

React Props, State, Event

BY PANOT WONGKHOT

FRONT-END PROGRAMMER BOOTCAMP

<https://github.com/panotza/frontend-bootcamp>



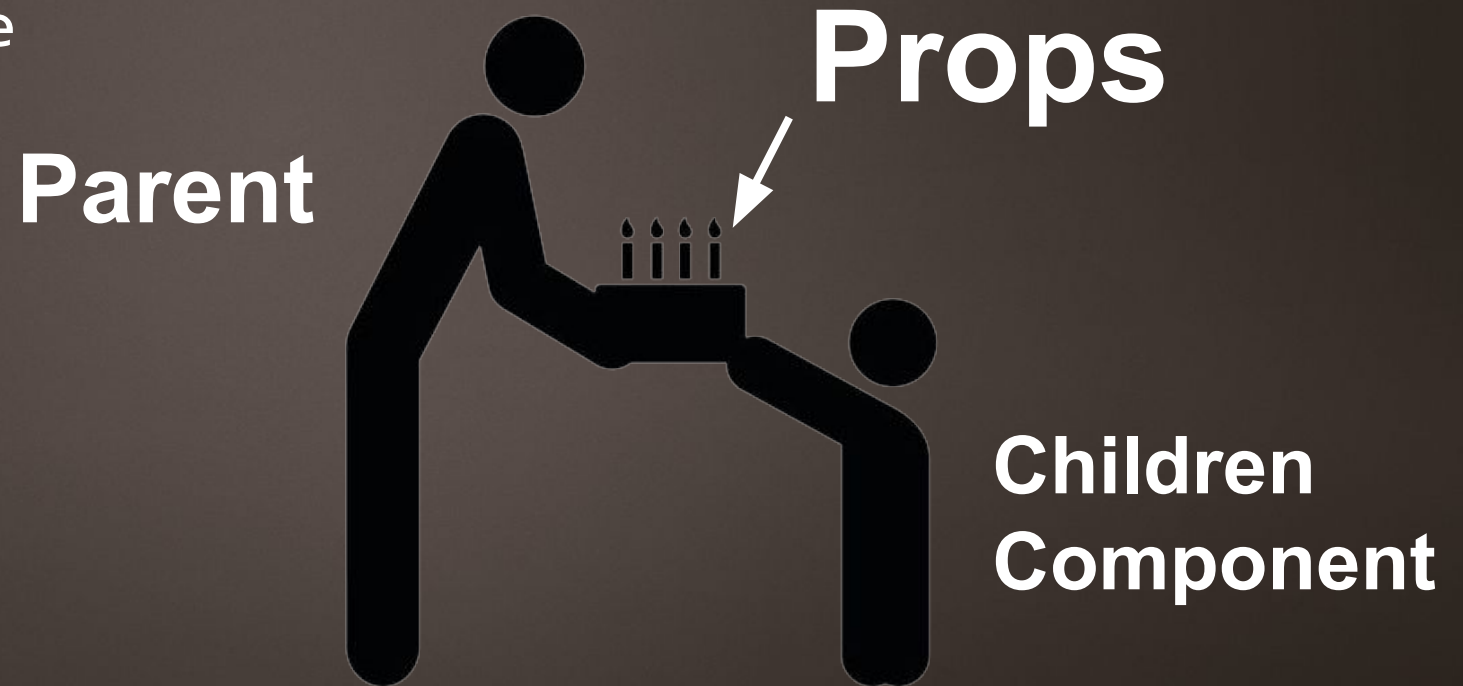
Intro to Props



Props

props (ย่อมาจาก properties) คือข้อมูลที่ใช้ภายใน Component โดย props จะถูกส่งมาจาก Parent ที่เรียกใช้ Component นั้นๆ (children component)

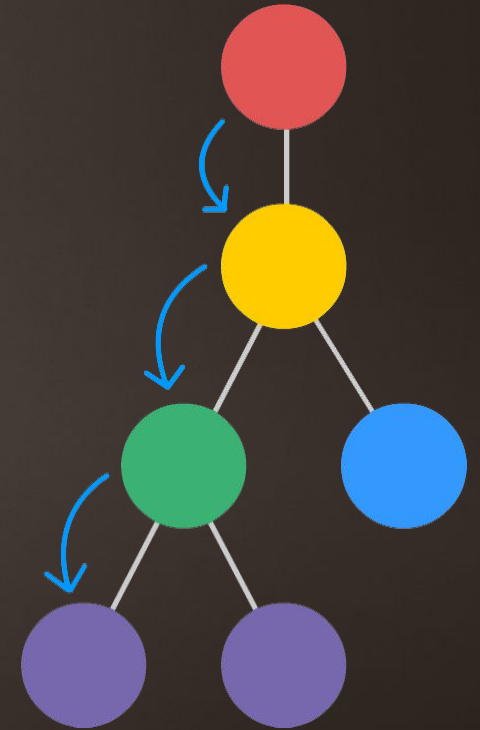
ใน Javascript props เป็นแค่ **object { }** ธรรมดาๆ นี้อเอง ดังนั้น props จึงมีได้หลายค่า และเก็บค่าในรูปแบบ key: value



Props: One way direction data flow

การที่ parent ส่ง props ไปหา children component ได้อย่างเดียว
ส่ง props ขึ้นไปหา parent ตัวเองไม่ได้ ส่ง props ไปหาญาติ (sibling) ด้านข้างไม่ได้
ทำให้เรามองภาพการไหลของ props ได้เหมือนน้ำตก
กล่าวคือข้อมูลของ props จะไหลลงเท่านั้น
เราเรียกการส่งข้อมูล pattern แบบนี้ว่า

One-way direction data flow



ข้อควรระวังเรื่อง Props

ห้ามแก้ไขค่า props โดยเด็ดขาด

เพราะ props ถูกส่งมาจาก parent เราจึงต้องเคารพค่าที่ parent ส่งมาให้
คิดซะว่า props มีสถานะเป็น **read only** หากมีกรณีที่เราต้อง
เปลี่ยนแปลงค่า props ให้คิดดูดีๆว่าเราออกแบบการส่งข้อมูลถูกหรือเปล่า



Passing Props



การส่ง Props

หากเคยเขียน html มาก่อนจะพบว่า การส่ง props ให้ Component มีความคล้ายกับการกำหนด attribute ใน html มาก เพียงแต่ว่าเรากำหนดชื่อ attribute เองได้ ส่งค่าเป็น type อื่นที่ไม่ใช่ string ได้ และไม่จำกัดจำนวน

