MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1998

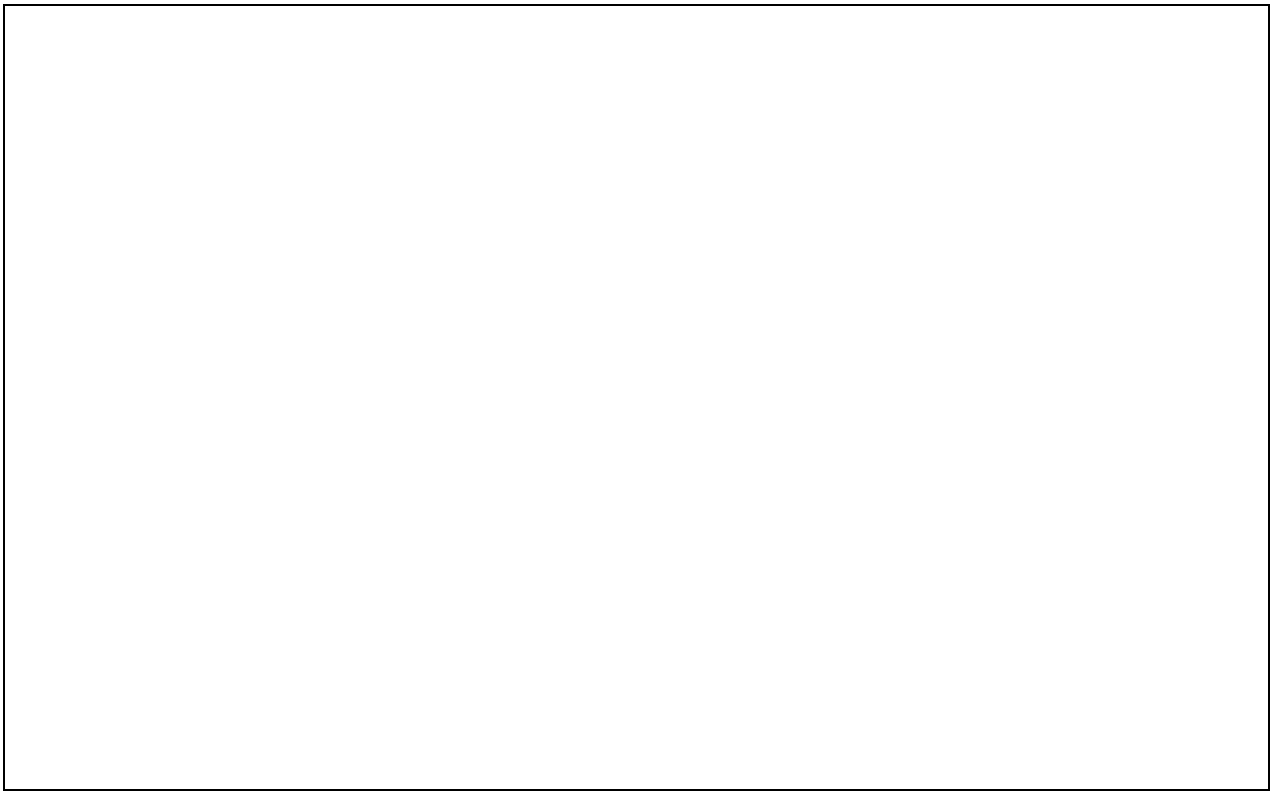
Lecture Notes – October 22, 1998

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 that not only compute and return values, but which may also affect their environment.
- We change our view of variables from names for values to places to hold values.

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**. An example:



Rules for Environment Diagrams

1. **Variables:**
   Look up the variable name in the specified environment frame and find the value it is bound to. If the variable is not in the current frame, look in enclosing frames by following the frame arrows. If you reach the Global Environment and the variable is still not bound, then the variable is unbound.

2. **Define Special Form:** `(define <var> <exp>)`
   Evaluate the expression, then bind the variable to this value in the current frame.

3. **Lambda Special Form:**
   Create a procedure object (two bubbles). The left bubble records the formal parameter list and the body of the procedure. The right bubble points to the environment in which the procedure was created (where the lambda was evaluated).

4. **Combinations:**
   Evaluate subexpressions (in any order), then apply value of operator subexpression to values of operands subexpressions.

