Learn Python Through Public Data Hacking

David Beazley
@dabeaz
http://www.dabeaz.com

Presented at PyCon'2013, Santa Clara, CA
March 13, 2013

Requirements

- Python 2.7 or 3.3
- Support files:

  http://www.dabeaz.com/pydata

- Also, datasets passed around on USB-key

Welcome!

- And now for something completely different
- This tutorial merges two topics
    - Learning Python
    - Public data sets
- I hope you find it to be fun

Primary Focus

- Learn Python through practical examples
- Learn by doing!
- Provide a few fun programming challenges

Not a Focus

- Data science
  - Statistics
  - GIS
  - Advanced Math
  - "Big Data"
- We are learning Python

Approach

- Coding! Coding! Coding! Coding!
- Introduce yourself to your neighbors
- You're going to work together
- A bit like a hackathon

Your Responsibilities

- Ask questions!
- Don't be afraid to try things
- Read the documentation!
- Ask for help if stuck

Ready, Set, Go...

Running Python

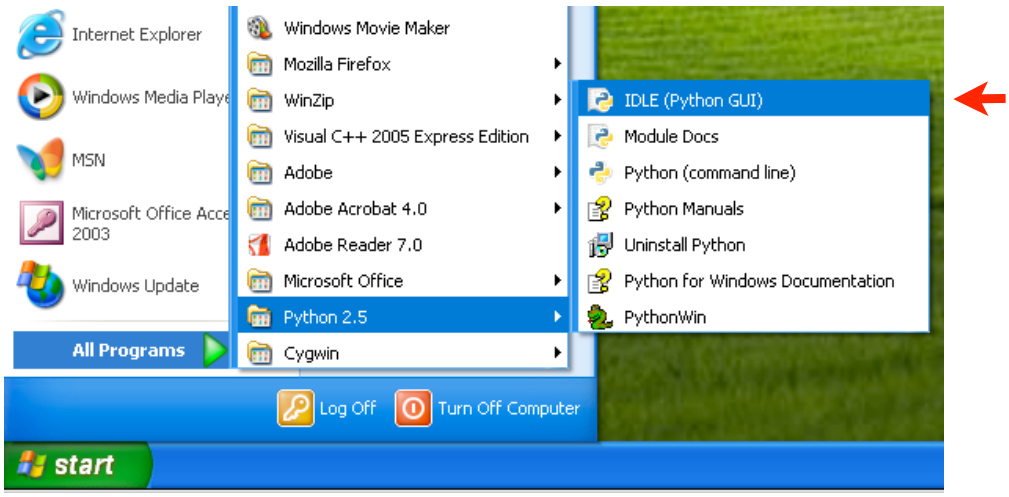
- Run it from a terminal

```
bash % python
Python 2.7.3 (default, Jun 13 2012, 15:29:09)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on da
Type "help", "copyright", "credits" or "license"
>>> print 'Hello World'
Hello World
>>> 3 + 4
7
>>>
```

- Start typing commands

IDLE

- Look for it in the "Start" menu



Interactive Mode

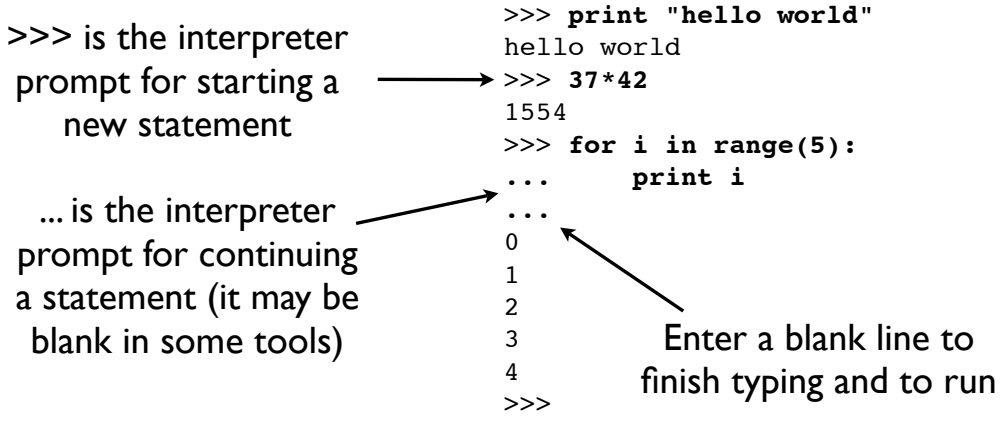
- The interpreter runs a "read-eval" loop

```
>>> print "hello world"
hello world
>>> 37*42
1554
>>> for i in range(5):
...     print i
...
0
1
2
3
4
>>>
```

- It runs what you type

Interactive Mode

- Some notes on using the interactive shell



Creating Programs

- Programs are put in .py files

```
# helloworld.py
print "hello world"
```

- Create with your favorite editor (e.g., emacs)
- Can also edit programs with IDLE or other Python IDE (too many to list)

Running Programs

- Running from the terminal
- Command line (Unix)

```
bash % python helloworld.py
hello world
bash %
```

