PYTHON LAB BOOK

Python For Programmers

UCSC Extension Online

Lab 15 Overriding

Topics

- Overriding
- *Has-A* vs *Is-A* relationships

v.4.0

lab14_1.py

```
 1 #!/usr/bin/env python
 2 """lab14_1.py Predict the output:"""
 3
 4 class X:
 5
 6     def __init__(self):
 7         self.x = 1
 8
 9     def Which(self):
10         print "X"
11
12 class A(X):
13
14     def __init__(self):
15         X.__init__(self)
16         self.y = 2
17
18 class Y:
19
20     def __init__(self):
21         self.z = 3
22
23     def Which(self):
24         print "Y"
25
26 class B(Y):
27
28     def __init__(self):
29         Y.__init__(self)
30         self.x = 4
31
32 class AB(A, B):
33     pass
34
35 class BA(B, A):
36     pass
37
38 ab = AB()
39 ba = BA()
40
41
42
43
44
```

```
45
46
47
48 """python -i lab14_1.py
49 >>> ab.x
50 1
51 >>> ab.y
52 2
53 >>> ab.z
54 Traceback (most recent call last):
55   File "<stdin>", line 1, in ?
56 AttributeError: AB instance has no attribute 'z'
57 >>> ab.Which()
58 X
59 >>> ba.x
60 4
61 >>> ba.y
62 Traceback (most recent call last):
63   File "<stdin>", line 1, in ?
64 AttributeError: BA instance has no attribute 'y'
65 >>> ba.z
66 3
67 >>> ba.Which()
68 Y
69 >>>"""
```

lab14_2.py

```
 1 #!/usr/bin/env python
 2 """
 3 lab14_2.py  Implement a Stack class.  It should
 4 have two functions: you add things to the top of your
 5 stack (usually called push); and you take things from
 6 the top of your stack, (usually called pop)."""
 7
 8 class Stack:
 9     def __init__(self, starter=None):
10         if starter == None:
11             self.things = []
12             return
13         try:
14             # Here we accept any sequence type
15             self.things = list(starter)
16         except TypeError:
17             # Here we accept any non-sequence type
18             self.things = [starter]
19
20     def push(self, thing):
21         self.things += [thing]
22
23     def pop(self):
24         try:
25             return self.things.pop()
26         except IndexError:
27             return None
28 def main():
29     box = Stack()
30     box.pop()
31     box.push('nickel')
32     box.push('dime')
33     print box.pop()
34     print box.pop()
35     print box.pop()
36     print "Initializations:"
37     for init in ([1, 2], "sunshine", 3.2):
38         s = Stack(init)
39         print "%s:" % (init)
40         while True:
41             popped = s.pop()
42             if popped:
43                 print popped,
44             else:
```

```
45                 print
46                 break
47
48 if __name__ == '__main__':
49     main()
50
51
52
53
54
55
56
57
58
59 """
60 $ lab14_2.py
61 dime
62 nickel
63 None
64 $ python -i lab_14_2.py
65 dime
66 nickel
67 None
68 None
69 Initializations:
70 [1, 2]:
71 2 1
72 sunshine:
73 e n i h s n u s
74 3.2:
75 3.2
76 >>> x = Stack('ab')
77 >>> x.pop()
78 'b'
79 >>> y = Stack(x)
80 >>> y.pop()
81 <__main__.Stack instance at 0xb7ea42ac>
82 >>> y = Stack(x)
83 >>> y.pop().pop()
84 'a'
85 >>>
86 """
```

