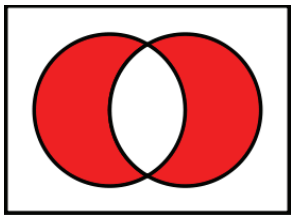
Compsci 101
Simple Sorting, Transform, Sets



Susan Rodger

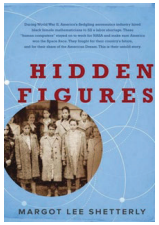
February 28, 2023

M is for …

- **Machine Learning**
  - Math, Stats, CompSci: learning at scale
- **Microsoft, Mozilla, Macintosh**
  - Software that changed the world?
- **Memory**
  - Storage space in the computer
  - From 64 Kilobytes to 16 Gigobytes!
- **Mouse, Mouse pad**
  - Easier to navigate

Margot Shetterly

- **Writer, Author of Hidden Figures**
- **Black Women NASA Scientists**
- **Gave a talk at Duke in 2016**



Katherine Johnson **Mary Jackson** **Dorothy Vaughn** **Christine Darden**

Announcements

- **Assignment 3 due Thursday, March 2**
  - Sakai quiz due today
- **Assignment 4 out Thursday!**
- **APT-4 is out and due Thursday March 9**
  - Can use some as practice for exam
- **Lab 7 Friday, there is a prelab available Thursday!**

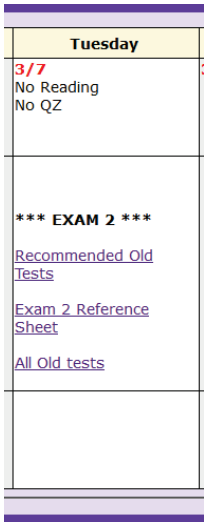
- **Do not discuss APT Quiz 1 until grades posted!**
  - A few have not take it yet due to travel or illness

PFTD

- **Simple Sorting**
- **Solving an APT**
- **Assignment 4**
- **Sets**

Exam 2 – in person – Tues, March 7

- **Exam is in class on paper – 10:15am**
  - Need pen or pencil
- **See materials under 3/7 date**
  - Exam 2 Reference sheet - part of exam
- **Covers**
  - topics /reading through Thursday
  - APTs through APT4
    - APT4 – write code on paper, then type in
  - Labs through Lab 7
    - Lab 7 - Parts 1-3
  - Assignments through Assignment 3



Exam 2 topics include …

- **List, tuples, list comprehensions**
- **Loops – for loop, while loop, indexing with a loop**
- **Reading from a file**
  - Converting data into a list of things
- **Parallel lists**
- **Sets – solving problems**
- **Dictionaries – only reading them and understanding output, no problem solving**
- **No turtles on the exam!**

Exam 2

- **Exam 2 is your own work!**
- **No looking at others exam or talking to others**
- **You cannot use any notes, books, computing devices, calculators, or any extra paper**
- **Bring only a pen or pencil**
- **The exam has extra white space and has the Exam 2 reference sheet as part of the exam.**

- **Do not discuss any problems on the exam with others until it is handed back**

Exam 2 – How to Study

- **Practice writing code on paper!**
- **Rewrite an APT**
- **Try to write code from lecture from scratch**
- **Try to write code from lab from scratch**
- **Practice from old exams**
- **Put up old Sakai quizzes, but better to practice writing code**
- **Look at Exam 2 reference sheet when writing code!**

Let's sort lists with sorted() function

- **Want list elements in sorted order**
  - Example: have list [17 , 7, 13, 3]
  - Want list [3, 7, 13, 17], in order
- **Built-in function: sorted(*sequence*)**
  - **Returns new list** of sequence in sorted order
  - Sequence could be list, tuple, string

Example

```
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
b.remove('et')
b.append(6)
b.insert(1,5)
c = sorted(b)
```

lst is [6, 2, 9, 4, 3]

Example

```
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
b.remove('et')
b.append(6)
b.insert(1,5)
c = sorted(b)
```

lst is [6, 2, 9, 4, 3]
lsta is [2, 3, 4, 6, 9]
b is ['ko', 'et', 'at', 'if']
c is ['at', 'et', 'if', 'ko']
b is ['ko', 'at', 'if']
b is ['ko', 'at', 'if', 6]
b is ['ko', 5, 'at', 'if', 6]
ERROR!!!!!!!! Cannot sort numbers and strings

