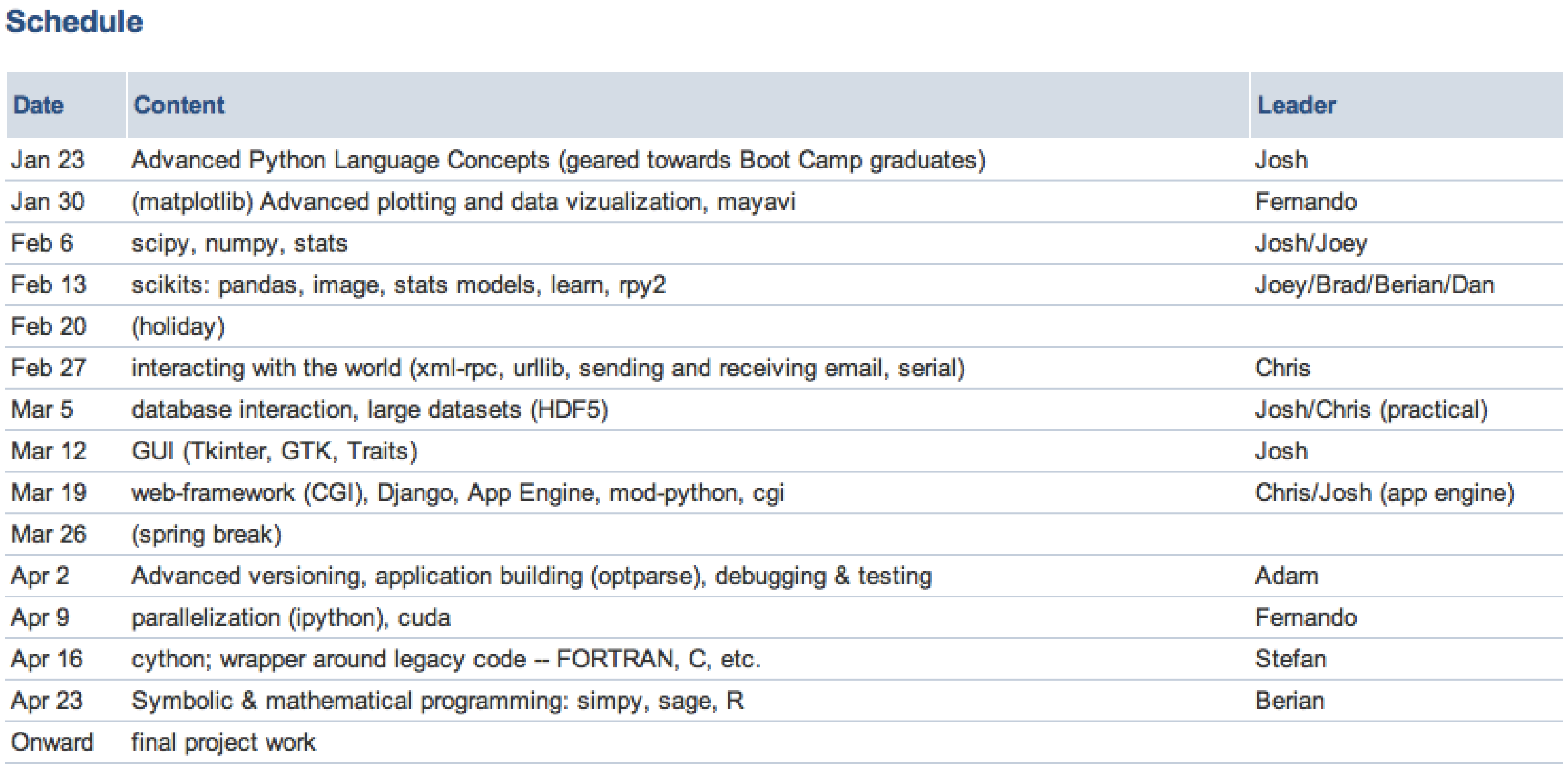
Python Computing for Scientific Research: Monday 2-5pm (219 DWINELLE; AY 250; 06072)



