PYTHON LAB BOOK

Python For Programmers

UCSC Extension Online

Lab 16 New Style Classes

Topics

- Useful attributes
- Iterators
- New style classes
- Attribute control (Optional)
- `property` (Optional)
- Static methods (Optional)
- Class methods (Optional)
- Diamond inheritance (Optional)

v.4.0

lab15.py

```
 1 #!/usr/bin/env python
 2 """lab15.py  A Clock class"""
 3
 4 import time
 5
 6 class Clock:
 7     """Clock() for now, or Clock(hr, min) or Clock(min) or
 8     Clock("1:20") or Clock(dict) where dict has keys 'hr' and 'min'."""
 9
10     def __init__(self, *args, **dict_args):
11         no_args = len(args)
12         if dict_args:
13             self.__get_dict_args(dict_args)
14         elif no_args <= 1:
15             if no_args == 0:
16                 # making args[0] Now
17                 args = [time.ctime()[11:16]]
18             if isinstance(args[0], str):
19                 self.hr, self.min = args[0].split(':')
20             elif isinstance(args[0], dict):
21                 self.__get_dict_args(args[0])
22             else:  # sequence or single value
23                 try:
24                     self.hr, self.min = args[0]
25                 except TypeError:
26                     self.hr = 0
27                     self.min = args[0]
28         elif no_args == 2:
29             self.hr, self.min = args
30         else:
31             raise TypeError, Clock.__doc__
32         self.__normalize()
33
34     def __add__(self, other):
35         return Clock(self.hr + other.hr, self.min + other.min)
36
37     def __cmp__(self, other):
38         return cmp(int(self), int(other))
39
40     def __eq__(self, other):
41         if cmp(self, other) == 0:
42             return True
43         return False
44
```

```
45      def __get_dict_args(self, dict_args):
46          try:
47              self.min = dict_args['min']
48              self.hr = dict_args['hr']
49          except KeyError:
50              raise TypeError, Clock.__doc__
51
52      def __int__(self):
53          return self.MinutesSince12()
54
55      def __neg__(self):
56          return Clock(-self.hr, -self.min)
57
58      def __normalize(self):
59          """Assumes that self.min and self.hr are floatable and makes
60          the values fit on a clock.
61          """
62          self.min = float(self.min) + (float(self.hr) - int(self.hr)) * 60
63          self.min = int(round(self.min))
64          self.hr = int(self.hr) + self.min//60
65          self.min %= 60
66          self.hr = 1 + (self.hr - 1) % 12
67
68      def __repr__(self):
69          return """Clock('%s')""" % str(self)
70
71      def __str__(self):
72          return "%2d:%02d" % (self.hr, self.min)
73
74      def __sub__(self, other):
75          return Clock(hr=self.hr - other.hr, min=self.min - other.min)
76
77      def MinutesSince12(self):
78          return (self.hr % 12) * 60 + self.min
79
80  def main():
81      Clock()
82      c1 = Clock(12, 59)
83      values = 0
84      for hrs in range(-2, 25, 2):
85          for mins in range(-10, 10):
86              c2 = Clock(hrs, mins)
87              assert eval(repr(c2)) == c2
88              cmp_value = int(c2)
89              assert Clock(int(c2)) == c2
```

```
 90             c_sum = c1 + c2
 91             c_diff = c1 - c2
 92             c3 = c_sum + c_diff # should be 2 * c1
 93             c4 = Clock(2* c1.hr, 2 * c1.min)
 94             assert c3 == c4
 95             c5 = -c2
 96             assert c_diff == c1 + c5
 97             values += 1
 98     hours, minutes = 2, 30
 99     clocks = [Clock(hours, minutes)]
100     clocks += [Clock((hours, minutes))]
101     clocks += [Clock("%d:%2d" % (hours, minutes))]
102     clocks += [Clock({'hr': hours, 'min': minutes})]
103     clocks += [Clock(hr=hours, min=minutes)]
104     for each in clocks[1:]:
105         assert clocks[0] == each
106     try:
107         Clock(1, 2, 3)
108     except TypeError:
109         pass
110     else:
111         print "Clock(1, 2, 3) failed to raise an error"
112
113 if __name__ == '__main__':
114     main()
115
116 """$ lab15.py
117 $"""
```

