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:

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

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

<table>
<thead>
<tr>
<th>function</th>
<th>1st call</th>
<th>2nd call</th>
<th>3rd call</th>
<th>4th call</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme_eval</td>
<td>2</td>
<td>5</td>
<td>14</td>
<td>24</td>
</tr>
<tr>
<td>scheme_apply</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

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

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

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

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

(scm> (define (add x y) (+ x y))

(scm> (add (- 5 3) (or 0 2)))
```

<table>
<thead>
<tr>
<th>function</th>
<th>1st call</th>
<th>2nd call</th>
<th>3rd call</th>
<th>4th call</th>
<th>5th call</th>
<th>6th call</th>
<th>7th call</th>
<th>8th call</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme_eval</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>scheme_apply</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>scheme_eval</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>scheme_apply</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>scheme_eval</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
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
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?