PYTHON LAB BOOK

Python For Programmers
UCSC Extension Online

Lab 12 Dynamic Code

Topics

- Dynamic Code Generation
- Modules:
- `subprocess`
- `glob`
- `profile`

v.4.0

lab11_1.py

```
 1 #!/usr/bin/env python
 2 """lab11_1.py
 3 Write a function that reads a file and finds all the
 4 numbers in the file and adds them up.
 5 """
 6
 7 import sys
 8 import os
 9
10 if __name__ == '__main__':   # Put apple first on the search path.
11     sys.path.insert(0, "..") # But, if you might want to import
12 else:                        # this module, you need all this.
13     sys.path.insert(0, os.path.join(os.path.split(__file__)[0], '..'))
14
15 import banana.total_text     # banana must have __init__.py
16
17 def TotalIt(stream, total=0):
18     """Returns the sum of all the numbers in stream, which is an open
19     file object."""
20     for line in stream:
21         total += banana.total_text.TotalText(line)
22     return total
23
24 def TotalFile(name):
25     """Returns the sum of all the numbers in the file."""
26     try:
27         open_file = open(name)
28         try:
29             return TotalIt(open_file)
30         finally:
31             open_file.close()
32     except IOError:
33         print "I can't read '%s'." % (name)
34
35 def main():
36     while True:
37         try:
38             name = raw_input('File name: ')
39         except (KeyboardInterrupt, EOFError):
40             print
41             break
42         if name == '':
43             break
44         total = TotalFile(name)
```

```
45         if total:
46             print name, 'totals to', total
47
48 if __name__ == '__main__':
49     main()
50
51 """
52 $ lab11_1.py
53 File name: ../../numbers.txt
54 ../../numbers.txt totals to 117.5
55 File name: no_file
56 I can't read 'no_file'.
57 File name:       (I hit Ctrl-D -->EOFError)
58 $ lab11_1.py
59 File name:       (I hit Ctrl-c Ctrl-c -->KeyboardInterrupt)
60 $ cat ../../numbers.txt
61 Here is 1. Add 2 makes 3 or maybe 12, depending on how you operate.
62 You might like 2.2 and that's enough unless you like "8.8" or maybe
63 1 more or maybe 87. . .5
64 $
65 """
```

lab11_2.py

```
#!/usr/bin/env python
"""lab11_2.py
Using os.path.walk to accumulate statistics through the files walked."""

import os

import apple.work_here_.lab11_1 as totaler

def TotalDeep(stats, dir_name, files):
    """Called by walk to return statistics into stats.
    stats = [number_of_files, total]"""
    print dir_name, "so far: %d files, adding to %d." % (stats[0], stats[1])
    for file in files:
        pname = dir_name + os.sep + file
        if not os.path.isfile(pname):
            continue
        try:
            stats[1] += totaler.TotalFile(pname)
            stats[0] += 1
        except IOError, msg:
            print pname, msg

def main():
    """stats is passed into walk for accumulating statistics.  Note that
    stats must be mutable for this to work."""
    stats = [0, 0]  # number_of_files, total
    os.path.walk('..', TotalDeep, stats)
    print "That's %d files, totaling to %d." % tuple(stats)

if __name__ == '__main__':
    main()

"""
$ lab11_2.py
./lab11_2.py
.. so far: 0 files, adding to 0.
../lab_06_Sequences so far: 1 files, adding to 0.
../lab_06_Sequences/.svn so far: 12 files, adding to 1235.
[much skipped]
That's 860 files, totaling to 7163428090.
$ """

```

lab11_3.py

```
 1 #!/usr/bin/env python
 2 """lab11_3.py tree command in python"""
 3 import os
 4
 5 def GatherFiles(node_d, dirname, fnames):
 6     node_d[dirname] = 'directory'
 7     for f in fnames:
 8         f_name = os.path.join(dirname, f)
 9         if os.path.isfile(f_name):
10             node_d[f_name] = 'file'
11             print f_name
12
13 def Tree(start_at):
14     node_d = {}
15     os.path.walk(start_at, GatherFiles, node_d)
16     directories = 0
17     files = 0
18     for node in sorted(node_d):
19         path, name = os.path.split(node)
20         slashes = path.count(os.sep)
21         print "  | " * slashes,
22         if path:
23             print " |--",
24         if node_d[node] == 'directory':
25             print os.sep + name
26             directories += 1
27         else:
28             print name
29             files += 1
30     print
31     print "%d directores, %d files" % (directories, files)
32
33 def main():
34     start_at = raw_input("Tree to start at which directory? ")
35     Tree(start_at)
36
37 if __name__ == '__main__':
38     main()
39
40 """$ ./lab11_3.py
41 Tree to start at which directory? cats
42  /cats
43   |-- cats.txt
44   |-- /deep_cats
```

```
45   |   |-- cats.txt
46   |   |-- /deeper_cats
47   |   |   |-- cats.txt
48   |   |   |-- more_cats.txt
49   |   |-- more_cats.txt
50   |-- more_cats.txt
51
52 3 directores, 7 files
53 $ """
54
```

