React

RUJUAN XING

What is React?

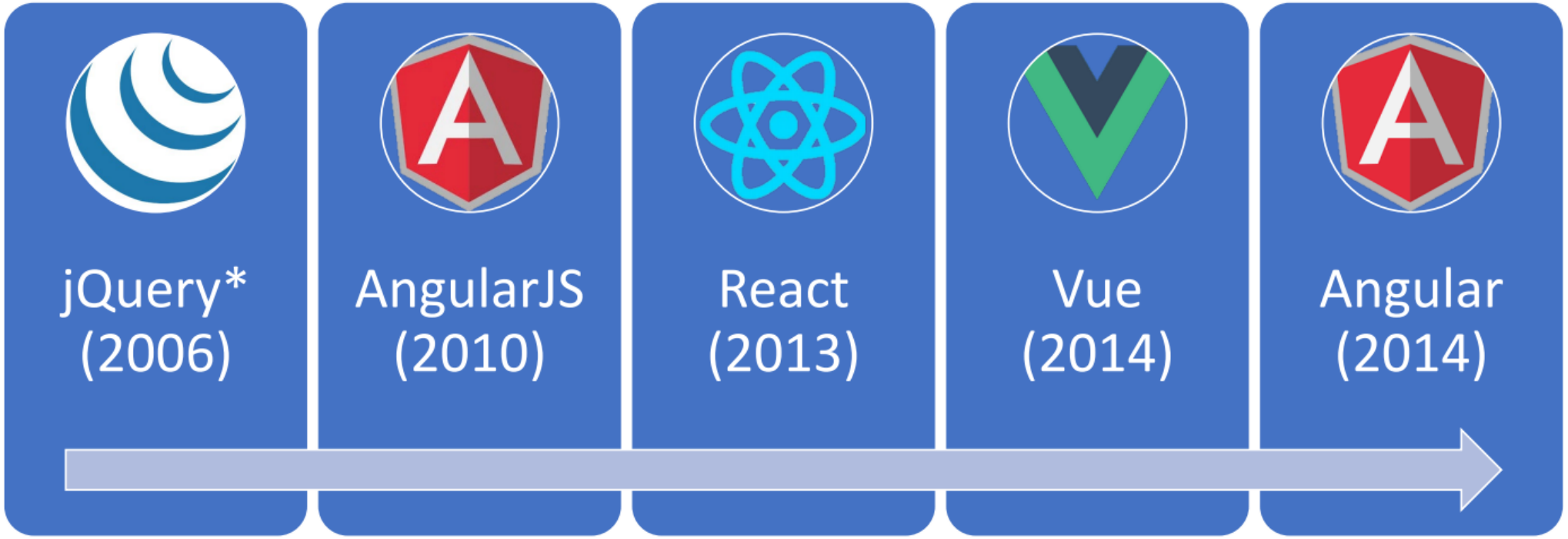
React.js, commonly referred to as React, is library for web and native user interfaces.

Example: If ask you display a collection of user stored in DB in a html page, what would you do?

1. Make request to fetch the user list
2. Process the data
3. Use DOM API to render the page

React is a JavaScript library that renders data into HTML.

Timeline of font-end JavaScript Frameworks/Libraries



Common tasks in front-end development

App state	Data definition, organization, and storage
User actions	Event handlers respond to user actions
Templates	Design and render HTML templates
Routing	Resolve URLs
Data fetching	Interact with server(s) through APIs and AJAX

Why React?

1. Complexity: Working with the DOM API directly often requires writing verbose and complex code, which can be error-prone and challenging to maintain.
2. Inefficiency with Frequent Updates: Continuous updates to the DOM can lead to a phenomenon called "reflow" or "layout thrashing," where the browser recalculates the layout and rendering of the entire page. This can be computationally expensive.
3. Performance Overhead: Direct manipulation of the DOM can be slow and resource-intensive, especially when dealing with a large number of elements. Frequent updates or changes to the DOM can lead to performance bottlenecks and affect the user experience.
4. Lack of Component Reusability: Reusing components across different parts of an application can be challenging.

