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

Recitation – Friday, February 13

0. Announcements

· Problem Set 1 is due **now**. Late work is not accepted. Solutions will be posted on the web.

· No classes Monday. Tuesday follows a Monday schedule. People in Tuesday tutorials should try to go to another tutorial.

1. Student Presentation

Here are some procedures that double or triple their argument:

```
(define double (lambda (x) (* 2 x)))
(define triple (lambda (x) (* 3 x)))
```

Write a procedure that takes a number as argument, and returns a procedure that multiplies its argument by the number, for example:

```
(define doubler (make-mul 2))
(doubler 5)
;Value: 10
```

2. Higher Order Procedures

Write a function `swap` that takes a function f of two variables, and returns a funtion of two variables that is f with the variables swapped: `(f x y) == ((swap f) y x)` For example,
`((swap -) 4 5)` ⇒ `1`.

```
(define swap



)
```

3. More Higher Order Procedures

Write a function `fmax` that takes two functions f and g (that each take a number) and returns a function, that when called with a number x, returns the max of $f(x)$ and $g(x)$. For example,

```
(define max-sqr-sqrt (fmax sqr sqrt))
(max-sqr-sqrt 3) == 9
(max-sqr-sqrt 0.25) == 0.5
```

```
(define fmax






)
```

4. Compose and Repeated

```
(define compose
  (lambda (f g)
    (lambda (x) (f (g x)))))

(define repeated
  (lambda (proc n)
    (if (= n 0)
        identity
        (compose proc (repeated proc (- n 1))))))
```

To see how this is working, use the substitution model to evaluate the folloing expression:

```
((repeated square 2) 3)
```

5. Let and Lambda

```
(let ((<var1> <exp1>)
      (<var2> <exp2>)
      ...
      (<varn> <expn>))
  <body>)
```

The let statement is equivalent to the following

```
((lambda (<var1> <var2> ... <varn>)
     <body>)
 <exp1>
 <exp2>
 ...
 <expn>)
```

Compare the following two examples. First use let:

```
(let ((a 2)
      (b 5))
  (+ a b))
```

And now use lambda.

```
((lambda (a b) (+ a b)) 2 5)
```

Question: Why won't this work? (Hint: Unravel the syntactic sugar!)

```
(let ((x 5)
      (y (* x x)))
  (+ x y))
```