Python Grab-bag

- Lambda functions
- Map, reduce, zip
- Alternative string formats
- Other File I/O and StringIO
- Making a string executable

Lambda Functions

(anonymous functions)

from Lisp & functional programming

```
>>> tmp = lambda x: x**2
>>> type(tmp)
<type 'function'>
>>> tmp(2)
4
>>> (lambda x,y: x**2+y)(2,4.5) # forget about creating a new function name...just do it!
8.5
>>> ## create a list of lambda functions
>>> lamfun = [lambda x: x**2, lambda x: x**3, \
             lambda y: math.sqrt(y) if y >= 0 else "Really? I mean really? %f" % y]
>>> for l in lamfun: print l(-1.3)
1.69
-2.197
Really? I mean really? -1.300000
>>>
```

lambda functions are meant to be short, one liners. If you need more complex functions, probably better just to name them

map, reduce, zip

Map is just another way to do list comprehension

`map(`*function*`, `*sequence*`)` calls *function*`(`*item*`)` for each of the sequence's items and returns a list of the return values

```
>>> def cube_it(x): return x**3
>>> map(cube_it,xrange(1,10))
[1, 8, 27, 64, 125, 216, 343, 512, 729]
>>> map(lambda x: x**3, xrange(1,10))
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Reduce returns one value

`reduce(`*function*`, `*sequence*`)` returns a single value constructed by calling the binary function *function* on the first two items of the sequence, then on the result and the next item, and so on

```
>>> reduce(lambda x,y: x + y, xrange(1,11))   # sum from 1 to 10
55
>>> %timeit reduce(lambda x,y: x + y, xrange(1,11))
100000 loops, best of 3: 2.07 us per loop
>>> %timeit sum(xrange(1,11))    # sum() is a built in function...it’s bound to be faster
1000000 loops, best of 3: 647 ns per loop
```

map, reduce, zip

zip()

built in function to pairwise concatenate items in iterables into a list of tuples

```
>>> zip(["I","you","them"],["=spam","=eggs","=dark knights"])
[('I', '=spam'), ('you', '=eggs'), ('them', '=dark knights')]
>>> zip(["I","you","them"],["=spam","=eggs","=dark knights"],["!","?","#"])
[('I', '=spam', '!'), ('you', '=eggs', '?'), ('them', '=dark knights', '#')]
>>> zip(["I","you","them"],["=spam","=eggs","=dark knights"],["!","?"])
[('I', '=spam', '!'), ('you', '=eggs', '?')]
>>>
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print 'What is your %s?  It is %s.' % (q, a)
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

not to be confused with zipfile module which exposes file compression

Alternative 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)
True
>>> "{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'
```

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)

Making a Script Executable

When a script/module is run from the command line, a special variable called `__name__` is set to `"__main__"`

```
# all your module stuff here

# at the bottom stick...
if __name__ == "__main__":
    """only executed if this module is called from the command line"""
    print "I was called from the command line!"
```

On the first line of a script, say what to run the script with (as with Perl):

```
#!/usr/bin/env python
"""doctring for this module"""
# all your module stuff here
```

set execute permissions of that script

```
BootCamp> chmod a+x script_name.py  ## this works in UNIX, Mac OSX
BootCamp> ./script_name.py
I was called from the command line!
```

file: modfun.py

```python
#!/usr/bin/env python
"""
Some functions written to demonstrate a bunch of concepts like modules, import
and command-line programming
"""
import os
import sys

def getinfo(path=".",show_version=True):
    """
Purpose: make simple us of os and sys modules
Input: path (default = "."), the directory you want to list
    """
    if show_version:
        print "-" * 40
        print "You are using Python version ",
        print sys.version
        print "-" * 40

    print "Files in the directory " + str(os.path.abspath(path)) + ":"
    for f in os.listdir(path): print "   " + f
    print "*" * 40

if __name__ == "__main__":
    """
Executed only if run from the command line.
call with
  modfun.py <dirname> <dirname> ...
If no dirname is given then list the files in the current path
    """
    if len(sys.argv) == 1:
        getinfo(".",show_version=True)
    else:
        for i,dir in enumerate(sys.argv[1:]):
            if os.path.isdir(dir):
                # if we have a directory then operate on it
                # only show the version info if it's the first directory
                getinfo(dir,show_version=(i==0))
            else:
                print "Directory: " + str(dir) + " does not exist."
```

```
BootCamp> ./modfun.py
----------------------------------------
You are using Python version  2.7.2 |EPD 7.2-2 (32-bit)| (r265:79063, Jan 11 2012, 15:13:03)
[GCC 4.0.1 (Apple Inc. build 5488)]
----------------------------------------
Files in the directory /Users/jbloom/Classes/BootCamp:
  basic training.key
  data structures.key
  modfun.html
  modfun.py
  modfun.pyc
   ...
****************************************
BootCamp> ./modfun.py . MySpamDir /tmp/
----------------------------------------
You are using Python version  2.7.2 |EPD 6.2-2 (32-bit)| (r265:79063, Jan 11 2012, 15:13:03)
[GCC 4.0.1 (Apple Inc. build 5488)]
----------------------------------------
Files in the directory /Users/jbloom/Classes/BootCamp:
  basic training.key
  data structures.key
  modfun.html
  modfun.py
  modfun.pyc
  modfun.py~
  modules_def_io.key
 ...
****************************************
Directory: MySpamDir does not exist.
****************************************
Files in the directory /tmp:
  .font-unix
  .ICE-unix
  .X0-lock
  .X11-unix
  dao.param
 ...
****************************************
BootCamp>
```

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

try it on the file *elie.info*