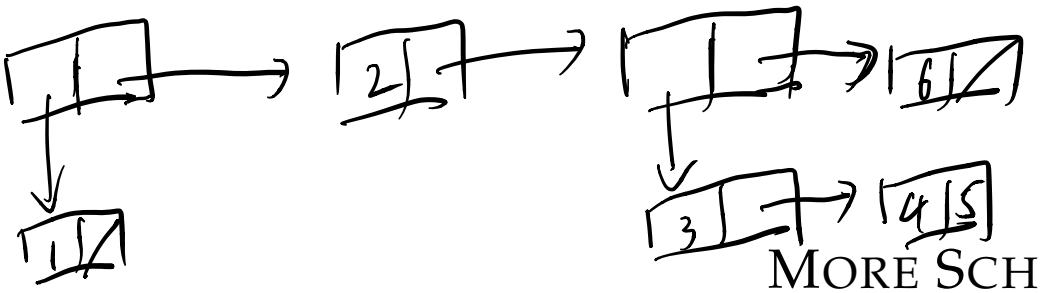
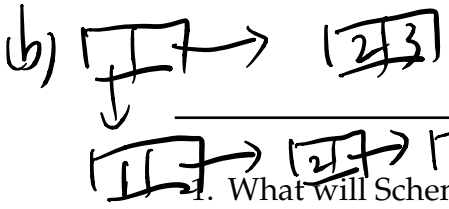
(a) 

MORE SCHEME

COMPUTER SCIENCE MENTORS 61A

April 2 to April 4, 2018

(b) 

1 Scheme

1. What will Scheme output? Draw box-and-pointer diagrams to help determine this.

(a) `(cons (cons 1 nil) (cons 2 (cons (cons 3 (cons 4 5)) (cons 6 nil))))` ((1) 2 (3 4.5) 6)

(b) `(cons (cons (car '(1 2 3)) (list 2 3 4)) (cons 2 3))` ((1 2 3 4) 2.3)

(c) `(define a 4)`
`((lambda (x y) (+ a)) 1 2)` (e)

(d) `((lambda (x y z) (y x)) 2 / 2)` 0.5

(e) `((lambda (x) (x x)) (lambda (y) 4))` 4

(f) `(define boom1 (/ 1 0))` Error: cannot divide by zero

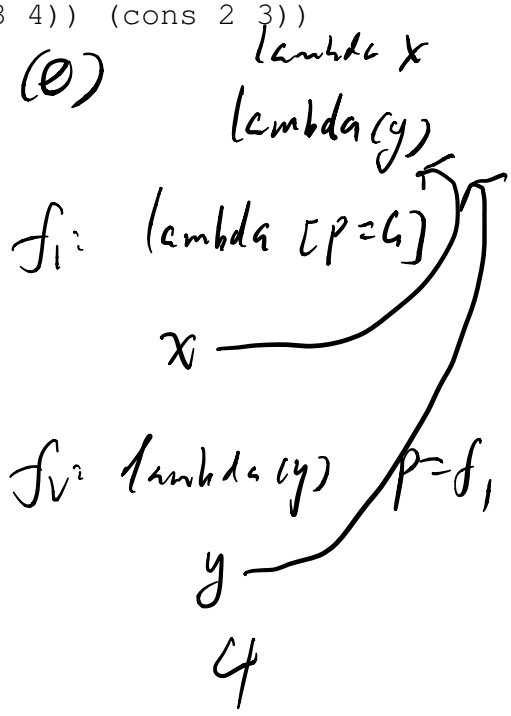
(g) `boom1` Error: unknown identifier boom1

(h) `(define boom2 (lambda () (/ 1 0)))` boom2.

(i) `(boom2)` Error: cannot divide by zero

(j) How can we rewrite boom2 without using the lambda operator?

(define (boom2) (/ 1 0))



2. What will Scheme output?.
   (a) `(if 0 (/ 1 0) 1)`
   Error: division by zero
   (b) `(and 1 #f (/ 1 0))`
   #f
   (c) `(and 1 2 3)`
   3
   (d) `(or #f #f 0 #f (/ 1 0))`
   0
   (e) `(or #f #f (/ 1 0) 3 4)`
   Error: division by zero
   (f) `(and (and) (or))`
   #f

3. **let** is a special form in Scheme which allows you to create local bindings. Consider the example

```
(let ((x 1)) (+ x 1))
```

Here, we assign `x` to `1`, and then evaluate the expression `(x + 1)` using that binding, returning `2`. However, outside of this expression, `x` would not be bound to anything.

