Props & State

CS568 – Web Application Development I

Assistant Professor Umur Inan
Computer Science Department
Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- Props
- State
- setState()
- Passing data to another component via props
- Passing method references between components
- setState with updater function
- Props with Constructor

Props

- Props are arguments passed to components
- Can be used to pass data from the parent component to child components. (One way, parent -> child)
- They are passed via HTML attributes
- Props are read-only!

Props in Functional Component

advancedHello.js

```
import React from 'react'
const advancedHello = (props) => {
   return (<h1>Hello {props.name} </h1>);
}
export default advancedHello;
```

App.js

```
function App() {
 return (
   <div className="App">
    <AdvancedHello name='Umur'></AdvancedHello>
    <AdvancedHello name='Aynur'></AdvancedHello>
   </div>
 );
}
```

Props in Class-Based Component

advancedGreeting.js

```
import React from 'react';

class AdvancedGreeting extends React.Component
{
   render() {
       return <h1>Hello {this.props.name}</h1>
   };
}

export default AdvancedGreeting;
```

App.js

```
function App() {
 return (
   <div className="App">
     <AdvancedGreeting name='Erol'></AdvancedGreeting>
     <AdvancedGreeting name='Ayse'></AdvancedGreeting>
   </div>
 );
}
```

Children Prop

Children is a special property of React components which contains any child elements defined within the component.

```
class AdvancedGreeting extends React.Component {
    render() {
        return <h1>Hello {this.props.name} {this.props.children}</h1>
    };
}

export default AdvancedGreeting;
```

Children Prop

```
function App() {
 return (
   <div className="App">
     <AdvancedGreeting name='Erol'>How are you?</AdvancedGreeting>
     <AdvancedGreeting name='Ayse'>We all miss you</AdvancedGreeting>
   </div>
 );
}
```

State

State is used to change the component. Changes to state trigger an UI update.

Only **class-based** components can define and use state. State is a property of the class component.

> We can still use state in functional components using something called hooks that is more flexible. We will learn hooks in the next course.

The state object is where you store property values that belongs to the component. You can store as many properties as needed.

Changing State

- To change a value in the state object, always use the **this.setState()** method!
- this.setState will ensure that the component knows it’s been updated and then calls the render() method.
- Changing the state means that React triggers an update when we call the setState function. This doesn't only mean the component's render function will be called, **but also that all its subsequent child components will re-render, regardless of whether their props have changed or not.**

props vs state

- props get passed to the component whereas state is managed within the component
- props is similar to function parameters whereas state is similar to variables declared within a function.

Increment Counter Example

```
class App extends React.Component {
 state={counter:1};
 incrementCounter = ()=>{
   this.setState({counter:this.state.counter +1});
 }
 render() {
   return (
     <div className="App">
       <input type='label' value={this.state.counter} />
       <button onClick={this.incrementCounter}>Increment
Counter</button>
…
```

setState

- Do not change the state directly.
- We **have to use setState** to manipulate the state.
- Merges the old state with the new one.
- Recap incrementCounter:

```
class App extends React.Component {
 state={counter:1};
 incrementCounter = ()=>{
   this.setState({counter: this.state.counter +1});
 }
}
```

Passing Data to Another Component via Props

```
state={students:[
    {name:'Alice', age:20},
    {name:'Bob', age:19}
]};
render() {
  return (
    <div className="App">
      <Student name={this.state.students[0].name}
               age={this.state.students[0].age}>
      </Student>
      <Student name={this.state.students[1].name}
               age={this.state.students[1].age}>
      </Student>
    </div>);
}
```

Merging State – Make older example

- Put another attribute to the state.
- Let’s call this attribute ‘dummy’
- We manipulate the state with useState method.
- ‘dummy’ attribute will be in the new state
- That is very important !!!

Merging State and Make Older

```
state = {
   students: [
     { name: 'Alice', age: 20 },
     { name: 'Bob', age: 19 }
   ],
   dummy : 'dummy value'
 };
 makeOlder = () => {
   this.setState({
     students: [
       { name: 'Alice', age: 25 },
       { name: 'Bob', age: 23 }
     ]
   });
   console.log(this.state);
 }
```

```
 render() {
   return (
     <div className="App">
       <Student name={this.state.students[0].name}
         age={this.state.students[0].age}>
       </Student>
       <Student name={this.state.students[1].name}
         age={this.state.students[1].age}>
       </Student>
       <button onClick={this.makeOlder}>Make
Older</button>
     </div>
   );
 }
```

Passing Method References Between Components

- No difference between passing other properties like name and passing method reference.
- In makeOlder example, we change the age when clicking on the button.
- But, what if we want to change the age when clicking on the name or age?

Passing Method References Between Components

```
render() { // App.js
   return (
     <div className="App">
       <Student
name={this.state.students[0].name}
         age={this.state.students[0].age}
         myClickHandler={this.makeOlder}>
       </Student>
       <Student
name={this.state.students[1].name}
         age={this.state.students[1].age}
         myClickHandler={this.makeOlder}>
       </Student>
       <button
onClick={this.makeOlder}>Make
Older</button>
     </div>
   );
 }
```

```
//student.js
class Student extends React.Component {
   render() {
       return (
           <div>
               <p onClick={this.props.myClickHandler}>I
am {this.props.name}.</p>
               <p>I am {this.props.age} years old.</p>
           </div>
       )
   };
}
```

setState with updater function

- Calls to setState are asynchronous. It improves performance.
- Don’t rely on this.state() to reflect the new value immediately after calling setState. Pass an updater function instead of an object if you need to compute values based on the current state.

```
this.setState((currentState) => {
    return {count: currentState.count + 1}
});
```

Props with Constructor

The props **should always be passed** to the constructor and also to the React.Component by using super **if the component has a constructor.**

```
import React from 'react';

class AdvancedGreeting extends React.Component {

    constructor(props){
        super(props);
    }

    render() {
        return <h1>Hello {this.props.name}</h1>
    };
}
export default AdvancedGreeting;
```