NPRs Python Notes

Nicholas P. Ross

August 2, 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 snowballed from there. 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 Real basics **4**
1.1 Versions 4
1.2 Style Guide for Python Code 4
1.3 Notebook 4

2 iPython **6**
2.1 iPython from Fernando Perez 6

3 Data Types **7**
3.1 e.g. Data types 7
3.2 Dictionaries 7

4 A general code example ;-) **8**

5 Britton's Classes :-) **10**
5.1 "If lost in the desert..." 10
5.2 Lists 10
5.3 Dictionaries and Maps 11

6 Key packages **13**

7 Class vs. an Instance **14**
7.1 Abstract Classes 14

8 Function **16**

9 Errors and fixes **18**
9.1 NameError 18

10 IDL to Python **19**

11 INPUT **20**
11.1 e.g. running a script from the Command Line 20

12 OUTPUT **21**

13 IDL Where **22**

14 v2 vs. v3 **23**
14.1 print 23
14.2 Division 23
14.3 Unbound Methods 23

15 Linear Algebra **25**

16 Gotchas **26**

17 A few General Notes **27**
17.1 What's the difference between raw_input() and input()? . . . 27
17.2 Loops . . . 27
17.3 List Comprehensions . . . 27
17.4 String Manipulation . . . 27
17.5 Array Manipulation . . . 28

18 A few general notes and commands **29**
18.1 join() . . . 29
18.2 eval() . . . 29
18.3 map() . . . 29
18.4 strip() . . . 30
18.5 exec() . . . 30

19 Statistics **31**

20 OO fundamentals **32**
20.1 Inheritance . . . 32
20.2 The `__init__` method . . . 32

21 General Wee Tips **34**

22 Glossary **35**

23 Useful Resources **37**

1 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...

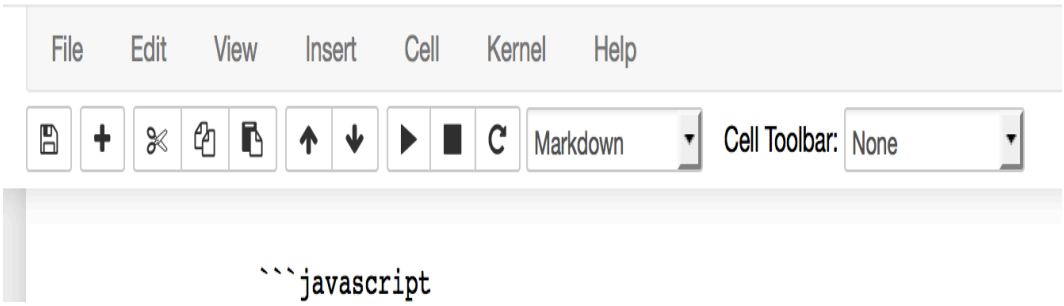

Figure 1: Clicking on the Cell Toolbar "Code", "Markdown" etc. will power what happens in the Clells!!!

(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 iPython

https://ipython.org/install.html

```
$ conda update ipython
```

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

2.1 iPython from Fernando Perez

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

3 Data Types

3.1 e.g. Data types

```
>>> n = 123
>>> f = 123.
>>> L = [1,2,3]
>>> a = (1,2,3)
>>> D = {1,2,3}
>>> x = {'1': '2', '3': 45}
>>> 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(x)
<class 'dict'>
>>> type(s)
<class 'str'>
```

3.2 Dictionaries

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

4 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.
```

---

5 Britton's Classes :-)

5.1 "If lost in the desert..."

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

5.2 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
```

5.3 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.

6 Key packages

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

To install:
`python3 -m pip install somepackage`

7 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.

7.1 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.

8 Function

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
```

9 Errors and fixes

9.1 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

10 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

11 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...

11.1 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)
```

12 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")
```

13 IDL Where

The Python3 equivalent of the IDL Where command is...

14 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`

14.1 print

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

14.2 Division

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

14.3 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>
```

15 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)`.

16 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... :-)

17 A few General Notes

17.1 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.

17.2 Loops

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

17.3 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]
>>>
```

17.4 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'
>>>
```

17.5 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
```

18 A few general notes and commands

18.1 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
```

18.2 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]
```

18.3 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]
>>>
```

18.4 strip()

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

18.5 exec()

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

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

19 Statistics

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

20 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

20.1 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).

20.2 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.

21 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.

22 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:

23 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.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

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[1]. 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.

[1]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.