PYTHON LAB BOOK

Python For Programmers

UCSC Extension Online

Lab 17 Developer Modules

Topics

- Context Manager class
- Module: `unittest`
- Module: `optparse`

lab16_1.py

```
 1 #!/usr/bin/env python
 2 """lab16_1.py A SortedDictionary class with only "description"
 3 allowed as an attribute -- using __setattr__"""
 4
 5 class SortedDictionary(dict):
 6
 7     allowed_attributes = 'description',
 8
 9     def keys(self):
10         return sorted(dict.keys(self))
11         # more robust is:
12         # return sorted(super(type(self), self).keys())
13
14     def __iter__(self):
15         """If we don't define this, it will use the regular dictionary
16         __iter__ which does not call SortedDictionary.keys()."""
17
18         for each in self.keys():
19             yield each
20
21     def __setattr__(self, attribute_name, value):
22         if attribute_name not in SortedDictionary.allowed_attributes:
23             raise TypeError, "can't set attributes of class %s" \
24                   % self.__class__.__name__
25         self.__dict__[attribute_name] =  value
26
27 def main():
28     for d in ( {'Zero':0, 'False':0, 'None':0, 'True':1}, # dictionary
29                {},                                    # empty dictionary
30                (('calling birds', 4), ('french hens', 3), # tuples
31                 ('turtle doves', 2), ('partridge in a pear tree', 1))
32                ):
33         sorted_dict = SortedDictionary(d)
34         regular_dict = dict(d)
35         print "regular_dict:", regular_dict.keys()
36         print " sorted_dict:", sorted_dict.keys()
37         print "         for:", ', '.join([str(k) for k in sorted_dict])
38
39     sorted_dict.description = "Fourth Day of Christmas"
40     print "sorted_dict.description =", sorted_dict.description
41     try:
42         regular_dict.description = "Fourth Day of Christmas"
43     except AttributeError:
44         pass
```

```
45     else:
46         print "Unexpected behavior!"
47     sorted_dict.x = 3
48
49 if __name__ == '__main__':
50     main()
51
52 """
53 $ lab16_1.py
54 regular_dict: ['False', 'Zero', 'True', 'None']
55  sorted_dict: ['False', 'None', 'True', 'Zero']
56          for: False, None, True, Zero
57 regular_dict: []
58  sorted_dict: []
59          for:
60 regular_dict: ['turtle doves', 'french hens',\
61                 'partridge in a pear tree', 'calling birds']
62  sorted_dict: ['calling birds', 'french hens',\
63                 'partridge in a pear tree', 'turtle doves']
64          for: calling birds, french hens, \
65               partridge in a pear tree, turtle doves
66 sorted_dict.description = Fourth Day of Christmas
67 Traceback (most recent call last):
68   File "./lab16_1.py", line 58, in <module>
69     main()
70   File "./lab16_1.py", line 55, in main
71     sorted_dict.x = 3
72   File "./lab16_1.py", line 32, in __setattr__
73     % self.__class__.__name__
74 TypeError: can't set attributes of class SortedDictionary
75 $ """
```

lab16_2.py

```
 1 #!/usr/bin/env python
 2 """lab16_2.py A Money class"""
 3 import sys
 4 if __name__ == '__main__':
 5     sys.path.insert(0, "..")
 6 else:
 7     sys.path.insert(0, os.path.join(os.path.split(__file__)[0], '..'))
 8
 9 import lab_08_Comprehensions.lab08_4 as make_money_string
10
11 class Money(float):
12
13     def __add__(self, other):
14         return Money(float.__add__(self, other))
15
16     def __div__(self, number):
17         return Money(float.__div__(self, number))
18
19     def __rmul__(self, number):
20         return Money(float.__mul__(self, number))
21
22     def __mul__(self, number):
23         return Money(float.__mul__(self, number))
24
25     def __neg__(self):
26         return Money(float.__neg__(self))
27
28     def __repr__(self):
29         return """Money('%f')""" % self
30
31     def __str__(self):
32         return make_money_string.MakeMoneyString(self)
33
34     def __sub__(self, other):
35         return Money(float.__sub__(self, other))
36
37 def main():
38     print Money(-123.21)
39     print Money(40.50)
40     print Money(-1001.011)
41     print Money(123456789.999)
42     print Money(.10)
43     print Money(.01)
44     print Money(.055)
```

```
45     print 'add:', Money(10) + Money(20), '==', Money(30)
46     print 'repr:', eval(repr(Money(44.123))), '==', Money(44.123)
47     print 'sub:', Money(44.333) - Money(55.444), '==', Money(-11.111)
48     print 'neg:', -Money(10.00), '==', Money(-10.00)
49     print 'mult:', 2 * Money(-11.11), '==', Money(-22.22), \
50           '==', Money(11.11) * -2
51     print 'div:', (Money(44.44))/4, '==', Money(11.11)
52
53 if __name__ == '__main__':
54     main()
55
56 """
57 $ lab16_2.py
58 -$123.21
59 $40.50
60 -$1,001.01
61 $123,456,790.00
62 $0.10
63 $0.01
64 $0.06
65 add: $30.00 == $30.00
66 repr: $44.12 == $44.12
67 sub: -$11.11 == -$11.11
68 neg: -$10.00 == -$10.00
69 mult: -$22.22 == -$22.22 == -$22.22
70 div: $11.11 == $11.11
71 $ """
```

