Scientific Computing: Lecture 3

- Functions
- Random Numbers
- More I/O
- Practice Exercises

CLASS NOTES

- You should be finishing Chap. 2 this week.
- HW00 due by midnight Friday into the Box folder
- You should have a functional python system now installed.
- See me if you have concerns or questions.

Function Basics

- Functions provide a way to perform some operation. Makes code more 'modular' and efficient.
- Use `def` keyword to define a function name and input parameters. ':' and indentation required to define the block of code in the function
- Almost always want function to return SOMETHING to the main program. Good programming practice to ALWAYS return something – even just a '1' of successful or '0' if it failed.

```
#Define the function
def sum3(n1,n2,n3):
        result= n1 + n2 + n3
        print 'Sum of the numbers is %2.3f' % result
        return result

#Use the function
total = sum3(10.3,20.5,5.1)
```

More on functions

- Position of arguments is important!

```
def funct(a,b,x,c=1,printFlag=True):
    result = a +b*x**c
    if printFlag:
       print 'For a = %2.2f, b=%2.2f, x=%2.2f, and c=%2.2f \n' \
             % (a,b,x,c)
    return result
```

- Can also pass an arbitrary number of parameters by adding a '*' in front of the parameter. All params get grouped in a tuple which can be looped over:

```
def sumnums(*nums):
   sum = 0
   for num in nums:
       sum += num
   return sum
#Usage
total = sumnums(1,2,3,4,5,20,45)
```

more on functions

- Functions can be called from within other functions.

```
#Look at last slide for sumnums definition
def avg(*nums):
        return sumnums(nums)/len(nums)
```

- Can also pass a function as an argument to another function! (This is pretty unique to Python)

```
def powerlaw(x,m,b): return m*x**b
scipy.curve_fit(powerlaw,xdata,ydata,p0=(1.0,1.5))
```

- Functions can return multiple items in a tuple.

```
def simplestats(*nums):
        avg = avg(nums)
        max=max(nums)
        min=min(nums)
        return avg,max,min
mystats=simplestats(10,20,8,20,24)
#OR assign the return values directly
avg1,max1,min1=simplestats(10,20,8,20,24)
```

Documenting functions

- Documentation is an important (and often overlooked) part of programming. But it's important to people reading and using your code (including you, one year later!)
- Docstrings are the standard method in Python.
- Users access this string by `help(functionName)`
- Any string right after the definition will do, but a triple quote is convenient. It maintains the formatting when returned to the user.

```
def myavg(*nums):
        '''
        Function to average a sequence of numbers.
        Usage: testavg = avg(70,80,75,99,98)
        Input:  arbitrary length series of integers or floats
        Output: a float equal to the average
        History: version 0.1, last updated Jan. 23, 2012 by JRG
        '''
        return sumnums(nums)/len(nums)
```

Practice Excercise

- What does the following function do?

```
def squareit(somenums):
    for i in range(len(somenums)):
        somenums[i]=somenums[i]**2
    return 0
```

- Now alter the function to return a NEW list and leave the original list unchanged.
  - Hint: use range() function to create a list of proper length.

Variable scope in functions

- ‘Scope’ means where a variable is defined. Typically labeled as ‘global’ (known everywhere) or ‘local’ (only known within the block). How and where each variable is defined is called the “namespace”.
- Any variable defined in the ‘main’ part of the program is global, meaning it is known inside the functions of that program.
- Variables defined inside functions are local and are not known outside – even after the function is called.

```
def funct():
        a= 1
        print a

a=2
print a # prints ‘2’
funct()#prints ‘1’ when called
print a # prints ‘2’ again – ‘a’ is unchanged globally
```

Variable scope in functions

- Lists and dictionaries are different.
  - If they are created outside the function, then changed inside the function, the changes are global.

```
def funct():
        a.sort()
        a.append(4)
        print a

a=[3,2,1]
print a # prints [3,2,1]
funct()#prints [1,2,3,4] when called
print a # prints [1,2,3,4] -> change is global
```

Exercises

- Write a function that takes in a list of numbers and returns the sum of the numbers. Call it ‘sumnums’

- Write a second function that uses ‘sumnums’ to then compute and return the average of those numbers.

- Write a third function to compute the standard deviation of those numbers
(N is total number of numbers):

- You will need to import ‘sqrt’ from the ‘math’ module.

$$\sigma = \sqrt{\frac{\sum_{i=0}^{N}(x_i - \bar{x})^2}{N}}$$

Random numbers

- Core Python has a random number module: “random”

```
import random as r
r.random() # returns random number b/t [0,1)
r.uniform(a,b) #returns a random float between a and b with
               equal prob.
r.gauss(mean,stddev) #returns random float with gaussian
       probability centered on mean and width of stddev
```

- Other interesting method is ‘shuffle’ which returns the list in a random order (like shuffling a deck of cards). Note that it actually changes the list (‘in place’).

```
mylist = [‘A’,‘K’, ‘Q’, ‘J’]
r.shuffle(mylist)  -> [‘A’,’J’,‘K’, ‘Q’]
```

More i/o in python

- Output to screen usually handled by formatted print statements.

- Other useful things to improve formatting output are
'\n' -> inserts a line break and '\t' inserts a tab

- Formatted output takes the form:

```
' Some text: %s \n a float: %2.4f \n \
integer: %i \n scientific: %2.2e' % \
('hello',3.14159, 101, 12563.667)
```

this prints out -

```
Some text: hello
a float: 3.1416
integer: 101
scientific: 1.26e+04
```

- f: float, g: float with best guess on format
i: integer, s: string, e: scientific notation

file output

- Can also write print output to a file rather than the screen.
- 'open' command creates a *file object* with methods to write lines to the file and close it. It creates a text file.
- Put in a loop to write lists of things like data to save

```
outfile=open('test.txt','w') # 'w' to write, 'r' to read
outfile.write('First line \n second line \t %1.2f' % (25.4) )
outfile.write('='*30) #prints 30 '=' characters
x=[1,2,3,4,5]
y=[1,4,9,16,25]
for xi,yi in zip(x,y):
      outfile.write(' \n %2.2f \t %2.2f' % (xi,yi))
outfile.close()
```

file input

- Similar procedure to read in data from a file

- `readlines` method creates a list of strings with each element a line in the file.

- Handy string method is "`split()`" which splits up a string into a list of strings split by the argument – None: any white space, '\t': at tabs, ',': at commas. You get the idea…

```
infile=open('data.dat','r')
lines = infile.readlines()
# returns a list of each line in file
infile.close()
x=[]
y=[]
for line in lines:
        if line[0] == '#': continue  #Comments
        x.append( float( line.split()[0]))
        y.append( float( line.split()[1]))
```