copies.py

```
 1 #!/usr/bin/env python
 2 """copies.py
 3
 4 Demonstrating shallow and deep copies.  With dict.copy(), you get a
 5 shallow copy where the dictionary values are references of the same
 6 values that are in the original dictionary.
 7 """
 8
 9 import copy
10
11 nest = {'a':[1,2,3], 'b':[11,12,13]}
12 nest_copy = nest.copy()
13 print '     nest:', nest
14 print
15 print "Hopefully, if you change one, you don't change the other."
16 nest['b'] = [21, 22, 23]
17 print "After nest['b'] = [21, 22, 23]"
18 print '     nest:', nest
19 print 'nest_copy:', nest_copy
20 print
21 print "OK.  That worked.  But what if you change an element of a list,"
22 print "because the copy has a reference to the list, "\
23       "both reflect the change."
24 nest['a'][1] = 'surprise'
25 print "After nest['a'][1] = 'surprise'"
26 print '     nest:', nest
27 print 'nest_copy:', nest_copy
28 print
29 print "If you don't like that behavior, you can do a 'deepcopy'."
30 deep_copy = copy.deepcopy(nest)
31 nest['a'][1] = 'independence'
32 print "After nest['a'][1] = 'independence'"
33 print '     nest:', nest
34 print 'deep_copy:', deep_copy
35 """
36
37
38
39
40
41
42
43
44
```

```
45
46 $ copies.py
47      nest: {'a': [1, 2, 3], 'b': [11, 12, 13]}
48
49 Hopefully, if you change one, you don't change the other.
50 After nest['b'] = [21, 22, 23]
51      nest: {'a': [1, 2, 3], 'b': [21, 22, 23]}
52 nest_copy: {'a': [1, 2, 3], 'b': [11, 12, 13]}
53
54 OK.  That worked.  But what if you change an element of a list,
55 because the copy has a reference to the list, both reflect the change.
56 After nest['a'][1] = 'surprise'
57      nest: {'a': [1, 'surprise', 3], 'b': [21, 22, 23]}
58 nest_copy: {'a': [1, 'surprise', 3], 'b': [11, 12, 13]}
59
60 If you don't like that behavior, you can do a 'deepcopy'.
61 After nest['a'][1] = 'independence'
62      nest: {'a': [1, 'independence', 3], 'b': [21, 22, 23]}
63 deep_copy: {'a': [1, 'surprise', 3], 'b': [21, 22, 23]}
64 $"""
```

dynamic.py

```
 1 #!/usr/bin/env python
 2 """Demonstrates the exec statement and eval function, used
 3 for dynamic code generation.
 4 """
 5 import sys
 6
 7 VARIABLES = ("name", "zip", "phone", "SSN")
 8
 9 def GetVariables(variables):
10     for each in variables:
11         answer = raw_input("%s please: " % each)
12         exec "%s = '%s'" % (each, answer) in globals()
13
14 def PrintVariables(variables):
15     for each in variables:
16         print each, '=', eval(each)
17
18 def main():
19     if len(sys.argv) > 1:
20         variables = sys.argv[1:]
21     else:
22         variables = VARIABLES
23     GetVariables(variables)
24     PrintVariables(variables)
25
26 if __name__ == '__main__':
27     main()
28 """
29 $ dynamic.py name money_in_pocket
30 name please: Linda
31 money_in_pocket please: $1.25
32 name = Linda
33 money_in_pocket = $1.25
34 $ dynamic.py
35 name please: Marilyn
36 zip please: 94043
37 phone please: 650 814-4435
38 SSN please: XXX-XX-XXXX
39 name = Marilyn
40 zip = 94043
41 phone = 650 814-4435
42 SSN = XXX-XX-XXXX
43 $"""
```

