# Ternary operators and functions in JSX

So you've explored several ways to define components in React; this includes function declarations, function expressions and arrow functions.

As you continue with building your knowledge of React syntax, you'll learn to make more use of JSX and embedded JSX expressions.

In this reading, you will become familiar with how to use ternary expressions to achieve a random return, as well as how to invoke functions inside of JSX expressions.

## A different way of writing an if...else conditional

You are likely familiar with the structure of an if...else conditional. Here is a quick refresher:

```
let name = 'Bob';
if (name == 'Bob') {
    console.log('Hello, Bob');
} else {
    console.log('Hello, Friend');
};

```

The above code works as follows:

1. First, I declare a `name` variable and set it to a string of `"Bob"`.
2. Next, I use the `if` statement to check if the value of the name variable is `"Bob"`. If it is, I want to `console.log` the word `"Bob"`.
3. Otherwise, if the name variable's value is not `"Bob"`, the `else` block will execute and output the words `"Hello, Friend"` in the console.

Above, I gave you an example of using an `if...else` conditional. Did you know that there is another, different way, to effectively do the same thing? It's known as the **ternary operator**. A ternary operator in JavaScript uses two distict characters: the first oneis **the question mark**, that is, the `?` character. To the left of the `?` character, you put *a condition that you'd like to check for.* Just like I did in the above `if...else` statement, the condition I'm checking is `name == 'Bob'`. In other words, I'm asking the JavaScript engine to look at the value that's stored inside the `name` variable, and to verify if that value is the same as `'Bob'`. If it is, then the JavaScript engine will return the boolean value of `true`. If the value of the `name` variable is something different from `'Bob'`, the value that the JavaScript engine returns will be the boolean value of `false`.

Here is the code that reflects the explanation in the previous paragraph:

```
name == 'Bob' ?
```

Note that the above code is incompleted. I have the condition that I'm checking (the `name == 'Bob'` part). I also have the `?` character, that is, the first of the two characters needed to construct a syntactically valid ternary operator. However, I still need the second character, which is the colon, that is the `:` character. This character is placed after the question mark character. I can now expand my code to include this as well:

```
name == 'Bob' ? :
```

This brings me a step closer to completing my ternary operator. Although I've added the characters needed to construct the ternary operator, I still need to add the return values. In other words, if `name == 'Bob'` evaluates to true, I want to return the words, "Yes, it is Bob!". Otherwise, I want to return the words "I don't know this person".

```
name == Bob ? "Yes, it is Bob" : "I don't know this person";
```

This, in essence, is how the ternary operator works. It's just some shorthand syntax that I can use as a replacement for the `if` statement. To prove that this is really the case, here's my starting if...else example, written as a ternary operator:

```
let name = 'Bob';
name == 'Bob' ? console.log('Hello, Bob') : console.log('Hello, Friend');

```

## Using ternary expressions in JSX

Let's examine an example of a component which uses a ternary expression to randomly change the text that is displayed.

```
function Example() {
    return (
        <div className="heading">
            <h1>{Math.random() >= 0.5 ? "Over 0.5" : "Under 0.5"}</h1>
        </div>
    );
};
```

Inside the `<h1>` element, the curly braces signal to React that you want it to parse the code inside as regular

JavaScript.

Then, inside the curly braces, you can add a ternary statement. Every ternary statement conceptually, expressed in pseudo-code, works like this:

```
comparison ? true : false
```

In the actual code example at the start of this lesson item, the comparison part, which goes to the left of the question mark, is using the >= (greater-than-or-equal-to operator), to return a Boolean value. If the result of the comparison evaluates to **`true`**, then the string "Over 0.5" gets returned. In other words, whatever sits between the question mark and the semi-colon character will get returned. Otherwise, if the result of the comparison evaluates to **`false`**, then the string "Under 0.5" gets returned. In other words, the value that sits to the right of the colon character will get returned from the ternary expression.

This is how you can use a ternary expression to check for a condition right inside a component and return a value dynamically.

## Using function calls in JSX

Another way to work with an expression in JSX is to invoke a function. Function invocation is an expression because every expression returns a value, and function invocation will always return a value, even when that return value is **`undefined`**.

Like the previous example, you can use function invocation inside JSX to return a random number:

```
function Example2() {
    return (
        <div className="heading">
            <h1>Here's a random number from 0 to 10:
                { Math.floor(Math.random() * 10) + 1 }
            </h1>
        </div>
    );
};
```

In the **`Example2`** component, built-in **`Math.floor()`** and **`Math.random()`** methods are being used, as well as some number values and arithmetic operators, to display a random number between 0 and 10.

You can also extract this functionality into a separate function:

```
function Example3() {

    const getRandomNum = () => Math.floor(Math.random() * 10) + 1

    return (
```

```
        <div className="heading">
            <h1>Here's a random number from 0 to 10: { getRandomNum() }</h1>
        </div>
    );
};
```

The `getRandomNum()` function can also be written as a function declaration, or as a function expression. It does not have to be an arrow function.

But let's observe both alternatives: the function expression *and* the function declaration.

Function expression:

```
const getRandomNum = function() {
    return Math.floor(Math.random() * 10) + 1
} ;
```

Function declaration:

```
function getRandomNum() {
    return Math.floor(Math.random() *10) + 1
};
```

Of course, there are many other examples. The ones used here are there to help you understand how versatile and seamless the JSX syntax is. As you improve your React skills, you will find many creative ways of using JavaScript expressions in JSX.

Now that you have completed this reading, you have learned about a few more ways that you can use expressions in JSX.