REACT CHAPTER 3 &4

Writing Markup with JSX

JSX is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file. Although there are other ways to write components, most React developers prefer the conciseness of JSX, and most codebases use it.

You will learn
Why React mixes markup with rendering logic
How JSX is different from HTML
How to display information with JSX
JSX: Putting markup into JavaScript
The Web has been built on HTML, CSS, and JavaScript. For many years, web developers kept content in HTML, design in CSS, and logic in JavaScript—often in separate files! Content was marked up inside HTML while the page's logic lived separately in JavaScript:

HTML markup with purple background and a div with two child tags: p and form.
HTML

Three JavaScript handlers with yellow background: onSubmit, onLogin, and onClick.
JavaScript

But as the Web became more interactive, logic increasingly determined content. JavaScript was in charge of the HTML! This is why in React, rendering logic and markup live together in the same place—components.

React component with HTML and JavaScript from previous examples mixed. Function name is Sidebar which calls the function isLoggedIn, highlighted in yellow. Nested inside the function highlighted in purple is the p tag from before, and a Form tag referencing the component shown in the next diagram.
Sidebar.js React component

React component with HTML and JavaScript from previous examples mixed. Function name is Form containing two handlers onClick and onSubmit highlighted in yellow. Following the handlers is HTML highlighted in purple. The HTML contains a form element with a nested input element, each with an onClick prop.
Form.js React component

Keeping a button's rendering logic and markup together ensures that they stay in sync with each other on every edit. Conversely, details that are unrelated, such as the button's markup and a sidebar's markup, are isolated from each other, making it safer to change either of them on their own.

Each React component is a JavaScript function that may contain some markup that React renders into the browser. React components use a syntax extension called JSX to represent that markup. JSX looks a lot like HTML, but it is a bit stricter and can display

dynamic information. The best way to understand this is to convert some HTML markup to JSX markup.

Note
JSX and React are two separate things. They're often used together, but you can use them independently of each other. JSX is a syntax extension, while React is a JavaScript library.

Converting HTML to JSX

Suppose that you have some (perfectly valid) HTML:

```
<h1>Hedy Lamarr's Todos</h1>
<img
  src="https://i.imgur.com/yXOvdOSs.jpg"
  alt="Hedy Lamarr"
  class="photo"
>
<ul>
    <li>Invent new traffic lights
    <li>Rehearse a movie scene
    <li>Improve the spectrum technology
</ul>
```

And you want to put it into your component:

```
export default function TodoList() {
  return (
    // ???
  )
}
```

If you copy and paste it as is, it will not work:

App.js

```
export default function TodoList() {
  return (
    // This doesn't quite work!
    <h1>Hedy Lamarr's Todos</h1>
    <img
      src="https://i.imgur.com/yXOvdOSs.jpg"
      alt="Hedy Lamarr"
      class="photo"
    >
    <ul>
      <li>Invent new traffic lights
```

```
      <li>Rehearse a movie scene
      <li>Improve the spectrum technology
    </ul>
  );
}
```

Error
/src/App.js: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? (5:4)

```
  3 |     // This doesn't quite work!
  4 |     <h1>Hedy Lamarr's Todos</h1>
> 5 |     <img
    |     ^
  6 |       src="https://i.imgur.com/yXOvdOSs.jpg"
  7 |       alt="Hedy Lamarr"
  8 |       class="photo"
```

This is because JSX is stricter and has a few more rules than HTML! If you read the error messages above, they'll guide you to fix the markup, or you can follow the guide below.

Note
Most of the time, React's on-screen error messages will help you find where the problem is. Give them a read if you get stuck!

