Introduction to Python and Webscraping

Robert Vesco

Yale School of Management

May 29, 2014

Class Objectives

- Programming is hard

`http://techcrunch.com/2014/05/24/dont-believe-anyone-who-tells-you-learning-to-code-is-easy/`

- Introduce basic python and webscraping
- Provide skills & knowledge not in online tutorials
- Tools that can be used with any programming language

Plan

- 9 - 9:15: Setup issues
- 9:15 - 9:30 Python in Scientific Computing
- 9:30 - 9:45 Anaconda & Spyder
- 9:45 - 10:30 Command line basics
- 10:30 - 12:00 Python Basics
- 12:00 - 12:30 Lunch
- 12:30 - 3:00 Python Webscraping
- 3:00 - 4:30 Practice with your own site
- 4:30 - 5:00 Other Tools, Development Environment

Abbreviated/Opinionated History of Programming Languages

- C, C++
- Awk, Sed & shell scripts
- Practical Extraction and Reporting (perl)
- S (R precursor)
- Java
- Ruby (perl 2.0)
- R
- Python
- Julia (R 2.0)

Python and Stats

Python and Jobs

Python Considerations

Support For

- Readability & Consistency (pythonic)
- Fairly fast
- Not Java
- Used in biz ops & domains

Support Against

- Backward compatibility
- Fragile package dependencies
- Fragmentation
- Complementary Assets for Science

The many faces and versions of Python

- Cython (main)
- IronPython (.net)
- PyPy (JIT)
- Jython (compiles to java)
- Ipython (scientific and interactive)

Version 2 vs 3

Interactive Python (IPYTHON)

- Designed for interactive work & scientists
- Lots of useful features
  - Tab completion
  - object?, object??
  - %run scriptname
  - press up shows last command
  - %who shows all variables
  - !cmd lets you run terminal commands
- Terminal friendly

Anaconda and Spydyer

- Anaconda is a pre-packaged python distribution for scientists
- Spyder is an IDE (Integrated Development Environment)
- Open a terminal or click spyder

```
anaconda/bin/spyder
```

- Open terminal within spyder

Why Terminals and Command Line Programs?

- Troubleshooting python programs
- Managing programs and files
- Right tool for some jobs

Shells vs Terminals

- Shells are programs (like python) that help you interact computer.
    - csh (c shell, mostly seen on older servers)
    - bash (most common)
    - zsh (most convenient)
- Terminals are wrappers around shells (iterm2 for macs)
- .bashrc, .cshrc, .zshrc are configuration files for shells

Paths

- One of the biggest causes of angst
- Exists at system and user levels
- Order matters; best set in configuration

```
#in bash, zsh
export PATH="$PATH:/usr/local/bin/python" and press Enter.
#in windows (dos)
path %path%;C:\Python
```

CD - Change Directory

```
pwd #your current path or %pwd

mkdir test_dir #create directory

ls -laG #Show all files in directory

cd test_dir #folder = directory

cd ../../ #move up two directories

cd - #move back to last directory

cd #move to home directory

cd ~/test_dir #move to folder relative to home directory

touch test_dir/test_file.txt

rmdir test_dir #must be empty, so fails

rm -rf test_dir #-rf = recursive and force -- dangerous
```

Open files in text editor

- Mac

```
open -t filename.ext #default editor for extension
open -a TextEdit filename.ext #forces textedit
#alias textedit='open -a TextEdit' For .bashrc
```

Find

http://www.tecmint.com/35-practical-examples-of-linux-find-command/

Finding programs and scripts

- Depends on operating system

```
where python
whereis python
which python
```

wget example

```
wget -r -H -l1 -erobots=off -nd -A 'pa02*.zip'
    http://www.google.com/googlebooks/uspto-patents-applications-text

# -r = recursive
# -H = to span domains, ie can leave blog ???
# -l1 = only to the depth of one
# -erobots=off = ignore robots.txt
# -nd = don't follow directory structure, just drop all files into
    folder
# -A 'pa01*.zip" = download only links with this regex

xargs -i wget
    'http://storage.googleapis.com/patents/grant_full_text/2012/{}'
    < list2012missing.txt
```

view source

Programming Concepts

- Types (int, strings)
- Data Structures
- Variables
- Flow structures
- Function, Objects and Modules
- Scripting and Programs

Hello World

Version 2 - Print Statement

```
print "hello world"
```

```
hello world
```

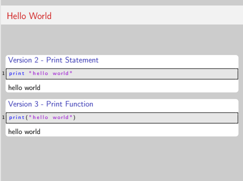
Version 3 - Print Function

```
print("hello world")
```

```
hello world
```

