TAIL RECURSION, INTERPRETERS, AND ITERATORS

COMPUTER SCIENCE MENTORS 61A

November 6 to November 10, 2017

1 Tail Recursion

1. What is a tail context/tail call? What is a tail recursive function?

2. Why are tail calls useful for recursive functions?

Answer the following questions with respect to the following function:

```
(define (sum-list lst)
  (if (null? lst)
    0
    (+ (car lst) (sum-list (cdr lst)))
  )
)
```

3. Why is sum-list not a tail call? Optional: draw out the environment diagram of this sum-list with list: (1 2 3). When do you add 2 and 3?

4. Rewrite sum-list in a tail recursive context.

2 Interpreters

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

```
(define (square x) (* x x))
(+ (square 3) (- 3 2))
```

scheme_eval	2	5	14	24
scheme_apply	1	2	3	4

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

```
scm> (+ 1 2)
3
```

scheme_eval	1	3	4	6
scheme_apply	1	2	3	4

```
scm> (if 1 (+ 2 3) (/ 1 0))
5
```

scheme_eval	1	3	4	6
scheme_apply	1	2	3	4

```
scm> (or #f (and (+ 1 2) 'apple) (- 5 2))
apple
```

scheme_eval	6	8	9	10
scheme_apply	1	2	3	4

```
scm> (define (add x y) (+ x y))
add
scm> (add (- 5 3) (or 0 2))
2
```

scheme_eval	12	13	14	15
scheme_apply	1	2	3	4

7. 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 Iterators

8. What is difference between an iterator and an iterable?