CS 3721: Programming Languages Lab
Lab #03: Recursion over Lists in Scheme

The purpose of this recitation is to review how to define functions in Scheme and to practice building recursive functions to solve various problems. To define base and recursive cases separately, Scheme supports two control-flow operations, *if* and *cond*. Additionally, it allows local variables being declared and used via the *let* operator. The following are some examples using *if*, *cond* and *let*.

```
(if (>= 3 2) 'abc 5)
(cond ((symbol? 'x) 3)
      ((number? 'x) 'y) (else 'foo))
(let ((x 2) (y 3)) (+ x y))
```

To further assist you, the following *atom*? function takes a single parameter x and returns whether is x is an atomic value.

```
(define atom? (lambda (x) (or (number? x) (symbol? x) (boolean? x))))
```

1. Define a function *len* which takes an arbitrary parameter x, and returns the number of atomic values contained within x. Test your code with the following expressions.

   ```
   > (len 'x)
   1
   > (len '(x 3))
   2
   > (len '((1 2) 3) )
   3
   > (len '((2 3) (1 2)) )
   4
   ```

2. Define a function *count-number* which takes an arbitrary parameter x, and returns the number of numeric values (i.e., numbers) contained within x. Test your code with the following expressions.

   ```
   > (count-number 'x)
   0
   > (count-number '(x 3))
   1
   > (count-number '((1 2) 3) )
   3
   > (count-number '((a 3) (1 2)) )
   3
   ```

3. Define a function *sum* which takes an arbitrary parameter x, and returns the sum of all numeric values (i.e., numbers) contained within x. Test your code with the following expressions.

   ```
   > (sum 'x)
   0
   ```

```
> (sum '(x 3))
3
> (sum '((1 2) 3) )
6
> (sum '((a 3) (1 2)) )
6
```

4. Define a recursive function *Repeat* which takes three parameters,a function (lambda expression) f, a number n, an arbitrary value x. This function uses x as the starting parameter and returns the result of apply function f^n to x; that is, it returns the result of $(f(f...(fx)...))$, where f is repeated n times. Use the following examples to test your function.

```
 > (Repeat (lambda (x) (+ x 2)) 3 3)
9
> (Repeat (lambda (x) (* x 5)) 5 5)
15625
>  (Repeat cdr 5 '(1 2 3 4 5 6 7))
(list 6 7)
> (Repeat (lambda (x) (cons x (cons x '()))) 3 3)
(list (list (list 3 3) (list 3 3)) (list (list 3 3) (list 3 3)))
```

5. (extra-credit) Define a function *Check* which takes two parameters, x and y. The function returns true if x and y are identical; it returns false otherwise. Note that the *eq*? operator provided by Scheme checks whether two atomic values are equal but does not recursively check whether two nested lists are identical, but your function is required to do so. Test your code with the following expressions.

```
> (Check 'x 'y)
false
> (Check 'x 'x)
true
> (Check '(1 2) '(1 2))
true
> (Check '(2 3) '(1 2))
false
```

After you are done, save your scheme solution file and submit your file online at

`http://www.cs.utsa.edu/~cs3723`