CS3:
Introduction to Symbolic Programming

Lecture 7:
The last of recursion (almost)

Fall 2007

Nate Titterton
nate@berkeley.edu

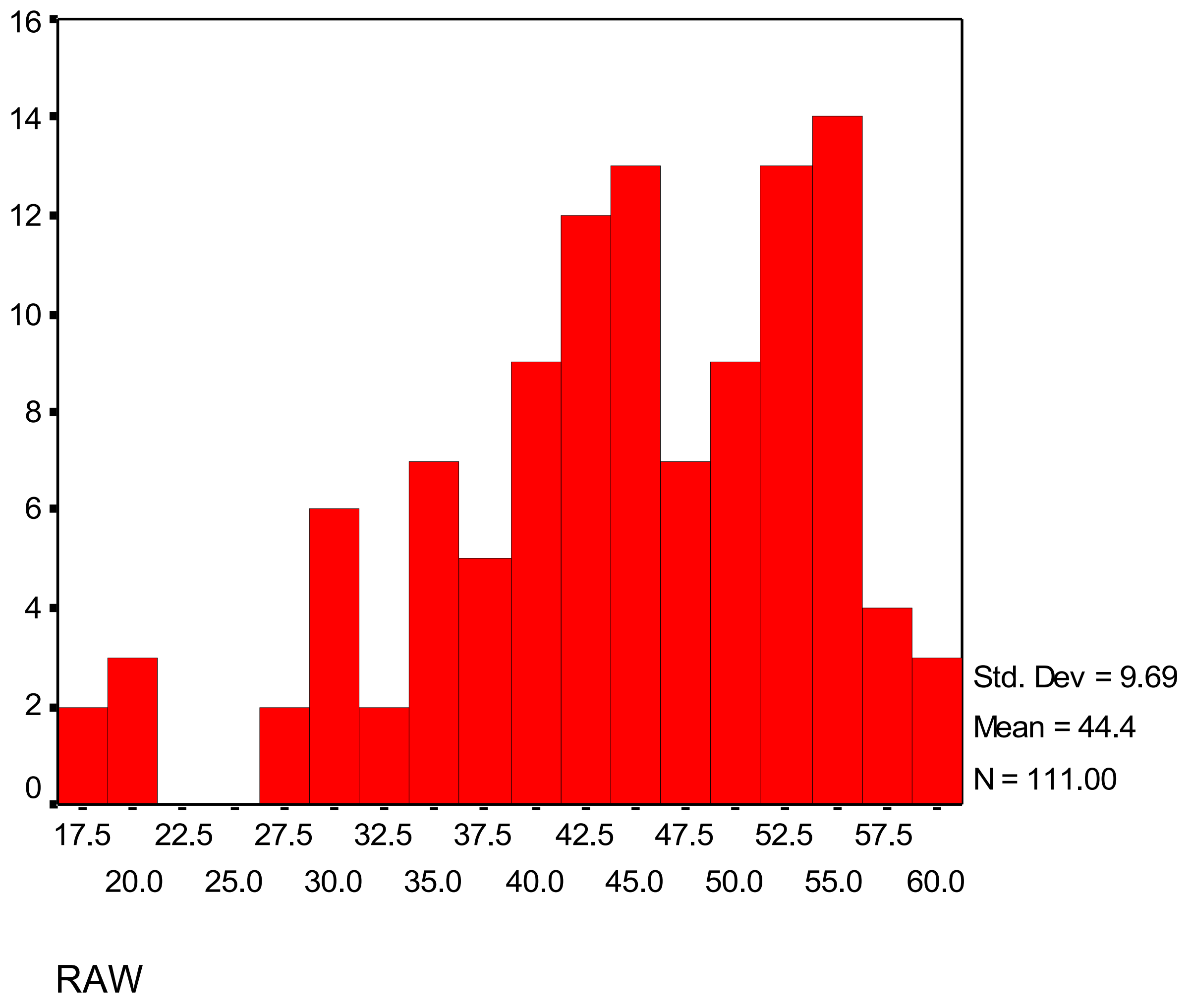
Schedule

6	Oct 1-4	Lecture: *Midterm 1* Lab: Recursion with multiple arguments Reading: Simply Scheme ch. 14
7	Oct 8-12	Lecture: Advanced Recursion Lab: Advanced Recursion Miniproject 2: Number spelling
8	Oct 15-19	Lecture: Higher Order Functions Lab: Introduction to HOF Reading: Simply Scheme, Ch 9, 10 "DbD" HOF version Note: MP#2 due Tue/Wed
9	Oct 22-26	Lecture: HOF Lab: Advanced HOF, Tic Tac Toe

