

# MORE SCHEME

---

COMPUTER SCIENCE MENTORS CS 61A

April 8 to April 10, 2019

---

## **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 nil)) (cons 6 nil))))`

**Solution:**

`((1) 2 (3 4 5) 6)`

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

**Solution:** `((1 2 3 4) 2)`

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

**Solution:**

4

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

**Solution:**

0.5

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

**Solution:** 4

- (f) `(define boom1 (/ 1 0))`

**Solution:** Error: Zero Division

- (g) `boom1`

**Solution:** Error: boom1 not defined

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

**Solution:** boom2

- (i) `(boom2)`

**Solution:** Error: Zero Division

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

**Solution:**

`(define boom2 (/ 1 0))`  
 CS61A Spring 2019: Catherine Cang and Alex Stennet, with Albert Xu, Allen Yang, Allen Yu, Amy Huang, Chris Niu, Danelle Nachum, Evan Corriere, Jack Coyle, Jade Singh, Jason Chang, Jemmy Zhou, Kevin Wang, Lucas Pan, Mary Xu, Michelle Cheung, Olivia Shiah, Patricia Ouyang, Richard Roggenkemper, Samy Cherfaoui, Shreya Sahoo, Varun Jhunjhunwalla

## 2. What will Scheme output?.

(a) `(if 0 (/ 1 0) 1)`**Solution:**

Error: Zero Division

(b) `(and 1 #f (/ 1 0))`**Solution:**

#f

(c) `(and 1 2 3)`**Solution:**

3

(d) `(or #f #f 0 #f (/ 1 0))`**Solution:**

0

(e) `(or #f #f (/ 1 0) 3 4)`**Solution:**

Error: Zero Division

(f) `(and (and) (or))`**Solution:**

#f

## 3. What will Scheme output?

(a) **(define** c 2)

**Solution:**

c

(b) **(eval** 'c)

**Solution:**

2

(c) **'**(cons 1 nil)

**Solution:**

(cons 1 nil)

(d) **(eval** '(cons 1 nil))

**Solution:**

(1)

(e) **(eval** (list 'if '(even? c) 1 2))

**Solution:**

2

## 2 Scoping

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

**Solution:**

- **Lexical:** The parent of a frame is the frame in which a procedure was defined (used in Python).
- **Dynamic:** The parent of a frame is the frame in which a procedure is called (keep an eye out for this in the Scheme project).

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)
```

**Solution:**

- **Lexical:** 20
- **Dynamic:** 12

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

**Solution:** Assign parents when you create a frame (do not set parents when defining functions!). The parent in this case is the frame in which you called this function.

---

### 3 Code-Writing

---

1. Define `is-prefix`, which takes in a list `p` and a list `lst` and determines if `p` is a prefix of `lst`. That is, it determines if `lst` starts with all the elements in `p`.

```
; Doctests:
scm> (is-prefix '() '())
#t
scm> (is-prefix '() '(1 2))
#t
scm> (is-prefix '(1) '(1 2))
#t
scm> (is-prefix '(2) '(1 2))
#f
; Note here p is longer than lst
scm> (is-prefix '(1 2) '(1))
#f

(define (is-prefix p lst)
```

```
)
```

**Solution:**

```
; is-prefix with nested if statements
(define (is-prefix p lst)
  (if (null? p)
      #t
      (if (null? lst)
          #f
          (and
            (= (car p) (car lst))
            (is-prefix (cdr p) (cdr lst)))))))

; is-prefix with a cond statement
(define (is-prefix p lst)
  (cond ((null? p) #t)
        ((null? lst) #f)
        (else (and (= (car p) (car lst))
                     (is-prefix (cdr p) (cdr lst))))))
```

2. Define `apply-multiple` which takes in a single argument function `f`, a nonnegative integer `n`, and a value `x` and returns the result of applying `f` to `x` a total of `n` times.

```
;doctests
```

```
scm> (apply-multiple (lambda (x) (* x x)) 3 2)
256
scm> (apply-multiple (lambda (x) (+ x 1)) 10 1)
11
scm> (apply-multiple (lambda (x) (* 1000 x)) 0 5)
5
```

```
(define apply-multiple (f n x)
```

)

**Solution:**

```
(define (apply-multiple f n x)
  (if (= n 0)
    x
    (f (apply-multiple f (- n 1) x)))))
```

**Alternate solution:**

```
(define (apply-multiple f n x)
  (if (= n 0)
    x
    (apply-multiple f (- n 1) (f x)))))
```