MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.081—Introduction to EECS I
Spring Semester, 2007

Work for Week 2

Issued: Tuesday, Feb. 13

This handout contains:

- Software Lab for Tuesday, February 13
- Pre-Lab exercises due Thursday, February 15 at 2PM; you should come and do them in lab on Wednesday, February 14.
- Robot Lab for Thursday, February 15 (read this before coming to Lab)
- Post-lab exercises due *Thursday* Feb. 22. There is no lecture on Tuesday, Feb. 20 (MIT virtual Monday).

Higher-order procedures; Behaviors and utility functions

This week's work gives you practice with *higher-order procedures*, i.e., procedures that manipulate other procedures. Tuesday's lecture and post-class lab cover Python's support for *functional programming*, including *list comprehension*. Wednesday's homework and Thursday's lab apply higher-order procedures to the tasks of specifying robot *behaviors* with the aid of *utility functions*.

Tuesday Software Lab: Higher-order procedures

This session gives you practice with Python's basic tools for functional programming: higher-order procedures and list comprehension.

A higher-order procedure is a procedure that takes takes procedures as inputs and/or returns procedures as results. The basic example we'll work with now is a procedure called `plot` which takes a function `f` as input and graphs it.

Start by loading the file `feb-13-class.py`. It begins with the following code:

```
# import the graphing package from SoaR
import SoaR
from SoaR.Util.GraphingWindow import GraphingWindow

# import sine and cosine
import Numeric
from Numeric import sin
from Numeric import cos

# import the version of addition for use with reduce
import operator
from operator import add as add

# set up a variable for the plotting window
plotWindow = None
```

```
# plot a given function f
def plot(f):
    global plotWindow
    if plotWindow: plotWindow.close()
    xmax=10
    xvals = [i/100. for i in range(0,int(xmax*100))]
    yvals = [f(x) for x in xvals]
    ymax = reduce(max,[abs(y) for y in yvals])
    graphmax = 1.1 * ymax
    plotWindow = GraphingWindow(500, 500, 0, xmax, -graphmax, graphmax, "Plot")
    plotWindow.graphContinuous(f)
    plotWindow.helpIdle()
```

The `plot` procedure opens a graphing window and plots `f` in the interval from 0 to 10. The system then waits (hangs) until you close the plotting window with the mouse.[1] Note that the input `f` to `plot` is a procedure.

Question 1. What do you expect `plot(lambda x: x)` to produce? Try it and see.

Question 2. Make a plot of the sine function. Do you have to write `plot(lambda x: sin(x))` or can you simply write `plot(sin)`? Try it and see. Make sure to ask if the result isn't what you expect.

Observe how `plot` uses list comprehension and `reduce` with `max` to compute the ranges for the axes. If you've programmed before, you probably expected that this would be done using some kind of loop. One of the hallmarks of functional style is to use functions operating on sets of elements, rather than writing explicit loops with `for` or `while`.

Now let's plot something more interesting—superpositions of sine waves of increasing frequency:

$$\sin x + \frac{1}{2}\sin 2x + \frac{1}{3}\sin 3x + \frac{1}{4}\sin 4x + \cdots$$

out to more and more terms.

Let's call $saw(n)$ the function whose value at x is the sum of the first $n-1$ terms of this sequence. Then we can compute *saw* as

```
def saw(n):
    return lambda x: reduce(add,
                            [1.0/j * sin(j*x) for j in range(1,n)])
```

(This definition is included in the file you loaded.) Here `saw` is a procedure that takes n as input and returns a function of x, which we can pass to `plot`. Observe again how we've used list comprehension and `reduce` with `add` rather than write an explicit loop to sum the terms.

Question 3. Plot $saw(n)$ for various values of n, starting with $n = 2$, which is just a sine wave (do you see why?) up to large values of n, like 100. It should become obvious why we've named the procedure *saw*.

[1] We're embarrassed about setting things up this way. It's a kludge to get around a bad interaction between the SoaR robot system and Python's Idle editor.

As you'll learn later (in 18.03 and/or 6.003), any periodic function can be expressed as a *Fourier series*, i.e., a superposition of sines and cosines of the form

$$a_1 \sin x + b_1 \cos x + a_2 \sin 2x + b_2 \cos 2x + \cdots + a_k \sin kx + b_k \cos kx + \cdots$$

Question 4. Define and plot the function *squareWave*(n) that produces partial sums of the Fourier expansion of a square wave:

$$\sin x + \frac{1}{3}\sin 3x + \frac{1}{5}\sin 5x + \frac{1}{7}\sin 7x + \cdots$$

Do this with list comprehension, using a construct of the form:
`[<expression in j> for j in <range> if <condition>]`

Question 5. Do the same thing for a triangle wave, whose Fourier expansion is

$$\cos x + \frac{1}{9}\cos 3x + \frac{1}{25}\cos 5x + \frac{1}{49}\cos 7x + \cdots$$

Now let's rewrite these programs using an "even more functional" style, similar to what you'll do in lab this week.

The following procedure **addf** (included in the file you loaded) takes as inputs two functions f and g and returns a function whose value at any x is $f(x) + g(x)$:

```
def addf(f,g):
    return lambda x: f(x)+g(x)
```

The important thing to observe is that the inputs to **addf**, as well as the result, are *functions*, i.e., **addf** transforms functions to functions.

Similarly, we have **scalef**, which takes a function f and a number s and return the function whose value at x is $s \times f(x)$, and **expandf**, which produces the function whose value at x is $f(s \times x)$.

Convince yourself that an equivalent way to write the **saw** procedure is

```
def saw2(n):
    return reduce(addf,
                  [scalef(expandf(sin,j),1.0/j) for j in range(1,n)])
```

and make some plots. The definitions of **addf**, **scalef**, **expandf**, and **saw2** are included in the file you loaded.

Question 6. Write similar definitions for square waves and triangle waves and check the results by plotting them.

For the rest of this period, play around with these procedures as you have time, doing things that interest you. Here are some ideas to try. You needn't do all of them.