Lecture 2

Steven Walton
PS 232 Computational Methods
Department of Physics
Embry-Riddle Aeronautical University
Prescott, AZ 86301

September 3, 2014

Importing A Module

Now that we have talked about modules, let's use them. One command that we are going to have to know is `import`. What this does is tells python that you care going to use the module specified. So for example, let's create a random array.

```
import numpy                    # We want to import the module numpy for
                                # the command random
s = numpy.random.random(5)      # Take special note to how this is called.
                                # Do you have to do it this way?
print s
```

You will see here that we are outputting the value of s, which is an array of 5 random numbers between 0 and 1. Notice the tricky structure of how we called this this random command. We had to use all those periods. What this means is that we are using the module NumPy, then using the library of Random, and the command random in it. This can be tiresome to write over and over, especially if we only want to be calling random from NumPy and nothing else. We have a way to deal with this, and to take it one step further. See if you can understand the following code.

```
from numpy import random as ran
s = ran.random(5)
print s
```

So instead of importing all of numpy, we saved some time by just importing the random library. We also used "as" to shorten the name. This can be a big help when we are using different modules. Let's do a better example, we'll create 10 random points and plot them.

```
from numpy import random as ran
import matplotlib.pyplt as plt      # Python's plotting module

x = ran.random(5)                   # remember it's between (0,1)
y = ran.random(5)

plt.plot(x,y, 'ro')                 # Extremely similar to MATLAB's
                                    # plotting function
plt.show()
```

Now we see two ways to actually do the same thing, since pyplt is a library in matplotlib. Now you may be asking what the third parameter of the plot command is ('ro'). The r means that it will plot in red, and the o means that it will plot circles. The default for plt is that it will connect the points with a line and the colour will be blue. See the manual for other options, it is extremely similar to MATLAB's plotting function.

Types and Arrays

One of the difficult things in programming is memory management. In something like C++ you have to define things like character, float, long, boolean, or others. Python does a lot of this for you. If we want to set a variable a to 100 we can do that by `a=100`. If we want to set a to a sentence `a = "We do it like this."`. For now we won't address these issues because python will take care of most of this for you.
We will address arrays though, since they will be more useful to us in a mathematical programming perspective. Again, python will automatically take care of this type for you without having to worry about memory management. Let's make a list of five numbers, from 0 to 5. Then what we want to do is output the number three. We can do that quite easily.

```
list = [0,1,2,3,4,5]    # Makes an array from 0-5
print list[3]           # Prints 3
```

Pay careful attention to how this works, the listing starts at 0. So `list[0]` outputs 0, and `list[5]` outputs 5. It is important to remember that the ordering starts from 0, and not 1. Now let's say that we want to create a matrix. For this we are going to create a 3x3 matrix, then output the second row, and the the number in the third row and third column. Play with this and make sure that you are able to access every part of the matrix.

```
matrix = [[1,2,3],[4,5,6],[7,8,9]]  # 3x3 matrix
print matrix[1]                     # prints second row
print matrix[:][1]                  # also prints second row
print matrix[2][2]                  # prints 9
```

For a more friendly looking version of this same thing we can use numpy's matrix command. After importing it, there are two ways we can make the same exact matrix.

```
np.matrix('1 2 3; 4 5 6; 7 8 9')
np.matrix([[1,2,3],[4,5,6],[7,8,9]])
```

You'll notice that the output of these is much friendlier looking. We can use the same structure to accessing the elements. There are some key differences between these two functions. One useful thing about the normal array in python is that we can append it and remove elements from it. So let's add the number 1 to the end of our matrix, and then remove it. (Calling it a matrix is misleading, as you'll see. But it is close, that we can think of it this way for now).

```
matrix.append(1)      # [[ 1, 2, 3], [4, 5, 6], [7, 8, 9], 1]
matrix.pop(3)         # removes 1 (4th element)
```

This is extremely useful if we need an array that needs to be updated. We can not do this with the matrix from numpy (try it yourself). What is nice about the matrix command is that we have all the same matrix commands avaliable to us. (See numpy.matrix)