PYTHON LAB BOOK

Python For Programmers

UCSC Extension Online

Lab 5 Scope

Topics

- Indentifier scope
- Default arguments
- Keyword arguments

v.4.0

lab04_1.py

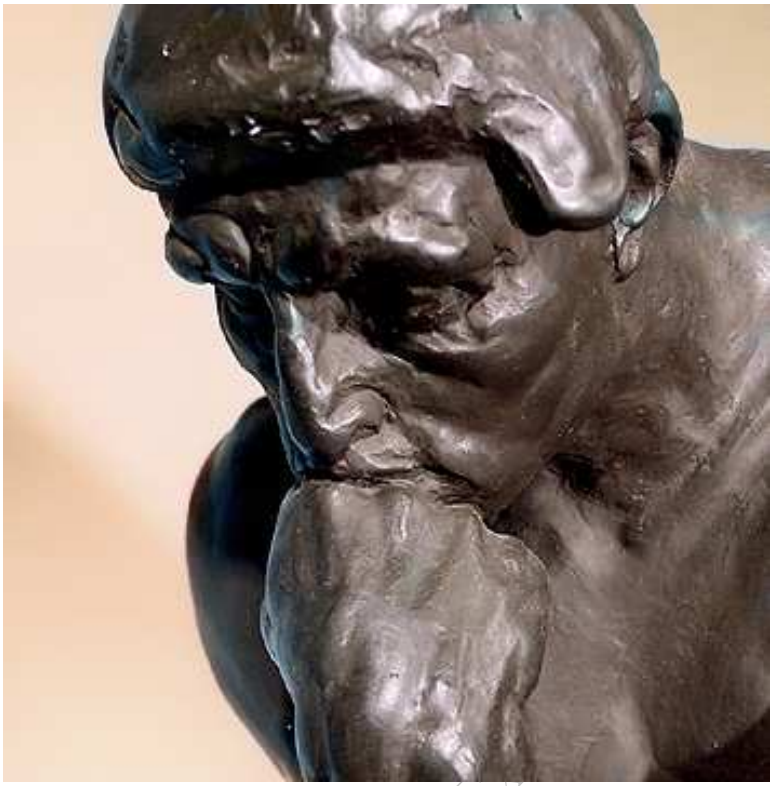
```
#!/usr/bin/env python
"""lab04_1.py -- Coin flip Experiment. """
import random

def Coin():
    """Simulates the flip of a coin."""

    if random.randrange(0, 2) == 0:
        return "tails"
    return "heads"

def Experiment():
    """Flips coins until it gets 3 head in a row."""

    heads = count = 0
    while heads < 3:
        count += 1
        if Coin() == 'heads':
            heads += 1
        else:    # 'tails'
            heads = 0
    return count
total = 0
for n in range(10):
    flips = Experiment()
#    print flips, "flips"
    total += flips
print "On average, it took %.1f coin flips to get 3 in a row." % (total/10.0)
"""
$ lab04_1.py
On average, it took 16.3 coin flips to get 3 in a row.
$ lab04_1.py
On average, it took 14.5 coin flips to get 3 in a row.
$ lab04_1.py
On average, it took 10.6 coin flips to get 3 in a row.
$ """
```

lab04_3.py

```
 1 #!/usr/bin/env python
 2 """ lab04_3.py
 3 Introspect the random module, and particularly the randrange()
 4 function it provides.  Use this module to write a "Flashcard"
 5 function.  Your function should ask the user:
 6
 7 What is 9 times 3
 8
 9 where 9 and 3 are any numbers from 0 to 12, randomly chosen.
10
11 If the user gets the answer right, your function should say, "Right!"
12 and then return a 1.  If the answer is wrong, say "Almost, the right
13 answer is 27" and return a 0.
14
15 Write a function called Quiz(n) that calls your Flashcard function n
16 times and reports the percentage of right answers like this,
17 "Score is 90". It also returns this percentage.
18
19 Make another function called Feedback(p) that receives a percentage,
20 0 - 100.  If p is 100 says,"Perfect!"; if it's 90-99, say "Excellent";
21 80-89, say "Very good"; 70-79, say "Good enough"; <= 69, "You need
22 more practice".
23
24 Test all that in your program, calling Quiz(10) and then pass the
25 returned value into Feedback().
26
27 Make a new function called "Praise" that takes no arguments.  It
28 prints one of (at least) 5 phrases of Praise, chosen randomly.  It
29 might say, "Right On!", or "Good work!", for example.  Call this
30 Praise() function from your Flashcard() function whenever your user
31 gets the answer right.
32 """
33 import random
34 def Flashcard():
35     n1 = random.randrange(13)
36     n2 = random.randrange(13)
37     answer = n1 * n2
38     print "What is %d times %d? " % (n1, n2)
39     while True:
40         try:
41             response = int(raw_input())
42         except ValueError:
43             print "%d times %d is a number.  What number? " % (n1, n2)
44             continue
```

```
45         if response == answer:
46             print "Right!",
47             Praise()
48             return 1
49         else:
50             print "Almost, the right answer is %d." % answer
51             return 0
52 def Quiz(n):
53     score = 0
54     for time in range(n):
55         score += Flashcard()
56     score =  int(100.0 * score/n)
57     print "Score is %d." % score
58     return score
59 
60 def Feedback(p):
61     if p == 100:
62         print 'Perfect!'
63     elif p >= 90:
64         print 'Excellent'
65     elif p >= 80:
66         print 'Very good'
67     elif p >= 70:
68         print 'Good enough'
69     else:
70         print 'You need more practice'
71 
72 def Praise():
73     pats = ("That's great!", "You're right!",
74             "Good work!", "Right On!", "Superb!")
75     print pats[random.randrange(len(pats))]
76 
77 Feedback(Quiz(10))
78 
79 """
80 
81 $ lab04_3.py
82 What is 12 times 9?
83 a
84 12 times 9 is a number.  What number?
85 108
86 Right! That's great!
87 What is 4 times 11?
88 44
89 Right! Superb!
```

```
 90 What is 2 times 0?
 91 2
 92 Almost, the right answer is 0.
 93 What is 9 times 11?
 94 99
 95 Right! That's great!
 96 What is 2 times 0?
 97 0
 98 Right! You're right!
 99 What is 2 times 6?
100 12
101 Right! Superb!
102 What is 3 times 4?
103 12
104 Right! You're right!
105 What is 4 times 4?
106 16
107 Right! Superb!
108 What is 2 times 8?
109 16
110 Right! Right On!
111 What is 7 times 12?
112 84
113 Right! Good work!
114 Score is 90.
115 Excellent
116 $
117 """
```



