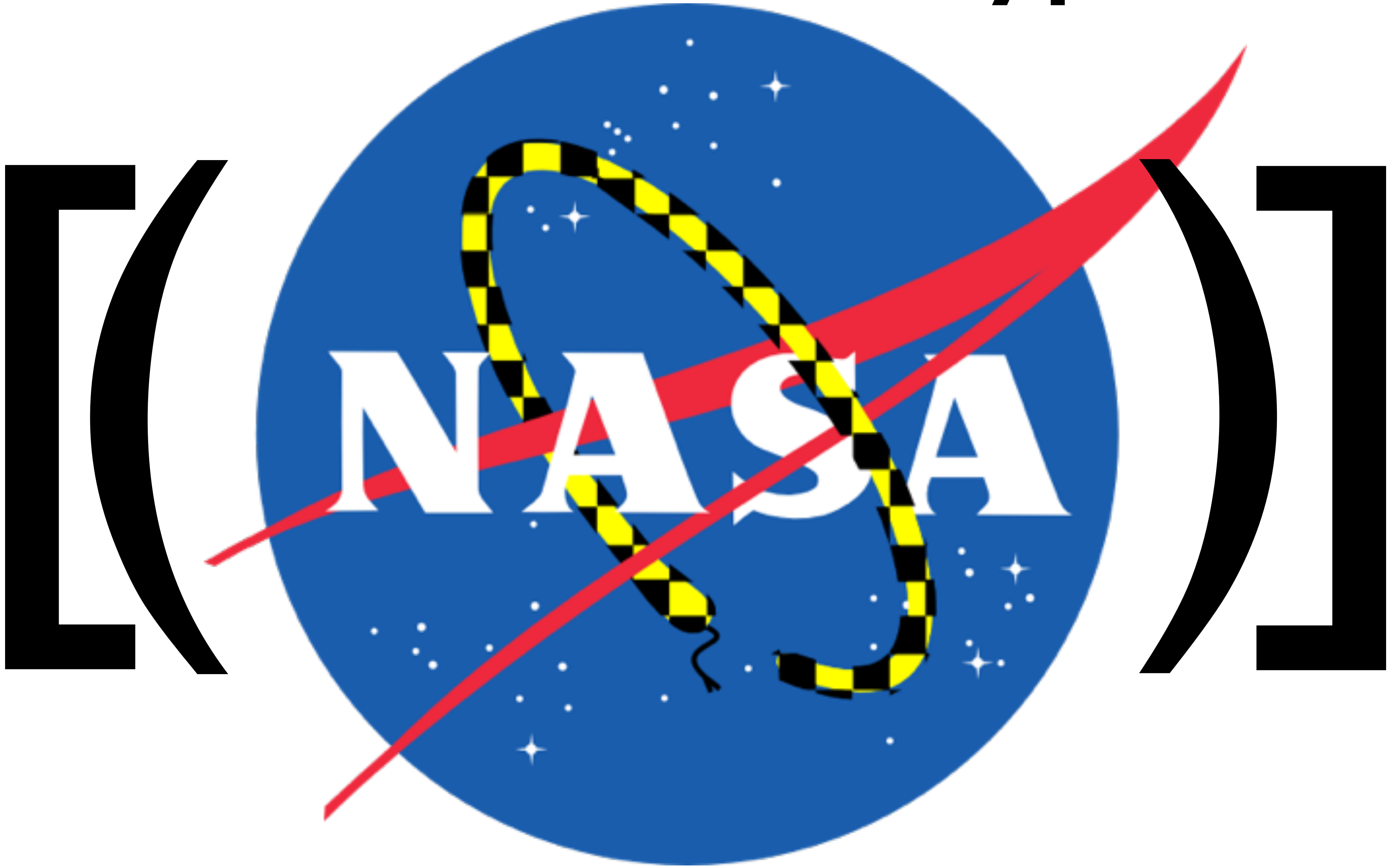
Advanced Data Types



Follow along in iPython notebook:
DataFiles_and_Notebooks/02_AdvancedDataStructures/02_AdvancedDataStructures.ipynb

There are 4 main types of collections of data:

("Sequence objects")

- **List**: a mutable array of data
- **Tuples**: ordered, immutable list
- **Sets:** unordered collection of unique elements
- **Dictionary**: keyword/value lookup