Schedule

Date	Content	Leader
Jan 23	Advanced Python Language Concepts (geared towards Boot Camp graduates)	Josh
Jan 30	(matplotlib) Advanced plotting and data vizualization, mayavi	Fernando
Feb 6	scipy, numpy, stats	Josh/Joey
Feb 13	scikits: pandas, image, stats models, learn, rpy2	Joey/Brad/Berian/Dan
Feb 20	(holiday)	
Feb 27	interacting with the world (xml-rpc, urllib, sending and receiving email, serial)	Chris
Mar 5	database interaction, large datasets (HDF5)	Josh/Chris (practical)
Mar 12	GUI (Tkinter, GTK, Traits)	Josh
Mar 19	web-framework (CGI), Django, App Engine, mod-python, cgi	Chris/Josh (app engine)
Mar 26	(spring break)	
Apr 2	Advanced versioning, application building (optparse), debugging & testing	Adam
Apr 9	parallelization (ipython), cuda	Fernando
Apr 16	cython; wrapper around legacy code -- FORTRAN, C, etc.	Stefan
Apr 23	Symbolic & mathematical programming: simpy, sage, R	Berian
Onward	final project work	

preliminary schedule

Advanced Strings & File I/O

8:15 - 8:30 homework review (optional)

8:30 - 9:30 Advanced Strings & File IO
- string methods + formatting
- regex
- read/write (writelines)
- subprocess
- StringIO

9:30-10:10 breakout
10:10 - 11:10 Advanced Stuff
- lambda functions
- filter, map, reduce, zip
- try/except/finally
- exec, eval

11:10 - 11:40 breakout
11:40 - 12:20 ipython/notebook

12:20 - 1 pm lunch
1 - 2:10 Object oriented programming
- classes
- methods
- instances

2:10 -2:40 breakout coffee
2:40 - 4:00 OOP (II)
- special methods (init, del, str, ...)
- with
- exception classes
- sub-classing and inheritance
- yield

4:00 - start homework

Strings can do operations on themselves:

`.lowercase(), .uppercase(),.capitalize()`

```
>>> "funKY tOwn".capitalize()
'Funky town'
>>> "funky tOwn".lowercase()
'funky town'
```

`.split([sep [,maxsplit]])`

```
>>> "funKY tOwn".split()
['funKY', 'tOwn']
>>> "funKY tOwn".capitalize().split()
['Funky', 'town']
>>> [x.capitalize() for x in "funKY tOwn".split()]
['Funky', 'Town']
>>> "I want to take you to, funKY tOwn".split("u")
['I want to take yo', ' to, f', 'nKY tOwn']
>>> "I want to take you to, funKY tOwn".split("you")
['I want to take ', ' to, funKY tOwn']
```

.strip(), .join(), .replace()

```
>>> csv_string = 'Dog,Cat,Spam,Defenestrate,1, 3.1415   \n\t'
>>> csv_string.strip()
'Dog,Cat,Spam,Defenestrate,1, 3.1415'
>>> clean_list = [x.strip() for x in csv_string.split(",")]
>>> clean_list
['Dog', 'Cat', 'Spam', 'Defenestrate', '1', '3.1415']
```

`.join()` allows you to glue a list of strings together with a certain string

```
>>> print ",".join(clean_list)
'Dog,Cat,Spam,Defenestrate,1,3.1415'
>>> print "\t".join(clean_list)
Dog  Cat  Spam Defenestrate 1   3.1415
```

`.replace()` strings in strings

```
>>> csv_string = 'Dog,Cat,Spam,Defenestrate,1, 3.1415   \n\t'
>>> alt_csv = csv_string.strip().replace(' ','')
>>> alt_csv
'Dog,Cat,Spam,Defenestrate,1,3.1415'
>>> print csv_string.strip().replace(' ','').replace(',','\t')
Dog  Cat  Spam Defenestrate 1   3.1415
```

