NPRs Python Notes

Nicholas P. Ross

November 8, 2016

Abstract

The is my (NPR's) set of Python notes. Things started as wanting to just be an "IDL to Python CheatSheet", and have naturally and organically snowballed from there. Suffice to say, when this document reached 47 pages long (and I started to take notes on how to do pytests ;-), this was no long a Cheat Sheet, and became something else; a general Python resource.

You will be able to find the latest version of these notes and indeed the .tex file at:
`https://github.com/d80b2t/Research_Notes`.

Contents

1 The REAL Basics

1.1 Versions

$ python3

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more
    information.
>>> import numpy
>>> print (numpy.__version__)
1.11.1

>>> import astropy
>>> print (astropy.__version__)
1.2.1

>>> import sys
>>> print (sys.version)
3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
```

1.2 Style Guide for Python Code

PEP 8 – Style Guide for Python Code.
Use 4 spaces per indentation level.
Python 3 disallows mixing the use of tabs and spaces for indentation.
Code in the core Python distribution should always use UTF-8.
Imports should usually be on separate lines.
Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.
Avoid extraneous whitespace in the following situations: Immediately inside parentheses, brackets or braces; Immediately before a comma, semicolon, or colon; Immediately before the open parenthesis that starts the argument list of a function call; *More than one space around an assignment (or other) operator to align it with another.*

1.3 Notebook

Click on the NBviewer...
Then you can see the e.g. html of the notebook.
But to change/execute it, then all you have to do is click the download button...
Then put it on gitHub/Dropbox etc...

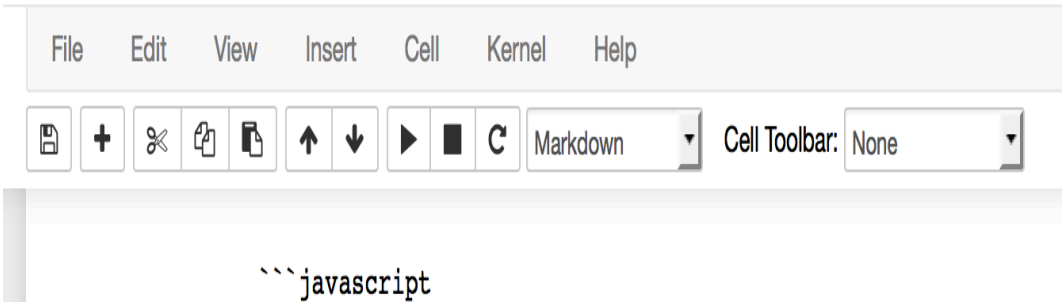

Figure 1: Clicking on the Cell Toolbar "Code", "Markdown" etc. will power what happens in the Clells!!!

(I need to learn about "Tmox" and "SCreen" Terminal emulators...)

Run a code cell using Shift-Enter or
Alt-Enter runs the current cell and inserts a new one below. Ctrl-Enter run the current cell and enters command mode.
Google: "ipython beyond plain python"

http://nbviewer.ipython.org/github/fperez/cit2013/blob/master/06-IPython%20-%20beyond%20plain%20Python.ipynb

iPython NB power = power of python + power of the command line with "!" + "%" and "%%" "magics"...

http://nbviewer.ipython.org/github/ipython/ipython/blob/1.x/examples/notebooks/Part%204%20-%20Markdown%20Cells.ipynb

https://github.com/profjsb/python-bootcamp

2 The General Basics

HEAVILY BORROWED/COPIED FROM: Python Data Science Handbook by Jake VanderPlas[1].

Comments are marked by

End-of-line Terminates a Statement

Semicolon can Optionally Termnate a Statement

Indentation: Whitespace Matters!

Indent code blocks by four spaces.

```
    for i in range(100):
        # indentation indicates code block
        total += i
```

Whitespace Within Lines Does Not Matter

Python Variables are Pointers

So when you write `x=4`, you are essentially defining a *pointer* named `x` which points to some memory bucket containing the value 4. Note one consequence of this: because Python variables just point to various objects, there is no need to "declare" the variable, or even require the variable to always point to information of the same type. This is the sense in which people say Python is *dynamically-typed*: variable names can point to objects of any type.

```
>>> x=[1,2,3]
>>> y = x
>>> print(y)
[1, 2, 3]
>>> x.append(4)  # add 4 to the end of x
>>> print(y)         # y is modified as well!
[1, 2, 3, 4]
>>> x = 'something else'
>>> print(y)       # y is unchanged.
[1, 2, 3, 4]
>>>
```

Everything is an Object

Python is an object-oriented programming language, and in Python everything is an object. Python is NOT a type-free language. Python has types;

[1] Who is awesome, so stop being stingy and go buy his book!!! :-)

however, the types are linked not to the variable names but **to the objects themselves**. When we say that everything in Python is an object, we really mean that everything is an object: even the attributes and methods of objects are themselves objects with their own type information:

```
>>> x=4.0
>>> x
4.0
>>> x.is_integer()
True
>>> type(x.is_integer)
<class 'builtin_function_or_method'>
>>>
```

Operators

Identity Operators

```
>>> a = [1,2,3]
>>> b = [1,2,3]
>>> a == b
True
>>> a is b
False
>>> a is not b
True
>>>
```

Conditional Statements: if-elif-else:

```
x=-15
if  x==0:
    print(x, "is zero")
elifx>0:
    print(x, "is positive")
elifx<0:
    print(x, "is negative")
else:
    print(x, "is unlike anything I've ever seen...")

-15 is negative
```

for loops

```
>>> for i in range(10):
...         print(i, end=' ')
0 1 2 3 4 5 6 7 8 9 >>>
>>>
>>>for i in range(10):
...       print(i, "\n")
...
0


1


2
.
.
.
9
```

3 iPython

https://ipython.org/install.html

```
$ conda update ipython
```

```
$ sudo pip3 install ipython[all]
```
Then `$ ipython3 notebook`

3.1 iPython from Fernando Perez

Try: tmpnb.org **VERY USEFUL**
http://www.pythonforbeginners.com/basics/ipython-a-short-introduction

4 Data Types

`int`, `float`, `complex`, `bool`, `str` and `NoneType` are simple/scaler types.
`list`, `tuple`, `dict` and `set` are Data Structures.

