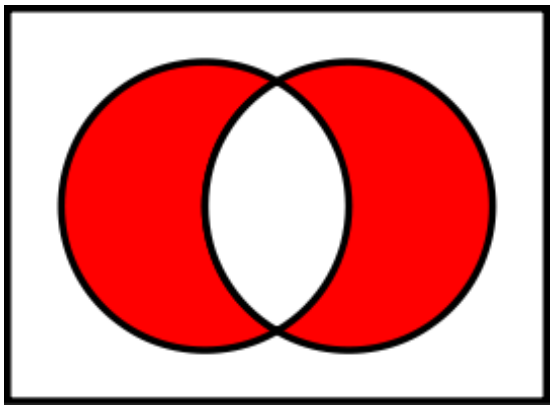
Compsci 101
Simple Sorting, Transform, Sets



Susan Rodger

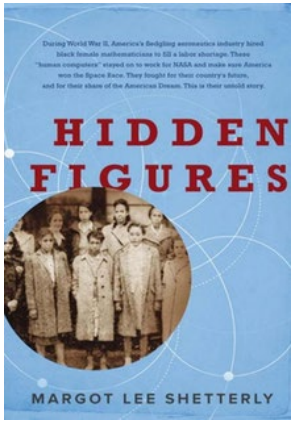
Oct 18, 2022

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

- **APT-4 is out and due Thursday October 27**
- **Assignment 3 due Thursday, Oct 20**
  - Sakai quiz due today
- **Lab 6 Friday, there is a prelab available now!**

- **Do not discuss APT Quiz 1 until grades posted!**

- **All Assign, APT, APT quiz 2 dates now on calendar!**

- **Last chance for regrades for Exam 1 is tonight 11pm**

Prof Rodger no office hours today

- **I will be at the majors fair in Penn Pavillion**
  - from 1-4pm if you want to ask questions about CompSci major

- **Thursday office hours will be online only as I am traveling after class**

PFTD

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

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 = (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())

lst is (7, 4, 1, 8, 3, 2)

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]

WOTO-1 Sorting
http://bit.ly/101f22-1018-1

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.

Specification

```
filename: TxMsg.py

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

    # you write code here
```

- 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.

Examples

Examples

1. `"text message"`

   Returns `"tx msg"`

5. `"aeiou bcdfghjklmnpqrstvwxyz"`

   Returns: `"aeiou b"`

WOTO-2 – TxMsg
http://bit.ly/101f22-1018-2

Write helper function *transform*

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

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`?

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)

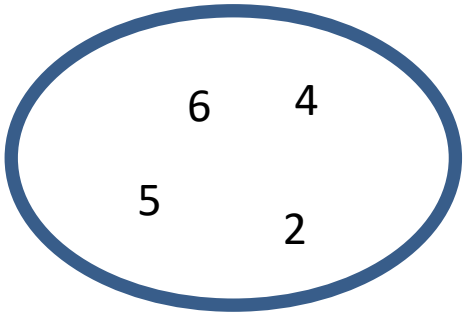
y.add(6)

y.add(4)

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

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

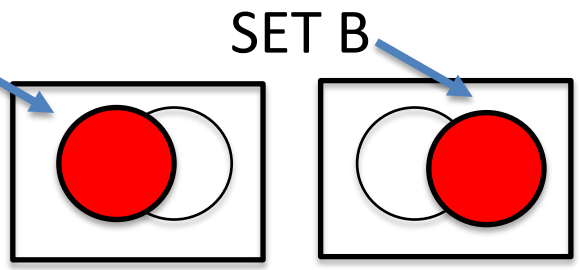
```
a = [2, 3, 6, 3, 2, 7]
b = set(a)

c = list(b)
```

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

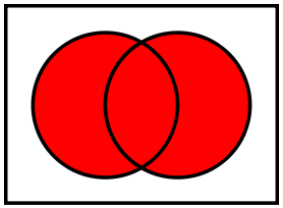
Python Sets

- **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 >=

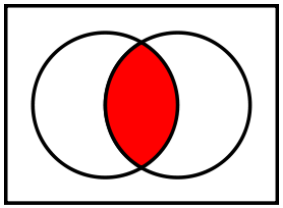
Python Set Operators



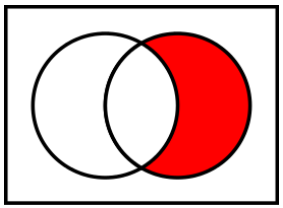
- **Using sets and set operations often useful**
- **A | B, set union**
  - Everything



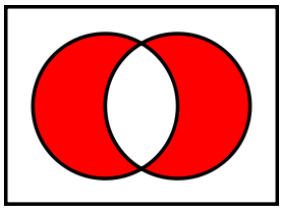
- **A & B, set intersection**
  - *Only* in both



- **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`

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

Set Operations – Union and Intersection

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

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

Set Operations - 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)
```

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)

WOTO-3 Sets
http://bit.ly/101f22-1018-3