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

Work for Week 1

This handout contains:

- Software Lab for Tuesday, February 6
- Pre-Lab exercises due Thursday, February 8 at 2PM; you should come and do them in lab on Wednesday, February 7. **This week: Everyone should come and bring their laptop, if they have one. We'll have a software installation party.**
- Robot Lab for Thursday, February 8
- Post-lab exercises due Tuesday, February 13 at 2PM

All pre- and post-lab exercises are to be handed in individually. Please see the collaboration policy on the web site for more information.

Getting started

Schedule In general, the weekly schedule for 6.081 will be like this:

- Tuesday lecture from 2–3:30, followed by a software lab from 3:30 - 5:00. We will also hand out the homework for the coming week. The homework will include a part that is due on Thursday before lab and a part that is due the following Tuesday before lecture. The homework will have parts to write up and turn in, as well as parts to be done with the online tutor.

- Wednesday evening homework help session from 6–10. Attendance at these sessions is strongly recommended. You are welcome to do the homework for Thursday on your own if you prefer, or in a self-organized study group. But we suggest that your time will be much better spent if you do your pre-lab homework in the staffed Wednesday sessions. When programming, especially, it's easy to get lost in rat-holes and spend enormous amounts of time digging yourself out. Doing your work when there are staff members around can make things much easier.

- Thursday afternoon lab. Each lab will end with a short quiz based on the pre-lab assignment as well as the material covered during the lab.

- Post-lab homework, due before the following Tuesday lecture.

Reading This is such a cool new way of looking at the material of intro EECS, there is no textbook. We will be producing draft notes each week, and those will be your primary reading resource. In addition, for Python, we recommend finding a basic language reference that you like and also reading *How to Think Like a Computer Scientist*.

- Official Python tutorial: `http://http://docs.python.org/tut/tut.html`

- *How to Think Like a Computer Scientist: Learning with Python*, by Allen Downey, Jeffrey Elkner, Chris Meyers. This book assumes no programming experience. It is not a Python reference manual; it is a computer science text that uses Python as an example. Highly recommended. `http://www.greenteapress.com/thinkpython/thinkCSpy.pdf`
- *Learning Python*, by David Ascher and Mark Lutz. This book also assumes very little/no programming experience, and is longer (not so great for learning Python in a hurry, but covers topics in great detail, so is good as a reference guide if you know what you're looking for). `http://proquest.safaribooksonline.com.libproxy.mit.edu/0596002815?tocview=true` (You need an MIT Certificate to view this one)
- *Learn Python in 10 Minutes*, by Poromenos. If you have very little time and a fair bit of programming experience in another language, this tutorial covers Python's syntax quickly. `http://www.poromenos.org/tutorials/python`
- Look at some of the other reference material on the class web site under "Resource Material" (under the "General Information" menu at the top of the web page).[1]

Software Lab

Starting and using IDLE

IDLE is a simple way to edit and run Python programs. To start it on one of the lab laptops, go to a shell and type

```
> idle-python2.4
```

Python Shell The Python shell acts a little bit like a calculator. You type in expressions, Python evaluates them, and then prints the result. So, if you type

```
>>> 4 + 4
```

It will print out `8`. Play with the shell a little bit.

Edit a file You'll want to use the shell to test things out, but not to write your programs. If you want to start defining procedures, you should open a new file (choose `New Window` from the `File` menu) and write your definitions in there. So, start by making a file containing this definition:

```
a = 'hi'
b = 7
def f(x):
    return x + 1
```

Now, choose `Run Module` from IDLE's `Run` menu. It will act as if you have typed the text of your file into the shell. If there was something obviously syntactically wrong with your file (parentheses not closed, for example), IDLE will tell you about it and highlight the point in your file where the problem is. Otherwise, the shell will print out something like

[1] If you use Windows with Internet Explorer as your web browser, you won't see the drop-down menu under "General Information". There's a "Resource Material" link at the bottom of the home page that you can follow instead.

```
>>> ================================ RESTART ================================
>>>
```

And now you can ask it to evaluate expressions, including things you've defined in your file.

Question 1. Use the Python shell to compute `f(f(f(b)))`.

Question 2. What happens if you do `f(a)`?

