CSCI 134 Fall 2021:
List Methods & Mutability

October 06, 2021

Shikha Singh, 9AM
Jeannie Albrecht, 10AM
Announcements & Logistics

- **Lab 3** graded feedback will be returned soon: make sure to review it
- **Lab 4 Part 1** is due tonight/tomorrow at 10 pm
  - We will run some tests on `voting.py` and return minimal feedback
- **Part 2** is due Oct 13/14 at 10 pm. No lab on Oct 11/12
  - We will let you know about herd meetings and TA hours for this weekend/next week
  - We will have office hours during reading period
- You can attempt *plurality* and *borda* in Part 2 based on what you know now. If you want to implement *rankedChoice*, you need *while loops* which we will get on Friday: can also read the textbook Chapter 7.3
- **Midterm reminder**: Wed Oct 20th: Slots: 6 - 7:30 pm, 8 - 9:30 pm
- **Midterm review**: Monday, Oct 18th 7 - 8 pm

Do You Have Any Questions?
Honor Code Reminder

• “Any work that is not your own is considered a violation of the Honor Code.”

• “Help locating errors and interpreting error messages are allowed, but a student may only receive help in correcting errors of syntax; help in correcting errors of logic is strictly forbidden.”

• “In general, if you are taking photos of someone else’s screen, looking at someone else’s screen, or telling someone else what to type, it is likely your/their work is no longer the work of an individual student.”

• If you aren’t sure if something is considered a violation, just ask (beforehand)!
Last Time

- Reviewed **CSV file reading** and accessing **lists of lists**
- Used **random** module:
  - `random.randint(0, num)` generates random integers between 0 and num (**both inclusive!!**)
- Used our knowledge about lists and loops to analyze interesting properties of our data
  - Focused on maintaining the state of variables when looping, and how to update state based on conditionals
  - Example functions: **mostVowels, leastVowels**
Today’s Plan

• Learn about writing and appending to files
• Review useful list methods:
  • Methods that don't modify lists: .index(), .count()
  • Methods that do modify lists:
    • .append(), .extend(), .insert(),
      .remove(), .pop(), .sort()
• Discuss sorted() function
  • Sorting strings using the ASCII values of their characters
• Discuss mutability and aliasing in Python
Writing to Files

• We know how to **read from** files

• We can also **write to** files

• We can write (or save) all the results that we are computing into a file (a persistent structure). To open a **new** file for writing, we use `open` with the mode `'w'`.

```python
fYears = len(yearList(allStudents, 25))
sophYears = len(yearList(allStudents, 24))
jYears = len(yearList(allStudents, 23))
sYears = len(yearList(allStudents, 22))
mostVowelNames = ',
'.join(mostVowels(firstNames))
leastVowelNames = ',
'.join(leastVowels(firstNames))
with open('studentFacts.txt', 'w') as sFile:
sFile.write('Fun facts about CS134 students:
')  # need newlines
sFile.write('Students with most vowels in their name: {}
'.format(mostVowelNames))
sFile.write('Students with least vowels in their name: {}
'.format(leastVowelNames))
sFile.write('No. of first years in CS134: {}
'.format(fYears))
sFile.write('No. of sophomores in CS134: {}
'.format(sophYears))
sFile.write('No. of juniors in CS134: {}
'.format(jYears))
sFile.write('No. of seniors in CS134: {}
'.format(sYears))
```
Aside: Format Printing for Python Strings

• A quick way to build strings with a particular form is to use the `.format` function on them.

Syntax: `myString.format(*args)`

`*args` means the functions expects zero or more arguments.

• For every pair of braces (`{}`), format consumes one argument.

• Arguments are implicitly converted to a string (with `str()`) and concatenated with the remaining parts of the format string.

• Especially useful in printing: called `format printing`.

```
In [8]: "Hello, you {} world{}".format("silly","!")  # creates a new string
Out[8]: 'Hello, you silly world!'

In [9]: print("Hello, {}.").format("you silly world!"))
   Hello, you silly world!.
```
Appending to Files

- If a file already has something in it, opening it in `w` mode again will erase all of its past contents.
- If we need to **append** something to an **existing** file without erasing the contents, we open it in append `a` mode.

```python
with open('studentFacts.txt', 'a') as sFile:
    sFile.write('Goodbye.
')
```
Summary of List Methods

Methods that do not modify list:
.index(), .count()
Useful List Methods: `index()`

- `myList.index(item)`: returns the first index (int) of item in myList if it is present, else throws an error
  - Method *does not modify* the list it is called on

Example.

```python
>>> myList = ['1', '7', '3', '4', '5']
>>> myList.index('3')  # return index of 3
2
>>> myList.index('10')
ValueError
```
Useful List Methods: `count()`

- `myList.count([item])`: counts and returns the number of times `item` appears in `myList` (as an `int`)
- Method *does not modify* the list it is called on

