JSX syntax and the arrow function

Components as Function Expressions

Up to this point, you've likely only observed ES5 function declarations used to define components in React. However, this is not the only way to do it.

In this reading, you learn about some alternative approaches, specifically by using function expressions and arrow functions.

Function Expressions

Let's start with a function declaration used as a component in React:

```
function Nav(props) {
    return (
        <ul>
            <li>{props.first}</li>
        </ul>
    )
}
```

This component's code returns a list item containing the value of the 'first' prop.

Now, let's change this function declaration to a function expression:

```
const Nav = function(props) {
    return (
        <ul>
            <li>{props.first}</li>
        </ul>
    )
}
```

The component is, for the most part, the same. The only thing that's changed is that you're now using an anonymous (nameless) function, and assigning this anonymous function declaration to a variable declared using the **`const`** keyword, and the name **`Nav`**. The rest of the code is identical.

Changing a component from a function declaration to a function expression doesn't change its behavior, or how you write the code to render the **`Nav`** component. It's still the same:

```
<Nav first="Home" />
```

You can also take this concept a step further, using arrow functions.

Components as Arrow Functions

Arrow functions are a core feature of the ES6 version of JavaScript.

One of the main benefits of using arrow functions is its shorter syntax.

Consider the Nav function expression written as an arrow function:

```
const Nav = (props) => {
    return (
        <ul>
            <li>{props.first}</li>
        </ul>
    )
}
```

So, the way to think about this is the following:

- The arrow itself can be thought of as the replacement for the **`function`** keyword.
- The parameters that this arrow function accepts are listed before the arrow itself.

To reiterate, take the smallest possible **anonymous ES5 function**:

```
const example = function() {}
```

And then observe how this is written as an arrow function:

```
const example = () => {}
```

Another important rule regarding arrow functions is that using the parentheses is optional if there's a single parameter that a function accepts.

In other words, another correct way to write the previous Nav arrow function component would be to drop the parentheses around 'props':

```
const Nav = props => {
    return (
        <ul>
            <li>{props.first}</li>
        </ul>
    )
}
```

In all other cases, when you write arrow functions, **for any number of parameters other than a single parameter, using parentheses around parameters is compulsory**.

For example, if your `Nav` component wasn't accepting any parameters, you'd code it with empty parentheses:

```
const Nav = () => {
    return (
        <ul>
            <li>Home</li>
        </ul>
    )
}
```

Another interesting thing about arrow functions is the **implicit return**. However, it only works if it's on the same line of code as the arrow itself. In other words, the implicit return works if your entire component is a single line of code.

To demonstrate how this works, let's re-write the `Nav` component as a one-liner:

```
const Nav = () => <ul><li>Home</li></ul>
```

Note that with the implicit return, you don't even have to use the curly braces that are compulsory function body delimiters in all other cases.

Using Arrow Functions in Other Situations

In React, just like in plain JavaScript, arrow functions can be used in many different situations. One such situation is using it with, for example, the `forEach()` built-in array method.

For example:

```
[10, 20, 30].forEach(item => item * 10)
```

The output of the above vanilla JavaScript line of code would be three number values:

100
200
300

As a side-note, the term "vanilla JavaScript" is often used to describe the plain, regular JavaScript language syntax, without any framework-specific or library-specific code. For example, React is a library, so in this context, saying that a piece of code is "vanilla JavaScript" means that it doesn't need any special library to run. It can run in "plain"

JavaScript without any additional dependencies.

You could also write this code in ES5 syntax:

```
[10, 20, 30].forEach(function(item) {
    return item * 10
}
)
```

Regardless of how you write it, the `forEach()` method can be run on an array. The `forEach()` method accepts a single parameter: **an anonymous function**. If you write this anonymous function in ES5 syntax, then it would contain a return statement:

```
function(item) {
    return item * 10
}
```

If you write it as an ES6 function instead, it can be simplified as one line:

```
item => item * 10
```

Both these functions perform the exact same task. Only the syntax is different. The ES6 function is a lot shorter because:

- The arrow function has a single parameter, so you do not need to add parentheses around the item parameter (to the left of the arrow)
- Since the arrow function fits on one line of code, you don't need to use curly braces around the function body, or the return keyword; it's implicit

Arrow functions are used extensively in JSX in React, and getting used to their syntax and being able to "mentally parse" it as you read it is an important skill to have and helps you get better at writing React apps.

Now that you have completed this reading, you've learned about some alternative approaches, specifically by using function expressions and arrow functions.