Karmel Allison • UCSD BioSys Boot • 2013

PYTHON

an introduction

First things first

This is **NOT** a comprehensive guide.

This is an **overview** of concepts that assume you know a few things about programming.

If you find yourself interested in the details, **Google**.

If you find yourself lost and confused, consider taking a **basic CS course**:

CSE 5A, CSE 12, CSE 101

Outline

1. Why Python?
2. Some historical/cultural knowledge
3. Basic language concepts
4. NumPy and SciPy
5. Pandas
6. Appendix

Note that I have pulled liberally from various sources, including

- Python Workshop (ACM@UIUC) slides, 1999 (http://www.python.org/doc/essays/ppt/acm-ws/index.htm)

WHY?

Why use Python?

- Rules are enforced
- Easy to learn, easy to use
- A vibrant scientific community
- Major investments from big companies (i.e., Google)
- It can do anything [X] can do, usually better
- If you need it to be faster too, C bindings exist, with LLVM in progress

```
for language in (‘R’, ‘matlab’, ‘C’, ‘C++’, ‘Java’, ‘Ruby’):
    print ‘‘‘There are advantages
             and disadvantages to {0},
             but I prefer Python’’’.format(language)
```

Why not use R?

"I have been worried for some time that R isn't going to provide the base that we're going to need for statistical computation in the future. (It may well be that the future is already upon us.) There are certainly efficiency problems (speed and memory use), but there are more fundamental issues too. Some of these were inherited from S and some are peculiar to R.

One of the worst problems is scoping. Consider the following little gem.

```
f =function() {
if (runif(1) > .5)
x = 10
x
}
```

The x being returned by this function is randomly local or global. There are other examples where variables alternate between local and non-local throughout the body of a function. No sensible language would allow this. It's ugly and it makes optimisation really difficult. This isn't the only problem, even weirder things happen because of interactions between scoping and lazy evaluation.

In light of this, **I have come to the conclusion that rather than "fixing" R, it would be much more productive to simply start over and build something better.** I think the best you could hope for by fixing the efficiency problems in R would be to boost performance by a small multiple, or perhaps as much as an order of magnitude. This probably isn't enough to justify the effort (Luke Tierney has been working on R compilation for over a decade now)..."

