Tuple

denoted with parentheses

In [1]:
```
t = (12,-1)
print type(t)
```

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

In [2]:
```
print isinstance(t,tuple)
print len(t)
```

```
True
2
```

In [3]:
```
t = (12,"monty",True,-1.23e6)
print t[1]
```

```
monty
```

In [4]:
```
print t[-1]
```

```
-1230000.0
```

In [5]:
```
t[-2:]  # get the last two elements, return as a tuple
```

Out[5]:
```
(True, -1230000.0)
```

In [6]:
```
x = (True) ; print type(x)
x = (True,) ; print type(x)
```

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

In [7]:
```
type(()), len(())
```

Out[7]:
```
(tuple, 0)
```

In [8]:
```
type((,))
```

```
  File "<ipython-input-8-21eccbe9b1de>", line 1
    type((,))
          ^
SyntaxError: invalid syntax
```

single-element tuples look like `(element,)`

cannot change a tuple but you can create new one with concatenation

In [9]:
```
t[2] = False
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/Users/jbloom/Classes/python-
bootcamp/DataFiles_and_Notebooks/02_AdvancedDataStructures/<ipython-input-9-
9365ccccf007> in <module>()
```

```
----> 1 t[2] = False

TypeError: 'tuple' object does not support item assignment
```

In [10]:
```
t[0:2], False, t[3:]
```

Out[10]:
```
((12, 'monty'), False, (-1230000.0,))
```

In [11]:
```
## the above it
## not what we wanted... need to concatenate
t[0:2] + False + t[3:]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/Users/jbloom/Classes/python-
bootcamp/DataFiles_and_Notebooks/02_AdvancedDataStructures/<ipython-input-11-
73d4c94ec2bf> in <module>()
      1 ## the above it

      2 ## not what we wanted... need to concatenate

----> 3 t[0:2] + False + t[3:]

TypeError: can only concatenate tuple (not "bool") to tuple
```

In [12]:
```
y = t[0:2] + (False,) + t[3:] ; print y
```

```
(12, 'monty', False, -1230000.0)
```

In [13]:
```
t*2
```

Out[13]:
```
(12, 'monty', True, -1230000.0, 12, 'monty', True, -1230000.0)
```

List

denoted with a brackets

In [14]:
```
v = [1,2,3] ; print len(v), type(v)
```

```
3 <type 'list'>
```

In [15]:
```
v[0:2]
```

Out[15]:
```
[1, 2]
```

In [16]:
```
v = ["eggs","spam",-1,("monty","python"),[-1.2,-3.5]]
len(v)
```

Out[16]:
```
5
```

In [17]:
```
v[0] ="green egg"
v[1] += ",love it."
v[-1]
```

Out[17]:
```
[-1.2, -3.5]
```

In [18]:
```
v[-1][1] = None ; print v
```

```
['green egg', 'spam,love it.', -1, ('monty', 'python'), [-1.2, None]]
```

In [19]:
```
v = v[2:] ; print v
```

```
[-1, ('monty', 'python'), [-1.2, None]]
```

In [20]:
```
# let's make a proto-array out of nested lists
vv = [ [1,2], [3,4] ]
```

In [21]:
```
print len(vv)
```

```
2
```

In [22]:
```
determinant = vv[0][0]*vv[1][1] - vv[0][1]*vv[1][0]
print determinant
```

```
-2
```

the main point here: lists are **changeable**

[back to slides]

List

lists can be extended, appended, and popped

In [23]:
```
v = [1,2,3]
v.append(4)
v.append([-5]) ; print v
```

```
[1, 2, 3, 4, [-5]]
```

In [24]:
```
v = v[:4]
w = ['elderberries', 'eggs']
v + w
```

Out[24]:
```
[1, 2, 3, 4, 'elderberries', 'eggs']
```

In [25]:
```
v.extend(w) ; print v
```

```
[1, 2, 3, 4, 'elderberries', 'eggs']
```

In [26]:
```
v.pop()
```

Out[26]:
```
'eggs'
```

In [27]:
```
print v
```

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

In [28]:
```
v.pop(0) ; print v ## pop the first element
```

```
[2, 3, 4, 'elderberries']
```

- `.append()`: adds a new element
- `.extend()`: concatenates a list/element

- `.pop()`: remove an element

lists can be searched, sorted, & counted

In [29]:
```
v = [1,3, 2, 3, 4, 'elderberries']
v.sort() ; print v
```

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

`reverse` is a keyword of the `.sort()` method

In [30]:
```
v.sort(reverse=True) ; print v
```

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

`.sort()` changes the the list in place

In [31]:
```
v.index(4)   ## lookup the index of the entry 4
```

Out[31]:
```
1
```

In [32]:
```
v.index(3)
```

Out[32]:
```
2
```

In [33]:
```
v.count(3)
```

Out[33]:
```
2
```

In [34]:
```
v.insert(0,"it's full of stars") ; print v
```

```
["it's full of stars", 'elderberries', 4, 3, 3, 2, 1]
```