Features of React

1. **Component-Based Architecture:** React is built around a component-based architecture where the user interface is divided into reusable, self-contained components. Components can be composed to create complex user interfaces, making it easier to manage and maintain code.
2. **Declarative Syntax**: React uses a declarative approach, where developers describe how the UI should look based on the current application state. This simplifies UI development and reduces the risk of bugs related to manual DOM manipulation.
3. **Virtual DOM + Diffing Algorithm**: React uses a virtual representation of the Document Object Model (DOM) called the Virtual DOM. This allows React to efficiently update the actual DOM by only re-rendering components when their state or props change. This results in improved performance and responsiveness.
4. **React Native**: React Native is a framework that extends React to build native mobile applications for iOS and Android using the same component-based approach. This enables code sharing between web and mobile applications.

Hello World

1. react.development.js: is the core library for building user interfaces with a focus on component-based architecture and declarative rendering.
2. react-dom.development.js: specializes in rendering React Components into web browsers.
3. babel.min.js: JSX allows developers to write declarative user interfaces in a more concise and readable manner. Babel can transpile JSX code into regular JavaScript so that it can be executed in web browsers.

The order of scripts matters.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello World</title>
    <!-- React Core Library -->
    <script type="text/javascript" src="./libraries/react.development.js"></script>

    <!-- React DOM - used by react to do DOM operations -->
    <script type="text/javascript" src="./libraries/react-dom.development.js"></script>

    <!-- Used to transfer jsx to js -->
    <script type="text/javascript" src="./libraries/babel.min.js"></script>
</head>
<body>
    <!-- container is used to by virtual dom to render the content-->
    <div id="container"></div>

    <!-- make sure type is text/babel, because we wrote jsx syntax here -->
    <script type="text/babel">
        // make sure we don't have quotes on the value, because it's not a String
        const vDOM = <h1>Hello World</h1>;
        // React 17
        // ReactDOM.render(vDOM, document.getElementById("container"));

        //  React 18
        const root = ReactDOM.createRoot(document.getElementById("container"));
        root.render(vDOM);
    </script>

</body>
</html>
```

Two ways to create Virtual DOM

```
<body>
<!-- container is used to by virtual dom to render the content-->
<div id="container"></div>
<!-- React Core Library -->
    <script type="text/javascript" src="./libraries/react.development.js"></script>

    <!-- React DOM - used by react to do DOM operations -->
    <script type="text/javascript" src="./libraries/react-dom.development.js"></script>

    <!-- Used to transfer jsx to js -->
    <script type="text/javascript" src="./libraries/babel.min.js"></script>

<!-- make sure type is text/babel, because we wrote jsx syntax here -->
<script type="text/babel">
    // make sure we don't have quotes on the value, because it's not a String
    const vDOM = (
        <h1 id="title">
            <span>Hello World</span>
        </h1>
    );

    //  React 18
    const root = ReactDOM.createRoot(document.getElementById("container"));
    root.render(vDOM);
</script>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello World</title>
    <!-- React Core Library -->
    <script type="text/javascript" src="./libraries/react.development.js"></script>

    <!-- React DOM - used by react to do DOM operations -->
    <script type="text/javascript" src="./libraries/react-dom.development.js"></script>

</head>
<body>
<!-- container is used to by virtual dom to render the content-->
<div id="container"></div>

<script type="text/javascript">
    const spanVDOM = React.createElement('span', null, 'Hello World');
    const vDOM = React.createElement('h1', {id: 'title'}, spanVDOM);
    //  React 18
    const root = ReactDOM.createRoot(document.getElementById("container"));
    root.render(vDOM);
</script>