- Command shell (Windows)

```
C:\SomeFolder>helloworld.py
hello world

C:\SomeFolder>c:\python27\python helloworld.py
hello world
```

Pro-Tip

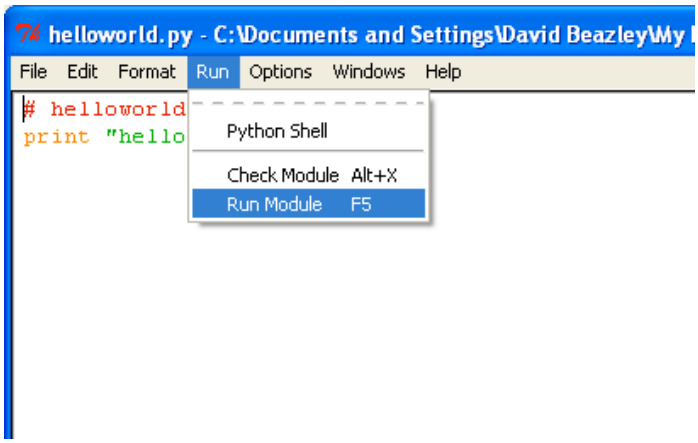
- Use python -i

```
bash % python -i helloworld.py
hello world
>>>
```

- It runs your program and then enters the interactive shell
- Great for debugging, exploration, etc.

Running Programs (IDLE)

- Select "Run Module" from editor



- Will see output in IDLE shell window

Python 101 : Statements

- A Python program is a sequence of statements
- Each statement is terminated by a newline
- Statements are executed one after the other until you reach the end of the file.

Python 101 : Comments

- Comments are denoted by #

```
# This is a comment
height    = 442           # Meters
```

- Extend to the end of the line

Python 101: Variables

- A variable is just a name for some value
- Name consists of letters, digits, and _.
- Must start with a letter or _

```
height = 442
user_name = "Dave"
filename1 = 'Data/data.csv'
```


Python 101 : Basic Types

- Numbers

```
a = 12345           # Integer
b = 123.45          # Floating point
```

- Text Strings

```
name = 'Dave'
filename = "Data/stocks.dat"
```

- Nothing (a placeholder)

```
f = None
```

Python 101 : Math

- Math operations behave normally

```
y = 2 * x**2 - 3 * x + 10
z = (x + y) / 2.0
```

- Potential Gotcha: Integer Division in Python 2

```
>>> 7/4
1
>>> 2/3
0
```

- Use decimals if it matters

```
>>> 7.0/4
1.75
```


Python 101 : Text Strings

```
a = 'Hello'
b = 'World'
```

- A few common operations

```
>>> len(a)                  # Length
5
>>> a + b                   # Concatenation
'HelloWorld'
>>> a.upper()               # Case convert
'HELLO'
>>> a.startswith('Hell') # Prefix Test
True
>>> a.replace('H', 'M')  # Replacement
'Mello
>>>
```

Python 101: Conversions

- To convert values

```
a = int(x)          # Convert x to integer
b = float(x)        # Convert x to float
c = str(x)          # Convert x to string
```

- Example:

```
>>> xs = '123'
>>> xs + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' c
>>> int(xs) + 10
133
>>>
```


Python 101 : Conditionals

- If-else

```
if a < b:
    print "Computer says no"
else:
    print "Computer says yes"
```

- If-elif-else

```
if a < b:
    print "Computer says not enough"
elif a > b:
    print "Computer says too much"
else:
    print "Computer says just right"
```

Python 101 : Relations

- Relational operators

```
<  >  <=  >=  ==  !=
```

- Boolean expressions (and, or, not)

```
if b >= a and b <= c:
    print "b is between a and c"

if not (b < a or b > c):
    print "b is still between a and c"
```


Python 101: Looping

- while executes a loop

```
n = 10
while n > 10:
    print 'T-minus', n
    n = n - 1
print 'Blastoff!'
```

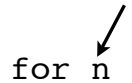
- Executes the indented statements underneath while the condition is true

Python 101: Iteration

- for iterates over a sequence of data

```
names = ['Dave', 'Paula', 'Thomas', 'Lewis']
for name in names:
    print name
```

- Processes the items one at a time
- Note: variable name doesn't matter



```
for n in names:
    print n
```


Python 101 : Indentation

- There is a preferred indentation style
  - Always use spaces
  - Use 4 spaces per level
  - Avoid tabs
- Always use a Python-aware editor

Python 101 : Printing

- The print statement (Python 2)

```
print x
print x, y, z
print "Your name is", name
print x,                        # Omits newline
```

- The print function (Python 3)

```
print(x)
print(x, y, z)
print("Your name is", name)
print(x, end=' ')               # Omits newline
```


Python 101: Files

- Opening a file

```
f = open("foo.txt","r")  # Open for reading
f = open("bar.txt","w")  # Open for writing
```

- To read data

```
data = f.read()          # Read all data
```

- To write text to a file

```
g.write("some text\n")
```

Python 101: File Iteration

- Reading a file one line at a time

```
f = open("foo.txt","r")
for line in f:
    # Process the line
    ...
f.close()
```

- Extremely common with data processing

Python 101: Functions

- Defining a new function

```
def hello(name):
    print('Hello %s!' % name)

def distance(lat1, lat2):
    'Return approx miles between lat1 and lat2'
    return 69 * abs(lat1 - lat2)
```

- Example:

```
>>> hello('Guido')
Hello Guido!
>>> distance(41.980262, 42.031662)
3.5465999999995788
>>>
```

Python 101: Imports

- There is a huge library of functions
- Example: math functions

```
import math

x = math.sin(2)
y = math.cos(2)
```

- Reading from the web

```
import urllib    # urllib.request on Py3

u = urllib.urlopen('http://www.python.org)
data = u.read()
```


Coding Challenge

"*The Traveling Suitcase*"

The Traveling Suitcase

Travis traveled to Chicago and took the Clark Street #22 bus up to Dave's office.