dynamic2.py

```
 1 #!/usr/bin/env python
 2 """
 3 Demonstrates the setattr and getattr functions for dynamic
 4 code generation, which is preferred to exec and eval.
 5
 6 The first argument to setattr and getattr is the namespace
 7 where you expect the variable to land.  sys.modules is
 8 helpful here if you want it in the current namespace.
 9 """
10 import sys
11
12 VARIABLES = ("name", "zip", "phone", "SSN")
13
14 def GetVariables(variables):
15     for each in variables:
16         answer = raw_input("%s please: " % each)
17         setattr(sys.modules[__name__], each, answer)
18
19 def PrintVariables(variables):
20     for each in variables:
21         print each, '=', getattr(sys.modules[__name__], each)
22
23 def main():
24     if len(sys.argv) > 1:
25         variables = sys.argv[1:]
26     else:
27         variables = VARIABLES
28     GetVariables(variables)
29     PrintVariables(variables)
30
31 if __name__ == '__main__':
32     main()
33
34 """
35 Same output.
36 """
```

piper.py

```
#!/usr/bin/env python
"""piper.py -- demonstrates running a shell-level command.  Stdout is
collected and piped into a file object which can be read as if it was
an open file.
"""
import sys
if __name__ == '__main__':
    sys.path.insert(0, "..")
else:
    sys.path.insert(0, os.path.join(os.path.split(__file__)[0], '..'))
import lab_10_Files.apple.banana.total_text as total_text
import subprocess

def Total_ps():
    """Returns the sum of all the numbers in a list
    of the processes running."""

    open_pipe = subprocess.Popen(["ps", "-ef"],
                                 stdout=subprocess.PIPE).stdout
    try:
        return total_text.TotalText(open_pipe.read())
    finally:
        open_pipe.close()

if __name__ == '__main__':
    print "Your lucky number:", Total_ps()

"""
$ piper.py
Your lucky number: 1055528.0
$
"""
```

find_.py

```
#!/usr/bin/env python
"""
find_.py starting_dir pattern

(be sure to escape the pattern: \*.py.  Or put it in quotes: '*.py')

finds the files in the directories starting at starting_dir that match
the pattern.  Demonstrates the glob module.

The glob module provides a way to find files that match a given
pattern with simple shell-style wildcards.

Here we use glob and walk to find all the files in a directory
structure that match the pattern, just like the 'find' command in unix.
"""
import glob
import os
import sys

def Finder(pattern, dirname, fnames):
    """Finds files that match the pattern in the dirname.
    fnames is ignored."""
    result = glob.glob(os.path.join(dirname, pattern))
    if result:
        print dirname  + ':'
        dlen = len(dirname)
        for each in result:
            print '   ' + each[dlen:]

def FindDeep(starting_dir, pattern):
    os.path.walk(starting_dir, Finder, pattern)

def main():
    if len(sys.argv) == 3:
        FindDeep(*sys.argv[1:])
    else:
        print __doc__

if __name__ == '__main__':
    main()
"""
$ find_.py .. "*.py"
../lab_06_Sequences:
   /key_sort.py    [much deleted]"""
```

prof.py

```
 1 #!/usr/bin/env python
 2 """prof.py Demonstrates the profiler which spits out info about the
 3 time it takes to run functions."""
 4
 5 __pychecker__ = 'no-local'
 6
 7 LIMIT = 10
 8 data = range(LIMIT)
 9
10 def TryWay(i):
11     try:
12         return data[i]
13     except:
14         return None
15
16 def TestWay(i):
17     if i < -len(data) or i > len(data) - 1:
18         return None
19     return data[i]
20
21 def TestWay2(i):
22     data_len = len(data)
23     if i < -data_len or i > data_len - 1:
24         return None
25     return data[i]
26
27 def TestThem(n):
28     for i in range(n): # pychecker complains that i is unused
29         TryWay(LIMIT * 2)
30         TestWay(LIMIT * 2)
31         TestWay2(LIMIT * 2)
32
33 if __name__ == '__main__':
34     import profile
35     profile.run('TestThem(10000)')
36 """
37
38
39
40
41
42
43
44
```

```
45
46
47
48
49 $ prof.py
50          60005 function calls in 0.596 CPU seconds
51
52    Ordered by: standard name
53
54    ncalls  tottime  percall  cumtime  percall filename:lineno(function)
55     30000    0.128    0.000    0.128    0.000 :0(len)
56         1    0.008    0.008    0.008    0.008 :0(range)
57         1    0.000    0.000    0.000    0.000 :0(setprofile)
58         1    0.000    0.000    0.596    0.596 <string>:1(<module>)
59     10000    0.156    0.000    0.236    0.000 prof.py:13(TestWay)
60     10000    0.124    0.000    0.172    0.000 prof.py:18(TestWay2)
61         1    0.128    0.128    0.596    0.596 prof.py:24(TestThem)
62     10000    0.052    0.000    0.052    0.000 prof.py:7(TryWay)
63         0    0.000             0.000          profile:0(profiler)
64         1    0.000    0.000    0.596    0.596 profile:0(TestThem(10000))
65 $      """
```

