Python "Cheat Sheet"

Nicholas P. Ross

July 27, 2016

Abstract

The is my (NPR's) version of a ∼Python "Cheat Sheet", (including some parts for folks that want to migrate form IDL).

1 Real basics

1.1 iPython

$ conda update ipython

1.2 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)]
```

1.3 iPython from Fernando Perez

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

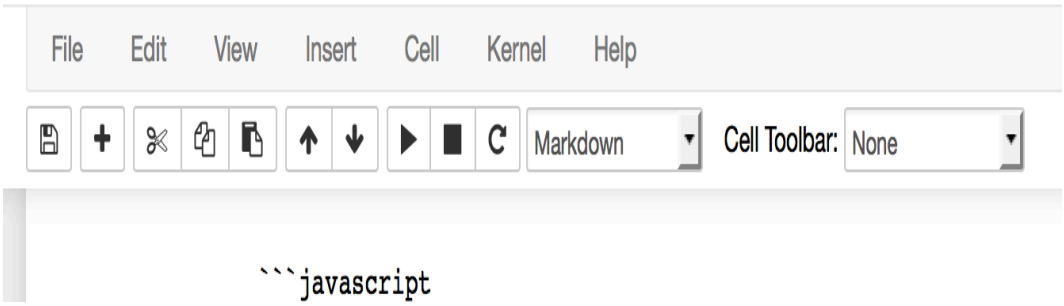

Figure 1: Clicking on the Cell Toolbar "Code", "Markdown" etc. will power what happens in the Clells!!!

1.4 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...
(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%20Markdown%20Cells.ipynb

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

2 Data Types

```
>>> n = 123
>>> f = 123.
>>> L = [1,2,3]
>>> a = (1,2,3)
>>> D = {1,2,3}
>>> s = '1,2,3'

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

3 Britton's Classes :-)

3.1 "If lost in the desert..."

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

3.2 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
```

3.3 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.

4 Key packages

`matplotib`
`numpy`
`scipy`
`ipython`
`pandas`
`sympy`
`nose`
`pyds9`
`pyFITS`
`yt`

5 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.

IDL code	Python code
.run 'foo.pro'	exec(open("./findSecondLargestNo.py").read())
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

- Inheritance: The transfer of the characteristics of a class to other classes that are derived from it.

6 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

7 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...
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...
```

8 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")
```

9 IDL Where...

10 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`

10.1 print

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

10.2 Division

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

11 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)`.

12 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... :-)

13 A few General Notes

13.1 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.

13.2 Loops

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

13.3 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]
>>>
```

13.4 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'
>>>
```

13.5 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
```

14 A few general notes and commands

14.1 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
```

14.2 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]
```

14.3 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]
>>>
```

14.4 strip()

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

14.5 exec()

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

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

15 Statistics

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

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.

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

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.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

Also, http://www.cv.nrao.edu/ aleroy/pytut/topic2/intro_fits_files.py