</body>
</html>
```

Two ways to create Virtual DOM

1. JSX
   - **User-Friendly Syntax**: JSX is a JavaScript extension that provides a syntax similar to HTML. It allows you to define React elements using a familiar markup-like structure, making it more accessible to developers, especially those with web development experience.
   - **Readability**: JSX code tends to be more readable and maintainable, as it resembles the structure of the rendered UI. It makes it easier to understand the component's hierarchy.
   - **Babel Transpilation**: JSX code needs to be transpiled by tools like Babel to convert it into standard JavaScript that browsers can understand. This transpilation step is a common part of the modern web development workflow.

2. JavaScript – `React.createElement`
   - **Programmatic Approach**: `React.createElement` is a JavaScript function provided by React that allows you to create React elements programmatically.
   - **Less User-Friendly**: The syntax of React.createElement is less user-friendly compared to JSX. It can be more verbose and harder to read, especially for complex component hierarchies.
   - **No Transpilation Required**: Since React.createElement is standard JavaScript, you don't need to transpile it.

Playground: https://infoheap.com/online-react-jsx-to-javascript/

Virtual DOM vs Real DOM

1. Virtual DOM is a JavaScript object stored in memory.
2. Virtual DOM is lightweight, Real DOM is heavyweight. (Compare their properties)
3. Virtual DOM is a representation of the real DOM, will be transferred into Real DOM and render on the page.

```
<script type="text/babel">
   // make sure we don't have quotes on the value, because it's not a String
   const vDOM = (
      <h1 id="title">
         <span>Hello World</span>
      </h1>
   );

   const root = ReactDOM.createRoot(document.getElementById("container"));
   root.render(vDOM);
   console.log("Virtual DOM", vDOM);
   console.log(typeof vDOM);
   console.log(vDOM instanceof Object);
   const realDOM = document.getElementById("container");
   console.log("Real DOM", realDOM);
   debugger;
</script>
```

JSX

JSX (JavaScript XML) is a syntax extension for JavaScript that is commonly used in React to describe the structure and elements of the user interface. JSX allows you to write HTML-like code within JavaScript, making it easier to define and render React components.

Rules:

1. Don't use quote when define virtual DOM.
2. JavaScript Expressions: To insert dynamic data, variables, or expressions into the rendered content, curly braces `{}` inside JSX elements. You cannot use statements.
3. Class Name: Use the `className` attribute to set the CSS class of an element.
4. Inline Style: Use the `style` attribute to set inline style with syntax `{{key:value}}`.
5. Only One Top-Level Element: all elements within a JSX expression must be wrapped in a single parent element.
6. Must have closing tag. Be aware of Self-Closing Tags.
7. The first letter of a tag
    1) If it's lower case, it'll be translated to HTML element with the same tag name. If couldn't find in HTML, throw error but still display.
    2) If it's upper case, react will render the component, if no component, throw error.
8. Comments: use syntax below, must inside the top-level element.

```
{/* THis is a comment*/}
```

JSX Exercise

Create a page display a list of locations which stored in Array.

- Fairfield
- Ottumwa
- Iowa City