.find()

incredibly useful searching, returning the index of the search

```
>>> s = 'My Funny Valentine'
>>> s.find("y")
1
>>> s.find("y",2)
7
>>> s[s.find("Funny"):]
'Funny Valentine'
>>> s.find("z")
-1
>>> ss = [s,"Argentine","American","Quarentine"]
>>> for thestring in ss:
      if thestring.find("tine") != -1:
         print "'" + str(thestring) + "' contains 'tine'."

'My Funny Valentine' contains 'tine'.
'Argentine' contains 'tine'.
'Quarentine' contains 'tine'.
>>>
```

`string` module

exposes useful variables and functions

```
>>> import string
>>> string.swapcase("fUNKY tOWN")
'Funky Town'
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.digits
'0123456789'
```

file: checkemail.py

```python
import string
## let's only allow .com, .edu, and .org email domains
allowed_domains = ["com","edu","org"]
## let's nix all the possible bad characters
disallowed = string.punctuation.replace(".","")
while True:
    res = raw_input("Enter your full email address: ")
    res = res.strip()   # get rid of extra spaces from a key-happy user
    if res.count("@") != 1:
        print "missing @ sign or too many @ signs"
        continue
    username,domain = res.split("@")

    ## let's look at the domain
    if domain.find(".") == -1:
        print "invalid domain name"
        continue
    if domain.split(".")[-1] not in allowed_domains:
        ## does this end as it should?
        print "invalid top-level domain...must be in " + ",".join(allowed_domains)
        continue
    goodtogo = True
    for s in domain:
        if s in disallowed:
            print "invalid character " + s
            ## cannot use continue here because then we only continue the for loop, not the while loop
            goodtogo = False

    ## if we're here then we're good on domain. Make sure that
    for s in username:
        if s in disallowed:
            print "invalid character " + s
            goodtogo = False

    if goodtogo:
        print "valid email. Thank you."
        break
```

example: check email address

```
BootCamp> python checkemail.py
Enter your full email address: josh.python.org
missing @ sign or too many @ signs
Enter your full email address: josh@pythonorg
invalid domain name
Enter your full email address: joshrocks!@python,.org
invalid character ,
invalid character !
Enter your full email address: joshrocks@python.org
valid email. Thank you.
BootCamp>
```

String Formatting

casting using `str()` is very limited
Python gives access to C-like string formatting

usage: “%(format)” % (variable)

```
>>> print "My favorite integer is %i and my favorite float is %f,\n" \
    " which to three decimal places is %.3f and in exponential form is %e" \
     % (3,math.pi,math.pi,math.pi)
My favorite integer is 3 and my favorite float is 3.141593,
 which to three decimal places is 3.142 and in exponential form is 3.141593e+00
```

common formats:
f (float), i (integer), s (string), g (nicely formatting floats)

http://docs.python.org/release/2.7.2/library/stdtypes.html#string-formatting-operations

String Formatting

% escapes “%”

```
>>> print "I promise to give 100%% effort whenever asked of %s." % ("me")
I promise to give 100% effort whenever asked of me.
```

+ and zero-padding

```
>>> print "%f\n%+f\n%+f\n%010f\n%10s" %
(math.pi,math.pi,-1.0*math.pi,math.pi,"pi")
3.141593
+3.141593
-3.141593
003.141593
        pi
```

String Formatting

the (new) preferred way

is *string*.format(value0,value1,....)

```
>>> 'on {0}, I feel {1}'.format("saturday","groovy")
'on saturday, I feel groovy'
>>> 'on {}, I feel {}'.format("saturday","groovy")
'on saturday, I feel groovy'
>>> 'on {0}, I feel {1}'.format(["saturday","groovy"])
IndexError: tuple index out of range
>>> 'on {0}, I feel {0}'.format(["saturday","groovy"])
"on ['saturday', 'groovy'], I feel ['saturday', 'groovy']"
>>> 'on {0}, I feel {0}'.format("saturday","groovy")
'on saturday, I feel saturday'
```

