**1.** Make a function called composedValue that takes two functions f1 and f2 and a value and returns f1(f2(value)), i.e., the first function called on the result of the second function called on the value.

```
function square(x) { return(x*x); }
function double(x) { return(x*2); }
composedValue(square, double, 5); --> 100 // I.e., square(double(5))
```

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            function composedValue(square,double,result){
                document.write("Value is :" +square(double(result)));
            }
            function square(result){
                return(result*result)
            }
            function double(result){
                return(result*2)
            }
            composedValue(square,double,5);
        </script>
    </body>
</html>
```

**2.** Make a function called compose that takes two functions f1 and f2 and returns a new function that, when called on a value, will return f1(f2(value)). Assume that f1 and f2 each take exactly one argument.

```
var f1 = compose(square, double);
f1(5); --> 100
f1(10); --> 400
var f2 = compose(double, square);
f2(5); --> 50
f2(10); --> 200
```

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            function compose(f1,f2){
                function f3(x){
                    document.write("Value is :" +f1(f2(x)));
                }
                return f3;
            }
            function f1(x){
                return(x*x*4);
            }
            function f2(x){
```

```
                return(x*x*2);
            }
            compose(f1,f2) (2);
        </script>
    </body>
</html>
```

**3.** Make a function called "find" that takes an array and a test function, and returns the first element of the array that "passes" (returns non-false for) the test. Don't use map, filter, or reduce.

```
function isEven(num) { return(num%2 == 0); }
isEven(3) --> false
isEven(4) --> true
find([1, 3, 5, 4, 2], isEven); --> 4
```

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            function isEven(n){
                if(n%2==0){
                    return true;
                }
                else{
                    return false;
                }
            }
            var a=[1,3,5,4,2];
            find(a,isEven);
            function find(a,isEven){
                for(i=0;i<a.length;i++){
                    if(isEven(a[i]==true)){
                        document.write(a[i]);
                        break;
                    }
                }
            }
        </script>
    </body>
</html>
```

**4.** Recent JavaScript versions added the "map" method of arrays, as we saw in the notes and used in the previous set of exercises. But, in earlier JavaScript versions, you had to write it yourself. Make a function called "map" that takes an array and a function, and returns a new array that is the result of calling the function on each element of the input array. Don't use map, filter, or reduce.

```
map([1, 2, 3, 4, 5], square); --> [1, 4, 9, 16, 25]
map([1, 4, 9, 16, 25], Math.sqrt); --> [1, 2, 3, 4, 5]
```

Hint: remember the push method of arrays.

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            var a=[1,2,3,4,5];
            function map(a,f1)
            {
                var b =[];
                for(i=0;i<a.length;i++)
                b.push(f1(a[i]));
                document.write(b);
            }
        </script>
    </body>
</html>
```

# FUNCTIONAL PROGRAMMING ADVANCED EXERCISES

**1.** Make a "pure" recursive version of find. That is, don't use any explicit loops (e.g. for loops or the forEach method), and don't use any local variables (e.g., var x = ...) inside the functions. Hint: remember the slice method of arrays.

```
function isEven(num) { return(num%2 == 0); }
isEven(3) --> false
isEven(4) --> true
find([1, 3, 5, 4, 2], isEven); --> 4
```

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            find(a,0,test)
            function find(a,i,isEven){
                if(isEven(a[i])==true){
                    document.write(isEven)
                }
```

```
                else{
                    i++;
                    find(1,3,5,4,2)
                }
            }
            function isEven(n){
                if(n%2==0){
                    return true
                }
                else{
                    return false
                }
            }
        </script>
    </body>
</html>
```

**2.** Make a "pure" recursive version of map. Hint: remember the slice and concat methods of arrays.

```
map([1, 2, 3, 4, 5], square); --> [1, 4, 9, 16, 25]
map([1, 4, 9, 16, 25], Math.sqrt); --> [1, 2, 3, 4, 5]
```

```
<!DOCTYPE html>
<html>
    <body>
        <script>
            i=0;
            f1 =square;
            console.log(map());
            function map()
        {
        if(i == a.length)
            return;
        else{
           document.write(f1(a[i]));
        i++;
        map();
        }
    }
        </script>
    </body>
</html>
```