1. Create a table called *Summer Movies* which contains the five rows of the table above.

2. Write a query to select the name and the rating of all live-action movies that are action movies and order them by rating.

3. Write a query to select the names of all movies which have the same genre. Make sure and get rid of duplicates.

4. Write a query to select all movies which score above a 7.0, ordered by their rating as well.
2 Tail Recursion

Consider the function sum-list:

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

1. Rewrite sum-list using tail recursion.

3 Streams

1. Why do we use streams? Why don’t we just use linked lists instead?

   Streams represented in scheme have very specific functions associated with them.
   Stream creation: `cons-stream`
   First element of a stream: `car`
   Rest of the stream: `stream-cdr`
   Empty Stream: `nil`
   To check for emptiness: `null?`

2. Define a function called `integers` that returns a stream of integers starting from `first`
3. What would Scheme Print?

```scheme
scm> (define ints (integers 1))

scm> (car (stream-cdr ints))

scm> (car ints)

scm> (car (stream-cdr (stream-cdr (stream-cdr ints))))

How many times did the stream have to compute a new value of rest for the last input?

```scheme
scm> (define s (cons-stream (car ints)
            (cons-stream (car (stream-cdr ints))
                        nil)))

scm> (stream-cdr s)
```
4. Write conditional_map_stream, a scheme function which goes through every element of a stream of numbers and returns a new stream which has either the original element if the function applied to the number was non-negative, or the value of the function applied to the original number otherwise.

```
scm> (define (f x) (- x 1))
f
scm> (define s (cons-stream 1
               (cons-stream 3
                   (cons-stream 12)))
s
scm> (define new (conditional_map_stream s f))
new
scm> (car new)
1
scm> (car (stream-cdr new))
2
```

(define (conditional_map_stream s f)
1. Define a set of facts to model the table of data below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Genre</th>
<th>Rating</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antman</td>
<td>Action</td>
<td>7.9</td>
<td>Live-action</td>
</tr>
<tr>
<td>Minions</td>
<td>Comedy</td>
<td>6.7</td>
<td>Animated</td>
</tr>
</tbody>
</table>

2. Write facts for odd-length, as shown below:
   logic> (odd-length (Minions are adorable))
   Success!
   logic> (odd-length (61a rocks))
   Failed

3. Write facts for reverse, a relation between two lists that is satisfied if and only if the second list is the reverse of the first list. Hint: use append (given below), which was defined in lecture.

   (fact (append () ?lst2 ?lst2))
   (fact (append (?elem . ?rest1) ?lst2 (?elem . ?rest2))
     (append ?rest1 ?lst2 ?rest2))