Lecture 5: An Introduction to Python Programming

References

- Unix Shell : http://swcarpentry.github.io/shell-novice/ (http://swcarpentry.github.io/shell-novice/)
- Python : http://swcarpentry.github.io/python-novice-inflammation/ (http://swcarpentry.github.io/python-novice-inflammation/)
- PSL : https://swcarpentry.github.io/python-second-language/ (https://swcarpentry.github.io/python-second-language/)

Installers

There are a number of different ways to install Python. For this class, we will use **miniconda**, which is based on **anaconda**, a powerful Python framework.

Miniconda can be found at https://conda.io/miniconda.html (https://conda.io/miniconda.html) . In this class we will use **Python 3**. There are subtle differences between Python 2.7 and Python 3; many groups are moving towards Python 3 so we will, as well.

Installation is easy: Download the script and run it:

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
    bash Miniconda3-latest-Linux-x86_64.sh
```

Open a new terminal and then do:

```
conda install matplotlib numpy jupyter notebook scipy
```

This notebook is housed at : /n/ursa1/A288C/sgriffin

That's all!

The labwork for this class will require you to install your own version of miniconda on the lab machines and then produce some simple Python scripts.

Once installed, which one are we using?

Make sure you are pointing towards the right version using the **which** command! There are at least 4 ways you may wind up running python:

1. python: the script interpreter

```
which python #shows you the first python version in you
r install
    python
```

Python scripts normally start with

```
#! /usr/bin/env python
```

so you can use them just like another unix command on the terminal/command line. The two characters there are called a 'shebang'; they force the interpreter to use a specified binary to execute a piece of code. This could be `/bin/bash` or `/bin/python` or `/bin/tcsh` depending on what you are doing.

2. ipython: interactive python

```
ipython
```

This is the way to quickly try something out but can quickly become clunky. Jupyter provides a web interface to iPython, which makes developing code and plotting very convenient.

3. graphical user interface

Here there are quite a few options, most of them have *introspection-based code completion* and *integrated debugger*

```
* PyCharm
* KDevelop
* PyDev (eclipse)
* Spyder (anaconda)
  You can install this:   *conda install spyder*
```

We will not be using these in this class, but it can be a great way for development.

4. jupyter: web interface

```
which jupyter #just making sure we're running the corre
ct binary
    jupyter notebook
```

Now watch your default browser opening a new tab. The rest of this lecture will be taking place in the notebooks.

Apart from the Software Carpentry (http://software-carpentry.org) lessons, a comprehensive python tutorial that we often will come back to is on https://www.tutorialspoint.com/python/index.htm (https://www.tutorialspoint.com/python/index.htm)

Introduction

Notebooks

To run your own version of Jupyter, you'll need to install the correct packages (and a few others we'll use):

```
conda install matplotlib numpy jupyter notebook scipy
```

What you are reading here is text in a cell in a notebook. A notebook can have many cells. This is a text (really: markdown) cell. It can display text, math, include figures etc. The next cell will contain some python code that you can execute in the browser. You can see this from the **In [ ]:** prompt.

Markdown has a special syntax for doing things, and you can write $\LaTeX$ code within it, which is handy.

But before that, let's show off with some math, $c^2 = a^2 + b^2$ and also

$$\frac{GM(r)}{r^2} = \frac{v^2}{r}$$

which should look familiar to most of you.

$$F = m \times a$$

1st leave heading: Markdown formatting

Making a bullet list

- a list
- uses stars
- for bullets

2nd level heading

Or an enumerated list, but it looks weird in raw format. Double click on this cell!

1. first
2. second
3. third
4. reddit

3rd level heading

And this link (http://www.astro.umd.edu/~teuben/ASTR288P/) is created with `[...](...)`, where you should note the use of the back-ticks.

Python Variables

Python, as opposed to other languages, has dynamic types.

In (e.g.) C, you need to declare variable types that in general cannot be mixed.

- int variable_1 = 1;
- float variable_2 = 38.44
- char* variable_3 = "EnzoMatrix"

In Python, we can now do something like:

In [19]:

```
a = 1.  #This is a float
b = 2   #This is an integer
c = "344444" #This is a string

print(a, b, c)

# and then do this:

c = 3.1
print(a,b,c)

c = "a+b"
print(a,b,c)

#in C this would cause an error!
```

```
1.0 2 344444
1.0 2 3.1
1.0 2 a+b
```

In order to execute this python code, click on the cell to activate it, then using SHIFT-ENTER. Note that as soon as the cell has executed, **In[]** (a python list) has obtained a sequence number; this tells you in what order a cell was executed.

In [16]:

```
#  another way of output
print("a=",a,"b=",b,"c=",c)

# yet another way of output
# This method automatically casts a and b to strings, but chang
ing the last '%s' to something incompatible would not work.
print("a= %d b= %d c= %d" % (a,b,c))

# and yet another way of output
print("a= %s b= %s c= %s" % (str(a),str(b),str(c)))
```

```
a= 1.0 b= 2 c= SomeStringWithManyCharacters

-------------------------------------------------
------------------------
TypeError                                 Traceback
 (most recent call last)
<ipython-input-16-82dbf5bcd91a> in <module>()
      4 # yet another way of output
      5 # This method automatically casts a and b t
o strings, but changing the last '%s' to something
 incompatible would not work.
----> 6 print("a= %d b= %d c= %d" % (a,b,c))
      7
      8 # and yet another way of output

TypeError: %d format: a number is required, not str
```

In [14]:

```
my_float = 3.455
print(str(my_float))
```

```
3.455
```

My preferred method:

In [9]:

```
a = 9.3188
b = 543
c = "Twelve"

print("{0:g} {1:d} {2:s}".format(a,b,c))
#notice the difference:
print("Flux={1:g} Space={0:d} Friday={2:s}".format(b,a,c))

# More on formatting:

print("{0:8.3g}".format(a))
print("{0:8d}".format(b))
print("{0:8s}".format(c))

print("\n------\n")
print("{0:08.3g}".format(a))
print("{0:08d}".format(b))
print("{0:8s}".format(c))

# find out the types of variables
print(type(a),type(b),type(c))
```

```
9.3188 543 Twelve
Flux=9.3188 Space=543 Friday=Twelve
    9.32
     543
Twelve

------

00009.32
00000543
Twelve
<class 'float'> <class 'int'> <class 'str'>
```

A little more on python data structures

Next to dynamic typing, one of the powers of python are a number of built-in data structures (lists, dictionaries, tuples and sets) that together with their built-in operators make for a very flexible scripting language. Here is a quick summary how you would initialize them in python:

data type	**assignment example**	**len(a)**
list	a = [1,2,3]	3
tuple	a = (1,2,3)	3
dictionary	a = {'key_1':1 ,'key_2':2 , 'key_3':3}	3
set	a = {1,2,3,2,1}	3

In [20]:

```
 a = {'key_1':1 ,'key_2':2 , 'key_3':3}
```

In [21]:

```
a['key_1']
```

Out[21]:

```
1
```

Lists in Python

In [23]:

```
# we saw the assignment:
a = [1, 2, 3]
print(len(a))

# a zero-length list is ok too
a = []
print(len(a))

# but if you need a lot more, typing becomes tedious, python ha
s a shortcut
a = list(range(0,10,1))
print(a)

# notice something odd with the last number?
```

```
3
0
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [ ]:

```

```

In [27]:

```
# slicing and copying
b=a[3:7]
print(a)
print(b)



b=a[3:7:2]
#b[0]=0
print(b)
print(a)

a.reverse()
print(a)
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[6, 5, 4, 3]
[6, 4]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Two short notes on python lists:

- The data types inside of a list don't have to be all the same.