Announcements

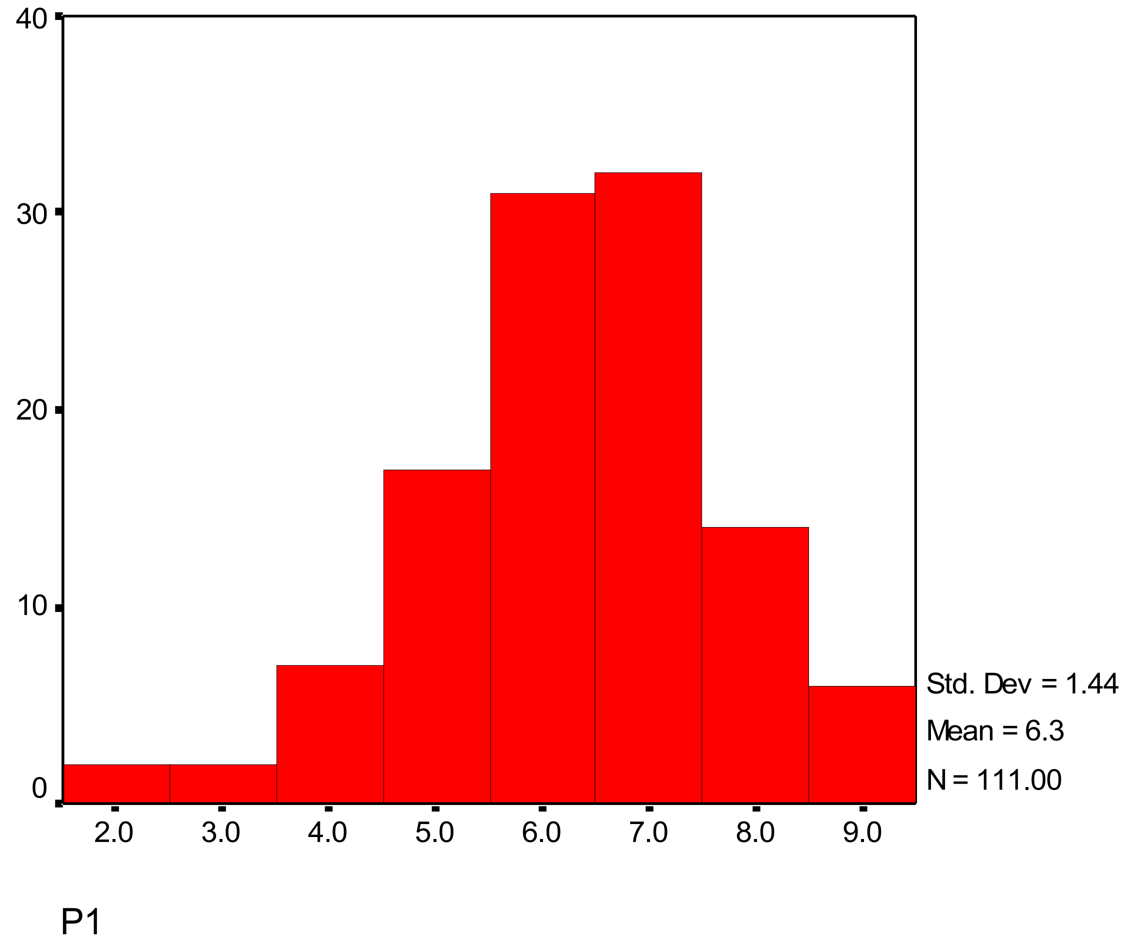
- Nate's office hours:
  - Wed, 2-4, 329 Soda
- Interviews:
  - A graduate class is doing usability analyses of UCWISE, and a bit of your time would really help

