

# 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)

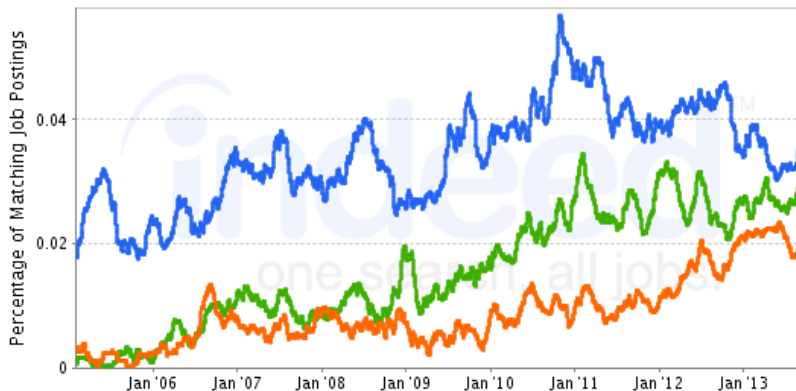
# Opinionated Semi-History of Programming Languages

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

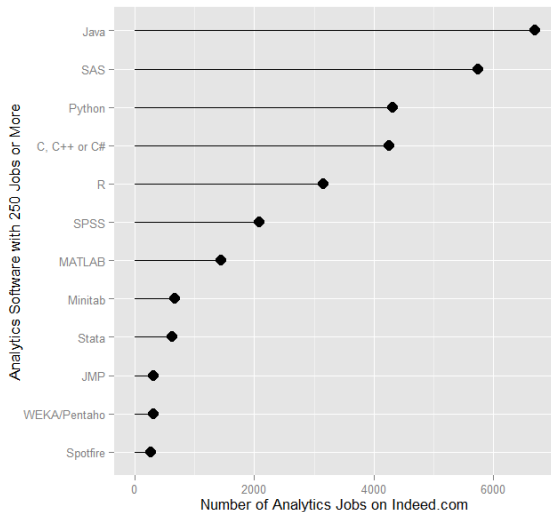
# Python and Stats

**Job Trends from Indeed.com**

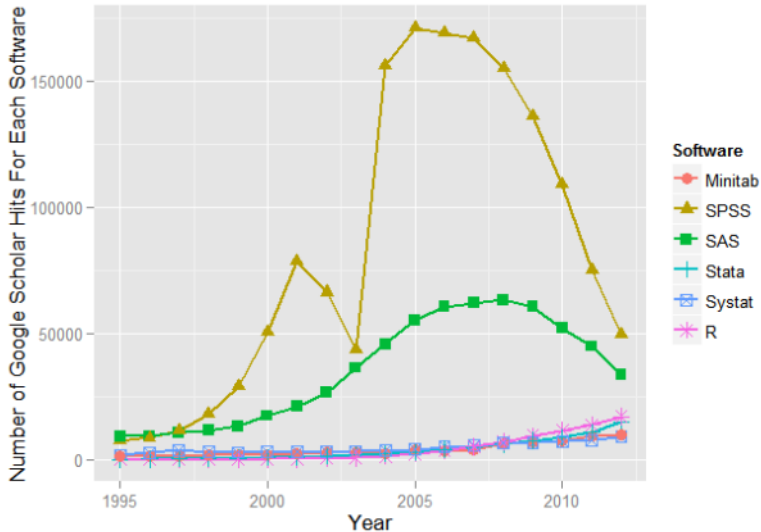
— R and regression — SAS and regression — Python and regression



# 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

```
1 pwd #your current path or %pwd
2
3 mkdir test_dir #create directory
4
5 ls -laG #Show all files in directory
6
7 cd test_dir #folder = directory
8
9 cd ../../ #move up two directories
10
11 cd - #move back to last directory
12
13 cd #move to home directory
14
15 cd ~/test_dir #move to folder relative to home directory
16
17 touch test_dir/test_file.txt
18
19 rmdir test_dir #must be empty, so fails
20
21 rm -rf test_dir #-rf = recursive and force — dangerous
```

# Open files in text editor

- Mac

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

- Windows

```
1 notepad filename.txt
```

- Terminal Viewer (useful for super large files)

```
1 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

```
1 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

```
1 chmod +x filename
```

# Finding programs and scripts

- Depends on operating system

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

# Simple Scripts

- Scripts should begin with `#!/PathToYourExecutable`

```
1 #non-standard script
2 echo "print 'hello world'" > test.py
3 cat test.py # shows contents
4 ls -laG #look at the file
5
6 python test.py
7
8 echo -e "#\!PATHTOYOURPYTHON \n print 'hello world'" > test.py
9 less test.py #spits it out to terminal to viewer
10 ./test.py
```

- How can we make the second way work?