– Ross Ihaka, creator of R

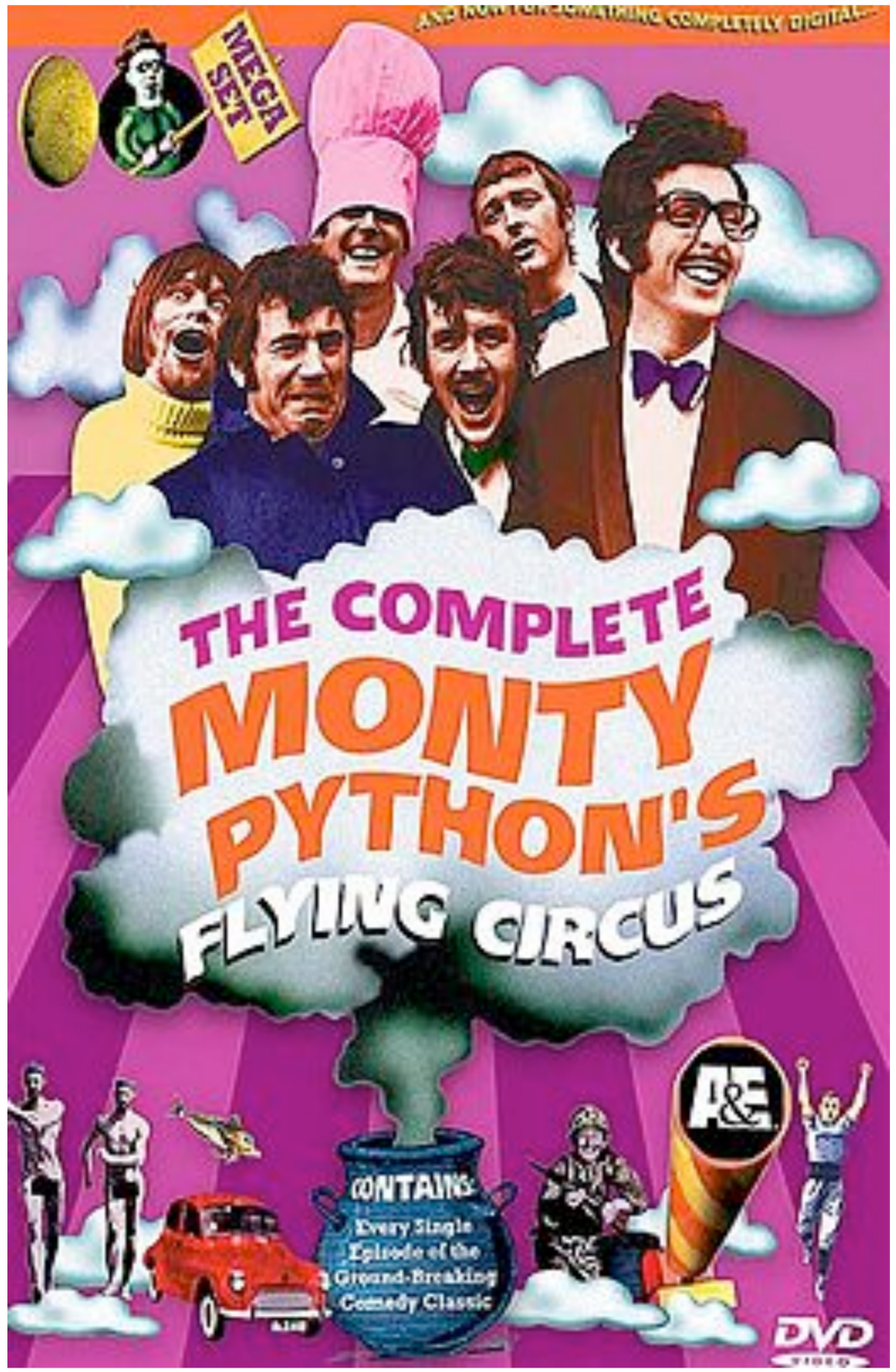
CULTURE

Some history & culture

Guido van Rossum, creator of Python
a.k.a Benevolent Dictator for Life

Some history & culture

- First conceived in the late 80s
- Development started in 1989
- Named for Monty Python
- Open-source
- Designed to look like pseudo-code



Some history & culture

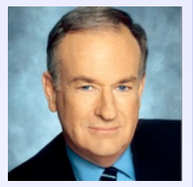
```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

BASICS

The basics

- Object-oriented, high-level, dynamically typed
- Not just Perl 2.0– can be used for scripting, but it does oh-so-much-more
- Compiled to interpreted byte code (*.pyc) automatically
- Garbage collection and memory management automatically handled via reference counting

We're doing it live!



```
# Open a Python console on your machine.
cmm171-164:~ karmel$ python
Enthought Python Distribution -- www.enthought.com
Version: 7.2-2 (64-bit)

Python 2.7.2 |EPD 7.2-2 (64-bit)| (default, Sep  7 2011, 16:31:15)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "packages", "demo" or "enthought" for more information.
>>> print('Hello World!')
Hello World!
```

Crucial concept #1

Whitespace matters.

Take those ugly brackets and helter-skelter code elsewhere!

No semicolons!

```
# An indent is four spaces
class MyObject(object):

    def my_function(self, some_list):
        for x in some_list:
            # Note that there are no brackets,
            # but indents define the code blocks
            x.perform_action()
        return True

# This will not work!
class MyObject(object):

    def my_function(self, some_list):
        for x in some_list:
           # Note that there are no brackets,
            # but indents define the code blocks
            x.perform_action()
        return True
```

Crucial concept #2

Everything is an object.

Like, everything.

What are the properties of an object?

- Can have attributes and methods
- Can be assigned to a variable
- Can be passed as an argument to a function

Everything is an object?

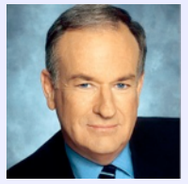
Yes. Strings, lists, functions, modules– everything.

What does this mean practically?

```
# Let's say we have a function
>>> def my_function():
...     ''' Returns a greeting when called. '''
...     return 'Hello World!'
...
# The function itself is an object
>>> my_function
<function my_function at 0x10049e848>

# It comes with built-in methods
>>> my_function.__doc__