```
a = [1.0, 2, "3", [1.0, 2, "3"], range(10,20,
2)]
```

- Each python object can have a number of member function. In a notebook you can find out about these:
  - **google**, stackoverflow, e.g. the online manuals https://docs.python.org/2/tutorial/datastructures.html#more-on-lists (https://docs.python.org/2/tutorial/datastructures.html#more-on-lists)
  - the inline **`dir()`** method in python

    ```
    dir(a)
    ```

    not very informative, but it does remind you of the names
  - python **introspection**

```
import inspect
inspect.getmembers(a)
```

- Use the ipython or notebook **TAB completion**: For our object `a` typing `a.<TAB>` should show a list of possible completions, move your cursor to the desired one

In [ ]:

```
print(a)
a.append("sean")
print(a)
```

Slicing Lists (and later arrays)

In [28]:

```
a = [1,2,3]
print("assigning a slice")
b=a[:]
print("a=",a)
print("b=",b)
print("---")
b[0] = 0
print("a=",a)
print("b=",b)
print("\n")
```

```
assigning a slice
a= [1, 2, 3]
b= [1, 2, 3]
---
a= [1, 2, 3]
b= [0, 2, 3]
```

In [29]:

```
a = [1,2,3]
print("simple assignment")
print("a=",a)
print("b=",b)

b=a #b is NOT a copy of a! It is simply another variable pointi
ng to the same place in memory.
b[0] = 0
print("a=",a)
print("b=",b)
print("---")
#change an element in 'a'
a[0] = 5
print("a=",a)
print("b=",b)
print("---")
#change an element in 'b'
b[2] = -9
print("a=",a)
print("b=",b)
```

```
simple assignment
a= [1, 2, 3]
b= [0, 2, 3]
a= [0, 2, 3]
b= [0, 2, 3]
---
a= [5, 2, 3]
b= [5, 2, 3]
---
a= [5, 2, -9]
b= [5, 2, -9]
```

In [ ]:

```
#but if we make a copy of a:
b = list(a)
b[2] = '17'
print("a=",a)
print("b=",b)
```

In [30]:

```
a = list(range(10,102,5))
print(a)

print("Last entry in a: {0:d}".format(a[-1]))
print("Second-last entry in the variable a: {0:d}".format(a[-2
]))
print()
#This works for strings, too:
c = "This is a long string"

print("Last entry in c: {0:s}".format(c[-1]))
print("Second-last entry in c: {0:s}".format(c[-2]))
```

```
[10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 7
0, 75, 80, 85, 90, 95, 100]
Last entry in a: 100
Second-last entry in the variable a: 95

Last entry in c: g
Second-last entry in c: n
```

In [33]:

```
a[-1]
```

Out[33]:

```
100
```

Python Control Flow

1. if/then/else
2. for-loop/else
3. while-loop/else
4. functions

1. if/then/else

Note there is no "`else if`" or need to indent this, python uses "`elif`". Again, note the indentation controls the flow.

In [44]:

```
a = 1.e-100
if a == 0.0: #How small is zero in programming, really?
    print('zero')
elif a > 10.0 or a < -10:
    print("a is too big")
    if a < 0:
        print("negative")
        if a < 100:
            #do something
            print()


    else:
        print("a is within the bounds.")

print("this is an extra line")
```

```
this is an extra line
```

In [47]:

```
float_var = 4.
int_var = 3

print(1./float_var)
print(1/int_var)
print(float_var/int_var)
```

```
0.25
0.3333333333333333
1.3333333333333333
```

Things to try :

- seting a=0.001 0.00001 etc. or try 1e-10 1e-50 1e-100 (when is it really zero in python?)
- a = 100
- a = -5

2. for loops

The for-loop in python runs over an iterator, for example a python list is the most common one to use.

In python one is also allowed to add an **else** clause, often overlooked!

In [49]:

```
my_list = [1,3,-1,10,100,0]

for i in my_list:
    print(i)
    if i <= 10:
        print("We're good!")

    if i>=3:
        print("Break!!")
        break

print("Out of the loop!")
```

```
1
We're good!
3
We're good!
Break!!
Out of the loop!
```

In [ ]:

```
for i in [1,3,-1,10,100,0]:
    if i<0:  continue
    if i>10: print("break"); break
    if i<3:  pass; print("pass")
    print(i)
else:
    print("This gets executed only if there is no break")
```

3. while loops

The python while loop needs a termination statement. As the for-loop, it also has the **else** clause. Note that any **for loop** can be re-written as a **while loop**.

In [54]:

```
a = 0
total = 0
while a<5:
    print(a, total)
    total += a
    a += 1
    if total>100:
        break
else:
    print("final sum",total)


total = 0
for a in [0, 1, 2, 3, 4]:
    total += a
    if total > 100:
        break
else:
    print("final sum", total)
```

```
0 0
1 0
2 1
3 3
4 6
final sum 10
final sum 10
```

4. functions

Functions are the classic analog of functions in languages like Fortran and C. python of course is object oriented, and so it has a `class` as well, much like C++ and java.

In [55]:

```
import math #We import this library to gain access to its funct
ionality.

def mysqrt(x):
    # this is my sqrt function (comment vs. docstring)
    '''
    This is the docstring for mysqrt!

    This function returns the square root of the argument, unle
ss x < 0.
    If x < 0, then a negative value is presented.
    '''


    if x < 0:
        return -math.sqrt(-x)
    else:
        return math.sqrt(x)



for x2 in [-4.0, 0.0, 4.0]:
    #print(x2)
    print(mysqrt(x2))
```

```
    #print(x2)

print(x2)
```

```
-2.0
0.0
2.0
4.0
```

In [56]:

```
help(mysqrt)

#This works for other functions with docstrings:
help(math.sqrt)
```

```
Help on function mysqrt in module __main__:

mysqrt(x)
    This is the docstring for mysqrt!

    This function returns the square root of the ar
gument, unless x < 0.
    If x < 0, then a negative value is presented.

Help on built-in function sqrt in module math:

sqrt(x, /)
    Return the square root of x.
```

In [67]:

```
a1 = 1.099
def globality_test(x):
    global a1
    print(a1+x)
    a1 = a1 + x
    # y = a1 - x
globality_test(2.55)

a1
```

```
3.649
```

In [72]:

```
def test():
    return 3, 4, "some character array"
```

In [74]:

```
my_tuple = test()
print(my_tuple)

var1, var2, var3 = test()

print(var1, var2, var3)
```

```
(3, 4, 'some character array')
3 4 some character array
```

In [75]:

```
## Adding lists and appending to them are **not** the same!

a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print(c)

print(a)
print(b)

a.append(b) #Doesn't do the same thing as above!
print(a)
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3]
[4, 5, 6]
[1, 2, 3, [4, 5, 6]]
```

In [76]:

```
import numpy as np #numpy is a very powerful Python module ofte
n used for handling arrays.

# Arrays and lists are NOT the same but do have some overlap in
 functionality.

my_array = np.array(4)
print(my_array)

# Arrays have a fixed size that is determined when they are ini
tialized and a corresponding 'type'.

print(np.array([4, 7, 7.8, 12], dtype=float))
print(np.array([4, 7, 7.8, 12], dtype=int))
print(np.array([4, 7, 7.8, 12], dtype=str))
```

```
4
[ 4.   7.   7.8 12. ]
[ 4  7  7 12]
['4' '7' '7.8' '12']
```

In [80]:

```
a=np.arange(10)
b=np.arange(11,21)

print(a)
print(b)


## With arrays we can easily do complicated math very easily.
c = a + b
d = 3*a*a + b + 2.0
print(c)
print(d)
```

```
[0 1 2 3 4 5 6 7 8 9]
[11 12 13 14 15 16 17 18 19 20]
[11 13 15 17 19 21 23 25 27 29]
[ 13.  17.  27.  43.  65.  93. 127. 167. 213. 265.]
```

In [ ]:

```
a = np.array([2, 5, 7])
b = np.array([2, 4, 7])
print(a,b)

print(3 *a)
```

In [ ]:

```
# Arrays can be multidimensional; here we will make a new array
 and reshape it into We can

c = np.arange(10)
c2=c.reshape(5,2)
c3=np.array(c.reshape(5,2)) #This is a **copy** so modification
s to 'c' and 'c2' will NOT affect c3.

