**JSX**

* Syntax extension to JS.
* Được recommend sử dụng với React để **miêu tả** UI should look like
* React không yêu cầu phải sử dụng JSX, tuy nhiên phần lớn devs cảm thấy nó dễ dùng và trực quan hơn.

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

* JSX cũng là Expression



**Updating the Rendered Element:**

* React element là bất biến (immutable). Một khi chúng ta đã create an element, chúng ta không thể thay đổi children or attributes.
* Một element giống như một single frame của movie, nó thể hiện UI ở một thời điểm nhất định
* Với những hiểu biết ở thời điểm hiện tại các duy nhất để update UI là create a new element, và pass vào **ReactDOM.render()**
  + React chỉ updates những gì cần thiết:
    - React DOM sẽ compares element và children của element đó với phiên bản trước đó, và chỉ applies DOM update những gì cần thiết để có được state mong muốn

****

**--------------- --------------- --------------- ---------------**

**Components và Props**

* Components:
  + Giúp split UI thành các thành phần độc lập, tăng tính tái sử dụng
* Function Components:

**src\components\Welcome.js**

import React from 'react';

**const** welcome = (props) **=>** {

  return <p>Welcome {props.name} to a React Application</p>

}

export default welcome;

* Class Component:

**src\App.js**

**class** App **extends** React.Component {

  render() {

    return (

      <div>

        <h1>Hello</h1>

        <Welcome name="Xuân Hiếu" />

      </div>

    );

  }

}

export default App;

* Khi mà React nhin thấy một elements được representing cho một user-defined component, nó sẽ passes JSX attributes và children (những gì nằm giữa start và closing tag) vào component này như là một single objectc, còn gọi là object **props.**
* Composing Components:
  + Component có thể refer đến các component khác trong output của các component này.
  + Điều này cho phép chúng ta sử dụng cùng một component abstraction cho bất cứ detail level nào mà chúng ta mong muốn, có thể là các button, dialogs …
* Extracting components:
  + Đừng ngại việc split components thành các thành phần nhỏ hơn.
* Props là Read-only:
  + Bất cứ khi nào chúng ta declare một component như là một function hoặc class, component này không bao giờ tự modify các props của chính nó.
  + React rất linh hoạt tuy nhiên nó có 1 **strict rule:** tất cả các components phải được sử dụng như là những pure functions và respect đến props của chính nó
  + **State** trong React cho phép components có thể thay đổi output tương ứng với user actions, network response … mà không hề vi phạm rule ở trên.

--------------- --------------- --------------- ---------------

**State and Lifecycle**

**src\components\Tick\Tick.js**

import React from 'react';

import Clock from './Clock/Clock';

**class** Tick **extends** React.Component {

  state = {

    date: new Date()

  }

  startClockHandler = () **=>** {

    setInterval(() **=>** {

**const** date = new Date();

      this.setState({

        date: date

      });

    }, 1000);

  }

  render() {

    return (

      <div>

        <Clock date={this.state.date} />

        <button onClick={this.startClockHandler}>Start Clock</button>

      </div>

    )

  }

}

export default Tick;

**src\components\Tick\Clock\Clock.js**

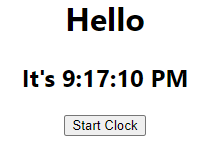
import React from 'react';

**const** clock = (props) **=>** {

  return <h2>It's {props.date.toLocaleTimeString()}</h2>

}

export default clock;

****

* Adding Lifecycle methods to a Class
  + Trong ứng dụng với nhiều component, có những lúc chúng ta sẽ cần free-up resources taken by components khi mà chúng được removed
  + Chúng ta muốn setup a timer mỗi khi Clock được rendered to DOM for the first time, nó sẽ called **mounting** in React:
    - **componentDidMount**: method run mỗi khi component output được rendered to DOM
    - Đây là nơi thích hợp để khởi tạo giá trị của timer, chúng ta sẽ không cần button để trigger việc start clock nữa.