This is a bullt-in function.
sorted "returns" a new list!

```
lst = [6, 2, 9, 4, 3]          lst  is [6, 2, 9, 4, 3]}}}}}
lsta = sorted(lst)             lsta is  [2, 3, 4, 6, 9]
b = ['ko', 'et', 'at', 'if']   b is   ['ko', 'et', 'at', 'if']
c = sorted(b)                  c is  ['at', 'et', 'if', 'ko']
b.remove('et')                 b is  ['ko', 'at', 'if']
b.append(6)                    b is  ['ko', 'at', 'if', 6]
b.insert(1,5)                  b is  ['ko', 5, 'at', 'if', 6]
c = sorted(b)                  ort
```

These three are list methods
(list dot methodName).
They mutate the list,
"change" the list.
There is NO return value

Example

```
lst = (7, 4, 1, 8, 3, 2)        lst  is    (7, 4, 1, 8, 3, 2)
lsta = sorted(lst)
b = ('ko', 'et', 'at', 'if')
c = sorted(b)
d = "word"
e = sorted(d)
f = 'go far'
g = sorted(f)
f = 'go far'
h = sorted(f.split())
```


Example

```
lst = (7, 4, 1, 8, 3, 2)        lst  is    (7, 4, 1, 8, 3, 2)
lsta = sorted(lst)              lsta is  [1, 2, 3, 4, 7, 8]
b = ('ko', 'et', 'at', 'if')    b is   ('ko', 'et', 'at', 'if')
c = sorted(b)                   c is  ['at', 'et', 'if', 'ko']
d = "word"                      d is  'word'
e = sorted(d)                   e is  ['d', 'o', 'r', 'w']
f = 'go far'                    f is 'go far'
g = sorted(f)                   g is [' ', 'a', 'f', 'g', 'o', 'r']
f = 'go far'                    f is 'go far'
h = sorted(f.split())           h is  ['far', 'go']
```


Now, sort lists with .sort() list method

- **Want to "change" list elements to sorted order**
  - lst is [17 , 7, 13, 3]
  - lst.sort()
  - Now **same** list lst is [3, 7, 13, 17], in order
- **List method: *list*.sort()**
  - List is **modified, now in sorted order**
  - There is NO return value
  - Only works with lists, can't modify strings, tuples

Compare sorted() with .sort()

```
lsta = [6, 2, 9, 4, 3]
lstb = sorted(lsta)

lsta.sort()
a = [7, 2, 9, 1]
b = a.sort()

c = (5, 6, 2, 1)
c.sort()
d = "word"
d.sort()
```

lsta is [6, 2, 9, 4, 3]

Compare sorted() with .sort()

```
lsta = [6, 2, 9, 4, 3]
lstb = sorted(lsta)

lsta.sort()
a = [7, 2, 9, 1]
b = a.sort()

c = (5, 6, 2, 1)
c.sort()  X
d = "word"
d.sort()  X
```

lsta is [6, 2, 9, 4, 3]
lstb is [2, 3, 4, 6, 9]
lsta is still [6, 2, 9, 4, 3]
lsta is [2, 3, 4, 6, 9]
a is [7, 2, 9, 1]
a is [1, 2, 7, 9]
b is None
c is (5, 6, 2, 1)
ERROR!!!! Can't change!
d is 'word'
ERROR!!!! Can't modify!

Compare sorted() with .sort()

```
lsta = [6, 2, 9, 4, 3]
lstb = sorted(lsta)

lsta.sort()
a = [7, 2, 9, 1]
b = a.sort()  X

a.sort()
```

sorted() does have a return value, save it in a variable!

Don't use .sort this way. It does not have a return value!

Use it this way for list a!

WOTO-1 Sorting

http://bit.ly/10123s-0228-1

APT - TxMsg

Problem Statement

Strange abbreviations are often used to write text messages on uncomfortable mobile devices. One particular strategy for encoding texts composed of alphabetic characters and spaces is the following:

- Spaces are maintained, and each word is encoded individually. A word is a consecutive string of alphabetic characters.
- If the word is composed only of vowels, it is written exactly as in the original message.
- If the word has at least one consonant, write only the consonants that do not have another consonant immediately before them. Do not write any vowels.
- The letters considered vowels in these rules are 'a', 'e', 'i', 'o' and 'u'. All other letters are considered consonants.