Midterm 1



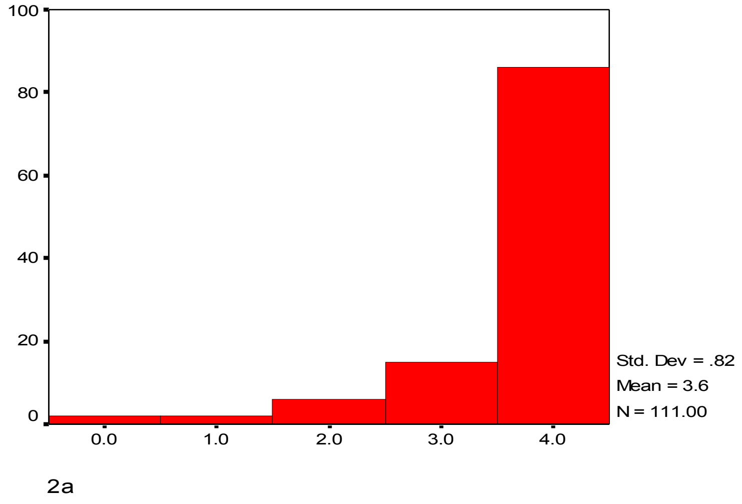
16
14
12
10
8
6
4
2
0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5
55.0
57.5
60.0
RAW
Std. Dev = 9.69
Mean = 44.4
N = 111.00

Question 1: fill in the blanks

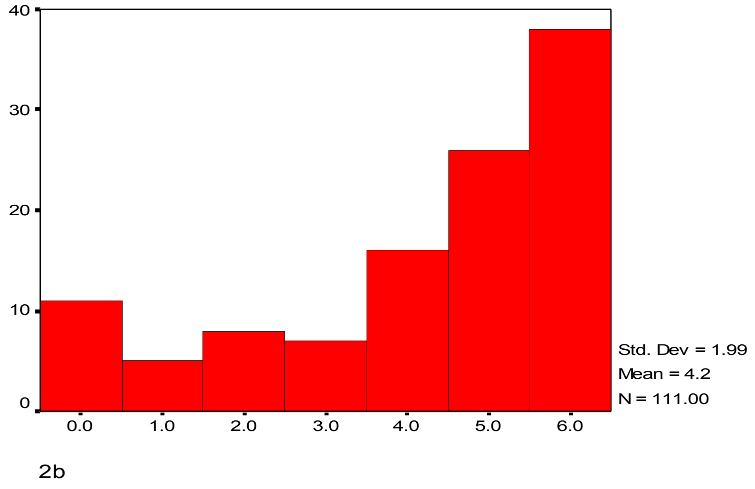


Question 2a/b: rating movies

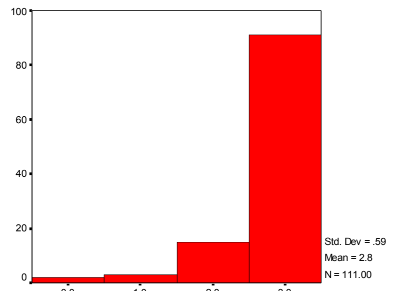
– can-watch?