เช่น

```
<Hello firstName="john" age={30} />
```

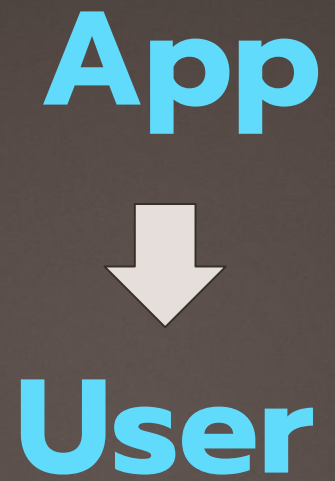
โดยเราให้ props ที่มี key ชื่อว่า firstName มีค่าเท่ากับ "john" และ key ชื่อว่า age มีค่าเท่ากับ 30 นั้นเอง



การรับ Props

- หากเป็น Component ที่สร้างจาก Function จะรับค่า props จาก **parameter** ที่ 1 ของ Function
- หากเป็น Component ที่สร้างจาก Class จะรับค่า props ได้จาก **this.props**





ตัวอย่างการส่ง props จาก App component ไปหา User component

Parent ส่งค่า prop ให้ component

ดูได้ใน props.html



```
class App extends React.Component {  
  render() {  
    return (  
      <div>  
        <UserFunc name="John" age={30} />  
        <UserClass name="John" age={30} />  
      </div>  
    );  
  }  
}  
  
ReactDOM.render(<App />, document.getElementById('root'));
```

Function component รับค่า props

ดูได้ใน props.html



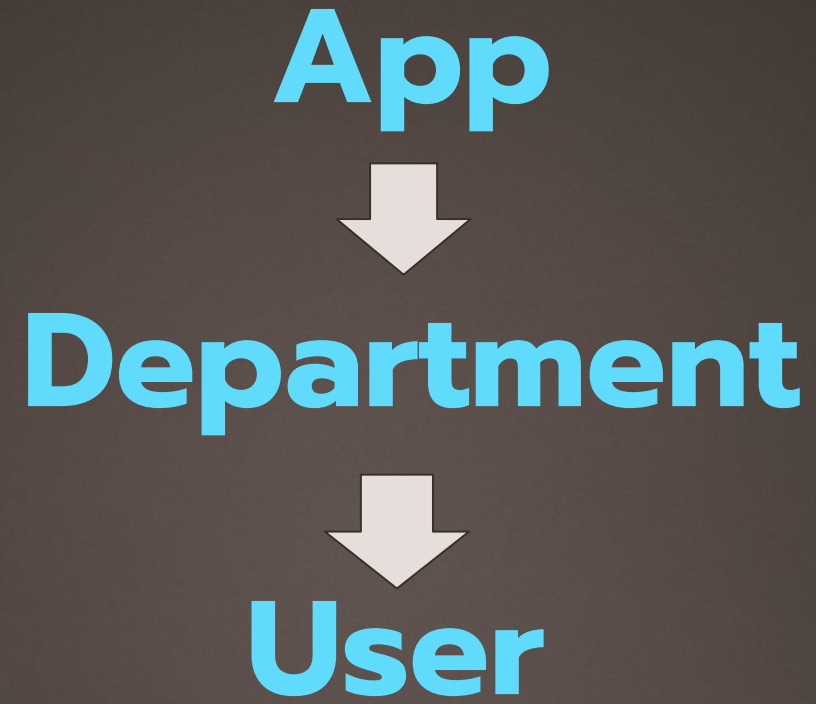
```
const UserFunc = (props) => {  
  return (  
    <div>  
      <h1>From User Function Component:</h1>  
      <p>name: {props.name}</p>  
      <p>age: {props.age}</p>  
    </div>  
  );  
}
```

Class component รับค่า props

ดูได้ใน props.html

```
class UserClass extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>From User Class Component:</h1>  
        <p>name: {this.props.name}</p>  
        <p>age: {this.props.age}</p>  
      </div>  
    );  
  }  
}
```





ตัวอย่างการส่ง props จาก App component ไปหา User component ที่อยู่ใน Department component

การส่งค่า props ให้ nested component

ดูได้ใน props-nested-component.html

```
class App extends React.Component {  
  render() {  
    return (  
      <Department name="John" age={30} />  
    );  
  }  
}  
  
ReactDOM.render(<App />, document.getElementById('root'));
```

การส่งค่า props ให้ nested component

ดูได้ใน props-nested-component.html

```
class Department extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>Department: Accounts</h1>  
        <User name={this.props.name} age={this.props.age} />  
      </div>  
    );  
  }  
}
```

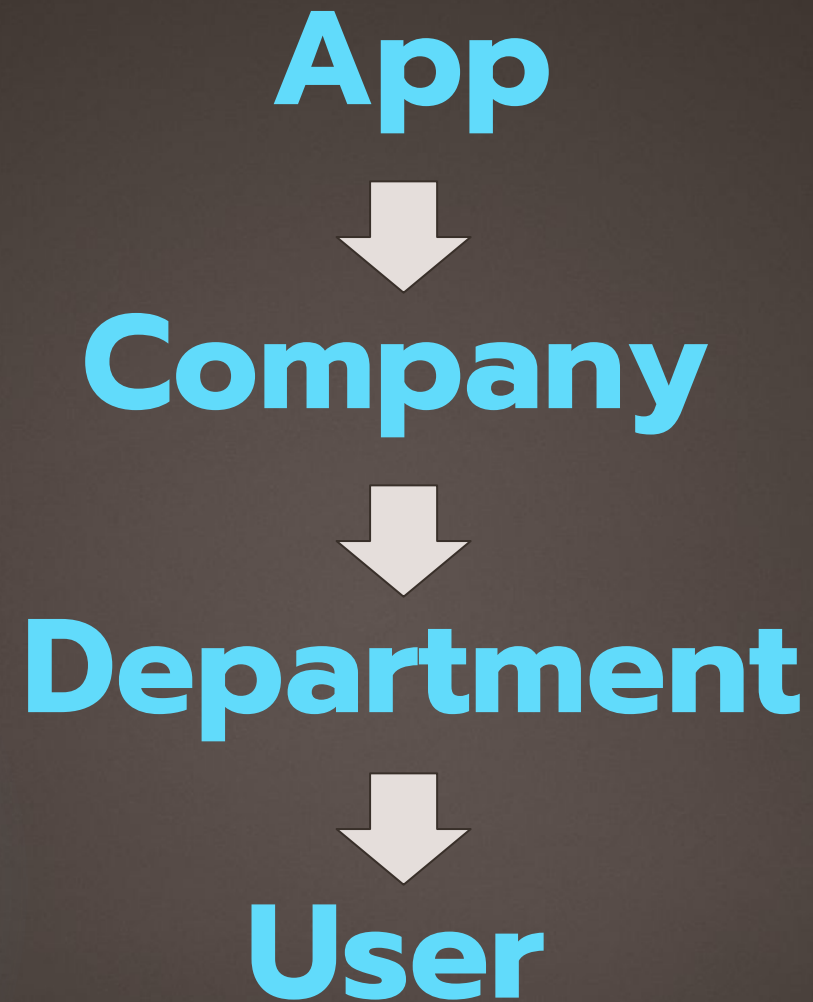
การส่งค่า props ให้ nested component



ดูได้ใน props-nested-component.html

```
const User = (props) => {  
  return (  
    <div>  
      <p>name: {props.name}</p>  
      <p>age: {props.age}</p>  
    </div>  
  );  
}
```





