React Props

Props

```
function Avatar() {
  return (
    <img
     className="avatar"
     src="https://i.imgur.com/1bX5QH6.jpg"
     alt="Lin Lanying"
     width={100}
     height={100}
    />
  );
}
```

01 React components use props to communicate with each other.

- like HTML attributes
- Child-Parent relation
- but can pass any Javascript data Ex. array, object, function

React Props

```
export default function App() {
  return (
    <div className="App">
      <RenderSomething text="Hello, world!" />
    </div>
  );
}

function RenderSomething(props) {
  return (
   <div>
     {props.text}
   </div>
  );
}
```

02

Every parent component can pass some information to its children components by giving them props.

App - Parent Component
RenderSomething - Children Component

NOTE: prop is an object!

- Props serve the same role as arguments serve for function
- React component only accept 1 argument which is "props"!!

React Props

```
export default function App() {
  return (
    <div className="App">
      <RenderSomething
        text="Hello, world!"
        text2="Another text"
        whatEverNameYouLike={999}
      />
    </div>
  );
}

function RenderSomething({ text, text2, whatEverNameYouLike }) {
  return (
   <div>
     <p>{text}</p>
     <p>{text2}</p>
     <p>{whatEverNameYouLike}</p>
   </div>
  );
}
```

03 Another approach using **Object Destructuring**

- We use this approach if we've already know what the values would be pass into the component.
- You can name the passed down props whatever you like

More detail on Destructuring

```
// Destructuring in Array
const array = ["Apple", "Orange", "Pineapple", "Watermelon"];

const [x, y] = array;
console.log(x, y); // Apple Orange

// Or with rest syntax
const [x, y, ...rest] = array;
console.log(rest); // ["Pineapple", "Watermelon"]
```

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to **unpack** values from **arrays**, or properties from **objects**, into distinct variables.

Destructuring Syntax
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

More detail on Destructuring

```
// Destructuring in Object literal
const person = {
 firstName: "John",
 lastName: "Doe",
 age: 20,
 job: "programmer",
};

const { firstName, lastName } = person;
console.log(firstName, lastName); // John Doe
```

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to **unpack** values from **arrays**, or properties from **objects**, into distinct variables.

Destructuring Syntax
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

React Props

```
export default function App() {
  return (
    <div className="App">
      <Greeting
        name="John"
        age={20}
      />
    </div>
  );
}

function Greeting({ name, age = 16}) {
  return (
   <div>
     <p>My name is {name}!</p>
     <p>I'm {age} years old.</p>
   </div>
  );
}
```

04 Set **default value** to props

- when the the age props is not provided, it fallback value is 16
- It uses the same syntax from destructuring object
- You can set default value during destructuring an object

React Props

```
export default function Parent() {
  return (
    <div className="App">
      <Child
        name="John"
        age={20}
      >
    </div>
  );
}

function Child(props) {
  return (
    <InnerChild {...props} />
  );
}
```

05 Forwarding Props with JSX spread syntax

Spread Syntax

```
// Spreading an object
const newPerson = {...person, job: "none"};

// Spreading an array
const newArray = [...array, "Mango"];
```

Spread syntax expand elements into places where arguments are wanted

Spread Syntax
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

React Props

```
export default function Parent() {
  return (
    <div className="App">
      <Child>
        <ul>
          <li>Apple</li>
          <li>Orange</li>
        </ul>
      </Child>
    </div>
  );
}

function Child({ children }) {
  return (
   <div>
     <h2>This is a list</h2>
     {children}
   </div>
  );
}
```

06 Passing **JSX** as children

The child component consider the ul tag as its children

React Props

```
export default function Parent() {
  const [count, setCount] = useState(0);
  return (
    <div className="App">
      <Child count={count} setCount={setCount} />
    </div>
  );
}

function Child({ count, setCount }) {

  function handleClick() {
    setCount(count + 1);
  }

  return (
   <div>
     <button onClick={handleClick}>{count}</button>
   </div>
  );
}
```

07 Props are **immutable.**

- You should not edit the props value. It's anti-pattern. (bad programming practice)
- But it's not static.
- When the data from props need to change the child component will ask parent component to change the state and pass the props down again. (Lifting State Up!)

Event Handling

Event Handling in React

```
// A button component without event handler
export default function Button() {
  return (
    <button>
      I don't do anything
    </button>
  );
}
```

01 React lets you add **event handlers** to your JSX. Event handlers are your own functions that will be triggered in response to interactions like clicking, hovering, focusing form inputs, and so on.