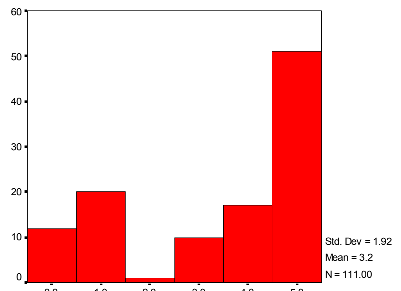
– best-movie



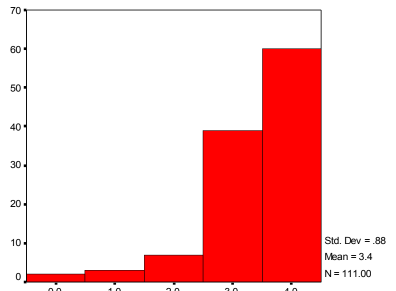
Q3: lights-out



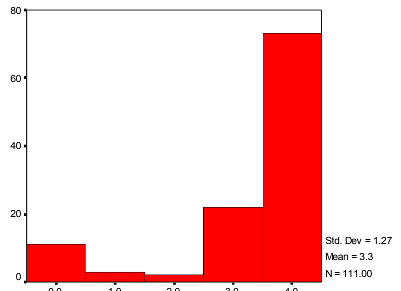
3a



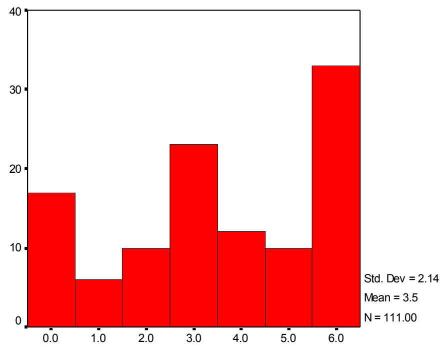
3b



3c



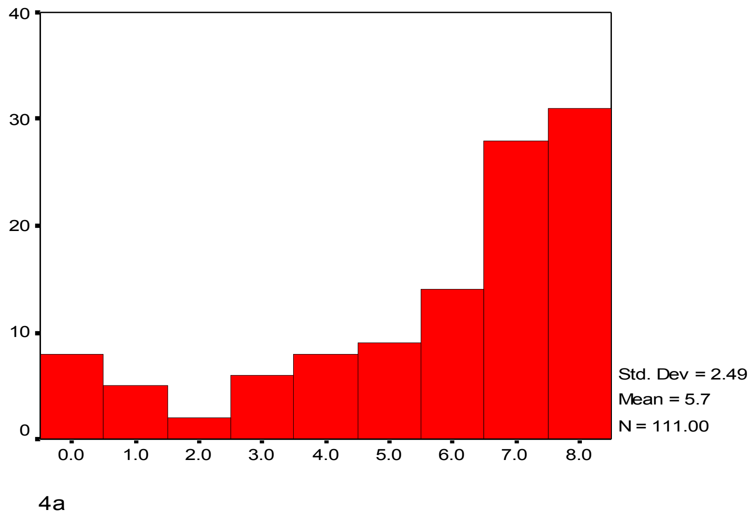
3d



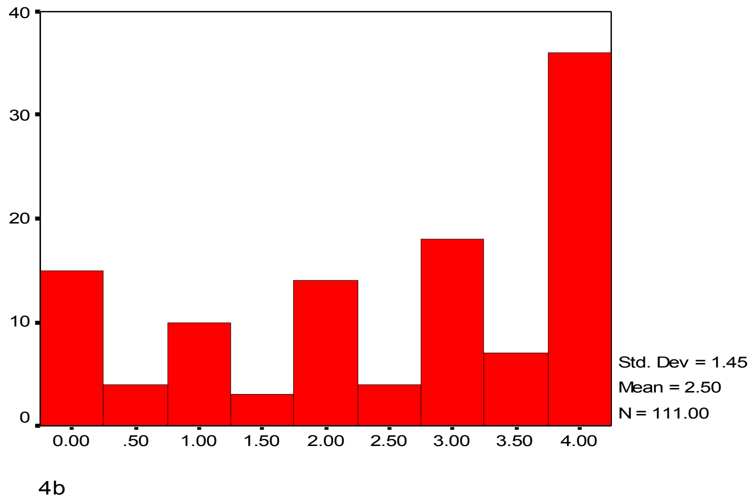
3e

Q4: Dbd with recursion

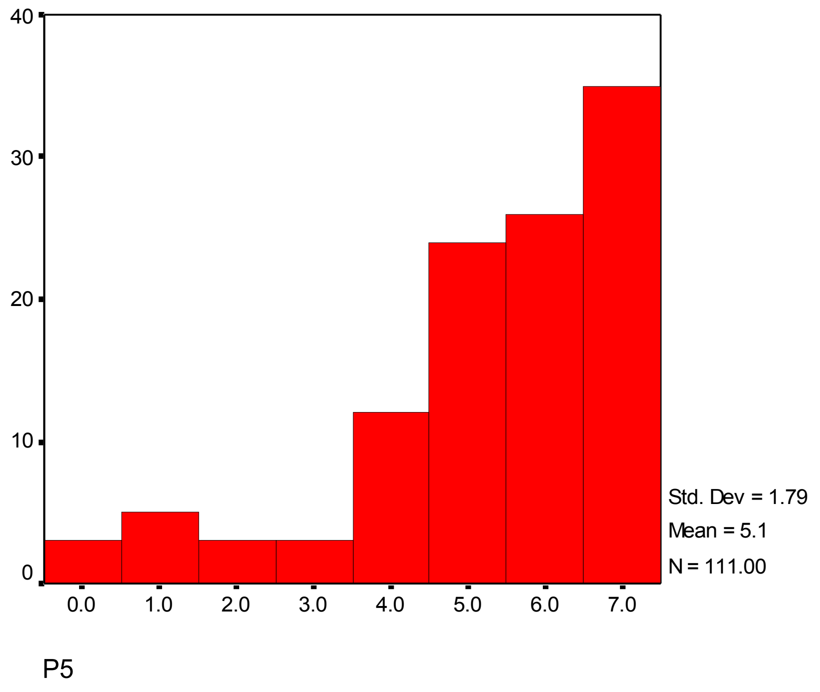
- `day-by-day` fill in the blanks



- test cases



Q5: 2ltr-words



Number Spelling (Miniproject #2)

- A program to write out names of almost any number
- Read *Simply Scheme*, page 233, which has hints
- Another hint (principle): don't force "everything" into the recursion.
  - Special/border cases may be easier to handle before you send yourself into a recursion

Goodbye recursion?

- Nope. We'll do more with recursion later

- What have we done in the last few weeks?
  - Work with roman numerals
  - "Advanced recursions": ones that work on multiple sentences, or do more than one thing at a time
  - `zip`, `merge`, `my-equal?`
  - Recursive patterns (map, filter, etc)
  - Sorting (insertion sort)
  - Accumulating recursion (e.g., using `so-far`)

`roman-sum-helper` (from lab)

Write roman-sum-helper:

```
(define (roman-sum number-sent)
  (if (empty? number-sent)
      0
      (roman-sum-helper (first number-sent)