For instance, "ps i love u" would be abbreviated as "p i lv u" while "please please me" would be abbreviated as "ps ps m". You will be given the original message in the string parameter `original`. Return a string with the message abbreviated using the described strategy.

Specification

```
filename: TxMsg.py

def getMessage(original):
    """
    return String that is 'textized' version
    of String parameter original
    """

    # you write code here
```

Examples

Examples

1. `"text message"`

   `Returns "tx msg"`

5. `"aeiou bcdfghjklmnpqrstvwxyz"`

   `Returns: "aeiou b"`

WOTO-2 – TxMsg

http://bit.ly/101s23-0228-2

Debugging APTs: Going green

```
def getMessage(original):
    ret = [ ]
    for word in original.split():
        ret.append(transform(word))
    return " ".join(ret)
```

- **TxMsg APT: from ideas to code to green**
  - What are the main parts of solving this problem?
  - Transform words in original string
  - Abstract that away at first
  - Finding words in original string - .split()
  - Use another function **transform** to focus on one word
  - Then put list of words translated back together

Write helper function *transform*

- **How?**
- **Use seven steps**
- **Work an example by hand**

Transform word - Step 1: work small example by hand

- **Word is “please”**
- **Letter is ‘p’, YES**
- **answer so far is “p”**
- **Letter is ‘l’, NO**
- **Letter is ‘e’, NO**
- **Letter is ‘a’, NO**
- **Letter is ‘s’, YES**
- **answer so far is “ps”**
- **Letter is ‘e’, NO**

Step 2: Describe what you did

- **Word is “please”, create an empty answer**
- **Letter is ‘p’, consonant, no letter before, YES**
- **Add ‘p’ to answer**
- **Letter is ‘l’, consonant, letter before “p”, NO**
- **Letter is ‘e’, vowel, letter before ‘l’, NO**
- **Letter is ‘a’, vowel, letter before ‘e’, NO**
- **Letter is ‘s’, consonant, letter before ‘a’, YES**
- **Add ‘s’ to answer**
- **Letter is ‘e’, vowel, letter before ‘s’, NO**
- **Answer is “ps”**

Step 3: Find Pattern and generalize

Need to initialize letter before, pick “a”

answer is empty

for each letter in word

If it is a **consonant**, and the **letter before** is a vowel, then add the letter to the answer

This letter is now the letter before

return answer

Step 4 – Work another example

- **Word is message**
- **Letter is ‘m’, before is ‘a’, add ‘m’ to answer**
- **Letter is ‘e’, before is ‘m’, NO**
- **Letter is ‘s’, before is ‘e’, add ‘s’ to answer**
- **Letter is ‘s’, before is ‘s’, NO**
- **Letter is ‘a’, before is ‘s’, NO**
- **Letter is ‘g’, before is ‘a’, add ‘g’ to answer**
- **Letter is ‘e’, before is ‘g’, NO**
- **Answer is “msg” WORKS!!**

Use vowel not part of word

Step 5: Translate to Code

```
# Letter before is “a”      # start with a vowel

# answer is empty

# for each letter in word
```


Step 5: Translate to Code

```
# Letter before is “a”      # start with a vowel
before = ‘a’
# answer is empty
answer = ""          # or this could be an empty list
# for each letter in word
for ch in word:
```


Step 5: Translate to Code (code)

```
#If it is a consonant, and the letter before is a
#vowel, then add the letter to the answer


#This letter is now the letter before

# return answer
```

Step 5: Translate to Code (code)

```
#If it is a consonant, and the letter before is a
#vowel, then add the letter to the answer
if !(isVowel(ch)) and isVowel(before):
    answer += ch
#This letter is now the letter before
before = ch
# return answer
return answer
```


Will our program work for?

- **STRING GET SHOULD GET**
- **green**
- **apple**
- **a**
- **aeiuo**
- **grrr**

Will our program work for?

- **STRING GET SHOULD GET**
- **green** 'gn' YES
- **apple** 'p' YES
- **a** '' 'a'
- **aeiuo** '' 'aeiou'
- **grrr** 'g' YES

Doesn't work when all vowels

Handle special cases first?
Write another helper function?

