render: function () {
        this.doSomething(this.callBack.bind(this));
      }
    };
    myObj.render();
    ```
- Do something…3
  cb 3
- It works!! Passing callBack function to doSomething DOES NOT automatically bind **this**. Use bind() to bind **this** of doSomething to **this** of callBack.
    ```

Callback function and bind()

- Note: arrow function perform binding automatically —
  - ```
    var myObj = {
      x: 3,
      callBack: function () { console.log("cb " + this.x); },
      doSomething: function(cb) {
        console.log("Do something…" + this.x);
        cb();
      },
      render: function () {
        this.doSomething(()=>this.callBack());
      }
    };
    myObj.render();
    ```
- Do something…3
  cb 3
- Don't do —
  ```
  this.doSomething(this.callBack());
  ```
  ```

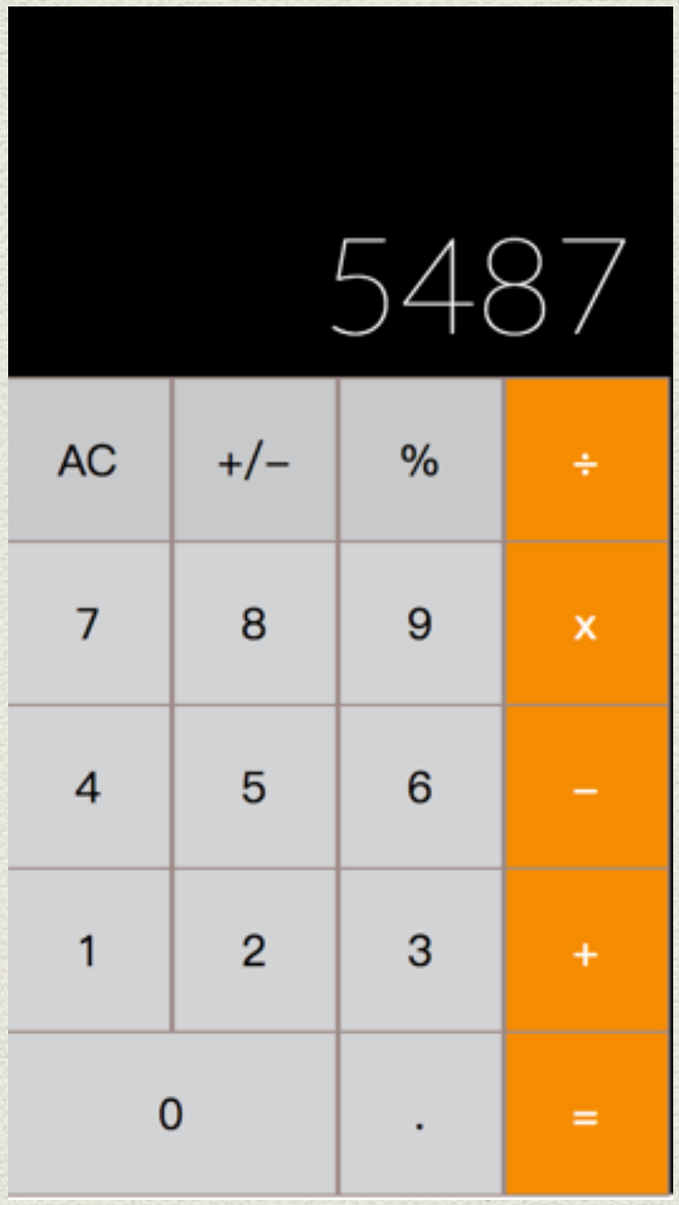
The inc/dec example last week…

- If for clarity reason we want to define three classes of components:

```
1.class Counter extends Component {
    constructor(props) { super(props);
      this.state = { num: 100 };
    }
    render() {
      return ( <span>
        <NumBox num={this.state.num} />
        <Button f={function(){ this.incNum()}.bind(this)} value='+' />
        <Button f={()=>this.decNum()} value='-' />
      </span>);
  }}
2.class NumBox extends Component {
    render() { return <h2>{this.props.num}</h2>; }
  }
3.class Button extends Component {
    render() {
      return
        <button onClick={this.props.f}>{this.props.value}</button>;
    }
  }
```

React.js 課堂練習

- **Calculator (iPhone look-alike)**
  1. 完成計算機之 UI
  2. 進階功能 (可以不用做) — 正負號 (+/-)、百分比(%)、小數點(.) 這三個按鍵只需要有 UI，不需要功能
  3. 置計算機 — AC (all clear) 按鍵，需要可以把計算狀態 (state) 全清空
  4. 數字按鍵 — 會跟一般計算機一樣按了會累積之後要當運算元的數字。例如依序按下 6, 1, 9, 8，會產生 6198 來當作運算元
     - 因為不實作小數當作運算元，所以第一個數字按零可以忽略
  5. 運算子按鍵 (+, -, x, ÷, =)
     - 如果前面已經有 pending 的運算則先把前面的運算求值：
       - 按下 3, +, 6：這時候畫面還是 6
       - 但再按下 (+, -, x, ÷, =) 任何一個就會更新運算結果
     - 如果前面也是運算子，則用新的運算子把舊的換掉
       - 按下 3, +, -, 1, =：這時候結果為 2，+ 會被 - 換掉
     - 如果前面是運算元，則先 pending 存至 state



React.js 課堂練習

- **Reference code**
  1. 完成計算機之 UI — 右圖為目前 ref code 的樣貌
  2. **進階功能** (可以不用做) — 正負號 (+/-)、百分比(%)、小數點(.) 這三個按鍵只需要有 UI，不需要功能
  3. **置計算機** — AC (all clear) 按鍵，需要可以把計算狀態 (state) 全清空
  4. 數字按鍵 — 會跟一般計算機一樣按了會累積之後要當運算元的數字。例如依序按下 6, 1, 9, 8，會產生 6198 來當作運算元
     - 因為不實作小數當作運算元，所以第一個數字按零可以忽略
  5. 運算子按鍵 (+, -, x, ÷, =)
     - 如果前面已經有 pending 的運算則先把前面的運算求值：
       - 按下 3, +, 6：這時候畫面還是 6
       - 但再按下 (+, -, x, ÷, =) 任何一個就會更新運算結果
     - 如果前面也是運算子，則用新的運算子把舊的換掉
       - 按下 3, +, -, 1, =：這時候結果為 2，+ 會被 - 換掉
     - 如果前面是運算元，則先 pending 存至 state

React.js 課堂練習

- **TODOs**
  1. Download ref code from Ceiba
  2. Uncompress it
  3. npm intall
  4. The codes to implement are in src directory
  5. Enjoy hacking!

Thinking in React

- Step 1: Break The UI Into A Component Hierarchy
- Step 2: Build A Static Version in React
- Step 3: Identify The Minimal (but complete) Representation Of UI State
- Step 4: Identify Where Your State Should Live
- Step 5: Add Inverse Data Flow