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

List

denoted with a brackets

```
>>> v = [1,2,3] ; print len(v), type(v)
3,<type "list">
>>> v[0:2]
[1,2]
>>> v = ["eggs","spam",-1,("monty","python"),[-1.2,-3.5]]
>>> len(v)
5
>>> v[0] ="green egg"
>>> v[1] += ",love it."
['green egg', 'spam,love it.', -1, ('monty', 'python'), [-1.2, -3.5]]
>>> v[-1]
[-1.2, -3.5]
>>> v[-1][1] = None ; print v
['green egg', 'spam,love it.', -1, ('monty', 'python'), [-1.2, None]]
>>> v = v[2:] ; print v
[-1, ('monty', 'python'), [-1.2, None]]
>>> # let's make a proto-array out of nested lists
>>> vv = [ [1,2], [3,4] ]
>>> determinant = vv[0][0]*vv[1][1] - vv[0][1]*vv[1][0]
```

lists are changeable

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).

List

lists can be extended, appended, and popped

```
>>> v.append([-5])
>>> print v
[1,2,3,4,[-5]]
>>> v = v[:4]
>>> w = ['elderberries', 'eggs']
>>> v + w
[1,2,3,4,'elderberries','eggs']
>>> v.extend(w) ; print v
[1,2,3,4,'elderberries','eggs']
>>> v.pop()
'eggs'
>>> print v
[1,2,3,4,'elderberries']
>>> v.pop(0) ; print v ## pop the first element
1
[2, 3, 4, 'elderberries']
```

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

List

lists can be searched, sorted, & counted

```
>>> v = [1,3, 2, 3, 4, 'elderberries']
>>> v.sort() ; print v
[1, 2, 3, 3, 4, 'elderberries']
>>> v.sort(reverse=True) ; print v
['elderberries', 4, 3, 3, 2, 1]
>>> v.index(4)   ## lookup the index of the entry 4
1
>>> v.index(3)  # get the first occurrence of the number 3
2
>>> v.count(3)
2
>>> v.insert(0,"it's full of stars") ; print v
["it's full of stars", 'elderberries', 4, 3, 3, 2, 1]
>>> v.remove(1) ; print v
["it's full of stars", 'elderberries', 4, 3, 3, 2]
```

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

ipython is your new best friend

quick look at what's available & what it does

special methods of lists,
we generally don't use

```
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

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
```

tab

tab

List

iteration

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

cat 3
window 6
defenestrate 12
>>>
>>> for i,x in enumerate(a):
       print i, x, len(x)
0 cat 3
1 window 6
2 defenestrate 12
>>> for x in a:
       print x,
cat window defenestrate
```

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

List

range()

```
>>> x = range(4) ; print x
[0, 1, 2, 3]
>>> total = 0
>>> for val in range(4):
        total += val
        print "By adding " + str(val) + " the total is now " + str(total)
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

```
>>> 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
```

List

range()

```
>>> 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
```

Tuple

denoted with parentheses

```
>>> t = (12,-1)
>>> print type(t)
<type "tuple">
>>> isinstance(t,tuple)
True
>>> len(t)
2
>>> t = (12,"monty",True,-1.23e6)
>>> t[1]
'monty'
>>> t[-1]
-1.23e6
>>> t[-2:]  # get the last two elements, return as a tuple
(True, -1230000.0)
>>> x = (True) ; type(x)
<type "bool">
>>> x = (True,) ; type(x)
<type "tuple">
>>> type(()), len(())
(<type "tuple">, 0)
```

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

Tuple

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

```
>>> t[2] = False
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> t[0:2], False, t[3:]
((12, 'monty'), True, (-1230000.0,))
>>> ## not what we wanted... need to concatenate
>>> t[0:2] + False + t[3:]
TypeError: can only concatenate tuple (not "bool") to tuple
>>> y = t[0:2] + (False,) + t[3:] ; print y
(12, 'monty', False, -1230000.0)
>>> t*2
(12, 'monty', True, -1230000.0, 12, 'monty', True, -1230000.0)
```

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

Why use tuples when you have lists?