ตัวอย่างการส่ง props จาก App component ไปหา User component ที่อยู่ใน
Department component ที่อยู่ใน Company component

การส่งค่า props ให้ nested component ภาค 2



ดูได้ใน props-deeper-nested-component.html

```
class App extends React.Component {  
  render() {  
    return (  
      <Company name="John" age={30} />  
    );  
  }  
}  
  
ReactDOM.render(<App />, document.getElementById('root'));
```

การส่งค่า props ให้ nested component ภาค 2



ดูได้ใน props-deeper-nested-component.html

```
class Company extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>Company: AirCnC</h1>  
        <Department name={this.props.name} age={this.props.age} />  
      </div>  
    );  
  }  
}
```

การส่งค่า props ให้ nested component ภาค 2



ดูได้ใน props-deeper-nested-component.html

```
class Department extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>Department: Accounts</h1>  
        <User name={this.props.name} age={this.props.age} />  
      </div>  
    );  
  }  
}
```

การส่งค่า props ให้ nested component ภาค 2



ดูได้ใน props-deeper-nested-component.html

```
const User = (props) => {  
  return (  
    <div>  
      <p>name: {props.name}</p>  
      <p>age: {props.age}</p>  
    </div>  
  );  
}
```


Props

Type อะไรก็ตามที่เราใส่ใน Object ได้ เราสามารถส่งเข้า props ได้หมดครับ

- Boolean
- Null
- Undefined
- Number
- String
- Object -> Array, Date
- Function






```
class App extends React.Component {  
  showNameFunc = (name) => {  
    return 'my name is ' + name;  
  }  
  render() {  
    const str = 'hello react!';  
    const number = 5;  
    const bool = false;  
    const nullValue = null;  
    const undefinedValue = undefined;  
    const obj = { name: 'somchai', b: 2 };  
    const arr = ['john', 2, 'smith', 4];  
    const now = new Date();
```

```
    return (
```

ดูได้ใน props-all-type.html

```
      <Display  
        myStr={str}  
        myNumber={number}  
        myBool={bool}  
        myTrueBool  
        myNull={nullValue}  
        myUndefined={undefinedValue}  
        myObj={obj}  
        myArray={arr}  
        myDate={now}  
        myShowNameFunc={this.showNameFunc}  
      />  
    );  
  }  
}  
  
ReactDOM.render(<App />,  
  document.getElementById('root'));
```



```

class Display extends React.Component {
  render() {
    return (
      <div>
        <p>string: {this.props.myStr}</p>
        <p>number: {this.props.myNumber}</p>
        <p>>false value: {this.props.myBool.toString()}</p>
        <p>>true value: {this.props.myTrueBool.toString()}</p>
        <p>null: {this.props.myNull === null && 'yes'}</p>
        <p>undefined: {this.props.myUndefined === undefined && 'yes'}</p>
        <p>object: {JSON.stringify(this.props.myObj)}</p>
        <p>array: {JSON.stringify(this.props.myArray)}</p>
        <p>Date: {this.props.myDate.toString()}</p>
        <p>function: {this.props.myShowNameFunc('John')}</p>
      </div>
    );
  }
}

```

props.children



Props: children

Component สามารถรับค่า children ของตัวเองได้

- หากเป็น Component ที่สร้างจาก Function จะรับค่า children จาก **props.children**
- หากเป็น Component ที่สร้างจาก Class จะรับค่า children ได้จาก **this.props.children**

การส่งค่า children ให้ component



ดูได้ใน props-children.html

```
class App extends React.Component {  
  render() {  
    return (  
      <div>  
        <UserFunc>Lorem ipsum</UserFunc>  
        <UserClass>Beacon ipsum</UserClass>  
      </div>  
    );  
  }  
}  
  
ReactDOM.render(<App />, document.getElementById('root'));
```

Function component รับค่า children



ดูได้ใน props-children.html

```
const UserFunc = (props) => {  
  return (  
    <div>  
      <h1>From User Function Component:</h1>  
      <p>{props.children}</p>  
    </div>  
  );  
}
```

Class component รับค่า children



ดูได้ใน props-children.html

```
class UserClass extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>From User Class Component:</h1>  
        <p>{this.props.children}</p>  
      </div>  
    );  
  }  
}
```



ช่วงรู้มือไร

ตอนที่เรากำหนด Style ด้วย className หรือ inline-style หรือแม้แต่
ตอน Loop ที่เรากำหนด key ให้กับ Component ก็ตาม
เราส่ง props ให้ Component อื่นๆ โดยที่เราไม่รู้ตัวไปแล้วเรียบร้อยแล้ว

กล่าวคือทุกอย่างที่เขียนใน tag ตามหลังชื่อ
Component เราเรียกว่า props ทั้งหมด




FlashBack



```
const helloReact = React.createElement(  
  'div',  
  null, <-- นี่แหละที่สำหรับใส่ Props  
  'Hello React!'  
) ;
```


FlashBack



```
const helloReact = React.createElement(  
  'div',  
  { firstName: "John",  
    lastName: "Doe",  
    key: 1 },  
  'Hello React!'  
);
```

FlashBack



```
const helloReact = (  
  <div  
    key={1}  
    firstName="John"  
    lastName="Doe"  
  >  
    Hello React  
  </div>  
)
```

FlashBack

▼ Object

```
  $$typeof: Symbol(react.element)
  key: "1"
  ▶ props: {firstName: "John", lastName: "Doe", children: "Hello React"}
  ref: null
  type: "div"
  _owner: null
  ▶ __proto__: Object
```

Lab 1: Props

เปิดไฟล์ lab.html ขึ้นมาแล้วลองทำสิ่งต่อไปนี้

- หัดส่ง props (โดยเฉพาะหัดส่งfunction ไปใน props)
- หัดเขียน component รับ props ทั้งแบบ class และ function
- หัดใช้ props children
- พยายามทำความเข้าใจการส่ง props ให้ nested component

Default Props



Default Props

ใช้เพื่อกำหนดค่าตั้งต้นให้ props ใน component หาก props ไม่ถูกส่งมาก็คะใช้ค่านี้แทน



การใช้ Default Props

● ● ● ใช้ได้ทั้ง Component แบบ Class และ Function *แนะนำ*

```
Component.defaultProps = {  
  ชื่อ prop: value,  
  ชื่อ prop: value  
};
```

