Day 6: Modules, Standard Library

Suggested reading: *Learning Python* (4th Ed.)

Ch.21: Modules: The Big Picture
Ch.22: Module Coding Basics

http://docs.python.org/release/2.4.3/modindex.html

Turn In Homework

Homework Review

Modules

Python Module = .py File

Why Write a Module?

- Maximize code reuse /
  Minimize code redundancy
- Organize code clearly (group related definitions)
- Make testable units of code
- Share data among separate script parts

(Does this list look familiar?)

Importing a Module

```
import module_name
```

1. Looks up or else creates **`module`** object
2. Assigns object to ***`module_name`***
3. Runs all top-level code in module!
4. Objects are bound to names in module namespace:
   - variables (via **`=`**)
   - functions (via **`def`**)
   - classes (via **`class`**)
   - other modules (via **`import`**)

Accessing Module Parts

module_name.*attribute*

- Same as accessing parts of an object
- Because a module *is* an object, which *is* a namespace

foo.py:

```
import math

f = 42

def circum(mult):
  d = 2 * f * mult
  return d * math.pi
```

bar.py:

```
import foo

print foo.math.pi

foo.f += 2

x = foo.circum(2)
```

Sample Module I

```
_c2f_ratio = 5.0 / 9.0  # "private"-ish
_f2c_ratio = 9.0 / 5.0  # "private"-ish

def convert_c2f(deg_f):
    return (deg_f - 32.0) * _c2f_ratio

def convert_f2c(deg_c):
    return (deg_c * _f2c_ratio) + 32.0
```

```
import temp_convert

f = 70.0
c = temp_convert.convert_f2c(f)
print "%.1f F = %.1f C" % (f, c)
```

Sample Module II

```
import csv

datafile = 'results.csv'

def load():
    results = []
    r = csv.reader(open(datafile, 'rb'))
    for row in reader:
        results.append(row)
    return results

def write(data):
    # ...
```

Standard Library

Built-In Functions

Review … and Some New Friends

```
  bool   float   int   long   str - dir   help
chr   hex   oct   ord - len   range - id   type
abs   max   min   pow   round   sum - raw_input
```

```
chr(97)    => 'a'
ord('a')   => 97

max(1, 2, 3, 4, 5)   => 5
round(2.5)           => 3.0
sum(range(1, 101))   => 5050

2 is (1 + 1)       => True
id(2), id(1 + 1) => (437788320, 437788320)
```

Sorting

Sort a `list` object *in place*:

```
list.sort(cmp, key, reverse)    # all opt.
```

```
names = ['Tim', 'Alain', 'scot', 'mat']
def lcmp(x, y):
    return cmp(x.lower(), y.lower())
names.sort()
names.sort(cmp=lcmp)
names.sort(key=str.lower, reverse=True)
```

Sort any iterable object and make new `list` object:

```
new_list = sorted(list, cmp, key, reverse)
```

Transforming

```
map(function, iterable, ...)
```

```
>>> map(chr, range(65, 91))
['A', 'B', 'C', …, 'Y', 'Z']
```

```
>>> map(str.capitalize, ('tim', 'mAt', …))
['Tim', 'Mat', …]
```

```
def mult(x, y):
    return x * y

a = (2, 3, 6, 4, 8)
b = (12, 8, 4, 6, 3)

twentyfour = map(mult, a, b)
```

Filtering

```
filter(function, iterable)
```

```
>>> t = ('Hi!', '42', '4.0', '-1', '1001')
>>> filter(str.isdigit, t)
('42', '1001')
```

```
def large(t):
    return t[1] > 100.0

p = {'cn': 1339.7, 'ph': 94.0, 'dk': 5.6,
     'us': 312.5, 'in': 1210.2, 'ki': 0.1}

big = dict(filter(large, p.items()))
```

Numbers

math

```
   pi   e   -   ceil   floor   fmod   fabs
  cos   sin   tan  -  degrees   radians
 exp   factorial   log   log10   pow   sqrt
```

```
2 * math.pi             => 6.283185307179586
math.log(math.e)        => 1.0

math.ceil(2.5)          => 3.0
math.floor(2.5)         => 2.0

math.cos(1.0)           => 0.5403023058681398
math.degrees(0.54)      => 30.939720937064457

math.sqrt(90.0)         => 9.486832980505138
```

decimal

- Does ***exact*** decimal floating point math
- Has pros *and* cons compared to regular floats
- Very useful for fixed-decimal math (e.g., finance)

```
>>> 0.1 + 0.1 + 0.1 - 0.3
5.551115123125783e-17

>>> import decimal
>>> Decimal('0.1') + Decimal('0.1') + \
... Decimal('0.1') - Decimal('0.3')
Decimal('0.0')
>>> print Decimal('0.0')
0.0
```

random

```
random  uniform  gauss  normalvariate  …
randint  choice  shuffle  sample - seed
```

```
random.random() => 0.26510474670383377
random.random() => 0.09321673054441892

random.normalvariate(100, 10) => 95.3900…

random.randint(1, 100) => 42

x = range(1, 6)
random.shuffle(x)
x => [2, 4, 1, 5, 3]

random.choice(['A', 'B', 'C', 'D']) => 'A'
```

