Python Applications

CS101 Lecture #13
Administrivia
Homework #7 is due Friday, Nov. 25.
Use the split('','') approach.
Warmup Quiz (No Real Quiz Today)
Which of the following sets of list methods *all* change the function in place (have no return value)?

A. split, append, extend  
B. del, index, upper  
C. read, readlines, close  
D. sort, reverse, append, extend
Which of the following sets of list methods *all* change the function in place (have no return value)?

A  split, append, extend
B  del, index, upper
C  read, readlines, close
D  sort, reverse, append, extend ★
Working with Containers
### Lists and Dictionaries

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>modifies in place</td>
</tr>
<tr>
<td>append</td>
<td>returns value</td>
</tr>
<tr>
<td>extend</td>
<td>index</td>
</tr>
<tr>
<td>reverse</td>
<td>count</td>
</tr>
<tr>
<td>sort</td>
<td>upper</td>
</tr>
<tr>
<td>del (not method)</td>
<td>isupper</td>
</tr>
<tr>
<td></td>
<td>etc.</td>
</tr>
</tbody>
</table>
### Working with Containers

<table>
<thead>
<tr>
<th>dict modifies in place</th>
<th>returns value</th>
</tr>
</thead>
<tbody>
<tr>
<td>del (not method)</td>
<td>values</td>
</tr>
<tr>
<td></td>
<td>keys</td>
</tr>
</tbody>
</table>

Note that there isn't a *sort* method!
### lists and dicts

<table>
<thead>
<tr>
<th>dict</th>
<th>modifies in place</th>
<th>returns value</th>
</tr>
</thead>
<tbody>
<tr>
<td>del (not method)</td>
<td></td>
<td>values keys</td>
</tr>
</tbody>
</table>

- Note that there isn’t a sort method!
# remember me?
def sortDictAsList( d):
    items = list( d.items() )
    items.sort( key=lambda x:x[1] )
    return items

d = { 'a':2, 'b':1, 'c':-1, 'd':14 }
sortDictAsList( d )
We want to know which plankton species has the largest population.
We want to know which plankton species has the largest population.

```python
from csv import DictReader
reader = DictReader(open('plankton.csv'))
plankdata = {}
for row in reader:
    plankdata[row['Species']] = \
        max(float(row['Near-shore, May-93']),
            float(row['Near-shore, Aug-93']),
            float(row['Off-shore, May-93']),
            float(row['Off-shore, Aug-93']))

sortDictAsList(plankdata)
```
Sometimes we have two lists that correspond to each other.

If we want to loop over both together, we have two approaches open:

```python
qs = [ 'name', 'quest', 'favourite colour' ]
as = [ 'Lancelot', 'the Holy Grail', 'blue' ]
# method 1:
for i in range(len(qs)):
    print( 'What is your %s? It is %s.'%(qs[i],as[i]) )
```

Sometimes we have two lists that correspond to each other.

If we want to loop over both together, we have two approaches open:

```python
qs = [ 'name', 'quest', 'favourite colour' ]
as = [ 'Lancelot', 'the Holy Grail', 'blue' ]

# method 1:
for i in range(len(qs)):
    print( 'What is your %s? It is %s.'%(qs[i],as[i]) )

# method 2:
for q,a in zip(qs,as):
    print( 'What is your %s? It is %s.'%(q,a) )
```
zip makes two lists *jointly iterable.*

Consider a function which compares two lists of measurements and determines for each pair of measurements which is larger:

```python
def pickLarger( a,b ):
    result = [ ] # a list of largest values
    for i,j in zip(a,b):
        result.append( max( i,j ) )
    return result
```
What if you need to know both an item and the index of the item?

```python
my_list = [ 'meter', 'kilogram', 'second' ]
# one way
for i in range( len(my_list) ):
    print( '%s is the %sth item.' % (my_list[i],i) )
# another way
for i, item in enumerate( my_list ):
    print( '%s is the %sth item.' % (item,i) )
```
What if you need to know both an item and the index of the item?

```python
my_list = [ 'meter', 'kilogram', 'second' ]
# one way
for i in range( len(my_list) ):
    print( '%s is the %sth item.' % (my_list[i],i) )

# another way
for i,item in enumerate( my_list ):
    print( '%s is the %sth item.' % (item,i) )
```
Both `zip` and `enumerate` are *convenience* functions!

There are multiple approaches!
Permutations are used in statistics to analyze all possible configurations of a group of things.

In engineering, for instance, you see them used in experimental design.

```python
# one way
for i in 'ABCD':
    for j in 'ABCD':
        if i == j:
            continue
        print( i, j )
```

```python
# another way
from itertools import permutations
for doublet in permutations( 'ABCD', 2 ):
    print( doublet )
```
Accessing lists

- *Permutations* are used in statistics to analyze all possible configurations of a group of things.
- In engineering, for instance, you see them used in experimental design.

```python
# one way
for i in 'ABCD':
    for j in 'ABCD':
        if i == j:
            continue
        print( i, j )

# another way
from itertools import permutations
for doublet in permutations( 'ABCD',2 ):
    print( doublet )
```
# how to figure out what directory Python is in
import os
os.getcwd() # Get Current Working Directory

# how to figure out what's in that directory
os.listdir('.')

# when submitting, use:
open( 'batting.csv' ) #(since in same dir)
Reminders
Homework #7 is due Friday, Nov. 25.
Use the `split(',')` approach.