Sometimes, immutability is what you need

```
>>> continents = (“North America”, “South America”, “Europe”, “Asia”,
“Australia”, “Antarctica”) # Something that you probably don’t want changed

>>> tasks = [“learn Python”,”eat dinner”,”climb Mt. Everest”] # Something you
might want to add and subtract from
```

since any comma-ordered sequence without parenthesis is also a tuple, so you’ll see them in other contexts

```
>>> first_name, last_name = “Jack”, “Hewitt” # This is a tuple assignment
>>> print “My name is”, first_name, last_name # This statement prints a tuple
My name is Jack Hewitt
>>> a, b, c = some_function(x,y) # This function returns a tuple (more later…)
```

Sets

denoted with a curly braces

```
>>> {1,2,3,"bingo"}
set(['bingo', 1, 2, 3])
>>> type({1,2,3,"bingo"})
<type 'set'>
>>> type({})
<type 'dict'>
>>> type(set())
<type 'set'>
>>> set("spamIam")
set(['a', 'p', 's', 'm', 'I'])
```

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

```
>>> a = set("sp"); b = set("am"); print a ; print b
set(['p', 's'])
set(['a', 'm'])
>>> c = set(["a","m"])
>>> c == b
True
>>> "p" in a
True
>>> "ps" in a
False
```

```
>>> q = set("spamIam")
>>> a.issubset(q)
True
>>> a | b
set(['a', 'p', 's', 'm'])
>>> q - (a | b)
set(['I'])
>>> q & (a | b)
set(['a', 'p', 's', 'm'])
```

Like lists, we can use as (unordered) buckets

.pop() gives us a random element

```
>>> q.remove("a")
>>> q.pop()
'p'
>>> q.pop()
's'
>>> q.pop()
'm'
>>> q.pop()
'I'
>>> q.pop()
Traceback (most recent call last):
  File "<ipython-input-39-16da542f89c5>", line 1, in <module>
    q.pop()
KeyError: 'pop from an empty set'
```

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

4 ways to make a Dictionary

```
>>> # number 1...you've seen this
>>> d = {"favorite cat": None, "favorite spam": "all"}
>>> # number 2
>>> d = dict(one = 1, two=2,cat = 'dog') ; print d
{'cat': 'dog', 'one': 1, 'two': 2}
>>> # number 3 ... just start filling in items/keys
>>> d = {}  # empty dictionary
>>> d['cat'] = 'dog'
>>> d['one'] = 1
>>> d['two'] = 2
>>> d
{'cat': 'dog', 'one': 1, 'two': 2}
>>> # number 4... start with a list of tuples
>>> mylist = [("cat","dog"), ("one",1),("two",2)]
>>> print dict(mylist)
{'cat': 'dog', 'one': 1, 'two': 2}
```

Dictionaries

they can be complicated (in a good way)

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

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

```
>>> phone_numbers = {'family': [('mom','642-2322'),('dad','534-2311')],\
                     'friends': [('Sylvia','652-2212')]}
>>> for group_type in ['friends','family']:
        print "Group " + group_type + ":"
        for info in phone_numbers[group_type]:
             print " ",info[0], info[1]
Group friends:
 Syvlia 652-2212
Group family:
 mom 642-2322
 dad 534-2311
```

Dictionaries

```
>>> phone_numbers = {'family': [('mom','642-2322'),('dad','534-2311')],\
                     'friends': [('Billy','652-2212')]}
>>> phone_numbers.keys()  # this will return a list, but you dont know in what order!
['friends','family']
>> phone_numbers.values()
[[('mom','642-2322'),('dad','534-2311')], [('Billy','652-2212')]]
```

.keys() and .values(): methods on dictionaries

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

we cannot ensure ordering here of the groups

```
>>> 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]
```

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

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

getting values

```
>>> phone_numbers['co-workers']
KeyError: 'co-workers'
>>> phone_numbers.has_key('co-workers')
False
>>> print phone_numbers.get('co-workers')     # no error!
None
>>> phone_numbers.get('friends') == phone_numbers['friends']
```

setting values

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

```
>>> phone_numbers['friends'].append(("Jeremy","232-1121"))# add to the friends list
>>> print phone_numbers
{'family': [('mom','642-2322'),('dad','534-2311')],
 'friends': [('Sylvia','652-2212'), ("Jeremy","232-1121")]}
>>> ## Sylvia’s number changed
>>> phone_numbers['friends'][0][1] = "532-1521"
TypeError: 'tuple' object does not support item assignment
>>> phone_numbers['friends'][0] = ("Sylvia","532-1521")
>>> ## I lost all my friends preparing for this Python class
>>> phone_numbers['friends'] = []                     # sets this to an empty list
>>> ## remove the friends key altogether
>>> print phone_numbers.pop('friends')
[]
>>> print phone_numbers
{'family': [('mom','642-2322'),('dad','534-2311')]}
>>> del phone_numbers['family']
```

