CSC324: Pattern Matching and Tail Recursion

1. Rewrite the recursive function `add1-list` using pattern matching

```
(define (add1-list lst)
  (if (null? lst)
    (list)
    (cons (+ 1 (first lst))
          (add1-list (rest lst)))))
```

Solution:

```
(define/match (add1-list lst)

  [(                )                                  ]

  [(                )                                  ])
```

2. Write a recursive function `list_sum` using pattern matching in Racket. Here is a Haskell implementation of the function:

```
list_sum []     = 0
list_sum (x:xs) = x + list_sum xs
```

Solution:

3. Rewrite this recursive function `factorial` to be tail recursive.

```
(define (factorial n)
    (if (equal? n 1)
        1
        (* n (factorial (- n 1)))))
```

4. Rewrite this recursive function `lst-double` to be tail recursive.

```
(define (lst-double lst)
    (if (empty? lst)
        '()
        (cons (* (first lst) 2)
              (lst-double (rest lst)))))
```

CSC324: Higher Order Functions and Currying

5. Implement the `foldl` function (Racket version). Is your implementation tail-recursive?

6. Write a function that takes a single argument `x`, and returns a new function that takes a list and checks whether `x` is in that list.

7. Implement `filter` using `foldl`