Lab 12 Optional

1. Find `lab_12_Dynamic_Code/soccer_team.py`. Note that the `ProcessTeam` and `PrintTeam` functions have calls to `eval` and `exec`. Replace them with calls to `setattr` and `getattr`.

   The data are in `lab_12_Dynamic_Code/Bees`.

2. The command to read a the current directory's file listing is:

   ```
   unix:  ls -l          windows: dir
   ```

   but the first argument to `subprocess.Popen`

   ```
   ['ls', '-l']          ['cmd', '/c', 'dir']
   ```

   Parse the file sizes and add them up. Use `os.path.getsize()` to do the same thing. You might use: `lab_11_Packages/apple/banana/total_text.py` Profile both methods.

3. On windows the call:

   ```
   c:\windows\system32\ipconfig.exe /all
   ```

   or on unix:

   ```
   ifconfig
   ```

   gives a lot of info about your network. Read this into a program that prints out your IP address.

   On unix, it is called `inet addr` in the `ifconfig` report. On windows, it is called `IP address`

soccer_team.py

```
 1 #!/usr/bin/env python
 2 """Processing team data using exec and eval."""
 3
 4 def NotifyForwards():
 5     return "Go for the goal!"
 6
 7 def NotifyDefenders():
 8     return "Block that kick!"
 9
10 def NotifyMidfielders():
11     return "Get that ball!"
12
13 def NotifyGoalies():
14     return "Guard the goal!"
15
16 def ProcessTeam(stream):
17     positions = []
18     for line in stream:
19         line = line.strip()
20         if not line:
21             continue
22         if line.endswith(':'):
23             position = line[:-1]
24             exec "%s = []" % (position) in globals()
25             positions += [position]
26             continue
27         details = line.split(' ', 1)
28         exec "%s += [details]" % (position)
29         exec "print 'Yeh %s #%s ' + Notify%s()" %\
30         (details[1], details[0], position)
31     return stream.name, positions
32
33 def PrintTeam(team_name, positions):
34     print '\n%s:' % team_name
35     for each in positions:
36         print '  %s:' % each
37         for player in sorted(eval(each)):
38             print '    ' + ': '.join(player)
39
40 def main(team_name = "Bees"):
41     team_name, positions = ProcessTeam(open(team_name))
42     PrintTeam(team_name, positions)
43
44 if __name__ == '__main__':
```

```
45     main()
46 """
47 $soccer_team.py
48 Yeh Bruce Penge #7 Go for the goal!
49 Yeh Maureen Mezzabo #1 Go for the goal!
50 Yeh Samantha Smith #8 Go for the goal!
51 Yeh Juvenal Ramirez #6 Go for the goal!
52 Yeh Xavier Perra #4 Get that ball!
53 Yeh Laura Dot #2 Get that ball!
54 Yeh Malcolm Diamond #5 Get that ball!
55 Yeh Mary Bart #9 Get that ball!
56 Yeh Linda Jarvis #3 Block that kick!
57 Yeh Jose Acosta #11 Guard the goal!
58 Yeh Tracy Lowe #10 Guard the goal!
59
60 Bees:
61   Forwards:
62     1: Maureen Mezzabo
63     6: Juvenal Ramirez
64     7: Bruce Penge
65     8: Samantha Smith
66   Midfielders:
67     2: Laura Dot
68     4: Xavier Perra
69     5: Malcolm Diamond
70     9: Mary Bart
71   Defenders:
72     3: Linda Jarvis
73   Goalies:
74     10: Tracy Lowe
75     11: Jose Acosta
76 $"""
```