5. **Compound Procedure Application:**

   (a) Draw a new frame.

   (b) Draw a dotted line from the procedure object to this new frame.

   (c) Figure out where the frame points to (its parent frame): find the frame that the right bubble of the procedure you are applying points to, and make the new frame point to the same place.

   (d) Bind the formal parameters of the procedure to their respective actual argument values in this new frame.

   (e) Evaluate the body of the procedure with respect to this new environment.

6. **Set! Special Form:** `(set! <var> <exp>)`
   Evaluate `<exp>` with respect to the current environment. Find and change the nearest binding for the variable in the current environment, and change its value.

7. **Let Special Form:**
   One could desugar the let into a lambda application. A short cut is to "hang" a new frame (that is, create a new frame that points to the current frame) with the variables of the let statement bound to their appropriate values (be careful when finding these values – they are evaluated in the current frame, not the new frame). Then evaluate the body of the let in the new environment.

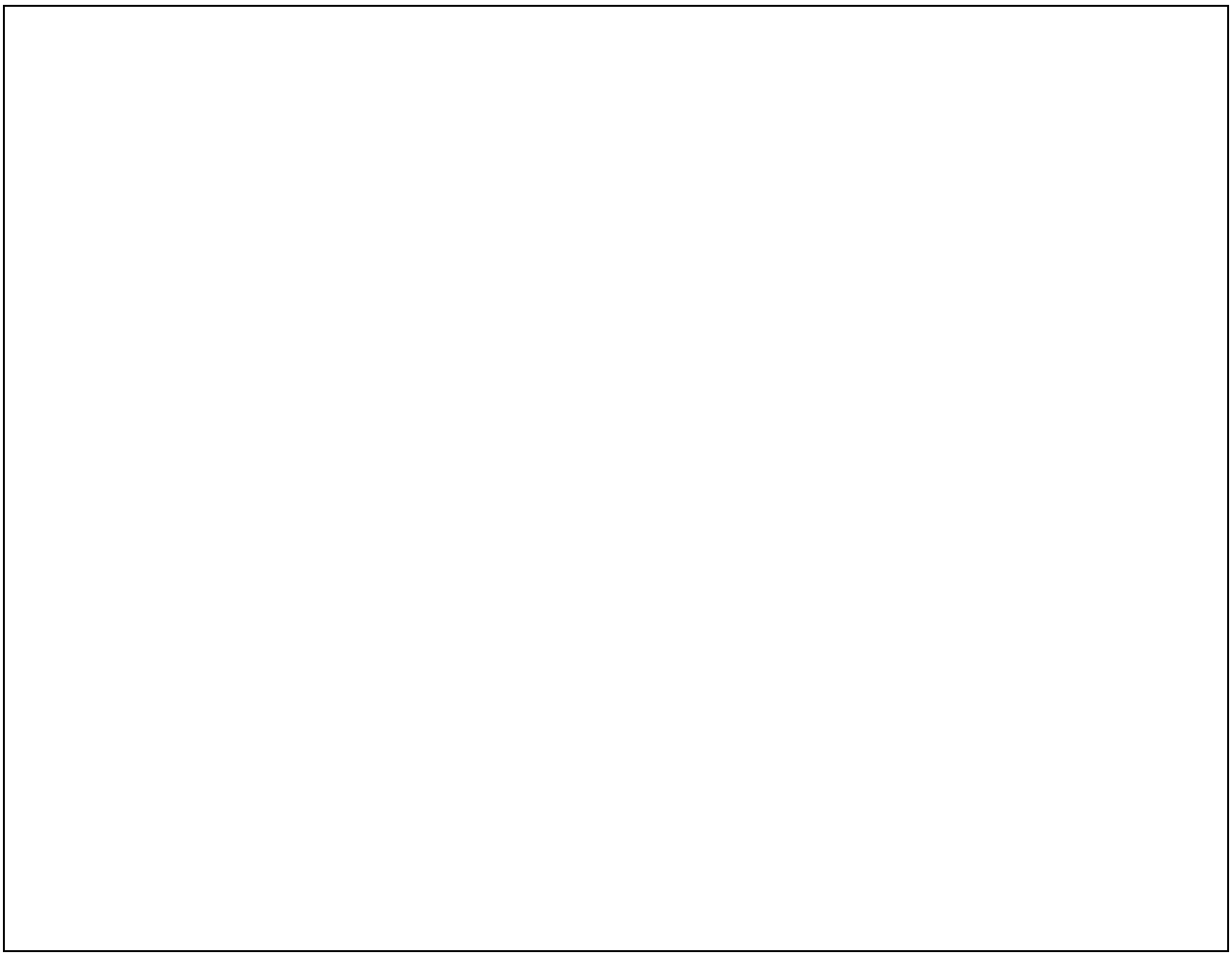
Sketch the environment diagram that evolves from the evaluation of the following code examples (each evaluated with respect to the global environment):

```
(define x 63)

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

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

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

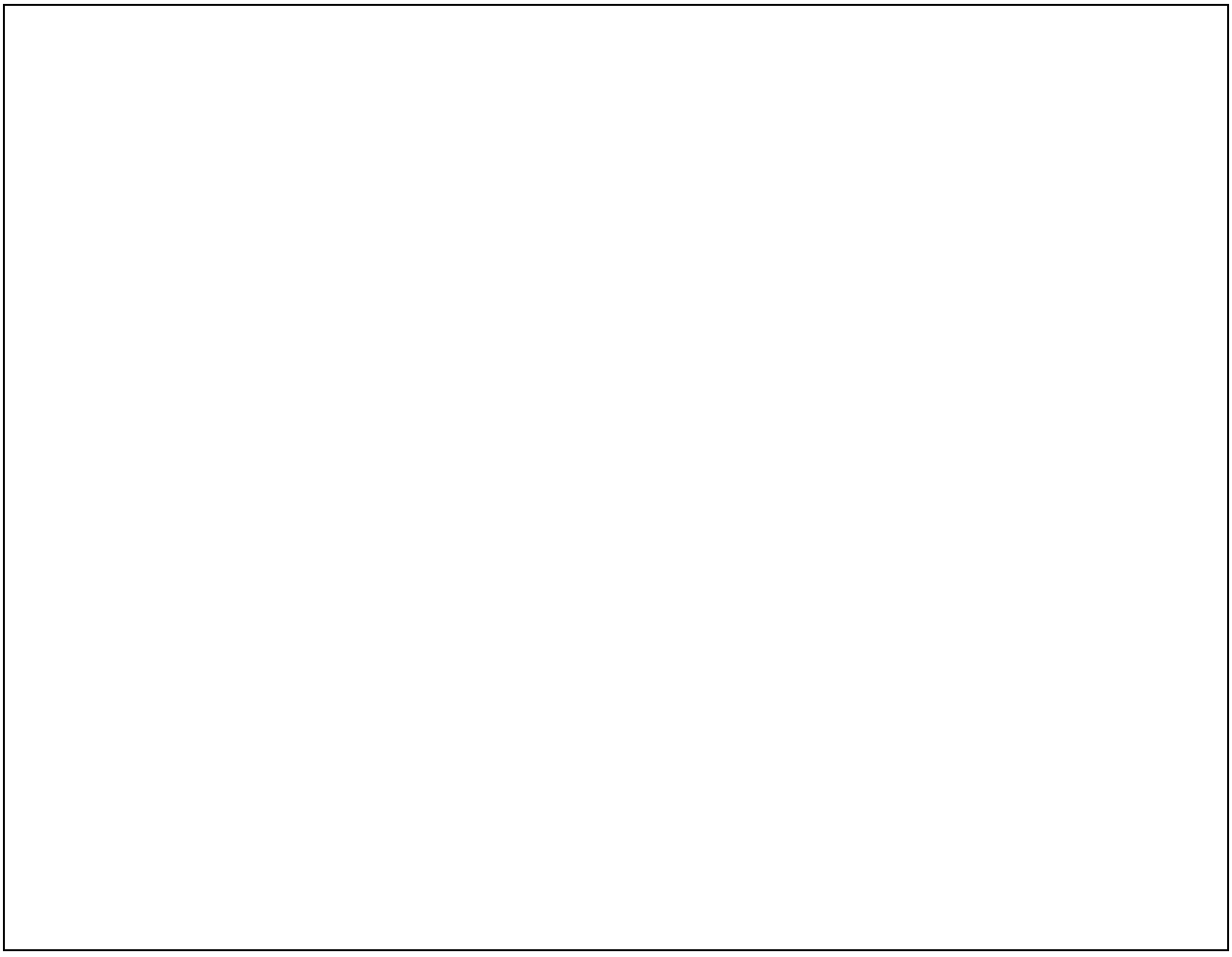


Environment diagram 1.

Here is a second example emphazing how *lexical scoping* is accomplished by nested frames.