In [35]:
```
v.remove(1) ; print v
```

```
["it's full of stars", 'elderberries', 4, 3, 3, 2]
```

[back]

List

iteration

In [37]:
```
a = ['cat', 'window', 'defenestrate']
for x in a:
       print x, len(x)
```

```
cat 3
window 6
defenestrate 12
```

In [38]:
```
for i,x in enumerate(a):
       print i, x, len(x)
```

```
0 cat 3
1 window 6
2 defenestrate 12
```

In [39]:
```
for x in a:
       print x,
```

```
cat window defenestrate
```

The syntax for iteration is...

```
for variable_name in iterable:
   # do something with variable_name
```

The `range()` function

In [41]:
```
x = range(4) ; print x
total = 0
for val in range(4):
        total += val
        print "By adding " + str(val) + \
              " the total is now " + str(total)
```

```
[0, 1, 2, 3]
By adding 0 the total is now 0
By adding 1 the total is now 1
By adding 2 the total is now 3
By adding 3 the total is now 6
```

`range([start,] stop[, step])` → list of integers

In [42]:
```
total = 0
for val in range(1,10,2):
    total += val
    print "By adding " + str(val) + \
          " the total is now " + str(total)
```

```
By adding 1 the total is now 1
By adding 3 the total is now 4
By adding 5 the total is now 9
By adding 7 the total is now 16
By adding 9 the total is now 25
```

In [43]:
```
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
    print i, a[i]
```

```
0 Mary
1 had
2 a
3 little
4 lamb
```

Sets

denoted with a curly braces

In [45]:
```
{1,2,3,"bingo"}
```

Out[45]:
```
set(['bingo', 1, 2, 3])
```

In [47]:
```
print type({1,2,3,"bingo"})
```

```
<type 'set'>
```

In [48]:
```
print  type({})
```

```
<type 'dict'>
```

In [50]:
```
print type(set())
```

```
<type 'set'>
```

In [51]:
```
set("spamIam")
```

Out[51]:
```
set(['a', 'p', 's', 'm', 'I'])
```

sets have unique elements. They can be compared, differenced, unionized, etc.

In [52]:
```
a = set("sp"); b = set("am"); print a ; print b
```

```
set(['p', 's'])
set(['a', 'm'])
```

In [53]:
```
c = set(["a","m"])
c == b
```

Out[53]:
```
True
```

In [54]:
```
"p" in a
```

Out[54]:
```
True
```

In [55]:
```
"ps" in a
```

Out[55]:
```
False
```

In [57]:
```
q = set("spamIam")
a.issubset(q)
```

Out[57]:
```
True
```

In [58]:
```
a | b
```

Out[58]:
```
set(['a', 'p', 's', 'm'])
```

In [59]:
```
q - (a | b)
```

Out[59]:
```
set(['I'])
```

In [60]:
```
q & (a | b)
```

Out[60]:
```
set(['a', 'p', 's', 'm'])
```

Like lists, we can use as (unordered) buckets `.pop()` gives us a random element

In [62]:
```
# this is pretty volitile...wont be the same
# order on all machines
for i in q & (a | b):
```

```
    print i,
```

```
a p s m
```

In [63]:
```
q.remove("a")
```

In [64]:
```
q.pop()
```

Out[64]:
```
'p'
```

In [65]:
```
print q.pop()
print q.pop()
```

```
s
m
```

In [66]:
```
print q.pop()
```

```
I
```

In [68]:
```
q.pop()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/Users/jbloom/Classes/python-
bootcamp/DataFiles_and_Notebooks/02_AdvancedDataStructures/<ipython-input-68-
16da542f89c5> in <module>()
----> 1 q.pop()

KeyError: 'pop from an empty set'
```

[back]

4 ways to make a Dictionary

In [69]:
```
# number 1...you've seen this
d = {"favorite cat": None, "favorite spam": "all"}
```

In [70]:
```
# number 2
d = dict(one = 1, two=2,cat = 'dog') ; print d
```

```
{'cat': 'dog', 'two': 2, 'one': 1}
```

In [71]:
```
# number 3 ... just start filling in items/keys
d = {}  # empty dictionary
d['cat'] = 'dog'
d['one'] = 1
d['two'] = 2
d
```

Out[71]:
```
{'cat': 'dog', 'one': 1, 'two': 2}
```

In [72]:
```
# number 4... start with a list of tuples
mylist = [("cat","dog"), ("one",1),("two",2)]
```

```
print dict(mylist)
```

```
{'one': 1, 'two': 2, 'cat': 'dog'}
```