To do the following exercises, use the shell for experimentation, and write new procedure definitions in your file. Whenever you change your file, you'll need to do `Run Module` again.

Archive your file When you do work on a lab laptop, you should always remember to mail or FTP your files back to yourselves. We don't guarantee that you'll always get the same laptop or that any files that you leave on it will remain there from week to week.

An easy way to keep your files safe is to use a webmail service. You may have a service such as Gmail or Hotmail before. We'll explain the process using MIT webmail.

To use MIT webmail, first, you go to: `http:\\webmail.mit.edu`. To log in, type your Athena account user name and its password. You can compose an email to your email address (`user@mit.edu`, where `user` is your account name), and attach your files.

Another way to protect your files is to copy the files from a lab laptop to your Athena account. You can do this by using the `scp` command. General usage of `scp` (when copying the file to the remote computer) is as follows.

```
file user@machine:file
```

For example, if you want to copy the file `lab1.py` to the directory `Courses/6.081` on your Athena account `user1`, you type:

```
scp lab1.py user1@athena.dialup.mit.edu:Courses/6.081/lab1.py
```

Note that the directory specified in the destination (`Courses/6.081`) must have been created before copying a file into it. Also note that `scp` command automatically overwrites the file you specified as the destination, without asking your permission. So, you have to be careful not to overwrite your important files using `scp`.

When you want to copy the file from Athena to a lab laptop, you can simply switch the destination and the source for `scp`:

```
scp user1@athena.dialup.mit.edu:Courses/6.081/lab1.py lab1.py
```

Exercises

Start by reading the sections of the Lecture 1 notes describing the new features in Python.

Question 3. What expression can you write to name a list with no elements?

Question 4. In Python, you can use `+` to concatenate two lists. Give an expression for adding the element 6 to the end of a list `y`.

Question 5. Write the procedure `range1`. It takes a single positive integer, `n`, as a argument and returns a list with elements starting with 0 and going up through `n-1`. Don't use `range`!

Question 6. Write a procedure that takes a list as input and returns a new list with every other element of the original list, starting with the first. You can do this using the things you already know; or you can go read the documentation about fancier uses of `range`.

If you get this far and still have time, start in on the post-lab, beginning with question 11.

Pre-Lab Exercises: To be done in lab on Feb 6

- Obtain a lab notebook (or just paper in a binder).
- Install Python, SoaR, and IDLE on your computer (If you really can't make it, follow the instructions at `http://courses.csail.mit.edu/6.01/spring07/software/`), or find a way to get access to an Athena computer that has these installed.
- Read this whole document.

SoaR

If you're using a lab laptop, start up SoaR by typing

```
> SoaR
```

in a shell. If you're using your own laptop, follow the instructions for the OS you're using. Load the simulator (press *Simulator*) and one of the worlds (for instance, `tutorial.py`), and drive the robot around using the joystick (press *Joystick* and then *Start*).

Now, we'll use a program to control the simulated robot. Select *Brain* and then `testBrain.py`. Then hit *Start*. The robot should zoom around the simulated world, avoiding obstacles.

Using the online tutor

Register for the online tutor at `http://sicp.csail.mit.edu/6.081` and do the exercises that are due for February 8.

Robot Lab

This lab is organized into three sections, each of which contains some checkpoint questions. Be sure to show your work to a staff member at the end of each checkpoint and be ready to explain your answers. There are some extra suggestions for explorations to do if you have more time.

Basics

Let's start by making the robot move. You can drive the robot around by hand, using the joystick. Plug your robot's serial cable into the robot and into your laptop, and turn the robot on. Now start SoaR by typing

```
> SoaR
```

then press *Pioneer*, and *Joystick* and *Start*. Click and drag in the resulting Joystick window to control the robot. This works with the simulator as well as with the real robot.

Our robots have *sonar* sensors for measuring the distance to obstacles in different directions and *odometry* sensors for measuring how far the robot has moved. It's easy to understand idealized sensors, but their actual behavior is more complicated. The goals of this lab are: to become familiar with reading the robot's sensors, and to get the robot to move. We also would like for you to come to appreciate that real-world sensors and effectors don't usually behave in the idealized way you might wish they would.

Brains for Beginners

SoaR interacts with the robot (real or simulated) via a *brain program*.

