MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1997

Lecture Notes – October 16, 1997

Environment Model

Today's lecture introduces a new model of evaluation, called the environment model. This model involves changing our perception of objects in our language.

- we change our view of procedures from functions to objects.
- we change our view of variables from names to places.

An environment is a sequence of **frames** and each frame is a table of **bindings**, each of which is **a pairing of variable and value**.

Rules for the Environment Model

- To evaluate a combination (other than a special form) first evaluate the subexpressions of the combination, and then apply the value of the operator subexpression to the values of the operand subexpressions.
- The value of a variable with respect to an environment is the value given by the binding of the variable in the first frame in the environment that contains such a binding.
- Evaluating a lambda expression produces a procedure object, which consists of two parts. The first is the code of the procedure, which is given by the text of the lambda expression. The second is the environment pointer, which points to the environment in which the lambda expression was evaluated to produce the procedure.
- `Define` adds a binding to a frame.
- A procedure object is applied to a set of arguments by constructing a frame, binding the formal parameters of the procedure to the actual arguments of the call, and then evaluating the body of the procedure in the context of the new environment contructed. The new frame has as its enclosing environment the environment part of the procedure object being applied.

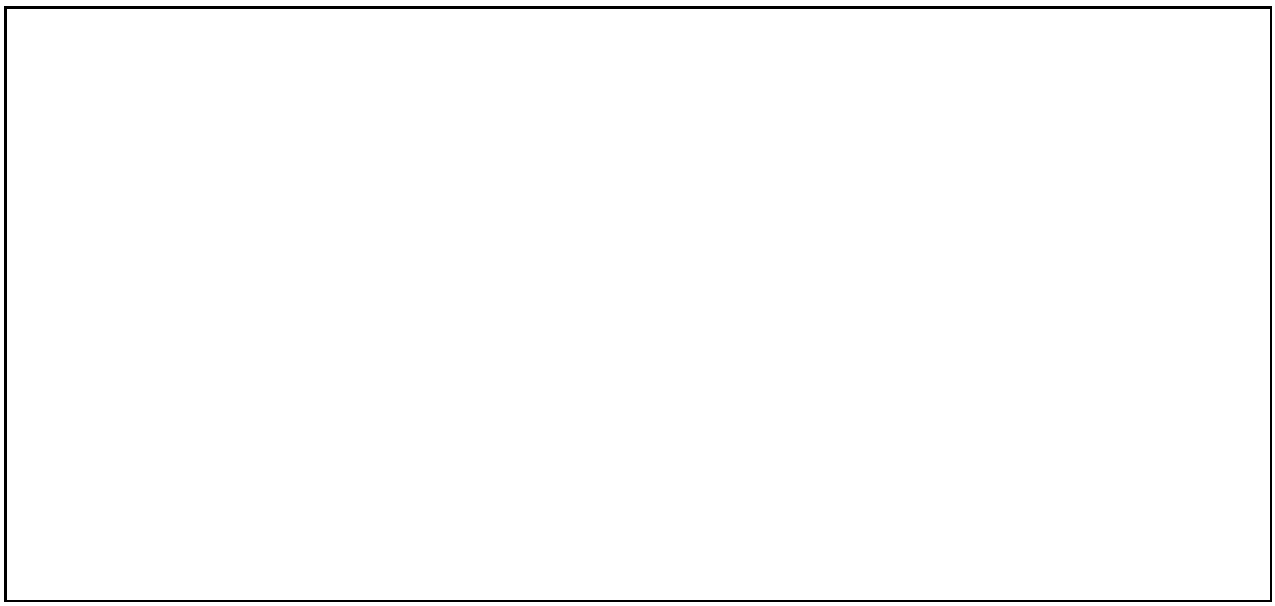
Sketch the environment diagram that evolves from the following code examples.

```
(define x 63)

(define square (lambda (x)
                 (* x x)))

(define sum-sq (lambda (x y)
                 (+ (square x) (square y))))

(sum-sq 3 4) ==> 25
```

