PYTHON LAB BOOK

Python For Programmers
UCSC Extension Online

Lab 9 Dictionaries

Topics

- Importing with `from`
- Dictionaries

```
Lab 08.1

>>> [2**x for x in range(8)]
[1, 2, 4, 8, 16, 32, 64, 128]

>>> [2**x for x in range(64)]

[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384,
32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304,
8388608, 16777216, 33554432, 67108864, 134217728, 268435456,
536870912, 1073741824, 2147483648L, 4294967296L, 8589934592L,
17179869184L, 34359738368L, 68719476736L, 137438953472L,
274877906944L, 549755813888L, 1099511627776L, 2199023255552L,
4398046511104L, 8796093022208L, 17592186044416L, 35184372088832L,
70368744177664L, 140737488355328L, 281474976710656L, 562949953421312L,
1125899906842624L, 2251799813685248L, 4503599627370496L,
9007199254740992L, 18014398509481984L, 36028797018963968L,
72057594037927936L, 144115188075855872L, 288230376151711744L,
576460752303423488L, 1152921504606846976L, 2305843009213693952L,
4611686018427387904L, 9223372036854775808L] >>>

Python calculations are hardware-limited.  The exact error depends
on the version of Python.
```

lab08_2.py

```
 1 #!/usr/bin/env python
 2 """lab08_2.py Use list comprehensions to make a deck of cards."""
 3
 4 def Cards():
 5     """Return a deck of cards as a list of strings."""
 6
 7     values = [str(x) for x in range(2, 11)] + ['Jack','Queen','King','Ace']
 8     suits = ('Clubs', 'Diamonds', 'Hearts', 'Spades')
 9     deck = [v + ' of ' +  s for s in suits for v in values] + ["Joker"] * 2
10     return deck
11
12 def main():
13     deck = Cards()
14     print "The deck contains:"
15     for i, card in enumerate(deck):
16         if card is deck[-1]:
17             print 'and %s.' % card,
18         else:
19             print '%s, ' % card,
20         if i % 4 == 3:
21             print
22
23 if __name__ == '__main__':
24     main()
25 """
26 $ lab08_2.py
27 The deck contains:
28 2 of Clubs,  3 of Clubs,  4 of Clubs,  5 of Clubs,
29 6 of Clubs,  7 of Clubs,  8 of Clubs,  9 of Clubs,
30 10 of Clubs,  Jack of Clubs,  Queen of Clubs,  King of Clubs,
31 Ace of Clubs,  2 of Diamonds,  3 of Diamonds,  4 of Diamonds,
32 5 of Diamonds,  6 of Diamonds,  7 of Diamonds,  8 of Diamonds,
33 9 of Diamonds,  10 of Diamonds,  Jack of Diamonds,  Queen of Diamonds,
34 King of Diamonds,  Ace of Diamonds,  2 of Hearts,  3 of Hearts,
35 4 of Hearts,  5 of Hearts,  6 of Hearts,  7 of Hearts,
36 8 of Hearts,  9 of Hearts,  10 of Hearts,  Jack of Hearts,
37 Queen of Hearts,  King of Hearts,  Ace of Hearts,  2 of Spades,
38 3 of Spades,  4 of Spades,  5 of Spades,  6 of Spades,
39 7 of Spades,  8 of Spades,  9 of Spades,  10 of Spades,
40 Jack of Spades,  Queen of Spades,  King of Spades,  Ace of Spades,
41 Joker,  and Joker.
42 $
43 """
```

lab08_3.py

```
 1 #!/usr/bin/env python
 2 """Use list comprehensions to produce the quiz output again, this time
 3 with the fewest possible lines of code."""
 4
 5 print '\n'.join(['',join(['%4d' % (n * m) \
 6                          for n in range(6)]) for m in range(6)])
 7 """
 8 $ lab08_3.py
 9    0   0   0   0   0   0
10    0   1   2   3   4   5
11    0   2   4   6   8  10
12    0   3   6   9  12  15
13    0   4   8  12  16  20
14    0   5  10  15  20  25
15 $
16 """
17
```

