

Part A: Abelson and Sussman, exercises 4.27 and 4.29.

Part B: In this lab exercise you will become familiar with the Logo programming language, for which you'll be writing an interpreter in project 4.

To begin, type `logo` at the Unix shell prompt — **not** from Scheme! You should see something like this:

```
Welcome to Berkeley Logo version 3.4
?
```

The question mark is the Logo prompt, like the `>` in Scheme. (Later, in some of the examples below, you'll see a `>` prompt from Logo, while in the middle of defining a procedure)

1. Type each of the following instruction lines and note the results. (A few of them will give error messages.) If you can't make sense of a result, ask for help.

<code>print 2 + 3</code>	<code>second "something</code>
<code>print 2+3</code>	<code>print second "piggies</code>
<code>print sum 2 3</code>	<code>pr second [another girl]</code>
<code>print (sum 2 3 4 5)</code>	<code>pr first second [carry that weight]</code>
<code>print sum 2 3 4 5</code>	<code>pr second second [i dig a pony]</code>
<code>2+3</code>	<code>to pr2nd :thing</code>
<code>print "yesterday</code>	<code>print first bf :thing</code>
<code>print "julia"</code>	<code>end</code>
<code>print revolution</code>	<code>pr2nd [the 1 after 909]</code>
<code>print [blue jay way]</code>	<code>print first pr2nd [hey jude]</code>
<code>show [eight days a week]</code>	<code>repeat 5 [print [this boy]]</code>
<code>show first [golden slumbers]</code>	<code>if 3 = 1+1 [print [the fool on the hill]]</code>
<code>print first bf [she loves you]</code>	<code>print ifelse 2=1+1 ~</code>
<code>pr first first bf [yellow submarine]</code>	<code>    [second [your mother should know]] ~</code>
<code>to second :stuff</code>	<code>    [first "help]</code>
<code>output first bf :stuff</code>	<code>print ifelse 3 = 1+2 ~</code>
<code>end</code>	<code>    [strawberry fields forever] ~</code>
	<code>    [penny lane]</code>
	<code>print ifelse 4 = 1+2 ~</code>
	<code>    ["flying] ~</code>
	<code>    [[all you need is love]]</code>

Continued on next page...

Lab Assignment 7.1 continued...

```
to greet :person
say [how are you,]
end

to say :saying
print sentence :saying :person
end

greet "ringo

show map "first [paperback writer]

show map [word first ? last ?] ~
    [lucy in the sky with diamonds]

to who :sent
foreach [pete roger john keith] "describe
end

to describe :person
print se :person :sent
end

who [sells out]

print :bass

make "bass "paul

print :bass

print bass

to bass
output [johnny cymbal]
end

print bass

print :bass

print "bass

to countdown :num
if :num=0 [print "blastoff stop]
print :num
countdown :num-1
end

countdown 5

to downup :word
print :word
if empty? bl :word [stop]
downup bl :word
print :word
end

downup "rain

;;; The following stuff will work
;;; only on an X workstation:

cs

repeat 4 [forward 100 rt 90]

cs

repeat 10 [repeat 5 [fd 150 rt 144] rt 36]

cs repeat 36 [repeat 4 [fd 100 rt 90]
    setpc remainder pencolor+1 8
    rt 10]

to tree :size
if :size < 3 [stop]
fd :size/2
lt 30 tree :size*3/4 rt 30
fd :size/3
rt 45 tree :size*2/3 lt 45
fd :size/6
bk :size
end

cs pu bk 100 pd ht tree 100
```

2. Devise an example that demonstrates that Logo uses dynamic scope rather than lexical scope. Your example should involve the use of a variable that would have a different value if Logo used lexical scope. Test your code with Berkeley Logo.

3. Explain the differences and similarities among the Logo operators " (double-quote), [ ] (square brackets), and : (colon).