context.py

```
 1 #!/usr/bin/env python
 2 """Making a context manager as a class from scratch."""
 3
 4 import sys
 5
 6 class OpenClose:
 7
 8     def __init__(self, file_name, mode="r"):
 9         self.file_name = file_name
10         self.mode = mode
11
12     def __enter__(self):
13         self.obj = open(self.file_name, self.mode)
14         return self.obj
15
16     def __exit__(self, exc_type, exc_val, exc_tb):
17         self.obj.close()
18
19 def PrintFile(file_name, fail_on_read=False):
20     try:
21         with OpenClose(file_name) as file_object:
22             for line in file_object:
23                 print line,
24                 if fail_on_read:
25                     raise IOError, "Failed while reading."
26     except IOError, msg:
27         print msg
28
29 def main(file_name="ram.tzu"):
30     print """\n    PrintFile("%s")""" % (file_name)
31     PrintFile(file_name)
32     print """\n    PrintFile("%s", fail_on_read=True)""" % (file_name)
33     PrintFile(file_name, fail_on_read=True)
34     print """\n    PrintFile("absent_file")"""
35     PrintFile("absent_file")
36
37 if __name__ == '__main__':
38     main()
39
40 """
41 $ ./context.py > context.out
42 $ diff context.out ../lab_10_File_IO_and_Packages/file2.out
43 $ """
```

context2.py

```
 1 #!/usr/bin/env python
 2 """Using the OpenClose context manager class. """
 3
 4 import context
 5
 6 def WriteFile(file_name, text):
 7     try:
 8         with context.OpenClose(file_name, "w") as file_object:
 9             file_object.write(text)
10     except IOError, msg:
11         print msg
12
13 if __name__ == '__main__':
14     WriteFile("sometimes", "Sometimes you win.\n")
15
16 """
17 $ context2.py
18 $ cat sometimes
19 Sometimes you win.
20 $
21 """
22
```

pyunit.py

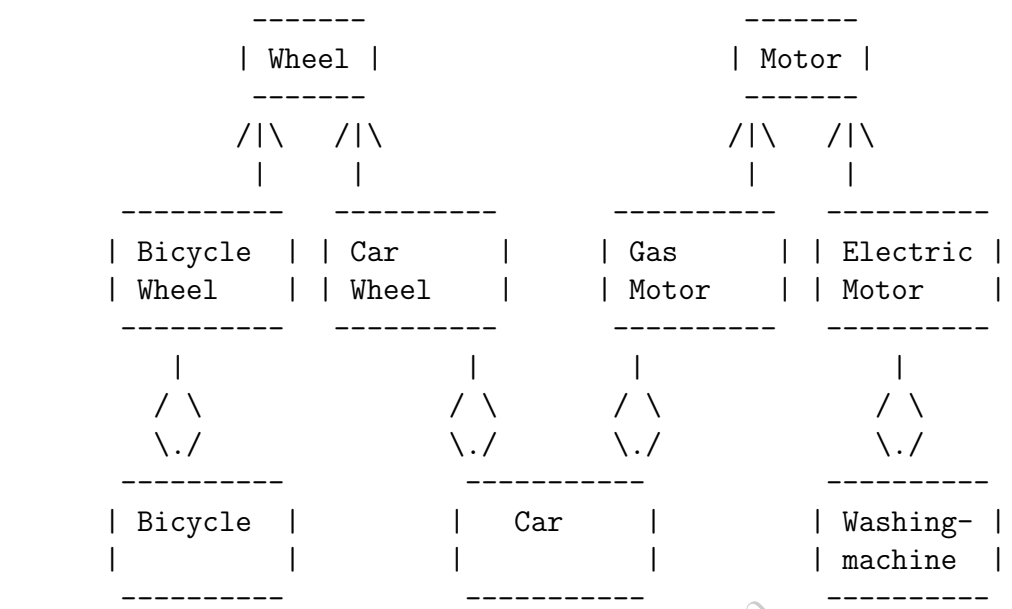
```
#!/usr/bin/env python
"""This example of unittest is taken from
http://www.python.org/doc/lib/module-unittest.html."""

import random
import unittest

class TestSequenceFunctions(unittest.TestCase):

    def setUp(self):
        self.seq = range(10)

    def testShuffle(self):
        # make sure the shuffled sequence does not lose any elements
        random.shuffle(self.seq)
        self.seq.sort()
        self.assertEqual(self.seq, range(10))

    def testChoice(self):
        element = random.choice(self.seq)
        self.assert_(element in self.seq)

    def testSample(self):
        self.assertRaises(ValueError, random.sample, self.seq, 20)
        for element in random.sample(self.seq, 5):
            self.assert_(element in self.seq)

if __name__ == '__main__':
    unittest.main()
"""
$ ./pyunit.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.006s

OK
$
"""
```

