React - Intro to JSX

Consider this variable declaration:

```
const element = <h1>Hello, world!</h1>;
```

This funny tag syntax is neither a string nor HTML.

It is called JSX, and it is a syntax extension to JavaScript. We recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript.

Note : Since JSX is closer to JavaScript than to HTML, React DOM uses `camelCase` property naming convention instead of HTML attribute names.

For example, `class` becomes `className` in JSX, and `tabindex` becomes `tabIndex`.

Embedding Expressions in JSX

You can embed any JavaScript expression in JSX by wrapping it in curly braces.

For example, 2 + 2, user.firstName, and formatName(user) are all valid expressions:

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

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

JSX is an Expression Too

After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

This means that you can use JSX inside of 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>;
}
```

JSX Represents Objects

There are two ways to create JSX objects.
These two examples are identical:

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

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

Rendering Elements

Elements are the smallest building blocks of React apps.

An element describes what you want to see on the screen:

```
const element = <h1>Hello, world</h1>;
```

Let’s say there is a `<div>` somewhere in your HTML file:

```
<div id="root"></div>
```

We call this a “root” DOM node because everything inside it will be managed by React DOM.

Applications built with just React usually have a single root DOM node.

Rendering Elements cont.

To render a React element into a root DOM node, pass both to `ReactDOM.render()`:

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

Lab - creating a react clock

1. Creating a new react project : https://github.com/facebook/create-react-app
    A. Open terminal run command npm install create-react-app -g
    B. Navigate to my document then run command create-react-app react-clock
    C. Navigate to react-clock folder : cd react-clock
    D. Run command npm start to start the react app.
2. Testing pre build app
    A. Open chrome navigate to localhost:3000. You should see a sample react app
    B. Open react-clock in VS code.
    C. Delete code inside the return block of App.js

Lab - creating a react clock

2. Testing pre build app

   D. Delete code inside the return block of index.js :

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

3. Creating Clock

   A. Add code to create and display the clock.

```
function tick() {
  const element = (
    <div>
      <h1>My First React Lab</h1>
      <h2>The time is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'));
}
```

Lab - creating a react clock

3. Creating Clock
    B. Add javascript code to update the DOM

```
setInterval(tick, 1000);
```

4. Modify the existing code. Change the color of the time to green and The font of the time to italic.

Components and Props

React Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

Functional and Class Components

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

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

OR

```
const welcome =(props)=> {
  return <h1>Hello, {props.name}</h1>;
}
```

This function is a valid React component because it accepts a single “props” (which stands for properties) object argument with data and returns a React element. We call such components “functional” because they are literally JavaScript functions.

Functional and Class Components cont.

You can also use an ES6 class to define a component:

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

Rendering a Component

Elements can also represent user-defined components:

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

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object “props”.

For example, this code renders “Hello, Lamar” on the page:

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

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

TAKE A MIN AND TRY IT OUT!!!!

State and Lifecycle

Consider the ticking clock example from our previous lab.

So far we have only learned one way to update the UI.

We call `ReactDOM.render()` to change the rendered output

State is similar to props, but it is private and fully controlled by the component.

We mentioned before that components defined as classes have some additional features.

Local state is exactly that: a feature available only to classes.

claim.Academy [st. louis]

Lab 2- creating react clock with component

1. Create a new react project lab-2.
2. Create a new js file called Clock.js
3. Import react/Component :

```
import React, { Component } from 'react';
```

4. Create clock component :

```
class Clock extends React.Component {
  render() {
    return (
      <div>
        <h1>My Second React Lab</h1>
        <h2>The time is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
export default Clock;
```

Lab 2- creating react clock with component

4. Add a class constructor that assigns the initial this.state:

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
  render() {
  ...
  ...
```

Note how we pass `props` to the base constructor:

Class components should always call the base constructor with `props`.

Lab 2- creating react clock with component

5. Import and add Clock component to index.js

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

Note how we pass `props` to the base constructor:

Class components should always call the base constructor with `props`.

claim.Academy [st. louis]

Lab 2- creating react clock with component

6. Adding Lifecycle Methods to a Class

Note：We want to set up a timer whenever the Clock is rendered to the DOM for the first time.

This is called “mounting” in React.

We also want to clear that timer whenever the DOM produced by the Clock is removed.

This is called “unmounting” in React.

The componentDidMount() hook runs after the component output has been rendered to the DOM.

This is a good place to set up a timer:

```
componentDidMount () {
    this.timerID = setInterval (
      () => this.tick(),
      1000
    );
  }
render() {
 ....
```

Lab 2- creating react clock with component

6. Adding Lifecycle Methods to a Class

We will tear down the timer in the `componentWillUnmount()` lifecycle hook:

```
componentWillUnmount() {
    clearInterval(this.timerID);
  }
render() {
 ....
```

7. Setting the state

Finally, we will implement a method called tick() that the Clock component will run every second.

It will use this.setState() to schedule updates to the component local state:

Lab 2- creating react clock with component

7. Setting the state

Finally, we will implement a method called tick() that the Clock component will run every second.

It will use this.setState() to schedule updates to the component local state:

```
tick() {
    this.setState({
      date: new Date()
    });
  }
render() {
 ....
```

8. Add functionality to randomly change the color of the displayed time for each second. Color choices : red, green and black.

claim.Academy
{st. louis}

claim.Academy
{st. louis}

claim.Academy
{st. louis}

claim.Academy
{st. louis}