print("c\n", c)
print("c2\n", c2)
print("c3\n", c3)
```

In [ ]:

```
c[0] = -999
c2[1,0] = -888 #[row, column] -- easy to confuse the order espe
cially for multidimensional arrays.
print("c\n", c)
print("c2\n", c2)
print("c3\n", c3)
```

In [ ]:

```
#We can slice out a specified column or row if we so choose.

d2=c.reshape(5,2)[:,0]
print(d2)
d2=c.reshape(5,2)[:,1]
print(d2)
```

In [ ]:

```
d2[0]=-1217

print(c)
```

Parsing data files

Again, like all things in programming, there are a large number of ways to do things.

numpy.genfromtxt is handy for reading in data but you can run into issues if the formatting is strange. Also, it loads the entire file into memory at once, which is impractical for very large data sets, but it can automatically handle (e.g.) comments in the data file and strip them out.

You can also **open** a file and read it line-by-line. Again, multiple ways to do this. The most Pythonic way we will indicate below.

The **pandas** library is handy for reading in large tables of data.

In [ ]:

```
#I'm going to generate a demo file on-the-fly:

f = open("demofile.txt", "w") #a == append, w=overwrite, x=crea
tes file, returns an error if one exists.
f.write("# This is the first line in the field. We will use a
 '#' to indicate comments.\n")
f.write("# We can imagine the column headers are kept as commen
ts.\n")
f.write("# Energy, Flux, FluxError\n")
f.write("# TeV, ph/m^2/s, ph/m^2/s\n")
for i in range(15):
    for j in range(3):
        val = np.random.normal(loc=4, scale=2.5)
        f.write("{0:.3f} ".format(val))
    f.write("\n")#end of line

f.close()

#Look at the file in class, add comments, etc.
```

In [ ]:

```
with open("demofile.txt", "r") as f:
    for line in f:
        # Do something with 'line'
        print(line) #there are extra newlines here (print autom
atically adds one).
```

In [ ]:

```
with open("demofile.txt", "r") as f:
    for line in f:
        # Do something with 'line'
        print(line.strip("\n"))
```

In [ ]:

```
with open("demofile.txt", "r") as f:
    for line in f:
        # Do something with 'line'
        print(line.strip("\n"))
        #We can select out comments:
        if '#' in line: #We could have chosen any character her
e and it applies to strings in general.
            print("--> This is a comment line!")

        if '#' not in line:
            print("--> This is not a comment line!")

        #Note: We could also use an 'if-else statement here if
 we wanted to.'
```

In [ ]:

```
#We can separate the lines into a list if we know the delimeter
s between values:

with open("demofile.txt", "r") as f:
    for line in f:
        # Do something with 'line'

        if '#' not in line:
            line = line.strip("\n") #strip out the newline
            print(line.split(" ")) #We see a blank character in
 the last column!
            split_line = line.split(" ")
            #We are splitting the line based on the ' ' charact
er, but this can be any character (e.g. ',' or '|' )

            #print only the second entry in every line:
            print(split_line[1])
            #They're read into Python as strings, so if we want
ed to do math, we'd have to convert them to floats, first:
            print(type(split_line[1]))
            val = split_line[1]

            if float(val) < 6:
```

```
            print("The value is SMALL!: ", val)
        else:
            print("The value is LARGE: ", val)


    print("---------")
```

In [ ]:

```
#We can also do things easily using np.genfromtxt:

data_array = np.genfromtxt("demofile.txt", comments='#', delimi
ter=' ')
print(data_array)
print(data_array.shape)
#But this fails if the data don't have a constant shape (e.g. o
ne line has an additional column).

#We can select entire columns easily:

print(data_array[:,0:2]) #first two columns
print(data_array[:,1::]) #last two columns
print(data_array[:,1]) #last two columns
```

For more references, check out my course notes for ASTR288P:

https://github.com/SeanCGriffin/astr288p_student (https://github.com/SeanCGriffin/astr288p_student)

Specifically lectures

In [ ]: