# CIS 61 - Lab 10 - Scheme

## Q1: Over or Under

Define a procedure `over-or-under` which takes in a number `x` and a number `y` and returns the following:

- -1 if `x` is less than `y`
- 0 if `x` is equal to `y`
- 1 if `x` is greater than `y`

```
(define (over-or-under x y)
    (cond
          ((> x y) 1)
          ((< x y) -1)
              (else 0))
)

;;; Tests
(over-or-under 1 2)
; expect -1
(over-or-under 2 1)
; expect 1
(over-or-under 1 1)
; expect 0
```

```
scm> (define (over-or-under x y)
    (cond
            ((> x y) 1)
            ((< x y) -1)
                (else 0))
...> )
over-or-under
scm> (over-or-under 1 2)
-1
scm> (over-or-under 2 1)
1
scm> (over-or-under 1 1)
0
```

# Q2: Filter

Write a procedure `filter-lst`, which takes a predicate `f` and a list `lst`, and returns a new list containing only elements of the list that satisfy the predicate. The output should contain the elements in the same order that they appeared in the original list.

```
(define (filter-lst f lst)
(cond ((null? lst) '())
      ((f (car lst)) (cons (car lst) (filter f (cdr lst))))
      (else (filter f (cdr lst)))))
)

;;; Tests
(define (even? x)
  (= (modulo x 2) 0))
(filter-lst even? '(0 1 1 2 3 5 8))
; expect (0 2 8)
```

```
scm> (define (filter f lst)
  (cond ((null? lst) '())
        ((f (car lst)) (cons (car lst) (filter f (cdr lst))))
        (else (filter f (cdr lst)))))
filter
scm> (define (even? x)
...>   (= (modulo x 2) 0))
even?
scm> (filter even? '(0 1 1 2 3 5 8))
(0 2 8)
```

# Q3: Make Adder

Write the procedure `make-adder` which takes in an initial number, `num`, and then returns a procedure. This returned procedure takes in a number `x` and returns the result of `x + num`.

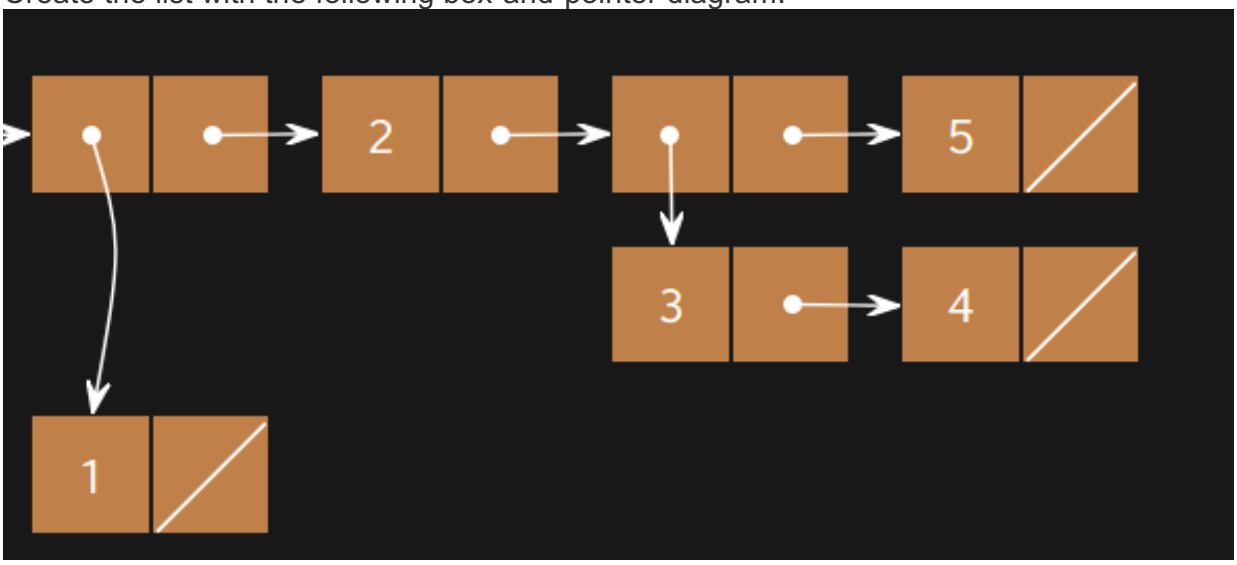
```
(define (make-adder num)
(lambda (x)(+ x num)))
)

;;; Tests
(define adder (make-adder 5))
(adder 8)
; expect 13
```

```
scm> (define (make-adder num)
...>   (lambda (x) (+ x num)))
make-adder
scm> (define adder (make-adder 5))
adder
scm> (adder 8)
13
```

