Introduction to Programming for Scientists

Prof. Steven Ludtke
N420, sludtke@bcm.edu

Team Learning 7
Making a command-line plotting tool

The goal

In this team learning exercise we will write a command-line program capable of plotting 2-D text files, without using numpy.loadtxt()

Preliminaries (10 min)

- Launch Jupyter as usual.
- Instead of creating a new Notebook, go to New and select Text File under Other.
- You now have an empty text file, which will be saved in whatever folder you launched Jupyter from.
- rename the text file, by clicking on "Untitled.txt" (the default name), and call it "test1.py" instead
- Put these two lines in the file:

```
#!/usr/bin/env python
print("Hello World")
```

- Save the file (File menu)
- Open a command-line window (different on the different platforms)
- cd <the folder where Jupyter runs from> (you can search for test1.py if you don't know)
- python test1.py
- You should see Hello World printed on the screen. If you get an error, ask someone for help, or try to debug the problem yourself.
- Now try just typing "test1.py" without python first
- On windows machine, this should work properly
- On Mac/Linux, you will get an error, because the file is not marked as executable. Type:
- chmod a+x test1.py
- then try "test1.py" again

Arguments (5 min)

Now you know how to create a program you can run from the command line. Next we need to make it accept a simple command-line argument.

- Modify your program so it contains:

```
#!/usr/bin/env python
from sys import argv
print("The arguments you provided are:")
print(argv)
```

- Now, run test1.py again, but this time follow the command with some arguments (anything is fine)
- Try running "python test1.py" with some arguments. Are the results the same or different?

Matplotlib from the command-line (5 min)

- Let's start with this (remember, you are modifying test1.py, not running in a Jupyter notebook):

```
#!/usr/bin/env python
from pylab import *

x=arange(0,3*pi,0.05)
y=cos(x)
plot(x,y)
show()
```

- Save the program and run it. It should open a plot window, and exit back to the command line as soon as you close the plot.

Reading from text files (10 min)

- For this next section we will need the file that we used in the last team learning exercise (curve.txt). Move that file to the same folder as test1.py. If you don't have it, it is still available on the class website.

In the previous team learning exercise we read this file using NumPy's loadtxt() function, which does all of the hard work for you. This time I want you to use the basic file-IO methods we learned about today to read the file and plot it on the screen.

- To read a text file, we must open it, read the contents, then close it. Try this:

```
#!/usr/bin/env python
from sys import argv
from pylab import *

filein=open(argv[1],"r")
lines=filein.readlines() # reads the entire file as a list of
strings
for line in lines:
    print(line,end='')        # end='' suppresses the new line
```

- If you run this as 'test1.py' alone, you will get an error. Try it and make sure you understand the error.
- Now try running 'test1.py curve.txt'. The file is quite long, so you will get a lot of output on the screen.
- With this program, the variable "lines" contains a list of strings. Each string contains one line of the file.

Your task (30 min)

Your final task is to modify the file reading program we just created

- First: turn "lines" into an 'x' array and 'y' array (without using loadtxt()!).
  - All of the basic NumPy objects/functions are already available thanks to "from pylab import *". eg - myarray = array([1,2,3,4,5]) will create an array object with 5 integers.
- Second: Make the program display a plot of the x/y arrays you created

- when you are done, you should have a program which, when you type 'test1.py curve.txt' it will open a plot window and plot whatever data is in curve.txt.
- Send in your test1.py as your completed result for this team learning exercise.

bonus: If this was too easy for you, try making use of argparse to add a couple of additional options to the program, for example, setting the x/y axis labels.