CS 61A Week 2 Lab
Monday afternoon, Tuesday, or Wednesday morning

This lab introduces a new special form, `lambda`.

1. Type each of the following into Scheme, and note the results. See when you can *predict* the results before letting Scheme do the computation.

```
(lambda (x) (+ x 3))

((lambda (x) (+ x 3) 7)
```

You can think of `lambda` as meaning "the function of...," e.g., "the function of `x` that returns `(+ x 3)`.

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

((make-adder 3) 7)

(define plus3 (make-adder 3))

(plus3 7)

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

(square 5)

(define sq (lambda (x) (* x x)))

(sq 5)

(define (try f) (f 3 5))

(try +)

(try word)
```

Continued on next page.

Week 2 continued...

2. Write a procedure `substitute` that takes three arguments: a sentence, an *old* word, and a *new* word. It should return a copy of the sentence, but with every occurrence of the old word replaced by the new word. For example:

```
> (substitute '(she loves you yeah yeah yeah) 'yeah 'maybe)
(she loves you maybe maybe maybe)
```

3. Consider a Scheme function `g` for which the expression

```
((g) 1)
```

returns the value 3 when evaluated. Determine how many arguments `g` has. In one word, also describe as best you can the *type* of value returned by `g`.

4. For each of the following expressions, what must `f` be in order for the evaluation of the expression to succeed, without causing an error? For each expression, give a definition of `f` such that evaluating the expression will not cause an error, and say what the expression's value will be, given your definition.

```
f
(f)
(f 3)
((f))
(((f)) 3)
```

5. Find the values of the expressions

```
((t 1+) 0)                    ((t (t 1+)) 0)                    (((t t) 1+) 0)
```

where `1+` is a primitive procedure that adds 1 to its argument, and `t` is defined as follows:

```
(define (t f)
  (lambda (x) (f (f (f x)))) )
```

Work this out yourself before you try it on the computer!

6. Find the values of the expressions

```
((t s) 0)                     ((t (t s)) 0)                     (((t t) s) 0)
```

where `t` is defined as in question 2 above, and `s` is defined as follows:

```
(define (s x)
  (+ 1 x))
```

7. Write and test the `make-tester` procedure. Given a word `w` as argument, `make-tester` returns a procedure of one argument `x` that returns true if `x` is equal to `w` and false otherwise. Examples:

```
> ((make-tester 'hal) 'hal)
#t
> ((make-tester 'hal) 'cs61a)
#f
> (define sicp-author-and-astronomer? (make-tester 'gerry))
> (sicp-author-and-astronomer? 'hal)
#f
> (sicp-author-and-astronomer? 'gerry)
#t
```