Practical use of Closures

The closure is a powerful tool in JavaScript. It is commonly used in functional programming languages, but often misunderstood. Like other JavaScript fundamentals, understanding closures is necessary to write expressive, concise and maintainable scripts.

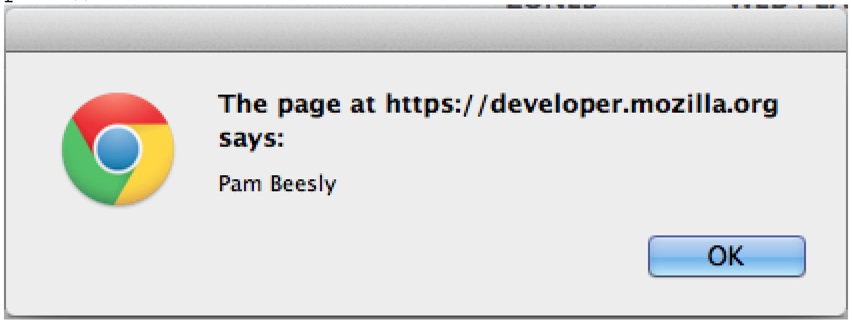
What is a closure?

At first glance, a closure is simply a function defined within another function. However, the power of closures is derived from the fact that the inner function remembers the environment in which it was created. In other words, the inner function has access to the outer function's variables and parameters.

What's it look like?

Below is an example of a closure (courtesy of Mozilla):

```
function pam() {
    var name = "Pam Beesly";
    function displayName() {
        alert (name);
    }
    displayName();
}
pam();
```



The page at https://developer.mozilla.org says:

Pam Beesly

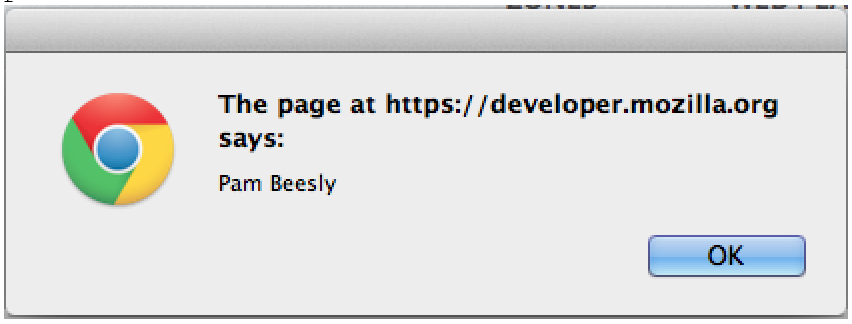
Our outer function—*pam*—does three things:

1. Define a local variable, *name*
2. Define a function, *displayName*
3. Call *displayName*

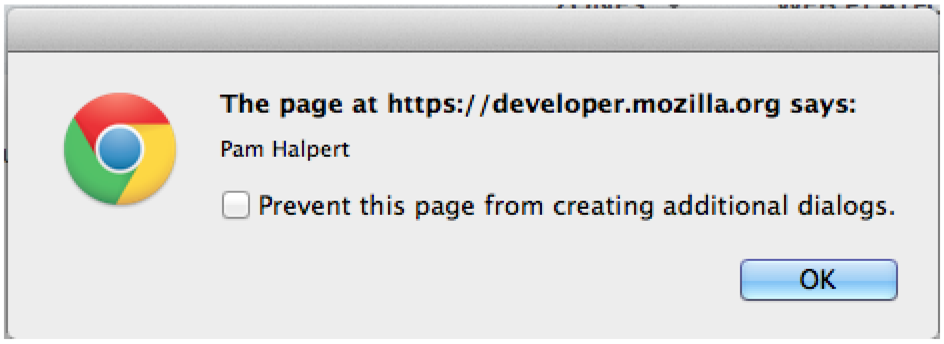
displayName doesn't define any local variables, yet it is able to alert *name* because *name* has been defined in the scope in which the closure was created—that of its outer function.

Closures can do more than just read their outer functions' local variables—they can overwrite them, too. Observe below:

```
function pam() {
    var name = "Pam Beesly";
    function displayName() {
        alert (name);
    }
    function setName(newName) {
        name = newName;
    }
    displayName();
    setName("Pam Halpert");
    displayName();
}
pam();
```


The page at https://developer.mozilla.org says:

Pam Beesly


The page at https://developer.mozilla.org says:

Pam Halpert

As we can see, closures are capable of not only reading, but also manipulating the variables of their outer functions.

