CSE 341 Section Handout #6
Cheat Sheet

Types
numbers: integers (3, 802), reals (3.4), rationals (3/4), complex (2+3.4i)
symbols: x, y, hello, r2d2
booleans: #t, #f
strings: "hello", "how are you?"
lists: (list 3 4 5) (list 98.5 "hello" (list 3 82.9) 73)

Constructs
function call: (f arg1 arg2 arg3 ... argN)
variable binding: (define sym expr)
function binding: (define (f p1 p2 ... pN) expr)
function binding (define (f p1 p2 ... pN)
with helpers: (define ...
             (define ...) expr)
let binding: (let ((sym1 e1) (sym2 e2) ... (symN eN)) expr)
let* binding: (let* ((sym1 e1) (sym2 e2) ... (symN eN)) expr)
if expression: (if test e1 e2)
cond expression: (cond (test1 e1)
                       (test2 e2) ...
                       (testN eN))
               (cond (test1 e1)
                       (test2 e2) ...
                       (else eN))

Useful procedures
arithmetic: +, -, *, /, modulo, quotient, remainder
mathematical: abs, sin, cos, max, min, expt, sqrt, floor, ceiling, truncate, round
relational: =, <, >, <=, >=
equality: eq?, eqv?, equal?
logical: and, or, not
higher-order: map, filter, foldl, foldr, sort, andmap, ormap

List procedures
length length of a list
car first element of a list
cdr rest of the list
cons takes a value and a list and joins them; ML's ::
append joins >= 2 lists together; ML's @
list forms a list from a sequence of values
member whether a value is in a list
remove removes one occurrence of a value from a list
null? is something an empty list?
pair? is something a nonempty list?
CSE 341 Section Handout #6

Questions

1. For each of the following definitions of a factorial function, identify the parenthesis error:
   
   a. (define (fact n) (if (= n 0) (1) (* n (fact (- n 1)))))
   b. (define (fact n) (if = n 0 l (* n (fact (- n 1)))))
   c. (define fact (n) (if (= n 0) 1 (* n (fact (- n 1)))))
   d. (define (fact n) (if (= n 0) 1 (* n fact (- n 1))))
   e. (define (fact n) (if (= n 0) 1 (* n ((fact) (- n 1)))))

2. Use the R5RS Scheme standard documentation web site to figure out the following:
   
   a. How do you form a comment in Scheme?
   b. Is there a syntax for multi-line comments?
   c. How is the expression (/ a b c d) evaluated (i.e., left-to-right or right-to-left)?
   d. How would you compare to see if one string is less than another?
   e. How can you sort a list of integers?

3. Define a function called days-in-month that takes an integer representing a month as an argument and that returns the number of days in that month. You may assume that the month value passed is between 1 and 12 inclusive. You may also assume that the month is not part of a leap year. The following table shows the number of days in each month:

<table>
<thead>
<tr>
<th>Month</th>
<th>1 Jan</th>
<th>2 Feb</th>
<th>3 Mar</th>
<th>4 Apr</th>
<th>5 May</th>
<th>6 Jun</th>
<th>7 Jul</th>
<th>8 Aug</th>
<th>9 Sep</th>
<th>10 Oct</th>
<th>11 Nov</th>
<th>12 Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Days</td>
<td>31</td>
<td>28</td>
<td>31</td>
<td>30</td>
<td>31</td>
<td>30</td>
<td>31</td>
<td>31</td>
<td>30</td>
<td>31</td>
<td>30</td>
<td>31</td>
</tr>
</tbody>
</table>

   For example, the call of (days-in-month 5) would return 31.

4. Define a function called pow that takes two integers as arguments and that returns the result of raising the first integer to the power of the second (i.e., (pow x y) should return x^y). You may assume that the power is not negative. For our purposes, we will assume that every integer to the 0 power is 1 (this isn't true of 0 to the 0, but that's okay). For example, (pow 2 10) should return 1024.

5. Define a function called sum-to that accepts an integer n and that computes the sum of the first n reciprocals. That is:

   \[ \sum_{i=1}^{n} \frac{1}{i} \]

   For example, (sum-to 3) should return \(1 + \frac{1}{2} + \frac{1}{3}\) = 1 \(\frac{5}{6}\). The function should return 0 if n is 0. You may assume that the function is not passed a negative value of n. Notice that unlike ML, Scheme can compute these values exactly as rational numbers rather than using the real type.
6. Define a procedure named \texttt{sum} that accepts a list of numbers as a parameter and returns the sum of all the numbers in the list. For example, the call of \((\texttt{sum (list 1 2 -3 4 5)})\) should return 9. \textit{(What happens if you put some real numbers in the list? Fractions? Etc.)}

7. Define a procedure named \texttt{stutter} that takes a list as an argument and that returns the list obtained by replacing every value in the list with two of that value. For example, the call of \((\texttt{stutter '(1 2 3)})\) should return \((1 1 2 2 3 3)\).

