Instructions

Form a small group. Start on problem 1.1. Check off with a staff member or discuss your solution process with a nearby group when you think everyone in your group understands how to solve problem 1.1. Then repeat for 1.2, 1.3, ... You may not move to the next problem until you check off or discuss with another group and everyone understands why the solution is what it is. You may use any course resources at your disposal: the purpose of this review session is to have everyone learning together as a group.

1 Scheme

1.1 What would Scheme display?

(a) > '(1 2 3)

(b) > '(1 . (2 . (3 . ()))))

(c) > '((1 . 2) . 3) 4 . (5 . 6))

(d) > (cons 1 2)

(e) > (cons 2 '())

(f) > (cons 1 (cons 2 '()))

(g) > (cons 1 (cons 2 3))

(h) > (cons (cons (car '(1 2 3))

        (list 2 3 4))

        (cons 2 3))

(i) > (car (cdr (car '((1 2) 3 (4 5))))))

(j) > (caddr '((1 2) 3 (4 5))))
1.2 Define sixty-ones. Return the number of times that 1 follows 6 in the list.

> (sixty-ones '(4 6 1 6 0 1))
1
> (sixty-ones '(1 6 1 4 6 1 6 0 1))
2
> (sixty-ones '(6 1 6 1 4 6 1 6 0 1))
3

1.3 Identify the bug(s) in this program.

> (sum-every-other '(1 2 3))
4
> (sum-every-other '())
0
> (sum-every-other '(1 2 3 4))
4
> (sum-every-other '(1 2 3 4 5))
9

(define (sum-every-other lst)
  (cond ((null? lst) lst)
        (else (+ (cdr lst)
                 (sum-every-other (caar lst)))))

1.4 (a) Implement add-to-all.

> (add-to-all 'foo '((1 2) (3 4) (5 6)))
((foo 1 2) (foo 3 4) (foo 5 6))

(b) Rewrite add-to-all tail-recursively.
1.5 Define sublists. Hint: use add-to-all.

```scheme
> (sublists '(1 2 3))

(() (3) (2) (2 3) (1) (1 3) (1 2) (1 2 3))
```

1.6 (a) Define append. In Scheme, append takes in two lists and returns a larger list.

```scheme
> (append '(1 2 3) '(4 5 6))

(1 2 3 4 5 6)
```

(b) Define reverse. Hint: use append.

```scheme
> (reverse '(1 2 3))

(3 2 1)
```

(c) Define reverse tail-recursively. Hint: use a helper function and cons.


2 Interpreters

2.1 Circle the number of calls to scheme_eval and scheme_apply for the code below.

(a) scm> (+ 1 2)
   
   \[ \begin{array}{llll}
   \text{scheme_eval} & 1 & 3 & 4 & 6 \\
   \text{scheme_apply} & 1 & 2 & 3 & 4 \\
   \end{array} \]

(b) scm> (if 1 (+ 2 3) (/ 1 0))
   
   \[ \begin{array}{llll}
   \text{scheme_eval} & 1 & 3 & 4 & 6 \\
   \text{scheme_apply} & 1 & 2 & 3 & 4 \\
   \end{array} \]

(c) scm> (or #f (and (+ 1 2) 'apple) (- 5 2))
   apple
   
   \[ \begin{array}{llll}
   \text{scheme_eval} & 6 & 8 & 9 & 10 \\
   \text{scheme_apply} & 1 & 2 & 3 & 4 \\
   \end{array} \]

(d) scm> (define (add x y) (+ x y))
   add
   scm> (add (- 5 3) (or 0 2))
   2
   
   \[ \begin{array}{llll}
   \text{scheme_eval} & 12 & 13 & 14 & 15 \\
   \text{scheme_apply} & 1 & 2 & 3 & 4 \\
   \end{array} \]

2.2 Identify the number of calls to scheme_eval and the number of calls to scheme_apply.

(a) scm> (define pi 3.14)
   pi
   scm> (define (hack x)
   (cond
      ((= x pi) pwned)
      ((< x 0) (hack pi))
      (else (hack (- x 1))))))
   hack
   
   (b) scm> (hack 3.14)
   pwned

(c) scm> ((lambda (x) (hack x)) 0)
   pwned
3 Streams

class Stream:
    """A lazily computed linked list."""

    class empty:
        """The empty stream."""

    empty = empty()

def __init__(self, first, compute_rest=empty):
    """A stream whose first element is FIRST and whose tail is either a stream or stream-returning parameterless function COMPUTE_REST."""

@property
def rest(self):
    """Return the rest of the stream, computing it if necessary."""

3.1 (a) What are the advantages or disadvantages of using a stream over a linked list?

(b) What’s the maximum size of a stream?

(c) What’s stored in first and rest? What are their types?

(d) When is the next element actually calculated?

3.2 Implement unique_stream which takes in a stream s and returns a new stream that contains only the unique elements of the input stream in the original order. Assume that s is finite.

def unique_stream(s):
After more than 100 years of operation, the Ringling Bros. circus is closing. A victory for animal rights advocates, the circus’ closure poses a challenge for the zoologists tasked with moving the circus’ animals to more suitable habitats.

The zoologists must first take the animals in a freight elevator with a weight limit of 2000. In order to speed up the process, the zoologists prefer to take groups of animals of the same species in the elevator, rather than one animal at a time.

Assume the zoologists will only put all of the animals of a particular species in the elevator, or take animals of that particular species one at a time.

You have access to the table animals, with columns containing the animals’ names, weights, and species.

4.1 Write a query that returns the collective weight and species of animals in a group where there is more than one animal of a particular species in a group, and the collective weight of the animals in the group is less than 2000.

Your query should yield the following result.

229 pig
1618 tiger
91 dog

4.2 To take the animals to their new habitats, the zoologists load the animals into trucks. The zoologists again want to take the animals in groups of the same species, but one of the trucks has a height limit of 5.0.

Write a query that returns the maximum height and species of animals in a group where the maximum height is less than 5.0. Your query may yield a species where there is only one animal of that particular species.

Your query should yield the following result.

4.1 pig
4 dog
4.9 zebra