PYTHON LAB BOOK

Python For Programmers

UCSC Extension Online

Lab 4 Functions

Topics

- Function protocols
- `import` and `reload`
- Module: `random`
- Introspection

v.4.0

lab03_1.py

```
#!/usr/bin/env python
"""lab03_1.py  How would you produce the following sequences using
the range operator?

        [3, 6, 9, 12]
        [-10, 210, 110]
        -1, -3, -5, -7,
"""
print range(3, 13, 3)
print range(-10, 211, 110)
for number in range(-1, -8, -2):
    print "%d," % (number),
print

"""

$ lab03_1.py
[3, 6, 9, 12]
[-10, 100, 210]
-1, -3, -5, -7,
$

"""
```

lab03_2.py

```
#!/usr/bin/env python
"""lab03_2.py
  2.  Produce this output using range and for:

      10, 9, 8, 7, 6, 5, 4, 3, 2, 1, BLASTOFF!!!
"""
for count in range(10, 0, -1):
    print "%d," % count,
print "BLASTOFF!!!"

"""
$ lab03_2.py
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, BLASTOFF!!!
$
"""
```

lab03_4.py

```
#!/usr/bin/env python
""" Produces:
Hi ya Manny!
Hi ya Moe!
Hi ya Jack!
"""
for name in 'Manny', 'Moe', 'Jack':
    print "Hi ya %s!" % name

"""
$ lab03_4.py
Hi ya Manny!
Hi ya Moe!
Hi ya Jack!
$

"""
```

lab03_5.py

```
#!/usr/bin/env python
"""lab03_5.py Prints the decimal equivalent of a binary string."""

string= raw_input("Binary string: ")
for ch in string:
    if ch not in "01":
        print string, 'is not a binary string.'
        break
else:
    # while method
    number = int(string)
    answer = 0
    power = 1
    while number > 0:
        answer += (number % 10) * power
        number /= 10
        power *= 2
    print 'while-loop: %s in binary is %d.' % (string, answer)

    # for method
    answer = 0
    power = 2 ** (len(string) - 1)
    for ch in string:
        answer += power * int(ch)
        power /= 2
    print 'for-loop: %s in binary is %d.' % (string, answer)

    # Python method
    print 'Easy way:', int(string, 2)

"""

$ lab03_5.py
Binary string: 1011
while-loop: 1011 in binary is 11.
for-loop: 1011 in binary is 11.
Easy way: 11
$ lab1_3_5.py
Binary string: 321
321 is not a binary string.
$ """
```

doubler.py

```
#!/usr/bin/env python
"""Function with one argument."""

def Doubler(x):
    return 2 * x

print Doubler(2)
print Doubler("Hi")
print Doubler(2.2)
"""

$ doubler.py
4
HiHi
4.4
$

"""
```

refs.py

```
#!/usr/bin/env python
"""Order matters."""

def Guitar():
    print 'strum. strum.'
    Drum()

Guitar()

def Drum():
    print 'boom! boom!'

"""
$ refs.py
strumming Guitar()
Traceback (innermost last):
  File "refs.py", line 8, in ?
    Guitar()
  File "refs.py", line 6, in Guitar
    Drum()
NameError: Drum
$
"""
```

refs2.py

```
#!/usr/bin/env python

def Guitar():
    print 'strum. strum.'
    Drum()

def Drum():
    print 'boom! boom!'

Guitar()

"""
$ refs2.py
strum. strum.
boom! boom!
$
"""
```

`import something`

is the command to bring something, i.e., other modules (files, libraries and packages) into your program.

Two important notes:

- `import` *runs* what it imports.
- `import` will not import the same module twice.

If you are using the interactive shell, you can run your module:

`import your_module`

but only once.