Useful Attributes

There are special built_in class attributes, available for all classes:

`your_class.__module__`	Module where your class is defined.
`your_class.__name__`	String name of your class
`your_class.__doc__`	Doc string for your class
`your_class.__dict__`	Dictionary of your class' attributes and values.
`your_class.__class__`	MetaClass of which your_class is an instance
`your_class.__bases__`	Tuple of your class' base classes

Instances have special built_in attributes:

`your_obj.__class__`	Class that instantiated your object.
`your_obj.__dict__`	Dictionary of your object's attributes and values.

These built_in functions are always available:

`globals()`	Returns a dictionary of the global attributes.
`locals()`	Returns a dictionary of the local attributes.

A *MetaClass* is the class that instantiated your class. There must be one since everything, even classes, are objects.

new_style_stack_def.py

```
#!/usr/bin/env python
"""A stack again, this time using the built-in "list" type.

A stack is just a list with a "push" method, since the list
already has a "pop".  When it is-a "list" it inherits all
the builtin facilities of the list.
"""

class Stack(list):

    def push(self, thing):
        list.append(self, thing)

if __name__ == '__main__':
    stack = Stack()
    stack.push('Gone With The Wind')
    stack.push('Maltese Falcon')
    stack.push('Fifth Element')
    print "The stack has a rather nice __str__ already:"
    print stack
    print "The stack has all the list facilities, plus the 'push':"
    print dir(stack)
    print "Sorting then popping:"
    stack.sort()
    print stack.pop()
"""
$ new_style_stack_def.py
The stack has a rather nice __str__ already:
['Gone With The Wind', 'Maltese Falcon', 'Fifth Element']
The stack has all the list facilities, plus the 'push':
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__delslice__', '__dict__', '__doc__', '__eq__', '__ge__',
 '__getattribute__', '__getitem__', '__getslice__', '__gt__',
 '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__',
 '__len__', '__lt__', '__module__', '__mul__', '__ne__', '__new__',
 '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
 '__setattr__', '__setitem__', '__setslice__', '__str__',
 '__weakref__', 'append', 'count', 'extend', 'index', 'insert', 'pop',
 'push', 'remove', 'reverse', 'sort']
Sorting then popping:
Maltese Falcon
$
"""
```

new_style_circle_def.py

```
 1 #!/usr/bin/env python
 2 """A Circle class, derived from the builtin list class.
 3
 4 All the facilities of a list are available for free,
 5 for using or overridding.
 6 """
 7 import sys
 8 if __name__ == '__main__':
 9     sys.path.insert(0, "..")
10 else:
11     sys.path.insert(0, os.path.join(os.path.split(__file__)[0], '..'))
12 from lab_15_Overriding.circle_def import main
13
14 class Circle(list):
15
16     def __init__(self, data, times):
17         list.__init__(self, data)
18         self.times = times
19
20     def __getitem__(self, i):
21         """circle[i] --> Circle.__getitem__(circle, i)."""
22         l_self = len(self)
23         if i >= self.times * l_self:
24             raise IndexError, \
25                   "Error raised: circle object only goes around %d times"\
26                   % self.times
27         return list.__getitem__(self, i % l_self)
28
29     def __iter__(self):
30         """Because we are inheriting from list, and it has it's own
31         __iter__, we need to override it to get all the functionality
32         we had before.
33         """
34         for i in range(self.times * len(self)):
35             yield self[i]
36
37 def TestList():
38     circle = Circle("tic", 3)
39     print "Testing list functions:", circle
40     circle += 'k'
41     print [x for x in circle]
42     circle.sort()
43     print circle
44
```

```
45 if __name__ == '__main__':
46     main()
47     TestList()
48
49 """
50 $ new_style_circle_def.py
51 main() produces the same output.
52 ---
53 Testing list functions: ['t', 'i', 'c']
54 ['t', 'i', 'c', 't', 'i', 'c', 't', 'i', 'c', 't', 'i', 'c']
55 ['c', 'i', 'k', 't']
56 $
57
58 Note: Using super, and in general avoiding hardcoding the literal name
59 of classes, makes your code more robust against change:
60
61 super(object)
62    Typical use to call a cooperative superclass method:
63    class C(B):
64        def meth(self, arg):
65            super(C, self).meth(arg)
66
67
68 Note that super returns an object, so you don't need to put 'self'
69 in the argument list when you make a class using 'super'.
70 """
```

