480 - April 4, 2008

Python Data Types:

list, tuples, strings, dictionaries, and sets

First: Read the free Python tutorial! http://docs.python.org/tut/. The examples below are mostly taken from there.

Lists

Python knows a number of compound data types, used to group together other values. The most versatile is the list, which can be written as a list of comma-separated values (items) between square brackets. List items need not all have the same type.

```
a = ['spam', 'eggs', 100, 1234]
a
```

```
['spam', 'eggs', 100, 1234]
```

Like string indices, list indices start at 0, and lists can be sliced, concatenated and so on:

```
a[0]
```

```
'spam'
```

```
a[3]
```

```
1234
```

```
a[-2]
```

```
100
```

```
a[1:-1]
```

```
['eggs', 100]
```

```
a[:2] + ['bacon', 2*2]
```

```
['spam', 'eggs', 'bacon', 4]
```

```
3*a[:3] + ['Boo!']
```

```
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100,
'Boo!']
```

```

```

It is possible to change individual elements of a list:

```
a
```

```
['spam', 'eggs', 100, 1234]
```

```
a[2] = a[2] + 23
a
```

```
['spam', 'eggs', 146, 1234]
```

```

```

Assignment to slices is also possible, and this can even change the size of the list or clear it entirely:

```
# Replace some items:
a[0:2] = [1, 12]
a
```

```
[1, 12, 146, 1234]
```

```
# Remove some:
a[0:2] = []
a
```

```
[146, 1234]
```

```
# Insert some:
a[1:1] = ['bletch', 'xyzzy']
a
```

```
[146, 'bletch', 'xyzzy', 1234]
```

```
# Insert (a copy of) a itself at the beginning
a[:0] = a
a
```

```
[146, 'bletch', 'xyzzy', 1234, 146, 'bletch', 'xyzzy', 1234]
```

```
# Clear the list: replace all items with an empty list
a[:] = []
a
```

```
[]
```

The built-in function len() applies to lists:

```
len(a)
```

```
0
```

It is possible to nest lists (create lists containing other lists), for example:

```
q = [2, 3]
p = [1, q, 4]
p
```

```
[1, [2, 3], 4]
```

```
len(p)
```

```
3
```

```
p[1]
```

```
[2, 3]
```

```
p[1][0]
```

```
2
```

```
p[1].append('xtra')     # See section 5.1
p
```

```
[1, [2, 3, 'xtra'], 4]
```

```
q
```

```
[2, 3, 'xtra']
```

Note that in the last example, p[1] and q really refer to the same object!!!

```
p[1] is q
```

```
True
```

```
q[0] = 'something'
```

```
p
```

```
[1, ['something', 3, 'xtra'], 4]
```

```

```

More List Operations

```
a = [66.25, 333, 333, 1, 1234.5]; a
```

```
[66.2500000000000, 333, 333, 1, 1234.50000000000]
```

```
print a.count(333), a.count(66.25), a.count('x')
```

```
2 1 0
```

```
a.insert(2, -1)
a
```

```
[66.2500000000000, 333, -1, 333, 1, 1234.50000000000]
```

```
a.append(333); a
```

```
[66.2500000000000, 333, -1, 333, 1, 1234.50000000000, 333]
```

```
a.index(333)
```

```
1
```

```
a.remove(333); a
```

```
[66.2500000000000, -1, 333, 1, 1234.50000000000, 333]
```

```
a.reverse(); a
```

```
[333, 1234.50000000000, 1, 333, -1, 66.2500000000000]
```

```
a.sort(); a
```

```
[-1, 1, 66.2500000000000, 333, 333, 1234.50000000000]
```

```

```

```

```

List Comprehension

```
freshfruit = ['  banana', '  loganberry ', 'passion fruit  ']
```

```
[weapon.strip() for weapon in freshfruit]
```

```
['banana', 'loganberry', 'passion fruit']
```

```
vec = [2, 4, 6]
```

```
[3*x for x in vec]
```

```
[6, 12, 18]
```

```
[3*x for x in vec if x > 3]
```

```
[12, 18]
```

```
[3*x for x in vec if x < 2]
```

```
[]
```

```
[[x,x^2] for x in vec]
```

```
[[2, 4], [4, 16], [6, 36]]
```

```
[x, x^2 for x in vec]   # error - parens required for tuples
```

```
Syntax Error:
    [x, x^2 for x in vec]       # error - parens required for tuple
```

```
[(x, x^2) for x in vec]
```

```
[(2, 4), (4, 16), (6, 36)]
```

```
vec1 = [2, 4, 6]
vec2 = [4, 3, -9]
[x*y for x in vec1 for y in vec2]
```

```
[8, 6, -18, 16, 12, -36, 24, 18, -54]
```

```
[x+y for x in vec1 for y in vec2]
```

```
[6, 5, -7, 8, 7, -5, 10, 9, -3]
```

```
[vec1[i]*vec2[i] for i in range(len(vec1))]
```

```
[8, 12, -54]
```

```
```

The del statement

```
a = [-1, 1, 66.25, 333, 333, 1234.5]; a
```

```
[-1, 1, 66.2500000000000, 333, 333, 1234.50000000000]
```

```
del a[0]
a
```

```
[1, 66.2500000000000, 333, 333, 1234.50000000000]
```

```
del a[2:4]
a
```

```
[1, 66.2500000000000, 1234.50000000000]
```

```
del a[:]
a
```

```
[]
```

```