***Problem**: He just left his suitcase on the bus!*

***Your task**: Get it back!*

Panic!

- Start the Python interpreter and type this

```
>>> import urllib
>>> u = urllib.urlopen('http://ctabustracker.com/
bustime/map/getBusesForRoute.jsp?route=22')
>>> data = u.read()
>>> f = open('rt22.xml', 'wb')
>>> f.write(data)
>>> f.close()
>>>
```

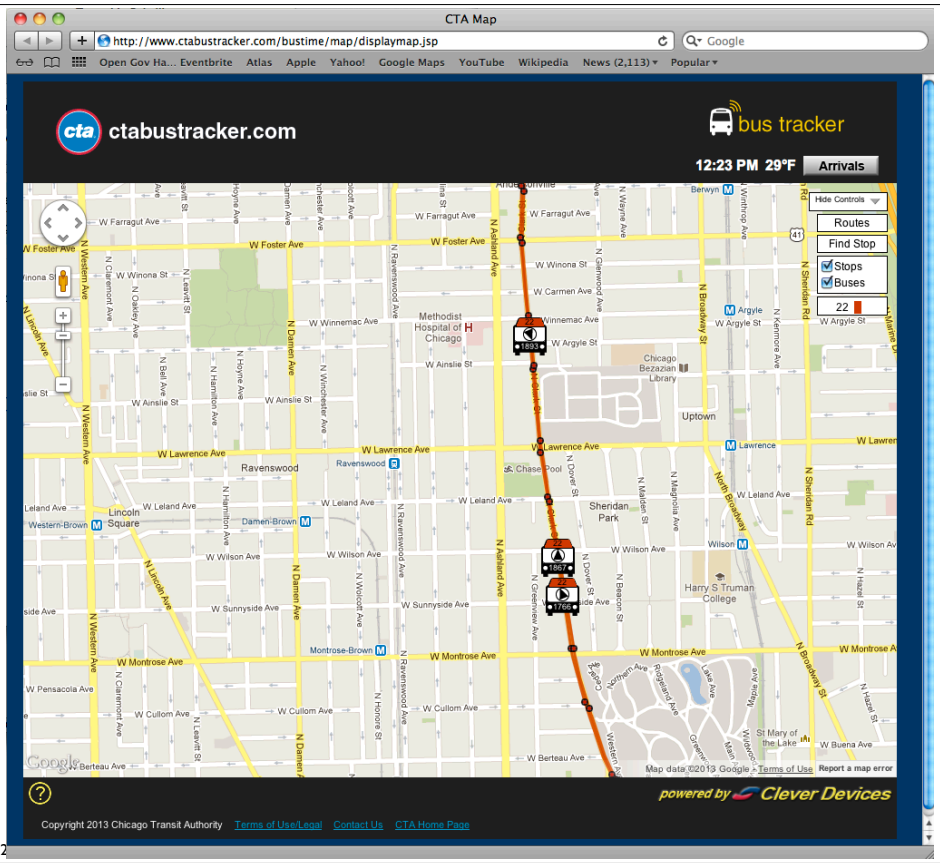
- Don't ask questions: you have 5 minutes...

Hacking Transit Data

- Many major cities provide a transit API
- Example: Chicago Transit Authority (CTA)

  http://www.transitchicago.com/developers/

- Available data:
  - Real-time GPS tracking
  - Stop predictions
  - Alerts



Here's the Data

```
<?xml version="1.0"?>
    <buses rt="22">
       <time>1:14 PM</time>
       <bus>
          <id>6801</id>
          <rt>22</rt>
          <d>North Bound</d>
          <dn>N</dn>
          <lat>41.875033214174465</lat>
          <lon>-87.62907409667969</lon>
          <pid>3932</pid>
          <pd>North Bound</pd>
          <run>P209</run>
          <fs>Howard</fs>
          <op>34058</op>
           ...
        </bus>
```


Here's the Data

```
<?xml version="1.0"?>
    <buses rt="22">
       <time>1:14 PM</time>
       <bus>
          <id>6801</id>
          <rt>22</rt>
          <d>North Bound</d>
          <dn>N</dn>
          <lat>41.875033214174465</lat>
          <lon>-87.62907409667969</lon>
          <pid>3932</pid>
          <pd>North Bound</pd>
          <run>P209</run>
          <fs>Howard</fs>
          <op>34058</op>
           ...
        </bus>
```

Your Challenge

- Task 1:

  Travis doesn't know the number of the bus he was riding. Find likely candidates by parsing the data just downloaded and identifying vehicles traveling northbound of Dave's office.