การใช้ Default Props แบบ 2

● ● ● ภายใน Class

```
static defaultProps = {  
  ชื่อ prop: value,  
  ชื่อ prop: value  
};
```

Default Props: Function Component



ดูได้ใน default-props.html

```
const UserFunc = (props) => {  
  return (  
    <div>  
      <h1>From User Function Component:</h1>  
      <p>First name: {props.firstName}</p>  
      <p>Last name: {props.lastName}</p>  
    </div>  
  );  
}
```

Default Props: Function Component



ดูได้ใน default-props.html

```
UserFunc.defaultProps = {  
  firstName: 'john',  
  lastName: 'Doe'  
};
```

Default Props: Class Component



ดูได้ใน default-props.html

```
class UserClass extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>From User Class Component:</h1>  
        <p>First name: {this.props.firstName}</p>  
        <p>Last name: {this.props.lastName}</p>  
      </div>  
    );  
  }  
}
```

Default Props: Class Component

ดูได้ใน default-props.html

```
UserClass.defaultProps = {  
  firstName: 'john',  
  lastName: 'Doe'  
};
```


Default Props: Class Component แบบ 2



ดูได้ใน default-props.html

```
class UserClass extends React.Component {  
  static defaultProps = {  
    firstName: 'john',  
    lastName: 'Doe'  
  };  
  
  render() {  
    return (  
      <div>  
        <h1>From User Class Component</h1>  
        <p>First name: {this.props.firstName}</p>  
        <p>Last name: {this.props.lastName}</p>  
      </div>  
    );  
  }  
}
```

PropTypes




PropTypes

Javascript เป็นภาษา Weak typing หาก App ของเรามีขนาดใหญ่ขึ้นเราอาจจะอยากให้มีการเช็ค type ของ props ที่ส่งเข้ามาว่าเป็นไปตามต้องการหรือไม่ ด้วยเหตุนี้เราจึงใช้ PropTypes เพื่อช่วยเช็ค type ของ props

- เราอาจจะมองว่า PropTypes ช่วยบอก specification ของ component ก็ได้

*เราต้องลง library ชื่อ prop-types เพิ่มหากต้องการจะใช้ PropTypes

วิธีลง PropTypes

 เปิด terminal ในโฟลเดอร์โปรเจกต์แล้วพิมพ์

```
npm install --save prop-types
```

หรือ

```
yarn add prop-types
```

การ Import PropTypes

● ● ● ด้านบนในไฟล์ component ของเรา

```
import PropTypes from 'prop-types';
```

การใช้ PropTypes

● ● ● ใช้ได้ทั้ง Component แบบ Class และ Function *แนะนำ*

```
Component.propTypes = {  
  ชื่อ prop: type ที่กำหนด,  
  ชื่อ prop: type ที่กำหนด  
};
```


การใช้ PropTypes แบบ 2

● ● ● ภายใน Class

```
static propTypes = {  
  ชื่อ prop: type ที่กำหนด,  
  ชื่อ prop: type ที่กำหนด  
};
```

Type ที่ใช้บ่อย



ดูเพิ่มเติมได้ที่ <https://reactjs.org/docs/typechecking-with-proptypes.html>

PropTypes.array

PropTypes.bool

PropTypes.func

PropTypes.number

PropTypes.object

PropTypes.string

PropTypes.oneOfType

PropTypes.arrayOf

PropTypes.objectOf

PropTypes.shape

PropTypes.func.isRequired

PropTypes: Function Component



ดูได้ใน prop-types.html

```
const UserFunc = (props) => {  
  return (  
    <div>  
      <h1>From User Function Component:</h1>  
      <p>name: {props.name}</p>  
      <p>age: {props.age}</p>  
    </div>  
  );  
}
```

PropTypes: Function Component



ดูได้ใน prop-types.html

```
UserFunc.propTypes = {  
  name: PropTypes.string,  
  age:  PropTypes.number  
};
```

PropTypes: Class Component



ดูได้ใน prop-types.html

```
class UserClass extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>From User Class Component:</h1>  
        <p>name: {this.props.name}</p>  
        <p>age: {this.props.age}</p>  
      </div>  
    );  
  }  
}
```

PropTypes: Class Component

ดูได้ใน prop-types.html

```
UserClass.propTypes = {  
  name: PropTypes.string,  
  age: PropTypes.number  
};
```


PropTypes: Class Component แบบ 2



ดูได้ใน prop-types.html

```
class UserClass extends React.Component {  
  static propTypes = {  
    name: PropTypes.string,  
    age: PropTypes.number  
  };  
  render() {  
    return (  
      <div>  
        <h1>From User Class Component</h1>  
        <p>name: {this.props.name}</p>  
        <p>age: {this.props.age}</p>  
      </div>  
    );  
  }  
}
```

- ❌ ▶ Warning: Failed prop type: Invalid prop `age` of type `string` supplied to `UserFunc`, expected `number`.
in UserFunc (created by App)
in App
- ❌ ▶ Warning: Failed prop type: Invalid prop `age` of type `string` supplied to `UserClass`, expected `number`.
in UserClass (created by App)
in App

Lab 2: Default Props & PropTypes

เปิดไฟล์ lab2.html ขึ้นมาแล้วลองทำสิ่งต่อไปนี้

- หัดใช้ Default Props
- หัดใช้ PropTypes



Handling Events



การจัดการ Event

ใน React เราสามารถจัดการ event ที่เกิดจาก action ของ user ได้เช่นเดียวกับเวลาที่เราเขียน HTML โดยจะจัดการผ่าน **Event Handler** และ **SyntheticEvent**

ตัวอย่าง event

- user คลิกปุ่ม
- user พิมพ์ในกล่อง input
- กล่อง input ถูก focus



Event Handler

Event handler ก็คือ **Callback Function** ธรรมดาๆ ที่เรากำหนดให้เป็นค่าของ event (props) ที่ส่งให้กับ element

โดยเราส่ง Event handler เข้าไปเพื่อที่จะรับ **SyntheticEvent** กลับมา

*ควรเขียน Function แบบ Arrow Function เพื่อแก้ปัญหา this เปลี่ยน

ดูรายชื่อ Event ทั้งหมดได้ที่ <https://reactjs.org/docs/events.html#supported-events>



SyntheticEvent

ใน React จะมีสิ่งที่เรียกว่า SyntheticEvent ซึ่งจะช่วยให้ event ในทุก Browser ทำงานได้ไม่แตกต่างกัน โดยเราจะได้รับ SyntheticEvent จาก Event Handler หลังจากเกิด Event นั้นเอง

