Hackinggroup – Python Workshop – Part 2

A tale about dutch ducks with a fable for British comedy

Thomas Kastner Michael Rodler

2012-12-16

Introduction – Wer san ma denn?

Michael Rodler

- aka f0rk, f0rki, f0rkmaster, Gabel, etc.
- Student SIB09
- 3 years coding python for fun
- 3 months coding python for profit

Thomas Kastner

- aka br3z3l, tom
- Student SIB08
- 4 years coding python for fun

Table of contents – Wos ma heit mochn

And because it's called a *Workshop* we will also write some code together ;)

python – syntactic sugar

```
>>> a = ""; b = "foo"
>>> x = a or b
>>> x = "a" * 21
>>> x = [1,2,3] * 5
```

python – syntactic sugar

```
>>> a = ""; b = "foo"
>>> x = a or b
>>> x = "a" * 21
>>> x = [1,2,3] * 5
```

python – some operators

```
>>> x = 42**2 % 1337
>>> 0xD5 & 0377 ^ 0xFF
```

python – syntactic sugar

```
>>> a = ""; b = "foo"
>>> x = a or b
>>> x = "a" * 21
>>> x = [1,2,3] * 5
```

python – some operators

```
>>> x = 42**2 % 1337
>>> 0xD5 & 0377 ^ 0xFF
```

python – converting types

```
>>> str(42)
>>> list((1,2,3,4))
>>> int("42")
```

python – os

```
>>> os.getloadavg()
>>> if os.getuid() == 0:
...     os.setuid(1000)
>>> open(os.devnull, 'w').write('This is sent to nirvana')
>>> os.killpg(1337,9)
```

python – os.walk

```
from os.path import join, getsize
for root, dirs, files in os.walk("./Code"):
    print root, "consumes",
    print sum([getsize(join(root, name)) for name in
        files]),
    print "bytes in", len(files), "non-directory files"
```

python – a fork bomb

```
import os
while True:
    pid = os.fork()
    if pid == 0:
        print "Hello I'm a child :D"
```

subprocces – execute processes

python

```
>>> p = subprocess.Popen(["ls", "-l", "-a"])
>>> retval = subprocess.call(["rm", "-f", "./somefile"])
```

subprocces – execute processes

python

```
>>> p = subprocess.Popen(["ls", "-l", "-a"])
>>> retval = subprocess.call(["rm", "-f", "./somefile"])
```

python

```
filename = input("What file would you like to display?\n")
subprocess.call("cat " + filename, shell=True)
```

subprocces – execute processes

python

```
>>> p = subprocess.Popen(["ls", "-l", "-a"])
>>> retval = subprocess.call(["rm", "-f", "./somefile"])
```

python

```
filename = input("What file would you like to display?\n")
subprocess.call("cat " + filename, shell=True)
```

type in: `non_existent; rm -rf / #`

subprocces – execute processes

python

```
>>> p = subprocess.Popen(["ls", "-l", "-a"])
>>> retval = subprocess.call(["rm", "-f", "./somefile"])
```

python

```
filename = input("What file would you like to display?\n")
subprocess.call("cat " + filename, shell=True)
```

type in: `non_existent; rm -rf / #`

Oh noes, command injection... this is bad :(

unfortunately, some complex commands require `shell=True`

→ *shlex* modul

python

```
>>> r = re.compile(r"(\w{31}=)")
>>> m = r.search("This is some random Text
    eS4H0NWnnFGd8cCUavc6m2DwjRUzm6h= which contains a flag
    ;)")
>>> if m:
...     print m.groups()
```

python

```
>>> r = re.compile(r"(\w{31}=)")
>>> m = r.search("This is some random Text
    eS4H0NWnnFGd8cCUavc6m2DwjRUzm6h= which contains a flag
    ;)")
>>> if m:
...     print m.groups()
```

python

```
>>> r = re.compile(r"\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}")
>>> valid = r.match("10.13.37.0") is not None
>>> valid = r.match(" 10.13.37.0") is not None
>>> valid = r.search(" 10.13.37.0") is not None
```