```
<script type="text/babel">
    const arr = ['Fairfield', 'Ottumwa', 'Iowa City'];
    const vDOM = (
        <div>
            { arr.map((location, index) => <li key={index}>{location}</li>) }
        </div>
    );
    const root = ReactDOM.createRoot(document.getElementById('container'));
    root.render(vDOM);
</script>
```

Module vs Component

Module:

- JavaScript modules allow you to break up your code into separate files.
- Example: 1 giant js file -> separate into multiple JS files(a.js, b.js, c.js, etc)

Components:

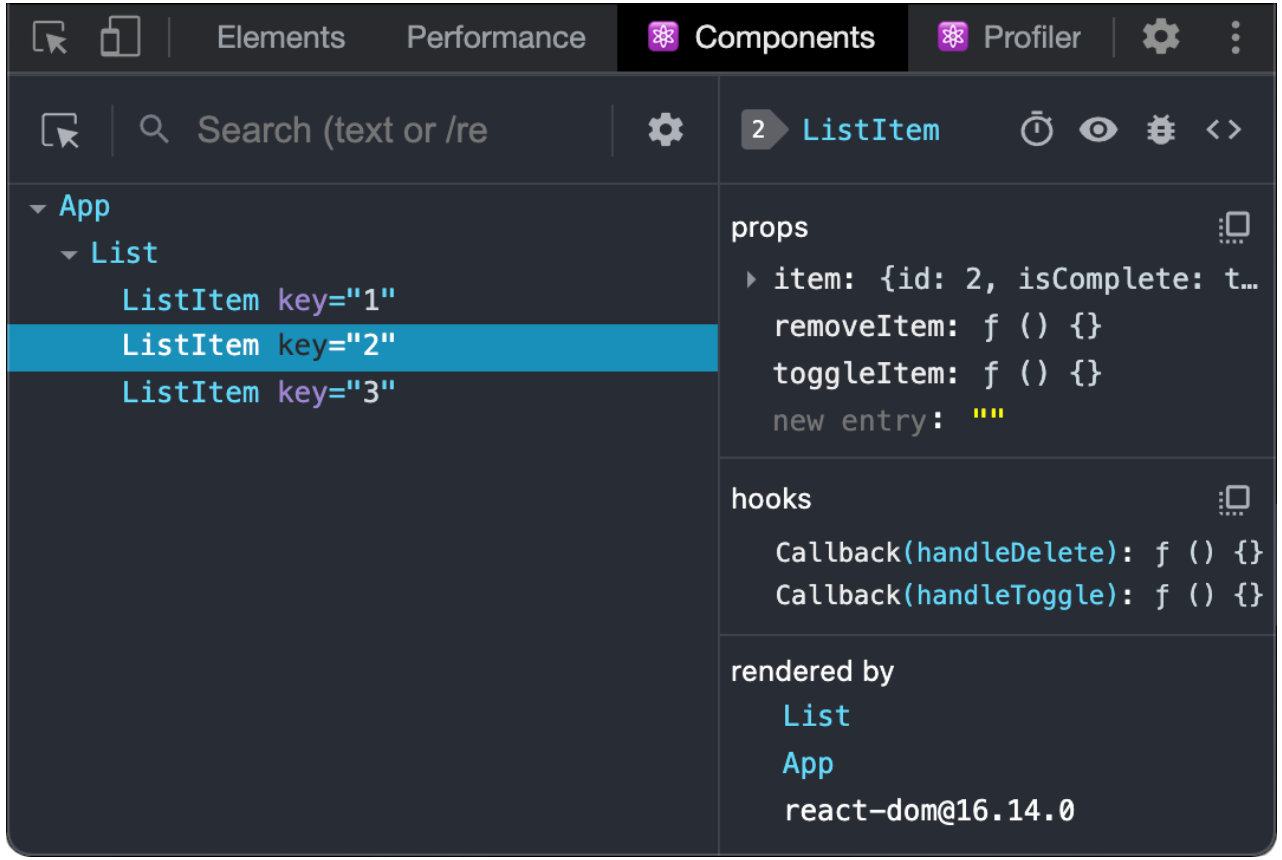
- let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Example: not only js file is splited, all resources such as html, css, js, videos, images, etc

React Developer Tools

Use React Developer Tools to inspect React components, edit props and state, and identify performance problems.

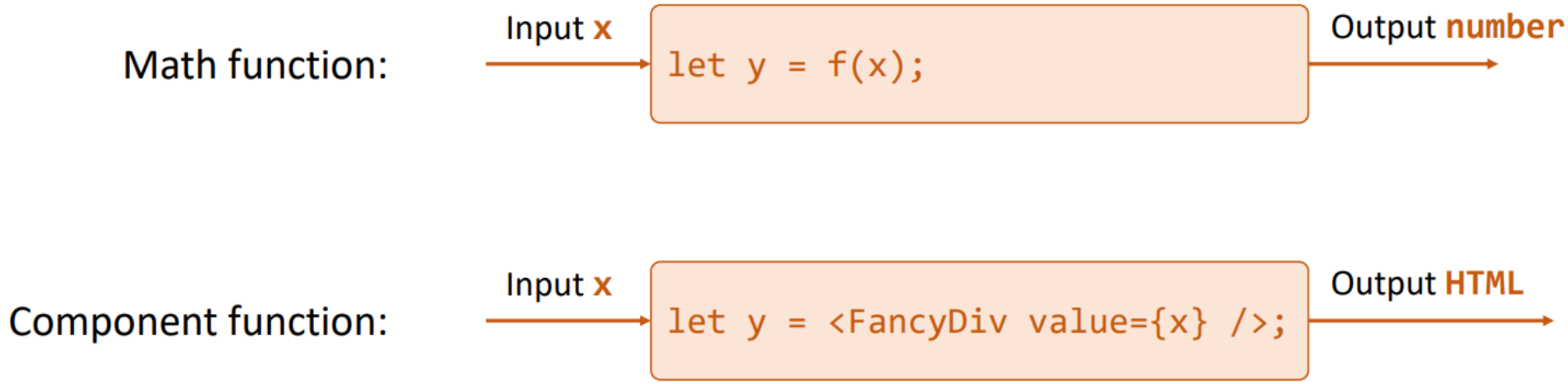
Install for **Chrome** Install for **Firefox**
Install for **Edge**

if you visit a website **built with React,** you will see the *Components* and *Profiler* panels.



Components

Components are functions for user interfaces



Types of Components

Two types of Components:

- Function Component
  - Function components are defined as functions and are used for simple, stateless presentation components.
  - With the introduction of React Hooks, functional components have become even more powerful and can manage state and side effects as well.
  - The name of function component must start with capital case.

- Class Component
  - A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.
  - The component also requires a render() method, this method returns JSX.
  - In older React code bases, you may find Class components primarily used. It is now suggested to use Function components along with Hooks, which were added in React 16.8.

Function Component Demo

The simplest way to define a component is to write a JavaScript function:

- Function name must start with capital case
- `this` keyword inside function component is `undefined`.