In [73]:
```
dict(mylist) == d
```

Out[73]:
```
True
```

Dictionaries: they can be complicated (in a good way)

In [75]:
```
d = {"favorite cat": None, "favorite spam": "all"}
```

In [78]:
```
d = {'favorites': {'cat': None, 'spam': 'all'}, \
     'least favorite': {'cat': 'all', 'spam': None}}
print d['least favorite']['cat']
```

```
all
```

remember: the backslash () allows you to across break lines. Not technically needed when defining a dictionary or list

In [79]:
```
phone_numbers = {'family': [('mom','642-2322'),('dad','534-2311')],\
                        'friends': [('Billy','652-2212')]}
```

In [80]:
```
for group_type in ['friends','family']:
        print "Group " + group_type + ":"
        for info in phone_numbers[group_type]:
             print " ",info[0], info[1]
```

```
Group friends:
  Billy 652-2212
Group family:
  mom 642-2322
  dad 534-2311
```

In [81]:
```
# this will return a list, but you dont know in what order!
phone_numbers.keys()
```

Out[81]:
```
['friends', 'family']
```

In [82]:
```
phone_numbers.values()
```

Out[82]:
```
[[('Billy', '652-2212')], [('mom', '642-2322'), ('dad', '534-2311')]]
```

`.keys()` and `.values()`: are called `methods` on dictionaries

In [83]:
```
for group_type in phone_numbers.keys():
        print "Group " + group_type + ":"
        for info in phone_numbers[group_type]:
             print " ",info[0], info[1]
```

```
Group friends:
  Billy 652-2212
Group family:
  mom 642-2322
  dad 534-2311
```

we cannot ensure ordering here of the groups

In [84]:
```
groups = phone_numbers.keys()
```

```
In [84]: groups = phone_numbers.keys()
         groups.sort()
         for group_type in groups:
                 print "Group " + group_type + ":"
                 for info in phone_numbers[group_type]:
                      print "  ",info[0], info[1]
```

```
Group family:
  mom 642-2322
  dad 534-2311
Group friends:
  Billy 652-2212
```

`.iteritems()` is a handy method, returning key,value pairs with each iteration

```
In [85]: for group_type, vals in phone_numbers.iteritems():
                 print "Group " + group_type + ":"
                 for info in vals:
                      print "  ",info[0], info[1]
```

```
Group friends:
  Billy 652-2212
Group family:
  mom 642-2322
  dad 534-2311
```

Some examples of getting values:

```
In [86]: phone_numbers['co-workers']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/Users/jbloom/Classes/python-
bootcamp/DataFiles_and_Notebooks/02_AdvancedDataStructures/<ipython-input-86-
92d1a5b9b960> in <module>()
----> 1 phone_numbers['co-workers']

KeyError: 'co-workers'
```

```
In [87]: phone_numbers.has_key('co-workers')
```

```
Out[87]: False
```

```
In [88]: print phone_numbers.get('co-workers')
```

```
None
```

```
In [89]: phone_numbers.get('friends') == phone_numbers['friends']
```

```
Out[89]: True
```

```
In [90]: print phone_numbers.get('co-workers',"all alone")
```

```
all alone
```

setting values
you can edit the values of keys and also `.pop()` & `del` to remove certain keys

```
In [91]: # add to the friends list
```

```
phone_numbers['friends'].append(("Marsha","232-1121"))
print phone_numbers
```

```
{'friends': [('Billy', '652-2212'), ('Marsha', '232-1121')], 'family': [('mom',
'642-2322'), ('dad', '534-2311')]}
```

In [92]:
```
## billy's number changed
phone_numbers['friends'][0][1] = "532-1521"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/Users/jbloom/Classes/python-
bootcamp/DataFiles_and_Notebooks/02_AdvancedDataStructures/<ipython-input-92-
564c1535cc4d> in <module>()
      1 ## billy's number changed

----> 2 phone_numbers['friends'][0][1] = "532-1521"

TypeError: 'tuple' object does not support item assignment
```

In [93]:
```
phone_numbers['friends'][0] = ("Billy","532-1521")
```

In [94]:
```
## I lost all my friends preparing for this Python class
phone_numbers['friends'] = [] # sets this to an empty list
```

In [95]:
```
## remove the friends key altogether
print phone_numbers.pop('friends')
```

```
[]
```

In [96]:
```
print phone_numbers
```

```
{'family': [('mom', '642-2322'), ('dad', '534-2311')]}
```

In [97]:
```
del phone_numbers['family']
```

In [98]:
```
print phone_numbers
```

```
{}
```

`.update()` method is very handy, like `.append()` for lists

In [99]:
```
phone_numbers.update({"friends": [("Billy's Brother, Bob", "532-1521")]})
print phone_numbers
```

```
{'friends': [("Billy's Brother, Bob", '532-1521')]}
```

[back]

In [ ]:
```

```