MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 Structure and Interpretation of Computer Programs
Spring, 2007

Recitation 4b, Wed. February 21

Let Dr. Kimberle Koile

In Scheme, `let` is a special form that is used as a way to bind the values of *local* variables, i.e., variables that are accessible only within a procedure (or other defined block of code).

The general form of `let` is:

```
(let ((<var1>  <exp1>)        ; each var is bound to the value of the  corresponding expression
      (<var2>  <exp2>)
         . . .
      (<varn>  <expn>))
  <body>)
```

Example

```
(define a 3)

(define (foo x)
  (let ((a  0))
    (+  x  a)))

(foo 10)  =>
```

`Let` desugars into an anonymous lambda.

```
(define (foo x)
   ((lambda (a)  (+  x  a))   0)))
```

Variables are bound in parallel, not sequentially.

```
(define a 3)

(let ((a 0)
      (b a))
  (+  a b))   =>
```

desugaring with more than one variable:

```
((lambda (a b) (+  a  b))   0   a)
```

To bind variables sequentially, use several `lets` or `let*`.

```
(define a 3)

(let ((a 0))
  (let ((b  a))
     (+  a  b))  =>


(let* ((a 0)
       (b a))
  (+  a  b))  =>
```

Problems

For each expression or set of expressions, give the value returned by evaluating the last expression in the set.

value

1.
```
((lambda (x)
     (let ((y  4))
       (+  x  y)))
   7)
```
[ ]

2.
```
(define  a  4)
(define  x  3)

(define  (foo x)
     (let ((a  1))
       (+  x  a))))
 (foo  2)
```
[ ]

3.
```
(let ((foo (lambda (x)  (*  2  x)))
      (y  4))
  (foo y)
 )
```
[ ]