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

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

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

2/24/22 Compsci 101, Spring 2022 1

2/24/22 Compsci 101, Spring 2022 2

2/24/22 Compsci 101, Spring 2022 3

2/24/22 Compsci 101, Spring 2022 4
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)
```

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)
```

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
```
Example

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

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)
lst = 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())

Now, sort lists with .sort() list method

- Want to “change” list elements to sorted order
  - lst is [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()`

<table>
<thead>
<tr>
<th>lsta = [6, 2, 9, 4, 3]</th>
<th>lstb = sorted(lsta)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lstb = sorted(lsta)</code></td>
<td>lstb is [6, 2, 9, 4, 3]</td>
</tr>
<tr>
<td><code>lstb = sorted(lsta)</code></td>
<td>lstb is [2, 3, 4, 6, 9]</td>
</tr>
<tr>
<td><code>lstb = sorted(lsta)</code></td>
<td>lstb is still [6, 2, 9, 4, 3]</td>
</tr>
<tr>
<td><code>lstb = sorted(lsta)</code></td>
<td>lstb is [2, 3, 4, 6, 9]</td>
</tr>
<tr>
<td><code>lstb = sorted(lsta)</code></td>
<td>lstb is [2, 3, 4, 6, 9]</td>
</tr>
</tbody>
</table>

```python
lsta = [6, 2, 9, 4, 3]
lsta.sort()  # lsta is still [6, 2, 9, 4, 3]
a = [7, 2, 9, 1]
b = a.sort()  # b is None
```

```python
c = (5, 6, 2, 1)
c.sort()  # c is (5, 6, 2, 1)
d = "word"
d.sort()  # d is 'word'
```

**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!**

---

**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

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

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

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

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

### Python Sets

- Operations on sets:
  - Modify:
    - add  
      ```python
      a.add(7)
      ```
    - clear
      ```python
      a.clear()
      ```
    - remove
      ```python
      a.remove(5)
      ```
  - Create a new set:
    ```python
    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><strong>Adding element</strong></td>
<td><code>x.append(elt)</code></td>
</tr>
<tr>
<td><strong>Size of collection</strong></td>
<td><code>len(x)</code></td>
</tr>
<tr>
<td><strong>Combine collections</strong></td>
<td><code>x + y</code></td>
</tr>
<tr>
<td><strong>Iterate over</strong></td>
<td><code>for elt in x:</code></td>
</tr>
<tr>
<td><strong>Element membership</strong></td>
<td><code>elt in x</code></td>
</tr>
<tr>
<td><strong>Index of an element</strong></td>
<td><code>x.index(elt)</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`

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
```

```python
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', 'yellow'] 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)
```

Set Operations – Difference

```python
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)
APT Eating Good

**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 "name:place-ate", `restaurant` a string, returns the number of unique name values where place-ate == `restaurant`.

```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](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 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
• 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

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