```
<script type="text/babel">
    function Demo(){
        return <h1>I'm a function component</h1>;
    }

    ReactDOM.createRoot(document.getElementById('container')).render(<Demo />);
</script>
```

Class Component Demo

- Must extends `React.Component`
- Needs to implement `render` method
- `this` keywords points to current instance of the class
- Not recommended to use now.

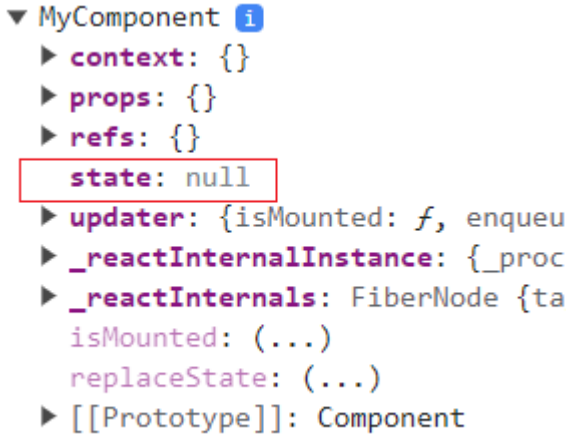
```
<script type="text/babel">
    class MyComponent extends React.Component {
        render(){
            console.log(this);
            return <h1>I'm a class component</h1>;
        }
    }