lab08_4.py

```
 1 #!/usr/bin/env python
 2 """lab08_4.py
 3 MoneyFormat(amount) returns a money representation of the amount
 4 """
 5
 6 def MoneyFormat(amount):
 7     """MoneyFormat(amount) returns a money representation of the amount."""
 8     neg = False
 9     if amount < 0:
10         amount *= -1
11         neg = True
12     money = "%.2f" % amount
13     chars = list(money)
14     chars.reverse()
15     parts = chars[:3]
16     for i, ch in enumerate(chars[3:]):
17         if i > 0 and i % 3 == 0:
18             parts += ','
19         parts += ch
20     parts.reverse()
21     if neg:
22         parts.insert(0, '-$')
23     else:
24         parts.insert(0, '$')
25     return ''.join(parts)
26
27 def main():
28     print MoneyFormat(-123.21)
29     print MoneyFormat(3)
30     print MoneyFormat(14.3123)
31     print MoneyFormat(1234567.89)
32     print MoneyFormat(-88.88)
33
34 if __name__ == '__main__':
35     main()
36 """
37 $ lab08_4.py
38 -$123.21
39 $3.00
40 $14.31
41 $1,234,567.89
42 -$88.88
43 $
44 """
```

mod0.py

```
 1 #!/usr/bin/env python
 2 """mod0.py You can use the 'from' keyword to import a module so
 3 that you can bring the specified attributes of the module into
 4 your local namespace.
 5 """
 6
 7 from math import pi
 8
 9 def Area(radius):
10     return pi * radius * radius
11
12 print Area(3)
13
14 """
15 $ mod0.py
16 28.2743338823
17 $
18 """
```

mod1.py

```
 1 #!/usr/bin/env python
 2 """mod1.py  You can use ->   from module import *
 3    to bring all the attributes into your local namespace.
 4    But this is usually considered to be bad practice.
 5    You lose track of which attributes are local and are
 6    not local.  And, here's another problem.
 7 """
 8
 9 x = 1
10 y = [1,2,3]
11
12 def Printx():
13     print x
14
15 def Printy():
16     print y
17 """
18 >>> from mod1 import *
19
20
21 >>> Printx()
22 1
23 >>> x = 2
24 >>> Printx()    <-- gets mod1 version
25 1
26 >>> print x     <-- gets local version
27 2
28
29
30 >>> Printy()
31 [1, 2, 3]
32 >>> y[1] = 'a'  <-- a list is mutable!
33 >>> y
34 [1, 'a', 3]     <-- local version
35 >>> Printy()
36 [1, 'a', 3]     <-- mod1 version
37 >>>
38
39
40
41
42
43
44
```

```
45
46
47
48
49
50
51
52 You can import certain attributes:
53
54 >>> from mod1 import Printx, Printy
55 >>> Printx()
56 1
57 >>> mod1.x = 3
58 Traceback (most recent call last):
59   File "<stdin>", line 1, in ?
60 NameError: name 'mod1' is not defined
61
62
63 >>> import mod1
64 >>> mod1.x = 3
65 >>> Printx()
66 3
67 >>> mod1.Printx()
68 3
69 >>> """
```

mod2.py

```
 1 #!/usr/bin/env python
 2 """You can keep your identifiers from being imported with
 3 from ... import * by prefixing a '_' to the name."""
 4
 5 _x = 1
 6 _y = [1,2,3]
 7
 8 def Printx():
 9     print _x
10
11 def Printy():
12     print _y
13
14 """
15
16 >>> from mod2 import *
17 >>> dir()
18 ['__builtins__', '__doc__', '__name__', 'Printx', 'Printy']
19
20 However, if they are asked for explicitly, they come:
21
22 >>> from mod2 import _x, _y
23 >>> dir()
24 ['__builtins__', '__doc__', '__name__', '_x', '_y']
25 >>> Printx()
26 1
27 >>> _x = 33   <-- makes a local _x
28 >>> Printx()
29 1
30
31 >>> Printy()
32 [1, 2, 3]
33 >>> _y[1] = 44 <-- affects mod2's y
34 >>> Printy()
35 [1, 44, 3]
36
37 >>> import mod2
38 >>> mod2._x = 8
39 >>> Printx()
40 8
41 >>>
42 """
```

mod3.py

```
 1 #!/usr/bin/env python
 2 """mod3.py  __all__ specifies identifiers for export"""
 3
 4 __all__ = ['Printx', 'Printy']
 5
 6 x = 1
 7 y = [1,2,3]
 8
 9 def Printx():
10     print x
11
12 def Printy():
13     print y
14
15 """
16
17 >>> from mod3 import *
18 >>> dir()
19 ['__builtins__', '__doc__', '__name__', 'Printx', 'Printy']
20
21 >>> from mod3 import x
22 >>> x
23 1
24 >>> from mod3 import y
25 >>> y
26 [1, 2, 3]
27 >>> y[1] = 'hey'
28 >>> Printy()
29 [1, 'hey', 3]
30 >>>
31
32
33 Another SAMPLE RUN:
34 >>> import mod3 as other
35 >>> other.x = 88
36 >>> other.Printx()
37 88
38 >>>
39 """
```

py_dict.py

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

"""Dictionary implementation for demonstrating a Python dictionary."""

