iPython Quick Tutorial & CheatSheet

Note: All of these commands are based on Python 2.x versions not 3.x and in particular are suited for ipython.

This by no means is comprehensive!! In fact there are many, many ways to do many of these tasks far more efficiently. This is mainly to get you quickly acquainted with some of the basic tasks for lab 1 and HW 1. I encourage all of you to search for new and better techniques to do similar tasks in ipython or python.

After starting ipython (ipython --pylab) try the following commands. To view output, either type out the variable name or print variable name. For example:

```
x=1
x
# or
print x

#Notice that anything following # is a comment.
```

1) Math tasks:

```
x=1
y=3
x+y
exp(x)
sin(y)
```

see https://docs.python.org/2/library/math.html for more functions (you don't need to put 'math.' in front of each function)

2) Lists and Arrays

Create an empty list:

```
stuff =[]
```

create a list of 100 elements all equal to 5:

```
stuff= [5]*100
```

create a list of words:

```
stuff = ['I','like','peanut','butter','and','jelly']
```

Access 3rd element in a list

```
stuff[2]
```

Change an element:

```
stuff[5]='chocolate'
print stuff
```

Change more than one element to the same value

```
stuff = [1]*10
```

```
stuff[3:5]=2
print stuff
```

Add an element to the end of a list/array:

```
stuff = [1]*5
stuff.append(2)
print stuff
```

Return the number of elements of a list:

```
len(stuff)
```

Arrays are multidimensional lists - they can also be just 1D too. Create a 1D array sequence using the package NumPy

```
import numpy as np
stuff=np.arange(0,10,0.2) #(start, end, increment)
```

NumPy is nice because you can utilize more powerful array functions than just a simple python array (list). Other NumPy Array Creation:

random array

```
stuff = np.random.rand(10)
```

create array of 100 elements of the same value (3):

```
stuff = np.array([3]*100)
```

change elements based on criteria:

```
stuff = np.arange(10)
stuff
    ## output array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
stuff[stuff>4] = 20
stuff
    ## output array([ 0,  1,  2,  3,  4, 20, 20, 20, 20,
20])
```

3) For Loops: iterate over a list (or array):

Tasks within a for-loop are indented. To get out of the for loop press return again. Ipython is thoughtful enough to indent for you as code is entered.

method 1:

```
stuff = ['I','like','peanut','butter','and','jelly']
for word in stuff: #We are iterating over each word
    print word
```

method 2:

```
for i in range(len(stuff)):
     print stuff[i]
#Here we are iterating over the element number in stuff
#In python, lists and arrays start at 0 and end at N-1
#where N is the number of elements. Range is a function
```

```
#that is telling python to iterate over elements 0
#through N-1. range(len(stuff)-1) would #be from 0 to N-
#2...
```

4) Plotting functions

The most basic plot is a 2d x-y plot. First create an array of x-values and then operate on it with some function. Then plot it
ex.

```
x=np.arange(-pi,pi,1/pi)
y=np.sin(x)
plot(x,y)
#If you want circles to indicate the data points try:
plot(x,y,'o')
```

try some other functions like `exp() x**3 log10()`...etc. Note you will have to close the plot window before plotting a new function otherwise it will keep plotting on the same window.

5) Output:

Saving output into a file: Saving array data to a file
From your last example, you should have an array of x's and an array of y's. Lets save this output to a column file called sine.txt

```
savetxt('sine.txt', column_stack((x,y)),fmt=('%5.4f','%5.4f'))
#Column stack puts the arrays in neat columns, fmt is a
#format statement that forces the values to be floats with 4
# decimal places
```

You can also save it by using this more inefficient way but you have more control over the output:

```
file="myoutput.txt" ##This creates a new file called
of = open(file,"w") ##myoutput.txt
#Make a for - loop and force data into a string.
#Include \n (try without and see what you get)
for i in range(len(xx)):
   of.write(str(xx[i])+"  "+str(yy[i])+"\n")
 # To just print to screen change + to ,:
   print str(xx[i])," ",str(yy[i])
```

Note! In some cases you won't be able to view the output file until you exit out of ipython.

6) Input

Now lets read the data back in. There are a couple of ways to do this. For an 'easy' simple text file with numeric columns you can use `loadtxt`. Lets call the new columns xx and yy

```
xx,yy=loadtxt('sine.txt',unpack=True)
```
now type xx and yy to see the results and compare to x and y.

You can also use `loadtxt` for floats, strings (text) and mixed types but you need to include some keywords in the command call since it assumes that it's reading floats. Do a google search on `loadtxt` for help on this. Also check out `genfromtxt`

Here is the most generic and 'clunky' way I know of reading in column data from a text file. The upshot is that this one works in all situations.

First create a sample file called myfile.txt which looks like:
```
A   45.63  231.1
B   -123.42  -96.1
C   0.12    3.01
```

Now read it in:
```
file="myfile.txt"
fl=open(file,"r") ## File is stored as variable fl
for lines in fl:
  lines=lines.strip() # removes return character at end
  col = lines.split() # splits line into separate columns
# extract the first column and call it 'name':
  name = col[0]
# extract the 2nd and 3 columns and call them x and y
  x = col[1]
  y = col[2]
```

As you will see these lists aren't saved in the environment. Lets just save the 2nd column (x) using the function append:
```
x=[]  #First declare x as empty array
fl=open(file,"r") ## File is stored as variable fl
for lines in fl:
  lines=lines.strip() # removes return character at end
  col = lines.split() # splits line into separate columns
  x.append(float(col[1]))
# Python automatically assumes that read data is a string, so
# you must force it to be a float by using float(). Sometimes
# numbers are read as integers. This is usually bad because doing
# math on an integer will round or truncate numbers. It's best to
# just explicitly declare data as floats unless of course it's a
# string or you want it to be an integer.
```

Some other commands
`%hist` will list out all commands you've used. This probably has a limit of the last 100.
`%paste` is handy if you type up all your code in a text file and then paste it into ipython. If you just use paste then the indentations aren't preserved and you get numerous errors.