```
>>> n = 123             # int, integers
>>> f = 123.             # floats, floating-point: i.e. real numbers
>>> L = [1,2,3]
>>> a = (1,2,3)
>>> D = {1,2,3}
>>> x = {'1': '2', '3': 45}
>>> s = '1,2,3'

>>> b = True             # boolean: True/False values
>>> nt = None          # special object indicating nulls

>>> type(n)
<class 'int'>
>>> type(f)
<class 'float'>
>>> type(L)
<class 'list'>
>>> type(a)
<class 'tuple'>
>>> type(D)
<class 'set'>
>>> type(x)
<class 'dict'>
>>> type(s)
<class 'str'>

>>> type(b)
<class 'bool'>
>>> type(nt)
<class 'NoneType'>
```

4.1 Lists

4.2 tuple

4.3 Dictionaries

```
>>> x = {'hello': 'Zed', 'name': 45}
>>> x['hello']
'Zed'
>>>
```

4.4 set

4.5 Pandas and DataFrames

```
import numpy
import matplotlib.pyplot as plt
import re
import pandas as pd

# The data directory PATH
data_dir = "/cos_pc19a_npr/data/WISE/W4/"

# The data directory FILENAME
data_file="WISE_W4_W4SNRge3_W4MPROlt4.0_nohdr.tbl"

data_in = data_dir+data_file

cols = ['designation', 'ra', 'dec', 'sigra', 'sigdec',
    'w1mpro', 'w1sigmpro', 'w1snr', 'w2mpro', 'w2sigmpro',
    'w2snr', 'w3mpro', 'w3sigmpro', 'w3snr', 'w4mpro',
    'w4sigmpro', 'w4snr', 'w4rchi2']

df = pd.read_table(data_in)

##
print(list(df.columns.values))

##
print(type(df))
```

```
In [5]: type(df)
Out[5]: pandas.core.frame.DataFrame
```

5 What the outputs of type() mean...

Okay, so I'm getting a little frustrated when I do a `type()` commad to see what the output is. So, I'm going to collect a few things here and see if I can decipher the various responses.

```
In [6]: hdulist = fits.open('tractor-0298p005.fits')
In [7]: data = fits.getdata('tractor-0298p005.fits',1)
In [8]: T = Table(data)

In [9]: type(hdulist)
Out[9]: astropy.io.fits.hdu.hdulist.HDUList

In [10]: type(data)
Out[10]: astropy.io.fits.fitsrec.FITS_rec

In [11]: type(T)
Out[11]: astropy.table.table.Table

In [12]: type(T['decam_apflux'])
Out[12]: astropy.table.column.Column

In [13]: i = np.argmin(np.hypot(T['ra'] - 29.9901, T['dec'] -
    0.5530))

In [14]: type(i)
Out[14]: numpy.int64

In [15]: ii = 64

In [16]: type(ii)
Out[16]: int
```

6 String Manipulation and Regular Expressions

7 A general code example ;-)

```
"""
Outline:

You have a certain amount of credit to spend at a book store. You
    want to buy two books and you want to spend all of your store
    credit. However, you have to carry the books a far distance so
    you want to buy the lightest pair of books possible.
Each book available in the bookstore is represented as a tuple of
    their price and weight. You are given a list of all books in
    the bookstore as follows:
[(price0, weight0), (price1, weight1), etc, (priceN, weightN)]
Print the indices of the two books you should buy and their
    combined weight.
"""


credit = 18
books = [(17, 5), (3, 55), (5, 12), (14, 9), (16, 1), (9, 5),
         (5, 6), (18, 13), (19, 7), (1, 20), (4, 12), (11, 1),
         (8, 6), (8, 18), (3, 4), (13, 7), (17, 22), (20, 7)]


# Point 1: Strong condition: PriceBookA + PriceBookB = 18.

# Want to take the LIST of books, it's not a long list, so happy to
    loop over -- indeed happy to loop over twice if needs be.

# Then generate a new list, goodPrice, of the pairs of books that
    have PriceBookA + PriceBookB = 18
goodPrice=[]
largeWeight = 100000.

for i in range(len(books)):
    PriceBookA = books[i][0]
    for j in range(len(books)):
        PriceBookB = books[j][0]

        if ((PriceBookA + PriceBookB) == 18) and (i != j):
            # goodPrice becomes the sum of the weights
            # if the sum of the weights of the books is less than
                largeWeight then (i) keep that sum and (ii) keep the
                indicies and then (iiI) set largeWeight to the new
                min weight.

            sumWeights = books[i][1]+books[j][1]
            if (sumWeights < largeWeight):
                largeWeight = sumWeights
```

```
                goodIndexA = i
                goodIndexB = j

#           goodPrice.append(books[i][1]+ )
print(largeWeight)
print(goodIndexA)
print(goodIndexB)


# Figure out which (limited) combinations of books satifisy this
    price condition

# Sort that set by sum of weights (WeightBookA + WeightBookB). Pick
    the minimum there.
```

8 Britton's Classes :-)

8.1 "If lost in the desert..."

>>> dir(thing)
>>> dir(thing)

8.2 Lists

```
>>> super_list = [0, [3,4,5], "Hello World!", range(5)]
>>> print super_list

[0, [3, 4, 5], 'Hello World!', [0, 1, 2, 3, 4]]
>>> print super_list[1]
[3, 4, 5]
>>> print super_list[-1]
[0, 1, 2, 3, 4]
>>> print super_list[1[0]]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not subscriptable
>>> print super_list[1][0]
  3

>>> c = range(10)
>>> print c
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> c.append(range(3))
>>> print c
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, [0, 1, 2]]
>>> c.extend(range(3))
>>> print c
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, [0, 1, 2], 0, 1, 2]
>>> del c[4]
>>> print c
[0, 1, 2, 3, 5, 6, 7, 8, 9, [0, 1, 2], 0, 1, 2]

>>> z = [42]*5
>>> [42, 42, 42, 42, 42]

>>> print super_list
[0, [3, 4, 5], 'Hello World!', [0, 1, 2, 3, 4]]
>>> print len(super_list)
4
>>> print len(super_list[-1])
5
```

8.3 Dictionaries and Maps

From: http://learnpythonthehardway.org/book/ex39.html: You are now going to learn about the Dictionary data structure in Python. A Dictionary (or "dict") is a way to store data just like a list, but instead of using only numbers to get the data, you can use almost anything. This lets you treat a dict like it's a database for storing and organizing data.

From: https://docs.python.org/3/tutorial/datastructures.html#dictionaries: Another useful data type built into Python is the dictionary (see Mapping Types dict). Dictionaries are sometimes found in other languages as associative memories or associative arrays. Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key. You cant use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().

It is best to think of a dictionary as an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: . Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with del. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.