**src\components\Tick\Tick.js**

  componentDidMount() {

    console.log('[Tick.js] componentDidMount');

    setInterval(() **=>** {

**const** date = new Date();

      this.setState({

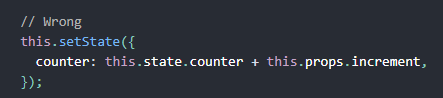
        date: date

      });

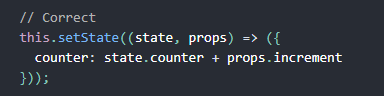
    }, 1000);

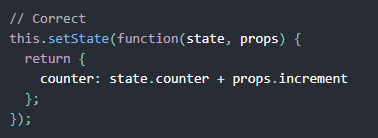
  }

* + Đồng thời chúng ta cũng muốn clear timer này mỗi khi DOM produced by **Tick** được remove, đây được gọi là **unmounting** trong React
    - **componentWillUnmount:** run mỗi khi outout của component bị removed khỏi DOM
* Sử dụng state đúng cách:
  + Không bao giờ modify state directly, thay vào đó hãy dùng **setState().** Nơi duy nhất mà chúng ta có thể assign this.state là constructor
  + State updates có thể Async
    - React có thể batch (gộp?) các multiple setState() calls thành 1 update để tăng hiệu suất
    - Bởi vì **this.props** và **this.state** có thể được updated async, nên chúng ta không nên update value dựa vào giá trị valu trước đó, ví dụ:

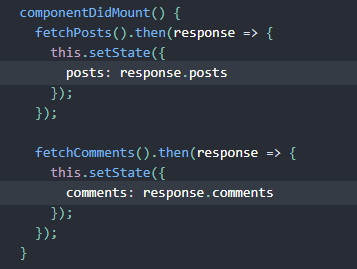
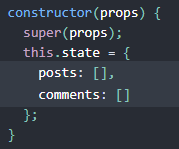


* + - Để fix, sử dụng second form của **setState()** accepts một function chứ không phải là một object.Function này sẽ nhận vào **previous state** là first arg và **props** ở cùng thời điểm mà update được applied là 2nd arg





* + State updates are Merged:
    - Khi chúng ta call **setState(),** React sẽ merge object mà chúng ta cung cấp vào current state



* + - Khi chúng ta update các state variable một cách riêng biệt, do đó giá trọ còn lại sẽ không bị ảnh hưởng và được merged vào giá trị của property được updated
* The Data flows down:
  + Cả parent và child components có thể biết được chắc chắn component là stateful hay stateless, và chúng cũng không care liệu chúng được defined như là các function hay class component. 🡺 lý do vì sao state còn được gọi là local hay encapsulated. Nó sẽ không thể bị accessible bởi bất cứ component nào ngoài component sở hữu và sets it.
  + Đây được gọi là “top-down” hay “unidirectional” data flow.

--------------- --------------- --------------- ---------------

Handling Events.

* Handling Events trong React elements tương tự với handling events đối với DOM elements, syntax có thể khác như sau:
  + React events are named using camelCase
  + Với JSX, có thể pass function như là event handler

**src\App.js**

**class** App **extends** React.Component {

**constructor**(props) {

    super(props);

    this.state = {

      displayClock: true

    }

*// This binding is necessary to make `this` work in the callback*

    this.hideClockHandler = this.hideClockHandler.bind(this);

  }

  hideClockHandler() {

    console.log('vao day')

    this.setState((state, props) **=>** ({

      displayClock: !state.displayClock

    }));

  }

  render() {

    return (

      <div className="App">

        <h1>Hello</h1>

        {this.state.displayClock ? <Tick /> : null}

        <button

          onClick={this.hideClockHandler}>

          Hide Clock

        </button>

      </div>

    );

  }

}

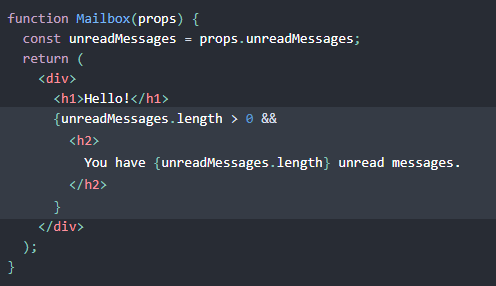
* Lưu ý trong việc dùng **this** trong JSX callbacks:
  + In JS, class methods không được bound by default, nên chúng ta phải bind trong contructor nếu không **this** sẽ là **undefined** khi function được called thực sự
  + Đây không phải là behavior của React
  + Sử dụng method dạng arrow function có thể thay thế việc dùng **bind.**
* Passing Args to Event Handler
  + Trong một vòng lặp, đôi khi chúng ta muốn truyền một extra param vào event handler.
  + 2 dòng code dưới đây có chức năng tương tự. Trong cả 2 trường hợp thì tham số thứ 2 sẽ là React event param,



**Conditional Rendering**

****

* Sử dụng Inline If với Logical && operator:



* Nếu như việc xử lý điều kiện trở nên quá phức tạp, đây là trường hợp mà chúng ta nên [**extract a component**](https://reactjs.org/docs/components-and-props.html#extracting-components)

--------------- --------------- --------------- ---------------

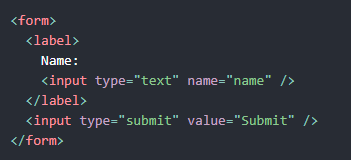
**Lists and Keys**

* Khi render một list các item hay component, phải có unique key giữa các items này.

--------------- --------------- --------------- ---------------

**Form**

* HTML Form elements work hơi khác so với các DOM element khác trong React, vì bản chất form elements thường keep các internal state. Ví dụ form dưới đây sẽ accepts a single name



* + Form này sẽ có một default HTML form behavior là sẽ browsing to a new page khi user submits the form. Trong React behavior này sẽ diễn ra bình thường, tuy nhiên most cases chúng ta sẽ muốn JS function handles the submission của form và access data user đã nhập 🡺 **Controlled Component**
* **Controlled Components:**

****

****

--------------- --------------- --------------- ---------------

**Lifting State Up**

Thông thường có những components sẽ cần tham chiếu đến sự thay đổi của cùng một data. Do đó chúng ta sẽ shared state up đến một ancestor chung của những component này.

**src\components\Calculator\Calculator.js**

import BoilingVerdict from './BoilingVerdict/BoilingVerdict';

import TemperatureInput from './TemperatureInput/TemperatureInput';

class Calculator extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      temperature: 20

    }

    this.enterTemperatureHandler = this.enterTemperatureHandler.bind(this);

  }

  toCelsius(temperature, scale) {

    if (scale === 'c') return temperature;

    return (temperature - 32) \* 5 / 9;

  }

  enterTemperatureHandler(scale, e) {

    const celValue = this.toCelsius(e.target.value, scale);

    this.setState({

      temperature: celValue

    });

  }

  render() {

    return(

      <div>

        <TemperatureInput

          scale='c'

          temperature={this.state.temperature}

          changed={(scale, e) => this.enterTemperatureHandler(scale, e)}

        />

        <TemperatureInput

          scale='f'

          temperature={this.state.temperature}

          changed={(scale, e) => this.enterTemperatureHandler(scale, e)}

        />

        <BoilingVerdict

          celsius={parseFloat(this.state.temperature)}

        />

      </div>

    )

  }

}

**src\components\Calculator\TemperatureInput\TemperatureInput.js**

import React from 'react';

const scaleNames  = {

  c: 'Celsius',

  f: 'Fahrenheit'

}

const temperatureInput = props => {

  const temperratureConvert = (temperature, scaleNames) => {

    if (scaleNames === 'c') return temperature;

    return (temperature \* 9 / 5) + 32;

  }

  return (

    <fieldset>

      <legend>

        Enter temperature in {scaleNames[props.scale]}:

      </legend>

      <input

        type="number"

        value={temperratureConvert(props.temperature, props.scale)}

        onChange={(e) => props.changed(props.scale, e)}

      />

    </fieldset>

  );

}

export default temperatureInput;

**Recaps:**

* Chỉ nên có duy nhất một “source of truth” cho bất cứ data nào thay đổi trong React App.

**--------------- --------------- --------------- ---------------**

**Composition vs Inheritance**

React có hệ thống composition model rất mạnh mẽ, do đó recommend using composition thay cho kế thừa để reuse code giữa các component

**src\components\Containers\FancyBorder.js**

import React from 'react';

import classes from './FancyBorder.module.css';

const fancyBorder = (props) => {

  const classArr = [classes.FancyBorder, classes['FancyBorder-' + props.color]];

  return (

    <div className={classArr.join(' ')}>

      {props.children}

    </div>

  );

}

export default fancyBorder;

**src\components\WelcomeDialog\WelcomeDialog.js**

import React from 'react';

import style from './WelcomeDialog.module.css';

import FancyBorder from '../Containers/FancyBorder';

const welcomeDialog = props => {

  return <FancyBorder color={props.color}>

    <h1 className={style.DialogTitle}>

      Welcome

    </h1>

    <p className={style.DialogMessage}>

      Thank you for visiting our spacecraft!