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

# Find

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

```
1 find dir -options option \;  
2  
3 find . -maxdepth 1 -type f -name "*.py" -print -exec chmod +x {} \;  
4 # . = current directory  
5 # maxdepth = 1 directory down  
6 # f = files only, not directories  
7 # *.py = all .py file endings  
8 # - exec = execute a command on found files  
9 # {} variable container for found files  
10 # 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

```
1 # last part creates a directory with that name. Use . If already  
  # in dir of choice.  
2 git clone git://github.com/rlvesco7/teaching_intro_python_web  
  python_web  
3 #or  
4 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

```

1 #in bash, zsh
2
3 #in windows (dos)
4 path %path%;C:\Python #temp
5 # see control panel > environment variables for permanent

```

- Macs/linux

```

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

```

# Terminal Configuration

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

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

- Add:

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

- now you need to update your config file

```
1 . .bashrc
2 #or
3 source .bashrc
```



# Wget

- Flexible, fast tool for downloading & spidering

```
1 wget -r -H -l1 -erobots=off -nd -A 'pa02*.zip'
   http://www.google.com/googlebooks/uspto-patents-applications-text
2
3 # -r = recursive
4 # -H = to span domains, ie can leave blog ???
5 # -l1 = only to the depth of one
6 # -erobots=off = ignore robots.txt
7 # -nd = don't follow directory structure, just drop all files into
   folder
8 # -A 'pa01*.zip" = download only links with this regex
9
10 #xargs -i wget
   'http://storage.googleapis.com/patents/grant_full_text/2012/{}'
   < list2012missing.txt
11
12 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>

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

# Head, Tails, Less

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

# Grep & Chaining Commands & STDOUT

- 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

```
1 grep -i "path" ~/.bashrc
2
3 la | grep org
4 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 Spyder

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

```
1 #cd ~/anaconda/bin/  
2 #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"

```
1 help(len)
```

```
1 Help on built-in function len in module __builtin__:
2
3 len(...)
4     len(object) -> integer
5
6     Return the number of items of a sequence or mapping.
```

# Hello World

## Version 2 - Print Statement

```
1 print "hello world"
```

## Version 3 - Print Function

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

hello world



# Comments in Python

```
1 # This is a single line comment
2 print "stuff" # This is also a comment
3
4 '''
5 Multiline comments
6 Are surround by triple-quoted strings
7 '''
```

# Basic Types

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

```
1 var1 = "test strings"
2 var2 = 3
3 type(var1)
4 type(var2)
5 var3 = str(3) # conversion is possible , sometimes
6 type(var3)
```

```
1 <type 'str'>
2 <type 'int'>
3 <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 #

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

```
1 1
2 [1, 'a']
3 [1, 'a', 2, 'b', 1]
4 ['a', 2, 'b', 1]
5 ['a', 'b']
```

# Lists: Adding and Removing Elements

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

```
1 [1, 'a', 2, 'b', 1]
2 >>> >>> >>> >>> [1, 'after second element', 2, 'b', 1, 'after
   second to last', 'add to end']
3 3
4 2
```

# Lists: Whole List Operations

```
1 # Concatenate two lists
2 ls.extend(["newlist added to old"])
3 ls.sort()
4 ls
5 ls.reverse()
6 ls
```

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

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

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

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

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



# Tuples

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

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

```
1 ()
2 (1,)
3 (1, 2, 3)
4 3
```

# Dictionaries

- Represented by key:value pairs. Known 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.

