Sorting

Michael Ernst
UW CSE 140
Winter 2013

Sorting

```
hamlet = "to be or not to be that is the
question whether tis nobler in the mind to
suffer".split()

print "hamlet:", hamlet

print "sorted(hamlet):", sorted(hamlet)
print "hamlet:", hamlet

print "hamlet.sort():", hamlet.sort()
print "hamlet:", hamlet
```

- Lists are mutable – they can be changed
  - including by functions

Customizing the sort order

Goal: sort a list of names *by last name*

```
names = ["Isaac Newton", "Albert Einstein", "Niels
Bohr", "Marie Curie", "Charles Darwin", "Louis
Pasteur", "Galileo Galilei", "Margaret Mead"]

print "names:", names
```

This does not work:

```
print "sorted(names):", sorted(names)
```

When sorting, how should we compare these names?

```
"Niels Bohr"
"Charles Darwin"
```

Sort key

A sort key is a different value that you use to sort a list, instead of the actual values in the list

```
def last_name(str):
    return str.split(" ")[1]

print 'last_name("Isaac Newton"):',
last_name("Isaac Newton")
```

Two ways to use a sort key:

1. Create a new list containing the sort key, and then sort it
2. Pass a key function to the sorted function

1. Use a sort key to create a new list

Create a different list that contains the sort key, sort it, then extract the relevant part

```
# keyed_names is a list of [lastname, fullname] lists (tuples would be better!)
keyed_names = []
for name in names:
  keyed.names.append([last_name(name), name])
# simpler:  keyed_names = [[last_name(name), name] for name in names]

print "keyed_names:", keyed_names
print "sorted(keyed_names):", sorted(keyed_names)

print "sorted(keyed_names, reverse = True):"
print sorted(keyed_names, reverse = True)
```

(This works because Python compares two elements that are lists *elementwise*.)

```
sorted_keyed_names = sorted(keyed_names, reverse = True)
sorted_names = []
for keyed_name in sorted_keyed_names:
  sorted_names.append(keyed_name[1])
# simpler:  sorted_names = [keyed_name[1] for keyed_name in sorted_keyed_names]
print "sorted_names:", sorted_names
```

Digression: Lexicographic Order

Aaron	[1, 9, 9]
Andrew	[2, 1]
Angie	[3]
with	[1]
withhold	[1,1]
withholding	[1,1,1]
Able	
Charlie	
baker	
delta	

2. Use a sort key as the `key` argument

Supply the **`key`** argument to the **`sorted`** function or the **`sort`** function

```
print "sorted(names, key = last_name):"
print sorted(names, key = last_name)

print "sorted(names, key = last_name, reverse = True):"
print sorted(names, key = last_name, reverse = True)

print sorted(names, key = len)

def last_name_len(name):
    return len(last_name(name))

print sorted(names, key = last_name_len)
```

itemgetter is a function that returns a function

```
import operator
operator.itemgetter(2, 7, 9, 10)

operator.itemgetter(2, 7, 9, 10)("dumbstricken")
operator.itemgetter(2, 5, 7, 9)("homesickness")
operator.itemgetter(2, 7, 9, 10)("pumpernickel")
operator.itemgetter(2, 3, 6, 7)("seminaked")
operator.itemgetter(1, 2, 4, 5)("smirker")

operator.itemgetter(9, 7, 6, 1)("beatnikism")
operator.itemgetter(14, 13, 5, 1)("Gedankenexperiment")
operator.itemgetter(12, 10, 9, 5)("mountebankism")
```

Using itemgetter

```
from operator import itemgetter

student_score = ('Robert', 8)
itemgetter(0)(student_score)
itemgetter(1)(student_score)

student_scores =
 [('Robert', 8), ('Alice', 9), ('Tina', 7)]
```

$\Rightarrow$ "Robert"

$\Rightarrow$ 8

- Sort the list by name:

```
sorted(student_scores, key=itemgetter(0) )
```

- Sort the list by score

```
sorted(student_scores, key=itemgetter(1) )
```

Sorting based on two criteria

Two approaches:

1. Use an itemgetter with two arguments
2. Sort twice (most important sort *last*)

```
student_scores = [('Robert', 8), ('Alice', 9),
('Tina', 10), ('James', 8)]
```

Goal: sort based on score; within score, by name

1. `sorted(student_scores, key=itemgetter(1,0) )`
2. `result = sorted(student_scores, key=itemgetter(0) )`
   `result = sorted(result, key=itemgetter(1) )`

More sorting based on two criteria

If you want to sort different criteria in different directions, you must use multiple calls to **`sort`** or **`sorted`**

```
student_scores = [('Robert', 8), ('Alice', 9),
('Tina', 10), ('James', 8)]
```

Goal: sort score from highest to lowest; within score, sort name alphabetically (= lowest to highest)

```
result = sorted(student_scores, key=itemgetter(0) )
result = sorted(result, key=itemgetter(1),
                        reverse=True)
```

Sorting: strings vs. numbers

- Sorting the powers of 5:

```
>>> sorted([125, 5, 3125, 625, 25])
[5, 25, 125, 625, 3125]
>>> sorted(["125", "5", "3125", "625", "25"])
['125', '25', '3125', '5', '625']
```

Top k with a dictionary

```
>>> uniquewords = { 'lol':45, 'omg':23, 'know':12 }
>>> uniquewords['omg']
```

```
def top10(doc):
  """Return a list of the top 10 most frequent words."""
  uniquewords = {}
  for word in doc:
    if uniquewords.has_key('omg'):
      uniquewords['omg'] = uniquewords['omg'] + 1
    else:
      uniquewords['omg'] = 1
  # now search for top 10 most frequent words
  bycount = [(pair[1], pair[0]) for pair in uniquewords.items()]
  bycount.sort()
  return bycount[0:10]
```

This "default" pattern is so common, there is a special method for it.

Top k with a dictionary

```
>>> uniquewords = { 'lol':45, 'omg':23, 'know':12 }
>>> uniquewords['omg']
```

```
def top10(doc):
  """Return a list of the top 10 most frequent words."""
  uniquewords = {}
  for word in doc:
    uniquewords['omg'] = uniquewords.set_default('omg', 0) + 1
  # now search for top 10 most frequent words
  bycount = [(pair[1], pair[0]) for pair in uniquewords.items()]
  bycount.sort()
  return bycount[0:10]
```

This "default" pattern is so common, there is a special method for it.