printable_stack_def.py

```
#!/usr/bin/env python
"""printable_stack_def.py Extending our stack, providing a
"special method", __str__, which is called whenever:
   1.  "%s" % (printable_stack_object)
   2.  str(printable_stack_object)
   3.  print printable_stack_object
"""

import lab14_2 as stack_def  # copy of lab exercise

class PrintableStack(stack_def.Stack):
    """This class will reveal itself, and the result looks
    like a stack."""
    def __str__(self):
        try:
            min_width = max([len(thing) for thing in self.things])
        except ValueError: # self.things was empty
            min_width = 4
            center = ' [] '
        else:
            center = '|\n|'.join([thing.center(min_width) \
                                  for thing in self.things])
        top = ' '+ '-' * min_width + '\n|'
        bottom = '|\n ' + '-' * min_width + '\n'
        return top + center + bottom

def main():
    box = stack_def.Stack()
    print box
    for food in ['bread','mayo','cheese']:
        print 'Stack pushing', food
        box.push(food)
    print box
    print 'PrintableStack shows its stack'
    pbox = PrintableStack()
    print pbox
    for food in ['bread','mayo','cheese']:
        print 'PrintableStack pushing', food
        pbox.push(food)
        print pbox
    for i in range(3):
        print 'PrintableStack popping', pbox.pop()
        print pbox
if __name__ == '__main__':
```

```
45     main()
46 """
47 $ python -i stack.py
48 <__main__.Stack instance at 0x814dfd4>
49 Stack pushing bread
50 Stack pushing mayo
51 Stack pushing cheese
52 <__main__.Stack instance at 0x814dfd4>
53 PrintableStack shows its stack
54  ----
55 | [] |
56  ----
57 PrintableStack pushing bread
58  -----
59 |bread|
60  -----
61 PrintableStack pushing mayo
62  -----
63 | mayo|
64 |bread|
65  -----
66 PrintableStack pushing cheese
67  ------
68 |cheese|
69 | mayo |
70 |bread |
71  ------
72 PrintableStack popping cheese
73  -----
74 | mayo|
75 |bread|
76  -----
77 PrintableStack popping mayo
78  -----
79 |bread|
80  -----
81 PrintableStack popping bread
82  ----
83 | [] |
84  ----
85 >>> dir(box)
86 ['__doc__', '__init__', '__module__', 'pop', 'push', 'things']
87 >>> print box.things
88 ['bread', 'mayo', 'cheese']     """
```

lab14_3.py

```
 1 #!/usr/bin/env python
 2 """lab14_3.py  Implement this inheritance tree:
 3
 4      Employee
 5         name
 6
 7      SalariedEmployee   --> Inherits from Employee
 8         Has a yearly salary.
 9
10      ContractEmployee   --> Inherits from Employee
11         Has an hourly rate
12 """
13 from __future__ import division
14 import sys
15
16 class Employee:
17     """Employee class, should only be instantiated in a subclass"""
18
19     def __init__(self, name):
20         self.name = name
21
22     def GiveRaise(self, percent):
23         """percent is the percent raise, where 100 doubles
24         the pay rate."""
25
26         percent /= 100.0
27         self.pay_rate *= 1 + percent
28
29     def PrintName(self):
30         print self.name,
31
32     def SetPayRate(self, pay_rate):
33         """The pay period is in the sub-class"""
34         try:
35             self.pay_rate = float(pay_rate)
36         except ValueError:
37             print >> sys.stderr, 'The rate must be a number.'
38
39 class SalariedEmployee(Employee):
40     """pay_rate is the yearly salary.  A pay period is 1 week."""
41
42     def CalculatePay(self, weeks):
43         try:
44             return self.pay_rate * weeks/52
```

```
45          except ValueError:
46              print 'How many weeks?'
47          except AttributeError:
48              print 'You must SetPayRate first'
49
50      def SetSalary(self, amt):
51          Employee.SetPayRate(self, amt)
52
53  class ContractEmployee(Employee):
54      """pay_rate is hourly pay.  A pay period is 1 hour."""
55
56      def CalculatePay(self, hours):
57          try:
58              return self.pay_rate * hours
59          except ValueError:
60              print >> sys.stderr, 'How man hours?'
61          except AttributeError:
62              print >> sys.stderr, 'You must SetPayRate first'
63          sys.exit(1)
64
65  def main():
66      joe = SalariedEmployee('Joe')
67      joe.SetSalary(52000)
68      joe.PrintName()
69      print "here's $%.2f for you. " % joe.CalculatePay(1)
70      joe.GiveRaise(2)
71      joe.PrintName()
72      print "here's $%.2f for you. " % joe.CalculatePay(2)
73
74      susan = ContractEmployee('Susan')
75      susan.PrintName()
76      susan.SetPayRate(100)
77      print "here's $%.2f for you. " % susan.CalculatePay(80)
78      susan.GiveRaise(2)
79      susan.PrintName()
80      print "here's $%.2f for you. " % susan.CalculatePay(80)
81
82      fred = Employee('Fred')
83      fred.SetPayRate(100)
84      try:
85          fred.CalculatePay(20) # Crash! No CalculatePay for Employee
86      except AttributeError:
87          pass
88
89  if __name__ == '__main__':
```

```
 90     main()
 91
 92 """
 93 $ lab14_3.py
 94 Joe here's $1000.00 for you.
 95 Joe here's $2040.00 for you.
 96 Susan here's $8000.00 for you.
 97 Susan here's $8160.00 for you.
 98 $
 99 """
100
```

