CME 193: Introduction to Scientific Python
Lecture 3: File I/O, Object-oriented Python, and Intro to NumPy

Austin Benson

Dan Frank

Institute for Computational and Mathematical Engineering (ICME)

April 11, 2013

Homework

- Overall, good job on homework 1.
- Homework 2 is due right now.
- Homework 3 is posted (longer than the first two assignments).

importing SciPy

```
# Example from:
# http://www.scipy.org/scipy_Example_List

# incorrect
from scipy import *

# correct
from scipy import integrate

value, err = integrate.quad(func=pow, a=0.,
                            b=1., args=(5,))
value  # integral of x^5 over [0,1]
```

Programming Languages and the system

- Matlab encourages a separation between the program and the computer system
- C provides powerful ways to control the computer system, but the code is verbose
- Python makes it very easy to interact with the computer system in basic ways

Python and the system

- For scientific computing, this balance between power and ease of programming makes Python a popular choice
- Today, we will focus on file I/O as our interaction with the computer system
- Please check out the `os` library (operating system): `http://docs.python.org/2/library/os.html`.

File reading

Suppose we have a text file of chemical compounds:

```
salt: NaCl
sugar: C6H1206
ethanol: CH3CH2OH
ammonia: NH3
```

We want to read this file and store the information in a dictionary.

File reading

```
f = open('compounds.txt', 'r')
```

'r' specifies that we want to read the file.

f is now a file object

File reading

Print the entire contents of the file:

```
f = open('compounds.txt', 'r')
contents = f.read()
print contents
```

File reading

Print each line individually:

```
f = open('compounds.txt', 'r')
for i, line in enumerate(f):
    print '(Line #' + str(i + 1) + ') ' + line
```

Prints "(Line #1) salt: NaCl", etc.

File reading

A verbose dictionary formulation:

```
f = open('compounds.txt', 'r')
compounds = {}
for line in f:
    split_line = line.split(':')
    name = split_line[0]
    formula = split_line[1]
    formula = formula.strip()
    compounds[name] = formula

f.close()
```

File reading

The "with" statement closes the file automatically. A more Pythonic implementation:

```
compounds = {}
with open('compounds.txt', 'r') as f:
    for line in f:
        compounds[line.split(':')[0]] = \
          line.split(':')[1].strip()
```

File reading

One liner:

```
compounds = dict([(line.split(':')[0],
                   line.split(':')[1].strip()) \
                   for line in open('compounds.txt',
                                    'r')])
```

... might be a bit much for one line.

File writing

Now we are going the other way. We have data in our Python program that we want to store.

File writing

Suppose we have scraped some web data from `www.reddit.com`.

File writing

reddit_data.py:

```
d1 = {'title': "The eyes says it all",
      'sub': 'aww', 'comments': 595}

d2 = {'title': "From typical youtube upload " + \
               "to serendipity in 30 seconds",
      'sub': 'AskReddit', 'comments': 6494}

d3 = {'title': "Use a decent host or don't " + \
               "even try at all...",
      'sub': 'AdviceAnimals', 'comments': 95}

data = [d1, d2, d3]
```

File writing

```
from reddit_data import *

with open('reddit1.txt', 'w') as f:
    for point in data:
        f.write(str(point) + '\n')
```

Note that we request write permission with ‘w’ when opening the file.

File writing

Result:

```
{'sub': 'aww', 'comments': 595, 'title': 'The eyes says it
{'sub': 'AskReddit', 'comments': 6494, 'title': 'From typi
{'sub': 'AdviceAnimals', 'comments': 95, 'title': "Use a de
```

any problems with this output?

File writing

We might want a little more structure to the output file:

```
from reddit_data import *

with open('reddit2.txt', 'w') as f:
  for i, point in enumerate(data):
    data = 'Post #%d\n' % i
    data += '\t%s\n' % point['title'][0:20]
    data += '\t\t%s (%d)\n' % (point['sub'],
                               point['comments'])
    f.write(data)
```

File writing

Result:

```
Post #0
        The eyes says it all
                aww (595)
Post #1
        From typical youtube
                AskReddit (6494)
Post #2
        Use a decent host or
                AdviceAnimals (95)
```

Classes

Classes:

- containers of data, information, and ideas
- basis for object-oriented programming

Python Classes

More specifically, Python classes:

- contain "instance variables" as data
- contain functions (sometimes called "methods" in the context of classes)
- structure can change on the fly

Differences from traditional OO

In languages like C++ and Java, classes provide data protection (public/private functions, friend classes, etc.). In Python, we just get the basics like inheritance.

It is up to the programmer to not abuse the classes. This works well in practice, and the code remains simple.

Stock Prices

```
class Stock():
    def __init__(self, name, symbol, prices=[]):
        self.name = name
        self.symbol = symbol
        self.prices = prices

google = Stock('Google', 'GOOG')
apple = Stock('Apple', 'APPL', [500.43, 570.60])

print google.symbol
print max(apple.prices)
```

Constructors

The `__init__()` function is the special class constructor. It is the function that gets called when we make the statement:

```
Stock('Google', 'GOOG').
```

self

The self parameter is a little weird.