                        (bf number-sent)
                        (first number-sent)) ) )
```

Roman-sum-helper takes three arguments:

```
(define (roman-sum-helper so-far number-list most-
 recent) ... )
```

`(roman-sum '(100 10 50 1 5))` **will recurse with:**

```
(roman-sum-helper 100 '(10 50 1 5) 100)
(roman-sum-helper 110 '(50 1 5) 10)
(roman-sum-helper 140 '(1 5) 50)
(roman-sum-helper 141 '(5) 1)
(roman-sum-helper 144 '( ) 5)
```

Accumulating or “tail”

- Accumulating recursions are sometimes called "tail" recursions (by TAs, me, etc).
  - But, not all recursions that keep track of a number are "tail" recursions.

Tail versus “embedded” recursions

- A <u>tail</u> recursion has no combiner, so it can end as soon as a base case is reached
  - Compilers can do this efficiently

- An <u>embedded</u> recursion needs to combine up all the recursive steps to form the answer
  - The poor compiler has to keep track everything

Tail or embedded? (1/3)

```
(define (my-count sent)
   (if (empty? sent)
      0
      (+ 1 (my-count (bf sent)))))
```

Embedded!

```
(my-count '(a b c d)) →
  (+ 1 (my-count '(b c d)))
  (+ 1 (+ 1 (my-count '(c d))))
  (+ 1 (+ 1 (+ 1 (my-count '(d)))))
  (+ 1 (+ 1 (+ 1 (+ 1 (my-count '()))))))
  (+ 1 (+ 1 (+ 1 (+ 1 0))))
  (+ 1 (+ 1 (+ 1 1)))
  (+ 1 (+ 1 2))
  (+ 1 3)
  4
```

