COMP 3322 Modern Technologies on World Wide Web

2nd semester 2017-2018

React (O2)

Dr. C Wu

Department of Computer Science
The University of Hong Kong

Overview

- React (or React.js, or ReactJS) is a JavaScript library for building composable user interfaces (UIs)
  - it corresponds to View in the Model-View-Controller (MVC) pattern, and can be used in combination with other JavaScript libraries or frameworks
  - it encourages the creation of reusable UI components, which present data that may change over time
  - it abstracts away the DOM from the programmers, offering a simpler programming model and better performance (speed, simplicity, and scalability)
- First deployed on Facebook's newsfeed in 2011; on Instagram.com in 2012; open-sourced in 2013
- Currently one of the most popular JavaScript libraries and has a large community behind it (Facebook, Instagram, individual developers and corporations)

Hello world example

load React and ReactDOM modules:

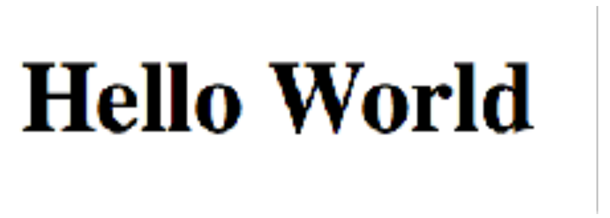
The ES6 **import** statement is used to import bindings exported by another module

an ES6 **const** declaration

```
import React from 'react';
import ReactDOM from 'react-dom';

const element = (
  <h1>
    Hello World
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

a JSX expression

render the <h1> element within an element of id 'root'



Hello World

ES6

- ECMAScript 6, also known as ECMAScript 2015, is the sixth major release of the ECMAScript language specification
  - **ECMAScript** is the “proper” name for the language commonly referred to as **JavaScript**
- New features: arrow function, class, template strings, destructuring, let + const, etc. (https://github.com/lukehoban/es6features)
- Babel: the compiler for ES6

Example new features

- An **arrow** function expression is a function shorthand using the **=>** syntax
  - it has a shorter syntax than a function expression

(1) `(param1, param2, …, paramN) => {statements}`

(2) `(param1, param2, …, paramN) => expression`

equivalent to

`(param1, param2, …, paramN) => {return expression;}`

() optional when there is only one parameter;
() is used when the function has no parameter

- example

```
var x = function (a, b) {return a + b}
```

equivalent to

```
var x = (a, b) => {return a + b;}
```

or

```
var x = (a, b) => a + b
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Example new features

- **class**
  - previous JavaScript does not have class, but object (which can inherit from another object)
  - ES6 introduce the class syntax

```
class App extends React.Component {
  render() {
    return (
      <h1>
        Hello World
      </h1>
    );
  }
}
```

The render() method is required in a subclass extending React.Component

return a React element created via JSX

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

Example new features

- The **let** statement declares a block-scoped variable, optionally initializing it to a value
  - let allows one to declare variables that are limited in scope to the block, statement, or expression on which it is used; var defines a variable globally, or locally to an entire function regardless of block scope

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2;  // same variable!
    console.log(x);  // 2
  }
  console.log(x);  // 2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2;  // different variable
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

Example new features

- The **const** statement declares a block-scoped variable (like the **let** statement) and initializes it to a value, while the variable cannot be redeclared

```
const age = 5;
```

```
const element = (
  <h1>
    Hello World
  </h1>
);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const

JSX

- **JSX** is a syntax extension to JavaScript, to produce React elements
  - it is recommended to be used with React
  - It is faster because optimization is performed while compiling it to JavaScript
  - It is also type-safe and most of the errors can be caught during compilation
  - It makes it easier and faster to write templates (for producing HTML elements)

https://reactjs.org/docs/introducing-jsx.html
https://reactjs.org/docs/jsx-in-depth.html

JSX expression

Examples

```
const element = <h1>Hello World</h1>;
```

```
const element = (
  <h1>
    Hello World
  </h1>
);
```

JSX expression

when splitting into multiple rows, recommend wrapping it in ( ) to avoid automatic semicolon insertion

Embed any JavaScript expression in JSX by wrapping it in curly braces

