Compsci 101
Sets, Simple Sorting

Susan Rodger
Feb 24, 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 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 March 3
  • Already looked at one in Lab, one in Lecture!
• Assignment 3 due Tuesday, March 1

• Lab 7 Friday, there is a prelab available now!
• No lab on Friday, March 4

• Take APT Quiz 1 – Feb. 24-27
  • Two parts – each part 1.5 hours, 2 APTs
  • Start on Sakai under quizzes
PFTD

• Simple Sorting
• Sets and APTs
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]  # lst is [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

\[
lst = [6, 2, 9, 4, 3] \\
lst\text{a} = \text{sorted}(\text{lst}) \\
b = ['ko', 'et', 'at', 'if'] \\
c = \text{sorted}(b) \\
b.\text{remove('et')} \\
b.\text{append(6)} \\
b.\text{insert(1, 5)} \\
c = \text{sorted}(b)
\]

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

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

lsta = sorted(lst)

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”

f = ‘go far’
f is ‘go far’

g = sorted(f)
g is ['d', 'o', 'r', 'w']

f = ‘go far’
f is ‘go far’

h = sorted(f.split())
h is ['far', 'go']
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()

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

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

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

\[ \text{lsta} = [6, 2, 9, 4, 3] \]
\[ \text{lstb} = \text{sorted(lsta)} \]

\[ \text{lsta.sort()} \]
\[ \text{a} = [7, 2, 9, 1] \]
\[ \text{b} = \text{a.sort()} \]

\[ \text{c} = (5, 6, 2, 1) \]
\[ \text{c.sort()} \]
\[ \text{d} = \text{“word”} \]
\[ \text{d.sort()} \]
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
```

Sorted list:
- `lstb` is `[2, 3, 4, 6, 9]`

Original list:
- `lsta` is `[6, 2, 9, 4, 3]`
- `lsta` is still `[6, 2, 9, 4, 3]`
- `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!
WOTO-1 Sorting

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

\[
x = [5, 3, 4, 3, 5, 1]
\]
\[
y = \text{set}(x)
\]
\[
y.\text{add}(6)
y.\text{add}(4)
\]
\[
x \text{ is } [5, 3, 4, 3, 5, 1]
\]
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

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

Don’t know order of elements!

\[
y.\text{add}(6)
y.\text{add}(4)
\]

\[
x \text{ is } [5, 3, 4, 3, 5, 1]
y \text{ is } \{3, 1, 4, 5\}
\]

\[
y \text{ is } \{3, 6, 1, 4, 5\}
\]
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 = set(a)
c = list(b)
```

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

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

\[
\begin{align*}
a \text{ is } &\ [2, 3, 6, 3, 2, 7] \\
b \text{ is } &\ \{2, 3, 6, 7\} \\
c \text{ is } &\ [2, 3, 6, 7 ]
\end{align*}
\]
Python Sets

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

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

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 | 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>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)
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!

```python
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!

```python
a = set([3, 5, 2, 1, 7, 2, 5])
b = sorted(a)
appears in

a is {3, 5, 2, 1, 7}
b is [1, 2, 3, 5, 7]
```
APT: EatingGood

Problem Statement

We want to know how many different people have eaten at a restaurant this past week. The parameter meals has strings in the format "name:restaurant" for a period of time. Sometimes a person eats at the same restaurant often.

Return the number of different people who have eaten at the eating establishment specified by parameter restaurant.

For example, "John Doe:Moes" shows that John Doe ate one meal at Moe's.

Write function howMany that given meals, a list of strings in the format above indicating where each person ate a meal, and restaurant, the name of a restaurant, returns the number of people that ate at least one meal at that restaurant.

Specification

```python
filename: EatingGood.py

def howMany(meals, restaurant):
    ""
    Parameter meals a list of strings with each in the format "name:place-ate". Parameter restaurant is a string
    return # unique name values where place-ate == restaurant
    ""

    # you write code here
    return 0
```

APT Eating Good Example

```python
meals = ["Sue:Elmos", "Sue:Elmos", "Sue:Elmos"]

restaurant = "Elmos"

returns 1
```
WOTO-3: APT Eating Good


- https://www2.cs.duke.edu/csed/pythonapt/eatinggood.html
APT Eating Code Idea

• Make an empty list
• Loop over each meal
  • Split the meal into person and restaurant
  • If the restaurant matches
    • If person not already in list
      – Add person to the list
• Return the length of the list
APT Eating Code Idea

• Make an empty list
• Loop over each meal
  • Split the meal into name and restaurant
  • If the restaurant matches
    • If name not already in list
      – Add name to the list
• Return the length of the list
APT Eating Code – Use set instead of list

• Make an empty list
  names = set()

• Loop over each meal
  • Split the meal into name and restaurant
  • If the restaurant matches
    • If name not already in list
      – Add name to the list
  • Return the length of the list
    return len(names)
APT Eating Code – Use set instead of list

• Make an **empty set**

• Loop over each meal
  • Split the meal into name and restaurant
  • If the restaurant matches
    • Add name to set
  
• Return the length of the **set**

```python
names = set()
names.add(name)
return len(names)
```
Lists or Set?

```python
if name not in names:
    names.add(name)
    names.append(name)
```

- For EatingGood we had to avoid adding the same element more than once
  - Lists store duplicates
  - Sets do not store duplicates