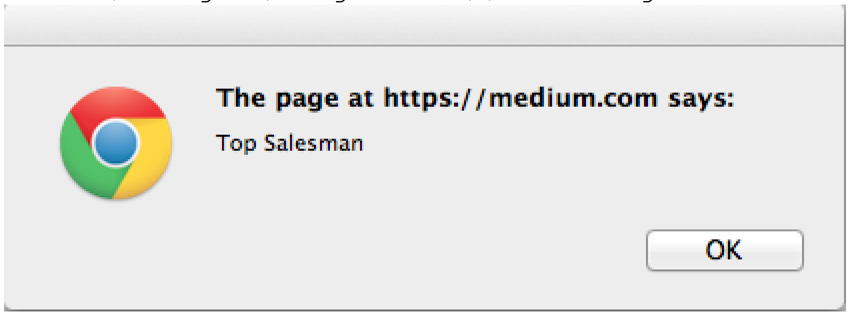
Function factories

One powerful use of closures is to use the outer function as a factory for creating functions that are somehow related.

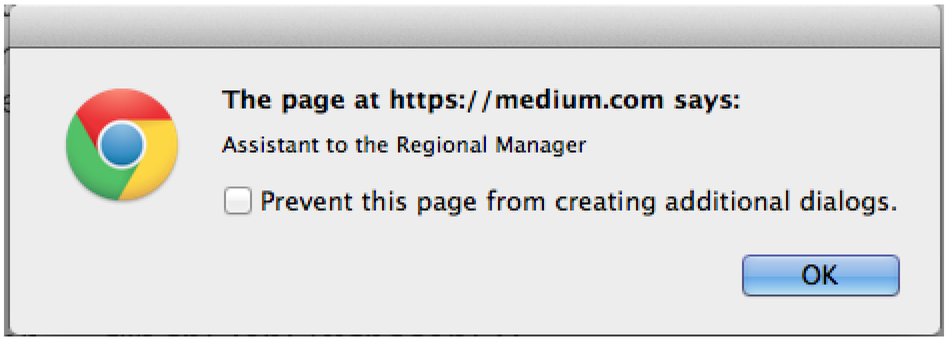
```
function dwightJob(title) {
    return function(prefix) {
        return prefix + ' ' + title;
    };
}

var sales = dwightJob('Salesman');
var manager = dwightJob('Manager');

alert(sales('Top'));  // Top Salesman
alert(manager('Assistant to the Regional')); // Assistant to the
Regional Manager
alert(manager('Regional')); // Regional Manager
```

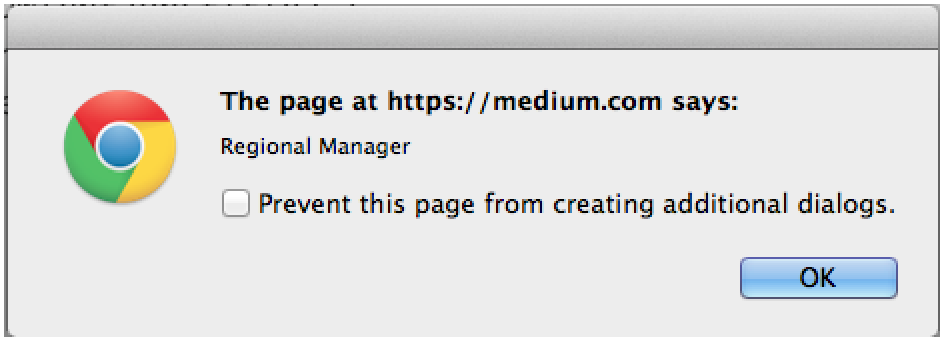

The page at https://medium.com/ says:

Top Salesman


The page at https://medium.com says:

Assistant to the Regional Manager

Prevent this page from creating additional dialogs.


The page at https://medium.com says:

Regional Manager

Prevent this page from creating additional dialogs.

Using closures as function factories is a great way to keep your JavaScript DRY. Five powerful lines of code allow us to create any number of functions with similar, yet unique purposes.