```
(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.

A third example, where a *memory* capability is shown.

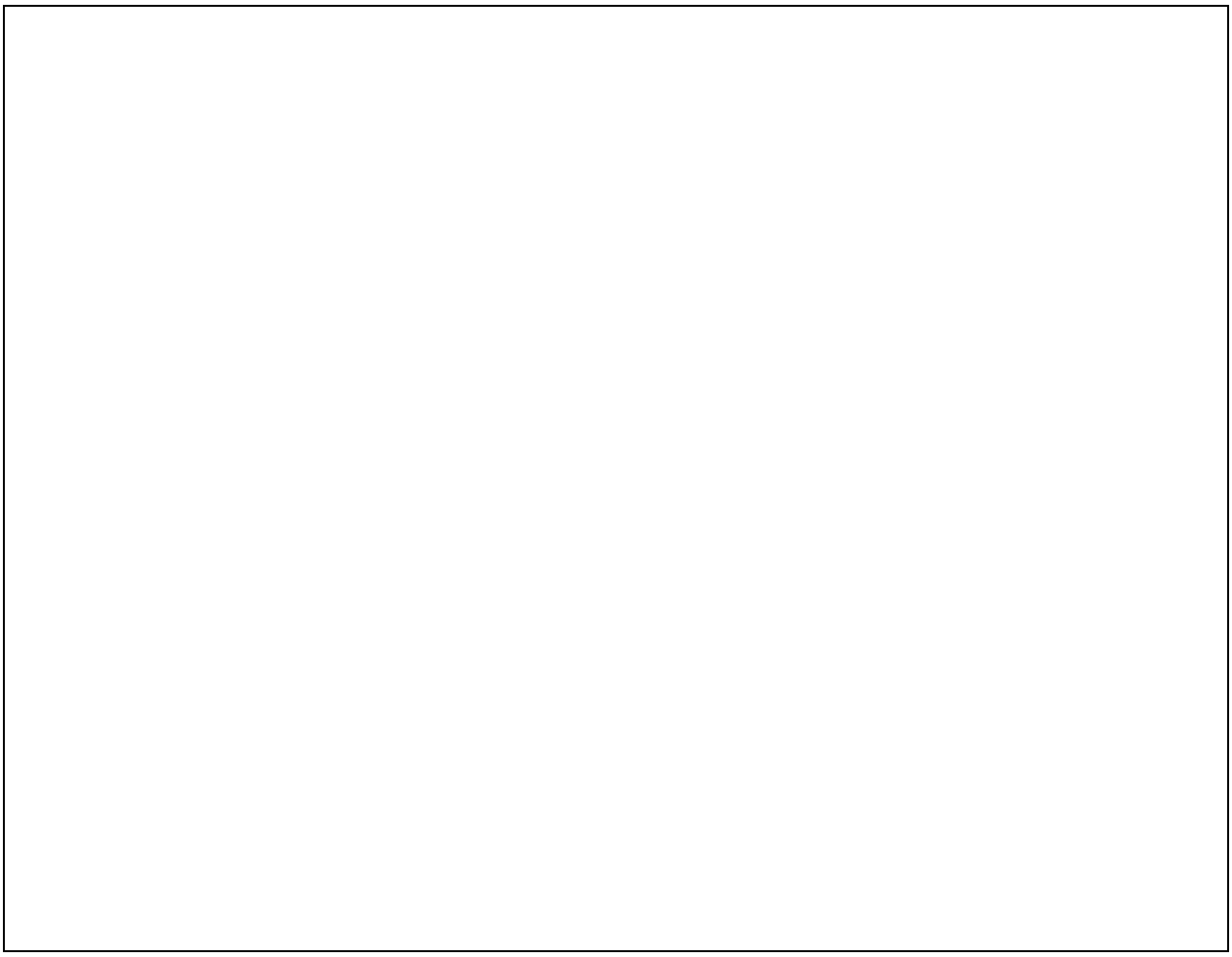
```
(define (make-adder n)
  (lambda (x) (+ x n)))

(define addthree (make-adder 3))

(define addfive (make-adder 5))

(addfive 7) ==> 12

(addthree 7) ==> 10
```



Environment diagram 3.