2014-05-29

Introduction to Python and Webscraping
Python
Basics
Hello World



Hello World

Version 2 - Print Statement

```
print "hello world"
```

```
hello world
```

Version 3 - Print Function

```
print("hello world")
```

```
hello world
```

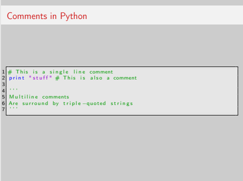
Note

- stuff and stuff

Comments in Python

```python
# This is a single line comment
print "stuff" # This is also a comment

'''
Multiline comments
Are surround by triple-quoted strings
'''
```



Comments in Python

```
# This is a single line comment
print "stuff" # This is also a comment

'''
Multiline comments
Are surround by triple−quoted strings
'''
```

Notes:

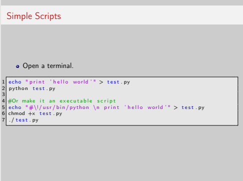
- stuff and stuff2

Simple Scripts

- Open a terminal.

```
echo "print 'hello world'" > test.py
python test.py

#Or make it an executable script
echo "#\!/usr/bin/python \n print 'hello world'" > test.py
chmod +x test.py
./test.py
```



Simple Scripts

- Open a terminal.

```
echo "print 'hello world'" > test.py
python test.py

#Or make it an executable script
echo "#\!/usr/bin/python \n print 'hello world'" > test.py
chmod +x test.py
./test.py
```

Notes:

Basic Types

- Numeric: int, float, long, complex
- Sequence: str, unicode, list, tuple, bytearray, buffer, xrange

```
var1 = "test strings"
var2 = 3
type(var1)
type(var2)
var3 = str(3) # conversion is possible, sometimes
type(var3)
```

```
<type 'str'>
<type 'int'>
<type 'str'>
```

Data Structures

- Often considered "types" or "compound types"
- Base python has
  - lists = ['apples',44, 'peaches']
  - tuples = read-only lists = ('apples',44,'peaches')
  - dictionaries = key:value pairs = {'firstname':'tom','lastname':'selleck'}

Lists: Slicing

- lists are flexible. They can be nested, shrunk, combined . . .
- Indexed starting with 0
- Limitation: searching for elements when you don't know index #

```
ls = [1,"a",2,"b", 1]
ls[0]
ls[0:2]
ls[:]
ls[1:]
ls[1:4:2] #last element in step. Easy way to get odd
```

```
1
[1, 'a']
[1, 'a', 2, 'b', 1]
['a', 2, 'b', 1]
['a', 'b']
```

Lists: Adding and Removing Elements

```
ls # pre
ls.append("add to end")
ls.insert(1,"after second element")
ls.insert(-1, "after second to last")
ls.remove('a') # by value, not index
ls # post
ls.index('b')
ls.count(1)
```

```
[1, 'a', 2, 'b', 1]
>>> >>> >>> >>> [1, 'after second element', 2, 'b', 1, 'after
    second to last', 'add to end']
3
2
```

Lists: Whole List Operations

```python
# Concatenate two lists
ls.extend(["newlist added to old"])
ls.sort()
ls
ls.reverse()
ls
```

```
[1, 1, 2, 'add to end', 'after second element', 'after second to last', 'b', 'newlist added to old']
['newlist added to old', 'b', 'after second to last', 'after second element', 'add to end', 2, 1, 1]
```

Lists: List Comprehensions

- Functions on list elements, like loops
- Not recommended for complex scenarios

```python
ls2 = [str(x) for x in ls]
ls2
## nested loop, += concat for strings
[[x+y for x in ls2] for y in ls2]
```

```
['1', 'a', '2', 'b', '1']
[['11', 'a1', '21', 'b1', '11'], ['1a', 'aa', '2a', 'ba', '1a'],
     ['12', 'a2', '22', 'b2', '12'], ['1b', 'ab', '2b', 'bb',
     '1b'], ['11', 'a1', '21', 'b1', '11']]
```

Sets

- Set are like lists, but must contain unique data and can't be nested
- Allows operations such a union and intersections

```
ls_dupes = [1,2,3,4,4,3]
st = set(ls_dupes)
print st
st2 = {1,2,3,5}
print st | st2 # union
print st & st2 # intersection
lss = list(st & st2) # convert back
```

```
>>> set([1, 2, 3, 4])
>>> set([1, 2, 3, 4, 5])
set([1, 2, 3])
>>> <type 'list'>
```