Dates and Times

time

```
time - localtime  gmtime
asctime  strftime - mktime  strptime
```

```
>>> time.time()
1319402100.752616

>>> time.localtime()
time.struct_time(tm_year=2011, tm_mon=10, tm_mday=23, tm_hour=15,
tm_min=35, tm_sec=20, tm_wday=6, tm_yday=296, tm_isdst=1)

>>> time.asctime()
'Sun Oct 23 15:36:51 2011'

>>> time.strftime('%d %B %Y - %H:%M')
'23 October 2011 - 15:36'
```

datetime

- Complex set of classes for date/time calculations
- Extra code required to handle timezones correctly

```
from datetime import datetime, timedelta

start = datetime.now()
wait = timedelta(weeks=1, hours=8)
then = start + wait

print then.strftime('%m/%d @ %H:%M')
```

calendar

- Mostly for printing fixed-width text calendars
- Does a few other date calculations

```
>>> calendar.prmonth(2011, 11)
    November 2011
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

Files

csv

```
Tim Cartwright,4265,2-4002,"CHTC, OSG"
Alain Roy,4261,5-5736,"CHTC, OSG"
Greg Thain,4279,0-0031,CHTC
```

- Used by Excel and others
- Wildly inconsistent, non-standardized format
- Hard to parse yourself, use the module instead
- Can offer hints as to particular CSV format

```
file = open(csv_file, 'rb')
reader = csv.reader(file)
for row in reader:
    # row is list, 1 item/CSV field
```

fnmatch

```
fnmatch  fnmatchcase  filter
```

```
file = raw_input('Enter filename: ')
if not fnmatch.fnmatch(file, '*.html'):
    print 'Please enter an HTML file!'
```

```
files = os.listdir('.')
for file in fnmatch.filter(files, '*.py'):
    print 'Python file:', file
```

glob

- Acts like shell wildcards
- Uses **`fnmatch`** for implementation

```
for pth in glob.glob('travel/201?/*.csv'):
    print 'Recent travel table:', pth
    file = open(pth, 'rb')
    reader = csv.reader(file)
    # ...
```

The System

sys

```
stdin  stdout  stderr - platform  version
  getrefcount - modules - path  prefix
getsizeof  float_info  long_info  maxint
```

plus lots more!

```
sys.platform       => 'linux2'
sys.version_info   => (2, 4, 3, 'final', 0)

sys.getrefcount(2)       => 33
sys.getrefcount(1234)    => 2

sys.maxint               => 2147483647

len(sys.modules)         => 30
```

os

```
chdir  getcwd - readlink  symlink  unlink
  chmod  chown  stat - remove  rename
 makedirs  mkdir  rmdir - listdir  walk
```

plus lots more!

```
for t in os.walk(path):
  # t = (dirname, subdirs, files)
  for f in t[2]:
    if fnmatch.fnmatch(f, '*.py'):
      print 'Py:', os.path.join(t[0], f)
```

os.path

```
abspath  realpath
basename  dirname  join  split
exists  isdir  isfile  islink
getatime  getctime  getmtime  getsize
```

```
dir = 'wibble/wobble'
file = raw_input('File? ')
path = os.path.join(dir, file)
if os.path.exists(path) and \
   os.path.getsize(path) > 0:
    print 'Path:', os.path.abspath(path)
    print 'Size:', os.path.getsize(path)
    print 'Mod:', os.path.getmtime(path)
```

shutil

```
copy   copy2   copytree
copymode   copystat - move - rmtree
```

```
shutil.copy('a.txt', 'b.txt')
shutil.copy2('a.txt', 'a.txt.backup')
shutil.move('dir1/a.txt', 'dir2/a.txt')
shutil.rmtree('be/really/careful')
```

Last 2 Slides!

Other Scripting Languages

- **Modules**
  - Code organization systems are fairly universal
  - Syntax, semantics, etc., vary considerably!
- **Standard Library**
  - Yes!
  - Part of what makes a language (not) suitable for a task
  - Huge variation

Homework

- Write source files to multiple target directories
- May modify file contents each time
- Use lots of standard Python modules/functions!!!
- [Optional:] Write a simple module for some of sol'n
- Be sure to label your printout

```
#!/usr/bin/env python

"""Homework for CS 368-2 (2012 Spring)
Assigned on Day 06, 2012-03-29
Written by <Your Name>
"""
```