A brain has the following structure:

```
def setup():
    print "Starting"
def step():
    print "Hello robot!"
```

Your brains should have this form, but you don't need to worry right now about anything except the `step` function. This function will be called by SoaR about 10 times per second. The above brain, if run, will start by printing `Starting`, and then just keep printing `Hello robot!` in the SoaR message window until you stop it.

Try saving this code as `reallysimplebrain.py` in the `brains` subfolder inside the `SoaR` folder and running it by opening the simulator in SoaR, choosing *Brain* instead of *Joystick*, selecting *reallysimplebrain.py*, and finally hitting *Start*.

As you can see, this brain is boring. Don't worry if you click the stop button and it keeps printing for a while. All that text has been stored up somewhere waiting to be printed.

Sonar

The robot has a set of eight ultrasonic transducers (familiarly known as sonars). They send out a pulse of sound and then "listen" for a reflection. The raw sensors return time-of-flight values, which is how long it took the sound to bounce back. The robot's processor converts these into distance estimates.

The command `sonarDistances()` returns a list containing the robot's current sonar readings, all in meters. Try saving the following code (as say, `"simplebrain.py"`) and running it on both the simulator and the real robot:

```
def step():
    print "sonarDistances:", sonarDistances()
```

This brain prints out the current sonar readings on every step. Play with putting something in front of different sensors and seeing how the values change. Which value corresponds to which sensor?

Motion

You can move the robot using the function `motorOutput(`*fvel*`,`*rvel*`)`, where *fvel* is the forward velocity of the robot and *rvel* is a rotational velocity. Experiment with this function a bit by adding it to a brain, just to get a feeling for how it works (but be careful not to crash the robot into anything—a velocity of 1 is *fast*, so use something a lot less). `motorOutput(0,0)` will make the robot stop.

A well-formed brain should have *exactly* one call to `motorOutput` on every step. The idea is that the brain will decide, on each step, how fast the robot should be moving.

Checkpoint (should be completed by 3:00 PM)

Demonstrate that you can

- Write a brain that will move the robot forward.
- Write a brain that will make the robot rotate.
- Write a brain that will print out the values of the front two sonar sensors.

Robot behavior

Now we'll explore integrating sonar and motion to do some interesting things.

- Write a brain that will move the robot forward until it is 30cm from an obstacle in front of it.
- Augment your first behavior so the robot backs up if something gets closer than 30cm.
- Write a brain that will try to move along, parallel to a wall next to it.

If you have time left over, write a program that can navigate the robot through a small opening in a wall in front of it, based on the sonar readings. It's fine for the robot to be pointed fairly directly at the opening. Start with a really big opening and then gradually make it smaller. See how small you can go! Don't worry if you have trouble getting this to work. It's hard to do it robustly, and we'll spend a lot of time in the coming weeks studying strategies for solving problems like this.

Checkpoint (4:15 PM)

Demonstrate your programs to your LA or TA. Be ready to explain how they work and why you made them the way you did.

Sonar Experiments

This section is optional, in that you shouldn't worry if you don't have time to do all (or any) of it.

In this section, we'll experiment systematically with the sonar sensors to see how they behave. You can think of the transducer as emitting a cone-shaped beam of sound waves, which are reflected off surfaces back to the detector. The physical details of the shape of the cone and the way the reflections work imply that sonar readings are often not what one would expect.

Checkpoint (4:45PM)

- What happens when something is very close to the sonar?
- What happens when the closest thing is very far away?
- How do the results vary as a function of the angle between the transducer and the surface that it is pointed toward?
- How do the results vary as a function of the material of the surface? Try bubble wrap: what's going on?
- How wide is the beam?

Post-Lab Exercises

All post-lab hand-ins should be written in clear English sentences and paragraphs. We expect at least a couple of sentences or a paragraph in answer to the reflection questions below. We're in interested in an explanation of your thinking, as well as the answer.

Reflection on lab results

Question 7. How well would your robot behaviors have worked if the step function were called once per second, rather than 10 times per second? What if it were called once per minute?

Question 8. Why do you think putting bubble-wrap on objects make the sonar readings more reliable?

Question 9. Imagine that the robot is driving forward toward a wall with an open door in it. When might it have trouble seeing the door?