Tuples

- Tuples are like lists, but they are immutable
- Memory efficient because python knows how much memory to allocate

```
tp = () # empty tuple
tp1 = (1,) #tuple with one element (comma required)
tp2 = (1,2,3)
tp
tp1
tp2
tp2[2] #slicing uses [] not ()
```

```
()
(1,)
(1, 2, 3)
3
```

Dictionaries

- Represented by key:value pairs. Know as hashes, maps, associative collections
- Key can be numbers or strings, but must be unique.
- Value can be mutable or not, can be combined with tuples
- Useful when you need a fast lookup based on custom key.

```
dct = {'first':1, 'second':2, 'third':3}
dct['second']
del(dct['third'])
dct.keys()
dct.values()
```

```
2
['second', 'first']
[2, 1]
```

Operators

Control structures

Strings

Strings vs Numbers

```
string = "123456"
number = 123456
string is number
int(string) is number # different "objects"
int(string)==number # testing equality of value
```

```
False
False
True
```

Strings vs lists of strings

```
a = [string]
b = [string]
a == b # compares equality
a is b # compares whether objects
```

```
>>> T
```

Objects, Methods and Functions

- Methods are function that operate on objects
- Object: dog Method: eat
- Functions

http://stackoverflow.com/questions/8108688/in-python-when-should-i-use-a-function-instead-of-a-method

```
var1.capitalize() # method on object
len(var1) # also method, but functional looking
```

```
'Test strings'
12
```

Modules

Dates

Functions

- parameter order matters, unless name=paramater
- anonymous functions use lambda keyword
- return statements without value return nothing
- Variables within function have local scope

```
def printnum( x, y ):
    """This passes a parameter to the print statement"""
    print x, y
    return

printnum(y=3, x="printing this:")
printnum("positional ordering matter if not named", 4)
```

```
printing this: 3
positional ordering matter is not named 4
```

Files I/O

CSV files - Basic

```
echo -e "header1, header2\n1,2\n3,4" > test.csv
```

```
import csv
fl = list(csv.reader(open("test.csv")))
header, values = fl[0], fl[1:]
header
values
fl
```

```
['head1', 'head2']
[['1', '2'], ['3', '4']]
[['head1', 'head2'], ['1', '2'], ['3', '4']]
```

CSV files - Custom

```
class customcsv(csv.Dialect):
    lineterminator = '\n'
    delimiter = ','
    quoting = csv.QUOTE_NONE

fl.csv = csv.reader("test.csv", dialect=customcsv)
fl.csv
```

CSV files - Pandas - read$_{csv}$

```
import pandas as pd
# header=none if not in file
# or read_table + sep(delimeter)
fldf = pd.read_csv("test.csv")
type(fldf) #type is different
fldf
```

```
<class 'pandas.core.frame.DataFrame'>
   head1  head2
0      1      2
1      3      4

[2 rows x 2 columns]
```

CSV files - Pandas - More Options

- nrow=5 => read 5 rows
- na$_{rep}$='NULL' => set null to NULL else empty
- index=FALSE => no indices in output
- cols=['header1','header2'] => specify columns
- For all options:

http://pandas.pydata.org/pandas-docs/version/0.13.1/generated/pandas.io.parsers.read_csv.html

CSV files - Pandas - to$_{csv}$

- Many of the same options as read$_{csv}$

http://pandas.pydata.org/pandas-docs/version/0.13.1/generated/pandas.DataFrame.to_csv.html

```
import os #to see directory contents
fldf
fldf.to_csv("files/test_out.csv")
os.listdir('files')
```

```
head1  head2
0      1      2
1      3      4

[2 rows x 2 columns]
>>> ['test_out.csv']
```

Getting Help

- help(function) gets you the "docstring"

```
help(len)
```

```
Help on built-in function len in module __builtin__:

len(...)
    len(object) -> integer

    Return the number of items of a sequence or mapping.
```

Regular Expression

Expressions

Classes/Objects

Common Packages

Scientific

- Numpy: N-dimensional arrays, C integration, linear algebra
- SciPy: Numerical integration, optimization, depends on Numpy
- Matplotlib: 2d plotting
- Pandas: Approximates R/Stata, data cleaning, dataframes
- Statsmodels: For statistical models

Webscraping

- BeautifulSoup

HTML/XML/JSON

- HTML is an implementation of XML (a meta language)
- JavaScript Object Notation (JSON) is replacing xml for speed and readability (api)

Firebug

- Firebug is tool that allow you to inspect the elements of a webpage directly.