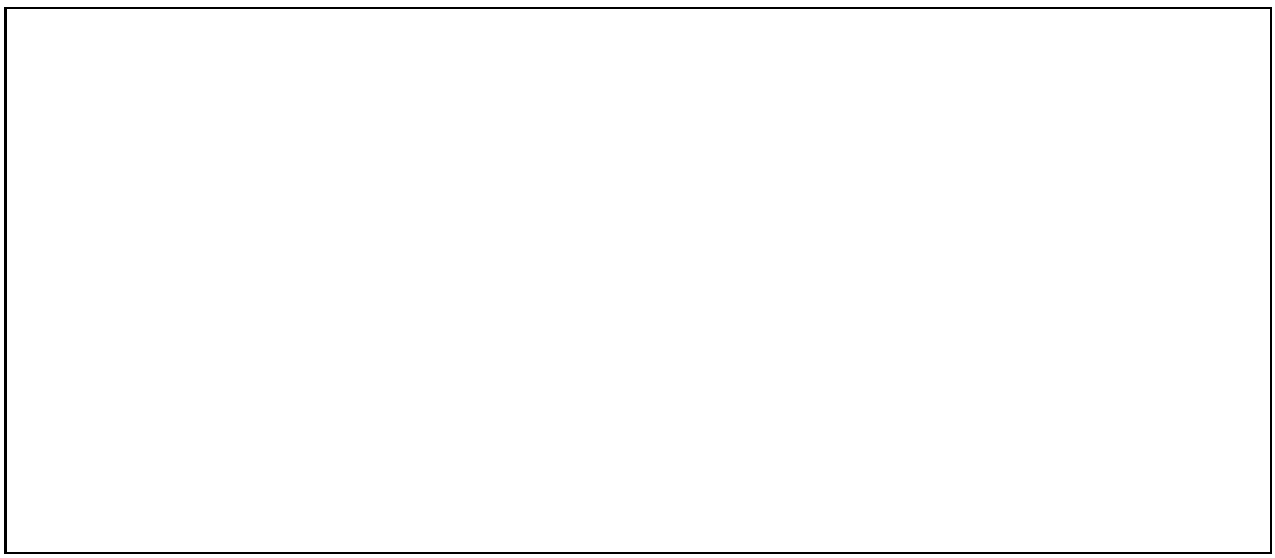


Environment diagram 1.

Here is a second example:

```
(define (sqrt x)
  (define (sqrt-iter guess)
    (if (good-enuf? guess)
        guess
        (sqrt-iter (improve guess))))
   (define (good-enuf? guess)
     (< (abs (- (square guess) x)) .001))
   (define (improve guess)
      (average guess (/ x guess)))
   (sqrt-iter 1))

(sqrt 2)
```



Environment diagram 2.

Third example:

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

(define addthree (make-adder 3))

(define addfive (make-adder 5))

(addfive 7)