' Returns a greeting when called. '
>>> my_function.__repr__
<method-wrapper '__repr__' of function object at 0x10049e848>

# __repr__ itself is a function.
# It returns the string representation of the object it belongs to.
>>> my_function.__repr__()
'<function my_function at 0x10049e848>'

# We can replace the method arbitrarily
>>> my_function.__repr__ = my_function
>>> my_function.__repr__()
'Hello World!'

# Or arbitrarily define new attributes
>>> my_function.counter = 1
>>> my_function.counter += 1
>>> my_function.counter
2
```

```
# Or we can pass the function as an argument to another function
>>> def wrapper_function(inner_f):
...     ''' Calls passed function. '''
...     return inner_f()
...
>>> wrapper_function(my_function)
'Hello World!'
>>> wrapper_function(wrapper_function.__repr__)
'<function wrapper_function at 0x10049e8c0>'
```

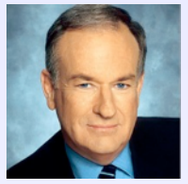
We're doing it live!



```
# I said everything is an object.
# What about basic types like integers?
# Does an int have __doc__ and __repr__ methods? What do they return?
# Can you assign an arbitrary attribute to an int?
```

```
# Or we can pass the function as an argument to another function
>>> def wrapper_function(inner_f):
...     ''' Calls passed function. '''
...     return inner_f()
...
>>> wrapper_function(my_function)
'Hello World!'
>>> wrapper_function(wrapper_function.__repr__)
'<function wrapper_function at 0x10049e8c0>'
```

We're doing it live!



```
# I said everything is an object.
# What about basic types like integers?
# Does an int have __doc__ and __repr__ methods? What do they return?
# Can you assign an arbitrary attribute to an int?
>>> var = 1
>>> type(var)
<type 'int'>
>>> var.__doc__
'int(x[, base]) -> integer\n\nConvert a string or number to an integer, if
possible.  A floating point...'
>>> var.__repr__()
'1'
>>> var.arbitrary_attribute = True
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no attribute 'arbitrary_attribute'
```

Crucial concept #3

Objects are dynamically typed, but not type-less.

(Think of Python types like gender– yes, there are rules, but the rules are easy to bend.)

Basic types:

- Integers (equivalent to C longs)
- Floating-point numbers (equivalent to C doubles)
- Long integers of non-limited length
- Complex numbers
- Strings
- Some others, such as type and function

Composite types:

- Lists
- Tuples
- Dictionaries (also called dicts, hashmaps, or associative arrays)

You can check type.