XPATH SQL for HTML/XML

- Xpath is a language that allows you to select "nodes" from xml
- Note: xpath 2.0 not implemented in all cases though many examples online
- Xpath 1.0 Tutorial

`http://www.zvon.org/comp/r/tut-XPath_1.html#Pages~List_of_XPat`

- Full reference

`http://www.w3.org/TR/xpath/`

JSON - Loading

```python
jsn = """
    {"name":"batman",
     "hobbies": ["fast cars", "fast planes", "spending money"],
    "buddy":"robin",
    "enemies": [{"name":"The Joker"},
                {"name":"The People of Gotham"}]
                }
"""
import json
#NOTE: loads for strings, load for files
rslt = json.loads(jsn) #put this into a form for python
print rslt
jsn_again = json.dumps(rslt) #back to json
```

```
{u'buddy': u'robin', u'enemies': [{u'name': u'The Joker'},
    {u'name': u'The People of Gotham'}], u'name': u'batman',
    u'hobbies': [u'fast cars', u'fast planes', u'spending money']}
```

JSON - Converting to DataFrames

```
enemies = pd.DataFrame(rslt['enemies'], columns=['name'])
enemies
```

```
        name
0                  The Joker
1                  The People of Gotham

[2 rows x 1 columns]
```

JSON - Example

```python
import json
import urllib2
import pprint import pprint
import pandas as pd

prefix="http://maps.googleapis.com/maps/api/geocode/json?address="
suffix="&sensor=false"
address="165%20Whitney%20Avenue,%20New%20Haven,%20CT"
url = prefix+address+suffix
j = urllib2.urlopen(url)
js = json.load(j)
type(js) #if in doubt, check type

#pprint(js)

#notice nested list, so use index to get into it
rstadd = js['results'][0]['address_components']

for rs in rstadd:
    print rs['short_name'], rs['types']

import pandas as pd
pd.DataFrame(rstadd)
```

view source

Regular Expressions (Regex)

- Regex came from perl, used to find text patterns
- To fragile for webscraping, but important complement

stuff
Stuff

Git

http://wildlyinaccurate.com/a-hackers-guide-to-git

Operators

Setting Up Your Development Environment

Top Aligned Blocks

Code

Cool Lots of Stuf
To talk
about

Result

pretty nice!

Beamer: Animated Bullets

- Trouble Shooting

Beamer: Animated Bullets

- Trouble Shooting
- A framework for thinking about programming

Beamer Columns

Stuff

- Truth is ephemeral

- What is right?
- What is Wrong?

setting python paths

```
:Setting environment variables (like PYTHONPATH)
:Create an emacs-lisp code block that looks like this:

:#+BEGIN_SRC emacs-lisp
:(setenv "PYTHONPATH" "/Users/neilsen/Development/obswatch-tru
:#+END_SRC
:Execute it, and it changes the environment accordingly.
:Note that you can also append to environment variables like th

:#+BEGIN_SRC emacs-lisp
:(setenv "PYTHONPATH" (concat (getenv "PYTHONPATH") ":" (geten
:#+END_SRC
:#+END_SRC
```

How to use virtualenv & pip

```
## run this on the command line
## assuming you are in your projects folder, create a new folder
mkdir projects1

cd projects1

## now create your virtualenv environment
## this will create a folder called "env".
## this will house a local version of python.
virtualenv env

## IMPORTANT.
## Now you need to activate your environment.
source env/bin/activate

## now you will be using a local version of python instead of your
## system's python

## to deactivate, simply type
deactivate
```

How to Share Ipython Notebooks

How to share your vagrant box

Testing Python Output

```python
a = ('b', 200)
b = ('x', 10)
c = ('q', -42)
return (a, b, c)
```

```
b   200
x    10
q   -42
```

Python Output

```python
a = ('b', 200)
b = ('x', 10)
c = ('q', -42)
return (a, b, c)
```

By removing the :exports both, you can export just the code and not the output. By replaceing it with :exports results, you can export the output without the source.

Using pip once virtualenv is activated

```
## again, these should be run on the command line.
## first, let's activate your virtual environment, if you haven't
## already
source env/bin/activate

## first, let's inspect what command are available in pip
pip help

## from this, we see that there are a number of commands we will
## find useful
pip list # this shows what programs are already installed
pip search numpy # this searches for packages named "numpy"
pip install numpy # this installs the numpy package.

## if you have many packages you want to install, you can
## create a requirements list
## this will create a file with a list of modules to install
## you can use your editor of choice to install this.
echo "numpy\nbeautifulsoup" > requirements.txt

## this will install all the packages in the text file.
## NOTE: you can specify the versions of module too. Sometimes
## this is important.
pip install -r requirements.txt

## now let's confirm that they installed correctly
pip list

## now if you are done with virtualenv remember to deactivate it
deactivate
```

Operators

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0