```
# http://www.tutorialspoint.com/python/python_dictionary.htm
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print("dict['Name']: ", dict['Name'])
print("dict['Age']: ", dict['Age'])
print("dict['Alice']: ", dict['Alice]')
```

From: http://openbookproject.net/thinkcs/python/english3e/dictionaries.html:

Hashing. The order of the pairs may not be what was expected. Python uses complex algorithms, designed for very fast access, to determine where the key:value pairs are stored in a dictionary. For our purposes we can think of this ordering as unpredictable. You also might wonder why we use dictionaries at all when the same concept of mapping a key to a value could be implemented using a list of tuples:

```
>>> {"apples": 430, "bananas": 312, "oranges": 525, "pears": 217}
{'pears': 217, 'apples': 430, 'oranges': 525, 'bananas': 312}
>>> [('apples', 430), ('bananas', 312), ('oranges', 525), ('pears',
```

```
    217)]
[('apples', 430), ('bananas', 312), ('oranges', 525), ('pears',
    217)]
```

The reason is dictionaries are very fast, implemented using a technique called hashing, which allows us to access a value very quickly. By contrast, the list of tuples implementation is slow. If we wanted to find a value associated with a key, we would have to iterate over every tuple, checking the 0th element. What if the key wasnt even in the list? We would have to get to the end of it to find out.

9 Packages

9.1 Install packages

To install:
`python3 -m pip install somepackage`

9.2 Key Packages

`astropy`
`healpy`
`ipython`
`matplotib`
`nose`
`numpy`
`pandas`
`reproject`
`scipy`
`sympy`
`pyFITS`
`yt`

9.3 PyPI - the Python Package Index

https://pypi.python.org/pypi
The Python Package Index is a repository of software for the Python programming language. There are currently 85904 packages here[2].

[2]As of Fri Aug 5 13:37:24 PDT 2016.

10 matplotib

From http://matplotlib.org/

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB* or Mathematica), web application servers, and six graphical user interface toolkits.

matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code. For a sampling, see the screenshots, thumbnail gallery, and examples directory

For simple plotting the pyplot interface provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

From http://stackoverflow.com/questions/12987624/confusion-between-numpy-scipy-matplotlib-and-pylab:

- pylab is part of matplotlib (in matplotlib.pylab) and tries to give you a MatLab like environment. matplotlib has a number of dependencies, among them numpy which it imports under the common alias np. scipy is not a dependency of matplotlib.

- If you run ipython –pylab an automatic import will put all symbols from matplotlib.pylab into global scope. Like you wrote numpy gets imported under the npalias. Symbols from matplotlib are available under the mpl alias.

```
> ipython --pylab
```

```
import matplotlib.pyplot as plt
```

11 Basemap

From http://basemaptutorial.readthedocs.io/en/latest/.

Basemap is a great tool for creating maps using python in a simple way. Its a matplotlib extension, so it has got all its features to create data visualizations, and adds the geographical projections and some datasets to be able to plot coast lines, countries, and so on directly from the library.

Basemap has got `some documentation`, but some things are a bit more difficult to find. I started this documentation to extend a little the original documentation and examples, but it grew a little, and now covers many of the basemap possibilities.

Examples: http://matplotlib.org/basemap/users/examples.htm

11.1 Setting up the map...

http://matplotlib.org/basemap/users/mapsetup.html

12 Class vs. an Instance

Difference between a class and an instance is an Object Oriented (OO) concept.

Python and Ruby both recommend `UpperCamelCase` for class names, `CAPITALIZED_WITH_UNDERSCORES` for constants, and `lowercase_separated_by_underscores` for other names.

And `snake_case` for variable names, function names, and method names.

Generally speaking, instance variables are for data unique to each instance and class variables are for attributes and methods shared by all instances of the class:

Definitions: from http://www.tutorialspoint.com/python/python_classes_objects.htm

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.
- Inheritance: The transfer of the characteristics of a class to other classes that are derived from it.

Maybe see also http://stackoverflow.com/questions/114214/class-method-differences-in-python-bound-unbound-and-static.

12.1 Abstract Classes

https://www.hackerrank.com/challenges/30-abstract-classes/tutorial

Case Study: Abstraction: This is is an essential feature of object-oriented programming. In essense, it's the separation between what a class does and how it's accomplished.

One real world example of this concept is a snack machine, where you give the machine money, make a selection, and the machine dispenses the snack. The only thing that matters is what the machine does (i.e.: dispenses the selected snack); you can easily buy a snack from any number of snack machines without knowing how the machine's internals are designed (i.e.: the implementation details).

Abstract Class This type of class can have abstract methods as well as defined methods, but it cannot be instantiated (meaning you cannot create a new instance of it). To use an abstract class, you must create and instantiate a subclass that extends the abstract class. Any abstract methods declared in an abstract class must be implemented by its subclasses (unless the subclass is also abstract).

abc – Abstract Base Classes Source code: Lib/abc.py

This module provides the infrastructure for defining abstract base classes (ABCs) in Python, as outlined in PEP 3119; see the PEP for why this was added to Python. (See also PEP 3141 and the numbers module regarding a type hierarchy for numbers based on ABCs.)

The collections module has some concrete classes that derive from ABCs; these can, of course, be further derived. In addition the collections.abc submodule has some ABCs that can be used to test whether a class or instance provides a particular interface, for example, is it hashable or a mapping.

13 Functions

N.B. Straight from: http://www.python-course.eu/python3_functions.php.

The concept of a function is one of the most important ones in mathematics. A common usage of functions in computer languages is to implement mathematical functions. Such a function is computing one or more results, which are entirely determined by the parameters passed to it.

In the most general sense, a function is a structuring element in programming languages to group a set of statements so they can be utilized more than once in a program. The only way to accomplish this without functions would be to reuse code by copying it and adapt it to its different context. Using functions usually enhances the comprehensibility and quality of the program. It also lowers the cost for development and maintenance of the software.

Functions are known under various names in programming languages, e.g. as subroutines, routines, procedures, methods, or subprograms.

A function in Python is defined by a def statement. The general syntax looks like this:

```
def function-name(Parameter list):
    statements, i.e. the function body
```

The parameter list consists of none or more parameters. Parameters are called arguments, if the function is called. The function body consists of indented statements. The function body gets executed every time the function is called.

Parameter can be mandatory or optional. The optional parameters (zero or more) must follow the mandatory parameters.

Function bodies can contain one or more return statement. They can be situated anywhere in the function body. A return statement ends the execution of the function call and "returns" the result, i.e. the value of the expression following the return keyword, to the caller. If the return statement is without an expression, the special value None is returned. If there is no return statement in the function code, the function ends, when the control flow reaches the end of the function body and the value value will be returned. Example:

```
def fahrenheit(T_in_celsius):
    """ returns the temperature in degrees Fahrenheit """
    return (T_in_celsius * 9 / 5) + 32

for t in (22.6, 25.8, 27.3, 29.8):
    print(t, ": ", fahrenheit(t))
```

The output of this script looks like this:

```
22.6 : 72.68
25.8 : 78.44
27.3 : 81.14
29.8 : 85.64
```

Optional Parameters.
Functions can have optional parameters, also called default parameters. Default parameters are parameters, which don't have to be given, if the function is called. In this case, the default values are used. We will demonstrate the operating principle of default parameters with an example. The following little script, which isn't very useful, greets a person. If no name is given, it will greet everybody:

```
def Hello(name="everybody"):
    """ Greets a person """
    print("Hello " + name + "!")

Hello("Peter")
Hello()
```

The output looks like this:

```
Hello Peter!
Hello everybody!
```

Docstring.
The first statement in the body of a function is usually a string, which can be accessed with `function_name.__doc__`. This statement is called Docstring. Example:

```
def Hello(name="everybody"):
    """ Greets a person """
    print("Hello " + name + "!")

print("The docstring of the function Hello: " + Hello.__doc__)
```

The output:

```
The docstring of the function Hello: Greets a person
```

14 Errors and fixes

14.1 NameError

Error message: “NameError: name ’now’ is not defined”
Solution: Use raw_input() for python2 and input() in python3. In python2, input() is the same as saying eval(raw_input())

IDL code	Python code
.run 'foo.pro'	exec(open("./findSecondLargestNo.py").read()) %run my_script.py (`ipython` only??)
data=READFITS('file',header)	data=pyfits.open('file')
tdata = mrdfits('SpIESch1ch2.fits',0, hdr)	tdata = data[0].data
tbdata = mrdfits('SpIESch1ch2.fits',1, hdr)	tdata = data[1].data
help, tbdata, /str	info(tbdata)
print, size(tbdata)	shape(tbdata)
print, tbdata[0].flux_aper_1	print tbdata.FLUX_APER_1[0]
help, tbdata.flux_aper_1	tbdata.FLUX_APER_1?
fluxaper = tbdata.flux_aper_1[2]	fluxaper = ???
(using fitsio)	d = fitsio.read('SpIESch1ch2.fits',1)

Table 1: IDL to Python

15 IDL to Python

Key links:
IDL to Numeric/numarray Mapping
NumPy for IDL users

http://mathesaurus.sourceforge.net/idl-numpy.html

http://mathesaurus.sourceforge.net/idl-python-xref.pdf

15.1 IDL Where

http://mathesaurus.sourceforge.net/idl-numpy.html
http://idl2python.blogspot.co.uk/2010/09/filtering-data-where-function.html
Hmmm, none of this was terribly helpful... //

16 INPUT

Just some general ways to get variables read-in and different 'tricks' to Python3 input.
https://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_3/File_IO
http://www.programiz.com/python-programming/file-operation
http://stackoverflow.com/questions/3925614/how-do-you-read-a-file-into-a-list-in-python

```
>>> s = eval(input())
>>> s = input().split()
asdf asdfasdf ddddf aa
>>> s
['asdf', 'asdfasdf', 'ddddf', 'aa']
```

```
>>> x, y, z, n = int(eval(input())), int(eval(input())),
    int(eval(input())), int(eval(input()))
>>> x, y, z, n = (int(eval(input())) for _ in range(4))
```

Would like some code to read in .dat files...

```
# Would like some code to read in .dat files...
with open('million_nos.dat') as f:
    lines = f.read().splitlines()
```

```
data = [line.strip() for line in open("million_nos.dat", 'r')]
## Still need to test this...
```

Would like some code to read in .FITS files... – see 17.

16.1 e.g. running a script from the Command Line

```
from sys import argv
# I want the "argument vector" from the system module

# When running script, you now expect/need three (additional)
    inputs/arguments..
script, first, second, third = argv

print("The script is called:", script)
print("Your first variable is:", first)
print("Your second variable is:", second)
print("Your third variable is:", third)
```

16.2 Pandas

Many thanks to Esther M-Q., John P., Joey F., Dave E., Steven B. and Nathan B for replies and helps!!!

Original question: So I have a very regular .tbl ASCII text file (first few lines of which are given below) and all I want to do is read this in (to an e.g. pandas DataFrame) and then just generate some simple variables and plots, such as R.A. vs. Dec, or color-magnitude plots. What would the general advice be here??

General idea: read the table using astropy.table and them transform it to a Pandas DataFrame with the method .to_pandas() See `http://docs.astropy.org/en/stable/table/pandas.html`.

"In this specific case I'm not exactly sure how you'd read in but looking up on the `read_table` docs it looks alright, but once you have the `dataframe` you access the columns bya `data[column_name]` which are (if it has been read in correctly) the headers.

The way to find out what columns it has read in is just
`print data.columns`

If it doesn't show up the headers you expect then it probably hasn't been read in properly and I'd recommend looking at the `read_table` docs.