```
>>> type('This is a string')
<type 'str'>
>>> type(1.234)
<type 'float'>
```

But you shouldn't. A rule of life that is also pythonic:

It's better to ask forgiveness than to ask permission.

In other words, don't type-check; just do what should work, and handle errors accordingly.

```
>>> def expects_a_number(number): return number + 10
...
>>> expects_a_number('This is a string')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 1, in expects_a_number
TypeError: cannot concatenate 'str' and 'int' objects
>>> def expects_a_number(number):
...     try: return number + 10
...     except TypeError: return 'Try again, silly!'
...
>>> expects_a_number('This is a string')
Try again, silly!
```

The basics: Numbers

The usual notations and operators

```
>>> 12*3, 12+3, 12**3
(36, 15, 1728)
>>> 12.3*4
49.2
>>> 0xff # Hexadecimals
255
>>> 0377 # Octals
255
>>> 0 < 1 <= -5
False
```

C-style shifting and masking

```
>>> 0xff & 0xf6
246
>>> 1<<16
65536
>>> 0&1, 0|1
(0, 1)
```

The basics: Numbers

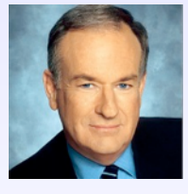
Integer division truncates in Python 2.x! (Known harmful)

```
>>> 12/5
2
>>> from __future__ import division
>>> 12/5
2.4
```

Longs and complex numbers

```
>>> 2**100
1267650600228229401496703205376L
>>> 1j**2
(-1+0j)
```

We're doing it live!



```
# What's the deal with floating-point arithmetic? (David Gay)
>>> 12*.4
4.800000000000001
>>> 12*.5
6.0
```

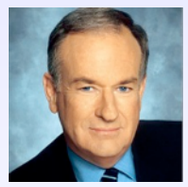
The basics: Strings

```
>>> 'spam'
'spam'
>>> 'spam' + 'eggs'
'spameggs'
>>> 'ni'*4
'nininini'
>>> '''Multi-line
... string'''
'Multi-line\nstring'
>>> "Also a string; doesn't matter whether you single or double, but be
consistent."

>>> 'Standard formatting styles %s, %.2f' % ('abc', 23.4554)
'Standard formatting styles abc, 23.46'
>>> 'Newer and hipper is to use ordered {} like these {}'.format('keys',2)
'Newer and hipper is to use ordered keys like these 2'
>>> 'Works with named {noun}, too.'.format(noun='keys')
'Works with named keys, too.'
```

The basics: Strings

```
>>> 'ACGT'[0]
'A'
>>> 'ACGT'[:2]
'AC'
>>> 'ACGT'[-1:]
'T'
>>> len('ACGT')
4
>>> 'ACGT' in 'ACGCTCGTCGCTACGCTCGACTCGCT'
False
>>> 'ACGT' < 'AGGT'
True
```

We're doing it live!

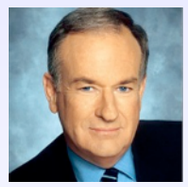


```
# What's the easiest way to reverse a string?
```

The basics: Strings

```
>>> 'ACGT'[0]
'A'
>>> 'ACGT'[1:3]
'CG'
>>> 'ACGT'[-1:]
'T'
>>> len('ACGT')
4
>>> 'ACGT' in 'ACGCTCGTCGCTACGCTCGACTCGCT'
False
>>> 'ACGT' < 'AGGT'
True
```

We're doing it live!



```
# What's the easiest way to reverse a string?
>>> 'ACGT'[::-1]
'TGCA'
```

The basics: Variables

No need to declare.

Not typed.

Can't start with a digit.

Usually underscored, all lowercase, but that's a matter of choice.

```
>>> my_seq = 'ACGCTCGTCGCTACGCTCGACTCGCT'
>>> if 'ACG' in my_seq:
...     my_seq = False
...
>>> my_seq
False
```

The basics: Lists

Flexible arrays, not Lisp-like linked lists

```
>>> a = [99, 'bottles of beer', ['on', 'the', 'wall']]
```

Same operators as for strings

```
>>> b = ['!']
>>> a+b
[99, 'bottles of beer', ['on', 'the', 'wall'], '!']
>>> b*3
['!', '!', '!']
>>> a[0]
99
>>> a[-1:]
[['on', 'the', 'wall']]
```

Item and slice assignment

```
>>> a[0] = 98
>>> a[1:2] = ['bottles', 'of', 'beer']
>>> a
[98, 'bottles', 'of', 'beer', ['on', 'the', 'wall']]
```

The basics: Lists

Learn to love lists.