STOP HERE…

- **You finish**
- **May need to debug**

Why use helper function 'transform'?

- **Structure of code is easier to reason about**
  - Harder to develop this way at the beginning
  - Similar to accumulate loop, build on what we know
- **We can debug pieces independently**
  - What if transform returns "" for every string?
  - Can we test transform independently of `getMessage`?

Assignment 4: Guess Word

- **We give you most of the functions to implement**
  - Partially for testing, partially for guiding you
- **But still more open ended than prior assignments**
- **If the doc does not tell you what to do:**
  - Your chance to decide on your own!
    - Okay to get it wrong on the first try
  - Discuss with TAs and friends, brainstorm!
- **Demo!**

Python Sets

- **Set – unordered collection of distinct items**
  - Unordered – can look at them one at a time, but cannot count on any order
  - Distinct - one copy of each

```
x = [5, 3, 4, 3, 5, 1]
y = set(x)

y.add(6)
y.add(4)
```

x is [5, 3, 4, 3, 5, 1]

Python Sets

- **Set – unordered collection of distinct items**
  - Unordered – can look at them one at a time, but cannot count on any order
  - Distinct - one copy of each

```
x = [5, 3, 4, 3, 5, 1]
y = set(x)

y.add(6)
y.add(4)
```

x is [5, 3, 4, 3, 5, 1]

y is {3, 1, 4, 5}

Don't know order of elements!

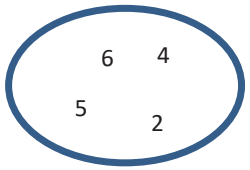
y is {3, 6, 1, 4, 5}

no change since 4 is a duplicate!

y is {3, 6, 1, 4, 5}

List vs Set

- **List**
  - Ordered, 3rd item, can have duplicates
  - Example: `x =[4, 6, 2, 4, 5, 2, 4]`
- **Set**
  - No duplicates, no ordering
  - Example: `y = set(x)`
- **Both**
  - Add, remove elements
  - Iterate over all elements



Python Sets

- **Can convert list to set, set to list**
  - Great to get rid of duplicates in a list

a = [2, 3, 6, 3, 2, 7]
b = set(a)

c = list(b)

a is [2, 3, 6, 3, 2, 7]

Python Sets

- **Can convert list to set, set to list**
  - Great to get rid of duplicates in a list

a = [2, 3, 6, 3, 2, 7]
b = set(a)

c = list(b)

a is [2, 3, 6, 3, 2, 7]
b is {3, 2, 7, 6}

c is [6, 7, 2, 3]

Python Sets

- **Can convert list to set, set to list**
  - Great to get rid of duplicates in a list

a = [2, 3, 6, 3, 2, 7]
b = set(a)

c = list(b)

a is [2, 3, 6, 3, 2, 7]
b is {3, 2, 7, 6}

Don't know order of elements in b

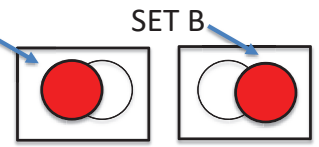
c is [6, 7, 2, 3]

Elements are ordered in c, but we don't know what order they will be in

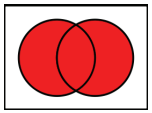
Python Sets

- **Operations on sets:**
  - Modify:
    - add a.add(7)
    - clear a.clear()
    - remove a.remove(5)
  - Create a new set: a = set([])
  - difference(-), intersection(&), union (|), symmetric_difference(^)
  - Boolean: issubset <=, issuperset >=

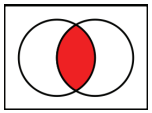
Python Set Operators



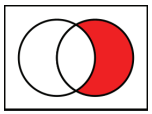
- **Using sets and set operations often useful**
- **A | B, set union**
  - Everything



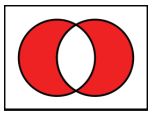
- **A & B, set intersection**
  - *Only* in both



- **B – A, set difference**
  - In B *and* not A



- **A ^ B, symmetric diff**
  - Only in A *or* only in B



List and Set, Similarities/Differences

	Function for List	Function for Set
Adding element	`x.append(elt)`	`x.add(elt)`
Size of collection	`len(x)`	`len(x)`
Combine collections	`x + y`	`x | y`
Iterate over	`for elt in x:`	`for elt in x:`
Element membership	`elt in x`	`elt in x`
Index of an element	`x.index(elt)`	`CANNOT DO THIS`