The Rules of JSX

1. Return a single root element
To return multiple elements from a component, wrap them with a single parent tag.

For example, you can use a :

```
<div>
  <h1>Hedy Lamarr's Todos</h1>
  <img
    src="https://i.imgur.com/yXOvdOSs.jpg"
    alt="Hedy Lamarr"
    class="photo"
  >
  <ul>
    ...
  </ul>
</div>
```

If you don't want to add an extra to your markup, you can write <> and </> instead:

```
<>
  <h1>Hedy Lamarr's Todos</h1>
```

```
  <img
    src="https://i.imgur.com/yXOvdOSs.jpg"
    alt="Hedy Lamarr"
    class="photo"
  >
  <ul>
    ...
  </ul>
</>
```

This empty tag is called a Fragment. Fragments let you group things without leaving any trace in the browser HTML tree.

Deep Dive
Why do multiple JSX tags need to be wrapped?

Hide Details
JSX looks like HTML, but under the hood it is transformed into plain JavaScript objects. You can’t return two objects from a function without wrapping them into an array. This explains why you also can’t return two JSX tags without wrapping them into another tag or a Fragment.

2. Close all the tags

JSX requires tags to be explicitly closed: self-closing tags like <img> must become <img />, and wrapping tags like oranges must be written as oranges.

This is how Hedy Lamarr’s image and list items look closed:

```
<>
  <img
    src="https://i.imgur.com/yXOvdOSs.jpg"
    alt="Hedy Lamarr"
    class="photo"
   />
  <ul>
    <li>Invent new traffic lights</li>
    <li>Rehearse a movie scene</li>
    <li>Improve the spectrum technology</li>
  </ul>
</>
```

3. camelCase all most of the things!

JSX turns into JavaScript and attributes written in JSX become keys of JavaScript objects. In your own components, you will often want to read those attributes into variables. But JavaScript has limitations on variable names. For example, their names can’t contain dashes or be reserved words like class.

This is why, in React, many HTML and SVG attributes are written in camelCase. For example, instead of stroke-width you use strokeWidth. Since class is a reserved word, in React you write className instead, named after the corresponding DOM property:

```
<img
```

```
  src="https://i.imgur.com/yXOvdOSs.jpg"
  alt="Hedy Lamarr"
  className="photo"
/>
```

You can find all these attributes in the list of DOM component props. If you get one wrong, don’t worry—React will print a message with a possible correction to the browser console.

Pitfall

For historical reasons, aria-* and data-* attributes are written as in HTML with dashes.

Pro-tip: Use a JSX Converter

Converting all these attributes in existing markup can be tedious! We recommend using a converter to translate your existing HTML and SVG to JSX. Converters are very useful in practice, but it’s still worth understanding what is going on so that you can comfortably write JSX on your own.

Here is your final result:

App.js

```
export default function TodoList() {
  return (
    <>
      <h1>Hedy Lamarr's Todos</h1>
      <img
        src="https://i.imgur.com/yXOvdOSs.jpg"
        alt="Hedy Lamarr"
        className="photo"
      />
      <ul>
        <li>Invent new traffic lights</li>
        <li>Rehearse a movie scene</li>
        <li>Improve the spectrum technology</li>
      </ul>
    </>
  );
}
```

Nested JSX elements

In order for the code to compile, a JSX expression must have exactly one outermost element. In the below block of code the <a> tag is the outermost element.

```
const myClasses = (
  <a href="https://www.codecademy.com">
    <h1>
      Sign Up!
    </h1>
  </a>
);
```

JSX Syntax and JavaScript

JSX is a syntax extension of JavaScript. It's used to create DOM elements which are then rendered in the React DOM.

A JavaScript file containing JSX will have to be compiled before it reaches a web browser. The code block shows some example JavaScript code that will need to be compiled.

```
import React from 'react';
import { createRoot } from 'react-dom/client';

const container = document.getElementById('app');
const root = createRoot(container);
root.render(<h1>Render me!</h1>);
```

Multiline JSX Expression

A JSX expression that spans multiple lines must be wrapped in parentheses: (and). In the example code, we see the opening parentheses on the same line as the constant declaration, before the JSX expression begins. We see the closing parentheses on the line following the end of the JSX expression.

```
const myList = (
  <ul>
    <li>item 1</li>
    <li>item 2</li>
    <li>item 3</li>
  </ul>
```

```
);
```

JSX syntax and HTML

In the block of code we see the similarities between JSX syntax and HTML: they both use the angle bracket opening and closing tags (`<h1>` and `</h1>`).