Comparison with other languages I

vs. C/C++

- write less code with more features in less time
- batteries included
- python bottlenecks can be optimized with modules written in C/C++

vs. Java

- faster to write code, less boilerplate code
- better api[1]
- python 2-50x slower than java
- python+psyco 1-5x slower than java
- real oop

Comparison with other languages II

vs. PHP

- faster than php
- more use cases
  - more libraries
- NOT ugly

conclusions

- start of python interpreter costs!
- rapid development
- **most performance bottlenecks are wrong algorithms**
- `http://wiki.python.org/moin/PythonSpeed`
  - new Interpreters: PyPy, Unladen Swallow, etc. are faster

[1] `paulbuchheit.blogspot.com/2007/05/amazingly-bad-apis.html`

List Comprehension and Generator Expression

python – list comprehensions

```
>>> n = [i for i in range(100) if i % 2]
>>> type(n)
<type 'list'>
>>> import os
>>> zipfiles = [n for _, _, n in os.walk(".") if
    n.endswith(".zip")]
```

- quickly construct lists
- list is constructed and then returned

List Comprehension and Generator Expression

python – generator expressions

```
>>> n = ( i**2 for i in range(100) if i & 1 == 0 )
>>> type(n)
<type 'generator'>
>>> import os
>>> zipfiles = (n for _, _, n in os.walk(".") if
    n.endswith(".zip"))
```

List Comprehension and Generator Expression

python – generator expressions

```
>>> n = ( i**2 for i in range(100) if i & 1 == 0 )
>>> type(n)
<type 'generator'>
>>> import os
>>> zipfiles = (n for _, _, n in os.walk(".") if
    n.endswith(".zip"))
```

- only one item is accessible
- therefore consumes less memory
- computed on the fly

Exceptions – oda wenn ois ind luft gehd

python

```
try:
    f = open("/dev/missing", "r")
except IOError, e:
    sys.stderr.write("Error: %s\n" % e.message)
    sys.exit(1)
```

Exceptions – oda wenn ois ind luft gehd

python

```
try:
    f = open("/dev/missing", "r")
except IOError, e:
    sys.stderr.write("Error: %s\n" % e.message)
    sys.exit(1)
```

python

```
try:
    import threading as _threading
except ImportError:
    import dummy_threading as _threading
```

It's easier to ask for forgiveness than permission (EAFP)

EVERYTHING IS AN OBJECT!

Everything is an Object

python

```
>>> x = 42; y = 42
>>> x == y
True
>>> x is y
True
>>> x = []; y = []
>>> x == y
True
>>> x is y
False
```

Numbers are actually singletons

- *is* checks for object identity
- $=$ checks for object equality

python – fun fact: functions are objects too

```
>>> def afunc(x):
...     afunc.x += x
...     return afunc.x
...
>>> afunc.x = 0
>>> afunc(1)
1
>>> afunc(2)
3
>>> afunc(1)
4
>>> afunc(10)
14
>>> afunc.x
14
>>> type(afunc)
<type 'function'>
```

python – defining a class

```
>>> class MyClass(object):
...     def __init__(self, y):
...         self.x = 42
...         self.y = y
...     def func(self):
...         return self.x
...
>>> c = MyClass(21)
>>> print c.func()
42
>>> print c.y
21
```

explicit inheritance from *object* is needed to specify "new-style" classes

everything is public. (private attributes are seldom needed)
mark for internal use by naming with preceding underscore

python – quasi private attribute

```
>>> class Fu(object):
...     def __init__(self):
...         self._x = 42
...         self.__y = 21
>>> f = Fu()
>>> f._x
42
>>> f.__y
AttributeError: 'Fu' object has no attribute '__y'
>>> f.__dict__
{'_Fu__y': 21, '_x': 42}
>>> f._Fu__y
21
```

python – multiple inheritance

```
>>> class A(object):
...     def foo(self):
...         print "A"
...
>>> class B(object):
...     def foo(self):
...         print "B"
...
>>> class C(A,B):
...     pass
...
>>> class D(B,A):
...     pass
...
```

python – multiple inheritance

```
>>> class A(object):
...     def foo(self):
...         print "A"
...
>>> class B(object):
...     def foo(self):
...         print "B"
...
>>> class C(A,B):
...     pass
...
>>> class D(B,A):
...     pass
...


>>> c = C()
>>> c.foo()
A
>>> d = D()
>>> d.foo()
B
```

