Compsci 101
Simple Sorting, Transform, Sets

Margot Shetterly
• Writer, Author of Hidden Figures
• Black Women NASA Scientists
• Gave a talk at Duke in 2016

Katherine Mary Dorothy Christine
Johnson Jackson Vaughn Darden

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 Gigabytes!
• Mouse, Mouse pad
  • Easier to navigate

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
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 – 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

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```

Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)
b = ['ko', 'et', 'at', 'if']
c = sorted(b)
```
Example

```python
lst = [6, 2, 9, 4, 3]
lsta = sorted(lst)  # lst is [2, 3, 4, 6, 9]
b = ['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)  # Error!!!!!! Cannot sort numbers and strings
```

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

Now, sort lists with .sort() list method

- Want to "change" list elements to sorted order
  - Ist is [17, 7, 13, 3]
  - Ist.sort()
  - Now same list Ist 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()

```python
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()
```

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

Use it this way for list a!

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

WOTO-1 Sorting

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.

Examples

1. "text message"
   Returns "tx msg"

5. "aeiou bcdfghjklmnpqrstvxyz"
   Returns: "aeiou b"

WOTO-2 – TxMsg

Debugging APTs: Going green

```python
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
**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!!

Step 5: Translate to Code

```python
# 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:
    #If it is a consonant, and the letter before is a vowel, then add the letter to the answer
    if not isVowel(ch) and isVowel(before):
        answer += ch
    #This letter is now the letter before
    before = ch

# return answer
```

Step 5: Translate to Code (code)

```python
# 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:
    #If it is a consonant, and the letter before is a vowel, then add the letter to the answer
    if not isVowel(ch) and isVowel(before):
        answer += ch
    #This letter is now the letter before
    before = ch

# return answer
```
**Step 5: Translate to Code (code)**

```python
# 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?**

<table>
<thead>
<tr>
<th>STRING</th>
<th>GET</th>
<th>SHOULD GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>'gn'</td>
<td>YES</td>
</tr>
<tr>
<td>apple</td>
<td>'p'</td>
<td>YES</td>
</tr>
<tr>
<td>a</td>
<td>''</td>
<td>Doesn't work when all vowels</td>
</tr>
<tr>
<td>aeiuo</td>
<td>'a'</td>
<td>YES</td>
</tr>
<tr>
<td>grrr</td>
<td>'g'</td>
<td>YES</td>
</tr>
</tbody>
</table>

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

```python
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!

no change since 4 is a duplicate!
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: \( y = \text{set}(x) \)

• 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

\[
\begin{align*}
  a &= [2, 3, 6, 3, 2, 7] \\
  b &= \text{set}(a) \\
  c &= \text{list}(b)
\end{align*}
\]
Python Sets

- Operations on sets:
  - Modify:
    - add  \texttt{a.add(7)}
    - clear  \texttt{a.clear()}
    - remove  \texttt{a.remove(5)}
  - Create a new set:  \texttt{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

<table>
<thead>
<tr>
<th>Function for List</th>
<th>Function for Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding element</td>
<td>\texttt{x.append(elt)}</td>
</tr>
<tr>
<td>Size of collection</td>
<td>\texttt{len(x)}</td>
</tr>
<tr>
<td>Combine collections</td>
<td>\texttt{x + y}</td>
</tr>
<tr>
<td>Iterate over</td>
<td>\texttt{for elt in x:}</td>
</tr>
<tr>
<td>Element membership</td>
<td>\texttt{elt in x}</td>
</tr>
<tr>
<td>Index of an element</td>
<td>\texttt{x.index(elt)}</td>
</tr>
</tbody>
</table>

- Lists are ordered and indexed, e.g., has a first or last
- Sets are not ordered, very fast, e.g., \texttt{if elt in x}
Creating and changing a set

```python
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']
```

colorSet is

```
set(['red', 'blue', 'yellow', 'purple'])
```

Creating and changing a set

```python
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'])
```

Set Operations – Union and Intersection

```python
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])
print(dukeColors | UScolors)
print(dukeColors & UScolors)
```

smallList is ['red', 'green', 'blue'] order?

colorSet is set(['purple', 'red', 'blue']) order?
Set Operations – Union and Intersection

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

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

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

Set Operations - Difference

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

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

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

Set Operations – Symmetric Difference

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

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

```python
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(['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)

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

WOTO-3 Sets