```
import pandas as pd

data = pd.read_table(data_file, header=0, sep='\s+')
# header = 0 to indicate where the column names are (it also
    works with header = 'infer')
# sep = '\s+' to indicate the way to split the columns, in
    this case separated with non constant white spaces

#To use a column, e.g. RA, just do:
data.RA or data['RA']

# Note, if you have a data file with a # in the first raw, you
    have to work on that to get it of it when splitting the
    columns, or go directly to the numpy solution np.loadtxt
```

```
from astropy.table import Table,Column
tdata = Table.read('filename',format='ascii')
```

```
firstcol = tdata.columns[0].data

# or alternatively (though I think this might be deprecated
    now):
from astropy.io import ascii
tdata = ascii.read('filename')
firstcol = tdata.columns[0].data
```

```
# If it helps, you can read a comma-separated list of numbers
    from an ASCII file into a numpy array like this

import numpy as np
data = np.loadtxt( filename )

# You can then pick out individual columns by slicing the array
 secondcolumn = data[:,1]

 #and then generate plots from that array using your favorite
     plotting package.
```

"I think there are two ways to select columns in pandas. Its on this page: http://pandas.pydata.org/pandas-docs/stable/indexing.html#selection-by-label

```
  data = np.loadtxt(path,skiprows=1)
  #Then you can select, eg. the 3 column, with:
 data[:,2]
```

```
import numpy as np

x = np.zeros(99)
y = np.zeros(99)

f = open('datums.dat', 'r')

n=0
for line in f:
    n=n+1
    line = line.strip()
    columns = line.split()
    x[n-1]=float(columns[0])/1000.
    y[n-1]=float(columns[1])/1000.

f.close()
```

17 FITS Files

```
#!/usr/bin/python

"""
Some baby code just to start to play around with FITS files in
    Python and AstroPy and
make some QSO color-color plots... ;-)
"""

"""
Links to FITS resources:
  http://docs.astropy.org/en/stable/io/fits/
  http://www.astropy.org/astropy-tutorials/FITS-images.html
  https://python4astronomers.github.io/astropy/fits.html
  https://gist.github.com/phn/3054997
  http://www.astropython.org/tutorials/pyfits-fits-files-in-python93/
"""

import numpy
import matplotlib.pyplot as plt
import pyfits

# Set the path to, and the name of, your data FITS file
data_path='/cos_pc19a_npr/data/DECaLS/'
data_file='decals-dr2-DR12Q.fits'

# 'data_full' just a variable name that NPR likes to use ;-)
data_full=data_path+data_file

# Reading in and seeing the Header information
pyfits.info(data_full)

header_primary = pyfits.getheader(data_full)
list(header_primary.keys())

# Open, and get info on, the FITS file:
hdulist = pyfits.open(data_full)
hdulist.info()

# Okay, what we really want to do... ;-)
data_table = pyfits.getdata(data_full)

# Quick check on the format/dimensions of the FITS table file...
print(type(data_table),'\n')
print('The number of rows of is.... ', data_table.shape,'\n')
print('The number of columns is... ', len(data_table.names),'\n\n')

## Interograting the FITS file...
```

```
# First row
print(data_table[0])

# First column
print(data_table.field(0))

# First row, First column
print(data_table[0][0])

# Names of individual columns
print(data_table.names, '\n\n')

##
## Now getting into it some more...
##

decam_flux = data_table.field('DECAM_FLUX')

print(numpy.ndarray.max(decam_flux))

numpy.histogram(decam_flux)

plt.hist(decam_flux, bins='auto')
plt.show()
## or even
plt.show(block=False)
```

18 OUTPUT

For the "write" statement, I think you have to put everything into a string format, otherwise it just barfs...
http://learnpythonthehardway.org/book/ex16.html

```
import random

size = 1000000
lis = random.sample(range(size), size)

outfile = open('temp.dat', 'w')
for i in range(len(lis)):
    outfile.write(str(lis[i])+'\n')

outfile.close()
```

```
  outfile = open('WISE_spectra_triples_4wget_temp.dat', 'w') \\
  for i in range(len(ra)):
    print i, ra[i]
    plate_out = str(plate[i])
    mjd_out = str(mjd[i])
    fiberid_out = str(fiberid[i])

    outfile.write(plate_out+"/spec-"+plate_out+"-"+mjd_out+"-"+fiberid_out.zfill(4)+".fits
        \n")
```

19 IDL Where

As always (!! ;-) there are many different ways to do something in Python3 for the equivalent in IDL. The IDL Where command is no exception...

19.1 `a in b`

.

19.2 numpy where

Generating Indices: np.where

```
import numpy as np
rand = np.random.RandomState(42)
X = rand.randint(10, size=(3, 4))
X[np.where(X % 2 == 0)]
```

20 v2 vs. v3

https://docs.python.org/3.0/library/2to3.html
https://docs.python.org/3/howto/pyporting.html
https://docs.python.org/3/howto/pyporting.html https://docs.python.org/2/library/2to3.html
$ `2to3 -w example.py`

20.1 print

`print a` vs. `print (a)` Thus, just use () all the time!!

20.2 Division

/ = truncating (integer floor) division in P2.x when using ints; float division in P3.x // = truncating div in P2.x, P3.x

20.3 Unbound Methods

As of Python 3.0: The concept of "unbound methods" has been removed from the language. When referencing a method as a class attribute, you now get a plain function object. So this example is valid python 3.X code, since there are no "unbound methods", just functions attached to class objects. 20:09

```
$ python
```

```
Python 2.7.11 |Anaconda 2.5.0 (x86_64)| (default, Dec 6 2015,
    18:57:58)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more
    information.
>>> import mystuff
>>> mystuff.tangerine
 'Living reflection of a dream'
>>> mystuff.MyStuff
 <class 'mystuff.MyStuff'>
>>> mystuff.MyStuff.apple
 <unbound method MyStuff.apple>
```

```
$ python3
```

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more
    information.
>>> import mystuff
>>> mystuff.tangerine
 'Living reflection of a dream'
```

```
>>> mystuff.MyStuff
 <class 'mystuff.MyStuff'>
>>> mystuff.MyStuff.apple
 <function MyStuff.apple at 0x1013499d8>
```

20.4 JSON objects

```
import json
path = 'ch02/usagov_bitly_data2012-03-16-1331923249.txt'
records = [json.loads(line) for line in open(path)]
```

having

```
records = [json.loads(line) for line in open(path, 'rb')]
```

from e.g. python 2 gives an error in python 3.

21 Linear Algebra

http://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html

```
import numpy as np
from scipy import linalg
A = np.array([[1,2],[3,4]])
linalg.inv(A)
A.dot(linalg.inv(A)) #double check
```

https://twitter.com/SciPyTip/status/756510468160774144 You can solve a linear system $\mathbf{Ax} = \mathbf{b}$ with `linalg.solve(A, b)`.

