**Coding Activity 1.3**

**Learning Outcome Addressed**

 7.Use closures to access variables within a function’s scope, an outer function’s scope, and global scope

**Using Closures To Understand Scope**

### Using Closures To Simulate Private Variables

You've now seen how important scope is and the relationship between variable scope and closures. As seen in the videos, closures allow you to affect a variable that you wouldn't normally have access to because it's outside of the scope. This is particularly useful in JSX where you are dealing with scope both in JavaScript and in HTML (because JSX files contain a mix of JavaScript and HTML). In this activity, you'll use a closure to simulate private variables. The goal of this activity is to see an example of closures and understand how they impact scope.

#### What is a private variable and how can it help you understand closures?

In some programming languages, an object will have inner variables and functions that can only be accessed after the object has been instantiated. They can be specified with keywords such as public and private. A public variable is a variable that can be accessed both inside and outside of that object block it was instantiated in. A private variable is a variable that can ONLY be accessed **within** the object block that it's defined in. Here's an example with a closure:

function outer() {

var counter = 0;

function inner() {

alert(counter);

counter++; // this increments the counter variable even when inner is called outside the scope of outer

}

return inner;

}

var x = outer(); // As we're calling outer here, x is a reference to inner

x(); // alerts 0, on the first call of inner, counter = 0, so 0 is alerted

x(): // alerts 1, on the second call of inner, we still have the reference to counter which is 1 even though counter does not exist at this place in the code

// but now if you try to write code that directly accesses the counter within the instance of the function outer

alert(x.counter); // alerts undefined rather than alerting 2 because counter by itself does not exist

An alert that is within the same scope that the variable is defined in can access and change the variable, but code outside the scope of outer can't even tell if the var counter exists. In this example, inner is a public function and counter is a private variable.

While getting more familiar with JavaScript and JSX, understanding scope is something you'll constantly struggle with. Although JavaScript doesn't support public and private keywords that can be used to manage access control of variables, understanding the principles behind how a closure works will be invaluable in your future JSX development.

#### Task instructions

In this activity you'll complete the getBookTitle, turnPage, and getPage functions returned by the BookObject closure according to the comments within the functions. Afterward, you can see the results of you providing these accessor functions to the scope outside of BookObject in the web console. You can even experiment with inputs and see if there's any way for you to change those variables outside of the instantiation of the object or the provided functions.

Hints:

* Try to access the variables within BookObject after the first one is instantiated. You'll notice that the variables inside of the *BookObject* function returns undefined when called in the console.logs at the bottom of the file. That is due to these variables not being accesible outside of the *BookObject* function scope. Make sure to keep checking your console logs to see your progress in writing the inner functions.

Task

Define the three methods in the return object within BookObject

//closure.js

/\*

Instructions:

Your task is to complete the described object which uses closure to create a book with name and current page.

Additionally the bookObject will return an inner object with functions that will allow the code outside bookObject to access and modify BookObject Scoped Variables.

\*/

function defineClosure() {

    function BookObject(name = "The Count of Monte Cristo", currentPage = 873) {

        console.log(`\n\nCreating new Book Object with name: ${name} and currentPage: ${currentPage}`);

        var currentPageNumber = currentPage;

        var bookName = name;

        // construct and return an object with the three methods defined below.

        // Your task is to fill in these methods according to the comments in order to to allow outside functions to access

        // and change your BookObject variables ONLY using these gateway functions

        // All methods have access to currentPageNumber and bookName variables, but they do not exist as regular variables outside of the bookObject

        return {

            "getBookTitle": function () {

                // your code here to return the bookName

            },

            "turnPage": function () {

                // your code here to increment the currentPageNumber

            },

            "getPage": function () {

                //your code here to return the currentPageNumber

            }

        };

    }

    var myBook = BookObject('Reckless', 28);

    console.log(`The book name is: ${myBook.getBookTitle()}`);

    console.log(`Accessing myBook's bookName variable directly: ${myBook.bookName}`);

    console.log(`Initial Page Number is: ${myBook.getPage()}`);

    myBook.turnPage();

    console.log(`New Page Number after changing currentPageNumber with turnPage function: ${myBook.getPage()}`);

    // creating a new instance of myBook, this resets all inner variables as its completely different object and scope

    myBook = BookObject();

    console.log(`The book name is: ${myBook.getBookTitle()}`);

    console.log(`Accessing myBook's bookName variable directly: ${myBook.bookName}`);

    console.log(`Initial Page Number is: ${myBook.getPage()}`);

    myBook.turnPage();

    console.log(`New Page Number after changing currentPageNumber with turnPage function: ${myBook.getPage()}`);

    return BookObject();

}

// Don't change this code

window.onload = () => {

    defineClosure();

};

//don't change this code

if (typeof module !== 'undefined') {

    module.exports = { defineClosure };

}

//index.html

<!DOCTYPE html>

<html>

  <head>

    <title>Using Closures to Simulate Private Variables</title>

  </head>

  <body>

    <p>Refresh this page and check web console to see your work.</p>

    <script src="./closure.js"></script>

  </body>

</html>