ASSIGNMENT, STATE AND ENVIRONMENTS 8

COMPUTER SCIENCE 61AS

Local State Variables

1. What are local state variables and why do we use them?

2. What will Scheme output?

```
(define (make-counter number)
  (lambda ()
     (begin (set! number (+ number 1))
            number)))

(define c1 (make-counter 1))
(define c2 (make-counter 1))

(c1)
_________

(c2)
_________
```

3. At this point, can we still use the substitution model of evaluation? Why or why not?

4. Fill in the blanks with the values of the expressions shown:

```
(define x 1)
(define foo
```

```
(let ((x x))
  (lambda ()
     (set! x (+ x 1))
     x)))

(foo)                    _____
(foo)                    _____
(foo)                    _____
x                        _____
```

The Environment Model of Evaluation

1. What is a frame? When are frames created?

2. What is the initial frame, if no frame has been created?

3. How is a variable evaluated?

4. What are the two things that create a new frame?

Environment Diagrams

Draw environment diagram for each, and say what the code evaluates to.

1.
```
(define (foo x)
(bar x 4))
(define (bar y z)
(- y z))
(foo 9)
```
_____

2.
```
(define x 10)
(define y 20)
(let  ((x y)
  (y x))
(set! x y)
(set! y x)
(- x y))
```
_____

y

_____

3.
```
(define (compose f g)
(lambda (x) (f (g x))))
(define mystery
       (compose
(lambda (x) (+ x 5))
(lambda (x) (* x 10))))
(mystery 123)
```
_____

OOP Below the Line

Lets revisit Tic-Tac-Toe! Rewrite the board class without using the OOP language. The code in OOP is given below. After you rewrite it, there will be a different way of instantiating a board, and a different way of calling a method on a board, analogous to the transformation of the counter class shown in the lecture notes/webcast.

Then, give a sample interaction which creates a board, and invokes play-move with piece x, and coordinates (1, 0). Show the environment diagram generated by this interaction.

```
(define-class (board)
  (instance-vars (grid (make-grid)))
  (method (piece x y) (get-piece grid x y))
```

```
  (method (play-move piece x y)
    (set! grid (next-grid grid piece x y)))
```