ใน Javascript SyntheticEvent ก็คือ `Object { }` ธรรมดาๆ นี่เอง



SyntheticEvent

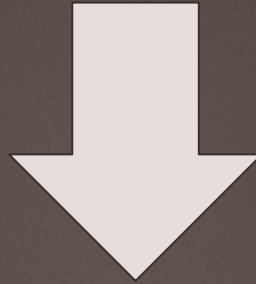
ใน SyntheticEvent มี key ที่ใช้บ่อยๆคือ

- .target ใช้สำหรับอ่านค่าจาก element ที่เกิด event
- .preventDefault() ใช้ยกเลิก default action ของ HTML



Func

Event Handler



Obj

SyntheticEvent



ตัวอย่างการจัดการ event



ดูได้ใน events-onclick.html

```
class App extends React.Component {  
  handleClick = (e) => {  
    alert('button was clicked');  
  }  
  render() {  
    return (  
      <button onClick={this.handleClick}>Click me</button>  
    );  
  }  
}
```



```
<button onClick={this.handleClick}>Click me</button>
```

Props คือ onClick

Event Handler



Synthetic Event

```
handleClick = (e) => {  
    alert('button was clicked');  
}
```


ตัวอย่างการอ่านค่าใน SyntheticEvent



ดูได้ใน events-onchange.html

```
class App extends React.Component {  
  handleChange = (e) => {  
    console.log(e.target.name);  
    console.log(e.target.value);  
  }  
  render() {  
    return (  
      <input name="myInput" onChange={this.handleChange} />  
    );  
  }  
}
```

ตัวอย่างการอ่านค่าใน SyntheticEvent



ดูได้ใน events-onkeyup.html

```
class App extends React.Component {  
  handleKeyUp = (e) => {  
    if (e.keyCode === 13) { // Enter key  
      console.log('Enter was pressed');  
    }  
  }  
  
  render() {  
    return (<input name="myInput" onKeyUp={this.handleKeyUp} />);  
  }  
}
```

ตัวอย่าง Form Event

ดูได้ใน events-form.html

```
class App extends React.Component {  
  handleSubmit = (e) => {  
    e.preventDefault();  
    alert('form was submitted');  
  }  
  render() {  
    return (  
      <form onSubmit={this.handleSubmit}>  
        <input type="submit" value="Submit" />  
      </form>  
    );  
  }  
}
```

ตัวอย่างการใส่ Parameter

ดูได้ใน `events-pass-parameter.html`

```
class App extends React.Component {  
  showAlert = (value) => {  
    alert('hello ' + value);  
  }  
  render() {  
    return (  
      <button  
        onClick={() => this.showAlert('john')}>  
        Click Me  
      </button>  
    );  
  }  
}
```

ตัวอย่าง children component เรียกใช้ Function ของ Parent



ดูได้ใน events-props.html

```
const MyTitle = (props) => {  
  return (<h1 onClick={() => props.showAlert('john')}>Click Me!</h1>);  
}  
  
class App extends React.Component {  
  showAlert = (value) => {  
    alert('hello ' + value);  
  }  
  
  render() {  
    return (<MyTitle showAlert={this.showAlert} />);  
  }  
}
```

ตัวอย่างการใช้ Event Handler ที่ผิดบ่อยๆ



```
class App extends React.Component {  
  handleClick = (e) => {  
    alert('button was clicked');  
  }  
  render() {  
    return (  
      <button onClick={this.handleClick()}>Click me</button>  
    );  
  }  
}
```

อย่าใส่ () ตามหลัง

ReactDOM.render(<App />, document.getElementById('root'));

คำเตือนเรื่อง SyntheticEvent

SyntheticEvent ใช้ระบบ pooling กล่าวคือ SyntheticEvent จะถูกนำกลับมาใช้ใหม่ (reuse) โดยการทำให้ข้อมูลเป็น null หลังจากรัน callback จบแล้ว ดังนั้นหากเราจะเข้าถึง SyntheticEvent แบบ async เราจึงจำเป็นต้องใช้ method persist ช่วย

อ่านเพิ่มเติมได้ที่ <https://reactjs.org/docs/events.html#event-pooling>



Lab 3: Event

- จาก lab 2
- ลองหัดใช้ event ดังนี้
- onClick, onChange, onKeyUp, onSubmit
- ลองใส่ parameter (สำคัญ)



State



State

state คือข้อมูลที่ใช้ภายใน Component สามารถเปลี่ยนแปลงค่าได้

ใน Javascript state เป็นแค่ `object { }` ธรรมดาๆ นี่เอง ที่เป็น property ของ class ดังนั้น state จึงมีได้หลายค่าและเก็บค่าในรูปแบบ `key: value` เราสามารถอ่านค่าได้จาก `this.state`

*State สามารถใช้ได้แค่ Component ที่สร้างจาก Class เท่านั้น



State - Immutable Object

ผู้สร้าง React แนะนำให้คิดว่า state มีคุณสมบัติเป็น **Immutable** Object (สร้างขึ้นมาแล้วแก้ไขหรือเปลี่ยนแปลงไม่ได้)

ดังนั้นเราจึง **ห้ามแก้ไข state โดยตรงเด็ดขาด**

หากจะเปลี่ยนแปลงค่า เราต้องสร้าง state ใหม่มาแทนที่ state เดิมผ่าน

Function **this.setState()**



การกำหนดค่าเริ่มต้นให้ State แบบที่ 1

ดูได้ใน initialize-state.html



```
class App extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      name: 'John',  
      age: 30,  
      skills: [{ id: 1, name: 'React' },  
               { id: 2, name: 'NodeJS' }]  
    };  
  }  
}
```

การกำหนดค่าเริ่มต้นให้ State แบบที่ 2

ดูได้ใน initialize-state.html

```
class App extends React.Component {  
  state = {  
    name: 'John',  
    age: 30,  
    skills: [{ id: 1, name: 'React' },  
              { id: 2, name: 'NodeJS' }]  
  };  
}
```

การกำหนดค่าเริ่มต้นให้ State

ดูได้ใน initialize-state.html



```
render() {  
  return (  
    <div>  
      <p>name: {this.state.name}</p>  
      <p>age: {this.state.age}</p>  
      <ul>  
        {this.state.skills.map(skill =>  
          <li key={skill.id}>{skill.name}</li>)}  
      </ul>  
    </div>  
  );  
}
```


รู้จักกับ setState()



setState

setState เป็น Function (Method) ของ class Component ใช้สำหรับเปลี่ยนแปลงค่า State ภายใน Component นั้นๆ

การทำงานของ setState คือการนำ state ใหม่ไปแทนที่ state เก่า
แต่เวลาใช้จริงๆ เราใส่แค่สิ่งที่เปลี่ยนแปลงให้ setState ก็เพียงพอแล้ว