```
>>> a = range(5)      # [0,1,2,3,4]
>>> a.append(5)       # [0,1,2,3,4,5]
>>> a.pop()           # [0,1,2,3,4]
5
>>> a.insert(0, 5.5) # [5.5,0,1,2,3,4]
>>> a.pop(0)          # [0,1,2,3,4]
5.5
>>> a.reverse()       # [4,3,2,1,0]
>>> a.sort()          # [0,1,2,3,4]
>>> list('ACGTGCGT')
['A', 'C', 'G', 'T', 'G', 'C', 'G', 'T']

>>> for x in a: print x**2
...
0
1
4
9
16
```

The basics: Tuples

Like lists, but immutable

```
>>> (0,1,2,3)
(0, 1, 2, 3)
>>> 0,1,2,3
(0, 1, 2, 3)

>>> point = 1,2,3
>>> point[1]
2

>>> tup_of_tups = (('one',1),('two',2),('three',3))
>>> for key, val in tup_of_tups: print key*val
...
one
twotwo
threethreethree
```

The basics: Dictionaries

Key, value stores (hashmaps)

```
>>> student_ids = {'Peter': 1232394, 'Paul': 9473723, 'Mary': 982345}
>>> student_ids['Peter']
1232394
>>> student_ids['Peter'] = 2395954
>>> student_ids['Joe']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Joe'
```

Can be created from lists of tuples

```
>>> d = dict(tup_of_tups)
>>> d
{'three': 3, 'two': 2, 'one': 1}
```

Keys can be any hashable type (= immutable), values can be anything

```
>>> {('tuple', 'key'): 'string val', 1.234: 'A float key!'}
```

The basics: Dictionaries

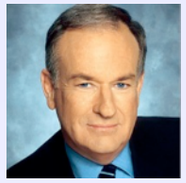
Purposefully unordered!

```
>>> student_ids = {'Peter': 1232394, 'Paul': 9473723, 'Mary': 982345}
>>> for key in student_ids: print key
...
Paul
Peter
Mary
```

Though as of Python 2.7, a special OrderedDict class exists

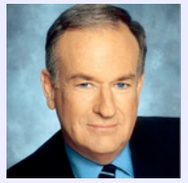
```
>>> from collections import OrderedDict
>>> student_ids_sorted = OrderedDict((('Peter', 1232394), ('Paul', 9473723),
('Mary', 982345)))
>>> for key in student_ids_sorted: print key
...
Peter
Paul
Mary
```

We're doing it live!

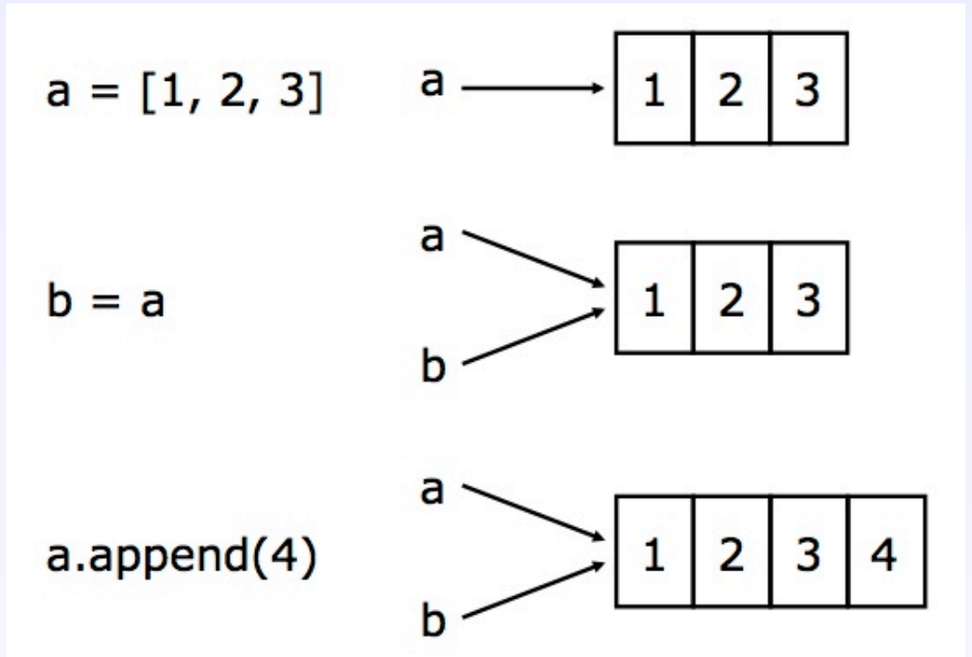


```
# Does Python assign by reference or copying?
# Clues:
# Try
>>> a = list('ACGT')
>>> b = a

