

Try to get as much done as possible, but don't panic if you don't finish everything.

0. Login to your user account and change your password – instructions are provided on the account form. Be aware that it may take several minutes for your new password to be recognized by all the machines.

1. Start the Emacs editor, either by typing `emacs` in your main window or by selecting it from the alt-middle mouse menu. (Your TA will show you how to do this.) From the `Help` menu, select the Emacs tutorial. You need not complete the entire tutorial at the first session, but you should do so eventually.

2. Start Scheme, either by typing `scm` in your main window or by typing `meta-S` in your Emacs window. Type each of the following expressions into Scheme, ending the line with the Enter (carriage return) key. **Think about the results!** Try to understand how Scheme interprets what you type.

```
3                (first 'hello)
(+ 2 3)          (first hello)
(+ 5 6 7 8)     (first (bf 'hello))
(+)             (+ (first 23) (last 45))
(sqrt 16)       (define pi 3.14159)
(+ (* 3 4) 5)   pi
+              'pi
'+            (+ pi 7)
'hello        (* pi pi)
'+ 2 3)       (define (square x) (* x x))
'(good morning) (square 5)
(first 274)     (square (+ 2 3))
(butfirst 274)
```

3. Use Emacs to create a file called `pigl.scm` in your directory containing the Pig Latin program shown below:

```
(define (pig1 wd)
  (if (pl-done? wd)
      (word wd 'ay)
      (pig1 (word (bf wd) (first wd)))))

(define (pl-done? wd)
  (vowel? (first wd)))

(define (vowel? letter)
  (member? letter '(a e i o u)))
```

If you end each line with the linefeed key, instead of the return key, Emacs will automatically indent the lines of your program properly.

4. Now run Scheme. You are going to create a transcript of a session using the file you just created, like this:

```
(transcript-on "lab1") ; This starts the transcript file.
(load "pig1.scm")     ; This reads in the file you created earlier.
(pigl 'scheme)        ; Try out your program.
                      ; Feel free to try more test cases here!
(trace pig1)          ; This is a debugging aid. Watch what happens
(pigl 'scheme)        ; when you run a traced procedure.
(transcript-off)
(exit)
```

5. Use `lpr` to print your transcript file.

Continued on next page.

Lab Assignment 1.1 continued...

6. Predict what Scheme will print in response to each of these expressions. *Then* try it and make sure your answer was correct, or if not, that you understand why!

```
(define a 3)
(define b (+ a 1))
(+ a b (* a b))
(= a b)
(if (and (> b a) (< b (* a b)))
    b
    a)
(cond ((= a 4) 6)
      ((= b 4) (+ 6 7 a))
      (else 25))
(+ 2 (if (> b a) b a))
(* (cond ((> a b) a)
        ((< a b) b)
        (else -1))
   (+ a 1))
((if (< a b) + -) a b)
```

7. In the shell, type the command

```
cp ~/cs61a/lib/plural.scm .
```

(Note the period at the end of the line!) This will copy a file from the class library to your own directory. Then, using emacs to edit the file, modify the procedure so that it correctly handles cases like `(plural 'boy)`.

8. Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

9. Write a procedure `dupls-removed` that, given a sentence as input, returns the result of removing duplicate words from the sentence. It should work this way:

```
> (dupls-removed '(a b c a e d e b))
(c a d e b)
> (dupls-removed '(a b c))
(a b c)
> (dupls-removed '(a a a a b a a))
(b a)
```