parser.py

```
 1 #!/usr/bin/env python
 2 """parser.py -f filename word-to-count [-q]
 3 Counts the number of times word-to-count appears in filename.
 4
 5 Demonstrates the optparse module for parsing the command line options
 6 and putting them into suitably-named identifiers in a namespace.
 7 See: http://www.python.org/lib/optparse-tutorial.html
 8 """
 9 import optparse
10 import string
11
12 def CollectCommand(parser):
13     # Here we harvest the command line and check that we have an
14     # argument left-over.
15     (options, args) = parser.parse_args()
16     if len(args) != 1:
17         parser.error("I need one word.")
18     if not options.filename:
19         parser.error("No file name given.")
20     return (options, args)
21
22 def main():
23     parser = SetUpParsing()
24     (options, args) = CollectCommand(parser)
25     the_word = args[0]
26     if options.verbose:
27         print "reading %s..." % options.filename,
28     count = ProcessFile(options.filename, the_word)
29     if options.verbose:
30         print "    %d occurances of '%s'" % (count, the_word)
31
32 def ProcessFile(filename, word):
33     count = 0
34     for line in file(filename):
35         count += [x.strip(string.punctuation) \
36                   for x in line.split()].count(word)
37     return count
38
39 def SetUpParsing():
40     # Here we call add_option repeatedly, once for every unix-style
41     # option we need for."""
42
43     parser = optparse.OptionParser(
44         """%prog -f filename [-q][-v=False] word
```

```
45     Counts the number of times word-to-count appears in filename.""")
46     parser.add_option("-f", "--file", dest="filename",
47                       help="read data from FILENAME")
48     parser.add_option("-v", "--verbose",
49                       action="store_true", dest="verbose", default=False)
50     parser.add_option("-q", "--quiet",
51                       action="store_false", dest="verbose")
52     return parser
53
54 if __name__ == "__main__":
55     main()
56 """
57 $ ./parser.py -f parser.py in -v
58 reading parser.py...     5 occurances of 'in'
59
60 $ ./parser.py
61 usage: parser.py -f filename [-q] word
62 Counts the number of times word-to-count appears in filename.
63
64 parser.py: error: I need one word.
65
66 $ ./parser.py aaa
67 usage: parser.py -f filename [-q] word
68 Counts the number of times word-to-count appears in filename.
69
70 parser.py: error: No file name given.
71
72 $ ./parser.py -x bb
73 usage: parser.py -f filename [-q] word
74 Counts the number of times word-to-count appears in filename.
75
76 parser.py: error: no such option: -x
77
78 $ ./parser.py -h
79 usage: parser.py -f filename [-q] word
80 Counts the number of times word-to-count appears in filename.
81
82 options:
83   -h, --help            show this help message and exit
84   -f FILENAME, --file=FILENAME
85                         read data from FILENAME
86   -v, --verbose
87   -q, --quiet
88 $ """
```

Quiz Answers




```
class Wheel:
    pass

class BicycleWheel(Wheel):
    pass

class CarWheel(Wheel):
    pass

class Bicycle:
    number_of_bicycles = 0
    def __init__(self):
       self.wheels = [BicycleWheel() for i in range(2)]
       Bicycle.number_of_bicycles += 1

class Car:
    def __init__(self):
       self.wheels = [CarWheel() for i in range(5)]
       self.motor = GasMotor()

class WashingMachine:
    def __init__(self):
        self.motor = ElectricMotor()
```

```
class Motor:
    pass

class GasMotor(Motor):
    pass

class ElectricMotor(Motor):
    pass
```

Lab 17

Important Note: If you develop a unittest under Idle, it will, after successfully running your test, generate an error when it quits. This error has nothing to do with your code.