When used in a React component, JSX will be rendered as HTML in the browser.

```
const title = <h1>Welcome all!</h1>
```

JSX attributes

The syntax of JSX attributes closely resembles that of HTML attributes. In the block of code, inside of the opening tag of the `<h1>` JSX element, we see an `id` attribute with the value `"example"`.

```
const example = <h1 id="example">JSX Attributes</h1>;
```

ReactDOM JavaScript library

The JavaScript library `react-dom/client` contains the `createRoot()` method, which is used to create a React root at the HTML element used as an argument. The React root renders JSX elements to the DOM by taking a JSX expression, creating a corresponding tree of DOM nodes, and adding that tree to the DOM.

The code example begins by creating a React root at the HTML element with the id `app` and storing it in `root`. Then, using `root`'s `render()` method, the JSX used as an argument is rendered.

```
import React from 'react';
import { createRoot } from 'react-dom/client';

const container = document.getElementById('app');
const root = createRoot(container);

root.render(<h1>This is an example.</h1>);
```

Embedding JavaScript in JSX

JavaScript expressions may be embedded within JSX expressions. The embedded JavaScript expression must be wrapped in curly braces.

In the provided example, we are embedding the JavaScript expression `10 * 10` within the `<h1>` tag. When this JSX expression is rendered to the DOM, the embedded JavaScript expression is evaluated and rendered as `100` as the content of the `<h1>` tag.

```
let expr = <h1>{10 * 10}</h1>;
// above will be rendered as <h1>100</h1>
```

The Virtual Dom

React uses Virtual DOM, which can be thought of as a blueprint of the DOM. When any changes are made to React elements, the Virtual DOM is updated. The Virtual DOM finds the differences between it and the DOM and re-renders only the elements in the DOM that changed. This makes the Virtual DOM faster and more efficient than updating the entire DOM.

JSX className

In JSX, you can't use the word `class`! You have to use `className` instead. This is because JSX gets translated into JavaScript, and `class` is a reserved word in JavaScript.

When JSX is rendered, JSX `className` attributes are automatically rendered as `class` attributes.

```
// When rendered, this JSX expression...
const heading = <h1 className="large-heading">Codecademy</h1>;

// ...will be rendered as this HTML
<h1 class="large-heading">Codecademy</h1>
```

JSX and conditional

In JSX, `&&` is commonly used to render an element based on a boolean condition. `&&` works best in conditionals that will sometimes do an action, but other times do nothing at all.

If the expression on the left of the `&&` evaluates as true, then the JSX on the right of the `&&` will be rendered. If the first expression is false, however, then the JSX to the right of the `&&` will be ignored and not rendered.

```
// All of the list items will display if
// baby is false and age is above 25
const tasty = (
  <ul>
    <li>Applesauce</li>
    { !baby && <li>Pizza</li> }
    { age > 15 && <li>Brussels Sprouts</li> }
    { age > 20 && <li>Oysters</li> }
    { age > 25 && <li>Grappa</li> }
  </ul>
```

```
);
```

JSX conditionals

JSX does not support if/else syntax in embedded JavaScript. There are three ways to express conditionals for use with JSX elements:

1. a ternary within curly braces in JSX
2. an if statement outside a JSX element, or
3. the && operator.

```
// Using ternary operator
const headline = (
  <h1>
    { age >= drinkingAge ? 'Buy Drink' : 'Do Teen Stuff' }
  </h1>
);

// Using if/else outside of JSX
let text;

if (age >= drinkingAge) { text = 'Buy Drink' }
else { text = 'Do Teen Stuff' }

const headline = <h1>{ text }</h1>

// Using && operator. Renders as empty div if length is 0
const unreadMessages = [ 'hello?', 'remember me!'];

const update = (
  <div>
    {unreadMessages.length > 0 &&
      <h1>
        You have {unreadMessages.length} unread messages.
      </h1>
    }
  </div>
);
```