*setState ทำงานแบบ Async

*setState มีผลทำให้ component render ใหม่



Pure function กับ setState

State เป็น Immutable Object ดังนั้นเราจึงควรใช้แต่ **Pure Function** ร่วมกับ setState เช่น Object.assign, concat, slice, spread operator



Anatomy of setState



ส่ง new state เป็น object { } หรือเป็น function

`this.setState(newState, callback);`

Callback เมื่อ setState เสร็จแล้ว

ตัวอย่างการใช้งาน setState

ดูได้ใน `setstate-basic.html`

```
class App extends React.Component {  
  handleChangeName = () => {  
    this.setState({ name: 'smith' });  
  }  
  <button onClick={this.handleChangeName}>change name</button>  
}
```

ตัวอย่างการใช้งาน setState

ดูได้ใน `setstate-basic.html`

```
class App extends React.Component {  
  handleChangeAge = () => {  
    this.setState({ age: 45 });  
  }  
  <button onClick={this.handleChangeAge}>change age</button>  
}
```

ตัวอย่างการใช้งาน setState

ดูได้ใน `setstate-basic.html`



```
class App extends React.Component {  
  handleChangeSkill = () => {  
    this.setState({  
      skills: [  
        { id: 3, name: 'Java' },  
        { id: 4, name: 'C++' },  
        { id: 5, name: 'Swift' }  
      ]  
    });  
  }  
  <button onClick={this.handleChangeSkill}>change skills</button>  
}
```


Real world use case



Controlled Component

คือ component ที่ value ของมันอ้างอิงค่าจาก state
controlled component มักจะ implement การ update ค่าของ state ด้วย event onChange ไม่เช่นนั้น controlled component จะไม่สามารถเปลี่ยนแปลง value ได้

*component ที่ value ของมันไม่ได้อ้างอิงค่าจาก state เรียกว่า
Uncontrolled Component



ตัวอย่างการใช้ Input กับ State



ดูได้ใน `setstate-input.html`

```
class App extends React.Component {  
  state = {  
    email: ''  
  };  
  handleChange = (e) => {  
    this.setState({ email: e.target.value });  
  }  
}
```

```
render() {  
  return (  
    <input  
      type="text"  
      value={this.state.email}  
      onChange={this.handleChange}  
    />  
  );  
}
```

ตัวอย่างการใช้ Form กับ State



ดูได้ใน `setstate-form.html`

```
class App extends React.Component {  
  state = {  
    email: '',  
    password: ''  
  };  
  handleSubmit = (e) => {  
    e.preventDefault();  
    console.log(this.state);  
  }  
  handleChange = (e) => {  
    this.setState({ [e.target.name]: e.target.value });  
  }  
}
```

ตัวอย่างการใช้ Form กับ State



ดูได้ใน `setstate-form.html`

```
render() {  
  return (  
    <form onSubmit={this.handleSubmit}>  
      email:<br />  
      <input  
        type="text"  
        name="email"  
        value={this.state.email}  
        onChange={this.handleChange}  
      /><br />  
    )  
  }  
}
```

```
password:<br />  
  <input  
    type="password"  
    name="password"  
    value={this.state.password}  
    onChange={this.handleChange}  
  /><br />  
  <button type="submit">Login</button>  
</form>  
)  
};
```

ส่ง State ไปใน props ให้กับ children component



ดูได้ใน passing-state.html

```
class App extends React.Component {  
  state = {  
    skills: [{ id: 1, name: 'React' }, { id: 2, name: 'NodeJS' } ]  
  };  
  render() {  
    return (<SkillList skills={this.state.skills} />);  
  }  
}
```

ส่ง State ไปใน props ให้กับ children component

ดูได้ใน passing-state.html

```
const SkillList = (props) => (  
  <ul>  
    {props.skills.map(skill =>  
      <li key={skill.id}>{skill.name}</li>)}  
  </ul>  
);
```


Array, Object กับ setState

การ setState Array กับ Object ใน Javascript เป็นเรื่องยากสำหรับมือใหม่เนื่องจากปัญหาเรื่อง Pointer เราจึงขอแนะนำให้ศึกษาเรื่อง Pointer, map, filter, spread operator ให้ชำนาญ ก็จะทำให้ลดความยากลงไปได้





ดูได้ใน [setstate-array-object.html](#)

```
state = {  
  skills: [  
    {  
      id: 1,  
      name: 'React'  
    },  
    {  
      id: 2,  
      name: 'NodeJS'  
    }  
  ]  
};
```

ตัวอย่างการเพิ่มสมาชิกใน array ด้วย setState



ดูได้ใน [setstate-array-object.html](#)

```
handleAddSkill = () => {  
  this.setState({  
    skills: [...this.state.skills, { id: 4, name: 'Java' }]  
  });  
}
```

```
<button onClick={this.handleAddSkill}>add skill</button>
```

ตัวอย่างการแก้ไขสมาชิกใน array ด้วย setState



ดูได้ใน `setstate-array-object.html`

```
editSkill = (id) => {  
  this.setState({  
    skills: this.state.skills.map(skill => (  
      (skill.id === id)  
      ? { id, name: 'Swift' }  
      : skill  
    ))  
  });  
}  
  
<span onClick={() => this.editSkill(1)}> - edit</span>
```

ตัวอย่างการลบสมาชิกใน array ด้วย setState



ดูได้ใน [setstate-array-object.html](#)

```
deleteSkill = (id) => {  
  this.setState({  
    skills: this.state.skills.filter(skill => skill.id !== id)  
  });  
}
```

```
<span onClick={() => this.deleteSkill(2)}> - x</span>
```

Props + State + Event





ดูได้ใน props-state-event.html

```
class App extends React.Component {  
  state = {  
    nextId: 1,  
    users: []  
  };  
  handleSubmit = (values) => {  
    this.setState({  
      nextId: this.state.nextId + 1,  
      users: [ ...this.state.users, {  
        id: this.state.nextId,  
        name: values  
      }  
    ]  
  });  
}
```


ดูได้ใน props-state-event.html

```
render() {  
  return (  
    <div>  
      <UserList users={this.state.users}/>  
      <UserForm onSubmit={this.handleSubmit}/>  
    </div>  
  );  
}
```



ดูได้ใน props-state-event.html

```
const UserList = (props) => {  
  return (  
    <ul>  
      {props.users.map(u => <li key={u.id}>{`name: ${u.name}`}</li>)}  
    </ul>  
  )  
}
```



ดูได้ใน props-state-event.html

```
class UserForm extends React.Component {  
  state = {  
    name: ''  
  };  
  handleSubmit = (e) => {  
    e.preventDefault();  
    this.props.onSubmit(this.state.name);  
    this.setState({ name: '' });  
  }  
  handleChange = (e) => {  
    this.setState({ name: e.target.value });  
  }  
}
```

```
render() {  
  return (  
    <form onSubmit={this.handleSubmit}>  
      <input  
        type="text"  
        name="name"  
        value={this.state.name}  
        onChange={this.handleChange}  
      /><br />  
      <button type="submit">add user</button>  
    </form>  
  );  
}
```