22 Pytests

22.1 High-level Overview

pytest: helps you write better programs The pytest framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

"pytest is a mature full-featured Python testing tool that helps you write better programs."

http://docs.pytest.org/en/latest/

Getting started: http://docs.pytest.org/en/latest/getting-started.html

https://media.readthedocs.org/pdf/pytest/latest/pytest.pdf

http://docs.pytest.org/en/latest/contents.html

22.2 Example of a (very) simple test:

An example of a simple test:

```
# content of test_sample.py
def func(x):
    return x + 1

def test_answer():
    assert func(3) == 5
To execute it:
```

```
$ pytest
```

```
======= test session starts ========
collected 1 items

test_sample.py F

======= FAILURES ========
_______ test_answer ________

    def test_answer():
>       assert func(3) == 5
E       assert 4 == 5
E        +  where 4 = func(3)

test_sample.py:5: AssertionError
======= 1 failed in 0.12 seconds ========
```

Due to pytests detailed assertion introspection, only plain assert statements are used. See Getting Started for more examples.

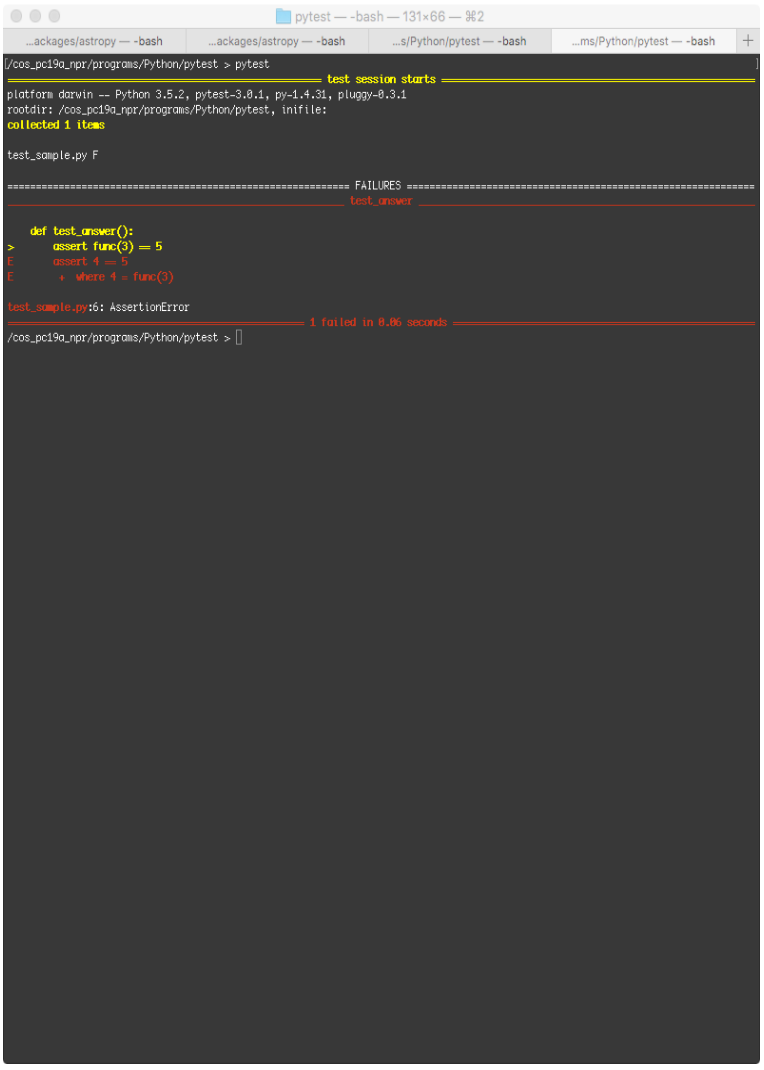
pytest — -bash — 131×66 — ⌘2
...ackages/astropy — -bash
...ackages/astropy — -bash
...s/Python/pytest — -bash
...ms/Python/pytest — -bash
/cos_pc19a_npr/programs/Python/pytest > pytest
test session starts
platform darwin -- Python 3.5.2, pytest-3.0.1, py-1.4.31, pluggy-0.3.1
rootdir: /cos_pc19a_npr/programs/Python/pytest, inifile:
collected 1 items
test_sample.py F
FAILURES
test_answer
def test_answer():
> assert func(3) == 5
E assert 4 == 5
E + where 4 = func(3)
test_sample.py:6: AssertionError
1 failed in 0.06 seconds
/cos_pc19a_npr/programs/Python/pytest >

Figure 2

23 Plots

From http://stackoverflow.com/questions/16522380/python-pandas-plot-is-a-no-show: Put

```
import matplotlib.pyplot as plt
```

at the top, and

```
plt.show()
```

at the end.

24 Pandas

(So.... this might well get split off into its separate document...)

Pandas - Python Data Analysis Library

```
import numpy
import matplotlib.pyplot as plt
import re
import pandas as pd

# The data directory PATH
data_dir = "/cos_pc19a_npr/data/WISE/W4/"

# The data directory FILENAME
data_file="WISE_W4_W4SNRge3_W4MPROlt4.0_nohdr.tbl"

data_in = data_dir+data_file

cols = ['designation', 'ra', 'dec', 'sigra', 'sigdec', 'w1mpro',
    'w1sigmpro', 'w1snr', 'w2mpro', 'w2sigmpro', 'w2snr', 'w3mpro',
    'w3sigmpro', 'w3snr', 'w4mpro', 'w4sigmpro', 'w4snr', 'w4rchi2']

## If using a file *WITH* a header
# df = pd.read_table(data_in, header=54)

df = pd.read_table(data_in)
```

24.1 DataFrames

So, pandas tends to work in these things called `DataFrames`. I (currently) don't understand DataFrames :-/

http://pandas.pydata.org/pandas-docs/stable/dsintro.html

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series

- Another DataFrame

Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

If axis labels are not passed, they will be constructed from the input data based on common sense rules.

```
In [11]: data = pd.read_table(path, delimiter='\s+')

In [12]: type(data)
Out[12]: pandas.core.frame.DataFrame

In [13]: df = pd.DataFrame(data, columns=cols)

In [14]: type(df)
Out[14]: pandas.core.frame.DataFrame

df = pd.DataFrame(data, columns=cols)
```

```
In [1]: from pandas import Series, DataFrame

In [2]: import pandas as pd
```

"Thus whenever you see `pd.` in the code, it's referring to pandas. Series and DataFrame are used so much that I find it easier to import them into the local namespace."

This is useful:
http://pandas.pydata.org/pandas-docs/version/0.15.2/10min.html
.

