Iteration and Recursion Examples

- The following JavaScript examples serve to illustrate more uses of iteration and recursion
- Note that there are other iteration examples both in handouts and in some of the assignment keys
- Recursion is relatively new, so we have a few more of those here — and note how some of them can also be written iteratively
- All of these examples are written as functions, so that you can use them in places other than the JavaScript scratch page on the course Web site

Item Counter

```
/**
 * Calculates the number of times that a target value appears
 * in a list.  When using this in a Web page, you'll probably
 * need split(", ") to convert comma-separated text into an
 * array, which is what this function expects to receive in
 * the list variable.
 */
function itemCounter(list, target) {
    var total = 0;
    var index = 0;
    while (index < list.length) {
        var test = list[index];
        if (target == test) {
            total = total + 1;
        }
        index = index + 1;
    }
    return total;
}
```

Iterative Factorial

```
/**
 * Calculates n!, or the product of all numbers from
 * n down to 1 (e.g., 5! = 5 x 4 x 3 x 2 x 1 = 120).
 *
 * Make sure to invoke parseInt() before passing n to
 * this function; also, we don't check for illegal input
 * such as non-integers, negative values, etc.
 */
function factorial(n) {
    var product = 1;
    while (n > 0) {
        product = product * n;
        n = n - 1;
    }
    return product;
}
```

Recursive Factorial

```
/**
 * Calculates n! using a recursive approach.  Which one
 * do you prefer?
 *
 * Same caveats with this program: no real error checking,
 * we require parseInt() beforehand, etc.
 */
function factorial(n) {
    if ((n == 1) || (n == 0)) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Iterative Summation

```
/**
 * Calculates the summation from 1 to n.  For example,
 * summation(5) = 5 + 4 + 3 + 2 + 1 = 15.
 *
 * No error checking here as well, but the correctness
 * of the function is easy to check: summation(n) has
 * a well-known shortcut: (n * (n + 1)) / 2.
 */
function summation(n) {
    var sum = 0;
    while (n > 0) {
        sum = sum + n;
        n = n - 1;
    }
    return sum;
}
```

Recursive Summation

```
/**
 * Like factorial(), summation() can also be written
 * recursively.  Can you sense a pattern between the
 * two “styles” for the same algorithm?
 */
function summation(n) {
    if (n == 1) {
        return 1;
    } else {
        return n + summation(n - 1);
    }
}
```

Fibonacci Calculator (Recursive Only)

```
/**
 * The Fibonacci sequence is a well-known mathematical
 * sequence that starts with 0 and 1.  Each successive
 * number in the sequence is the sum of the previous
 * two numbers; thus: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34...
 *
 * The sequence counts from 0, so Fib(0) = 0, Fib(1) = 1,
 * Fib(2) = 1, Fib(7) = 13, Fib(9) = 34, and so on.
 */
function fibonacci(n) {
    if ((n == 0) || (n == 1)) {
        return n;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

Palindrome Detector (Recursive Only)

```
/**
 * A palindrome is a word that reads the same both backward and forward (e.g., “madam,”
 * “rotor”).  This function detects whether str is a palindrome or not.
 *
 * To do its work, isPalindrome() uses some built-in properties and methods for strings:
 * length tells us the number of characters in the string, charAt() gives us the character
 * at some position, and substring() pulls out a subsection of the string.  For example,
 * "hello".substring(1, 4) is "ell" (characters 1 to, but not including, 4).
 */
function isPalindrome(str) {
    if ((str.length == 0) || (str.length == 1)) {
        // Empty and one-letter expressions are always palindromes.
        return true;
    } else {
        // charAt() starts counting at zero (surprise surprise).
        var firstChar = str.charAt(0);
        var lastChar = str.charAt(str.length - 1);
        if (firstChar == lastChar) {
            return isPalindrome(str.substring(1, str.length - 1));
        } else {
            return false;
        }
     }
 }
```