Frequently Used Python Modules

sys, os, shelve, math, re

sys: System specific utilities

1. argv # argv[0] – name of program
2. exit([*arg*])
3. stdin, stdout, stderr
4. byteorder # 'big' or 'little'
5. getrecursionlimit() / setrecursionlimit(*n*)
6. maxint
7. path # for modules; path[0]: directory with python script
8. platform # e.g. 'sunos5' or 'linux1'
9. ps1, ps2
10. version

Examples

```
import sys
sys.stdout = open('try.txt','w')
print sys.version, sys.platform
sys.exit(0)
```

or

```
from sys import *
# stdout = open('try.txt','w')
f = open('try.txt','w')
print >> f, version, platform
exit(0)
```


os: OS-related utilities

1. environ # environment variables. e.g. environ['HOME']
2. system(*command*)
3. chdir(*path*)
4. getcwd()
5. l = listdir('/etc')
6. f = tmpfile()
7. remove(*filepath*), unlink(*filepath*), rmdir(*path*)

Many other commands for file/directory/pipe manipulation, process management (killing processes, etc.)

os.path: path-related utilities

1. abspath(*path*) # abspath('../file1'): '/home/file1'
2. basename(*path*) # e.g. basename('/a/b/c') = 'c'
3. dirname(*path*) # e.g. dirname('/a/b/c') = '/a/b'
4. exists(*path*), isfile(*path*), isdir(*path*)
5. getsize(*path*)

If you find yourself using too many os and os.path features, consider writing a shell-script instead.

shelve: A persistent dictionary

```
import shelve
s=shelve.open('phones.txt')
s['Ram'] = ['080-223344']
s['Shyam'] = ['040-556677', 'hyd']
# s['Ram'].append('bangalore') #error
temp = s['Ram']
temp.append('bangalore')
s['Ram'] = temp
```

math, cmath

1. sin(x), cos(x), tan(x),
   asin(x), acos(x), atan(x), # arc sine, etc.
   sinh(x), cosh(x), tanh(x) # hyperbolic
2. ceil(x), floor(x)
3. exp(x), log(x[, *base*])
4. fabs(x) # absolute value
5. pi, e # constants

cmath provides same functions that work for complex numbers also.

Strings

```
dirs = ['', 'usr', 'bin', 'ls']
'/'.join(dirs)
# output: '/usr/bin/ls'
s = 'John smith'
print s.upper(), s.lower(), capwords(s)
```

Strings also support other operations like *simple* substitutions, searching for substrings, etc. But you may as well use re for those tasks.

re: Regular Expressions

- Python and some other tools (e.g. egrep) support extensions to regexps.
  - `"python|perl"`
  - `"p(ython|erl)"`
  - `(ab)*`
  - `(ab)+`
  - `(ab){2,4}`
  - `"^The"`
  - `"^$"`

re functions

1. search(*pattern*, *string*)
   - Search for pattern in string
2. match(*pattern*, *string*)
   - Matches pattern at *beginning* of string
3. split(*pattern*, *string*[,*maxsplit*=0])
   - Splits a string by occurances of pattern
4. findall(*pattern*, *string*) # returns a list
5. sub(*pat*, *repl, string*[,*count*=0])
6. escape(*string*) # escapes special re chars
7. compile(*pattern, flags*)

Examples

```
if (re.search('edu', url)): print "edu"
a = '1, 2,, 3,,, 4'
re.split('[, ]+', a)
for i in open('/etc/passwd'):
  x = split(':',i)
  print x[0], x[2] # login, uid
pat = "{name}"
text = "Dear {name},…"
text2 = re.sub(pat, "Mr.Ashok", text)
pat = '[a-zA-Z]+'
re.findall(pat, text)
pat = re.compile('<ol>.*?</ol>', re.M|re.S)
  # re.M: Multi-line; S: . matches \n also
t2 = re.sub(pat,'<ol></ol>',t)
```


Groups and Match Objects

search() and match() return match objects (evaluating to true) if a match is found, and None, otherwise

```
pat = 'a(bc(d)(e))f(g)'
m = re.match(pat, 'abcdefghij')
# group 0 is entire pattern
# group n is pattern within nth brackets
#  -> order of brackets = order of '('
print m.group(1) # output: 'bcde'
print m.start(1) # output: 1
print m.end(1)   # output: 5
print m.span(1)  # output: (1, 5)
```

Example

- To get the middle element from a url such as www.python.org

```
m = re.match(r'www\.(.*)\..{3}',
  'www.python.org')
print m.group(1)
```


Greedy and Non-greedy Patterns

```
emphasis_pat = r'\*(.+)\*'
replacement = r'<em>\1</em>'
text = '*This* is *it*!'
re.sub(emphasis_pat, replacement, text)
   # greedy output: '<em>This* is *it</em>!'
# To make it non-greedy:
emphasis_pat = r'\*(.+?)\*'
# Note: replacement can also be a function that
   takes a match object and returns a string
```


Templating Example

- We are given text like:
  - "Sum of 7 + 9 is [7+9]"
  - "[name='Mr.Xyz'] Hello, [name]"
- We need to evaluate or execute all expressions within [and] using python
- Substitute all such expressions with their evaluated values
- Applications:
  - mass email
  - mixing python code in html
  - mixing python code in bash

8-line Solution

```
scope = {} # variables, values
def replacement(match):
  code = match.group(1)
  try: # is it an expression?
     return str(eval(code,scope))
  except SyntaxError:#its a statement
     exec code in scope
     return ""
```