you can assign by argument position

```
>>> '{desire} to {place}'.format(desire='Fly me',place='The Moon')
'Fly me to The Moon'
>>> '{desire} to {place} or else I wont visit {place}.'.format(desire='Fly
me',place='The Moon')
'Fly me to The Moon or else I wont visit The Moon.'
>>> f = {"desire": "I want to take you", "place": "funky town"}
>>> '{desire} to {place}'.format(**f)
'I want to take you to funky town'
```

or by name

Formatting comes after a colon (:)

```
>>> ("%03.2f" % 3.14159) ==  "{:03.2f}".format(3.14159)
>>> "{0:03.2f}".format(3.14159,42)
'3.14'
>>> "{1:03.2f}".format(3.14159,42)
'42.00'
>>> # format also supports binary numbers
>>> "int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".format(42)
'int: 42;  hex: 2a;  oct: 52;  bin: 101010'
```

```
format_spec ::=  [[fill]align][sign][#][0][width][,][.precision][type]
fill        ::=  <a character other than '}'>
align       ::=  "<" | ">" | "=" | "^"
sign        ::=  "+" | "-" | " "
width       ::=  integer
precision   ::=  integer
type        ::=  "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X"
```

```
>>> "{:*^11}".format(" meh ")
'*** meh ***'
>>> "{:*<11}".format(" meh ")
' meh ******'
>>> "{:*>11}".format(" meh ")
'****** meh '
>>> "{:>11.2}".format(3.1415)
'        3.1'
```

Regular Expressions

complex string that defines search

`import re`

```
import re
>>> emailsearch = re.compile(r'[\w\-][\w\-\.]+@[\w\-][\w\-\.]+[a-zA-Z]{1,4}')
>>> emailsearch.findall("jbloom@python.org")
['jbloom@python.org']
>>> emailsearch.findall("jbloom@python!org")
[]
```

FYI...

```
>>> visacard   = re.compile("4\d{3}[\s-]?\d{4}[\s-]?\d{4}[\s-]?\d{4}")
>>> mastercard= re.compile("5[1-5]\d{2}[\s-]?\d{4}[\s-]?\d{4}[\s-]?\d{4}")
```

http://diveintopython.org/regular_expressions

File I/O (read/write)

`.open()` and `.close()` are builtin functions

```
>> file_stream = open(“mydata.dat”,”r”)
>> <type 'file'>
>> file_stream.close()
```

open modes: “r” *(read)*, “w” *(write)*, “r+” *(read + update)*, “rb” *(read as a binary stream, ...)*

Writing data: `.write()` or `.writelines()`

```
>>> f= open("test.dat","w")
>>> f.write("This is my first file I/O. Zing!")
>>> f.close()
>>> import os ; os.system("cat %s" % "test.dat")
This is my first file I/O. Zing!0
```

```
>>> f= open("test.dat","w")
>>> f.writelines(["This is my first file I/O.\n","Take that Dr. Zing!\n"])
>>> f.close() ; os.system("cat %s" % "test.dat")
This is my first file I/O.
Take that Dr. Zing!
0
```

Likewise, there is `.readlines()` and `.read()`

```
>>> f= open("test.dat","r")
>>> data = f.readlines()
>>> f.close() ; print data
This is my first file I/O.
Take that Dr. Zing!
>>>
```

file: tabbify_my_csv.py

```python
"""
small copy program that turns a csv file into a tabbed file
"""

import os

def tabbify(infilename,outfilename,ignore_comments=True,comment_chars="#;/"):
    """
INPUT: infilename
OUTPUT: creates a file called outfilename
    """
    if not os.path.exists(infilename):
        return  # do nothing if the file isn't there
    f = open(infilename,"r")
    o = open(outfilename,"w")
    inlines = f.readlines() ; f.close()
    outlines = []
    for l in inlines:
        if ignore_comments and (l[0] in comment_chars):
            outlines.append(l)
        else:
            outlines.append(l.replace(",","\t"))
    o.writelines(outlines) ; o.close()
```

```
BootCamp> cat google_share_price.csv
# Date,Open,High,Low,Close,Volume,Adj Close
2008-10-14,393.53,394.50,357.00,362.71,7784800,362.71
...
BootCamp> cat google_share_price.tab
# Date,Open,High,Low,Close,Volume,Adj Close
2008-10-14      393.53  394.50  357.00  362.71  7784800 362.71
....
```