welcomer_def.py

```
 1 #!/usr/bin/env python
 2 """
 3 Another inheritance example, using the previous examples
 4 by importing the old code.  This one implements __call__,
 5 __str__, and __del__ and a class attribute."""
 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 import lab_14_OOP.lab14_3 as employee_def
13 import lab_14_OOP.greeter5_def as greeter_def
14
15 class Welcomer(greeter_def.NamedGreeter, employee_def.SalariedEmployee):
16     """Inherits from Salaried Employee"""
17     welcomers = 0
18
19     def __init__(self, name):
20         Welcomer.welcomers += 1
21         employee_def.SalariedEmployee.__init__(self, name)
22
23     def __call__(self, something):
24         print something, "yourself!"
25
26     def __del__(self):
27         Welcomer.welcomers -= 1
28         print self.name, 'says "Oh no!"'
29
30     def __str__(self):
31         return self.name
32
33 def main():
34     joe = Welcomer('Joe')
35     joe.Greet()
36     joe.SetSalary(20000)
37     print joe, "here's $%.2f for you. " % joe.CalculatePay(80)
38     marsha = Welcomer('Marsha')
39     marsha.SetSalary(19500)
40     marsha('Get to work')
41     print marsha, "here's $%.2f for you. " % marsha.CalculatePay(80)
42     print 'There are %d welcomers.' % Welcomer.welcomers
43     joe('Goodbye')
44     print 'Deleting Joe'
```

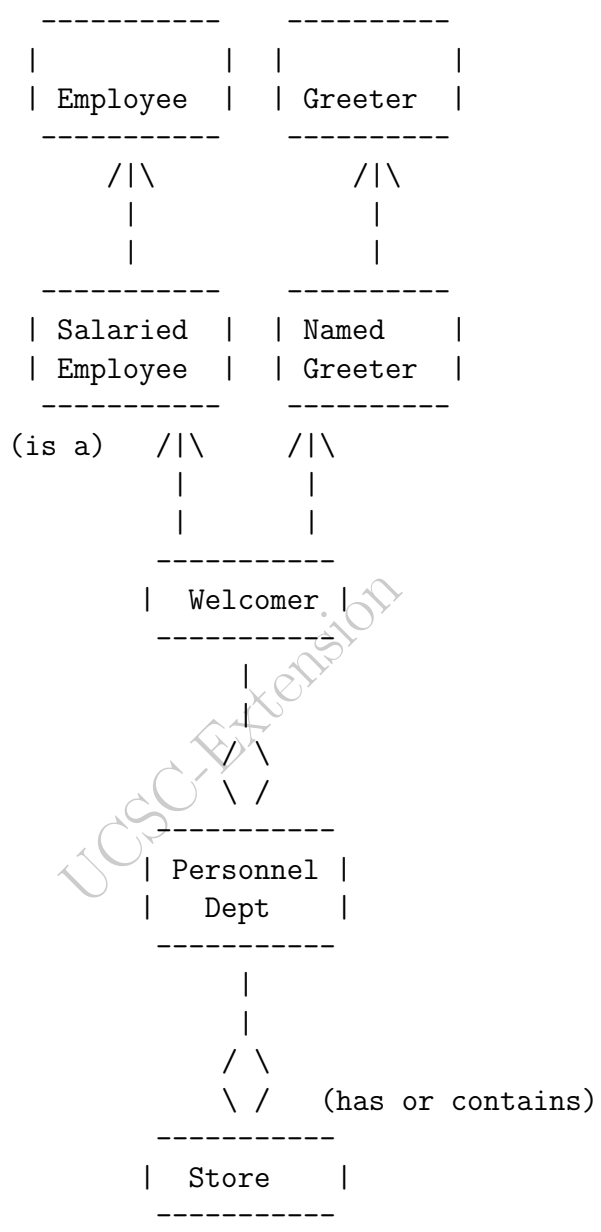
```
45     del joe
46     print 'There are %d welcomers.' % Welcomer.welcomers
47     marsha.Greet()
48
49 if __name__ == '__main__':
50     main()
51 """
52 $ welcomer_def.py
53 Hello World
54 I'm Joe
55 Joe here's $30769.23 for you.
56 Get to work yourself!
57 Marsha here's $30000.00 for you.
58 There are 2 welcomers.
59 Goodbye yourself!
60 Deleting Joe
61 Joe says "Oh no!"
62 There are 1 welcomers.
63 Hello World
64 I'm Marsha
65 Marsha says "Oh no!"
66 $
67 """
```