```

```

```

BIG FAT WARNING

```
v = [1]
w = [v,v,v]
w
```

```
[[1], [1], [1]]
```

```
v.append(-4)
```

```
w
```

```
[[1, -4], [1, -4], [1, -4]]
```

```
w[0] is w[1]
```

```
True
```

```
w[1] is w[2]
```

```
True
```

```

```

```
v = [1]
w = [copy(v) for _ in range(3)]
w
```

```
[[1], [1], [1]]
```

```
w[0].append(-4)
```

```
w
```

```
[[1, -4], [1], [1]]
```

```

```

```

```

Tuples

```
t = (12345, 54321, 'hello!')
t[0]
```

```
12345
```

```
t
```

```
(12345, 54321, 'hello!')
```

```
type(t)
```

```
<type 'tuple'>
```

```
t[0] = 5   # tuples are immutable
```

```
Exception (click to the left for traceback):
...
TypeError: 'tuple' object does not support item assignment
```

```
# Tuples may be nested:
u = (t, (1, 2, 3, 4, 5))
u
```

```
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

```
t = 12345, 54321, 'hello!'
t
```

```
(12345, 54321, 'hello!')
```

```
# tuple UNPACKING
x, y, z = t
```

```
print x, y, z
```

```
12345 54321 hello!
```

```
def foo(a,b):
    return a+b, a-b
```

```
c,d = foo(2,3)    # multiple return values!
```

```
print c, d
```

```
5 -1
```

```

```

```

```

```

```

Strings

```
'sage math'
```

```
'sage math'
```

```
'doesn\'t'
```

```
'doesn\\u0027t'
```

```
"doesn't"
```

```
"doesn't"
```

```
'"Yes," he said.'
```

```
'"Yes," he said.'
```

```
"\"Yes,\" he said."           # broken in sage-2.11
```

```
'"Yes," he said.'
```

```
hello = "This is a rather long string containing\n\
several lines of text just as you would do in C.\n\
    Note that whitespace at the beginning of the line is\
 significant."

print hello
```

```
This is a rather long string containing
several lines of text just as you would do in C.
    Note that whitespace at the beginning of the line is
significant.
```

```
hello = r"This is a rather long string containing\n\
several lines of text much as you would do in C."
print hello
```

```
This is a rather long string containing\nseveral lines of text mucl
as you would do in C.
```

```
print """
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
"""
```

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

```
word = "Sage"
```

```
word*5
```

```
'SageSageSageSageSage'
```

```
len(word * 5)
```

```
20
```

```
word[1]
```

```
'a'
```

```
word[0:2]
```

```
'Sa'
```

```
word[2:]
```

```
'ge'
```

```

```

```

```

Dictionaries

```
d = {'sage':'math', 1:[1,2,3]}; d
```

```
{1: [1, 2, 3], 'sage': 'math'}
```

```
d['sage']
```

```
'math'
```

```
d[1]
```

```
[1, 2, 3]
```

```
d.keys()
```

```
[1, 'sage']
```

```
d.values()
```

```
[[1, 2, 3], 'math']
```

```
d.has_key('sage')
```

```
True
```

```
'sage' in d
```

```
True
```

```
del d[1]
d
```

```
{'sage': 'math'}
```

```
dict( [(1, [1,2,3]), ('sage', 'math')])
```

```
{1: [1, 2, 3], 'sage': 'math'}
```

```
dict( [(x, x^2) for x in [1..5]] )
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

```
dict( (x, x^2) for x in [1..5] )
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

```
hash('sage')
```

```
-596024308
```

```
hash([1,2])
```

```
Exception (click to the left for traceback):
...
TypeError: list objects are unhashable
```

```
d = {[1,2]: 5}
```

```
Exception (click to the left for traceback):
...
TypeError: list objects are unhashable
```

```
hash( (1,2) )
```

```
1299869600
```

```
d = {(1,2): 5}
```

```
basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
fruit = set(basket)               # create a set without
duplicates
fruit
```

```
set(['orange', 'pear', 'apple', 'banana'])
```

Sets

```
set( (1,2,1,5,1,1) )
```

```
set([1, 2, 5])
```

```

```

```
a = set('abracadabra'); b = set('alacazam')
print a
```

```
set(['a', 'r', 'b', 'c', 'd'])
```

```
print b
```

```
set(['a', 'c', 'z', 'm', 'l'])
```

```
a - b    # letters in a but not in b
```

```
set(['r', 'b', 'd'])
```

```
a | b    # letters in either a or b
```

```
set(['a', 'c', 'b', 'd', 'm', 'l', 'r', 'z'])
```

```
a & b    # letters in both a and b
```

```
set(['a', 'c'])
```

```
v = range(10^6)
```

```
time 10^5 in v
```

```
True
CPU time: 0.16 s,  Wall time: 0.18 s
```

```
time w = set(v)
```

```
CPU time: 0.12 s,  Wall time: 0.12 s
```

```
time 10^5 in w
```

```
True
CPU time: 0.00 s,  Wall time: 0.00 s
```

```
Primes()
```

```
Set of all prime numbers: 2, 3, 5, 7, ...
```

```
type(Primes())
```

```
<class 'sage.sets.primes.Primes_class'>
```

```
5 in Primes()
```

```
True
```

```
6 in Primes()
```

```
False
```