Contemporary Physics 2007 Q1

Rec #1A : The Big Introduction

Welcome to Contemporary Physics, a class specifically designed to explore the physical nature of our universe. In lecture, Dr. Goldberg will teach you the theory and concepts. In recitation we will explore these ideas by programming models with the computer. Before we get started we need to understand some computing basics. This assignment is meant as a gentle introduction to Unix, and Python. Please ask questions at any step of the way.

1 Python : Introduction

Write the output of each step. You will turn in this paper for credit (put your name on it). It is very important that you complete and understand each step of this assignment; these skills will be expected of you in subsequent classes.

To use python on the command-line, type:

```
python
```

You should be at a prompt that looks like this: >>> While it has many other functions, python can be used as an over-grown calculator. Type

```
>>> 5+3
```

and press enter. Notice it returns exactly what you expect (write down the result next to each line). Lets try some more:

```
>>> 28*96
```

Nothing to complicated here. Python does have the nice capability of handling arbitrarily large integers, for example lets try 2^{80}.

```
>>> 2**80
```

The ** operator is short for exponentiation. Lets save some of these results. You can define a variable by:

```
>>> a=20
```

Here a is now defined to be the value of 20. Set $b = 50$. Anything that you can do with numbers, you can do with variables. For example try:

```
>>> a-b
```

There is an important difference between integer and decimal division, try this example:

```
>>> 5/2
```

```
>>> 5/2.
```

Every time you exit python, you will lose all your saved variables. However, you can write a python program that acts as a single session. Exit python by pressing Ctrl+D at the same time. Using Emacs open a new file called rec1.py.

```
emacs rec1.py &
```

The & allows you to use both the command-line and emacs at the same time. Lets create a simple program. Type the following into emacs and save the file. Go back into the command line and type:

```
 python rec1.py
```

to run the program.

Your first python program (write its output below):

```
a=3
b=6
print a,b
print a>b, a<b, a==b
```

The print command tells python to output the result to the screen. >, < and == are comparisons for "greater than", "less than" and "equals to" respectively. Take note that = means set equal to, and == means check to see if they are equal (a subtle difference, but one that comes up for beginning programmers). Go back to the emacs windows (rec1.py) and change the code and rerun it:

```
a=3
b=6
#print a,b
#print a>b, a<b, a==b

K = [1,7,8,10]
P = range(5)
print K
print K[0], K[1], K[3]
print
print P
```

Note that the previous lines were not printed out. The # tells python that the line is a comment, i.e. it will be ignored by the machine. K and P are lists. Lists are collections of objects, a handy tool that we will use many times. To access an element of a list, we can index it by $K[n]$, where n is the sequential element we want to grab. Notice that the first element is labeled $n = 0$ **not** $n = 1$. The command *range* creates a special list, one that is sequential, ordered from $0...n$. Lets add some more to our code by introducing loops. Append the following lines to the bottom of the code and rerun it again:

```
J = [ ]
for g in K:
    print g
    J.append( g**2 )
print J
```

If the code does not run, make sure that you have correctly indented the lines following the for statement. The first line creates an empty list, we will be adding something to it later. The for command is a loop. The variable g at each iteration of the loop is sent to the next value of K. The *append* command adds the value of g^2 sequentially to J. If you want you can plot this result (sketch the plot below).

```
from pylab import *
plot(K,J)
show()
```

The $from$ line tells python to use all the utilities in *pylab*. It only needs to be included once in the code, even if you have multiple plots. The *plot*(X, Y) command plots X vs. Y (or in our case K vs J). The *show* command displays the output to the screen. Notice there are several buttons on the plotting screen. The one that is disk-shaped allows you to save your graph to a file. Easy enough. Let's move on to the real assignment.

2 Projectile Motion : Not quite rocket science yet

To get your feet wet, and to start thinking in (programming) code we will start with a a model where the solution is already known. The solution for 2D projectile motion (with no air resistance) is simply:

$$x(t) = x_0 + v_0 \cos(\theta) t \tag{1}$$

$$y(t) = y_0 + v_0 \sin(\theta) t - \frac{g}{2} t^2 \tag{2}$$

Given that the particle starts at the origin, and is launched with an initial velocity vector $\vec{v} = (3, 4)\ m/s$, plot $x(t)$ vs $y(t)$ for time steps $\Delta t = .1\ s$. Save this plot to a file. Add your name and rec # as a comment to the python code.

To complete this assignment:

- Turn in the first sheet with your hand-written answers before the end of class.
- Get the above code to terminate when the particle is below the x-axis (i.e. when it hits the ground).
- Email your code and graph for **Projectile Motion** as an attachment to **hoppe@drexel.edu**. Make sure your name is included in the code. You will lose points for any assignment that does not include your name. I repeat, Put your name on your code and all turned in assignments.