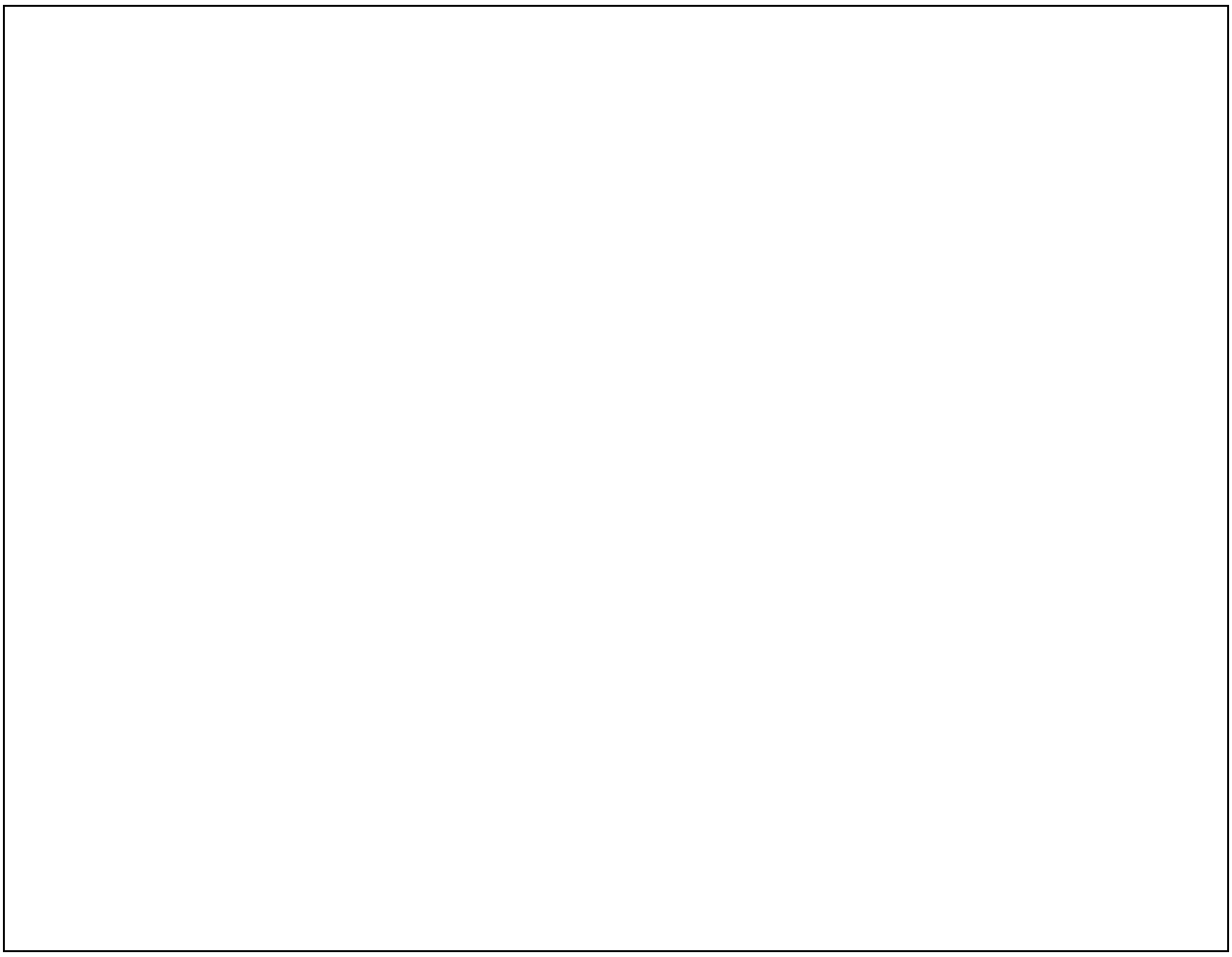
Another example, this time with *state* that changes:

