Lect 13 – Python Sorting

Rob Capra

INLS 490-172

Sorting

- We know how to sort a list:

```
>>> a = [3, 4, 5, 1, 2]
>>> print a
[3, 4, 5, 1, 2]
>>> print a.sort()
None
>>> print a
[1, 2, 3, 4, 5]
>>>
```

- But how do you sort a dictionary?

Sorting

- But how do you sort a dictionary?

```
>>> d = { 'd':4, 'b':2, 'a':1, 'c':3 }
>>> print d
{'a': 1, 'c': 3, 'b': 2, 'd': 4}
>>> d.sort()

Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    d.sort()
AttributeError: 'dict' object has no
attribute 'sort`
```

- ???

Sorting

- `sorted` will accept any iterable object:

```
>>> a = [3, 4, 5, 1, 2]
>>> d = { 'd':4, 'b':2, 'a':1, 'c':3 }
>>> sorted(a)
[1, 2, 3, 4, 5]
>>> sorted(d)
['a', 'b', 'c', 'd']
```

But for the dict, sorted returned a sorted list of only the keys

Sorting

- There are a number of ways to do “advanced” sorting in Python
  - http://docs.python.org/2/howto/sorting.html

- Today, we will look at one method that will allow us to sort a dict by its values
  - It will return a sorted list of tuples

sorted key

- `sorted` allows you to specify a **function** to be called on each list element prior to making the sorting comparisons
- The function should take one argument (the item) and return a key to use for sorting purposes.

```
s = "This this IS is UNC unc"
print s.split()
print
print sorted(s.split())
def tolower(i):
    return str.lower(i)
print
for w in s.split():
    print tolower(w)
print
print sorted(s.split(), key=tolower)
```

sorted, key, & itemgetter

- The itemgetter function in the operator module makes it each to access specific items to use as the sort key.

```
from operator import itemgetter
t = [('dave', 'B', 10), ('jane', 'B', 12), ('john',
'A', 15)]
print sorted(t, key=itemgetter(0))
print
print sorted(t, key=itemgetter(1))
print
print sorted(t, key=itemgetter(2))
print
```

Exercise 9

- Read data from file
- Count stuff
- Print out in sorted order by counts

```
course instructor semester
inls523 capra f13
inls760 capra sp14
inls509 arguello f13
inls523 haas f13
inls523 haas f11
inls760 capra s13
inls509 arguello f12
inls523 boone sp13
inls760 capra sp11
inls582 haas f13
inls512 haas sp13
```

ex9_courses.txt

```
inls523 4
inls760 3
inls509 2
inls512 1
inls582 1
```

```
inls760 capra 3
inls509 arguello 2
inls523 haas 2
inls523 boone 1
inls512 haas 1
inls582 haas 1
inls523 capra 1
```

Exercise 9 – Walk-through

- Step 1: Read the file

```
import codecs

fp = codecs.open("courses.txt", encoding="utf-8")
fp.readline()
cis_list = []
for line in fp:
    line = line.strip()
    fields = line.split();
    course = fields[0]
    instructor = fields[1]
    semester = fields[2]
    tmp = {}
    tmp['course'] = course
    tmp['instructor'] = instructor
    tmp['semester'] = semester
    cis_list.append(tmp)
fp.close()
```

Exercise 9 – Walk-through

- Step 2: Count stuff

```
c_counts = {}
for d in cis_list:
    c = d['course']
    c_counts[d['course']] = c_counts.get(d['course'],0) + 1


ci_counts = {}
for d in cis_list:
    c = d['course']
    i = d['instructor']
    ci_counts[(c,i)] = ci_counts.get((c,i),0) + 1
```

Exercise 9 – Walk-through

- Step 3a: Sort stuff

```
sorted_c_counts = sorted(c_counts.items(),
key=itemgetter(1), reverse=True)

sorted_ci_counts = sorted(ci_counts.iteritems(),
key=itemgetter(1), reverse=True)
```

In this case, because we use c_counts.items(), sorted is sorting a list of (key,value) tuples.

itemgetter(1) is specifying that sorted should sort on the second item in the (key,value) tuple, meaning that we will sort on the value, not the key.

Remember .items()?
Returns a list of (key,value) tuples

.items() vs. .iteritems()

iteritems is faster, more efficient, but does basically the same thing.

Exercise 9 – Walk-through

- Step 3b: Print sorted stuff

```
for (course,count) in sorted_c_counts:
    print course, count

for ((course,instructor), count) in sorted_ci_counts:
    print course, instructor, count
```