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
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
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

```python
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)
```

<table>
<thead>
<tr>
<th>lst</th>
<th>lsta</th>
<th>b</th>
<th>c</th>
<th>10/18/22 Compsci 101, Fall 2022</th>
</tr>
</thead>
<tbody>
<tr>
<td>[6, 2, 9, 4, 3]</td>
<td>[2, 3, 4, 6, 9]</td>
<td>['ko', 'at', 'if']</td>
<td>['at', 'et', 'if', 'ko']</td>
<td></td>
</tr>
<tr>
<td>lsta</td>
<td>sorted(lst)</td>
<td>b</td>
<td>c</td>
<td>10/18/22 Compsci 101, Fall 2022</td>
</tr>
<tr>
<td>[2, 3, 4, 6, 9]</td>
<td>[2, 3, 4, 6, 9]</td>
<td>['ko', 'at', 'if']</td>
<td>['at', 'et', 'if', 'ko']</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>['ko', 'et', 'at', 'if']</td>
<td>c</td>
<td>sorted(b)</td>
<td></td>
</tr>
<tr>
<td>['ko', 'at', 'if']</td>
<td>['ko', 'at', 'if', 6]</td>
<td>['at', 'et', 'if', 'ko']</td>
<td>['at', 'et', 'if', 'ko']</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>sorted(b)</td>
<td>b</td>
<td>insert(1, 5)</td>
<td></td>
</tr>
<tr>
<td>10/18/22 Compsci 101, Fall 2022</td>
<td>10/18/22 Compsci 101, Fall 2022</td>
<td>10/18/22 Compsci 101, Fall 2022</td>
<td>10/18/22 Compsci 101, Fall 2022</td>
<td></td>
</tr>
</tbody>
</table>

ERROR!!!!!!!!! Cannot sort numbers and strings
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
```
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)

Example

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]

These three are list methods (list dot methodname). They mutate the list, “change” the list.
There is NO return value

This is a built-in function. sorted “returns” a new list!
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)
lsta is [1, 2, 3, 4, 7, 8]
b is ('ko', 'et', 'at', 'if')
c is ['at', 'et', 'if', 'ko']
d is 'word'
e is ['d', 'o', 'r', 'w']
f is 'go far'
g is [' ', 'a', ' f', 'g', 'o', 'r']
f is 'go far'
h is ['far', 'go']
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)
h = sorted(f.split())

lst is (7, 4, 1, 8, 3, 2)
lsta is [1, 2, 3, 4, 7, 8]
b is (‘ko’, ‘et’, ‘at’, ‘if’)
c is [‘at’, ‘et’, ‘if’, ‘ko’]
d is ‘word’
e is [‘d’, ‘o’, ‘r’, ‘w’]
f is ‘go far’
g is [‘’, ‘a’, ‘’, ‘f’, ‘g’, ‘o’, ‘r’]
f is ‘go far’
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()`

```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()
```

`lsta` is `[6, 2, 9, 4, 3]`  
`lstb` is `[2, 3, 4, 6, 9]`  
`lsta` is still `[6, 2, 9, 4, 3]`

`a` is `[1, 2, 7, 9]`
`b` is `None`
Compare `sorted()` with `.sort()`

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

lsta.sort()

da = [7, 2, 9, 1]
b = a.sort()
c = (5, 6, 2, 1)
c.sort()  # ERROR!!!! Can’t change!
d = "word"
d.sort()  # ERROR!!!! Can’t modify!
```
Compare `sorted()` with `.sort()`

```python
lsta = [6, 2, 9, 4, 3]
lstb = sorted(lsta)
lsta.sort()
a = [7, 2, 9, 1]
b = a.sort()  # X
a.sort()
```

- `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`

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

Use it this way!

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

```
filename: TxMsg.py

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

    # you write code here
```
Examples

1. "text message"
   Returns "tx msg"

5. "aeiou bcdfghjklmnpqrstvwxyz"
   Returns: "aeiou b"
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!!
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)

```python
# 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
return answer
```
Will our program work for?

- STRING
- GET
- SHOULD GET
- green
- apple
- a
- aeiuo
- grrr
Will our program work for?

- STRING
- green
- apple
- a
- aeiuo
- grrr

<table>
<thead>
<tr>
<th>GET</th>
<th>SHOULD GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>'gn'</td>
<td>YES</td>
</tr>
<tr>
<td>'p'</td>
<td>YES</td>
</tr>
<tr>
<td>'a'</td>
<td>YES</td>
</tr>
<tr>
<td>'aeiou'</td>
<td></td>
</tr>
<tr>
<td>'g'</td>
<td>YES</td>
</tr>
</tbody>
</table>

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?
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]
```

10/18/22
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)
```

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

Don’t know order of elements!

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

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

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: \( y = \text{set}(x) \)

- **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

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

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

```python
a = [2, 3, 6, 3, 2, 7]
b = set(a)
c = list(b)
a is [2, 3, 6, 3, 2, 7]
b is {2, 3, 6, 7}
c is [2, 3, 6, 7]
```
Python Sets

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

\[
a = [2, 3, 6, 3, 2, 7] \\
b = \text{set}(a) \\
c = \text{list}(b)
\]

- a is \([2, 3, 6, 3, 2, 7]\)
- b is \([2, 3, 6, 7]\)
- c is \([2, 3, 6, 7]\)

Don’t know order of elements in b

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
- \( A \mid B \), 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></th>
<th>Function for List</th>
<th>Function for Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding element</td>
<td><code>x.append(elt)</code></td>
<td><code>x.add(elt)</code></td>
</tr>
<tr>
<td>Size of collection</td>
<td><code>len(x)</code></td>
<td><code>len(x)</code></td>
</tr>
<tr>
<td>Combine collections</td>
<td><code>x + y</code></td>
<td>`x</td>
</tr>
<tr>
<td>Iterate over</td>
<td><code>for elt in x:</code></td>
<td><code>for elt in x:</code></td>
</tr>
<tr>
<td>Element membership</td>
<td><code>elt in x</code></td>
<td><code>elt in x</code></td>
</tr>
<tr>
<td>Index of an element</td>
<td><code>x.index(elt)</code></td>
<td><code>CANNOT DO THIS</code></td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th></th>
<th>Function for List</th>
<th>Function for Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding element</td>
<td><code>x.append(elt)</code></td>
<td><code>x.add(elt)</code></td>
</tr>
<tr>
<td>Size of collection</td>
<td><code>len(x)</code></td>
<td><code>len(x)</code></td>
</tr>
<tr>
<td>Combine collections</td>
<td><code>x + y</code></td>
<td>`x</td>
</tr>
<tr>
<td>Iterate over</td>
<td><code>for elt in x:</code></td>
<td><code>for elt in x:</code></td>
</tr>
<tr>
<td>Element membership</td>
<td><code>elt in x</code></td>
<td><code>elt in x</code></td>
</tr>
<tr>
<td>Index of an element</td>
<td><code>x.index(elt)</code></td>
<td>CANNOT DO THIS</td>
</tr>
</tbody>
</table>

- 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)

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

colorSet is

order?
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)`
  - \textbf{Returns new list} of sequence in sorted order
  - Sequence could be list, tuple, string
  - Sequence could be set!

\[
a = \text{set( [3, 5, 2, 1, 7, 2, 5] )} \\
b = \text{sorted(a)}
\]

\[
a \text{ is } \{ 3, 5, 2, 1, 7 \} \\
b \text{ is } [ 1, 2, 3, 5, 7]
\]
WOTO-3 Sets