    ReactDOM.createRoot(document.getElementById('container')).render(<MyComponent />);
</script>
```

Component Cores: State

The state is a built-in React object that is used to contain data or information about the component.

A component's state can change over time; whenever it changes, the component re-renders.

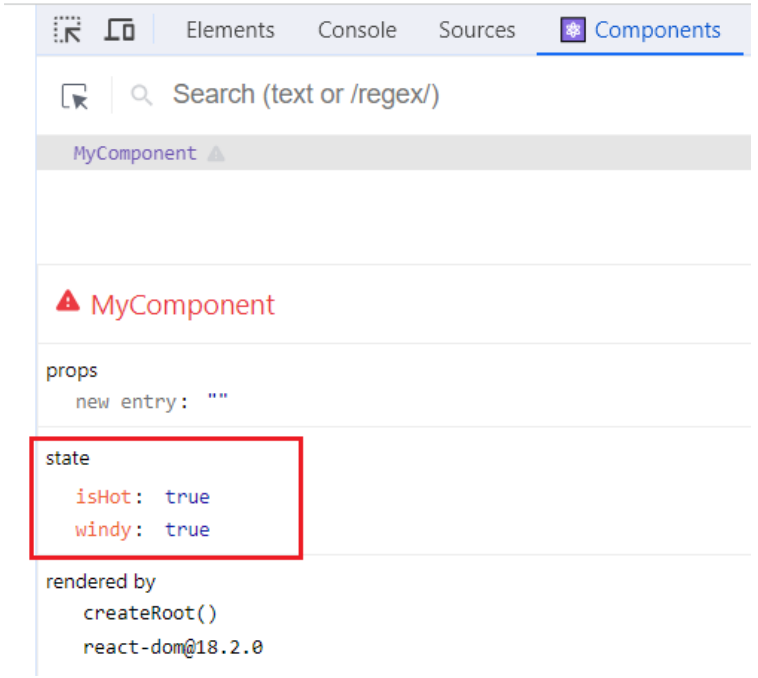
- A state can be modified based on user action or network changes
- Every time the state of an object changes, React re-renders the component to the browser
- The state object is initialized in the constructor
- The state object can store multiple properties using Object.
- `this.setState()` is used to change the value of the state object



Component Core: State - Init State

The way to initialize state in class component is via constructor.

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = {isHot: true, windy: true}
    }
    render() {
        const {isHot} = this.state;
        return <h1>Today is {isHot ? 'HOT!!!!' : 'COOL~~~'}</h1>;
    }
}

ReactDOM.createRoot(document.getElementById('container')).render(<MyComponent/>);
```



Component Core:
State - Change the state: setState

1. `this` keyword inside `render()`, it points to the instance of the component.
2. `this` keyword inside custom-defined methods, the value of `this` is `undefined`.
    1) bind `this` by using `bind()`
    2) Or use arrow function
3. To change the data being stored in state, must be `setState()`.
4. Every time we call setState(), render() is called.

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = {isHot: true, isWindy: true}
        this.changeWeather = this.changeWeather.bind(this);
    }

    render() {
        const {isHot, isWindy} = this.state;
        return <h1 onClick={this.changeWeather}>Today
            is {isHot ? 'HOT!!!!' : 'COOL~~~'} and {isWindy ? 'Wu~~~~~' : 'Hu'}</h1>;
    }

    changeWeather() {
        const isHot = this.state.isHot;
        const isWindy = this.state.isWindy;
        this.setState({isHot: !isHot, isWindy: !isWindy});
    }
}
```

Component Core: setState

The `setState` method is used to update the state of a component. React calls setState() asynchronously.

1. setState(stateChange, [callback]) - Updating State with an Object
   - This syntax is called with an object to update one or more state properties.
   - The callback function as the second argument to `setState` to perform an action after the state has been updated.

2. setState(updater, [callback]) - Updating State with a Function
   - This syntax is used with a function that receives the previous state and props and returns the updated state.
   - The callback function as the second argument to `setState` to perform an action after the state has been updated.

Component Cores: props

React components use props to communicate with each other. Every parent component can pass some information to its child components by giving them props. Props might remind you of HTML attributes, but you can pass any JavaScript value through them, including objects, arrays, and functions.

Props are the information that you pass to a JSX tag.

NOTEs:

1. Must use `{ }` if pass number type
2. If pass an object, can use syntax `{…objectName}`

Component Cores: props

```
class Person extends React.Component {

    render(){
        const {name, age, sex} = this.props;
        return (
            <ul>
                <li>name: {name}</li>
                <li>age: {age + 1}</li>
                <li>sex: {sex}</li>
            </ul>
        )
    }
}

const obj = {
    name: 'Jerry',
    age: 90,
    sex: 'male'
}

ReactDOM.createRoot(document.getElementById("container")).render(<Person name="John" age={20} sex="male"/>);
ReactDOM.createRoot(document.getElementById("container1")).render(<Person {...obj}/>);
ReactDOM.createRoot(document.getElementById("container2")).render(<Person name="Lily" age="18" sex="female"/>);
```

```
function Person(props){
    const {name, age, sex} = props;
    return (
        <ul>
            <li>name: {name}</li>
            <li>age: {age + 1}</li>
            <li>sex: {sex}</li>
        </ul>
    );
}
```

Component Cores: props PropTypes and Default Values

Need to import extra library. It's been removed from react core library since v15.5.

In modern React, suggest use TypeScript for static type checking.