Pythonic Thinking: About Identifiers

Assignment:

```
cups = 10
```

Reference:

```
value = .25 * cups
```

scope.py

```
 1 #!/usr/bin/env python
 2 """ scope.py  -- Scoping example."""
 3
 4 pies = 1 # global identifier
 5
 6 # alters the local identifier pies, shadows the global identifier
 7 def Apple():
 8    pies = 25
 9
10    print "\nlocal pies in Apple() is", pies, "after entering Apple()"
11    pies += 1
12    print "local pies in Apple() is", pies, "before exiting Apple()"
13
14 # alters the global identifier pies
15 def Berry():
16    global pies
17
18    print "\nglobal pies is", pies, "on entering Berry()"
19    pies *= 10
20    print "global pies is", pies, "on exiting Berry()"
21
22 def Cherry():
23    print "In Cherry(), you can access, but not alter the global pies = ",\
24    pies
25
26 def Mud():
27
28    print "In Mud(), you can't even access global pies if you try", \
29    "to assign into it! \nSee: ", pies
30    pies = 999
31
32 print "global pies is", pies
33
34 pies = 7
35 print "global pies is", pies
36
37 Apple()
38 Berry()
39 Apple()
40 Berry()
41
42 print "\nglobal pies is", pies
43
44 Cherry()
```

```
45 Mud()
46
47 """
48 global pies is 1
49 global pies is 7
50
51 local pies in Apple() is 25 after entering Apple()
52 local pies in Apple() is 26 before exiting Apple()
53
54 global pies is 7 on entering Berry()
55 global pies is 70 on exiting Berry()
56
57 local pies in Apple() is 25 after entering Apple()
58 local pies in Apple() is 26 before exiting Apple()
59
60 global pies is 70 on entering Berry()
61 global pies is 700 on exiting Berry()
62
63 global pies is 700
64 In Cherry(), you can access, but not alter the global pies =  700
65 In Mud(), you can't even access global pies if you try to assign into it!
66 See:
67 Traceback (most recent call last):
68   File "./scope.py", line 44, in <module>
69     Mud()
70   File "./scope.py", line 28, in Mud
71     to assign into it! \nSee: ", pies
72 UnboundLocalError: local identifier 'pies' referenced before assignment
73
74 $
75 """
76
```

default.py

```
 1 #!/usr/bin/env python
 2 """default.py -- demonstrates defaulted arguments."""
 3
 4 def GoToThePark(name, best_friend="Jose'"):
 5     print name, 'and', best_friend
 6     print 'go to the park'
 7     print 'play hide-and-seek'
 8     print 'till long after dark'
 9     print
10
11 GoToThePark('Charlie', 'Jacqueline')
12 GoToThePark('Judi')
13
14 """
15 $ default.py
16 Charlie and Jacqueline
17 go to the park
18 play hide-and-seek
19 till long after dark
20
21 Judi and Jose'
22 go to the park
23 play hide-and-seek
24 till long after dark
25
26 $
27
28 """
```

greet.py

```
 1 #!/usr/bin/env python
 2 """Demonstrates keyword arguments"""
 3
 4 def Greet(first, last):
 5     print 'Hello', first, last
 6
 7 Greet('Rocky','The Squirrel')
 8 Greet(last='Moose', first='Bullwinkle')
 9
10 """
11 $ greet.py
12 Hello Rocky The Squirrel
13 Hello Bullwinkle Moose
14 $
15 """
```

Python's Flexible Functions Protocols
If it's not ambiguous, it is supported.

```
LAB 05

1.  Write a function that returns a total cost from the sales
price and sales tax rate.  Both price and tax_rate should be in
the argument list.   Provide a default value for the tax rate
= 8.25%.  Test your function.

2.  Write a Breakfast function that takes five arguments:
meat, eggs, potatoes, toast, and beverage.  The default meat is
bacon, eggs are over easy, potatoes is hash browns, toast is
white, and beverage is coffee.  Using the defaults, the
function just says: "Here is your bacon and over easy eggs
with hash browns and white toast.  Can I bring you more coffee?"
When the arguments are changed, the sentence changes.

Call it 3 different times with different orders, scrambling the
arguments.
```