quasi private from a heir's viewpoint I

python

```
>>> class Fu(object):
...     def __init__(self):
...         self._x = 42
...         self.__y = 21
...
...     def barf(self):
...         return self.__y
>>> class Bla(Fu):
...     pass
>>> b = Bla()
>>> b.barf()
21
>>> b.__y = 23
>>> b.barf()
21
>>> print b.__dict__
{'__y': 23, '_Fu__y': 21, '_x': 42}
```

quasi private from a heir's viewpoint II

python

```
>>> class Bla(Fu):
...         def nom(self):
...             self.__y = 34
>>> b = Bla()
>>> b.barf()
21
>>> b.nom()
>>> b.barf()
21
>>> print b.__dict__
{'_Bla__y': 34, '_Fu__y': 21, '_x': 42}
```

python – static aka class fields

```
class Counter(object):
    count = 0
    def __init__(self):
        self.__class__.count += 1
```

python – static aka class fields

```
class Counter(object):
    count = 0
    def __init__(self):
        self.__class__.count += 1
```

python – class method

```
class Counter(object):
    count = 0
    def __init__(self):
        self.__class__.count += 1
    @classmethod
    def print_count(cls):
        print cls.count
```

python – static methods

```
class SomeClass(object):
    @staticmethod
    def tostring(arg):
        return str(arg)
```

python – static methods

```
class SomeClass(object):
    @staticmethod
    def tostring(arg):
        return str(arg)
```

- Put functions in classes, which logically belong there
- Clean-up namespace
- Modules probably better suited

python – properties example

```
class C(object):
    def __init__(self):
        self._x = None

    @property
    def x(self):
        """I'm the 'x' property."""
        return self._x

    @x.setter
    def x(self, value):
        if len(str(value)) <= 5:
            self._x = value

    @x.deleter
    def x(self):
        self._old_x = self._x
        self._x = None
```

python – properties example

```
>>> c = C()
>>> c.x = 12
>>> print c.x
12
>>> c.x = 8
>>> print c.x
8
>>> del c.x
>>> print c.x
None
>>> print c._old_x
8
```

python – silly example

```
>>> class Duck(object):
...     def quack(self):
...         print "Quaaaaaack!"
...
>>> class Person(object):
...     def quack(self):
...         print "The person imitates a duck."
...
>>> def action(duck):
...     duck.quack()
...
>>> donald = Duck()
>>> fork = Person()
>>> action(donald)
>>> action(fork)
```

→ http://docs.python.org/reference/datamodel.html#special-method-names

python – list style access

```
>>> class MegaList(object):
...     def __getitem__(self, index):
...         return index ** 2
...
>>> m = MegaList()
>>> m[15]
```

python – iterator protocol

```
>>> class Megalterator(object):
...     cur = 0
...     max = 14
...
...     def __iter__(self):
...         return self
...
...     def next(self):
...         if cur < max:
...             cur += 1
...             return cur
...         else:
...             raise StopIteration()
...
>>> m = Megalterator()
>>> for i in m:
...     print i
...
```

The cheeseshop – installing stuff

Python Package Index

`http://pypi.python.org`

python

```
[py@workshop ~]$ easy_install pelican
[py@workshop ~]$ easy_install pip
[py@workshop ~]$ pip uninstall pelican
```

installs python ".egg"s

python – urlopen

```
>>> site = urllib.urlopen("http://f0rki.at")
>>> print f.read()
```

urllib, urllib2 – working with URLs

python – urlopen

```
>>> site = urllib.urlopen("http://f0rki.at")
>>> print f.read()
```

python – url encoding

```
>>> import urllib
>>> params = urllib.urlencode({'spam': 1, 'eggs': 2,
    'bacon': 0})
>>> f =
    urllib.urlopen("http://www.example.com/cgi-bin/query",
    params)
>>> print f.read()
```