Event Handling in React

```
// With onClick event handler
export default function Button() {

  function handleClick() {
    alert('You clicked the button!');
  }

  return (
    <button onClick={handleClick}>
      Click me!
    </button>
  );
}
```

02 Adding event handlers

1. Define a function
2. Pass the function as prop

By convention, it is common to name **event handlers** as **handle** followed by the ***event name***

Event Handling in React

```
// Incorrect way to pass event handler function
export default function Button() {

  function handleClick() {
    alert('You clicked the button!');
  }

  // DO NOT DO THIS
  return (
    <button onClick={handleClick()}>
      Click me!
    </button>
  );
}
```

Functions passed to event handlers **must be passed**, not called.

`onClick={handleClick()}`

fired the function immediately during the rendering, without any clicks.

Because Javascript inside JSX { and } executes immediately.

Event Handling in React

```
function AlertButton({ message, children }) {

  function handleClick() {
    alert(message);
  }

  return (
    <button onClick={handleClick}>{children}</button>
  );
}

function Toolbar() {
  return (
    <AlertButton message="Hello!">Click me!</AlertButton>
  );
}
```

03 Reading props inside event handlers

Because event handlers are declared inside the component therefore they have access to the component's props.

Event Handling in React

```
function AlertButton({ onClick, children }) {
  return (
    <button onClick={onClick}>{children}</button>
  );
}

function Toolbar() {

  function alertUser() {
    alert("Alert! Alert!");
  }

  function playMusic() {
    alert("Playing music!");
  }

  return (
    <>
      <AlertButton onClick={alertUser}>Click me!</AlertButton>
      <AlertButton onClick={playMusic}>Play music</AlertButton>
    </>
  );
}
```

04 Passing event handlers as props

Parent can define the behavior of event handlers and pass down to its children.

In this way, Children components might or might have the same behavior of event handlers based on the props passed down by its parent.

Event Handling in React

```
function AlertButton({ onSmash, children }) {
  return (
    <button onClick={onSmash}>{children}</button>
  );
}

function Toolbar() {

  function alertUser() {
    alert("Alert! Alert!");
  }

  function playMusic() {
    alert("Playing music!");
  }

  return (
    <>
      <AlertButton onSmash={alertUser}>Click me!</AlertButton>
      <AlertButton onSmash={playMusic}>Play music</AlertButton>
    </>
  );
}
```

05 Naming event handlers props

You can name event handler props any way that you like.

In this example,
<button onClick={onSmash}>
shows that the browser <button> still needs a prop called onClick, but the prop name received by your custom AlertButton component is up to you!

NOTE: **Built-in components** like <button> and <div> only support browser event names

Have a look at
https://beta.reactjs.org/reference/react-dom/components/common#common-props

for more event names that you can use

Event Handling in React

```
function App() {

  function handleClick1() {
    alert("Outer Alert");
  }

  function handleClick2() {
    alert("Inner Alert");
  }

  return (
    <div onClick={handleClick1}>
      <button onClick={handleClick2}>Click me!</button>
    </div>
  );
}
```

06 Stop propagation

In this code, If we clicked the button it will fired alert box "Inner Alert" and then "Outer Alert"

This is called propagation, the event got *bubbling* up

Because the button is nested inside the <div> therefore clicking button also triggered the outer click handler

Event Handling in React

```
function App() {

  function handleClick1() {
    alert("Outer Alert");
  }

  function handleClick2() {
    alert("Inner Alert");
  }

  return (
    <div onClick={handleClick1}>
      <button onClick={(e) => {
        handleClick2();
        e.stopPropagation();
      }}>
        Click me!
      </button>
    </div>
  );
}
```

To stop propagation use e.stopPropagation()
(e stand for event)

Event Handling in React
Controlled Form

```
Normal form in HTML
<form>
  <label>
    Name:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Submit"/>
</form>
```

HTML form elements work a bit differently from other DOM elements in React, because form elements naturally keep some internal state.

We can access data and control the submit event by technique called "**Controlled Form**" or "**Controlled Components**".

Event Handling in React
Controlled Form

```
function CustomForm() {
  const [name, setName] = useState("");

  return (
    <form>
      <label htmlFor="nameinput">
        Name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
    </form>
  );
}
```

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input.

But in React mutable state is kept in the state property of components.

We can combine the two by making the React state be the “single source of truth”.

Event Handling in React
Controlled Form

```
function App() {

  function handleSubmit(e) {
    alert("form submitted");
    e.preventDefault();
  }

  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
}
```

e.preventDefault(); to prevent default behavior

Some browser events have default behavior associated with them. For example, a <form> submit event, which happens when a button inside of it is clicked, will reload the whole page by default.

To stop this add **e.preventDefault();** when hit submit