Week 9: Environment Diagrams

Practice Makes Perfect...Now grab a bunch of paper and...
Draw the environment diagram and also show the return value

Problem 001:

```
(define (make-adder n)
  (lambda (x) (+ x n)))

(make-adder 3)

((make-adder 3) 5)

(define (f x)
  (make-adder 3))

(f 5)

(define g (make-adder 3))

(g 5)
```

Problem 002:

```
(define (make-funny-adder n)
  (lambda (x)
    (if (equal? x 'new)
     (set! n (+ n 1))
     (+ x n))))

(define h (make-funny-adder 3))

(define j (make-funny-adder 7))

(h 5)
(h 5)
(h 'new)
(h 5)
(g 5)
```

Problem 003:

```
a) (let ((a 3))
     (+ 5 a))

b) (let ((a 3))
     (lambda (x) (+ x a)))

c) ((let ((a 3))
      (lambda (x) (+ x a)))
    5)
```

Problem 004:

```
(begin (define x 6) (define a 5))

(define (square a) (* a a))

(define (scale-square-nums a b)
  (* x (+ (square a) (square b))))

(square 5)

(scale-square-nums 3 4)
```

Problem 005:

```
(define (make-rectangular x y)
  (define (dispatch op)
    (cond ((eq? op 'real) x)
       ((eq? op 'imag) y)
       ((eq? op 'mag)
        (sqrt (+ (square x) (square y))))
       ((eq? op 'angle)
        (atan y x))))
  dispatch)

(define c1 (make-rectangular 3 4))

(c1 'mag)
```

Problem 006:

```
(define (make-prev last-value)
  (lambda (new)
    (define temp last-value)
    (set! last-value new)
    temp))

(define prev (make-prev '*first-call*))

(prev 'a)
```

Problem 007:

```
(define y 5)

(define (agent x)
  (let ((y 0))
    (lambda () (x) y)))

(define mission (agent (lambda () (set! y (+ y 1)))))

(mission)

(mission)

y
```

Problem 008:

```
(define answer 0)

(define (square f x)
  (let ((answer 0))
    (f x)
    answer))

(square (lambda (n) (set! answer (* n n))) 3)
```

Problem 009:

```
(define a 'carolen)

((lambda (a b) (b) a)
 a
 (let ((b 'steven))
   (lambda () (set! a b))))

a
```

Problem 010:

```
(define a 3)

((lambda (a)
   ((lambda (a) (a))
    (lambda () (set! a 'aye)))
   a)
 (* a a))
```

Problem 011:

```
(define snuffles (cons 1 2))

(let ()
  (define snuffles (cons 3 4))
  ((lambda () (set-cdr! snuffles (car snuffles))))
  snuffles)
```

Problem 012:

```
(define x 'x)

(define (changer x y)
  (y)
  (y)
  x)

(changer 16 (lambda () (set! x (* x x))))
```

Problem 013:

```
(define vlad
  (let ((a (lambda (vlad) (set! cdr vlad))))
    (lambda (cdr)
      (a car)
      (cdr '(1 2 3)))))

(vlad cdr)

(cdr '(1 2 3))
```

Problem 014:

```
(define c (lambda (x) (* x x)))

(define (yikes a b)
  (let ((c 5)
        (d (lambda () (set! c 'c))))
    (d)
    (define a-on-c (a c))
    c))

(yikes c c)
```

Problem 015:

```
(define elephant 2)

(let ((z (lambda ()
             (lambda (x) (set! elephant x))))
      (elephant 4))
  ((lambda (elephant)
     ((z) 'cheese))
    7))
```

Problem 016:

```
(define (bar x)
  (let ((z (lambda (b) (* x b)))
        (c x))
    (lambda (x)
      (set! c (z (* c x)))
       c)))

(define foo (bar 4))

(foo 3)
```

Problem 017:

```
(define cs61a
  (let ((a (lambda () (set! x (+ 1 x)) cs61a)))
    (lambda (x)
      (define (change-x) (set! x (* x x)) a)
       change-x)))

(define x 2)

(define one (cs61a 27))

(define two (one))
```

017 interaction!...

```
x

(one)

x

(two)

x
```

Problem 018:

```
(let ((proc (lambda (n f)
           (if (= n 1)
            1
            (* n (f (- n 1) f))))))
  (proc 4 proc))
```

Problem 019:

```
(define (map! func lst)
  (if (null? lst)
      'booyah!
      (begin (set-car! lst (func (car lst)))
          (map! func (cdr lst)))))

(define kaisen (list (cons 1 2) (cons 3 4)))

(map! car kaisen)
```