

Putting the List Processing Template to Work





From last lecture: List Template

```
;; f : list-of-symbols -> ...  
;; (define (f x)  
;;   (cond  
;;     [(empty? x) ...]  
;;     [else ... (first x) ...  
;;               ... (f (rest x)) ...]))
```



Plan for Today

- Practice with the list template
- Consuming and producing lists
 - “Batch processing”
- Consuming lists of structures
- Correct template for list of structures



Some Example List Functions

- **sum:** list-of-numbers -> number
- **count:** list-of-symbols -> number
- **replace:** symbol symbol list-of-symbols
-> list-of-symbols
- **scale:** number list-of-numbers
-> list-of-numbers
- **close-by:** list-of-posn -> list-of-posn
- **distances:** list-of-posn -> list-of-numbers



Implementing scale: Data Definition

```
;; A list-of-numbers is either
;;   - empty, or
;;   - (cons n l)
;; where
;;   n is a number and
;;   l is a list-of-numbers
```



Implementing scale: Function Definition

```
;; scale: number list-of-numbers
;;          -> list-of-numbers
;; Purpose: To scale each number in the
;;          given list by the given number.
;; Examples: (scale 2 empty) = empty
;; (scale 3 (cons 1 (cons 2 empty)))
;; = (cons 3 (cons 6 empty))
```



Implementing scale: Code

```
;; scale: number list-of-numbers
;;      -> list-of-numbers
(define (scale n lon)
  (cond
    [(empty? lon) empty]
    [else (cons (* n (first lon))
                 (scale n
                       (rest lon)))]))
```



Implementing close-by: Data Definition

```
;; A list-of-posns is either
;; - empty
;; - (cons p l)
;; where
;; p is a posn and
;; l is a list-of-posn
```



Implementing close-by: Template for List of Posns (first attempt)

```
;; f : list-of-posns -> ...
;; (define (f x)
;;   (cond
;;     [(empty? x) ...]
;;     [else ... (posn-x (first x)) ...
;;              ... (posn-y (first x)) ...
;;              ... (f (rest x)) ...]))
```



Implementing close-by: Function Definition

```
;; close-by: list-of-posn -> list-of-posn
;; Purpose: To accept a list of posns and
;;         return a list containing the posns
;;         that lie within the unit circle
;; Examples: (close-by empty) => empty
;; (close-by (cons (make-posn 1 1)
;;                (cons (make-posn 1/2
;;                       1/2)
;;                      empty)))
;; = (cons (make-posn 1/2 1/2) empty)
```



Implementing close-by: Code

```
;; close-by: list-of-posns -> list-of-posns
(define (close-by lop)
  (cond
    [(empty? lop) empty]
    [else (cond
              [(<= (sqrt (+ (sqr (posn-x (first lop)))
                            (sqr (posn-y (first lop))))) 1)
               (cons (first lop) (close-by (rest lop)))]
              [else (close-by (rest lop))])])])
```



Implementing close-by: Template for List of Posns (corrected back)

```
;; f : list-of-posns -> ...  
;; (define (f x)  
;;   (cond  
;;     [(empty? x) ...]  
;;     [else ... (first x)  
;;               ... (f (rest x)) ...]))
```



Implementing close-by: Code

```
;; close-by: list-of-posns -> list-of-posns
(define (close-by lop)
  (cond
    [(empty? lop) empty]
    [else (cond
              [(<= (sqrt (+ (sqr (posn-x (first lop)))
                            (sqr (posn-y (first lop))))) 1)
               (cons (first lop) (close-by (rest lop)))]
              [else (close-by (rest lop))])])])
```



Implementing close-by: Code

```
;; close-by: list-of-posns -> list-of-posns
(define (close-by lop)
  (cond
    [(empty? lop) empty]
    [else (cond
             [(<= (distance-to-0 (first lop)) 1)
              (cons (first lop) (close-by (rest lop)))]
             [else (close-by (rest lop))])]))
```



For Next Class

- Homework due Monday
- Reading:
 - Chapter 11 and companion notes
 - Natural numbers: A recursive subset of numbers
- Quiz:
 - Chapter 11