Python - A Crash Course

Antonio Lima

Networks and Distributed Systems 2012-2013 School of Computer Science - University of Birmingham 15 March 2013

Last week's Homework

Write a Python program that counts the occurrences of words in a wordlist.

```
counter = {}
with open(filename, "r") as f:
    for line in f:
        if line in counter:
            counter[line] += 1
        else:
        counter[line] = 1
```

We are asking permission. It's easier to ask forgiveness.

Last week's Homework

You ask forgiveness by executing a task and recovering if there is an error.

```
counter = {}
with open(filename, "r") as f:
    for line in f:
        try:
        counter[line] += 1
        except KeyError:
        counter[line] = 1
```

That's much better (one operation less). But there's even a more pythonic solution.

Last week's Homework

You ask forgiveness by executing a task and recovering if there is an error.

```
import collections
counter = collections.defaultdict(int)
with open(filename, "r") as f:
    for line in f:
        counter[line] += 1
```

5 lines, very readable, correct. Pythonic.

Why using Python for Networking?

- Well, why not?
- Many successful companies/startups are using it.
 - Google, Yahoo, Dropbox, Quora, Disqus, ...
- A lot of libraries are available.
- You can build prototypes quickly.

The socket module: not only sockets

The module provides access to the BSD socket interface. Very low-level.

```
>>> import socket
>>> hostname = 'google.com'
>>> addr = socket.gothostbyname(hostname)
'173.194.41.102'
>>> socket.getservbyname('ssh')
22
>>> socket.getservbyname('http')
80
```

The socket module: a client-server application

```
# Client
import socket
import sys
s = socket.socket(socket.AF INET, socket.SOCK DGRAM)
port = 1060, address = '127.0.0.1', max data = 65535
print 'Address before sending:', s.getsockname()
s.sendto('This is my message', (address, port))
print 'Address after sending', s.getsockname()
data, address = s.recvfrom(max data)
print 'The server', address, 'says', repr(data)
```

The socket module: not only sockets

```
# Server
import socket, sys
s = socket.socket(socket.AF INET, socket.SOCK DGRAM)
max data = 65535, port = 1060, address = '127.0.0.1'
s.bind((address, port))
print 'Listening at', s.getsockname()
while True:
    data, address = s.recvfrom(max data)
    print 'The client at', address, 'says', repr(data)
    s.sendto('Your data was %d bytes' % len(data), address)
```

Receiving a webpage

```
import socket, sys
sock = socket.socket()
sock.connect(('cs.bham.ac.uk', 80))
sock.send("GET / HTTP/1.0\r\n\r\n")
while 1:
    buf = sock.recv(4096)
    if not buf:
        break
    sys.stdout.write(buf)
sock.close()
```

Too low level. Time is money. It's not the right tool.

urlparse, urllib2

```
>>> p = urlparse.urlparse('http://bham.ac.uk:80/reg.php?
name=Antonio&role=student')
>>> p
ParseResult(scheme='http', netloc='bham.ac.uk:80', path='/
reg.php', params='', query='name=Antonio&role=student',
fragment='')
>>> urlparse.parse_qs(p.query)
{'role': ['student'], 'name': ['Antonio']}
>>> urlparse.urljoin('http://bham.ac.uk/', './courses')
'http://bham.ac.uk/courses'
```

urlparse, urllib2

However, this happens when you want to use authentication.

```
import urllib2
url = 'https://api.github.com'
req = urllib2.Request(url)
password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password manager.add password(None, gh url, 'user', 'pass')
auth manager = urllib2.HTTPBasicAuthHandler(password manager)
opener = urllib2.build opener(auth manager)
urllib2.install opener(opener)
handler = urllib2.urlopen(req)
print handler.getcode()
print handler.headers.getheader('content-type')
```

python-requests

It is not part of the standard library. You need to install it.

```
>>> import requests
>>> response = requests.get('https://api.github.com',
           auth=('user', 'pass'))
>>> print response.status code
200
>>> print response.encoding
'UTF-8'
>>> print response.text
>>> print response.json # if available
```

python-requests

Use this if you need advanced features:

- Keep-Alive.
- Streaming APIs support.
- •SSL Verification.
- Basic/digest authentication.
- Key/Value cookies support.
- Automatic decompression.

●...

Speaking about JSON...

Stands for JavaScript Object Notation. Increasingly used by Web Services, as an alternative to XML, as it has some advantages.

```
student = {
   "name": "John Smith",
   "age": 24,
   "marks": [4, 5, 4, 5, 3, 5],
}
student_json = json.dumps(student)
# Looks like '{"age": 24, "name": "John Smith", "marks": [4, 5, 4, 5, 3, 5]}'
json.loads(student_json) == student # True
```

How to install correctly

First, install and download pip (follow the instructions on the website). Then:

pip install python-requests

You can install many other packages using pip. If you want to explore even further the correct way to install packages, take a look at virtualenv and virtualenvwrapper.

Web programming

A quick web server.

```
python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000
```

Only serves static files and it is not very customizable.

Web programming

The SimpleHTTPServer is a subclass of BaseHTTPServer. We could subclass this to customize it for our needs. However, there are better options available.

- Micro web frameworks
 - bottle
 - web2py
 - cherrypy
 - ...
- "Medium" web frameworks
 - django
 - pylons
- "Large" web frameworks
 - zope (plone)

Bottle example

We will focus on micro webframeworks. They are quite similar.

```
from bottle import route, run, template
@route('/hello/:name')
def index(name='World'):
    return template('<b>Hello {{name}}</b>!', name=name)
run(host='localhost', port=8080)
```

Bottle example

The template system allows to separate the view from the controller, following model-view-controller pattern.

```
from bottle import route, run, template
@route('/hello')
@route('/hello/<name>')
@view('hello_template')
def hello(name='World'):
    return dict(name=name)

run(host='localhost', port=8080)
```

Bottle example

The template system allows to separate the view from the controller, following model-view-controller pattern.

Deploying a webapp

It is very important that you care about performances when deploying. Here are some basic usual improvements.

- Some micro webframeworks (such bottle) are single-threaded. Usually you can use a multi-threaded backend while not changing the application.
 - bottle.run(server='paste')
- Most of the resources are static. Use a high-performance webserver (nginx, lighttpd, Apache, Cherokee, ...) for those, and let the python webapp handle the dynamic pages.
- When the content is changing seldom and accessed many times (for example, web blogs), you should also consider using static site generators (for example jekyll) in conjunction with a high-performance webserver.

Crawlers, Scrapers

You are writing a crawler, i.e. an application that bulk downloads a set of resources from the web (either through a API or not) and a scraper, that analyzes the data.

- What are the problems you might encounter?
- What technologies and mechanisms to build a robust application?
- Want to try? Write an application that uses the Twitter streaming "filter" endpoint to get in realtime the tweets with the hashtag you choose.
 - https://dev.twitter.com/docs/api/1.1/post/statuses/filter
- Want more? Expand your application so that it tries to geocode the "location" of each tweet's author.