

# Sorting

BBM 101 - Introduction to Programming I

Hacettepe University  
Fall 2015

Fuat Akal, Aykut Erdem, Erkut Erdem, Vahid Garousi

Slides based on material prepared by Ruth Anderson, Michael Ernst and Bill Howe in the course CSE 140  
University of Washington

1

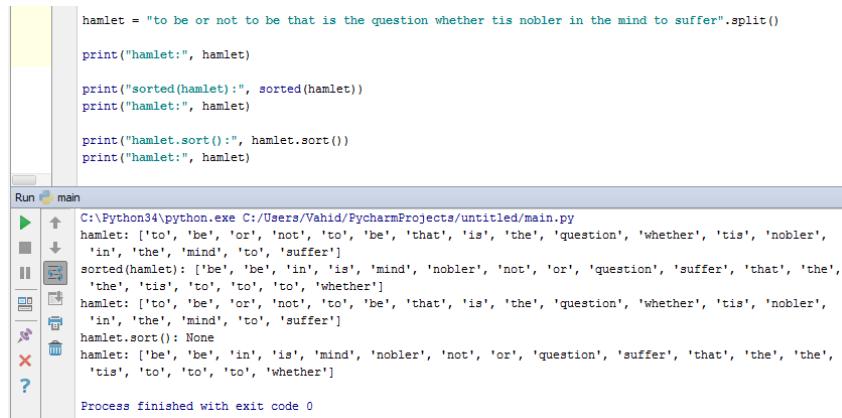
## Sorting

```
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

2

# 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)
```

Run main

```
C:\Python34\python.exe C:/Users/Vahid/PycharmProjects/untitled/main.py  
hamlet: ['to', 'be', 'or', 'not', 'to', 'be', 'that', 'is', 'the', 'question', 'whether', 'tis', 'nobler',  
'in', 'the', 'mind', 'to', 'suffer']  
sorted(hamlet): ['be', 'be', 'in', 'is', 'mind', 'nobler', 'not', 'or', 'question', 'suffer', 'that', 'the',  
'the', 'tis', 'to', 'to', 'to', 'whether']  
hamlet: ['to', 'be', 'or', 'not', 'to', 'be', 'that', 'is', 'the', 'question', 'whether', 'tis', 'nobler',  
'in', 'the', 'mind', 'to', 'suffer']  
hamlet.sort(): None  
hamlet: ['be', 'be', 'in', 'is', 'mind', 'nobler', 'not', 'or', 'question', 'suffer', 'that', 'the', 'the',  
'tis', 'to', 'to', 'to', 'whether']  
  
Process finished with exit code 0
```

3

## 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"
```

4

## 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

5

## 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:

```
names = ["Isaac Newton", "Fred 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])
```

1) Create the new list.

Take a look at the list you created, it can now be sorted:

```
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)
```

2) Sort the list new list.

```
sorted_names = []
for keyed_name in sorted_keyed_names:
    sorted_names.append(keyed_name[1])
```

3) Extract the relevant part.

```
print "sorted_names:", sorted_names
```

6

## 2. 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", "Fred Newton", "Niels Bohr"]
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)
```

7

## itemgetter is a Function that Returns a Function

```
import operator
All: ('m', 'i', 'k', 'e')

print(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")
```

8

## Using itemgetter

```
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) )
```

9

## Two Ways to Import itemgetter

```
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
```

10

## Sorting Based on Two Criteria

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)]
```

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

Approach #1:

```
sorted(student_scores, key=itemgetter(1,0) )
```

Approach #2:

```
sorted_by_name = sorted(student_scores, key=itemgetter(0) )
sorted_by_score = sorted(sorted_by_name, key=itemgetter(1) )
```

11

## Sort on Most Important Criteria LAST

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

```
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)]
```

12

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

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

13

## 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']
```

