Sorting

Ruth Anderson
UW CSE 160
Autumn 2020
sorted vs. sort

- **sorted(itr)** - is a function that takes an iterable as a parameter (e.g. sequence types: list, string, tuple) and **returns** a sorted version of that parameter

- **lst.sort()** - is a method that sorts the **list** that it is called on **in-place** (and returns **None**). **.sort()** can only be called on lists

```python
my_lst = [5, 3, 4, 2]
print(sorted(my_lst))  # Returns a new sorted list
                      # [2, 3, 4, 5]
print(my_lst)          # Does not modify original list
                      # [5, 3, 4, 2]
my_lst.sort()
print(my_lst)          # Modifies the list in place, returns None
                      # [2, 3, 4, 5]
```
sorted vs. sort example

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

See in python tutor

Returns a new sorted list (does not modify the original list)

Modifies the list in place, returns None
Customizing the sort order

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

```python

print("names:", names)

This does not work:

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

When sorting, how should we compare these names?

"Niels Bohr"
"Charles Darwin"
```
Aside: What does this do?

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

x = mystery("happy birthday")
print(x)
Sort key

• 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.

```python
fruits = ["watermelon", "fig", "apple"]
print(sorted(fruits))
print(sorted(fruits, key=len))
```

See in python tutor
• 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 function as a sort key to help us sort by last names:

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

```python
print('last_name("Isaac Newton"):', last_name("Isaac Newton"))
```
Use a sort key as the key argument

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

```
def last_name(str):
    return str.split(' ')[1]
	names = ["Isaac Newton", "Ada Lovelace", "Fig Newton", "Grace Hopper"]

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

Useful for creating a function that will return particular elements from a sequence (e.g. list, string, tuple):

```python
import operator

operator.itemgetter(2)([7, 3, 8]) ➞ 8
operator.itemgetter(0)([7, 3, 8]) ➞ 7
operator.itemgetter(1)([7, 3, 8]) ➞ 3
operator.itemgetter(0, 1)([7, 3, 8]) ➞ (7, 3)
operator.itemgetter(3)([7, 3, 8]) ➞

IndexError: list index out of range
```

Read the Documentation:
https://docs.python.org/3/library/operator.html
Tuples

• Immutable
  – cannot change elements
• Create using ()
• Use square brackets
  – to query and slice

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

Or

from operator import itemgetter
student_score = ('Robert', 8)
itemgetter(0)(student_score) ⇒ “Robert”
itemgetter(1)(student_score) ⇒ 8
```python
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))
```

Another way to import, allows you to call `itemgetter` directly.
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`)

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

Approach #1:
```
```python
calidated(thing)```
```
```python
`.sort` student_scores, `key=itemgetter(1,0))
```
```
Approach #2:
```
```python
`.sort` student_scores, `key=itemgetter(0))
```
```
`.sort` student_scores, `key=itemgetter(0))
```
```python
`.sort` 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**
## Digression: Lexicographic Order

<table>
<thead>
<tr>
<th>Word</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Aaron'</td>
<td>[1, 9, 9]</td>
</tr>
<tr>
<td>'Andrew'</td>
<td>[2, 1]</td>
</tr>
<tr>
<td>'Angie'</td>
<td>[3]</td>
</tr>
<tr>
<td>'with'</td>
<td>[1]</td>
</tr>
<tr>
<td>'withhold'</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>'withholding'</td>
<td>[1, 1, 1]</td>
</tr>
<tr>
<td>'Able'</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>'Charlie'</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>'baker'</td>
<td>[1, 1, 2]</td>
</tr>
<tr>
<td>'delta'</td>
<td>[1, 2]</td>
</tr>
</tbody>
</table>
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']
```
Aside: 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.