Sorting

Ruth Anderson
UW CSE 160
Winter 2017
sorted vs. sort

```python
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


print "names:", names

This does not work:

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

When sorting, how should we compare these names?

"Niels Bohr"
"Charles Darwin"
A sort key is a function that can be called on each list element to extract/create a value that will be used to make comparisons.

We can use this to sort on a value (e.g. “last_name”) other than the actual list element (e.g. “first_name last_name”).

We could use the following sort key to help us sort by last names:

```python
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 values returned by the sort key, and then sort it
2. Pass a key function to the sort or sorted function
1. Use a sort key to create a new list

Create a different list that contains the value returned by the sort key, sort it, then extract the relevant part:

```python
names = ["Isaac Newton", "Fig Newton", "Niels Bohr"]
# keyed_names is a list of [lastname, fullname] lists
keyed_names = []
for name in names:
    keyed_names.append([last_name(name), name])

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

1) Create the new list.

2) Sort the list new list. If there is a tie in last names, sort by next item in list: fullname

3) Extract the relevant part.

See in python tutor
Digression: Lexicographic Order

Aaron [1, 9, 9]
Andrew [2, 1]
Angie [3]

with [1]
withhold [1, 1]
withholding [1, 1, 1]

Able [1, 1]
Charlie [1, 1, 2]
baker [1, 2]
delta
2. Use a sort key as the key argument

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

```python
def last_name(str):
    return str.split(" ")[1]
	names = ["Isaac Newton", "Fig Newton", "Niels Bohr"]
print sorted(names, key = last_name)

print sorted(names, key = len)

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

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

If there is a tie in last names, preserves original order of values.

See in python tutor
itemgetter is a function that returns a function

```python
import operator
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")
```
from operator import itemgetter

student_score = ('Robert', 8)
itemgetter(0)(student_score) ⇒ “Robert”
itemgetter(1)(student_score) ⇒ 8

student_scores =
[(‘Robert’, 8), (‘Alice’, 9), (‘Tina’, 7)]

• Sort the list by **name**:  
  `sorted(student_scores, key=itemgetter(0))`

• Sort the list by **score**:  
  `sorted(student_scores, key=itemgetter(1))`
Two ways to Import `itemgetter`

```python
from operator import itemgetter

student_score = ('Robert', 8)
itemgetter(0)(student_score) ⇒ “Robert”
itemgetter(1)(student_score) ⇒ 8

Or

import operator

student_score = ('Robert', 8)
operator.itemgetter(0)(student_score) ⇒ “Robert”
operator.itemgetter(1)(student_score) ⇒ 8
```
Sorting based on two criteria

**Goal:** sort based on score; if there is a tie within score, sort by name

Two approaches:

- **Approach #1:** Use an itemgetter with two arguments
- **Approach #2:** Sort twice (most important sort **last**)

```python
code

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

Approach #1:
```
```python
coded
```sort`ed(student_scores, key=itemgetter(1,0))
```Approach #2:
```
```python
coded
```sort`ed_by_name = sorted(student_scores, key=itemgetter(0))
sorted_by_score = sorted(sorted_by_name, key=itemgetter(1))
```
Sort on most important criteria LAST

- Sorted by score (ascending), when there is a tie on score, sort using name

```python
from operator import itemgetter
student_scores = [('Robert', 8), ('Alice', 9), ('Tina', 10), ('James', 8)]

sorted_by_name = sorted(student_scores, key=itemgetter(0))
>>> sorted_by_name
[('Alice', 9), ('James', 8), ('Robert', 8), ('Tina', 10)]

sorted_by_score = sorted(sorted_by_name, key=itemgetter(1))
>>> sorted_by_score
[('James', 8), ('Robert', 8), ('Alice', 9), ('Tina', 10)]
```
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`

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

**Goal**: sort score from highest to lowest; if there is a tie within score, sort by name alphabetically (= lowest to highest)

```python
sorted_by_name = sorted(student_scores, key=itemgetter(0))
sorted_by_hi_score = sorted(sorted_by_name,
                           key=itemgetter(1), reverse=True)
```

Remember: Sort on most important criteria **LAST**
Sorting: strings vs. numbers

• Sorting the powers of 5:

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