# vs
>>> s = 'ACGT'
>>> t = s

# What happens when you change the value of a or s?
```

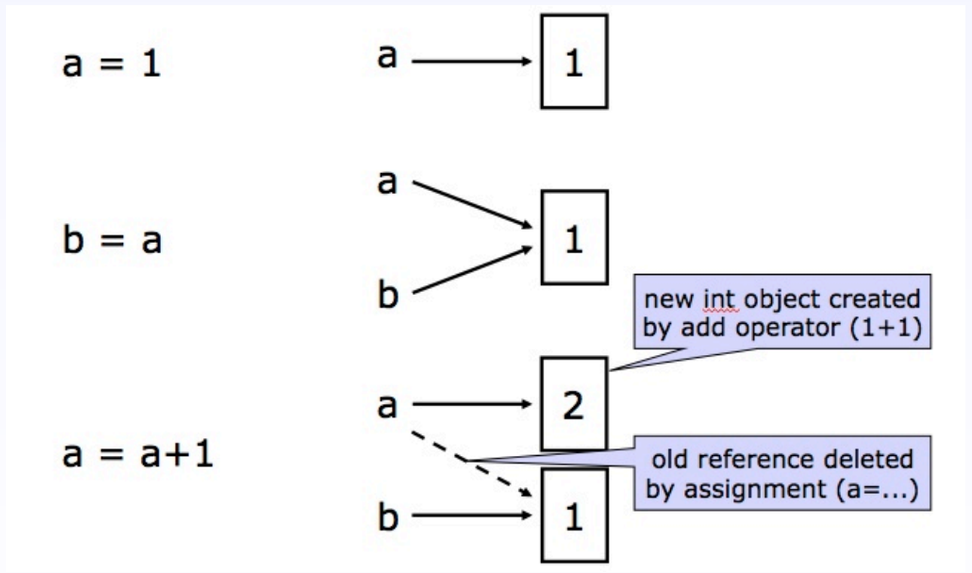
We're doing it live!



```
# Does Python assign by reference or copying?
>>> a = list('ACGT')
>>> b = a
>>> a
['A', 'C', 'G', 'T']
>>> b
['A', 'C', 'G', 'T']
>>> a[0] = 10
>>> a
[10, 'C', 'G', 'T']
>>> b
[10, 'C', 'G', 'T']
```




```
>>> s = 'ACGT'
>>> t = s
>>> s += 'NNN'
>>> s
'ACGTNNN'
>>> t
'ACGT'
```




```
# ANSWER: Assignment creates references, not copies.
# But immutables behave as if they are copies!
```

The magic: List comprehension

For loops, concisely.

```
>>> my_list = range(0,10)
>>> [x**2 for x in my_list]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [x**2 for x in my_list if x%2 == 0]
[0, 4, 16, 36, 64]
```

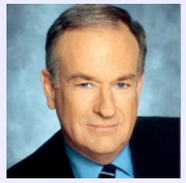
Works with any sequence object (strings, tuples, custom).

```
>>> [char*2 for char in 'ACGTGCCC']
['AA', 'CC', 'GG', 'TT', 'GG', 'CC', 'CC', 'CC']
```

Super fun when you know how to use them.

```
# Reverse complement?
>>> complement = {'A':'T', 'C':'G', 'G':'C', 'T':'A'}
>>> seq = 'ACGTACGTACT'
>>> ''.join([complement[base] for base in seq[::-1]])
'AGTACGTACGT'
```

We're doing it live!

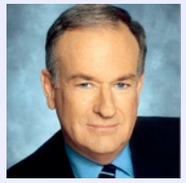


```
# You have a table of genes and expression levels:
```

Gene	Value
A	12.5
B	25
C	50
D	100
E	200
F	400

```
# The expression levels are raw tag counts.
# You want to normalize the raw tag counts.
# The raw values assume a total tag count of 1,500,000.
# You want to normalize the expression levels so that assume
# a total tag count of 10,000,000 tags.

# Create a data structure that represents the table above.
# Store the normalized version of the expression level values.
# Print out a new table with gene, normalized value pairs.
```

We're doing it live!



```
# You have a table of genes and expression levels:
```

Gene	Value
A	12.5
B	25
C	50
D	100
E	200
F	400

```
from __future__ import division
gene_table = (('A', 12.5),
              ('B', 25),
              ('C', 50),
              ('D', 100),
              ('E', 200),
              ('F', 400))
norm_table = ((gene, val*10000000/1500000) for gene, val in gene_table)
for gene, val in norm_table: print '{}\t{}'.format(gene,val)
A    83.3333333333
B    166.666666667
C    333.333333333
D    666.666666667
E    1333.33333333
F    2666.66666667
```

The rest: Python docs

There is oh-so-much more to learn and love about Python.

http://python.org

http://docs.python.org

http://mail.python.org/

http://stackoverflow.com/questions/tagged/python

NUMPY

+

SCIPY

NumPy: For serious math

The Python standard lib has many mathematical functions and routines:

complex numbers, Decimal objects, fractions, trig functions, logarithmic functions, random number generators, hyperbolic functions, etc.

But if you need bigger, matlabbier functions, NumPy is for you.

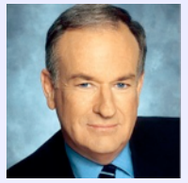
"NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a **multidimensional array** object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more."

NumPy: Arrays

At the core of NumPy is the array object.

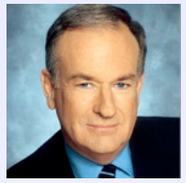
```
>>> np.array([[0,1,2],[3,4,5],[6,7,8]])
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> np.arange(0,40).reshape(2,2,10)
array([[[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]],

       [[20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35, 36, 37, 38, 39]]])
>>> np.array([[0,1,2],[3,4,5],[6,7,8]])*10
array([[ 0, 10, 20],
       [30, 40, 50],
       [60, 70, 80]])
>>> np.array([[0,1,2],[3,4,5],[6,7,8]])[::-1]
array([[6, 7, 8],
       [3, 4, 5],
       [0, 1, 2]])
```

We're doing it live!