Let's write some code

Exercises

- ✓ fetch url
- ✓ use `find_mail` to search for email adress
- ✓ write email list to file

behaves like a normal browser (with cookies and stuff)

python

```
>>> br = mechanize.Browser()
>>> br.open("http://www.example.com/")
>>> response1 = br.follow_link(text_regex=r"cheese\s*shop",
    nr=1)
>>> print br.title()
>>> print response1.geturl()
>>> print response1.read()
>>> br.select_form(name="order")
>>> br["someform"] = ["myusername", "password"]
>>> response2 = br.submit()
```

Protocol

- handles data in an asynchronous manner
- implements protocol parsing and handling
- never waits for an event

Factory

- persistent configuration is kept in a Factory class
- instantiate Protocol for each connection

Reactor

- watches sockets for events
- mainloop

Python threading API is inspired by Java threading API.

python – threading

Problems with Threads

The Global Interpreter Lock [2] [3] limits Python threads:

- real OS threads
- but only one CPU
- sometimes threads are slower than sequential computation
  - cpu intensive tasks
- GIL restricts access to Interpreter internals (and GIL unaware C-Extensions)

Solution:

- use *multiprocessing* module (very similar to threading API)
- Stackless Python `http://wiki.python.org/moin/StacklessPython`

---

[2] `http://wiki.python.org/moin/GlobalInterpreterLock`

[3] `http://www.dabeaz.com/GIL/`

python – unpacking ip header

```
>>> pkt =
    "E\x00\x00\x9c\x00\x00@\x00@\x11\xbe-\n\r%\x02\n\r%\x03"
>>> iphdr = struct.unpack("!BBHHHBBHII", pkt[0:20])
>>> print iphdr
(69, 0, 156, 0, 16384, 64, 17, 48685, 168633602, 168633603)
```

'!' → Network Byte-Order aka. Big-Endian

Format	C type
B	unsigned char
H	unsigned short
I	unsgined int

etc.

python – calling native printf

```
from ctypes import *
libc = CDLL("libc.so.6")
x = libc.time(None)
y = libc.printf("Test: %d, %f\n", 42, c_double(13.37))
print "c printf printed", y, "characters at", x
```

python – calling native printf

```
from ctypes import *
libc = CDLL("libc.so.6")
x = libc.time(None)
y = libc.printf("Test: %d, %f\n", 42, c_double(13.37))
print "c printf printed", y, "characters at", x
```

- Quick C Interaction
  - speed-ups
  - Accessing C library without bindings

Example:
PyDbg – open-source scriptable windows debugger, written in python using ctypes

paramiko

- `http://www.lag.net/paramiko/`
- many features
  - shell/command execution
  - Agent
  - SFTP Support
- both low and high level access

twisted

- `http://twistedmatrix.com/`
- event-driven networking engine
- Implements a large number of protocols
- very good framework
  - i.e. you'll have to do things their way

asyncore

- builtin `http://docs.python.org/library/asyncore.html`
- low-level socket handling

asynchat

- builtin `http://docs.python.org/library/asynchat.html`
- uses asyncore
- for protocols, with string terminated elements

Django

- `http://www.djangoproject.com/`
- "The Web Framework for perfectionists with deadlines."
- Model-View-Controller
- Database-Driven

CherryPy

- `http://www.cherrypy.org/`
- handles only HTTP
- more flexible

Jinja

- `http://jinja.pocoo.org/`
- mighty (html) templating engine

zope

- `http://www.zope.org/`
- Web application server
- recognized as "python killer app"

Python DB API 2.0

- Standard "Interface" for Database modules
- `http://www.python.org/dev/peps/pep-0249/`

sqlalchemy

- `http://www.sqlalchemy.org/`
- object-relational mapper
  - `http://elixir.ematia.de`
  - declarative extension

Alternative interpreters

- IPython
- BPython
- `http://wiki.python.org/moin/PythonEditors#EnhancedPythonshells`

editors

- vim
- scribes
- `http://wiki.python.org/moin/PythonEditors`

Usefull stuff II

debuggers

- pdb
- winpdb (rpdb2)
- pydb
- `http://wiki.python.org/moin/PythonDebuggers`

Integrated Development Environment

- Eclipse with Pydev
- NetBeans
- `http://wiki.python.org/moin/IntegratedDevelopmentEnvironments`

References/Links

- the authors' epic python knowledge
- `http://docs.python.org/`
- Dive into Python `http://diveintopython.org/`