If youre using IPython, tab completion for column names (as well as public attributes) is automatically enabled. Heres a subset of the attributes that will be completed: In [10]: df.$< TAB >$...

```
In [24]: type(df)
Out[24]: pandas.core.frame.DataFrame

In [25]: type(df.values)
Out[25]: numpy.ndarray

In [26]: type(df.values[0])
Out[26]: numpy.ndarray

In [27]: type(df.values[0][0])
```

```
Out[27]: str
```

http://pandas.pydata.org/pandas-docs/version/0.18.1/basics.html

Okay, taking time-out to do this:
http://synesthesiam.com/posts/an-introduction-to-pandas.html

24.2 "Counting Time Zones with Pandas"

From the book and website:
https://github.com/wesm/pydata-book and "Python for Data Analysis" by Wes McKinney, published by O'Reilly Media.

```
%matplotlib inline
from __future__ import division
from numpy.random import randn
from pandas import DataFrame, Series

import os
import json
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

plt.rc('figure', figsize=(10, 6))
np.set_printoptions(precision=4)

path = 'ch02/usagov_bitly_data2012-03-16-1331923249.txt'
lines = open(path).readlines()
records = [json.loads(line) for line in lines]

frame = DataFrame(records)

frame

frame.info()

frame.info
```

24.3 "Counting Time Zones with Pandas"

```
In [30]: df = pd.DataFrame(data, columns=['ra', 'dec'])
In [31]: type(data)
In [32]: type(ra)
Out[32]: pandas.core.series.Series
```

25 Gotchas

"follow up: PYTHONPATH is a hazardous environment variable, and should never include one Python's site-packages"

See 429 in history_20150113.txt and onwards... :-)

26 A few General Notes

26.1 What's the difference between raw_input() and input()?

The difference is that `raw_input()` does not exist in Python 3.x, while `input()` does. Actually, the old `raw_input()` has been renamed to `input()`, and the old `input()` is gone (but can easily be simulated by using `eval(input())`). Reference: http://stackoverflow.com/questions/4915361/whats-the-difference-between-raw-input-and-input-in-python3-x.

26.2 Loops

```
n = eval(input())
for _ in range(n):
   <indented code here>
```

26.3 List Comprehensions

```
>>> ListOfNumbers = [ x for x in range(10) ] # List of integers
    from 0 to 9
>>> ListOfNumbers

>>> ListOfThreeMultiples = [x for x in range(100) if x % 3 == 0]
    # Multiples of 3 below 10
>>> ListOfThreeMultiples
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48,
    51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93,
    96, 99]
>>>
```

26.4 String Manipulation

```
s = 'abababababababababab'
>>> print(*s)
a g a f g a s d g a s d f a s d f a s d f a s d f a s d f
>>> type(s[1::2])
<class 'str'>
>>> s[::2]
'aaaaaaaaaa'
>>> s[1::2]
'bbbbbbbbbb'
>>>
```

26.5 Array Manipulation

```
>>> arr = [1,2,3,4]
>>> print(arr[::1])
[1, 2, 3, 4]
>>> print(arr[::-1])
[4, 3, 2, 1]
>>> print(" ".join(map(str, arr[::1])))
1 2 3 4
>>> print(" ".join(map(str, arr[::-1])))
4 3 2 1
```

27 A few general notes and commands

27.1 join()

Description: The method `join()` returns a string in which the string elements of sequence have been joined by str separator.
Syntax: Following is the syntax for `join()` method: `str.join(sequence)`.
Parameters: sequence – This is a sequence of the elements to be joined.
Example:

```
s = "-";
seq = ("a", "b", "c"); # This is sequence of strings.
print s.join( seq )
a-b-c
```

27.2 eval()

The eval function lets a python program run python code within itself.

```
x = 1
eval('x + 1')
2
eval('x')
1
```

```
l
[5, 5]
cmd
'insert(0,5)'
eval("l."+cmd)
print l
[5, 5, 5]
```

27.3 map()

`map(function, iterable, ...)`
Return an iterator that applies function to every item of iterable, yielding the results.

```
>>> def cube(x): return x*x*x
...
>>> map(cube,range(1,11))
<map object at 0x101c182e8>
>>> list(map(cube,range(1,11)))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
>>>
```

The `list()` is needed in Python 3.x.

```
def f(x): return x % 2 != 0 and x % 3 != 0
...
>>> filter(f,range(2,25))
<filter object at 0x101c18390>
>>> list(filter(f,range(2,25)))
[5, 7, 11, 13, 17, 19, 23]
>>>
```

27.4 strip()

```
>>> str = "0000000this is string example....wow!!!0000000";
>>> print (str.strip( '0' ))
this is string example....wow!!!
```

27.5 exec()

Run whole programs from the python3 command prompt (see also 1

```
>>> exec(open("./findSecondLargestNo.py").read())
```

28 Statistics

```
>>> import statistics as s
>>> s.mean([1, 2, 3, 4, 4])
2.8
```

29 OO fundamentals

This is (probably) a whole nother 'cheat sheet'/book, but I *really* need to know about at least the basics of this stuff, so here are me notes!!

http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/

http://www.jamesbooth.com/OOPBasics.htm

http://www.bentodev.org/oo.html

http://www.johnloomis.org/ece538/notes/oop_principles/oop_wikipedia.html

29.1 Inheritance

Defintion: Definies the relationship between the superclass and the subclass. "Parent" is the superclass, the child is the "subclass". A class that is derived from another class is called a subclass (also a derived class, extended class, or child class). The class from which the subclass is derived is called a superclass (also a base class or a parent class).

29.2 The `__init__` method

http://www.ibiblio.org/g2swap/byteofpython/read/class-init.html
The `__init__` method is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object. Notice the double underscore both in the beginning and at the end in the name.

http://stackoverflow.com/questions/625083/python-init-and-self-what-do-they-do
I'm learning the Python programming language, and I've come across certain things I don't fully understand. I'm coming from a C background, but I never went far with that either. What I'm trying to figure out is: In a method:

```
def method(self, blah):
  def __init__(?):
     ...
     ...
```

What does self do? what is it meant to be? and is it mandatory?
What does the `__init__` method do? why is it necessary? etc

In this code:

```
class A(object):
   def __init__(self):
```

```
        self.x = 'Hello'

     def method_a(self, foo):
        print self.x + ' ' + foo
    \begin{lstlisting}
... the self variable represents the instance of the object
itself. Most object-oriented languages pass this as a hidden
    parameter
to the methods defined on an object; Python does not. You have to
declare it explicitly. When you create an instance of the A class
    and
call its methods, it will be passed automatically, as in ...

a = A()             # We do not pass any argument to the __init__
    method
a.method_a('Sailor!') # We only pass a single argument
The __init__ method is roughly what represents a constructor in
    Python. When you call A() Python creates an object for you, and
    passes it as the first parameter to the __init__ method. Any
    additional parameters (e.g., A(24, 'Hello')) will also get
    passed as arguments--in this case causing an exception to be
    raised, since th
     \begin{lstlisting}
```

The `__init__` method is analogous to a constructor in C++, C# or Java.

