Introduction to Python and Webscraping

Robert Vesco

Yale School of Management
Initiative on Leadership and Organization (ILO)

May 30, 2014

Class Objectives

- Introduce basic python relevant to webscraping
- Provide skills & knowledge not in online tutorials

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

- Tools that can be used with any programming language
- Real time googling, SO and problem-solving
- Provide some guidance for your personal projects

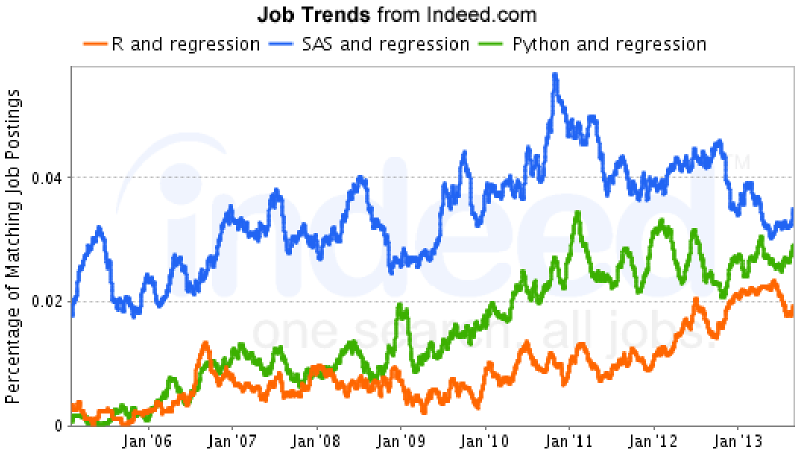
Plan

- Content
  - Why Python?
  - Working From the Command Line
  - Python
  - Webscraping
  - Discuss sites YOU want to scrape
  - Development environments
- Breaks
  - 10:30 (10 min)
  - 12:00 Lunch (30 min)
  - 1:30 (10 min)

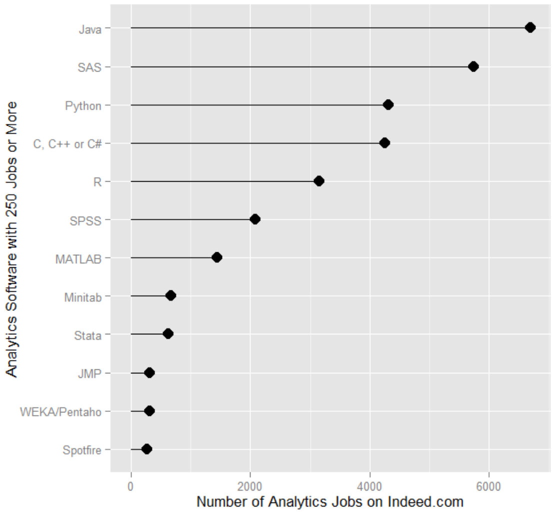
Opinionated Semi-History of Programming Languages

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

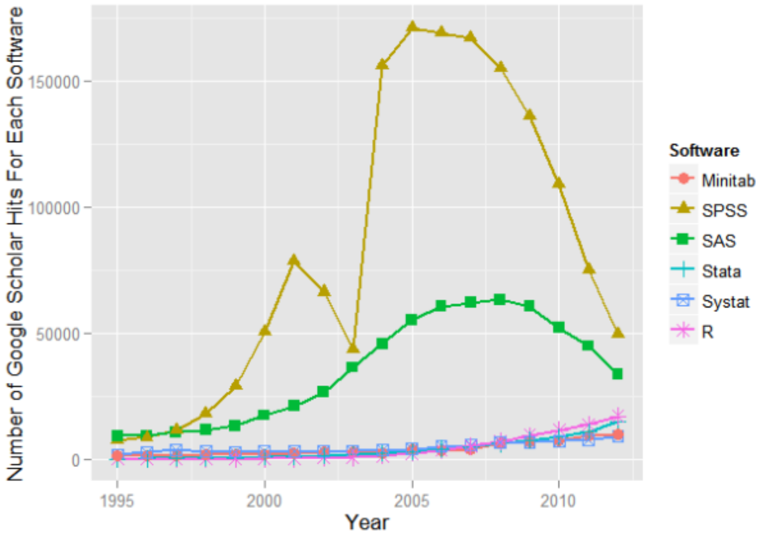
Python and Stats



Python and Stats



Stats and Journals



Homogenization of Programming

`http://www.talyarkoni.org/blog/2013/11/18/`
`the-homogenization-of-scientific-computing-or-why-python-is-st`