Embedding JavaScript code in JSX

Any text between JSX tags will be read as text content, not as JavaScript. In order for the text to be read as JavaScript, the code must be embedded between curly braces `{ }`.

```
<p>{ Math.random() }</p>

// Above JSX will be rendered something like this:
<p>0.88</p>
```

JSX element event listeners

In JSX, event listeners are specified as attributes on elements. An event listener attribute's *name* should be written in camelCase, such as `onClick` for an `onclick` event, and `onMouseOver` for an `onmouseover` event.

An event listener attribute's *value* should be a function. Event listener functions can be declared inline or as variables and they can optionally take one argument representing the event.

```
// Basic example
const handleClick = () => alert("Hello world!");

const button = <button onClick={handleClick}>Click here</button>;

// Example with event parameter
const handleMouseOver = (event) => event.target.style.color = 'purple';

const button2 = <div onMouseOver={handleMouseOver}>Drag here to change color</div>;
```

Setting JSX attribute values with embedded JavaScript

When writing JSX, it's common to set attributes using embedded JavaScript variables.

```
const introClass = "introduction";
const introParagraph = <p className={introClass}>Hello world</p>;
```

JSX .map() method

The array method `map()` comes up often in React. It's good to get in the habit of using it alongside JSX.

If you want to create a list of JSX elements from a given array, then `map()` over each element in the array, returning a list item for each one.

```
const strings = ['Home', 'Shop', 'About Me'];

const listItems = strings.map(string => <li>{string}</li>);

<ul>{listItems}</ul>
```

JSX empty elements syntax

In JSX, empty elements must explicitly be closed using a closing slash at the end of their tag: `<tagName />`.

A couple examples of empty element tags that must explicitly be closed include `<br>` and `<img>`.

```
<br />
<img src="example_url" />
```

`React.createElement()` Creates Virtual DOM Elements

The `React.createElement()` function is used by React to actually create virtual DOM elements from JSX. When the JSX is compiled, it is replaced by calls to `React.createElement()`.

You usually won't write this function yourself, but it's useful to know about.

```
// The following JSX...
const h1 = <h1 className="header">Hello world</h1>;

// ...will be compiled to the following:
const h1 = React.createElement(
  'h1',
  {
    className: 'header',
  },
  'Hello world'
);
```

JSX `key` attribute

In JSX elements in a list, the `key` attribute is used to uniquely identify individual elements. It is declared like any other attribute.

Keys can help performance because they allow React to keep track of whether individual list items should be rendered, or if the order of individual items is important.

```
<ul>
  <li key="key1">One</li>
  <li key="key2">Two</li>
  <li key="key3">Three</li>
  <li key="key4">Four</li>
</ul>
```

Next

JavaScript in JSX with Curly Braces

JSX lets you write HTML-like markup inside a JavaScript file, keeping rendering logic and content in the same place. Sometimes you will want to add a little JavaScript logic or reference a dynamic property inside that markup. In this situation, you can use curly braces in your JSX to open a window to JavaScript.

You will learn
How to pass strings with quotes
How to reference a JavaScript variable inside JSX with curly braces
How to call a JavaScript function inside JSX with curly braces
How to use a JavaScript object inside JSX with curly braces
Passing strings with quotes
When you want to pass a string attribute to JSX, you put it in single or double quotes:

App.js

```
export default function Avatar() {
  return (
    <img
      className="avatar"
      src="https://i.imgur.com/7vQD0fPs.jpg"
      alt="Gregorio Y. Zara"
    />
  );
}
```

Here, "https://i.imgur.com/7vQD0fPs.jpg" and "Gregorio Y. Zara" are being passed as strings.