circle_def.py

```
 1 #!/usr/bin/env python
 2 """A Circle class, acheived by overriding __getitem__ which provides
 3 the behavior for indexing, i.e., [].  This also provides the correct
 4 cyclical behavior whenever an iterator is used, i.e., for,
 5 enumerate() and sorted().  reversed() needs __reversed__ defined.
 6 """
 7 class Circle:
 8
 9     def __init__(self, data, times):
10         """Put the 'data' in a circle that goes around 'times' times."""
11         self.data = data
12         self.times = times
13
14     def __getitem__(self, i):
15         """circle[i] --> Circle.__getitem__(circle, i)."""
16         l_self = len(self)
17         if i >= self.times * l_self:
18             raise IndexError, \
19                   "Error raised: circle object only goes around %d times"\
20                   % self.times
21         return self.data[i % l_self]
22
23     def __len__(self):
24         return len(self.data)
25
26 def main():
27     circle = Circle("around", 3)
28
29     print "Works with circle[i], for i > len(circle) too:"
30     for i in range(4 * len(circle)):
31         try:
32             print "circle[%2d] = %s" % (i, circle[i])
33         except IndexError, msg:
34             print msg
35             break
36
37     print "Works with sorted:"
38     print sorted(circle)
39
40     print "Works for nested loops:"
41     small_circle = Circle("XO", 2)
42     for i, elementi in enumerate(small_circle):
43         print "small_circle[%d] = %s" % (i, elementi)
44
```

```
45     for i, elementi in enumerate(small_circle):
46         for j, elementj in enumerate(small_circle):
47             print "%3d:%3d -> %s%s" % (i, j,
48                                        elementi, elementj)
49 if __name__ == '__main__':
50     main()
51
52 """
53 $ circle_def.py
54 Works with circle[i], for i > len(circle) too:
55 circle[ 0] = a
56 circle[ 1] = r
57 circle[ 2] = o
58 circle[ 3] = u
59 circle[ 4] = n
60 circle[ 5] = d
61 circle[ 6] = a
62 circle[ 7] = r
63 circle[ 8] = o
64 circle[ 9] = u
65 circle[10] = n
66 circle[11] = d
67 circle[12] = a
68 circle[13] = r
69 circle[14] = o
70 circle[15] = u
71 circle[16] = n
72 circle[17] = d
73 Error raised: circle object only goes around 3 times
74 Works with sorted:
75 ['a', 'a', 'a', 'd', 'd', 'd', 'n', 'n', 'n', 'o', ...
76 Works for nested loops:
77 small_circle[0] = X
78 small_circle[1] = O          (continued from bottom)
79 small_circle[2] = X           1:  3 -> OO
80 small_circle[3] = O           2:  0 -> XX
81   0:  0 -> XX                 2:  1 -> XO
82   0:  1 -> XO                 2:  2 -> XX
83   0:  2 -> XX                 2:  3 -> XO
84   0:  3 -> XO                 3:  0 -> OX
85   1:  0 -> OX                 3:  1 -> OO
86   1:  1 -> OO                 3:  2 -> OX
87   1:  2 -> OX                 3:  3 -> OO
88 $ """
```