att.py

```
 1 #!/usr/bin/env python
 2 """Classes are blueprints for name spaces.  Attributes
 3 can be added on an object by object basis.  Feature or
 4 flaw?"""
 5
 6 class NameSpace:
 7
 8     def __init__(self):
 9         self.variable = 10
10
11     def __str__(self):
12         return str(self.__dict__)
13
14 def main():
15     george = NameSpace()
16     george.age = '44'
17     george.job = 'coder'
18
19     dinner = NameSpace()
20     dinner.maindish = 'stew'
21     dinner.dessert = 'pie'
22     dinner.varable = '101'
23
24     print george
25     print dinner
26
27 if __name__ == '__main__':
28     main()
29 """
30 $ att.py
31 {'variable': 10, 'age': '44', 'job': 'coder'}
32 {'variable': 10, 'dessert': 'pie', 'varable': '101', 'maindish': 'stew'}
33 $
34
35 """
```

att2.py

```
#!/usr/bin/env python
"""att2.py  (Optional)
You can override assignment and referencing attributes.

referencing:

      object.x -> calls __getattr__ (if provided)

  but only if the x attribute does not exist.

assignment:

      object.x = 3  -> calls __setattr__ (if provided)
"""
class Secret:
    """The secret can only be set on initialization: s = Secret("rose");
    and only the allowed attributes can be set. """

    allowed_attributes = ("members", "purpose")

    def __init__(self, secret):
        self.__dict__['_Secret__secret'] = secret
        # Here we snuck around __setattr__ by adding directly to the
        # __dict__, so that we could disallow the secret to be set
        # after initialization.  And, we had to do our own name
        # mangling to keep it pseudo-secret.

    def IsSecret(self, word):
        return self.__secret == word

    def __getattr__(self, attribute_name):
        if attribute_name == 'secret':
            return "a secret!"
        raise AttributeError, "%s instance has no attribute '%s'" \
              % (self.__class__.__name__, attribute_name)

    def __setattr__(self, attribute_name, value):
        if attribute_name == "secret":
            raise NameError, "You can't change the %s to %s" % \
                  (attribute_name, value)
        elif attribute_name in Secret.allowed_attributes:
            self.__dict__[attribute_name] = value
            # setattr(self, attribute_name) would loop forever!
        else:
```

```
45             raise AttributeError, "%s is not an attribute for class %s"\
46                   % (attribute_name, str(self.__class__).split('.')[1])
47
48 def main():
49     club = Secret('snake')
50     print "club.secret is", club.secret
51     print "club.IsSecret('snake')", club.IsSecret('snake')
52     try:
53         print "club.x is", club.x
54     except AttributeError, msg:
55         print "\nError: ", msg
56     try:
57         print "club.members is", club.members
58     except AttributeError, msg:
59         print "\nError: ", msg
60     club.members = 7
61     print "club.members is", club.members
62     try:
63         club.secret = 'lizard'
64     except NameError, msg:
65         print "\nError: ", msg
66     try:
67         print "Setting club.x",
68         club.x = 'cucumber'
69     except AttributeError, msg:
70         print "\nError: ", msg
71     print dir(club)
72
73 if __name__ == '__main__':
74     main()
75 """$ att2.py
76 club.secret is a secret!
77 club.IsSecret('snake') True
78 club.x is
79 Error:  Secret instance has no attribute 'x'
80 club.members is
81 Error:  Secret instance has no attribute 'members'
82 club.members is 7
83
84 Error:  You can't change the secret to lizard
85 Setting club.x
86 Error:  x is not an attribute for class Secret
87 ['_Secret__secret', '__doc__', '__getattr__', '__init__', '__module__',
88  '__setattr__', 'allowed_attributes', 'IsSecret', 'members']
89 $ """
```