- TLDR: One tool for many problems

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 (python to c to python)
- IronPython (.net)
- PyPy (JIT)
- Jython (Java)
- Ipython (scientific and interactive)

Version 2 vs 3

Python 3 is killing Python

`https://medium.com/@deliciousrobots/5d2ad703365d/`

Python 3 can revive Python

`https://medium.com/p/2a7af4788b10`

- Python 3 is to Python what Windows 8 is to Windows
- Important: Python 3 broke compatibility

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

Why Terminals and Command Line Programs?

- Troubleshooting python programs
- Managing programs and files (very important for webscraping)
- Right tool for some jobs

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
```

- Windows

```
notepad filename.txt
```

- Terminal Viewer (useful for super large files)

```
less -SN filename.txt
```

Sudo, Elevated Rights, Admin

- Mac/Linux: sudo cmd file
- Windows: runas /user:admin
- Best to minimize programs running at elevated rights
- Modifying system files usually require this.

File Permissions

```
ls -laG #show all files and permissions
```

D = directory
4 = Read (r)
2 = Write (w)
1 = Execute(x)
777 = All rights for User, Group, Everyone <= BAD

- What is rwx-rw-r– in numerical permissions?
- When will sudo be needed?
- Scripts will often need execution rights

```
chmod +x filename
```

Finding programs and scripts

- Depends on operating system

```
where programname
which programname #will give unix style path on windows
whereis programname #not on all programs
```

Simple Scripts

- Scripts should begin with #!PathToYourExecutable

```
#non-standard script
echo "print 'hello world'" > test.py
cat test.py # shows contents
ls -laG #look at the file

python test.py

echo -e "#\!PATHTOYOURPYTHON \n print 'hello world'" > test.py
less test.py #spits it out to terminal to viewer
./test.py
```

- How can we make the second way work?

```
which env
# !#/path/to/env python #absolute path not needed
```

Find

- Flexibly file finder – very important for webscraping
- Criteria: permissions, size, date, users, file type, dir ...

```
find dir -options option \;

find . -maxdepth 1 -type f -name "*.py" -print -exec chmod +x {} \;
# . = current directory
# maxdepth = 1 directory down
# f = files only, not directories
# *.py = all .py file endings
# - exec = execute a command on found files
# {} variable container for found files
# need to be closed \; when using exec
```

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

Git

- Git is a code versioning tool, but used to redistribute software
- Github is a website that hosts repositories

```
# last part creates a directory with that name. Use . If already in dir of choice.
git clone git://github.com/rlvesco7/teaching_intro_python_web python_web
#or
git clone https://github.com/rlvesco7/teaching_intro_python_web python_web
```

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; read first > read second

```
#in bash, zsh

#in windows (dos)
path %path%;C:\Python #temp
# see control panel > environment variables for permanent
```

- Macs/linux

```
/etc/paths #admin levels for mac
/etc/environment #admin
~/.bashrc #user level for mac/linux
export PATH="$PATH:/usr/local/bin/python"
PATH=$PATH:/my/new/path #temporary
```

Terminal Configuration

http://www.tldp.org/LDP/abs/html/sample-bashrc.html

```
cd  # changes to your home directory
#open -t .bashrc #mac
#notepad .bashrc #windows
#gedit .bashrc #likely on linux
```

- Add:

```
alias la = "ls -lahG" #shortcut for listing files in directory
#export PATH="$HOME/python_web/wget/:$PATH" #for windows
```

- now you need to update your config file

```
. .bashrc
#or
source .bashrc
```

Wget

- Flexible, fast tool for downloading & spidering

