Data Structures

There are four main types of collections of data:

("Sequence objects")

- List: a mutable array of data
- Tuples: ordered, immutable list
- Sets: unordered collection of unique elements

The value in each element can be whatever (type) you want

List

lists can be extended & appended

```
>>> v = [1,2,3]
>>> v.append(4)
>>> print v
[1,2,3,4]
```

Lists can be considered **objects**.
Objects are like animals: they know how to do stuff (like eat and sleep), they know how to interact with others (like make children), and they have characteristics (like height, weight).

"Knowing how to do stuff" with itself is called a **method**. In this case "append" is a method which, when invoked, is an action that changes the characteristics (the data vector of the list itself).

ipython is your new best friend

quick look at what's available & what it does

```
In [205]: v.
v.__add__            v.__getattribute__   v.__le__             v.__reversed__       v.index
v.__class__          v.__getitem__        v.__len__            v.__rmul__           v.insert
v.__contains__       v.__getslice__       v.__lt__             v.__setattr__        v.pop
v.__delattr__        v.__gt__             v.__mul__            v.__setitem__        v.remove
v.__delitem__        v.__hash__           v.__ne__             v.__setslice__       v.reverse
v.__delslice__       v.__iadd__           v.__new__            v.__str__            v.sort
v.__doc__            v.__imul__           v.__reduce__         v.append
v.__eq__             v.__init__           v.__reduce_ex__      v.count
v.__ge__             v.__iter__           v.__repr__           v.extend
```

tab

tab

these are special methods of lists, we generally dont use them

```
In [205]: v.re
v.remove   v.reverse

In [205]: v.remove?
Type:        builtin_function_or_method
Base Class: <type 'builtin_function_or_method'>
String Form:    <built-in method remove of list object at 0x10169b710>
Namespace:  Interactive
Docstring:
    L.remove(value) -- remove first occurrence of value
```

Dictionaries

denoted with a curly braces and colons

```
>>> d = {"favorite cat": None, "favorite spam": "all"}
```

these are key: value, key: value, ...

```
>>> print d["favorite cat"]
None
>>> d[0]   ## this is not a list and you dont have a keyword = 0
KeyError: 0
>>> e = {"favorite cat": None, "favorite spam": "all", 1: 'loneliest number'}
>>> e[1] is 'loneliest number'
True
>>> e
{1: 'loneliest number', 'favorite cat': None, 'favorite spam': 'all'}
```

dictionaries are UNORDERED*. You cannot assume that one key comes before or after another

* you can use a special type of ordered dict if you really need it:
http://docs.python.org/whatsnew/2.7.html#pep-372-adding-an-ordered-dictionary-to-collections

Casting back and forth

You can go back and forth between tuples and lists

```
>>> a = [1,2,3,("b",1)]
>>> b = tuple(a) ; print b
(1, 2, 3, ('b', 1))
>>> print list(b)
[1, 2, 3, ('b', 1)]
>>> set(a)
set([1, 2, 3, ('b', 1)])
>>> list(set("spam"))
['a', 'p', 's', 'm']
```

casting only affects top-level structure, not the elements

List Comprehension

You can create lists "on the fly" by asking simple questions of other iterateable data structures

example: I want a list of all numbers from 0 - 100 whose lowest two bits are both one (e.g., 3, 7, ...) but is not divisible by 11

Old Way

```
>>> mylist = []
>>> for num in range(101):
...     if (num & 2) and (num & 1) and (num % 11 != 0.0):
...         mylist.append(num)
>>> print mylist
[3, 7, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 59, 63, 67, 71, 75, 79, 83, 87, 91, 95]
```

New Way

```
>>> mylist=[num for num in range(101) if (num & 2) and (num & 1) and (num % 11 != 0.0)]
>>> print mylist
[3, 7, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 59, 63, 67, 71, 75, 79, 83, 87, 91, 95]
```

List Comprehension

example: I want a list of all mesons whose masses are between 100 and 1000 MeV

```
>>> particles = [{"name":"π+"  ,"mass": 139.57018}, {"name":"π0"  ,"mass": 134.9766},
             {"name":"η5"  ,"mass": 47.853}, {"name":"η'(958)","mass": 957.78},
             {"name":"ηc(1S)", "mass": 2980.5}, {"name": "ηb(1S)","mass": 9388.9},
             {"name":"K+",  "mass": 493.677}, {"name":"K0"  ,"mass": 497.614},
             {"name":"K0S" ,"mass":  497.614}, {"name":"K0L" ,"mass":  497.614},
             {"name":"D+"  ,"mass": 1869.62}, {"name":"D0"  ,"mass": 1864.84},
             {"name":"D+s" ,"mass":  1968.49}, {"name":"B+"  ,"mass": 5279.15},
             {"name":"B0"  ,"mass": 5279.5}, {"name":"B0s" ,"mass":  5366.3},
             {"name":"B+c" ,"mass":    6277}]
>>> my_mesons = [ (x['name'],x['mass']) for \
                  x in particles if x['mass'] <= 1000.0 and x['mass'] >= 100.0]
>>> # get the average
>>> tot = 0.0 ; for x in my_mesons: tot+= x[1]
>>> print "The average meson mass in this range is " + str(tot/len(my_mesons)) + " MeV/c^2."
The average meson mass in this range is 459.835111429 MeV/c^2.
>>> my_mesons[0][0]
'\xcf\x80+'
>>> print my_mesons[0][0]
π+
```

data source: http://en.wikipedia.org/wiki/List_of_mesons

Breakout Session Work

consider the following data (file: airline.py):

```
airports = {"DCA": "Washington, D.C.", "IAD": "Dulles", "LHR": "London-Heathrow", \
            "SVO": "Moscow", "CDA": "Chicago-Midway", "SBA": "Santa Barbara", "LAX": "Los Angeles",\
            "JFK": "New York City", "MIA": "Miami", "AUM": "Austin, Minnesota"}

# airline, number, heading to, gate, time (decimal hours)
flights = [("Southwest",145,"DCA",1,6.00),("United",31,"IAD",1,7.1),("United",302,"LHR",5,6.5),\
          ("Aeroflot",34,"SVO",5,9.00),("Southwest",146,"CDA",1,9.60), ("United",46,"LAX",5,6.5),\
          ("Southwest",23,"SBA",6,12.5),("United",2,"LAX",10,12.5),("Southwest",59,"LAX",11,14.5),\
          ("American", 1,"JFK",12,11.3),("USAirways", 8,"MIA",20,13.1),("United",2032,"MIA",21,15.1),\
          ("SpamAir",1,"AUM",42,14.4)]
```

1. print out a schedule organized by airline:

```
Flight         Destination             Gate   Time
-------------------------------------------------
Aeroflot 34    Moscow                    5    9.0
American 1     New York City            12   11.3
Southwest 23   Santa Barbara             6   12.5
Southwest 59   Los Angeles              11   14.5
...
```

2. print out a schedule organized by time

hint: you'll need to do a manual sorting on the last element of each flight element, before beginning the printing loop