att3.py

```
#!/usr/bin/env python
"""att3.py  (Optional)
'property' gives you total control over a particular attribute.
It is only available in "New Style" classes, which inherit from
the "object" superclass.
"""
class Secret(object):

    def __init__(self, secret_word):
        self.__secret = secret_word

    def GetSecret(self):
        return "I'll never tell."

    def SetSecret(self, new_secret):
        try:
            self.__secret += new_secret
        except AttributeError:
            raise AttributeError, "You can't start over."

    def DelSecret(self):
        print "No more secrets!"
        del self.__secret

    secret = property(GetSecret, SetSecret, DelSecret,
                      "I've got the secret.")

    def IsSecret(self, trial):
        return trial == self.__secret

def main():
    word = Secret('fish')
    print "word.secret =", word.secret
    print "word.IsSecret('fish')", word.IsSecret('fish')
    word.secret = 'gills'
    print "word.IsSecret('gills')", word.IsSecret('gills')
    print "word.IsSecret('fishgills')", word.IsSecret('fishgills')
    print Secret.secret.__doc__
    del word.secret
    word.secret = 'flounder'

if __name__ == '__main__':
    main()

```

```
45 """
46 $ att3.py
47 word.secret = I'll never tell.
48 word.IsSecret('fish') True
49 word.IsSecret('gills') False
50 word.IsSecret('fishgills') True
51 I've got the secret.
52 No more secrets!
53 Traceback (most recent call last):
54   File "./att3.py", line 43, in <module>
55     main()
56   File "./att3.py", line 40, in main
57     word.secret = 'flounder'
58   File "./att3.py", line 19, in SetSecret
59     raise AttributeError, "You can't start over."
60 AttributeError: You can't start over.
61 $"""
```

static.py

```
 1 #!/usr/bin/env python
 2 """static.py (Optional) Class variables are supported and work nicely,
 3 but there is no obvious way to call a method unless you have an
 4 object."""
 5
 6 class Static:
 7     number = 0
 8
 9     def __init__(self):
10         Static.number += 1
11         self.number = Static.number
12
13     def __str__(self):
14         return "%d of %d" % (self.number, Static.number)
15
16 def main():
17     objects = [Static() for i in range(3)]
18     print ', '.join([str(obj) for obj in objects])
19
20 if __name__ == '__main__':
21     main()
22 """
23 $ static.py
24 1 of 3, 2 of 3, 3 of 3
25 $ """
26
```