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

```
>>> phone_numbers.update({"friends": [("Billy's Brother, Bob", "532-1521")]})
>>> print phone_numbers
{'family': [('mom','642-2322'),('dad','534-2311')],
 "friends": [(“Sylvia's Friend, Dave", "532-1521")]}
```

List Comprehension

You can create lists "on the fly" by asking simple questions of other iterate-able 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

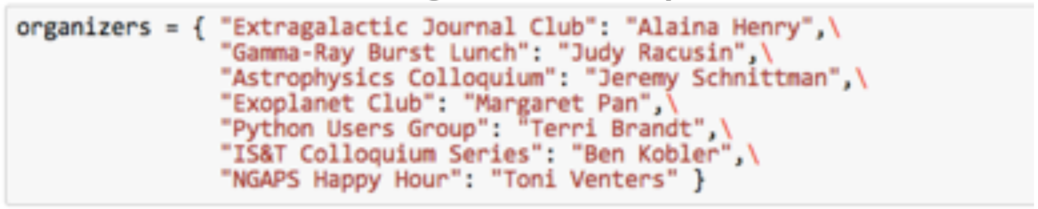
example: I want a list of all mesons whose masses are between 100 and 1000 MeV

```
>>> 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+
```

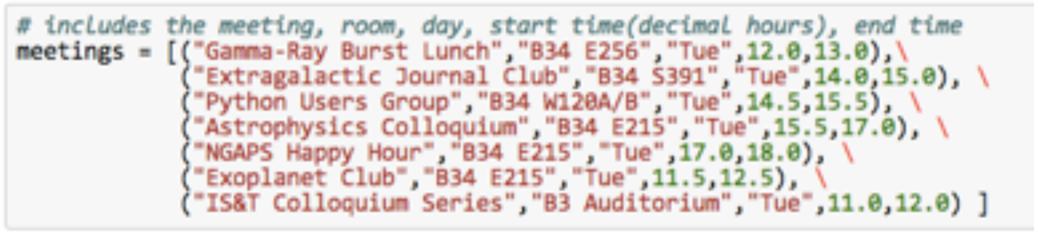
data source: http://en.wikipedia.org/wiki/List_of_mesons

Breakout Session Work

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



```
organizers = { "Extragalactic Journal Club": "Alaina Henry",\
               "Gamma-Ray Burst Lunch": "Judy Racusin",\
               "Astrophysics Colloquium": "Jeremy Schnittman",\
               "Exoplanet Club": "Margaret Pan",\
               "Python Users Group": "Terri Brandt",\
               "IS&T Colloquium Series": "Ben Kobler",\
               "NGAPS Happy Hour": "Toni Venters" }
```



```
# includes the meeting, room, day, start time(decimal hours), end time
meetings = [("Gamma-Ray Burst Lunch","B34 E256","Tue",12.0,13.0),\
            ("Extragalactic Journal Club","B34 S391","Tue",14.0,15.0), \
            ("Python Users Group","B34 W120A/B","Tue",14.5,15.5), \
            ("Astrophysics Colloquium","B34 E215","Tue",15.5,17.0), \
            ("NGAPS Happy Hour","B34 E215","Tue",17.0,18.0), \
            ("Exoplanet Club","B34 E215","Tue",11.5,12.5), \
            ("IS&T Colloquium Series","B3 Auditorium","Tue",11.0,12.0) ]
```

1. print out a schedule organized by meeting:

```
Meeting                             Room No.       Day  Time       Organizer
-------------------------------------------------------------------------------
IS&T Colloquium Series          B3 Auditorium      Tue  11.0       Ben Kobler
Exoplanet Club                  B34 E215           Tue  11.5       Margaret Pan
Gamma-Ray Burst Lunch           B34 E256           Tue  12.0       Judy Racusin
Extragalactic Journal Club      B34 S391           Tue  14.0       Alaina Henry
...
```

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

Extra credit: how would you handle meetings on a different day?

Ask not what you can do for your country. Ask what's for lunch.

(Orson Welles)

[redacted]