```
# Create two random 10x10 arrays.
>>> import numpy as np
>>> a = np.random.rand(10,10)
>>> b = np.random.rand(10,10)

# Create a 10x10 array made up of alternating horizontal stripes
# from the two random arrays.
```

We're doing it live!



```
# Create two random 10x10 arrays.
>>> import numpy as np
>>> a = np.random.rand(10,10)
>>> b = np.random.rand(10,10)

# Solution 1:
c = []
for i in range(0,len(a)):
    if i % 2 == 0:
        c.append(a[i])
    else:
        c.append(b[i])
c = np.array(c)

# Solution 2:
c = np.vstack([a,b])[[i + i%2*len(a) for i in range(0,len(a))]]
```

... and that's just the beginning:

masked arrays, transforms, linear algebra, polynomials,
stats, sorting and searching, indexing, all sorts of array-based algorithms,
etc.

SciPy: For serious science

High-level data manipulation, analysis, and visualization routines.

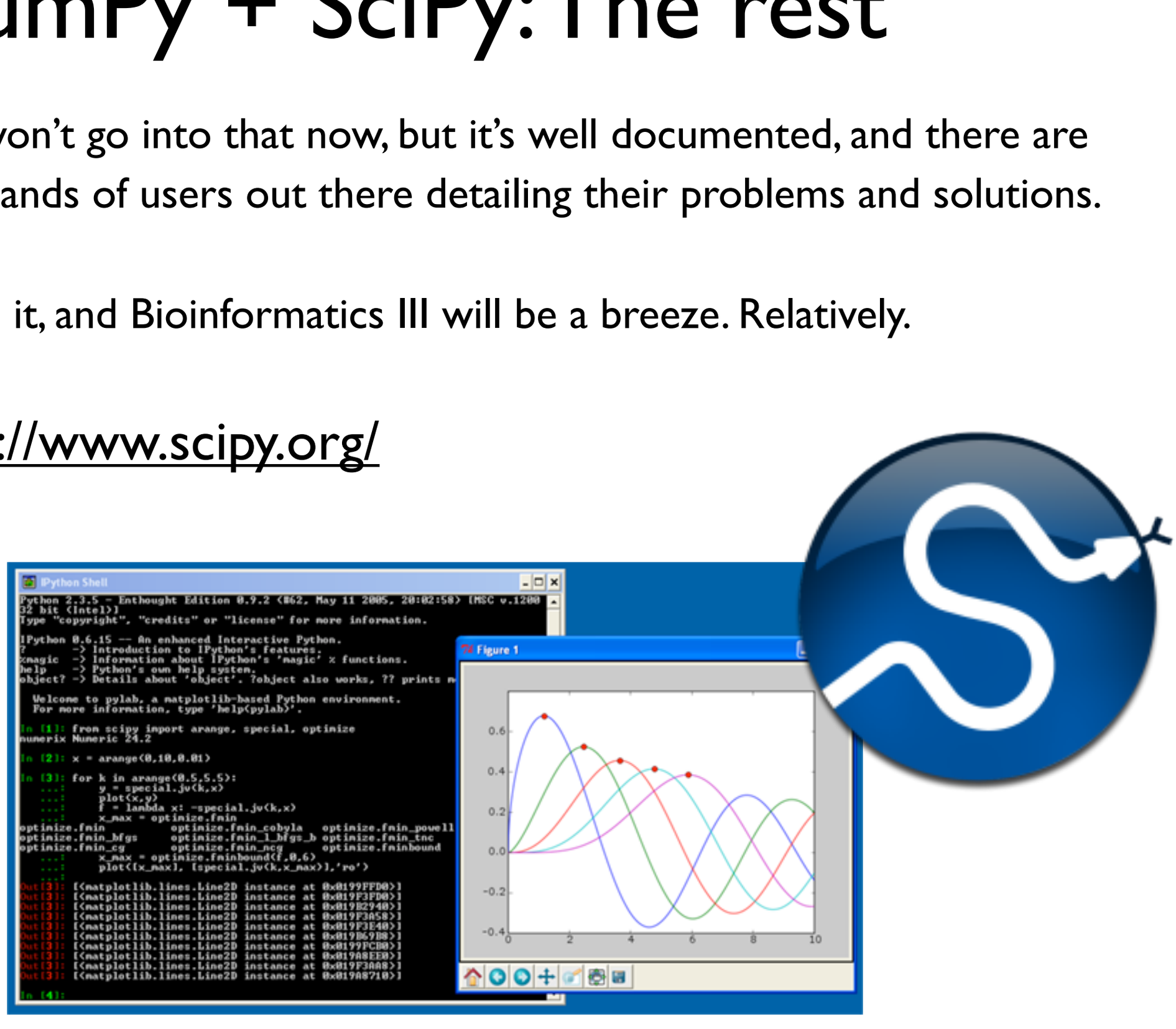
Subpackage	Description
`cluster`	Clustering algorithms
`constants`	Physical and mathematical constants
`fftpack`	Fast Fourier Transform routines
`integrate`	Integration and ordinary differential equation solvers
`interpolate`	Interpolation and smoothing splines
`io`	Input and Output
`linalg`	Linear algebra
`ndimage`	N-dimensional image processing
`odr`	Orthogonal distance regression
`optimize`	Optimization and root-finding routines
`signal`	Signal processing
`sparse`	Sparse matrices and associated routines
`spatial`	Spatial data structures and algorithms
`special`	Special functions
`stats`	Statistical distributions and functions
`weave`	C/C++ integration

NumPy + SciPy: The rest

We won't go into that now, but it's well documented, and there are thousands of users out there detailing their problems and solutions.

Learn it, and Bioinformatics III will be a breeze. Relatively.

http://www.scipy.org/



PANDAS

YOUR TASK

Your Homework: The task

1. Download this microarray gene expression data set for 4 cell lines:
   http://www.broadinstitute.org/mpr/publications/projects/SOM_Methods_and_Applications/data_set_HL60_U937_NB4_Jurkat.txt
2. Read the description of the data set:
   http://www.broadinstitute.org/mpr/publications/projects/SOM_Methods_and_Applications/Datasets_description.txt
3. Using Python and pandas:
   a. Import the data file.
   b. Convert the data file into a DataFrame.
   c. Write functions to answer questions A - G (following page).

Your Homework: Questions

A. How many distinct genes are represented in the data set?
B. Which two time points are the most highly correlated for each cell type?
C. Which two cell types are the most similar?
D. It is often useful to know which genes change very little across samples for the sake of normalization or calibration. Based on this data set, what are ten good candidates for genes to use to calibrate machinery or analyses across all these samples?
E. Do any genes show two-fold higher expression at 24 hours versus 0 hours for all four cell types? If so, which ones?
F. Which genes are differentially regulated (at least two-fold higher or lower) in HL60 cells as compared to U937 cells at 0 hours?
G. Take the list of Gene Accession codes from (F), and run them through the DAVID ontology analyzer. (at http://david.abcc.ncifcrf.gov/summary.jsp . These are GenBank Accession codes.) Are there any enriched ontology terms?

Your Homework: Restrictions

- You may work in groups, but each student should turn in their own assignment.
- Python is not just for scripting! Your final code should be structured as an object that has methods that separately import the file, determine each desired answer, etc. (See example on following page.)
- Points (i.e., mad props) given for style, clarity, and reusability in code.
- You should turn in the answers to questions A - G as a **plain text file**, and also **the code** used to generate the answers. Do not print or email the answers and code!
  - **Create a directory** in the Github repo:

    Homework/python-microarrays/your_name/
  - **Add** the answers (.txt) file and the code (.py) file to your directory.
  - Make sure to **commit your changes** to the biosys-2013 branch of the repo.

Your Homework: Example