py_dict = {}   # empty dictionary

# initializing a dictionary

py_dict2 = {'break':'break out of a loop and skip the else',
            'continue':'go to the next iteration of the loop',
            'for':'set up looping'}

# Updating py_dict1 with py_dict2's keys and values.  If py_dict2 has
# keys already in py_dict, py_dict2's values will replace the old
# values for the key.

py_dict.update(py_dict2)

# And you can just add an entry

py_dict['pass'] = 'throw the ball'

# If you add an entry with a duplicate key, the new meaning will be
# the one that sticks:

py_dict['pass'] = 'do nothing'

def CollectEntries():
    """Collects a bunch of new entries for the dictionary"""
    while True:
        word = raw_input('Word: ')
        if not word:
            return
        meaning = raw_input('Meaning: ')
        py_dict[word] = meaning

def FindDefinitions():
    """Reports a key:value pair for a given key"""
    while True:
        word = raw_input('Word to find: ')
        if not word:
            return
        try:
            print '%s : %s' % (word, py_dict[word])
```

```
45         except KeyError:
46             print '%s is not in the dictionary.' % word
47
48 def MakePrompt(choices):
49     choice_list = sorted(choices)
50     guts = ', '.join(['(%s)%s' % (choice[0], choice[1:])\
51                       for choice in choice_list])
52     return 'Choose ' + guts + ' (enter to quit) '
53
54 def PrintEntries():
55     """Prints out the dictionary entries, sorted by key"""
56     for word in sorted(py_dict):
57         print '%s : %s' % (word, py_dict[word])
58
59 def main():
60     """Runs the user interface for dictionary manipulation."""
61     # The choices dictionary has function names for values.
62     choices = {'add': CollectEntries, 'find': FindDefinitions,
63                'print': PrintEntries}
64     prompt = MakePrompt(choices)
65
66     while True:
67         raw_choice = raw_input(prompt)
68         if not raw_choice:
69             break
70         given_choice = raw_choice[0].lower()
71         for maybe_choice in choices:
72             if maybe_choice[0] == given_choice:
73                 # The appropriate function is called
74                 # using the dictionary value for the name
75                 # of the function.
76                 choices[maybe_choice]()
77                 break
78         else:
79             print '%s is not an acceptible choice.' % raw_choice
80
81 if __name__ == '__main__':
82     main()
83 """
84 $ py_dict.py
85 Choose (a)dd, (f)ind, (p)rint (enter to quit) p
86 break : break out of a loop and skip the else
87 continue : go to the next iteration of the loop
88 for: set up looping
89 pass : do nothing
```

```
 90 Choose (a)dd, (f)ind, (p)rint (enter to quit) a
 91 Word: yield
 92 Meaning: return and start here with the next call to next()
 93 Word:
 94 Choose (a)dd, (f)ind, (p)rint (enter to quit) f
 95 Word to find: for
 96 for: set up looping
 97 Word to find: range
 98 range is not in the dictionary.
 99 Word to find:
100 Choose (a)dd, (f)ind, (p)rint (enter to quit)
101 $"""
```

Lab 09

The built-in `dict()` function provides a very flexible way to create a dictionary:

```
squares = dict(one=1, two=4, three=9)
```

Try it in the Python shell. Then see what functions are available for dictionaries:

```
dir(squares)              or dir(dict)
```

Popular functions are `keys`, `items` and `values`. Check out the documentation for these functions like this:

```
help(dict.keys)
```

If you have any questions about these or others that interest you, raise your hand. In particular, you will like `dict.items()` as you do this lab.

You also might like the behaviors of `for`, `in`, and `sorted`.

Collect and extract `labs.zip` from WebCT. Find: `labs/lab_09_Dictionaries/py_dict.py`

Edit the program so that is has another choice in the menu: `(d)efinitions`.

This new option will print out the dictionary alphabetically by the meanings:

```
Choose (a)dd, (d)efinitions, (f)ind, (p)rint (enter to quit) d
break out of a loop and skip the else : break
do nothing : pass
go to the next iteration of the loop : continue
set up looping : for
Choose (a)dd, (d)efinitions, (f)ind, (p)rint (enter to quit)
$
```