```
const element = (
  <h1>
    Hello, {formatName(user)}
  </h1>
);
```

If a tag is empty, we may close it immediately with />, like in XML:

```
const element = <img src={user.avatarUrl} />;
```

JSX expression

If we want to return more than one elements in one statement, we need to wrap the elements within one container element:

```
class App extends React.Component {
   render() {
      return (
         <div>
            <h1>Header 1</h1>
            <h2>Header 2</h2>
            <p>This is the content</p>
         </div>
      );
   }
}
```

We can use JSX expression inside **if** statements and **for** loops, assign it to variables, accept it as arguments, and return it from functions

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}</h1>;
  }
  return <h1>Hello, Stranger</h1>;
}
```

React elements

- Descriptions of what you want to see on the screen

```
const element = (
  <h1 className="greeting'>
    Hello World
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

after compilation, JSX expression is compiled into a React.createElement() function call

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello World!'
);
```

React elements are immutable: once you create an element, you cannot change its children or attributes.

React components

- Conceptually, **components** are like JavaScript functions
  - They accept inputs (called "props") and return React elements describing what should appear on the screen
- **Components** allow us to split the UI into independent, reusable pieces, and design each piece in isolation
- *Functional components*: write a JavaScript function to define a component

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- *Class components*: use an ES6 class to define a component

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

Examples

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

<=>

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

ReactDOM.render(
  <Welcome name="Sara" />,
  document.getElementById('root')
);
```

<=>

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

ReactDOM.render(
  <Welcome name="Sara" />,
  document.getElementById('root')
);
```

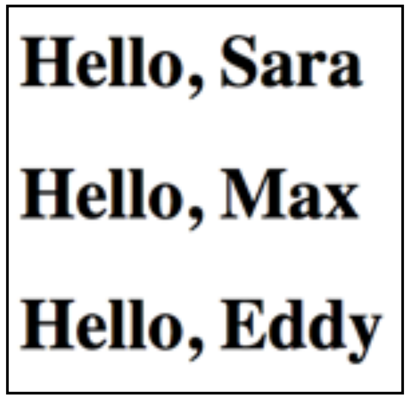
When React sees an element representing a user-defined component, it passes JSX attributes to this component in a single props object

Always start a component name with a capital letter

React components

- In a React app, a button, a form, a dialog, a screen are commonly expressed as components
- Components can use other components in their output:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Max" />
      <Welcome name="Eddy" />
    </div>
  );
}

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



```
Hello, Sara

Hello, Max

Hello, Eddy
```

Props

- Props ("properties") are read-only: a component must never modify its props
  - class components should always call super() in its constructor (if it has a constructor); if using this.props in the constructor, need to call the constructor and super() with props

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

```
class Welcome extends React.Component {
  constructor(props) {
     super(props);
     console.log(this.props.name);
   }

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

States

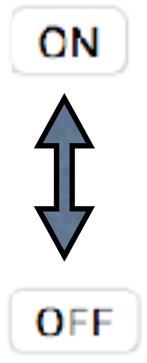
- States are similar to props, but they are private and fully controlled by the component where they are defined
  - only class component can have states

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
          isToggleOn: true
    };

    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState({
      isToggleOn: !this.state.isToggleOn
    });
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

there can be many key-value pairs in the state object;

the only place where you can initialise the states is the constructor;

use this.setState() in later code to modify state



Event handling

- Handling events on React elements is very similar to handling events on regular DOM elements, except the following:
  - React events are named using camelCase, rather than lowercase
  - pass a function as the event handler in JSX, rather than a string

With plain HTML and JavaScript:

```
<button onclick="handleClick()">
  Click me
</button>
```

In React:

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

  - you cannot return false to prevent default behavior in React; you must call preventDefault explicitly

With plain HTML and JavaScript:

```
<a href="#" onclick="return false">
  Click me
</a>
```

(to prevent the default link behavior of opening a new page)

In React:

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

Event handling

- For events on elements returned by a component, a common pattern is to have the event handlers as methods in the component

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
          isToggleOn: true
    };

    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState({
      isToggleOn: !this.state.isToggleOn
    });
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

The binding is necessary to make '**this**' work in handleClick()

Event handling

equivalent to:

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
           isToggleOn: true
    };

    ~~this.handleClick = this.handleClick.bind(this~~);
  }

  handleClick() {
    this.setState({
      isToggleOn: !this.state.isToggleOn
    });
  }

  render() {
    return (
      <button onClick={this.handleClick.bind(this)}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

Event handling

also equivalent to:

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
          isToggleOn: true
    };

    ~~this.handleClick = this.handleClick.bind(this~~);
  }

  handleClick() {
    this.setState({
      isToggleOn: !this.state.isToggleOn
    });
  }

  render() {
    return (
      <button onClick={(e)=>this.handleClick(e)}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

Event handling

- Pass arguments to an event handler:

```
<button onClick={(e) => this.deleteRow(id, e)}>
    Delete Row
</button>
```

⇕

```
<button onClick={this.deleteRow.bind(this, id)}>
    Delete Row
</button>
```

In both cases, the e argument (representing the React event triggered) will be passed as a second argument after id: with an arrow function, we have to pass it explicitly; with bind, it is automatically forwarded into the function call

Conditional rendering

- Use JavaScript operators such as if or the conditional operator (condition ? expr1 : expr2) to conditionally create elements

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}

function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

```
function LoginButton(props) {
  return (
    <button onClick={props.onClick}>
      Login
    </button>
  );
}

function LogoutButton(props) {
  return (
    <button onClick={props.onClick}>
      Logout
    </button>
  );
}
```

```
class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.handleLoginClick = this.handleLoginClick.bind(this);
    this.handleLogoutClick = this.handleLogoutClick.bind(this);
    this.state = {isLoggedIn: false};
  }

  handleLoginClick() {
    this.setState({isLoggedIn: true});
  }

  handleLogoutClick() {
    this.setState({isLoggedIn: false});
  }

  render() {
    let button = null;
    if (this.state.isLoggedIn) {
      button = <LogoutButton onClick={this.handleLogoutClick} />;
    } else {
      button = <LoginButton onClick={this.handleLoginClick} />;
    }

    return (
      <div>
        {button}
      </div>
    );
  }
}

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

Use variable to store element, to conditionally render a part of the component

```
function LoginButton(props) {
  return (
    <button onClick={props.onClick}>
      Login
    </button>
  );
}

function LogoutButton(props) {
  return (
    <button onClick={props.onClick}>
      Logout
    </button>
  );
}
```

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      {isLoggedIn ? (
        <LogoutButton
onClick={this.handleLogoutClick} />
      ) : (
        <LoginButton
onClick={this.handleLoginClick} />
      )}
    </div>
  );
}
```

<=>

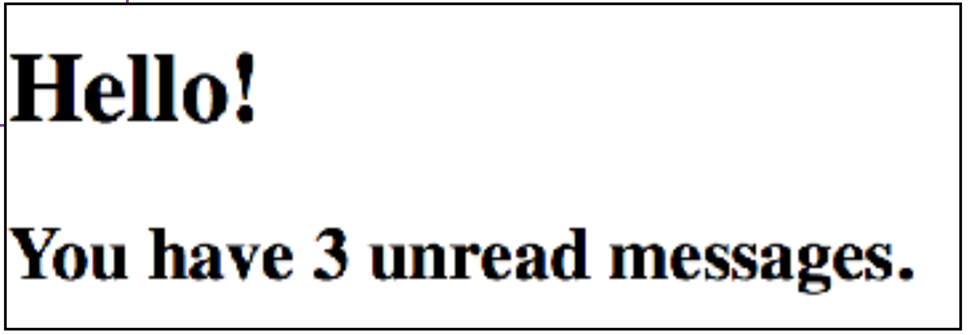
```
class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.handleLoginClick = this.handleLoginClick.bind(this);
    this.handleLogoutClick = this.handleLogoutClick.bind(this);
    this.state = {isLoggedIn: false};
  }

  handleLoginClick() {
    this.setState({isLoggedIn: true});
  }

  handleLogoutClick() {
    this.setState({isLoggedIn: false});
  }

  render() {
    let button = null;
    if (this.state.isLoggedIn) {
      button = <LogoutButton onClick={this.handleLogoutClick} />;
    } else {
      button = <LoginButton onClick={this.handleLoginClick} />;
    }

    return (
      <div>
        {button}
      </div>
    );
  }
}

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

Conditional rendering

Another conditional rendering example:

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {unreadMessages.length > 0 &&
        <h2>
          You have {unreadMessages.length} unread messages.
        </h2>
      }
    </div>
  );
}

const messages = ['Re: React', 'Re: Angular', 'Re:Express'];
ReactDOM.render(
  <Mailbox unreadMessages={messages} />,
  document.getElementById('root')
);
```

evaluate to the right if unreadMessages.length > 0 is true



Hello!

You have 3 unread messages.

Lists and Keys

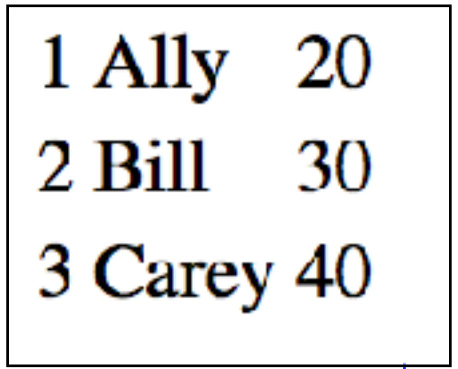
- Loop through a list (array):

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map((number) => number * 2);
```

The **map()** method creates a new array with the results of calling a provided function on every element in the calling array.

Keys can be given to elements inside a list (array) to identify these elements

```
class TableRow extends React.Component {
  render() {
    return (
      <tr>
        <td>{this.props.data.id}</td>
        <td>{this.props.data.name}</td>
        <td>{this.props.data.age}</td>
      </tr>
    );
  }
}

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      data:
      [
        {
          "id":1,
          "name":"Ally",
          "age":"20"
        },
        {
          "id":2,
          "name":"Bill",
          "age":"30"
        },
        {
          "id":3,
          "name":"Carey",
          "age":"40"
        }
      ]
    }
  }
  render() {
    return (
      <div>
        <table>
          <tbody>
            {this.state.data.map((person, index) =>
<TableRow key = {index}  data = {person} />)}
          </tbody>
        </table>
      </div>
    );
  }
}

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

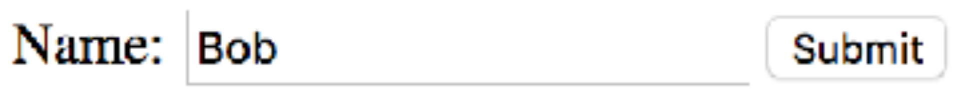


```
1 Ally  20
2 Bill  30
3 Carey 40
```

Controlled component

- In a controlled component, a handler function is associated with every state mutation
  - For example, in a form component, the handler functions modify or validate user input

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: 'Bob'};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        Name: <input type="text" value={this.state.value} onChange={this.handleChange} />
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
ReactDOM.render(
  <NameForm />,
  document.getElementById('root')
);
```



default behavior is to browse to a new page when the user submits the form

the React component that renders a form also controls what happens in that form on subsequent user input

More about React

- There are many online examples and discussions on how React can work with node.js/express.js framework, as the front-end to achieve a complete Web app
  - AJAX calls: use a third-party HTTP library (jQuery, axios,superagent, fetch, etc.), e.g.,

```
import axios from 'axios';
```

```
axios.get(‘http://www.example.com/${this.props.subreddit}.json')
      .then(res => {
        const posts = res.data.children.map(obj => obj.data);
        this.setState({ posts });
      });
```

  - example of using React with node.js/express.js:

    https://hackernoon.com/how-to-combine-a-nodejs-back-end-with-a-reactjs-front-end-app-ea9b24715032

  - The MERN stack: MongoDB, Express.js, React, and Node.js (http://mern.io)

- References
  - https://reactjs.org/