  Dave's office is located at:

```
latitude   41.980262
longitude -87.668452
```


Your Challenge

- Task 2:

  Write a program that periodically monitors the identified buses and reports their current distance from Dave's office.

  When the bus gets closer than 0.5 miles, have the program issue an alert by popping up a web-page showing the bus location on a map.

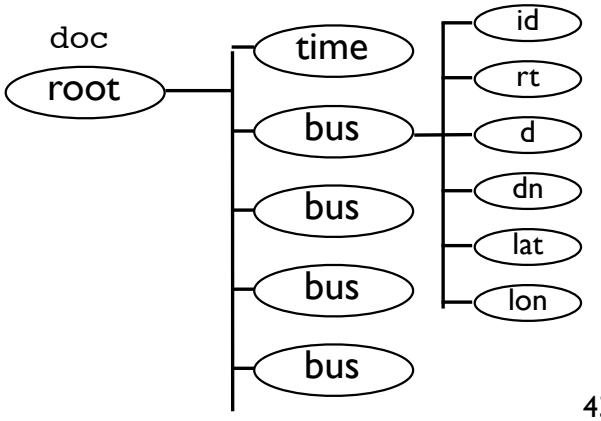
  Travis will meet the bus and get his suitcase.

Parsing XML

- Parsing a document into a tree

```
from xml.etree.ElementTree import parse
doc = parse('rt22.xml')
```

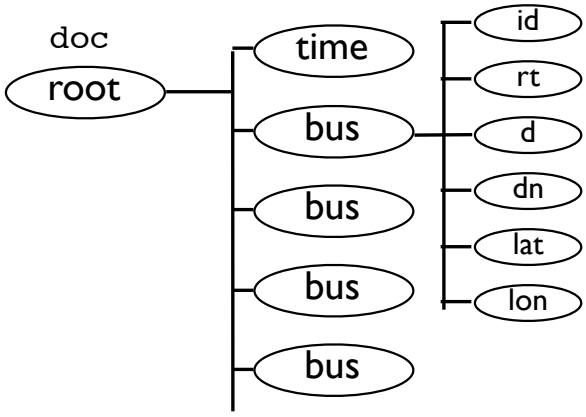
```
<?xml version="1.0"?>
  <buses rt="22">
     <time>1:14 PM</time>
     <bus>
        <id>6801</id>
        <rt>22</rt>
        <d>North Bound</d>
        <dn>N</dn>
        <lat>41.875033214174465</lat>
        <lon>-87.62907409667969</lon>
        <pid>3932</pid>
        <pd>North Bound</pd>
        <run>P209</run>
        <fs>Howard</fs>
        <op>34058</op>
           ...
     </bus>
```



Parsing XML

- Iterating over specific element type

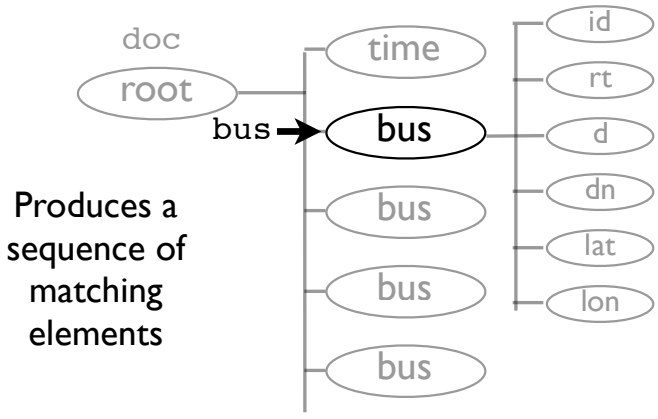
```
for bus in doc.findall('bus'):
    ...
```



Parsing XML

- Iterating over specific element type

```
for bus in doc.findall('bus'):
    ...
```

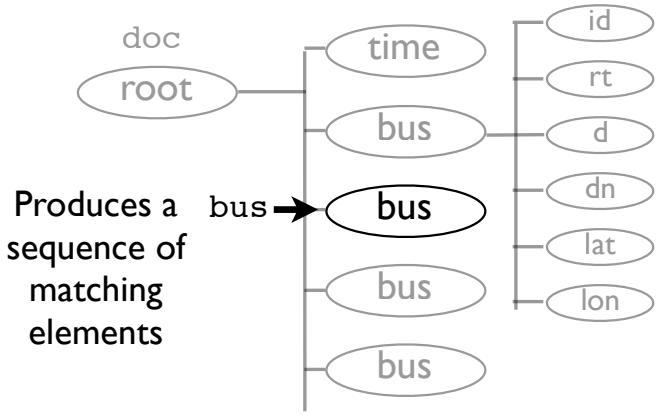


Produces a sequence of matching elements

Parsing XML

- Iterating over specific element type

```
for bus in doc.findall('bus'):
    ...
```