(addthree 7)
```

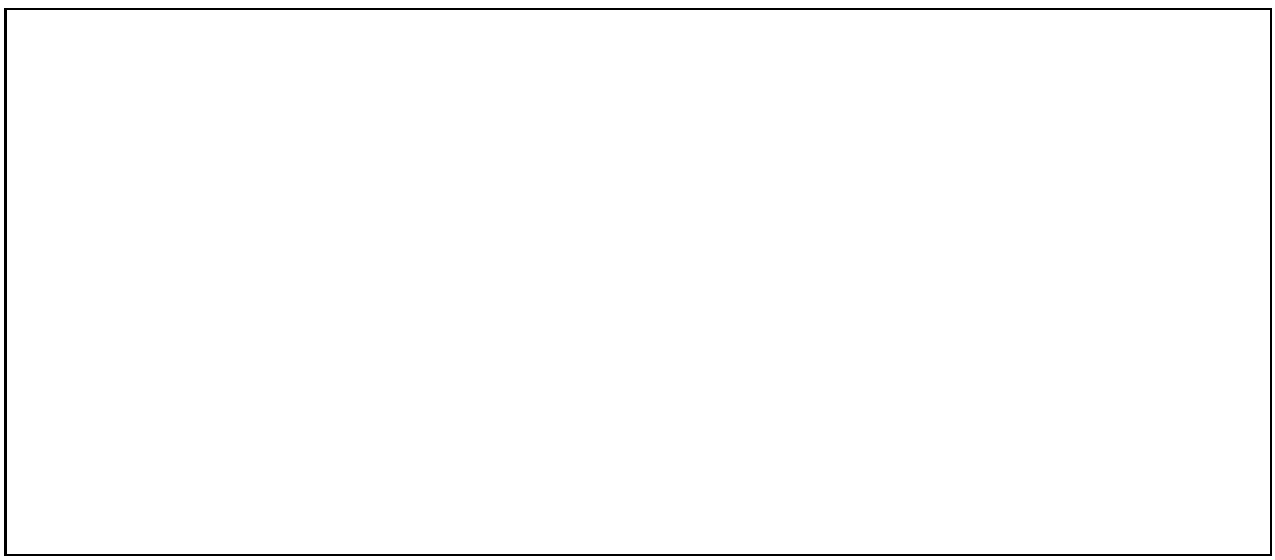


Figure 3. Environment diagram 3.

Environments enable us to understand how we can use procedures as representations for data abstractions. For example, consider the following method for building complex numbers:

```
(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 'imag)
```

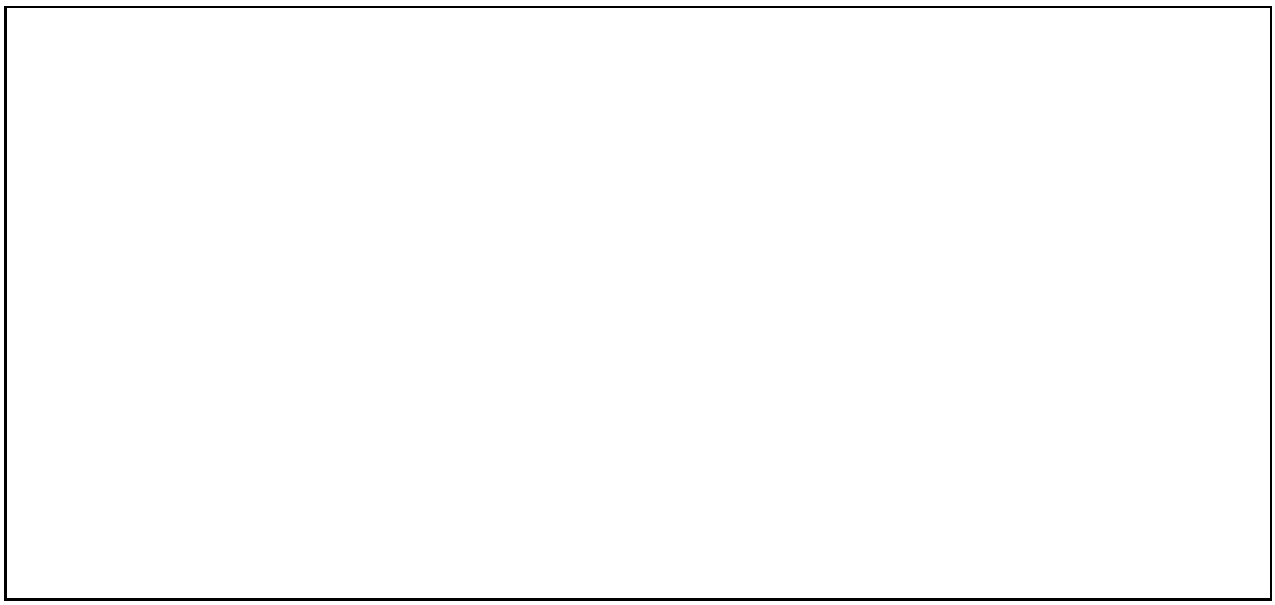


Figure 4. Environment diagram 4.