    </p>

  </FancyBorder>

}

export default welcomeDialog;

**src\App.js**

    return (

      <div className="App">

        <h1>Testing Application</h1>

        <WelcomeDialog color="blue"/>

        <br />

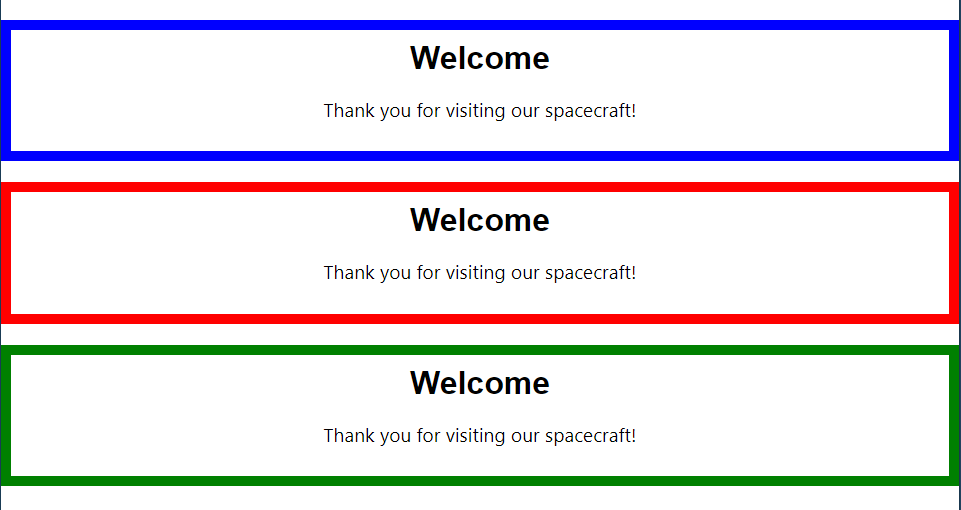
        <WelcomeDialog color="red"/>

        <br />

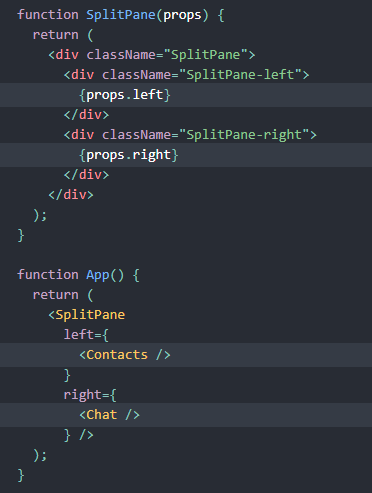
        <WelcomeDialog color="green"/>

      </div>

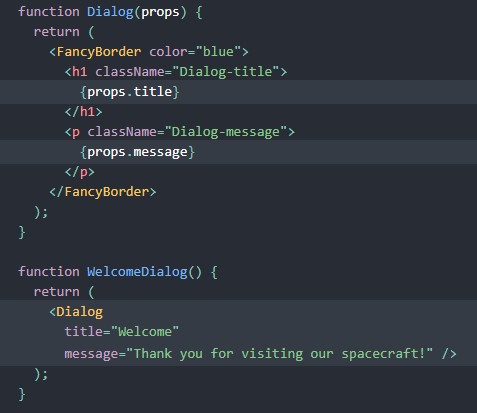
    );



Có những trường hợp, chúng ta muốn có multiple “holes” trong component, thay vì dùng **props.children,** chúng ta có thể làm như sau:



* Đôi khi chúng ta có thể coi **WelcomeDialog** là một case của **Dialog,** khi đó chúng ta có thể implêmnt component của chúng ta như sau:



**--------------- --------------- --------------- ---------------**

**Thinking in React**

**--------------- --------------- --------------- --------------- --------------- --------------- --------------- --------------- --------------- --------------- --------------- ---------------**

**ADVANCED GUIDES**