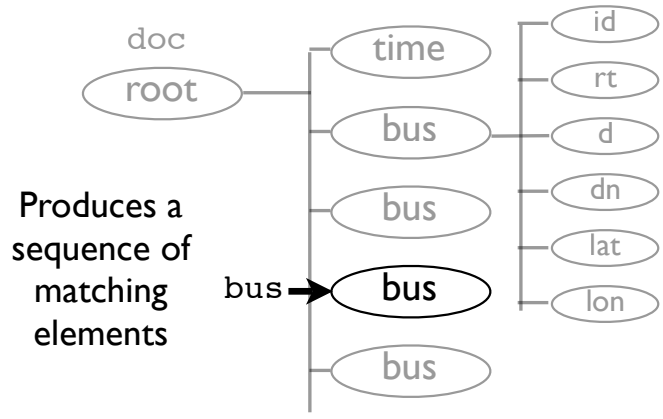


Produces a sequence of matching elements

Parsing XML

- Iterating over specific element type

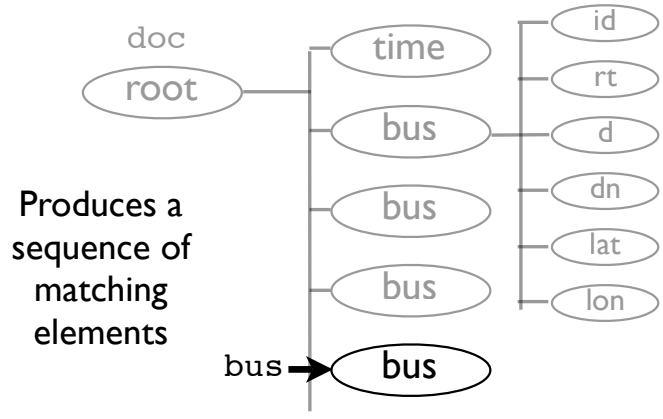
```
for bus in doc.findall('bus'):
    ...
```



Parsing XML

- Iterating over specific element type

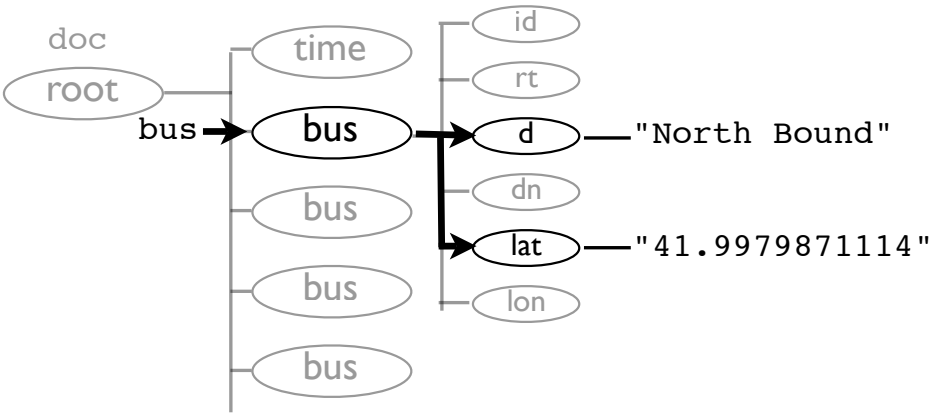
```
for bus in doc.findall('bus'):
    ...
```



Parsing XML

- Extracting data : elem.findtext()

```
for bus in doc.findall('bus'):
    d = bus.findtext('d')
    lat = float(bus.findtext('lat'))
```



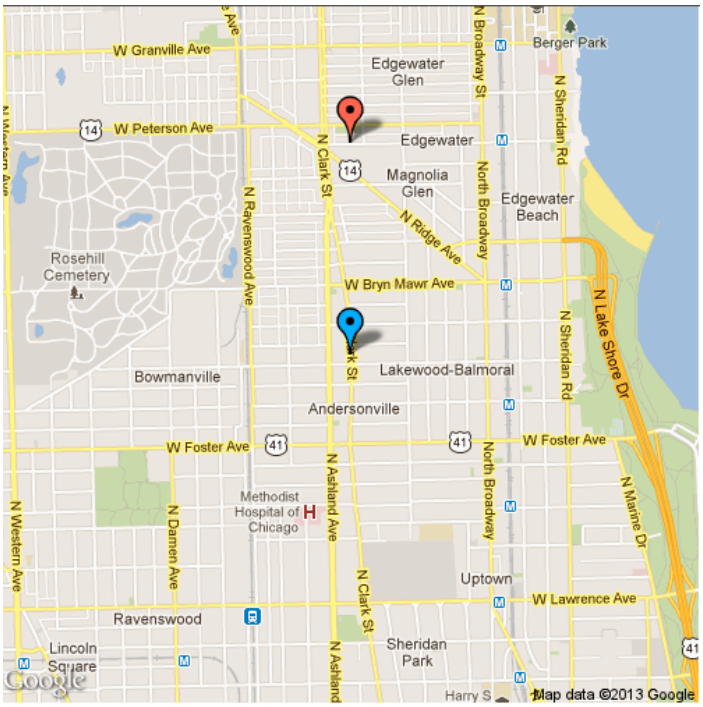
Mapping

- To display a map : Maybe Google Static Maps

https://developers.google.com/maps/documentation/staticmaps/

- To show a page in a browser

```
import webbrowser
webbrowser.open('http://...')
```



Go Code...