```
Person.propTypes = {
    name: PropTypes.string.isRequired,
    age: PropTypes.number
}

Person.defaultProps = {
    age: 18,
    sex: 'male'
}
```

Component Cores: refs

`Refs` provide a way to access DOM nodes or React elements created in the render method. It can be used to “remember” itself. Don’t trigger re-render when you change it.

1. String Refs – easy, but not recommended to use
2. Callback Refs
3. Creating Refs

Component Cores: refs - String Refs

Use `ref` attribute in your JSX, access it through `this.refs` which holds all string `refs`.

```
class MyComponent extends React.Component {

    showData = () => { alert(this.refs.input1.value); }

    popData = () => { alert(this.refs.input2.value); }

    render(){
        return (
            <div>
                <input ref="input1" placeholder="please enter message" />
                <button onClick={this.showData}>Click to see data</button>
                <input ref="input2" placeholder="please enter message again" onBlur={this.popData}/>
            </div>
        );
    }
}

ReactDOM.createRoot(document.getElementById("container")).render(<MyComponent/>);
```

Component Cores: refs - Callback Refs

In the callback function, the current node is passed to the callback function as the first argument. Then we set the node as a property of current component instance.

```
class MyComponent extends React.Component {

    showData = () => {    alert(this.input1.value);   }
    setInputRef = c => {    this.input2 = c;    }
    popData = (c) => {    alert(this.input2.value);  }

    render(){
        return (
            <div>
                <input ref={c => {console.log(c); this.input1 = c;}} placeholder="please enter message" />
                <button onClick={this.showData}>Click to see data</button>
                <input ref={this.setInputRef} placeholder="please enter message again" onBlur={this.popData}/>
            </div>
        );
    }
}

ReactDOM.createRoot(document.getElementById("container")).render(<MyComponent/>);
```

Component Cores: refs - Creating Refs

Use method `React.createRef()`, assign it to a property to the instance. Each created ref is only used for 1 target.

```
class MyComponent extends React.Component {

    inputRef1 = React.createRef();
    inputRef2 = React.createRef();
    showData = () => { alert(this.inputRef1.current.value); }

    popData = () => { alert(this.inputRef2.current.value); }

    render(){
        return (
            <div>
                <input ref={this.inputRef1} placeholder="please enter message" />
                <button onClick={this.showData}>Click to see data</button>
                <input ref={this.inputRef2} placeholder="please enter message again" onBlur={this.popData}/>
            </div>
        );
    }
}

ReactDOM.createRoot(document.getElementById("container")).render(<MyComponent/>);
```

When to use Refs

Typically, you will use a ref when your component needs to “step outside” React and communicate with external APIs—often a browser API that won’t impact the appearance of the component. Here are a few of these rare situations:

1. Storing timeout IDs
2. Storing and manipulating DOM elements, which we cover on the next page
3. Storing other objects that aren’t necessary to calculate the JSX.

If your component needs to store some value, but it doesn’t impact the rendering logic, choose refs.

Don’t overuse Refs.

Responding to Event

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.

1. Register through onEvent property, be aware of the camel case used here.
   a. React customs onEvent, not using the original DOM event.
   b. Events in React are handled through event delegation (delegated to the outermost element of the component)
2. Can get the DOM object via event.target. Don’t overuse Refs.

Responding to Event Demo

```
<script type="text/babel">
    class MyComponent extends React.Component {
        popData = (event) => {
            alert(event.target.value);
        }

        render() {
            return (
                <div>
                    <input placeholder="please enter message again" onBlur={this.popData}/>
                </div>
            );
        }
    }