If you are using the interactive shell and you change your module, you might like:

`reload(your_module)`

- `reload` only works after the module was successfully loaded.
- `import` and `reload` have different syntaxes.

randrange.py

```
#!/usr/bin/env python
""" randrange.py  Rolls dice, demonstrating random.randrange(),
and a tuple with accessing a particular element with an index.
"""
import random
doubles = ("Can't happen", "Snake eyes!", "Little joe!", "Hard six!",
           "Hard eight!", "Fever!", "Box cars!")

def Rollem():
  dice = random.randrange(1, 7), random.randrange(1, 7)
  print "%d and %d" % dice
  if dice[0] == dice[1]:
    print doubles[dice[0]]

while True:
  response = raw_input("Ready to roll?")
  if response[0] in "Qq":
    break
  Rollem()
"""
$ randrange.py
Ready to roll?y
3 and 4
Ready to roll?
6 and 2
Ready to roll?
1 and 1
Snake eyes!
Ready to roll?
6 and 2
Ready to roll?
2 and 2
Little joe!
Ready to roll?
3 and 6
Ready to roll?
2 and 5
Ready to roll?
1 and 4
Ready to roll?
4 and 6
Ready to roll?q
$ """
```

quiz.py

```
#!/usr/bin/env python
"""
Quiz 1 (Lab 04)
Reports the user’s name and letter grade.
"""

name = raw_input("What’s your name: ")
score = int(raw_input("Score please: "))
print name, "your grade is",
if score >= 90:
    print ’A.’
elif score >= 80:
    print ’B.’
elif score >= 70:
    print ’C.’
elif score >= 60:
    print ’D.’
else:
    print ’F.’

"""
$ quiz.py
What’s your name: Jean
Score please: 92
Jean your grade is A.
$
"""
```

quiz2.py

```
#!/usr/bin/env python
"""Formatted output -- quiz answers"""

print "%4X" % (8 * 16 + 10)
print "%10.3f" % 232.346
print "%-12.2e|" % 2.33e+02
print "%-30s|" % "left"
print "%8.8s|" % "Tamale Pie"
print "%+.2f" % 3.13
"""
$ quiz2.py
  8A
   232.346
2.33e+02    |
left                          |
Tamale P|
+3.13
$
"""
```

Lab 04

1. Write a function called "Coin" that emulates the flip of a coin, returning "heads" or "tails".

   Write a function called "Experiment" that flips coins until it gets three heads in a row. It returns the number of flips it took to get three heads in a row.

   Have the program run the experiment 10 times and give the average number of flips it takes to get 3 heads in a row.

2. Introspection:

   In Idle or on the Python command line, import the math module:

```
>>> import math
```

   Now try:

```
>>> help(math)
```

   and there's the documentation for the module! Is that more than you wanted to know?

```
>>> dir(math)
```

   This produces a list of all the attributes in the module.

   The attributes without leading underscores are meant for you to use via the "dot" operator. You can get help on specific attributes available:

```
>>> help(math.sqrt)            (math-dot-sqrt)
```

   Try calling a function call:

```
>>> math.sqrt(9)
```

   and, just to check the precision, try this:

```
>>> math.sqrt(1.23456789) * math.sqrt(1.23456789)
```

Lab 04 Continued

3. Introspect the random module, and particularly the randrange() function it provides. Use this module to write a "Flashcard" function. Your function should ask the user:

What is 9 times 3

where 9 and 3 are any numbers from 0 to 12, randomly chosen.

If the user gets the answer right, your function should say, "Right!" and then return a 1. If the answer is wrong, say "Almost, the right answer is 27" and return a 0.

Write a function called Quiz(n) that calls your flashcard function n times and reports the percentage of right answers like this, "Score is 90". It also returns this percentage.

Make another function called Feedback(p) that receives a percentage, 0 - 100. If p is 100 says, "Perfect!"; if it’s 90-99, say "Excellent"; 80-89, say "Very good"; 70-79, say "Good enough"; <= 69, "You need more practice".

Test all that in your program, calling Quiz(10) and then pass the returned value into Feedback().

Make a new function called "Praise" that takes no arguments. It prints one of (at least) 5 phrases of praise, chosen randomly. It might say, "Right On!", or "Good work!", for example. Call this Praise() function from your Flashcard() function whenever your user gets the answer right.