30 Minutes

- Talk to your neighbors
- Consult handy cheat-sheet
- http://www.dabeaz.com/pydata

New Concepts

Data Structures

- Real programs have more complex data
- Example: A place marker

```
Bus 6541 at 41.980262, -87.668452
```

- An "object" with three parts
  - Label ("6541")
  - Latitude (41.980262)
  - Longitude (-87.668452)

Tuples

- A collection of related values grouped together
- Example:

```
bus = ('6541', 41.980262, -87.668452)
```

- Analogy: A row in a database table
- A single object with multiple parts

Tuples (cont)

- Tuple contents are ordered (like an array)

```
bus = ('6541', 41.980262, -87.668452)
id = bus[0]       # '6541'
lat = bus[1]      # 41.980262
lon = bus[2]      # -87.668452
```

- However, the contents can't be modified

```
>>> bus[0] = '1234'
TypeError: object does not support item
assignment
```

Tuple Unpacking

- Unpacking values from a tuple

```
bus = ('6541', 41.980262, -87.668452)

id, lat, lon = bus
# id = '6541'
# lat = 41.980262
# lon = -87.668452
```

- This is extremely common
- Example: Unpacking database row into vars

Dictionaries

- A collection of values indexed by "keys"
- Example:

```
bus = {
   'id' : '6541',
   'lat' : 41.980262,
   'lon' : -87.668452
}
```

- Use:

```
>>> bus['id']
'6541'
>>> bus['lat'] = 42.003172
>>>
```

Lists

- An ordered sequence of items

```
names = ['Dave', 'Paula', 'Thomas']
```

- A few operations

```
>>> len(names)
3
>>> names.append('Lewis')
>>> names
['Dave', 'Paula', 'Thomas', 'Lewis']
>>> names[0]
'Dave'
>>>
```


List Usage

- Typically hold items of the same type

```
nums = [10, 20, 30]

buses = [
    ('1412', 41.8750332142, -87.6290740967),
    ('1406', 42.0126361553, -87.6747320322),
    ('1307', 41.8886332973, -87.6295552408),
    ('1875', 41.9996211482, -87.6711741429),
    ('1780', 41.9097633362, -87.6315689087),
]
```

Dicts as Lookup Tables

- Use a dict for fast, random lookups
- Example: Bus locations

```
bus_locs = {
    '1412': (41.8750332142, -87.6290740967),
    '1406': (42.0126361553, -87.6747320322),
    '1307': (41.8886332973, -87.6295552408),
    '1875': (41.9996211482, -87.6711741429),
    '1780': (41.9097633362, -87.6315689087),
}

>>> bus_locs['1307']
(41.8886332973, -87.6295552408)
>>>
```


Sets

- An ordered collections of unique items

```
ids = set(['1412', '1406', '1307', '1875'])
```

- Common operations

```
>>> ids.add('1642')
>>> ids.remove('1406')
>>> '1307' in ids
True
>>> '1871' in ids
False
>>>
```

- Useful for detecting duplicates, related tasks

Coding Challenge

"Diabolical Road Biking"

Problem

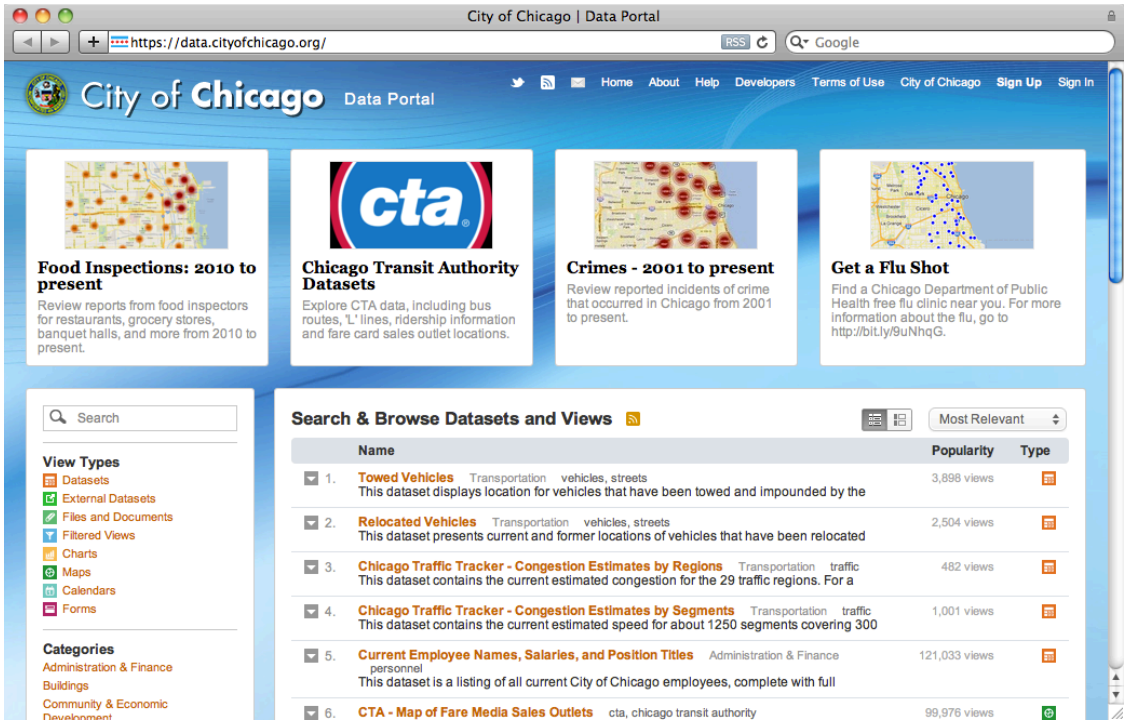
Not content to ride your bike on the lakefront path, you seek a new road biking challenge involving large potholes and heavy traffic.