ตัวอย่างที่ไม่ควรทำ



เหตุผลเพราะว่าห้ามแก้ไข state โดยตรง

```
editSkill = () => {  
    this.state.skills[0].name = 'C++';  
}
```

ตัวอย่างที่ไม่ควรทำ



เหตุผลเพราะว่า splice ไม่ใช่ pure function

```
deleteSkill = () => {  
  this.state.skills.splice(1, 1);  
}
```

การ setState แบบ Function (use case 1)

ในกรณีที่เรา setState state ซ้ำกันใน Function เดียวกันมากกว่าหนึ่งครั้ง

```
incrementCount = () => {  
  this.setState({count : this.state.count + 1})  
  this.setState({count : this.state.count + 1})  
}
```

setState ทำงานแบบ Async ในท้ายที่สุดไม่ว่าเราจะ setState ที่ครั้งก็ตาม setState จะรวมการเปลี่ยนแปลงทุกอย่างในรอบนั้นให้เหลือครั้งเดียว state ที่ซ้ำกันก็จะถูกทับด้วย state ใหม่เสมอ เราแก้ไขปัญหานี้ด้วยการใช้ setState แบบ Function ได้

ms setState &&& Function (use case 1)



```
incrementCount = () => {  
  this.setState((prevState, props) => ({  
    count: prevState.count + 1  
  }));  
  this.setState((prevState, props) => ({  
    count: prevState.count + 1  
  }));  
}
```

การ setState แบบ Function (use case 2)

ในกรณีที่เรา setState ด้วยการอ้างอิงจาก state หรือ props เดิม เช่น

```
this.setState({ counter: this.state.counter + this.props.increment });
```

เนื่องจาก this.props และ this.state มีโอกาสจะถูก update แบบ async ทำให้เราไม่ควรใช้ค่าพวกนี้มาคำนวณ state ต่อไป ดังนั้นเราจึงควรใช้ setState แบบ Function เพื่อแก้ปัญหานี้



ms setState &&& Function (use case 2)



```
this.setState((prevState, props) => ({  
  counter: prevState.counter + props.increment  
}));
```

Callback ของ setState

ด้วยความที่ setState ทำงานแบบ async setState จึงมี callback เพื่อให้รู้ว่า setState เสร็จเรียบร้อยแล้ว



Callback แทน setState



```
this.setState({ count : 5 }, () =>  
console.log('setState was finished'));
```

Lab 4: State

จาก lab 3

- ทดลองหัดใช้ state
- ทดลองใช้ state กับ input, form, array, object
- ส่ง state เข้าไปใน props ให้ children component



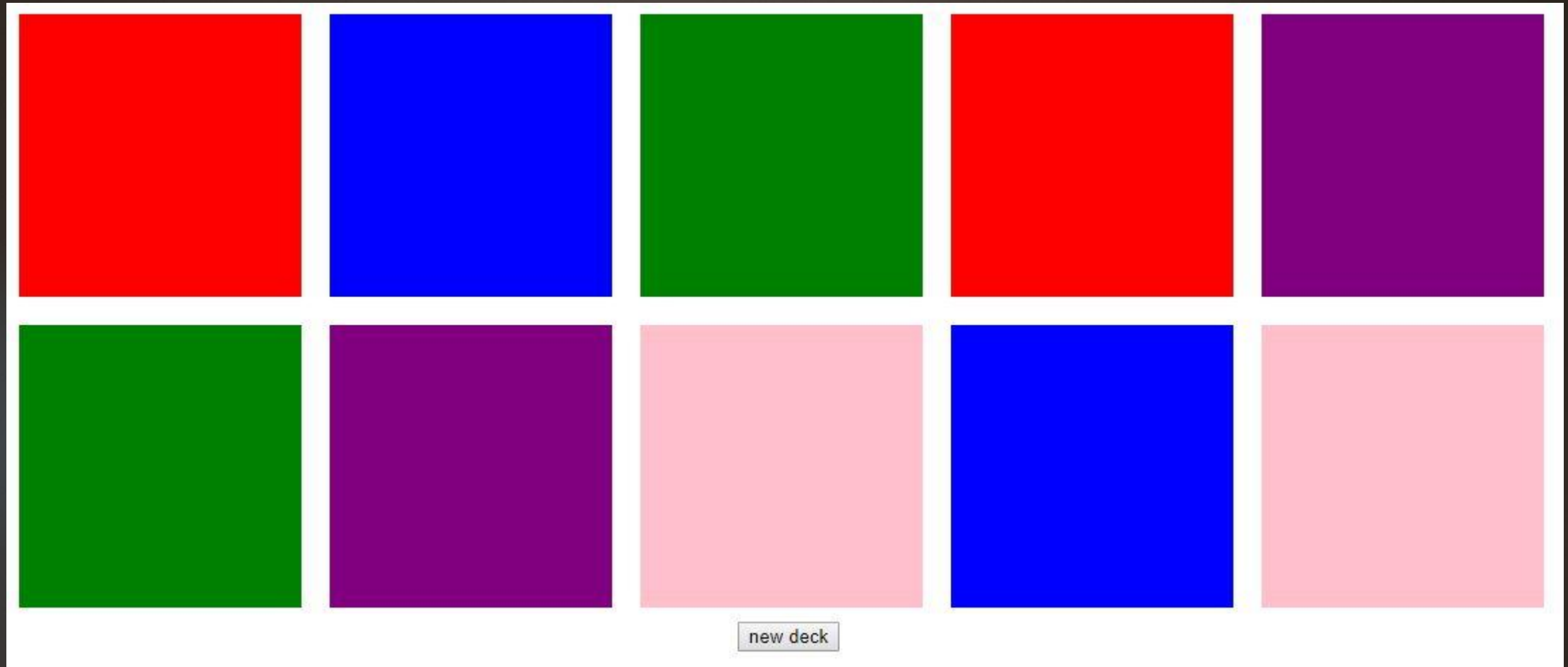
การบ้าน #1 - สร้าง Card Deck

- ใช้ Create React App ในการสร้าง Project
- สร้าง App Card Deck ขึ้นมาโดยให้มี **Feature** ดังนี้
- ใช้ **state** และ **props** ในการทำ
- สร้างการ์ดจาก ['red', 'blue', 'green', 'purple', 'pink'] สีละ 2 การ์ด
- โดยการ์ดที่สร้างขึ้นมาจะต้องถูก **shuffle** ก่อนสร้าง
- มีปุ่ม **new deck** เอาไว้ shuffle การ์ดใหม่
- เมื่อคลิกที่การ์ดจะมี **popup** บอกว่า การ์ดสีอะไร