Each `let` special form has a corresponding lambda equivalent. The equivalent lambda expression for the above example is

```
((lambda (x) (+ x 1)) 1)
```

The following line of code does not work. Why? Write the lambda equivalent of the ~~`let` expressions~~.

```
(let ((foo 3)
      (bar (+ foo 2)))
     (+ foo bar))
```

unknown identifier: foo.

还在 create frame 中，无法在此期间 eval local variable (人家还不存在！)

((lambda (foo bar) (+ foo bar)) 3 foo+2)

Lexical scoping: the parent of the call expression is the environment where the procedure was defined.

2 Scoping

1. What is the difference between dynamic and lexical scoping?

Dynamic scoping: the parent of the call expression is the environment where the call expression is evaluated;

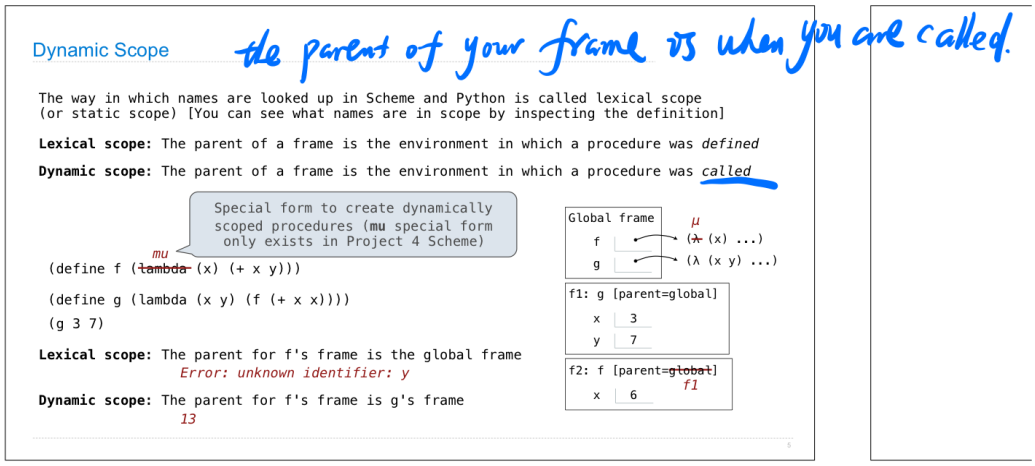
2. What would this print using lexical scoping? What would it print using dynamic scoping?

```
a = 2
def foo():
    a = 10
    return lambda x: x + a
bar = foo()
bar(10)
```

Lexical scoping: 20 (10+10)

dynamic scoping: 12 (10+2)

3. How would you modify an environment diagram to represent dynamic scoping?



3 Code-Writing

1. Implement `waldo`. `waldo` returns `#t` if the symbol waldo is in a list. You may assume that the list passed in is well-formed.

```
scm> (waldo '(1 4 waldo))
#t
scm> (waldo '())
#f
scm> (waldo '(1 4 9))
#f
```

```
(define (waldo lst)
  (if (null? lst)
      #f
      (if (eq? (car lst) 'waldo)
          #t
          (waldo (cdr lst)))))
)
```

2. **Extra challenge:** Define `waldo` so that it returns the index of the list where the symbol waldo was found (if waldo is not in the list, return `#f`).

```
scm> (waldo '(1 4 waldo))
2
scm> (waldo '())
#f
scm> (waldo '(1 4 9))
#f
```

```
(define (waldo lst)
    (define (find-waldo lst ind)
        (if (null? lst)
            #f
            (if (eq? (car lst) 'waldo)
                ind
                (find-waldo (cdr lst) (+ ind 1)))))
    (find-waldo lst 0)
)
```