Your Task: *Find the five most post-apocalyptic pothole-filled 10-block sections of road in Chicago.*

Bonus: *Identify the worst road based on historical data involving actual number of patched potholes.*

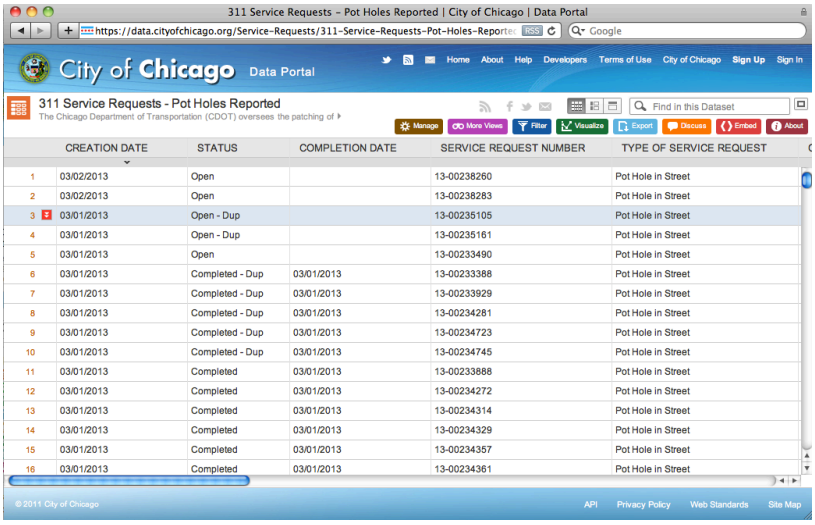
Data Portals

- Many cities are publishing datasets online
  - http://data.cityofchicago.org
  - https://data.sfgov.org/
  - https://explore.data.gov/
- You can download and play with data



Pothole Data

https://data.cityofchicago.org/Service-Requests/311-Service-Requests-Pot-Holes-Reported/7as2-ds3y



Getting the Data

- You can download from the website
- I have provided a copy on USB-key

```
Data/potholes.csv
```

- Approx: 31 MB, 137000 lines

Parsing CSV Data

- You will need to parse CSV data

```
import csv

f = open('potholes.csv')
for row in csv.DictReader(f):
    addr = row['STREET ADDRESS']
    num = row['NUMBER OF POTHOLES FILLED ON BLOCK']
```

- Use the CSV module

Tabulating Data

- You'll probably need to make lookup tables

```
potholes_by_block = {}

f = open('potholes.csv')
for row in csv.DictReader(f):
    ...
    potholes_by_block[block] += num_potholes
    ...
```

- Use a dict. Map keys to counts.

String Splitting

- You might need to manipulate strings

```
>>> addr = '350 N STATE ST'
>>> parts = addr.split()
>>> parts
['350', 'N', 'STATE', 'ST']
>>> num = parts[0]
>>> parts[0] = num[:-2] + 'XX'
>>> parts
['3XX', 'N', 'STATE', 'ST']
>>> ' '.join(parts)
'3XX N STATE ST'
>>>
```

- For example, to rewrite addresses

Data Reduction/Sorting

- Some useful data manipulation functions

```
>>> nums = [50, 10, 5, 7, -2, 8]
>>> min(nums)
-2
>>> max(nums)
50
>>> sorted(nums)
[-2, 5, 7, 8, 10, 50]
>>> sorted(nums, reverse=True)
[50, 10, 8, 7, 5, -2]
>>>
```

Exception Handling

- You might need to account for bad data

```
for row in csv.DictReader(f):
    try:
       n = int(row['NUMBER OF POTHOLES FILLED'])
    except ValueError:
       n = 0
    ...
```

- Use try-except to catch exceptions (if needed)

Code...