```
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

unzip \* #to unzip a bunch of zip files
```

view source

Sed

- Stream Editor – operates line by line
- Cleaning text files

`http://sed.sourceforge.net/sed1line.txt`

```
sed '1d' #deletes first line of a text file
sed "s/$//" #convert unix to dos
sed 's/^/     /' #insert 5 blank spaces
# substitute "foo" with "bar" ONLY for lines which contain "baz"
sed '/baz/s/foo/bar/g'
```

Head, Tails, Less

```
head filename
head filename > newsmallfile
tail filename
less -SN filename #add line numbers and don't wrap
```

Grep & Chaining Commands & SDTOUT

- Grep is for finding text within files and standard output
- | "the pipe" is for chaining commands together
- > redirects stdout to file
- Can be combined with python scripts

```
grep -i "path" ~/.bashrc

la | grep org
cat pdfwget.txt | head | grep wget #grep is taking in stdout
```

- Exercise: Find how to integrate system command with your statistical system (sas, r, stata)

Anaconda and Spydyer

- Anaconda is a pre-packaged python distribution for scientists
- Spyder is an IDE (Integrated Development Environment)
- Open a terminal and type: spyder&

```
#cd ~/anaconda/bin/
#spyder
```

Programming Concepts

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

Getting Help

- http://stackoverflow.com/
- https://docs.python.org/2/tutorial/
- http://www.tutorialspoint.com/python/
- http://www.codecademy.com/tracks/python
- 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.
```

Hello World

Version 2 - Print Statement

```
print "hello world"
```

Version 3 - Print Function

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

hello world

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
'''
```

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')
  - sets = like lists but unique values only
  - 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

```
# 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

```
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]
```

Control structures

- Python assumes non-zero, non-null values are true
- zero, null = false
- if statement; if ... else ; nested ifs

```
var = 100
if ( var  == 100 ) : print "Value of expression is 100"
print "Good bye!"
# spacing , 4 spaces
if expression:
    statement(s)
else:
    statement(s)
```

Control Structures

- else if
- no switches or cases in python

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```

Loops

- For
- While

```
for num in range(10,20):  #to iterate between 10 to 20
    for i in range(2,num): #to iterate on the factors of the number
        if num%i == 0:       #to determine the first factor
            j=num/i          #to calculate the second factor
            print '%d equals %d * %d' % (num,i,j)
            break #to move to the next number, the #first FOR
    else:                    # else part of the loop
        print num, 'is a prime number'
```

```
...
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

Loops

- While loops are very useful for webscraping because you don't know ex ante when conditions end
- The dreaded infinite loop

```python
count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```

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
```

```
>>> True
False
```

Objects, Methods and Functions

- Methods are functions that operate on objects
- Object: dog Method: eat
- Classes of objects: felines => cats , lions
- Functions vs Objects/Methods

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
```

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
```

Modules

- For science: numpy, scipy, statsmodels, lxml, beautifulsoup, pandas . . .

```
import modulename
import modulename as shortname
import functionx from module name
```

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

- troublesome

```
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']
```

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 is a tool that allows you to inspect elements
- xpath plugin for firebug is useful . . . but only in firefox

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/`

XML - Loading

```
xml = """
    <root>
        <name type="superhero">Batman</name>
            <sidekick>Batty</sidekick>
        <contact type="email">riseup@batman.com</contact>
        <contact type="phone">555-1212</contact>
    </root>
            """

from lxml import objectify
root = objectify.fromstring(xml) #use parse from file

print root.tag
print root.text
print root.attrib

print root.name.tag
print root.name.text
print root.name.attrib

for con in root.contact:
    print con.text
    print con.attrib
```

view source

JSON - Loading

```
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 - 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

`http://www.rexegg.com/regex-quickstart.html`

- Let's discuss what you're interested in
- Key things
  - How to get pages
  - How to parse
  - How to overcome efforts to not let you scrape
- Anaconda is good, but it may have limitations
- Virtual Env
- Homebrew
- Babun
- Iterm2
- Virtual Machines/Servers

`http://www.codecademy.com/tracks/python`