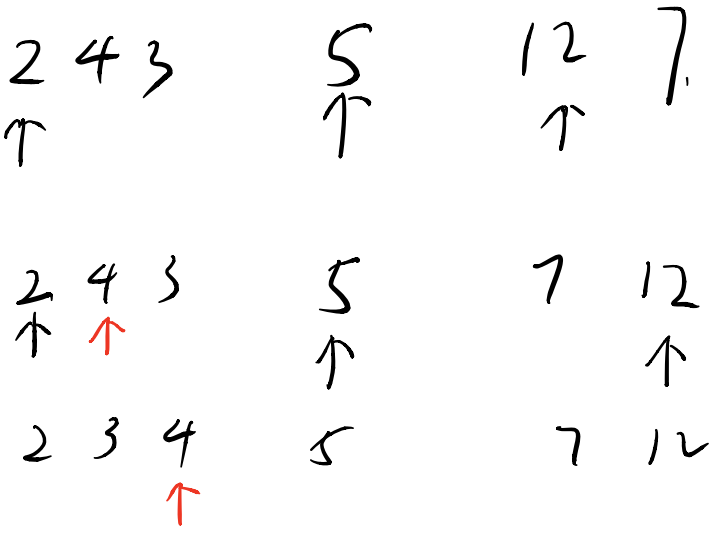
4 Challenge Question

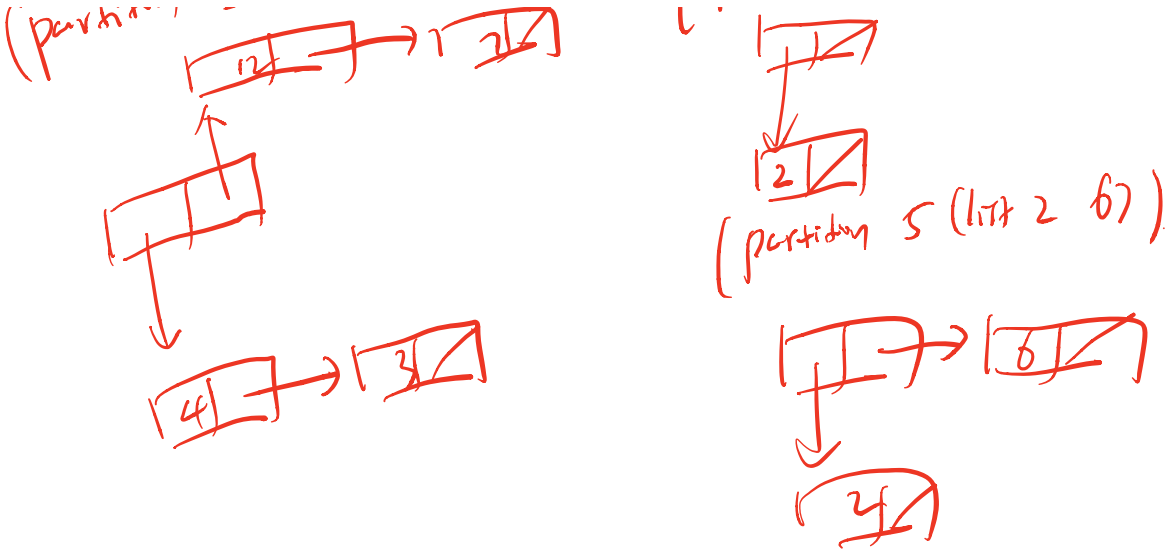
3. **(Optional)** The quicksort sorting algorithm is an efficient and commonly used algorithm to order the elements of a list. We choose one element of the list to be the pivot element and partition the remaining elements into two lists: one of elements less than the pivot and one of elements greater than the pivot. We recursively sort the two lists, which gives us a sorted list of all the elements less than the pivot and all the elements greater than the pivot, which we can then combine with the pivot for a completely sorted list.

   Implement `quicksort` in Scheme. Choose the first element of the list as the pivot. You may assume that all elements are distinct. Hint: you may want to use a helper function.

   You may additionally want to use the built-in `append` function, which takes in two lists and returns a new list containing the elements of the first list followed by the elements of the second list. You can also use `filter` procedure, which takes in a one-argument function and a list and returns a new list containing only the elements of the original list for which the function returns true, although it is not required.

```
scm> (quicksort (list 5 2 4 3 12 7))
(2 3 4 5 7 12)
```




12
3
4
3
2
2
(partition 5 (list 2 6))
6
4