shuffle function: <https://30secondsofcode.org/#shuffle>



การบ้าน #1 - สร้าง Card Deck

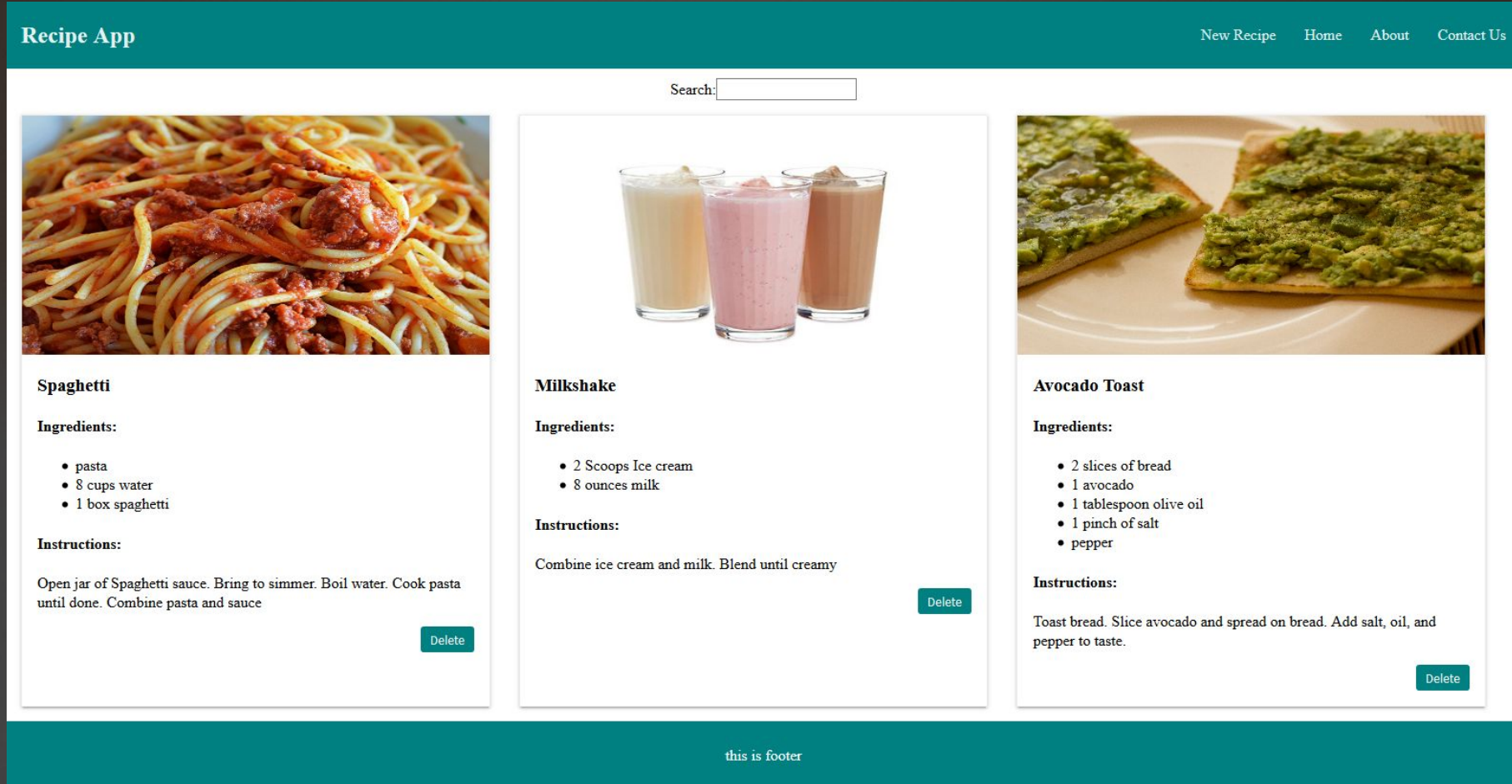


การบ้าน #2 - Recipe ภาคต่อ

- จากการบ้าน Recipe คราวที่แล้ว ให้เพิ่ม **Feature** ดังนี้
- ใช้ **state** และ **props** ในการทำ
- มี ปุ่ม **Delete recipe** และต้องลบได้จริง
- กล่อง **search** ต้องพิมพ์แล้ว search ได้เลย (**case-insensitive**) โดยค้นหาจาก title



การบ้าน #2 - Recipe ภาคสอง



ดูภาพขนาดใหญ่ได้ที่ <https://ibb.co/eGypn>

การบ้าน #3 - Recipe ภาคสาม

- จากการบ้าน Recipe คราวที่แล้ว ให้เพิ่ม **Feature** ดังนี้
- มี **Form** เอาไว้เพิ่ม **recipe** (จะแสดงเมื่อกดเมนู new recipe)
- มีปุ่ม **X** สามารถปิด **Form** ได้
- สามารถเพิ่ม **ingredient** ได้มากกว่า 1
- กด add แล้ว ต้องมี **Recipe** เพิ่ม และ **Form** ปิดตัวอัตโนมัติและต้อง **clear form** ก่อนปิด



การบ้าน #3 - Recipe ภาควิชา

Recipe App

New Recipe Home About Contact Us

Search:

Add new recipe X

Title:




Instructions:

Ingredients:

+

Image:

Add recipe



ดูภาพขนาดใหญ่ได้ที่ <https://ibb.co/esXF27>

การบ้าน #4 - Card Matching Game

- ข้อนี้เป็น Optional (ทำหรือไม่ทำก็ได้)
- จากการบ้านวันนี้ข้อที่ 1
- นำมาสร้างเป็นเกมส์จับคู่การ์ด โดยมี **Feature** ดังนี้
- การ์ดตอนเริ่มให้ปิดการ์ดด้วยสีเทาทั้งหมด
- คลิกที่การ์ดแล้ว **การ์ดหงายแสดงสี**
- หากคลิกที่การ์ดอื่น แล้วสีเหมือนกัน ให้หงายการ์ดทั้งสองใบค้างไว้
- หากไม่เหมือนกันให้หน่วงเวลา 1 วินาที ห้ามคลิก และปิดการ์ด
- เปลี่ยนจากปุ่ม new deck เป็น **new game** กดแล้วต้อง reset game **สุมการ์ดใหม่ให้**



สำหรับหาความรู้เพิ่มเติม

Using a function in `setState` instead of an object

<https://medium.com/@wisecobbler/using-a-function-in-setstate-instead-of-an-object-1f5cfd6e55d1>

State Updates May Be Asynchronous

<https://reactjs.org/docs/state-and-lifecycle.html#state-updates-may-be-asynchronous>