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

```
1 2
2 ['second', 'first']
3 [2, 1]
```

# Control structures

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

```
1 var = 100
2 if ( var == 100 ) : print "Value of expression is 100"
3 print "Good bye!"
4 # spacing , 4 spaces
5 if expression:
6     statement(s)
7 else:
8     statement(s)
```

# Control Structures

- else if
- no switches or cases in python

```
1 if expression1:  
2     statement(s)  
3 elif expression2:  
4     statement(s)  
5 elif expression3:  
6     statement(s)  
7 else:  
8     statement(s)
```

# Loops

- For
- While

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

```
1 ...
2 17 is a prime number
3 18 equals 2 * 9
4 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

```
1 count = 0
2 while (count < 9):
3     print 'The count is:', count
4     count = count + 1
5
6 print "Good bye!"
```

# Strings

## Strings vs Numbers

```
1 string = "123456"  
2 number = 123456  
3 string is number  
4 int(string) is number # different "objects"  
5 int(string)==number # testing equality of value
```

```
1 False  
2 False  
3 True
```

## Strings vs lists of strings

```
1 a = [string]  
2 b = [string]  
3 a == b # compares equality  
4 a is b # compares whether objects
```

```
1 >>> True  
2 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>

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

```
1 'Test strings'
2 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

```
1 def printnum( x, y ):  
2     """This passes a parameter to the print statement"""  
3     print x, y  
4     return  
5  
6 printnum(y=3, x="printing this:")  
7 printnum("positional ordering matter if not named", 4)
```

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

# Modules

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

```
1 import modulename
2 import modulename as shortname
3 import functionx from module name
```

# CSV files - Basic

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

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

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

# CSV files - Custom

- troublesome

```
1 class customcsv(csv.Dialect):  
2     lineterminator = '\n'  
3     delimiter = ','  
4     quoting = csv.QUOTE_NONE  
5  
6 fl.csv = csv.reader("test.csv", dialect=customcsv)  
7 fl.csv
```

# CSV files - Pandas - read<sub>csv</sub>

```
1 import pandas as pd
2 # header=None if not in file
3 # or read_table + sep(delimiter)
4 fldf = pd.read_csv("test.csv")
5 type(fldf) #type is different
6 fldf
```

```
1 <class 'pandas.core.frame.DataFrame'>
2      head1  head2
3  0         1      2
4  1         3      4
5
6 [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](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<sub>csv</sub>

[http://pandas.pydata.org/pandas-docs/version/0.13.1/generated/pandas.DataFrame.to\\_csv.html](http://pandas.pydata.org/pandas-docs/version/0.13.1/generated/pandas.DataFrame.to_csv.html)

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

```
1 head1  head2
2 0      1      2
3 1      3      4
4
5 [2 rows x 2 columns]
6 >>> ['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\\_XPath](http://www.zvon.org/comp/r/tut-XPath_1.html#Pages~List_of_XPath)

- Full reference

<http://www.w3.org/TR/xpath/>

# XML - Loading

```
1 xml = """
2     <root>
3         <name type="superhero">Batman</name>
4         <sidekick>Batty</sidekick>
5         <contact type="email">riseup@batman.com</contact>
6         <contact type="phone">555-1212</contact>
7     </root>
8     """
9
10 from lxml import objectify
11 root = objectify.fromstring(xml) #use parse from file
12
13 print root.tag
14 print root.text
15 print root.attrib
16
17 print root.name.tag
18 print root.name.text
19 print root.name.attrib
20
21 for con in root.contact:
22     print con.text
23     print con.attrib
```

[view source](#)

# JSON - Loading

```
1 jsn = """
2     {"name": "batman",
3      "hobbies": ["fast cars", "fast planes", "spending money"],
4      "buddy": "robin",
5      "enemies": [{"name": "The Joker"},
6                  {"name": "The People of Gotham"}]
7     }
8 """
9 import json
10 #NOTE: loads for strings, load for files
11 rslt = json.loads(jsn) #put this into a form for python
12 print rslt
13 jsn_again = json.dumps(rslt) #back to json
```

```
1 {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

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

```
1      name  
2 0      The Joker  
3 1  The People of Gotham  
4  
5 [2 rows x 1 columns]
```

# JSON - Converting to DataFrames

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

```
1      name
2 0      The Joker
3 1  The People of Gotham
4
5 [2 rows x 1 columns]
```

# JSON - Example

```
1 import json
2 import urllib2
3 import pprint
4 import pandas as pd
5
6 prefix="http://maps.googleapis.com/maps/api/geocode/json?address="
7 suffix("&sensor=false")
8 address="165%20Whitney%20Avenue,%20New%20Haven,%20CT"
9 url = prefix+address+suffix
10 j = urllib2.urlopen(url)
11 js = json.load(j)
12 type(js) #if in doubt, check type
13
14 #pprint(js)
15
16 #notice nested list, so use index to get into it
17 rstadd = js['results'][0]['address_components']
18
19 for rs in rstadd:
20     print rs['short_name'], rs['types']
21
22 import pandas as pd
23 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>