static2.py

```
 1 #!/usr/bin/env python
 2 """(Optional)
 3 @staticmethod and @classmethod built-in decorators let you invoke
 4 methods without having objects."""
 5
 6 import static
 7
 8 class Static2(static.Static):
 9
10     @classmethod
11     def JumpUp(cls, number):      # cls will be the class
12         print 'In classmethod(JumpUp), cls =', cls
13         static.Static.number += number
14
15     @staticmethod
16     def StartOver():           # no self for a static method!
17         static.Static.number = 0
18
19 def main():
20     objects = [Static2() for i in range(3)]
21     print ', '.join([str(obj) for obj in objects])
22     Static2.StartOver()
23     objects += [Static2() for i in range(3)]
24     print 'After StartOver()', ', '.join([str(obj) for obj in objects])
25     Static2.JumpUp(100)
26     objects += [Static2() for i in range(3)]
27     print 'After JumpUp()', ', '.join([str(obj) for obj in objects])
28
29 if __name__ == '__main__':
30     main()
31
32 """
33 $ static2.py
34 1 of 3, 2 of 3, 3 of 3
35 After StartOver() 1 of 3, 2 of 3, 3 of 3, 1 of 3, 2 of 3, 3 of 3
36 In classmethod(JumpUp), self = __main__.Static2
37 After JumpUp() 1 of 106, 2 of 106, 3 of 106, 1 of 106, 2 of 106, 3 of 106,
38  104 of 106, 105 of 106, 106 of 106
39 $ """
40
```

diamond.py

```
 1 #!/usr/bin/env python
 2 """
 3 Method Resolution For Diamond Inheritance (Optional)
 4
 5                 ----------
 6                |    A     |
 7                |  -----   |
 8                | __init__ |
 9                |self.x = 1|
10                 ----------
11                /|\    /|\
12          -------       ----------
13         |   B   |     |    C     |
14         |  ---  |     |  -----   |
15         |       |     | __init__ |
16          -------      |self.x = 2|
17            /|\         ----------
18             |            /|\
19             ----      ----
20                 |    |
21                 ------
22                |   D  |
23                 ------
24
25 Multiple inheritance from super classes that share a common super super class.
26    - - -  - - -  - - -  - - -  - - -  - - -  - - -  - - -  - - -
27
28 Resolution order for new style classes in a diamond pattern:
29
30 In the classic case, always left to right, depth first:
31
32 D().x = 1
33
34 In a new-style class, when the classes inherit from "object":
35
36 """
37
38
39
40
41
42
43
44
```

```
45 class A(object):
46     def __init__(self):
47         self.x = 1
48 class B(A):
49     pass
50 class C(A):
51     def __init__(self):
52         self.x = 2
53 class D(B, C):
54     pass
55
56 print D().x
57 """
58 $ diamond.py
59 2
60 $
61    The new rules are:
62    List all the classes visited in the classic case:
63    [D, B, A, C, A]
64    If there are duplicates, eliminate all but the last:
65    [D, B, C, A]
66    And that's the search order.
67 """
```

Lab 16

Optional Reading:

`www.geocities.com/foetsch/python/new_style_classes.htm`

If you have Chun, Chapter 13 is good.

1. Make a SortedDictionary class that inherits from the built-in dictionary, but the keys() method for your class returns a sorted list of keys. Also, provide an __iter__ that iterates a sorted list of keys.

   Make sure that any style of instantiation that you can use on a regular dictionary works on yours:

   ```
   {1:'1', 2:'2'}, {}, ((1, '1'), (2, '2'))
   ```

   (Optional) Allow someone instantiating your class to add a `description` attribute to an object, but no other attributes.

2. (from Chun, 13-17) Make a Money class that inherits from `float`. Your class should override some magic methods, especially `__str__()` so that printing the value looks like money:

   $3,264.04

   -$2,101,100.10

   My test is:

   ```
   print Money(-123.21)
   print Money(40.50)
   print Money(-1001.011)
   print Money(123456789.999)
   print Money(.10)
   print Money(.01)
   print Money(.055)
   print 'add:', Money(10) + Money(20), '==', Money(30)
   print 'repr:', eval(repr(Money(44.123))), '==', Money(44.123)
   print 'sub:', Money(44.333) - Money(55.444), '==', Money(-11.111)
   print 'neg:', -Money(10.00), '==', Money(-10.00)
   print 'mult:', 2 * Money(-11.11), '==', Money(-22.22), '==', \
Money(11.11) * -2
   print 'div:', (Money(44.44))/4, '==', Money(11.11)
   ```

   Much of the work is done in:

   `lab/lab_16_Comprehensions/lab08_4.py`