

Python Tutorial

Day 4

Today

- environment variables
- `sys` module
- more regular expression stuff
- refine `SIDB`
- interacting with commands
- `global` warning

Environment Variables

- hang out in `os` module
- access with dictionary: `os.environ['HOME']`
- depending on platform, assigning to and deleting elements works as expected in the current process
- child processes will always inherit changed environment

Digression: Deleting

- mutable collection objects can have things removed from them
- the function is `del`
- the special method called is `__delitem__`
- list deletion: `del spam[3]` removes element 3
- dictionary deletion: `del spam['eggs']`

sys module

- `sys` remembers things about how Python was compiled
- `sys.platform` will say `darwin`, `sunos5`, etc.
- `sys.version` the long version string:

```
>>> print sys.version
'2.3.5 (#1, Mar 20 2005, 20:38:20)
[GCC 3.3 20030304 (Apple Computer, Inc. build 1809)]'
>>>
```

more sys

- `sys.stdin`, `sys.stdout`, `sys.stderr` give better control of IO than `print` and `raw_input`
- list `sys.argv` holds command line arguments
- list `sys.path` holds Python library search path; append to this to add your own libraries to the path

regexp: removing HTML tags

- very complex problem
- I just grabbed on off the internet
- translated from perl
- `sub` method does substitution
- this time I compiled the regexp

```
dt = re.compile(r"""/?(\w+)(\s*\w*\s*=\s*("[^"]*"|'[^']*'|[\^>]*))*/?>""",
    re.MULTILINE)

for line in sys.stdin.readlines():
    print dt.sub("", line),
```

Group extraction

- when a regexp with groups matches, you can pull out those groups
- groups: sub-sections of the regexp within parentheses
- here's a web log entry — want just the date for 404 (missing page) errors

```
38.100.225.3 - - [22/Mar/2006:10:39:05 -0600]  
"GET /generaladmin/visfaculty.html HTTP/1.0" 200 13455  
"- " "sproose/0.1-alpha (sproose crawler; http://www.sproose.com/bot.html;  
crawler@sproose.com) "
```

```

import sys
import re

# 152.1.119.190 - - [22/Mar/2006:12:15:59 -0600]
# "GET /favicon.ico HTTP/1.1" 404 283 "-" "Mozilla/5.0
# (X11; U; Linux i686; en-US; rv:1.8.0.1)
# Gecko/20060124 Firefox/1.5.0.1"

err = re.compile(r"\"\"\"[\d\.]*\s+\s+\s+\s+\"([\d\.]*)\s+\"([\d\.]*)\s+\"\"\"")

for line in sys.stdin.readlines():
    match = err.search(line)
    if match and match.group(3) == "404":
        print '%s: %s "%s"' % (match.group(3),
                               match.group(1), match.group(2))

#-----
404: 22/Mar/2006:12:18:20 -0600 "GET /netforum HTTP/1.1"
404: 22/Mar/2006:12:19:39 -0600 "GET /favicon.ico HTTP/1.1"
404: 22/Mar/2006:12:19:39 -0600 "GET /favicon.ico HTTP/1.1"

```

More special methods for SIDB

- last time I wrote `select_exact` and `select_match` methods
- let's make grabbing a hostname a special case
- we can use `__getitem__` method to make it look like an array

```
def __getitem__(self, hostname):
    return self.select_exact('hostname', hostname)

# ----
>>> db = InventoryDB("stuff")
>>> for a in db['wazor']:
...     print a
...
wazor (solaris), used by William Annis in J4/503.
>>>
```

pdb

- there's a debugging tool in the `pdb` module
- in interactive code, just import it

```
>>> db = InventoryDB("/etc/sudoers")
Traceback (most recent call last):
...
IOError: [Errno 13] Permission denied: '/etc/sudoers'
>>> import pdb
>>> pdb.pm()
> /private/var/automount/w/w05/a/annis/talks_articles/pythontut/sidb2.py(45)l
-> f = open(self.dbfile, "r")
(Pdb) p self.dbfile
'/etc/sudoers'
(Pdb) q
>>>
```

More pdb

- pdb has standard step, backtrace, etc., including help
- `python -m pdb SCRIPT.py` starts pdb immediately, then you can step through

```
~/pythontut $ python2.4 -m pdb sidb2.py
> /w/w05/a/annis/talks_articles/pythontut/sidb2.py(5)?()
-> import pwd
(Pdb) step
> /w/w05/a/annis/talks_articles/pythontut/sidb2.py(6)?()
-> import os
(Pdb) step
> /w/w05/a/annis/talks_articles/pythontut/sidb2.py(7)?()
-> import cPickle
(Pdb)
```

Global Trouble

- python uses “lexical scope”
- as we have seen, naming a variable by assigning to it is sufficient
- variables defined in functions stay there
- in fact, by default assignment in a function creates a **local** name
- if you want to change a global variable in a function, you have to say so, with `global`

```
>>> spam = "and eggs"
>>> def foo():
...     spam = "not ham"
...
>>> foo()
>>> spam
'and eggs'
>>> def bar():
...     global spam
...     spam = "not ham"
...
>>> bar()
>>> spam
'not ham'
>>>
```

Personal Library

- writing your own libraries is simple
- just name the file `SOMETHING.py`
- then you can do `import SOMETHING` or `from SOMETHING import spam`
- the current directory is in the python library path, but:

```
>>> import sys
>>> sys.path.append("/u/annis/code/pythonlib")
>>> import ...
```

Library Hierarchy

- you may create hierarchies of libraries
- each directory must contain `__init__.py`

```
~/pythontut $ mkdir mylib
~/pythontut $ touch mylib/__init__.py
~/pythontut $ touch mylib/spam.py
~/pythontut $ python
Python 2.3.5 (#1, Mar 20 2005, 20:38:20)
[GCC 3.3 20030304 (Apple Computer, Inc. build 1809)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import mylib
>>> import mylib.spam
>>>
```

Useful Idiom

- say you want to run a command and parse the output
- you *could* open a pipe by hand, etc.
- there's a convenient library: `commands`

```
>>> import commands
>>> m = commands.getoutput("uname -a")
>>> m
'Darwin cydonia.biostat.wisc.edu 8.3.0 Darwin Kernel Version 8.3.0:
Mon Oct  3 20:04:04 PDT 2005;
root:xnu-792.6.22.obj~2/RELEASE_PPC Power Macintosh powerpc'
>>>
```

Next Week

- some CGI and web examples
- ideas?