# Event handling and embedded expressions

In this reading, you'll learn the different ways to embed expressions in event handlers in React:

- With an inline anonymous ES5 function
- With an inline, anonymous ES6 function (an arrow function)
- Using a separate function declaration
- Using a separate function expression

You may find this reading useful as a reference sheet.

For clarity and simplicity: a function will simply console log some words. This will allow you to compare the difference in syntax between these four approaches, while the result of the event handling will always be the same: just some words output to the console.

## Handling events using inline anonymous ES5 functions

This approach allows you to directly pass in an ES5 function declaration as the `onClick` event-handling attribute's value:

```
<button onClick={function() {console.log('first example')}}>
    An inline anonymous ES5 function event handler
</button>
```

Although it's possible to write your click handlers using this syntax, it's not a common approach and you will not find such code very often in React apps.

## Handling events using inline anonymous ES6 functions (arrow functions)

With this approach, you can directly pass in an ES6 function declaration as the `onClick` event-handling attribute's value:

```
<button onClick={() => console.log('second example')}>
    An inline anonymous ES6 function event handler
</button>
```

This approach is much more common then the previous one. If you want to keep all your logic inside the JSX expression assigned to the onClick attribute, use this syntax.

## Handling events using separate function declarations

With this approach, you declare a separate ES5 function declaration, and then you reference its name in the event-handling `onClick` attribute, as follows:

```
function App() {
    function thirdExample() {
        console.log('third example');
    };
    return (
        <div className="thirdExample">
            <button onClick={thirdExample}>
                using a separate function declaration
            </button>
        </div>
    );
};
export default App;
```

This syntax makes sense to be used when your onClick logic is too complex to easily fit into an anonymous function. While this example is not really showing this scenario, imagine a function that has, for example, 20 lines of code, and that needs to be ran when the click event is triggered. This is a perfect use-case for a separate function declaration.

## Handling events using separate function expressions

**Tip**: A way to determine if a function is defined as an expression or a declaration is: if it does not start the line with the keyword `function`, then it's an expression.

In the following example, you're assigning an anonymous ES6 arrow function to a `const` variable – hence, this is a function expression.

You're then using this const variable's name to handle the `onClick` event, so this is an example of handling events using a separate function expression.

```
function App() {
    const fourthExample = () => console.log('fourth example');

    return (
        <div className="fourthExample">
            <button onClick={fourthExample}>
                using a separate function expression
            </button>
        </div>
  );
};
export default App;
```

The syntax in this example is very common in React. It uses arrow functions, but also allows us to handle situations where our separate function expression spans multiple lines of code.

In this reading lesson item, you've learned the several types of functions you can use to handle events in React. Some of those are more common than others, but now that you know all the different ways of doing this, you can understand other people's code more easily, as well as choose the syntax that best suits your given use case, such as a specific

company coding style guide.