Question 10. Imagine that the robot is driving toward a thin pole. When might it have trouble seeing the pole? What can you say about the accuracy with which it could determine the pole's position relative to the robot?

Arrays

Python doesn't have a formal array data type (though there are some very nice libraries for numerical computation, such as `numpy`, that support arrays). But we can make two-dimensional arrays using lists of lists.

Question 11. Write a procedure `zeroVector(n)` that makes a one-dimensional array of dimension `n` filled with zeros. That is, it should be a list of `n` zeros.

Question 12. If you did `v = zeroVector(10)`, what expression would you use to change the third element to have value 6?

Question 13. Write a procedure `zeroArray(m,n)` that makes a two-dimensional array of dimension `m` by `n` filled with zeros. That is, it should be a list of `m` lists, each of which is a list of `n` zeros.

Question 14. If you did `a = zeroArray(3, 6)`, what expression would you use to change the element at coordinates (1,4) to have value 100? (Assume that the coordinates start at 0).

Question 15. Monty P decided to write these solutions to the previous problems:

```
def zeroVector(n):
    return [0]*n

def zeroArray(m,n):
    return [zeroVector(n)]*m
```

They have a certain charm and elegance about them. But also a certain problem. What happens if Monty makes array, and then sets one value to be 3? Can you explain why?

Question 16. Write a procedure `addElements(a)` to add up all of the elements in a two-dimensional array. Do it in two different ways. Which one do you think is better?

Association Lists

The idea of a dictionary is very important and useful in programming. A dictionary allows you to associate *values* with *keys*. In an actual English dictionary, the words are the keys and the definitions are the values. So, given a key, you can look up the value. In a phone book, names are the keys and phone numbers are the values.

Python has a built-in dictionary data structure, which you can read about in the online documentation. We can make our own, simpler and less efficient data structure as a list of lists. It operates like this:

```
d = emptyAlist()
>>> addEntry(d, 4, 5)
>>> addEntry(d, 5, None)
>>> addEntry(d, 'ben', 'boa')
>>> addEntry(d, 'kim', 'krait')
>>> d
[[4, 5], [5, None], ['ben', 'boa'], ['kim', 'krait']]
>>> addEntry(d, 'bella', 'bi-colored-python-rock-snake')
>>> d
[[4, 5], [5, None], ['ben', 'boa'], ['kim', 'krait'],
 ['bella', 'bi-colored-python-rock-snake']]
```

```
>>> lookup(d, 'ben')
['ben', 'boa']
>>> lookup(d, 'biz')
>>>
```

Question 17. Write the `emptyAlist`, `addEntry`, and `lookup` procedures.

Question 18. Why do we have lookup return both the key and the value? (Think about what would happen, in the example alist above if we did `lookup(d, 5)`?)

Question 19. Roughly how much longer will it take to look something up in an alist with $2n$ elements than one with n elements? Why?

Side effects

There are lots of other things you can do with a list. If `y` is a list (for instance, you do `y = [1, 2, 3]`), and you type `dir(y)`, you get a huge list of things that you can do with a list. Most won't make sense now, but you can see, for example that `sort` is something you can do to a list.

Question 20. What happens if you do

```
>>> l = [4, 2, 1]
>>> l.sort()
>>> l
```

Now, try

```
>>> l = [4, 2, 1]
>>> sorted(l)
>>> l
```

What's the difference between these two ways of sorting? When would you prefer to use one method as opposed to the other?

Quadratic roots

Question 21. Write a procedure `quadraticRoots(a, b, c)`, which returns a list containing both roots of the quadratic equation $ax^2 + bx + c = 0$. You can use `x**0.5` to compute the square root of `x`.

Question 22. What answer do you get for `quadraticRoots(1, 2, 3)`? Why?

Question 23. Python actually has complex numbers as a primitive data type. There are two ways to make a complex number:[a]

```
>>> complex(1, 2)
(1+2j)
>>> 1+2j
(1+2j)
```

Use Python to compute $\sqrt{-1}$.

Question 24. Change your `quadraticRoots` procedure so that it returns complex roots (but still takes floats as input).

[a] You're probably used to using i for $\sqrt{-1}$. Just to confuse you, we're going to use j instead. Why? Because to an electrical engineer, i means current, and that's that.