But what if you want to dynamically specify the src or alt text? You could use a value from JavaScript by replacing " and " with { and }:

App.js

export default function Avatar() {

```
  const avatar = 'https://i.imgur.com/7vQD0fPs.jpg';
  const description = 'Gregorio Y. Zara';
  return (
    <img
      className="avatar"
      src={avatar}
      alt={description}
    />
  );
}
```

Notice the difference between className="avatar", which specifies an "avatar" CSS class name that makes the image round, and src={avatar} that reads the value of the JavaScript variable called avatar. That's because curly braces let you work with JavaScript right there in your markup!

Using curly braces: A window into the JavaScript world
JSX is a special way of writing JavaScript. That means it's possible to use JavaScript inside it—with curly braces { }. The example below first declares a name for the scientist, name, then embeds it with curly braces inside the <h1>:

App.js

```
export default function TodoList() {
  const name = 'Gregorio Y. Zara';
  return (
    <h1>{name}'s To Do List</h1>
  );
}
```

Try changing the name's value from 'Gregorio Y. Zara' to 'Hedy Lamarr'. See how the list title changes?

Any JavaScript expression will work between curly braces, including function calls like formatDate():

App.js

```
const today = new Date();

function formatDate(date) {
  return new Intl.DateTimeFormat(
    'en-US',
    { weekday: 'long' }
  ).format(date);
}
```

```
export default function TodoList() {
  return (
    <h1>To Do List for {formatDate(today)}</h1>
  );
}
```

Where to use curly braces

You can only use curly braces in two ways inside JSX:

As text directly inside a JSX tag: <h1>{name}'s To Do List</h1> works, but <{tag}>Gregorio Y. Zara's To Do List</{tag}> will not.
As attributes immediately following the = sign: src={avatar} will read the avatar variable, but src="{avatar}" will pass the string "{avatar}".
Using “double curlies”: CSS and other objects in JSX
In addition to strings, numbers, and other JavaScript expressions, you can even pass objects in JSX. Objects are also denoted with curly braces, like { name: "Hedy Lamarr", inventions: 5 }. Therefore, to pass a JS object in JSX, you must wrap the object in another pair of curly braces: person={{ name: "Hedy Lamarr", inventions: 5 }}.

You may see this with inline CSS styles in JSX. React does not require you to use inline styles (CSS classes work great for most cases). But when you need an inline style, you pass an object to the style attribute:

App.js

```
export default function TodoList() {
  return (
    <ul style={{
      backgroundColor: 'black',
      color: 'pink'
    }}>
      <li>Improve the videophone</li>
      <li>Prepare aeronautics lectures</li>
      <li>Work on the alcohol-fuelled engine</li>
    </ul>
  );
}
```

Try changing the values of backgroundColor and color.

You can really see the JavaScript object inside the curly braces when you write it like this:

```
<ul style={
  {
    backgroundColor: 'black',
    color: 'pink'
```

```
  }
}>
```
The next time you see {{ and }} in JSX, know that it’s nothing more than an object inside the JSX curlies!

Pitfall
Inline style properties are written in camelCase. For example, HTML <ul style="background-color: black"> would be written as <ul style={{ backgroundColor: 'black' }}> in your component.

More fun with JavaScript objects and curly braces
You can move several expressions into one object, and reference them in your JSX inside curly braces:

App.js

```
const person = {
  name: 'Gregorio Y. Zara',
  theme: {
    backgroundColor: 'black',
    color: 'pink'
  }
};

export default function TodoList() {
  return (
    <div style={person.theme}>
      <h1>{person.name}'s Todos</h1>
      <img
        className="avatar"
        src="https://i.imgur.com/7vQD0fPs.jpg"
        alt="Gregorio Y. Zara"
      />
      <ul>
        <li>Improve the videophone</li>
        <li>Prepare aeronautics lectures</li>
        <li>Work on the alcohol-fuelled engine</li>
      </ul>
    </div>
  );
}
```