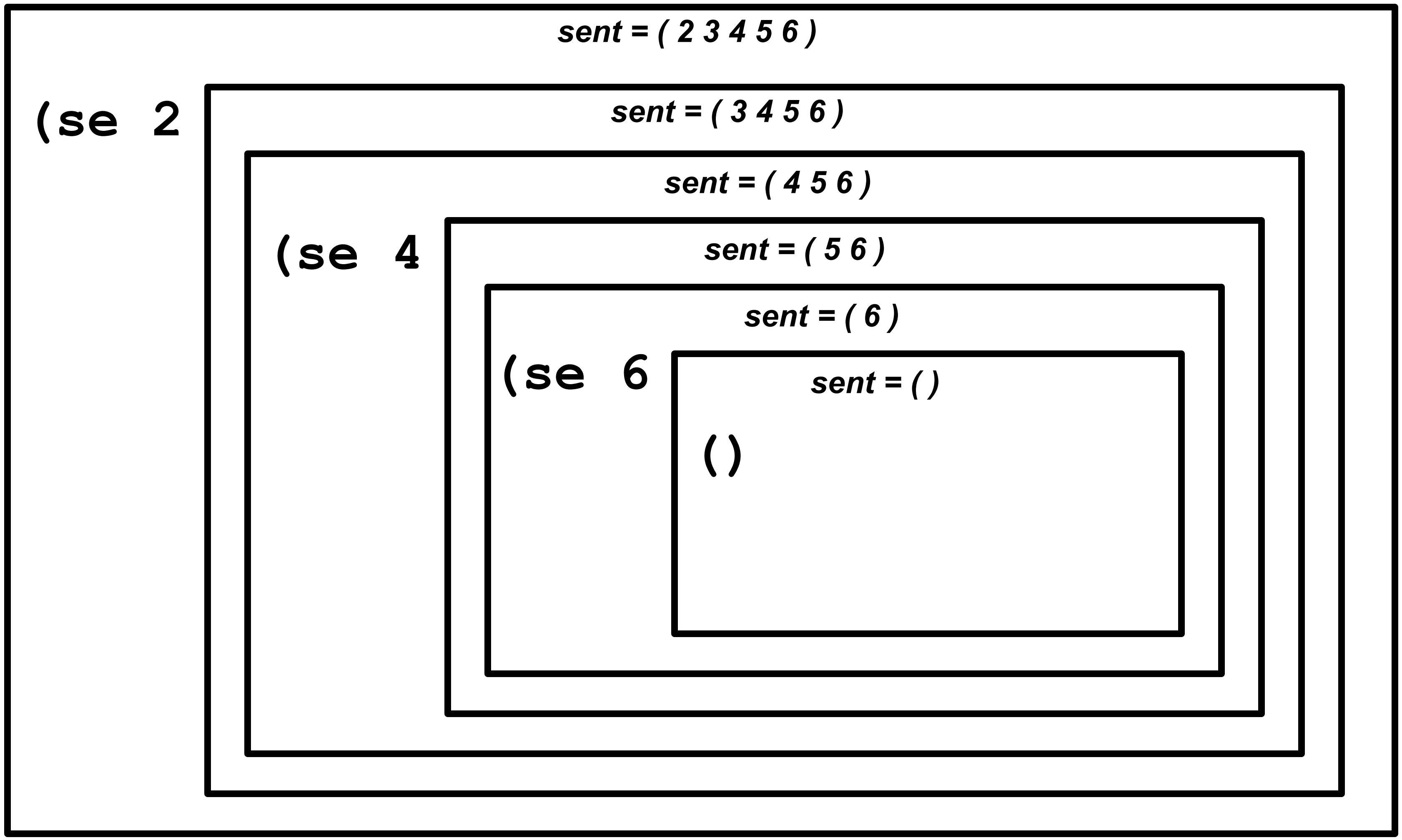
Tail or embedded? (2/3)

```
(define (find-evens sent)
  (cond ((empty? sent)          ;base case
         '()               )
        ((odd? (first sent)) ;rec case 1
         (find-evens (bf sent)) )
        (else                   ;rec case 2: even
         (se (first sent)
             (find-evens (bf sent))) )
       ))
```

```
> (find-evens '(2 3 4 5 6 7))

  (se 2 (se 4 (se 6 '())))
  (2 4 6)
```

```
> (find-evens '(2 3 4 5 6))
```



```
➔  (se 2 (se 4 (se 6 ()))
➔  (2 4 6)
```

Tail or embedded? (3/3)

```
(define (sent-max sent)
  (if (empty? sent)
    '()
    (sent-max-helper (bf sent) (first sent))))


(define (sent-max-helper sent max-so-far)
  (if (empty? sent)
    max-so-far
    (sent-max-helper (bf sent)
                     (if (> max-so-far (first sent))
                       max-so-far
                       (first sent)))))
```

Any other questions