- Lists are ordered and indexed, e.g., has a first or last
- Sets are **not** ordered, very fast, e.g., `if elt in x`

List and Set, Similarities/Differences

	Function for List	Function for Set
Adding element	`x.append(elt)`	`x.add(elt)`
Size of collection	`len(x)`	`len(x)`
Combine collections	`x + y`	`x | y`
Iterate over	`for elt in x:`	`for elt in x:`
Element membership	`elt in x`	`elt in x`
Index of an element	`x.index(elt)`	`CANNOT DO THIS`

- Lists are ordered and indexed, e.g., has a first or last
- Sets are **not** ordered, very fast, e.g., `if elt in x`

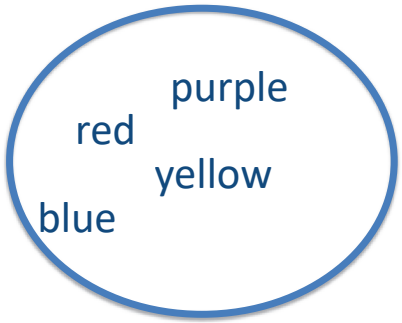
Creating and changing a set

```
colorList = ['red', 'blue', 'red', 'red', 'green']
colorSet = set(colorList)
smallList = list(colorSet)
colorSet.clear()
colorSet.add("yellow")
colorSet.add("red")
colorSet.add("blue")
colorSet.add("yellow")
colorSet.add("purple")
colorSet.remove("yellow")
```

smallList is

Creating and changing a set

```
colorList = ['red', 'blue', 'red', 'red', 'green']
colorSet = set(colorList)
smallList = list(colorSet)
colorSet.clear()
colorSet.add("yellow")
colorSet.add("red")
colorSet.add("blue")
colorSet.add("yellow")
colorSet.add("purple")
colorSet.remove("yellow")
```



smallList is [‘red’, ‘green’, ‘blue’] order?
colorSet is

Creating and changing a set

```
colorList = ['red', 'blue', 'red', 'red', 'green']
colorSet = set(colorList)
smallList = list(colorSet)
colorSet.clear()
colorSet.add("yellow")
colorSet.add("red")
colorSet.add("blue")
colorSet.add("yellow")
colorSet.add("purple")
colorSet.remove("yellow")
```

smallList is [‘red’, ‘green’, ‘blue’] order?
colorSet is set([“purple”, “red”, “blue”]) order?

Set Operations – Union and Intersection

```
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])

print(dukeColors | UScolors)
print(dukeColors & UScolors)
```

Set Operations – Union and Intersection

```
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])

print(dukeColors | UScolors)
print(dukeColors & UScolors)
```

set(['blue', 'black', 'white', 'red'])
set(['blue', 'white'])

Set Operations - Difference

```
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])

print(dukeColors - UScolors)
print(UScolors - dukeColors)
```


Set Operations - Difference

```
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])

print(dukeColors - UScolors)
print(UScolors - dukeColors)
```

set(['black'])
set(['red'])

Set Operations – Symmetric Difference

```
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])

print(dukeColors ^ UScolors)
print(UScolors ^ dukeColors)
```

Set Operations – Symmetric Difference

```
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])

print(dukeColors ^ UScolors)
print(UScolors ^ dukeColors)
```

set(['black', 'red'])
set(['black', 'red'])

Let's sort lists with sorted() function

- **Built-in function: sorted(*sequence*)**
  - **Returns new list** of sequence in sorted order
  - Sequence could be list, tuple, string
  - **Sequence could be set!**

a = set([3, 5, 2, 1, 7, 2, 5])
b = sorted(a)

Let's sort lists with sorted() function

- **Built-in function: sorted(*sequence*)**
  - **Returns new list** of sequence in sorted order
  - Sequence could be list, tuple, string
  - **Sequence could be set!**

a = set([3, 5, 2, 1, 7, 2, 5])
b = sorted(a)

a is { 3, 5, 2, 1, 7 }
b is [1, 2, 3, 5, 7]

WOTO-3 Sets
http://bit.ly/101s23-0228-3