File I/O (read/write)

`shutil` module is preferred for copying, archiving & removing files/directories

http://docs.python.org/library/shutil.html#module-shutil

`tempfile` module is used for the creation of temporary directories and files

```
>>> import tempfile
>>> tmp = tempfile.TemporaryFile() ; type(tmp)
<type 'file'>
>>> tmp = tempfile.NamedTemporaryFile(suffix=".csv",\
                                      prefix="boot",dir="/tmp",delete=False)
>>> tmp.name
'/tmp/bootG2zoE8.csv'
>>> tmp.write("# stock phrases of today's youth\nWassup?!,OMG,LOL,BRB,Python\n")
>>> tmp.close() ; import os ; os.system("cat %s" % tmp.name)
# stock phrases of today's youth
Wassup?!,OMG,LOL,BRB,Python
0
```

http://www.doughellmann.com/PyMOTW/tempfile/

StringIO module

handy for making file-like objects out of strings

```
>>> import StringIO
>>> myfile = StringIO.StringIO( \
             "# stock phrases of today's youth\nWassup?!,OMG,LOL,BRB,Python\n")
>>> myfile.getvalue()  ## get what we just wrote
"# stock phrases of today's youth\nWassup?!,OMG,LOL,BRB,Python\n"
>>> myfile.seek(0)     ## go back to the beginning
>>> myfile.readlines()
["# stock phrases of today's youth\n", 'Wassup?!,OMG,LOL,BRB,Python\n']
>>> myfile.close()
>>> myfile.write(“not gonna happen”)
ValueError: I/O operation on closed file
>>> myfile = StringIO.StringIO("# stock phrases of today's youth
\nWassup?!,OMG,LOL,BRB,Python\n")
>>> myfile.seek(2)  ; myfile.write("silly") ; myfile.seek(0)
>>> myfile.readlines()
["# silly phrases of today's youth\n", 'Wassup?!,OMG,LOL,BRB,Python\n']
```

(`cStringIO` is actually faster but doesn’t work on some platforms)

subprocess module

subprocess is the preferred way to interact with other programs, as you might do on the command line

```
>>> from subprocess import *
>>> p = Popen("ls", shell=True, stdout=PIPE)  # list the directory
>>> p.pid  # get the process ID of the new subprocess
12121
>>> print p.stdout.readlines()
['Archive.zip\n', 'Day1BreakoutSolutions\n', 'Day1Files\n', 'LecturePDFs\n',
'Object_Oriented_I.key\n',...]
>>> p = Popen("spamalot", shell=True, stdout=PIPE,stderr=PIPE)
>>> print p.stderr.readlines()
['/bin/sh: spamalot: command not found\n']
```

it's often advisable to wait until the subprocess has finished

```
>>> # this returns immediately
>>> p = Popen("find .. -name '*.py'", shell=True, stdout=PIPE,stderr=PIPE)
>>> os.waitpid(p.pid, 0)  ## this will block until the search is done
['../py4science/examples/pyrex/trailstats/setup.py\n',
 '../py4science/examples/qsort.py\n',
 '../py4science/examples/quad_newton.py\n']
```

http://docs.python.org/library/subprocess.html

Breakout Work

build a command-line utility file which copies the input file to another file and:

1. reverses the ending of the file name
   e.g. *josh.dat* is copied to *josh.tad*

2. deletes every other line

3. changes every occurrence of the words:
   love → hate, not → is, is → not

4. sets every number to half its original value
   e.g. *I like 3.14 and you like 2*
   → I like 1.57 and you like 1

5. count the number of words "astrology" and "physics"

try it on the file *elie.info*