Simplified Class Diagram for a Store, using
UML - Unified Modeling Language



store_def.py

```
#!/usr/bin/env python
"""
Optional, except for the 'has a' relationship.

Here we implement the diagram, demonstrating a 'contains a'
or 'has a' relationship between classes, and a pseudo-private
attribute and method.  Inheritance is a 'is a' relationship.

Below, in PersonnelDept.__init__, notice that there is
self.__welcomers.  This is a "private" attribute because it has 2
leading underscores and no more than 1 trailing underscore.  It gets
mangled to _PersonnelDept__welcomers (_ClassName + attribute_name) so
that, from outside the class, __welcomers does not exist.
"""
import welcomer_def

class Store:
    def __init__(self, name):
        self.name = name
        self.hr = PersonnelDept(self.name) # Store *has a*
                                           # PersonnelDept
class PersonnelDept:
    def __init__(self, name):
        self.__welcomers = []
        self.store_name = name

    def Hire(self, name):
        new_guy = welcomer_def.Welcomer(name)
        self.__welcomers += [new_guy]
        print self.store_name, "welcomes %s, our new welcomer." % new_guy
        return new_guy

    def __find(self, name):
        for (i, worker) in enumerate(self.__welcomers):
            if worker.name == name:
                return i
        return -1

    def Fire(self, name):
        index = self.__find(name)
        if index == -1:
            return name, "doesn't work here."
        x = self.__welcomers.pop(index)
        print "%s, you are terminated.  Thank you and good luck." % x
```

```
45
46
47
48
49
50 def main():
51     flormart = Store('FlorMart')
52     jane = flormart.hr.Hire('Jane')
53     jane.SetSalary(20000)
54     jane.Greet()
55     print jane, "here's $%.2f for you. " % jane.CalculatePay(2)
56     print """Calling Jane("You're in trouble")"""
57     print jane, 'replies:'
58     jane("You're in trouble")
59     flormart.hr.Fire('Jane')
60
61 if __name__ == '__main__':
62     main()
63 """
64 $ python -i store_def.py
65 FlorMart welcomes Jane, our new welcomer.
66 Hello World
67 I'm Jane
68 Jane here's $769.23 for you.
69 Calling Jane("You're in trouble")
70 Jane replies:
71 You're in trouble yourself!
72 Jane, you are terminated.  Thank you and good luck.
73 Jane says "Oh no!"
74 >>> store = Store("A Store")
75 >>> print store.hr.__welcomers
76 Traceback (most recent call last):
77   File "<stdin>", line 1, in ?
78 AttributeError: PersonnelDept instance has no attribute '__welcomers'
79 >>> print store.hr._PersonnelDept__welcomers
80 []
81 >>>
82 $
83 """
```

Lab 15

Make a Clock class.

It can be initialized like this: `t = Clock(2, 30)` to make a time = 2 hours and 30 minutes.

Values should be manipulated so that $minutes < 60$ and $hours < 13$.

Override some of these:

- `str()` by providing `__str__()`.
- `+` by providing `__add__()`. When the interpreter sees:

  ```
  c1 = Clock(2, 30)
  c2 = Clock(1, 15)
  c3 = c1 + c2
  ```

  if you have provided `__add__()` for your `Clock` class, it will call:

  ```
  Clock.__add__(c1, c2)
  ```

  You want `c3` to also be an object of your `Clock` class.
- `-` by providing `__sub__()`. Have it return the minutes since 12:00.
- `repr()` by providing `__repr__()`. Test that your `__repr__` returns a string that is an evaluatable Python expression.

Time and energy permitting, allow other styles of initialization:

- `Clock()` defaults to the current time.
- `Clock((2, 30))` a tuple
- `Clock("2:30")` a string
- `Clock({'hr':2, 'min':30})` a dictionary
- `Clock(min=30, hr=2)` keywords

Yes, my solution is in labs.zip for later. Don't look at it. It'll confuse you and waste your time at this point.