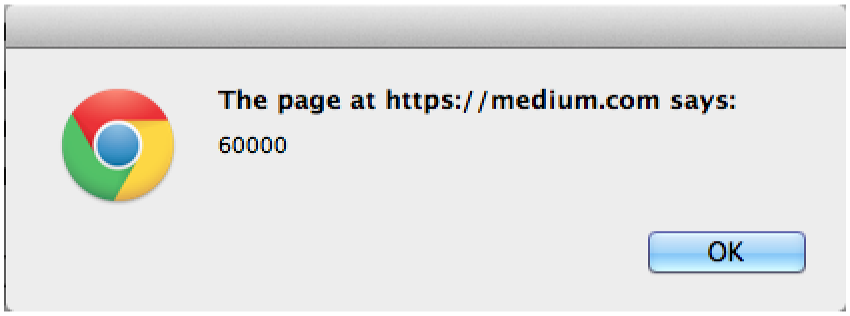
Namespacing private functions

Many object-oriented languages provide the ability to declare methods as either public or private. JavaScript doesn't have this functionality built in, but it does allow to emulate this functionality through the use of closures, which is known as the module pattern.

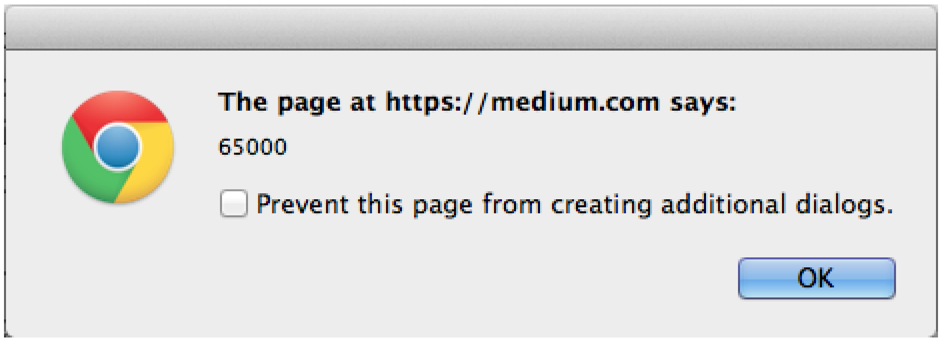
```
var dwightSalary = (function() {
    var salary = 60000;
    function changeBy(amount) {
        salary += amount;
    }
    return {
        raise: function() {
            changeBy(5000);
        },
        lower: function() {
            changeBy(-5000);
        },
        currentAmount: function() {
            return salary;
        }
    };
})();

alert(dwightSalary.currentAmount()); // $60,000
dwightSalary.raise();
alert(dwightSalary.currentAmount()); // $65,000
dwightSalary.lower();
dwightSalary.lower();
alert(dwightSalary.currentAmount()); // $55,000

dwightSalary.changeBy(10000) // TypeError: undefined is not a
function
```

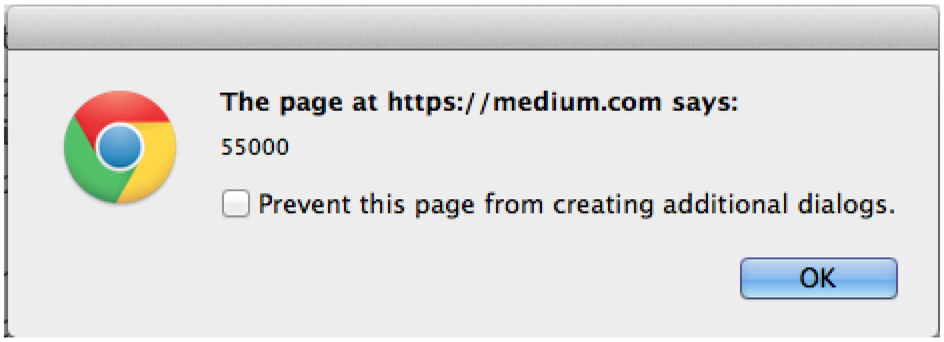

The page at https://medium.com says:

60000


The page at https://medium.com says:

65000

Prevent this page from creating additional dialogs.


The page at https://medium.com says:

55000

Prevent this page from creating additional dialogs.

Using closures to namespace private functions keeps more general namespaces clean, preventing naming collisions. Neither the *salary* variable nor the *changeBy* function are available outside of *dwightSalary*. However, *raise, lower* and *currentAmount* all have access to them and can be called on *dwightSalary*.

These are a few popular uses for closures. You'll surely encounter closures used for other purposes, but these are a couple simple ways to incorporate closures into your code in an immediately useful way.