40 Minutes

Hint: This problem requires more thought than actual coding

(The solution is small)

Power Tools

(Python powered)

List Comprehensions

- Creates a new list by applying an operation to each element of a sequence.

```
>>> a = [1,2,3,4,5]
>>> b = [2*x for x in a]
>>> b
[2, 4, 6, 8, 10]
>>>
```

- Shorthand for this:

```
>>> b = []
>>> for x in a:
...     b.append(2*x)
...
>>>
```

List Comprehensions

- A list comprehension can also filter

```
>>> a = [1, -5, 4, 2, -2, 10]
>>> b = [2*x for x in a if x > 0]
>>> b
[2, 8, 4, 20]
>>>
```


List Comp: Examples

- Collecting the values of a specific field

```
addrs = [r['STREET ADDRESS'] for r in records]
```

- Performing database-like queries

```
filled = [r for r in records
            if r['STATUS'] == 'Completed']
```

- Building new data structures

```
locs = [ (r['LATITUDE'], r['LONGITUDE'])
         for r in records ]
```

Simplified Tabulation

- Counter objects

```
from collections import Counter

words = ['yes','but','no','but','yes']
wordcounts = Counter(words)

>>> wordcounts['yes']
2
>>> wordcounts.most_common()
[('yes', 2), ('but', 2), ('no', 1)]
>>>
```


Advanced Sorting

- Use of a key-function

```
records.sort(key=lambda p: p['COMPLETION DATE'])
records.sort(key=lambda p: p['ZIP'])
```

- lambda: creates a tiny in-line function

```
f = lambda p: p['COMPLETION DATE']

# Same as
def f(p):
    return p['COMPLETION DATE']
```

- Result of key func determines sort order

Grouping of Data

- Iterating over groups of sorted data

```
from itertools import groupby
groups = groupby(records, key=lambda r: r['ZIP'])
for zipcode, group in groups:
    for r in group:
        # All records with same zip-code
        ...
```

- Note: data must already be sorted by field

```
records.sort(key=lambda r: r['ZIP'])
```


Index Building

- Building indices to data

```
from collections import defaultdict

zip_index = defaultdict(list)
for r in records:
    zip_index[r['ZIP']].append(r)
```

- Builds a dictionary

```
zip_index = {
    '60640' : [ rec, rec, ... ],
    '60637' : [ rec, rec, rec, ... ],
    ...
}
```

Third Party Libraries

- Many useful packages
  - numpy/scipy (array processing)
  - matplotlib (plotting)
  - pandas (statistics, data analysis)
  - requests (interacting with APIs)
  - ipython (better interactive shell)
  - Too many others to list

Coding Challenge

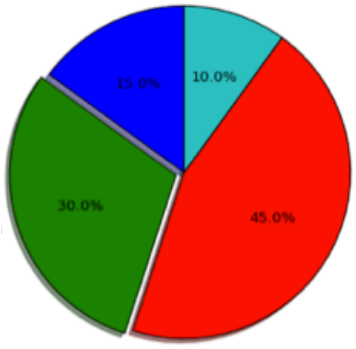
"Hmmmm.... Pies"

Problem

You're ravenously hungry after all of that biking, but you can never be too careful.

Problem

You're ravenously hungry after all of that biking, but you can never be too careful.



Your Task: *Analyze Chicago's food inspection data and make a series of tasty pie charts and tables*

The Data

https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5

- It's a 77MB CSV file. Don't download
- Available on USB key (passed around)
- New challenges abound!

Problems of Interest

- Outcomes of a health-inspection (pass, fail)
- Risk levels
- Breakdown of establishment types
- Most common code violations
- Use your imagination...

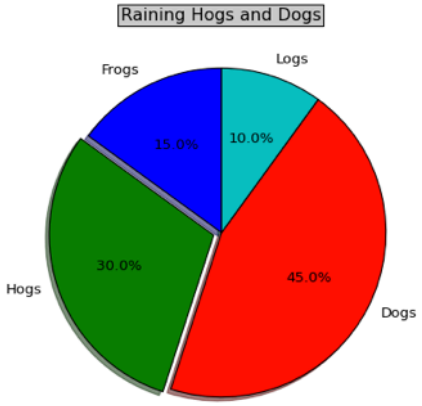
To Make Charts...

matplotlib

home | downloads | search | examples | gallery | citation | docs »

pylab_examples example code: pie_demo.py

(Source code, png, hires.png, pdf)



You're going to have to install some packages...

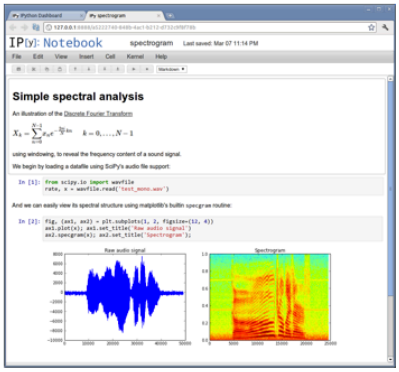
Bleeding Edge

IP[y]: IPython
Interactive Computing

Install · Docs · Videos · Notebook Viewer · News · Cite · Donate

The IPython Notebook

The IPython Notebook is a web-based interactive computational environment where you can combine code execution, text, mathematics, plots and rich media into a single document:



Code

45 Minutes

- Code should not be long
- For plotting/ipython consider EPD-Free, Anaconda CE, or other distribution
- See samples at http://www.dabeaz.com/pydata

Where To Go From Here?

- Python coding
  - Functions, modules, classes, objects
- Data analysis
  - Numpy/Scipy, pandas, matplotlib
- Data sources
  - Open government, data portals, etc.

Final Comments

- Thanks!
- Hope you had some fun!
- Learned at least a few new things
- Follow me on Twitter: @dabeaz