    ReactDOM.createRoot(document.getElementById("container")).render(<MyComponent/>);
</script>
```

React Uncontrolled Components

In React, when working with form elements like input fields, there are two main approaches to managing and updating their values: controlled components and uncontrolled components.

Uncontrolled Components:

1. DOM-Based: Uncontrolled components rely on the DOM to manage and update the value of form elements. React does not maintain direct control over the value.
2. No State Updates: The value of the form element is managed by the DOM. It's not updated through React's state management but rather accessed using references to DOM elements.
3. Implicit Updates: When the user interacts with an uncontrolled form element, the DOM updates the value directly. React does not manage the value update explicitly.

React Uncontrolled Components Demo

```
<script type="text/babel">
    class Login extends React.Component {

        login = () => {
            const {username, password} = this;
            alert(`Username is: ${username.value}, password: ${password.value}`);
        }

        render() {
            return (
                <form>
                    <input ref={c => this.username = c} name="username"/>
                    <input ref={c => this.password = c} type="password" name="password"/>
                    <button onClick={this.login}>Login</button>
                </form>
            );
        }
    }

    ReactDOM.createRoot(document.getElementById('container')).render(<Login />);
</script>
```

React Controlled Components

1. State-Based: Controlled components are tightly coupled with React's state management. They use React state to store and update the value of form elements.
2. Controlled by React: The value of the form element is controlled by React. This means that the value is set and updated through React's state management, and React maintains full control over the form element's value.
3. Explicit Updates: When the user interacts with a controlled form element, an event handler is used to update the state with the new value, causing the component to re-render and reflect the updated value.

React Controlled Components Demo

```
<script type="text/babel">
    class Login extends React.Component {

        changeUsername = (event)=> {   this.setState({username: event.target.value});  }

        changePassword = (event)=> {
            this.setState({password: event.target.value});
        }

        login = () => {
            const {username, password} = this.state;
            alert(`Username is: ${username}, password: ${password}`);
        }

        render() {
            return (
                <form>
                    <input onChange={this.changeUsername} name="username"/>
                    <input onChange={this.changePassword} type="password" name="password"/>
                    <button onClick={this.login}>Login</button>
                </form>
            );
        }
    }
    ReactDOM.createRoot(document.getElementById('container')).render(<Login />);
</script>
```

No Function Currying used

```
class Login extends React.Component {

    saveFormData = (type, event) => {
        this.setState({[type]: event.target.value});
    }

    login = (event) => {
        event.preventDefault();
        const {username, password} = this.state;
        alert(`Username is: ${username}, password: ${password}`);
    }

    render() {
        return (
            <form>
                <input onChange={event => this.saveFormData('username', event)} name="username"/>
                <input onChange={event =>  this.saveFormData('password', event)} type="password" name="password"/>
                <button onClick={this.login}>Login</button>
            </form>
        );
    }
}
```

Higher-Order Function: Function Currying

A “higher-order function” is a function that accepts functions as parameters and/or returns a function.

Currying is a transformation of functions that translates a function from callable as `f(a, b, c)` into callable as `f(a)(b)(c)`. Currying doesn’t call a function. It just transforms it.

```
class Login extends React.Component {

    saveFormData = (type) => {
        return event => {
            this.setState({[type]: event.target.value});
        };
    }

    login = () => {
        const {username, password} = this.state;
        alert(`Username is: ${username}, password: ${password}`);
    }

    render() {
        return (
            <form>
                <input onChange={this.saveFormData('username')} />
                <input onChange={this.saveFormData('password')} type="password"/>
                <button onClick={this.login}>Login</button>
            </form>
        );
    }
}
```

Main Points

A good design reflects re-usability and adaptability and most importantly traceability of requirements. A good web design promotes stability in the business application and flexibility in the presentation layer. The Field of Pure Creative Intelligence is characterized by the qualities of stability and flexibility.

Transcendental consciousness is the experience of pure consciousness, the unified field of physics. Just by having this experience one gains this wholeness of knowledge and actions will be accord with all the laws of nature.