# Q4: Make a List

Create the list with the following box-and-pointer diagram:




```
(define lst
 (cons (cons 1 nil)
       (cons 2
             (cons (cons 3 (cons 4 nil))
                   (cons 5 nil))))

)
```

```
scm> (define lst
  (cons (cons 1 nil)
        (cons 2
              (cons (cons 3 (cons 4 nil))
                    (cons 5 nil))))
...> )
lst
scm>
```

# Q5: Compose

Write the procedure `composed`, which takes in procedures `f` and `g` and outputs a new procedure. This new procedure takes in a number `x` and outputs the result of calling `f` on `g` of `x`.

```
(define (composed f g)
(lambda(x)(f(g x)))
)
```

```
scm> (define (composed f g)
  (lambda (x) (f (g x)))
...> )
composed
scm>
```

# Q6: Remove

Implement a procedure `remove` that takes in a list and returns a new list with *all* instances of `item` removed from `lst`. You may assume the list will only consist of numbers and will not have nested lists.
*Hint*: You might find the `filter` procedure useful.

```
(define (remove item lst)
  (filter (lambda (x) (not (= x item))) lst))
)

;;; Tests
(remove 3 nil)
; expect ()
(remove 3 '(1 3 5))
; expect (1 5)
(remove 5 '(5 3 5 5 1 4 5 4))
; expect (3 1 4 4)
```

```
scm> (define (remove item lst)
...>   (filter (lambda (x) (not (= x item))) lst))
remove
scm> (remove 3 nil)
()
scm> (remove 3 '(1 3 5))
(1 5)
scm> (remove 5 '(5 3 5 5 1 4 5 4))
(3 1 4 4)
```

# Q7: No Repeats

Implement `no-repeats`, which takes a list of numbers `s` as input and returns a list that has all of the unique elements of `s` in the order that they first appear, but no repeats. For example, `(no-repeats (list 5 4 5 4 2 2))` evaluates to `(5 4 2)`.

*Hints*: To test if two numbers are equal, use the `=` procedure. To test if two numbers are not equal, use the `not` procedure in combination with `=`. You may find it helpful to use the `filter` procedure.

```
(define (no-repeats s)
(if (null? s)
    s
    (cons (car s)
      (no-repeats (filter (lambda (x) (not (= (car s) x))) (cdr
s))))))
)
;;; Tests
(no-repeats (list 5 4 5 4 2 2))
; expect (5 4 2)
```

```
scm> (define (no-repeats s)
  (if (null? s)
        s
        (cons (car s)
...>           (no-repeats (filter (lambda (x) (not (= (car s) x))) (cdr s))))))
no-repeats
scm> (no-repeats (list 5 4 5 4 2 2))
(5 4 2)
scm>
```

# Q8: Substitute

Write a procedure `substitute` that takes three arguments: a list `s`, an `old` word, and a `new` word. It returns a list with the elements of `s`, but with every occurrence of `old` replaced by `new`, even within sub-lists.

*Hint*: The built-in `pair?` predicate returns True if its argument is a `cons` pair.

*Hint*: The `=` operator will only let you compare numbers, but using `equal?` or `eq?` will let you compare symbols as well as numbers. For more information, check out the Scheme Built-in Procedure Reference.

```
(define (substitute s old new)
  (cond
   ((null? s) s)
   ((equal? (car s) old)
      (cons new (substitute (cdr s) old new)))
   ((pair? (car s))
      (cons (substitute (car s) old new) (substitute (cdr s) old
new)))
   (else
      (cons (car s) (substitute (cdr s) old new)))))

)
```

```
scm> (define (substitute s old new)
  (cond
   ((null? s) s)
   ((equal? (car s) old)
      (cons new (substitute (cdr s) old new)))
   ((pair? (car s))
      (cons (substitute (car s) old new) (substitute (cdr s) old new)))
   (else
...>       (cons (car s) (substitute (cdr s) old new)))))
substitute
```

# Q9: Sub All

Write `sub-all`, which takes a list `s`, a list of `old` words, and a list of `new` words; the last two lists must be the same length. It returns a list with the elements of `s`, but with each word that occurs in the second argument replaced by the corresponding word of the third argument. You may use `substitute` in your solution. Assume that `olds` and `news` have no elements in common.

```
(define (sub-all s olds news)
  (cond
   ((null? olds) s)
   (else (sub-all (substitute s (car olds) (car news)) (cdr olds)
(cdr news))))
)
```

```
scm> (define (sub-all s olds news)
  (cond
   ((null? olds) s)
   (else (sub-all (substitute s (car olds) (car news)) (cdr olds) (cdr news)))))
...> )
sub-all
```