8. Define a procedure named \texttt{multiples} that accepts two integer parameters \texttt{n} and \texttt{k} that returns a list of the first \texttt{n} multiples of \texttt{k}. For example, the call of \((\texttt{multiples 3 5})\) should return \((5 10 15)\).

9. 
   a. Write a procedure named \texttt{positive-sum} that that takes a list as an argument and that returns the sum of the \textit{positive} numbers in the list. works on lists of integers only; for example, the call of \((\texttt{positive-sum '}(1 -5 2 3 -6 4 7))\) should return 17. Use your code from the previous \texttt{sum} problem as a basis to get you started.
   
   b. Modify your function so that it can handle lists where some of the elements are non-numbers (skip them). The list might contain inner lists; skip them entirely. (In other words, don't worry about any numbers that might appear inside of any inner lists). For example, the call of \((\texttt{positive-sum '}(1 a b 3.4 -5 "hello" (2 -1 3) -8))\) should return 4.4.

10. Define a procedure named \texttt{flatten} that takes a list as an argument and that returns the list obtained by eliminating internal list structures. For example, the call of:
    \((\texttt{flatten '}(1 2 a (b c (d e (f)) g) ()) () 13))\) should return \((1 2 a b c d e f g 13)\).
1.

Recall that the correct definition is:

```
(define (fact n) (if (= n 0) 1 (* n (fact (- n 1)))))
```

The errors are as follows:

a.  (define (fact n) (if (= n 0) (1) (* n (fact (- n 1)))))
   (1) is not a function

b.  (define (fact n) (if = n 0 1 (* n (fact (- n 1)))))
   the if has 5 arguments

c.  (define fact (n) (if (= n 0) 1 (* n (fact (- n 1)))))
   bad define with 3 arguments instead of 2

d.  (define (fact n) (if (= n 0) 1 (* n fact (- n 1))))
   the call on * includes fact as if it were a number

e.  (define (fact n) (if (= n 0) 1 (* n ((fact) (- n 1)))))
   (fact) is a bad call

2.  This information can be found in the R5RS standard:

   a.  For the question about comments, go to the index and look up "comment" to find that anything
       after a semi-colon is considered a comment.

   b.  Scheme has only single-line comments.

   c.  In evaluating, (/ a b c d), the standard says "associating to the left", which means it is
       evaluated as, (((a / b) / c) / d).

   d.  Looking through the index for things that begin with "string", you'll find a function string<?
       which you can call by saying, (string<? "hello" "there").

   e.  You can sort a list of integers with an expression such as, (sort '(1 5 2 7 4 8 3) <).

3.

```
(define (days-in-month m)
  (cond ((or (= m 9) (= m 4) (= m 6) (= m 11)) 30)
        ((= m 2) 28)
        (else 31)))
```

4.

```
(define (pow x y)
  (if (= 0 y) 1
      (* x (pow x (- y 1)))))
```

5.

```
(define (sum-to n)
  (if (= 1 n) 1
      (+ (/ 1 n) (sum-to (- n 1)))))
```
6. 
(define (sum lst)
  (if (null? lst) 0
      (+ (car lst) (sum (cdr lst)))))

7. 
(define (stutter lst)
  (if (null? lst)
      ()
      (cons (car lst) (cons (car lst) (stutter (cdr lst))))))

8. 
(define (multiples n m)
  (define (explore i)
    (if (> i n)
      ()
      (cons (* i m) (explore (+ i 1))))
  (explore 1))

9. 
;a) 
(define (positive-sum lst)
  (cond ((null? lst) 0)
        ((>= (car lst) 0) (+ (car lst) (positive-sum (cdr lst))))
        (else (positive-sum (cdr lst)))))
;b) (ignoring non-numbers)
(define (positive-sum lst)
  (cond ((null? lst) 0)
        ((and (number? (car lst)) (>= (car lst) 0))
         (+ (car lst) (positive-sum (cdr lst))))
        (else (positive-sum (cdr lst)))))

10. 
(define (flatten lst)
    (cond ((null? lst) ()))
         ((list? (car lst))
          (append (flatten (car lst)) (flatten (cdr lst))))
         (else (cons (car lst) (flatten (cdr lst))))))