The self variable is a reference to the class object that you are modifying. For example:

```
self.symbol = symbol.
```

says to modify the instance variable `symbol` in this class instantiation. On the right-hand-side, symbol is the name of a local variable (from the parameters).

Functions in classes

Classes can have functions:

```
class Stock():
    def __init__(self, name, symbol, prices=[]):
        self.name = name
        self.symbol = symbol
        self.prices = prices

    def high_price(self):
        if len(self.prices) is 0:
            return 'MISSING PRICES'
        return max(self.prices)

apple = Stock('Apple', 'APPL', [500.43, 570.60])
print apple.high_price()
```

Functions in classes

Notice how the `high_price()` function uses `self` to get the maximum price from that particular stock.

Glorified dictionaries?

If you think that classes are like dictionaries, you are right:

```
def Stock(name, symbol, prices=[]):
    def high_price(_self):
        if len(_self['prices']) is 0:
            return 'MISSING PRICES'
        return max(_self['prices'])
    s = {'name': name, 'symbol': symbol,
         'prices': prices}
    s['high_price'] = lambda(x): high_price(s)
    return s

apple = Stock('Apple', 'APPL', [500.43, 570.60])
print apple['high_price'](None)
```

Glorified dictionaries?

The dictionary version is messy, and classes are cleaner.

The subject of how to implement classes is material for a Programming Languages/Compilers course.

Inheritance

Inheritance is a way for classes to share structure.

A class can "inherit" the functions and data from a parent class.

Stock options

A stock option is like a stock. When purchasing a stock option, we purchase the "right to buy" the stock at a certain price at a certain time in the future.

We want to augment our Stock class with information about the option.

Stock options

```
from stocks2 import *
class StockOption(Stock):
  def __init__(self, name, symbol,
                opt_price, date, prices=[]):
    Stock.__init__(self, name, symbol, prices)
    self.opt_price = opt_price
    self.date_available = date

fb_opt = StockOption('Facebook', 'FB', 24.56,
                     'Mar. 1, 2013', [19.56, 20.13])
print fb_opt.high_price()
```

Stock options

The `high_price()` method in the `StockOption` class is inherited from the `Stock` class.

Alternatively, we could override the method.

Override

```
from stocks2 import *
class StockOption(Stock):
  def __init__(self, name, symbol,
               opt_price, date, prices=[]):
    Stock.__init__(self, name, symbol, prices)
    self.opt_price = opt_price
    self.date_available = date

  def high_price(self):
    if len(self.prices) is 0:
      return self.opt_price
    return max(self.opt_price, max(self.prices))
```

Python goodies

```
from stocks2 import *
class Portfolio():
  def __init__(self):
    google = Stock('Google', 'GOOG')
    facebook = Stock('Facebook', 'FB', [19.56])
    self.stocks = [google, facebook]

  def __contains__(self, key):
    for s in self.stocks:
      if key in [s.symbol, s.name]: return True
    return False

portfolio = Portfolio()
if 'FB' in portfolio: print 'I own Facebook stock!'
```

Python goodies

The __contains__() function is a special class function designed to work with the `in` operator.

There are other special class functions. For example, there is one for iterators (`for item in my_class`).

NumPy

```
import numpy as np

list_matrix = [[1, 3, 4], [2, 3, 5], [5, 7, 9]]
A = np.array(list_matrix)
b = np.array([4, 4, 4])

# Solve for Ax = b
x = np.linalg.solve(A, b)
```

NumPy

Linear classifier:

```
import numpy as np

def svm_classify(w, b, x):
    return np.dot(w, x) - b > 0

w = [-1.3, 4.555, 7]
b = 9.0
points = [[8.11, 3.42, 11.2], [-4.9, 4.557, 7.08]]
labels = [svm_classify(w, b, p) for p in points]
```

ndarrays

- At the core of the NumPy package, is the `ndarray` object which encapsulates n-dimensional arrays of homogeneous data.
- Many operations performed using `ndarray` objects execute in compiled code for performance
- The standard scientific packages use `ndarray`

ndarray creation

```
import numpy as np

normal_arr = [[1.2, 2.3], [-3.1, 4.77]]
ndarr = np.array(normal_arr)

ndarr.shape # (2, 2)
```

ndarray creation

```
import numpy as np

identity10 = np.eye(10)
ones4x2 = np.ones((4, 2))
```

Element access

```
import numpy as np

A = np.ones(4)
A[0, 0] += 2
A12 = A[1, 2]

first_row = A[0,:]
last_col = A[:,-1]
```

End

Assignment 3 is posted on the course web site (due Tuesday, April 16). **Longer** than homeworks 1 and 2.

Next time:

1. Dan is lecturing
2. More NumPy
3. SciPy