30 Jupyter Notebooks

31 General Wee Tips

Need points that are evenly spaced on a log scale? Use `np.logscale(start, stop, base)`
By convention, matplotlib is imported as mpl. Also by convention, matplotlib.pyplot is imported as plt.

32 Outstanding Questions

32.1 PythonTeX

Should I/one be using PythonTeX??
https://github.com/gpoore/pythontex
https://www.ctan.org/pkg/pythontex?lang=en
http://tug.ctan.org/macros/latex2e/contrib/pythontex/pythontex.pdf
https://tug.org/tug2013/slides/Mertz-A_Gentle_Introduction_to_PythonTeX.pdf
http://tex.stackexchange.com/questions/212359/how-to-run-pythontex

Also:
http://texfigure.readthedocs.io/en/latest/

33 Glossary

Argument: The actual value of a parameter, e.g. in `methodOne(5)`, the argument passed as variable `x` is `5`.

Class: In OOP is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods). Class is a blueprint defining the charactaristics and behaviors of an object of that class type. Class names should be written in CamelCase, starting with a capital letter.
Each class has two types of variables: class variables and instance variables; class variables point to the same (static) variable across all instances of a class, and instance variables have distinct values that vary from instance to instance.

Class Constructor: is a

Class methods: are methods that are called on a class rather than an instance. They are typically used as part of an object meta-model. I.e, for each class defined an instance of the class object in the meta-model is created.

Class variable: is a variable defined in a class of which a single copy exists, regardless of how many instances of the class exist.

Getters: is a

Instances: is a

Instance variable: is a variable defined in a class for which each instantiated object of the class has a separate copy, or instance. An instance variable is similar to a class variable.

Method: in OOP is a procedure associated with an object.
The same dichotomy between instance and class members applies to methods as well; a class may have both instance methods and class methods.

Object: can be a variable, a data structure, or a function, and as such, is a location in memory having a value and possibly referenced by an identifier.

Package: is a group of similar types of classes. There are two main types of package; user-defined packages and built-in packages. We import packages to get access to classes, methods, properties, etc.

Parameter: A parenthetical variable in a function or constructor declaration. e.g. in `methodOne(int x)`, the parameter is `int x`.

Scope: This term refers to the region of the program to which an identifier applies. While it is not good practice, you can declare multiple variables within a program that use the same identifier as long as the identifiers have differing scopes; some exceptions to this are: A constructor or method

parameter will often have the same name as a class field it's intended to initialize or modify. It is customary to use i as the condition variable in a for-loop (and, in cases of nested for-loops, to use j as the condition variable for the inner loop).

Setters:

34 Useful Resources

Borrows, begs and steals from:

General Python Resources

http://docs.python.org/3.5/tutorial/
http://docs.scipy.org/doc/numpy/reference/routines.array-manipulation.html
http://www.scipy-lectures.org/intro/numpy/numpy.html
https://sites.google.com/site/aslugsguidetopython/
https://sites.google.com/site/aslugsguidetopython/data-analysis/array-manipulation

Inter-active links

http://interactivepython.org/runestone/static/pythonds/SortSearch/TheBubbleSort.html
http://pythoncentral.io/time-a-python-function/

Teaching yourself Python

https://olimex.wordpress.com/2014/06/12/collection-of-51-free-e-books-for-python-programming/
http://www.tutorialspoint.com/python/
http://www.tutorialspoint.com/python/python_classes_objects.htm
http://codingbat.com/python
https://wiki.python.org/moin/ProblemSets
https://www.hackerrank.com/

IDL to Python

http://www.astro.umd.edu/*sim*mbk/idl-numpy.html
http://www.cv.nrao.edu/ aleroy/pytut/topic2/intro_fits_files.py http://www.johnny-lin.com/cdat_tips/tips_array/idl2num.html
http://www.astrobetter.com/idl-vs-python/
http://www.astrobetter.com/wiki/tiki-index.php?page=Python+Switchers+Guide
http://mathesaurus.sourceforge.net/
http://mathesaurus.sourceforge.net/idl-numpy.html
http://www.scicoder.org/mapping-idl-to-python/
http://mathesaurus.sourceforge.net/idl-python-xref.pdf

http://www.thelearningpoint.net/computer-science/learning-python-programming-and-data-structures/learning-python-programming-and-data-structures–tutorial-

15–generators-and-list-comprehensions
https://jeffknupp.com/blog/2014/06/18/improve-your-python-python-classes-and-object-oriented-programming/
http://learnpythonthehardway.org/
http://learnpythonthehardway.org/book/ex40.html

Astro Python

http://www.iac.es/sieinvens/siepedia/pmwiki.php?n=HOWTOs.EmpezandoPython
http://danmoser.github.io/notes/python_astro.html

Transitioning to Data Science

Words and links from http://insightdatascience.com/blog/transition_to_ds.html.

Programming: There are many languages for conducting data science work: Python, R, MATLAB, Stata, SAS, and so on. However, we've found the the general trend in data science is towards Python[3]. Python is a general purpose programming language that has a growing number of modules for data analysis, including SciPy, Numpy, Pandas, StatsModels, and Scikit-learn, as well as many visualization tools like seaborn, matplotlib, and ggplot.

Action Items:

- To get started, Codecademy has an excellent python course that only takes an estimated 13 hours to complete.
- Google's Python Class. remains a perennial favorite among Insight Fellows.
- If you have a bit more time, we recommend Zed Shaw's excellent Learn Python the Hard Way.
- Become familiar with the Jupyter notebook, which is increasingly popular among data scientists for sharing code and ideas.

[3]See e.g., CodeEval blog; Breakdown of the 9 Most In-Demand Programming Languages; http://statisticstimes.com/tech/top-computer-languages.php and of course, Tiobe.