Computer Lab 1b - Python Tutorial

The simulation programs we will use in the first half of this course are all Python based. Python is a high-level programming language such as FORTRAN, C, Java or Perl. Unlike FORTRAN or C it is an interpreted language. This means that you do not compile the code and then run a machine language executable. Rather you run the source code file as the program and the computer interprets this on the fly (the computer does some intermediate level compiling that you do not need to be concerned with).

Python has highly readable source code and yet includes very powerful object oriented methods. Because it is interpreted, it is slower than a compiled language but Python makes it very easy to call C language executable modules for compute-intensive routines.

The goal of today's lab is to become familiar with the syntax of common Python instructions so that we will be able to understand Python code and do modest editing of Python programs. This is by no means a comprehensive tutorial. We will be using Python version 3; most computers have Python installed but it may be version 2 that has some syntax differences. If you want to practice on your own laptop I can help you install the current Python version.

There are many resources to help you learn Python both online (http://www.python.org/) and in hardcopy books (http://python.oreilly.com/).

For today's lab you should log on to a Mac using your JHED-ID and password, open a terminal window type **`tcsh`** (unless you have setup your terminal app to default to tcsh), and change (**`cd`**) to the */Volumes/[your_FlashDriveName]/lab1* directory (or just subdirectory *lab1/* in your home directory).

Launch a second terminal window, and cd to the *lab1/* directory. (The second window can be launched by typing ⌘, **`n`** while the cursor is in the first terminal window.) In the first window you will edit a python script (or code) file, and in the second you will run that python script file after each edit.

Answer the questions highlighted in yellow and send the answers to achin14@jhu.edu either in the body of an email or as an attached file with your JHEDID in the name (e.g. *JHEDID_lab1b.txt* if made with vim or *JHEDID_lab1b.docx* if made with MSWord).

I. Values and Types

A value is a letter or number that a program manipulates. Values containing letters are called strings (`str`) and values of numbers are called either integer (`int`) if they have no decimals or `float` if they have decimals. In the first terminal window launch a **vim** editing session with a file called *lab.py* (**`vim lab.py`**). Insert (**i**) the following line and save the file (**`:w`**), but don't quit the editing session,

```
print("Hello, World!", type("Hello, World!"))
```

then in a second terminal window, change to the same directory as the editing session and type,

```
python3 lab.py
```

You should see the string, `Hello World!` appear on the screen followed by its value type in brackets `<type 'str'>.`

Now insert the following lines in your *lab.py* file. (Type **o** with the cursor on the single line and you will be in insertion mode on a new line.) After each line save the file (**:w**), run the program (**python3 lab.py**) from the other terminal window and observe the output on the screen.

```
print(17, type(17))
print(3.14, type(3.14))
```

Did you get the correct value types printed to the screen?

II. Assigning variables

In a program one usually assigns a value to a variable and then uses the variable in various algorithms. Type the following six additional lines in *lab.py*, save, and run the program.

```
message = "Hello, World!"
n = 17
pi = 3.14
print(message)
print(n)
print(pi)
```

There are rules for defining variables.
1. Can the variable name start with numbers?
2. Can they have mathematical operators in the name (+, -, *, /, ?)?
Try naming variables with these characters to find the answers.

III. Evaluate expressions using operators

An expression is a combination of values, variables and operators. An operator is a symbol that represents a math function. Add each of the following three groups of lines to the file *lab.py*, save, and run the program after inserting each group.

```
x = 17 + 2
print(x)

y = pi/2
print(y)

fruit = "banana"
bakedgood = " nut bread"
print(fruit + bakedgood)
```

© Patrick Fleming, Johns Hopkins University

Notice that strings must be in quotes but variable names are not in quotes (even if the variable is a string).

3. When we "add" two strings what is this operation usually called?

IV. Type conversion

It is sometimes convenient (or necessary) to convert one value type to another; from integer to float for example. Python has built-in functions that do this. You defined the variables **n** and **pi** above. Let's change their types.

```
print(n, type(n))
n = float(n)
print(n, type(n))

print(pi, type(pi))
pi = int(pi)
print(pi, type(pi))
pi = str(pi)
print(pi, type(pi))
```

What happens if you add `str(pi)` to `str(pi)`? Try it.

