6.001 SICP

- Pairs and lists
- Common list operations
- Higher-order list procedures (if there is time)

Pairs (cons cells)
- \((\text{cons } x\text{-exp} \ y\text{-exp}) \rightarrow \text{Pair}\); type: \(x, x\rightarrow\) Pair
  - Where \(x\text{-exp}\) evaluates to a value \(x\text{-val}\), and \(y\text{-exp}\) evaluates to a value \(y\text{-val}\)
  - Returns a pair \(\text{Pair}\) whose
    car-part is \(x\text{-val}\) and whose
cdr-part is \(y\text{-val}\)
- \((\text{car } \text{Pair}) \rightarrow \text{Pair}\); type: Pair\(\rightarrow\) Pair\(\rightarrow\) Pair
  - Returns the car-part of the pair \(\text{Pair}\)
- \((\text{cdr } \text{Pair}) \rightarrow \text{Pair}\); type: Pair\(\rightarrow\) Pair\(\rightarrow\) Pair
  - Returns the cdr-part of the pair \(\text{Pair}\)

Pair - Box and pointer diagram

Lists - Box and pointer diagram

Printed representation

Cons, car, cdr
Cons, car, cdr

(define thing (cons (cons 1 nil) (cons 2 (cons 3 nil))))
thing ==>
((1) 2 3)
(cons 1 nil) ==> (1)
(cons 1 (cons 2 nil)) ==> (1 2)
(car thing) ==> (1)
(car thing) ==> (2 3)
(car (car thing)) ==> 1
(car (cdr (car thing))) ==> 3

cdr'ing down a list

(define (list-ref lst n)
  (if (= n 0)
    (car lst)
    (list-ref (cdr lst) (- n 1)))

append two lists

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

More list drill

x => (() )
y => (1 2 3 )
z => (1 (2 3 ) (4 ) )
w => (1 2 3 4 5 )
(length x)
(length y)
(length z)
(list-ref z 2)
(append x y)
(cons x y)
More list drill

x => ((()))
y => ((1 2 3))
z => ((1 (2 3)) ((4)))
w => ((1 2 3 4 5))

(length x)
1
(length y)
3
(length z)
3
(list-ref z 2)
((4))
(append x y)
((()) 1 2 3)
(consp x y)
(((()) 1 2 3) 6.001 SICP 13

Writing some procedures

The procedure copy-some that copies the first n elements of a list
(copy-some 3 (list 1 2 3 4 5)) => (1 2 3)

(define (copy-some n lst)

  (if (= n 0)
    nil
    (cons (car lst)
      (copy-some (- n 1) (cdr lst))))
)

• What is the complexity of this procedure? Θ(n)

Orders of Growth

What is the order of growth of last-k?

(define (last-k k lst)
  (if (= (length lst) k)
    lst
    (last-k k (cdr lst))))

Θ( n² )

Filtering a List

(define (filter pred lst)
  (cond ((null? lst) nil)
        ((pred (car lst))
         (cons (car lst)
           (filter pred (cdr lst))))
        (else (filter pred (cdr lst))))

Accumulating Results

(define (accumulate op init lst)
  (if (null? lst)
    init
    (op (car lst)
      (accumulate op init
        (cdr lst))))

6.001 SICP 19
Higher-Order List Procedures

(define (map proc lst)
  (if (null? lst)
      nil
      (cons (proc (car lst))
            (map proc (cdr lst))))))

(define (square-list lst)
  (map square lst))

(square-list (list 1 2 3)) == (1 4 9)

(map square (1 2 3))
(cons (square 1) (map square (2 3)))
(cons (square 1) (cons (square 2) (map square (3))))
(cons (square 1) (cons (square 2) (cons (square 3) nil)))

Lists and higher-order procedures

x => (( ))
y => (1 2 3)
z => (1 (2 3) (4))
w => (1 2 3 4 5)

(map (lambda (x) (cons x nil)) y)
(map inc w)
(filter odd? w)
(map inc (filter odd? w))
(filter odd? (map inc w))
(accumulate + 0 (map inc (filter odd? w)))