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

(define counter (make-counter 0))

(counter) ==> 1

(counter) ==> 2
```



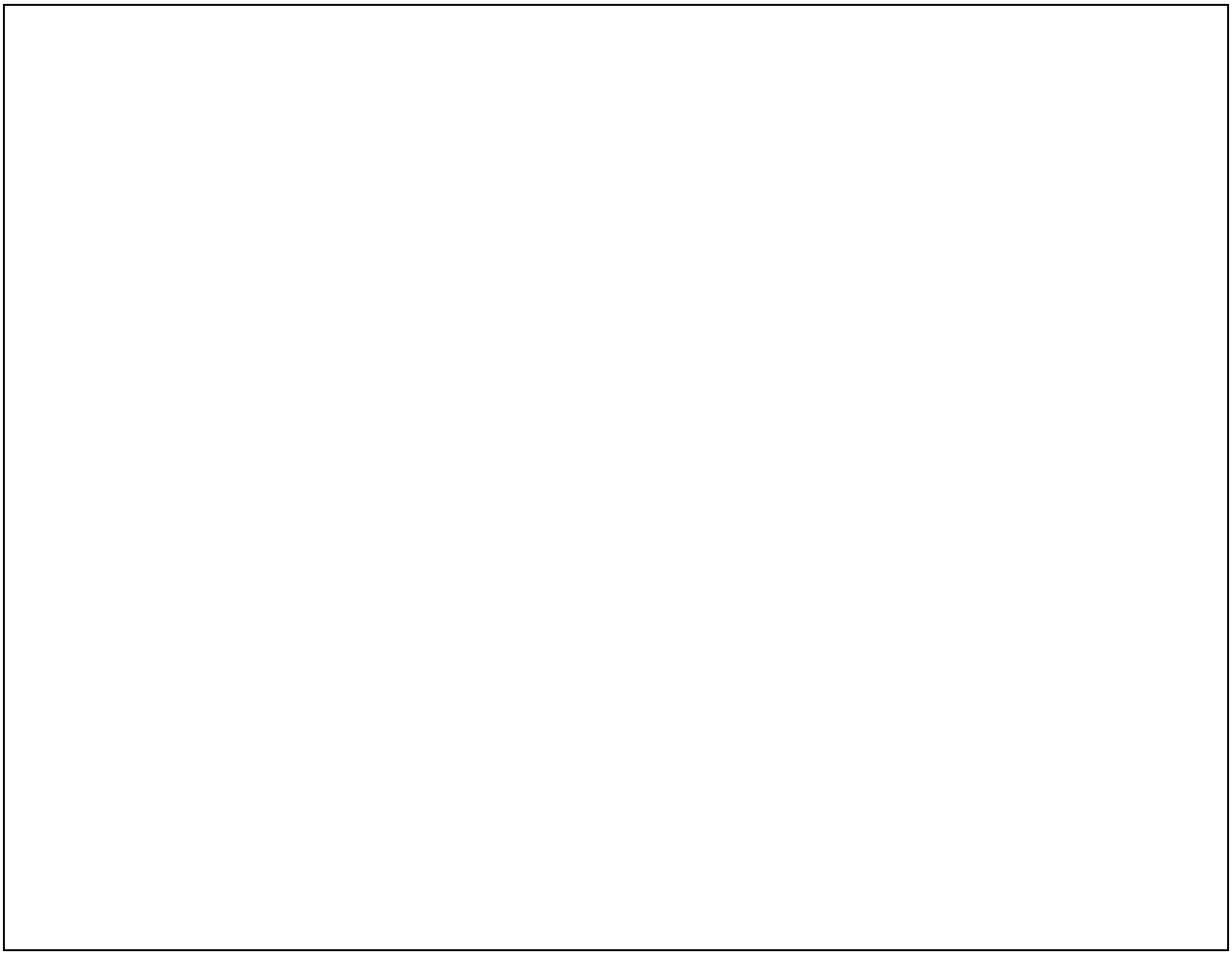
Environment diagram 4.

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)
```



Environment diagram 5.