```
from pandas import DataFrame
from pandas.io import parsers

class ExpressionAnalyzer(object):
    '''
    Responsible for importing and analyzing gene expression data.
    Expects data with named samples as columns and genes as rows.
    '''
    def import_file(self, filename):
        ''' Import data from file and return a DataFrame object. '''
        ...
...

if __name__ == '__main__':
    ea = ExpressionAnalyzer()
    ea.import_file('path_to_file')
    ...

========
cmm171-164:~ karmel$ python expression_analyzer.py
```

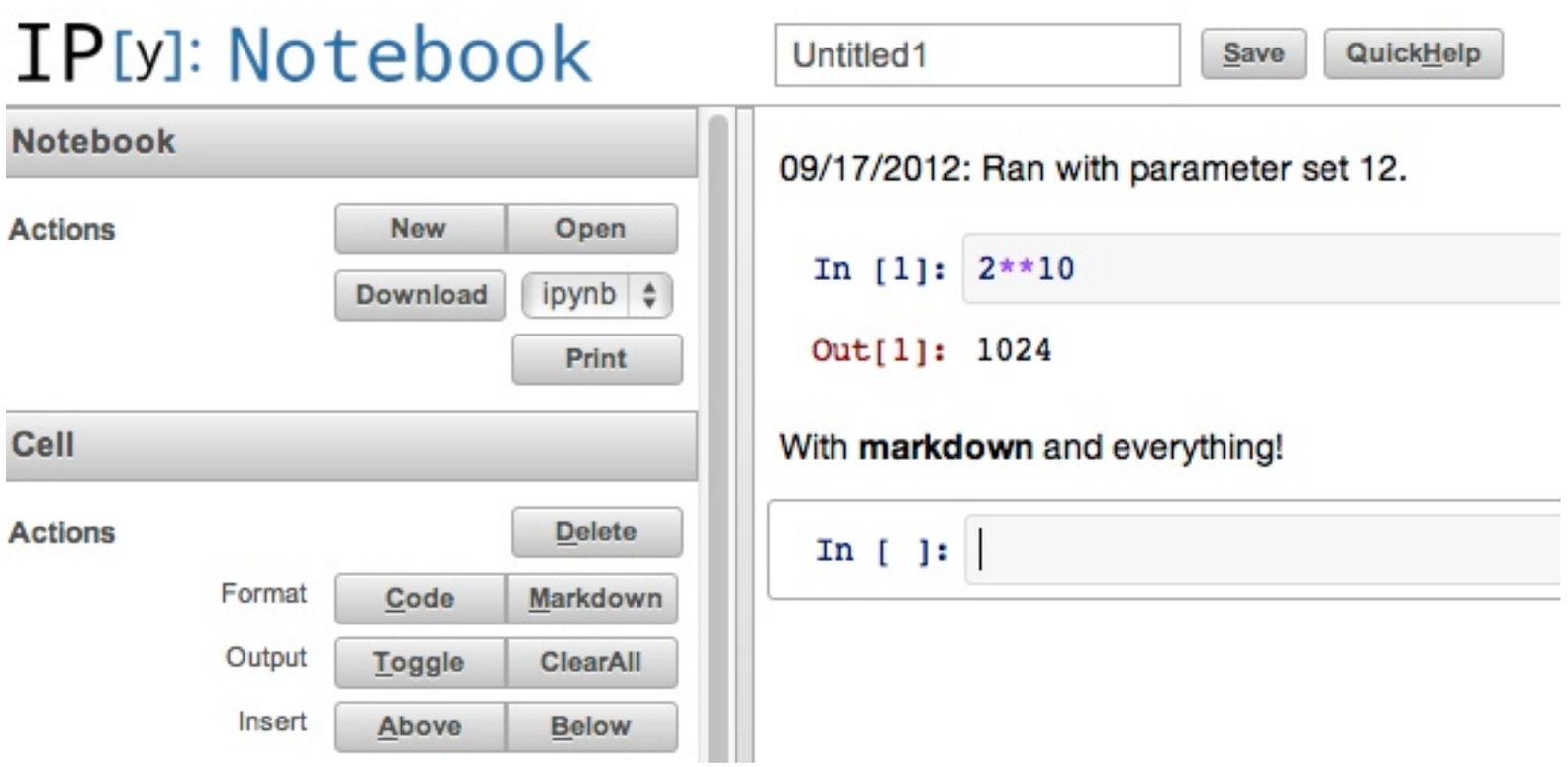
APPENDIX

Other useful packages

Matplotlib for all your visualization needs: http://matplotlib.org

ipython has syntax highlighting, code completion in the console: http://ipython.org
bpython is an alternative with the same: http://bpython-interpreter.org

ipython notebook is super-hip among bioinformaticists–
the lab notebook, re-imagined for the computational among us:
http://ipython.org/ipython-doc/dev/interactive/htmlnotebook.html



Python 2 or 3?

Major, backwards-incompatible changes were introduced with Python 3.

Cool stuff, like non-truncating division.

That was way back in 2008. Until recently, many of the extensions and external libraries are still trying to update everything to be compatible with 3.x.

It's coming soon. (Really– not just the eventual soon.)

As soon as Anaconda updates to Python 3, I will consider that my switching point.

Installing Python

Many OSs come with a default install of Python.

Installing the scientific packages is a pain in the neck, largely because of the low-level optimization libraries they are built on.

The best advice you will receive all day:

Unless you intend to be writing code for commercial applications, download the **Anaconda distribution** of Python, which comes with NumPy, SciPy, Pandas, Matplotlib, and many more crucial libraries, all in one easy binary. **Free!**

Trust me on this: https://store.continuum.io/cshop/anaconda/

An IDE?

I use Aptana, a slightly lighter-weight, Python/Ruby-focused version of the Eclipse platform plus the PyDev plugin:
http://www.aptana.com
http://www.eclipse.org
http://pydev.org

For simple, one-file editing I sometimes use TextMate:
http://macromates.com (Mac OS X, not free)

Sublime Text comes highly recommended:
http://www.sublimetext.com

But there are many, many IDEs out there– find one you like:
http://wiki.python.org/moin/IntegratedDevelopmentEnvironments

Being an Engineer in San Diego

Take advantage of the fact that we live in a city full of scientists and engineers! Not just on campus!

There are some great groups, events, and people out there:

- San Diego Python users group: http://www.meetup.com/pythonsd/
- Connect (Life Sciences/Tech events): http://www.connect.org
- SD Bioinformatics Forum: http://www.sdbioinfo.org
- SD BioTechnology Network: http://sdbn.org
- SD Software Industries Council: http://www.sdsic.org/
- SD Biotechnology Discussion Group: http://www.sd-biotech.org

... etc.

If you're interested but timid, let me know, and we can coordinate.

Resources

Python

- http://www.python.org
- http://docs.python.org
- http://mail.python.org/
- O'Reilly books: http://oreilly.com/python/
- http://stackoverflow.com/questions/tagged/python
- Google's class: http://code.google.com/edu/languages/google-python-class/
- Codecademy's courses: http://www.codecademy.com/tracks/python
- Coursera's course: https://www.coursera.org/course/interactivepython
- PyCon and other videos: http://pyvideo.org
- "learn python" or "python tutorials" or "python [whatever your question is]"

Resources

NumPy and SciPy

- http://www.scipy.org
- http://docs.scipy.org/
- http://scipy.org/Mailing_Lists
- http://www.scipy.org/NumPy_for_Matlab_Users
- http://stackoverflow.com/questions/tagged/numpy
- http://stackoverflow.com/questions/tagged/scipy
- PyData videos: http://pyvideo.org/category/18/pydata-2012
- SciPy conference videos: http://pyvideo.org/category/20/scipy_2012
- Travis Oliphant created SciPy: https://twitter.com/teoliphant

Resources

Pandas

- http://pandas.pydata.org
- Python for Data Analysis: http://shop.oreilly.com/product/0636920023784.do
- Wes McKinney maintains Pandas: http://blog.wesmckinney.com
- http://twitter.com/wesmckinn

Other

- Enthought: http://www.enthought.com/products/getepd.php
- ipython: http://ipython.org
- Fernando Perez maintains iP-notebook: http://fperez.org
- https://twitter.com/fperez_org
- C. Titus Brown writes about Python & Bioinfo: http://ivory.idyll.org/blog/

THANKS!