14

## Different sorting algorithms

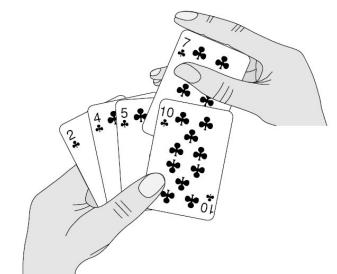
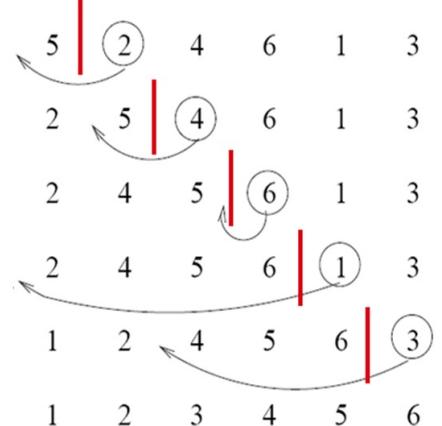
- 3.1 Simple sorts
  - 3.1.1 Insertion sort
  - 3.1.2 Selection sort
- 3.2 Efficient sorts
  - 3.2.1 Merge sort
  - 3.2.2 Heapsort
  - 3.2.3 Quicksort
- 3.3 Bubble sort and variants
  - 3.3.1 Bubble sort
  - 3.3.2 Shell sort
  - 3.3.3 Comb sort
- 3.4 Distribution sort
  - 3.4.1 Counting sort
  - 3.4.2 Bucket sort
  - 3.4.3 Radix sort

The screenshot shows the Wikipedia article on "Sorting algorithm". The page title is "Sorting algorithm" and it is described as an "Algorithm that puts elements of a list in a certain order". It defines a sorting algorithm as an algorithm that puts elements of a list in a certain order, which requires input data to be in sorted lists. It also notes that sorting is often useful for comparison sorting algorithms.

15

## Insertion sort

- Idea:



Done !

16

## Insertion sort

A screenshot of an online Python editor from [interactivepython.org](http://interactivepython.org). The title bar says "Problem Solving with Algorithms and Data Structures". The code area contains the following Python script:

```
1 def insertionSort(alist):
2     for index in range(1,len(alist)):
3         currentvalue = alist[index]
4         position = index
5
6         while position>0 and alist[position-1]>current
7             alist[position]=alist[position-1]
8             position = position-1
9
10        alist[position]=currentvalue
11
12    alist = [54,26,93,17,77,31,44,55,20]
13    insertionSort(alist)
14    print(alist)
15
16
```

The output window shows the initial list [17, 20, 26, 31, 44, 54, 55, 77, 93] and the sorted list [17, 20, 26, 31, 44, 54, 55, 77, 93].

17

## Bubble Sort

- It repeatedly steps through the list to be sorted,
- compares each pair of adjacent items and swaps them if they are in the wrong order.
- The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.
- The algorithm, which is a comparison sort, is named for the way smaller elements "bubble" to the top of the list.



18

## Bubble Sort

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)
```

19

[www.sorting-algorithms.com](http://www.sorting-algorithms.com)

### Sorting Algorithm Animations

Problem Size: [20](#) · [30](#) · [40](#) · [50](#)   Magnification: [1x](#) · [2x](#) · [3x](#)

Algorithm: [Insertion](#) · [Selection](#) · [Bubble](#) · [Shell](#) · [Merge](#) · [Heap](#) · [Quick](#) · [Quick3](#)

Initial Condition: [Random](#) · [Nearly Sorted](#) · [Reversed](#) · [Few Unique](#)

	<a href="#">Insertion</a>	<a href="#">Selection</a>	<a href="#">Bubble</a>	<a href="#">Shell</a>	<a href="#">Merge</a>	<a href="#">Heap</a>	<a href="#">Quick</a>	<a href="#">Quick3</a>
<a href="#">Random</a>								
<a href="#">Nearly Sorted</a>								
<a href="#">Reversed</a>								
<a href="#">Few Unique</a>								

20