1. Develop a test class for either your Clock class, or your Money class, or both, as your time and energy allows.

   You can use my solutions:

   `lab_16_New_Style_Classes/lab16_2.py`

   `lab_15_Overriding/lab15.py`

2. Create a command-line program to deal card games.

   (a) Import and use:

   - `labs/lab_12_Function_Fancies/lab12_3.py`, if we studied generators.
   - `labs/lab_08_Comprehensions/lab08_2.py`, if not.
     My unittest for this project is `test_play_cards.py`, next, and at `labs/lab_17_Developer_Modules/test_play_cards.py`.

   In either case:

   - `lab17_2.py` – deals 4 hands of 5 cards each, the default.
   - `lab17_2.py -p 6 -c 3` – deals 6 hands (for 6 players) of 3 cards each

3. If you are familiar with signal handling, use:

   `labs/lab_12_Function_Fancies/timeout_decorator0.py` and introspect the `signal` module to make a TimeOut context handler so that this call works:

```
with TimeOut(2) as ticker:
    try:
       time.sleep(5)
    except TimeOut:
       print "Sleeping 5 timed out!"

with TimeOut(5) as ticker:
    try:
       time.sleep(2)
       print "Sleeping 2 didn't time out."
    except TimeOut:
       print "Timed out!"
```

test_play_cards.py

```
 1 #!/usr/bin/env python
 2 """Test for lab17_1_2.py."""
 3
 4 import unittest
 5 import sys
 6 import lab17_2
 7
 8 # A Deck object is an iterator
 9 print lab17_2.Deck()
10 whole_deck = sorted(lab17_2.Deck())
11
12 class TestPlayCards(unittest.TestCase):
13
14     def testSmall(self):
15         little = lab17_2.GameDealer(1, 1).DealGame()
16         self.assertEqual(len(little), 1)
17         self.assertEqual(len(little[0]), 1)
18         self.assert_(little[0][0] in whole_deck)
19
20     def testZilch(self):
21         self.assertEqual([], lab17_2.GameDealer(0, 1).DealGame())
22         self.assertEqual([[]], lab17_2.GameDealer(1, 0).DealGame())
23         self.assertEqual([], lab17_2.GameDealer(0, 0).DealGame())
24
25     def testWholeDeck(self):
26         all = lab17_2.GameDealer(9, 6).DealGame()
27         for hand in all:
28             self.assertEqual(len(hand), 6)
29         self.assertEqual(len(all), 9)
30         all_collapsed = reduce(lambda x,y: x + y, all)
31         all_collapsed.sort()
32         self.assertEqual(all_collapsed, whole_deck)
33
34     def testTooMany(self):
35         too_many = lab17_2.GameDealer(11, 5).DealGame()
36         too_many_collapsed = reduce(lambda x,y: x + y, too_many)
37         self.assert_('Blank' in too_many_collapsed)
38         too_many_collapsed.remove('Blank')
39         too_many_collapsed.sort()
40         self.assertEqual(too_many_collapsed, whole_deck)
41
42     def testWayTooMany(self):
43         way_too_many = lab17_2.GameDealer(11, 6).DealGame()
44         way_too_many_collapsed = reduce(lambda x,y: x + y, way_too_many)
```

```
45         self.assertEqual(len(way_too_many_collapsed), 66)
46         self.assertEqual(way_too_many_collapsed.count('Blank'), 12)
47         for i in range(12):
48             way_too_many_collapsed.remove('Blank')
49         way_too_many_collapsed.sort()
50         self.assertEqual(way_too_many_collapsed, whole_deck)
51
52 if __name__ == '__main__':
53     unittest.main()
54
55 """
56 $ test_play_cards.py
57 .....
58 ----------------------------------------------------------------------
59 Ran 5 tests in 0.006s
60
61 OK
62 $
63
64 """
```