



# COM S 342

Recitation 10/7/2019 –  
10/9/2019

Topic

○ FuncLang programming

○ Q&A



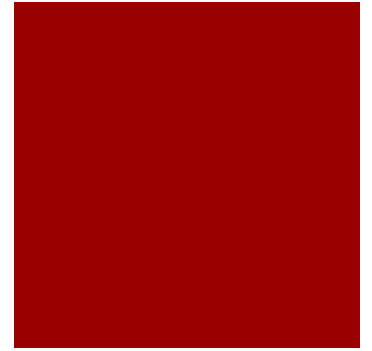
# FuncLang

- Recursion

- Higher Order Functions



# Recursion Examples



## ○ Subsum

- Given a list of integers (list  $n_1 n_2 \dots n_k$ )
- Return a list of integers (list  $r_1 r_2 \dots r_{k-1}$ ) where  $r_1 = n_1 + n_2$ ,  $r_2 = n_2 + n_3$ ,  $\dots$ ,  $r_{k-1} = n_{k-1} + n_k$
- Return 0 if  $k < 2$

## ○ Expand List

- Given one list, whose elements are lists (called sublists)
- Return a list whose elements are the elements of sublists

# Subsum



```
(define subsum
  (lambda (lst)
    (if (null? (cdr lst))
        0
        (subsumhelp lst)
    )
  )
)

(define subsumhelp
  (lambda (lst)
    (if (null? (cdr (cdr lst)))
        (list (+ (car lst) (car (cdr lst))))
        (cons (+ (car lst) (car (cdr lst))) (subsumhelp (cdr lst)))
    )
  )
)
```

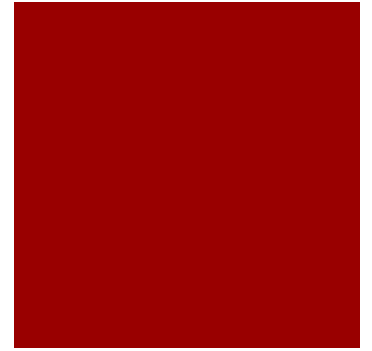
# Expand Lists

- The last sublist is the basement
- Append the elements of  $n-1$ st sublist
- Recursively do step 2 until all sublists are expanded

Given:

```
(define append
  (lambda (lst1 lst2)
    (if (null? lst1)
        lst2
        (if (null? lst2)
            lst1
            (cons (car lst1) (append (cdr lst1) lst2))
        )
    )
  )
)
```

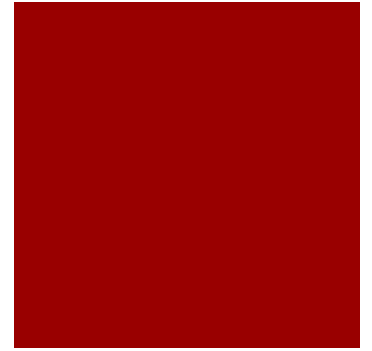
# Expand Lists



- The last sublist is the basement
- Append the elements of  $n-1$ st sublist
- Recursively do step 2 until all sublists are expanded

```
(define expand
  (lambda (lst)
    (if (null? lst)
        (list)
        (if (null? (cdr lst))
            (car lst)
            (append (car lst) (expand (cdr lst))))))
  )
)
```

# Sum N



○  $result = (+\ N\ (N-1)\ \dots\ 2\ 1)$

```
(define sum
  (lambda (n)
    (if (= n 1)
        1
        (+ n (sum (- n 1)))))
  )
)
```



# Sum N



○ *result* = (+ 1 2 ... (-N 1) N)

```
(define sum
  (lambda (n)
    (if (= n 1)
        1
        (help 1 n))
  )
)

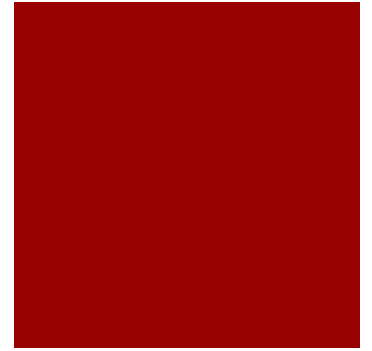
(define help
  (lambda (x n)
    (if (< x n)
        (+ x (help (+ 1 x) n))
        n)
  )
)
)
```

# Higher Order Function



- Repeat a transformation function on an object  $n$  times
  - $(\text{repeat } f \ n \ o)$ , where
    - $f$  is the function to be applied
    - $n$  is the number of times
    - $o$  is the object (a number)

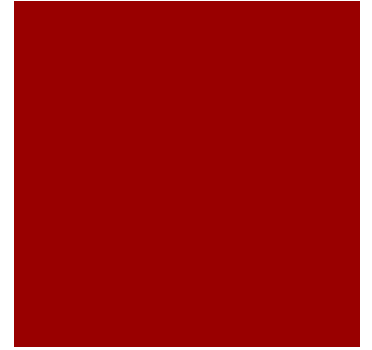
# Repeat Transformations



```
(define repeat
  (lambda (f n o)
    (if (= n 0)
        o
        (repeat f (- n 1) (f o)))
    )
  )
)
```

```
(define double (lambda (x) (* 2 x)))
```

# Repeat Transformations



(repeat double 3 1)

0: (repeat double 2 (double 1))

1:(repeat double 1 (double (double 1)))

2: (repeat double 0 (double (double (double 1))))

3: (double (double (double 1)))

4: (double (double 2))

5: (double 4)

6: 8

Q&A