V. Importing function modules

The functions above (`float, int, str`) are built into the base of Python. Many other functions are supplied with Python but they are in separate modules that must be imported before the functions may be used. The higher math functions are examples of this.

```
import math
print(math.sqrt(25))
pi = math.pi
print(pi)
angle = ((2.0*pi)/360.0) * 180.0
print(angle)
print(math.cos(angle))
```

The math module also contains an accurate value for **pi**. This **pi** value is a **function** of the math module. Other functions of the math module include **sqrt** and **cos**.

4. Do the trigonometric functions in python accept angles in degrees or radians?

(Hint: In your second terminal window enter the following commands to launch an interactive python session:

```
python3
import math
help(math)    [read the screen for the answer]
              [Note: You are inside a more session here when
              viewing the help file. To exit the more session
              and get back to the python prompt enter a q.]
ctrl–d        [press simultaneously to quit python session]
```

You can find help on any python command or module with this procedure.)

© Patrick Fleming, Johns Hopkins University

5. What other accurate data value is included in the math module? Hint: Use the **help(math)** command as above and the space bar to scroll down the list of included items in this module.

A list of modules in Python and the functions they contain is available at <http://docs.python.org/3/py-modindex.html >

VI. User created functions

You can define new functions to provide an easy means to perform repetitive tasks with one command. These functions can be quite complicated (then we would more properly call them procedures) but we will start with a simple one that prints a string of dashes. Notice that the second line is indented and the third line is not. Python uses indentation to define operational parts of the code. There is no "end" statement as in many other programming languages. The tradition is to use four spaces of indentation but any number is OK as long as you are consistent throughout the program. After the function is defined it is "called" simply by typing its name.

```
def newline():
    print("------")

newline()
```

6. Is it possible to print more than one variable onto a line? Try it with comma separated variables.

VII. Function variables are local

When you assign a variable within a function it is defined only within that function. The rest of the program does not know about it. Insert the following lines in *lab.py* and run the program.

```
def printTwice(arg):
    two = arg + arg
    print(two)

printTwice('spam')
```

Now type the following additional line at the end of your file and run the program.

```
print(two)
```

What happened? In order to make the program run again let's comment out the offending line. Put a "#" at the beginning of the last line and run the program again.

```
#print(two)
```

© Patrick Fleming, Johns Hopkins University

VIII. Boolean expressions and conditional execution

A boolean expression is an expression that is either true or false. Conditional execution (**if**, **else** statements) can be controlled by the result of a Boolean expression.

```
x = 1
y = 2
if x < y:
    print("x < y")
elif x == y:
    print("x = y")
else:
    print("x > y")
```

Now assign the `x` variable to a value of `2` and run the program again.

Note the syntax for the equal operator for *comparing* two variables is different than the equal operator for *defining* a variable.

IX. Return values (Useful functions, procedures, subroutines)

The functions we have used above did useful things. But they did not return a value. This ability is the heart of a useful function: We may want the function to perform a calculation or string manipulation and return the answer. Mathematically a function maps values to other values. Type the following two groups of lines in separately and run the program after each group. Note the indentation.

```
# Group 1
def area(radius):
    return math.pi * radius**2

print("Area of circle = ",area(1.414))

# Group 2
def dist2D(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dist2 = dx**2 + dy**2
    dist = math.sqrt(dist2)
    return dist

print("Distance between points = ", dist2D(1,1,2,2))
```

(Remember that we imported the `math` module earlier).

Functions are sometimes referred to as subroutines or procedures in computer code. What is the difference between these? For our purposes, none. Traditionally functions only return a value; subroutines and procedures perform other tasks besides pure value mapping.

Note: For the homework you may want to calculate the distance between two

points in three dimensional space, e.g. between two atoms in a molecule. The code for this problem could look like the following:

```
def dist3D(x1, y1, z1, x2, y2, z2):
    dx = x2 - x1
    dy = y2 - y1
    dz = z2 – z1
    dist2 = dx**2 + dy**2 + dz**2
    dist = math.sqrt(dist2)
    return dist
```

X. Composition (Calling one function within another)

This example shows not only the power of composition but also how tricky it could be to read the source code of a program and follow the logic. We are calling subroutines (**`area`**, **`dist2D`**) that may have been created in an entirely different area of the code.

```
def area2(x1,y1,x2,y2):
    radius = dist2D(x1,y1,x2,y2)
    result = area(radius)
    return result

print(area2(1,1,2,2))
```

Good programmers like to write separate modules that perform specific tasks. This is a better method of programming than writing one giant routine that does everything at once. It is much easier to debug small routines than big ones.

XI. Iteration

One of the great benefits of computer programs is the ability to perform repetitive tasks. The **`for`** statement and the **`while`** statement are two ways to perform iteration. Note the indentation to define the **`while`** loop.

(Make sure **`fruit`** is defined somewhere above this next code. It should be a "banana" if you followed the tutorial so far.)

```
for char in fruit:
    print(char)

def countdown(n):
    while n > 0:
        print(n)
        n = n-1

countdown(10)
```

What happens if you accidently start a while loop for which the cutoff criteria is never met?

Try the following code:

```
p = 1
while p == 1:
      print(p)
```

How do you stop the program? (Press **ctrl** and **c** simultaneously).
Now delete the above three lines (**dd** while cursor is on the line)

XII. Lists

Lists are one of the most used concepts in Python. A list is an ordered set of values. Each value (or element) in the list may be addressed by its index. Each element may be an integer, float or string.

```
list1 = [1,2,3,4]
print(list1)
```

It is important to remember that for Python indexing starts with zero, not 1. Notice the numbers returned with the following two commands.

```
print(list1[0])
print(list1[3])
```

Note: Python is different than shell scripting in this regard. Remember that with awk, $1 means the first column, not the second column.

We can count backwards from the last index. 7. How many list items are printed with the following command?

```
print(list1[0:-2])
```

You can iterate through the list indices to print out each member of the list. You may want to do this for a homework assignment.

```
print(" ")
for i in range(len(list1)):
    print(list1[i])
```

where "len" means length. Note that "i" can be anything, e.g. "n" or "item" , etc. A range() is an iterable object. You will see it many times in the code we are using.

Lists can contain strings as well as numbers (strings must be in quotes; double or single)

```
list3 = ['one','two','three']
print(list3)

print(list3[1])
```

Some built-in functions exist for lists. For example, you can query for membership in the list.

```
if 'two' in list3:
    print(1)
```

```
else:
    print(0)
```

And you can perform operations on lists.

```
list4 = ['four','five']
all = list3 + list4
print(all)
```

You can use the indexing of the list to extract a portion of it.

```
print(all[0:4])
```

Indexing is also called 'slicing' and you will use this method extensively. Choosing the indices to specify a subset of the list is a little tricky. If it helps, you can think of it this way: reading the list from left to right, the first slice index specifies the first element you want, and the second slice index specifies *the first element you don't want*. Notice that the above slice returns the first four elements but not the fifth.

Lists are mutable. That is, they may be changed.

```
del all[1:4]
print(all)
```

XIII. Objects and Methods

Lists are objects within Python and, as with many objects, they have functions associated with them. These functions are called methods. A method is a function that comes along with or is inherited with the object. The general syntax to call a method is:

```
object.method(arguments)
```

For example, the **`append`** method is associated with lists and is a convenient way to extend a list:

```
print(list1)
list1.append(5)
print(list1)
```

Use the list index method to find the index of the integer 3 in `list1`. 8. What is this index?

```
print(list1.index(3))
```

XIV. Files

Of course, one of the most useful functions of a program is to read and write files on the computer disk. Below are three groups of lines that show one method to handle simple input/output tasks. The formatting of output can become quite

complicated: Here we use only one formatting symbol; the line return (**`\n`**). The **`'w'`** indicates that the file should be opened with *writing* permission and the **`'r'`** indicates that the file should be opened with *reading* permission. Enter each group of lines in *lab.py* and run the program after entering each group.

```
file = open('test.dat', 'w')
file.write('# Mydata \n')
file.write('1.0     1.0\n2.0    4.0')
file.close()
```

The `write` and `close` methods are associated with the **file** object.

Now, in your second terminal window, look at the file *test.dat* with the **less** command.

```
less test.dat
```

Here is how to read the file from disk, print it to the screen, and write it to another file.

```
file = open('test.dat', 'r')
print(file.readline())
print(file.readline())
file.close()

file1 = open('test.dat', 'r')
file2 = open('copy.dat', 'w')
while 1:
    line = file1.readline()
    if line == '':
        break
    if line[0] == '#':
        continue
    file2.write(line)

file1.close()
file2.close()
```

And finally, look at the file *copy.dat* with the **less** command.

9. What is accomplished by the conditional lines before the **`continue`** and **`break`** commands? In other words, what do the "#" and "nothing" mean as you scan through the lines of a file?