**Example.**

```python
>>> myList = [2, 3, 2, 1, 2, 4, 1]
>>> c = myList.count(2)
>>> c
3
>>> myList
[2, 3, 2, 1, 2, 4, 1]
```
Summary of List Methods

Methods that DO modify list:

.append(), .extend(),
.pop(), .insert(), .remove(), .sort()
(Aside) Direct Modification: Assignment

`myList[index] = item`: though not a method, an assignment can modify a list directly

**Example.**

```python
myList[1] = 7 # assign 7 to index 1 of myList
```

**myList Before**

\[1, 2, 3, 4]\n
**myList After**

\[1, 7, 3, 4]\n
append()

myList.append(item)  :  appends item to end of list

Example.

myList.append(5)  # insert 5 at the end of the list

myList Before
[1, 7, 3, 4]

myList After
[1, 7, 3, 4, 5]
**extend()**

myList.extend([itemList]): appends all the items in itemList to the end of myList.

**Example.**

myList.extend([6, 8])  # insert both 6 and 8 at the end of the list

<table>
<thead>
<tr>
<th>myList Before</th>
<th>myList After</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 7, 3, 4, 5]</td>
<td>[1, 7, 3, 4, 5, 6, 8]</td>
</tr>
</tbody>
</table>
**pop()**

`myList.pop(index)`: Removes the item at a given index (int) and returns it. If no index is given, by default, `pop()` removes and returns the last item from the list.

**Example.**

```python
val = myList.pop(3)  # returns val = 4
```

```
[1, 7, 3, 4, 5, 6, 8]  # myList Before
```

```
[1, 7, 3, 5, 6, 8]  # myList After
```
**pop()**

**myList.pop(index):** Removes the item at a given index (**int**) and returns it. If no index is given, by default, **pop()** removes and returns the last item from the list.

**Example.**

val = myList.pop()  
returns  
val = 8

---

**myList Before**  
[1, 7, 3, 5, 6, 8]

**myList After**  
[1, 7, 3, 5, 6]
myList.insert(index, item): inserts item at index (int) in myList, all items to the right of index shift over to make room

Example.

myList.insert(0, 11)  # insert 11 at index 0

myList Before

[1, 7, 3, 5, 6]

myList After

[11, 1, 7, 3, 5, 6]
myList.insert(index, item): inserts item at index (int) in myList, all items to the right of index shift over to make room

Example.
myList.insert(10,12)  # insert 12 at index 10

**myList Before**
[11, 1, 7, 3, 5, 6]

**myList After**
[11, 1, 7, 3, 5, 6, 12]
remove()

**myList.remove(item):** removes *item* from *myList*, all items to the right of removed item shift to the left by one

(Unlike `pop()`, item is not returned!)

**Example.**

```
myList.remove(12)  # remove 12 from myList
```

**myList Before**

```
[11, 1, 7, 3, 5, 6, 12]
```

**myList After**

```
[11, 1, 7, 3, 5, 6]
```
sort()

`myList.sort(item)`: sorts the list in place in ascending order

**Example.**

`myList.sort()`  # sort by mutating `myList`

```
myList Before: [11, 1, 7, 3, 5, 6]
myList After: [1, 3, 5, 6, 7, 11]
```
sort() vs sorted()

- `sort()` *method* is only for lists and sorts by mutating the list in place.
- Python provides a built in *function* `sorted()` that can be used to sort any sequence (strings, lists, tuples). It returns a new sorted list, and does NOT modify the original sequence!

**Example.**

```python
list1 = [6, 3, 4];  list2 = [6, 3, 4]
list1.sort() # sort by mutating list1
sorted(list2) # returns a new sorted list
```

<table>
<thead>
<tr>
<th>list1 Before</th>
<th>list1 After</th>
<th>list2 Before</th>
<th>list2 After</th>
<th>Does not change!</th>
</tr>
</thead>
<tbody>
<tr>
<td>[6, 3, 4]</td>
<td>[3, 4, 6]</td>
<td>[6, 3, 4]</td>
<td>[6, 3, 4]</td>
<td></td>
</tr>
</tbody>
</table>
Sorting Strings

- Strings are sorted based on the **ASCII values** of their characters.
- ASCII stands for **“American Standard Code for Information Interchange”**
- Common character encoding scheme for electronic communication (that is, anything sent on the Internet!)
- Special characters come first, followed by capital letters, then lowercase.
- Characters encoded using integers from **0–127**.
- Can use Python functions `ord()` and `chr()` to work with these:
  - `ord(str)`: takes a character and returns its ASCII value as `int`.
  - `chr(int)`: takes an ASCII value as `int` and returns its corresponding character (str).
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>[NULL]</td>
<td>32</td>
<td>20</td>
<td>[SPACE]</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>[START OF HEADING]</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>[START OF TEXT]</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>[END OF TEXT]</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>[END OF TRANSMISSION]</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>[ENQUIRY]</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>[ACKNOWLEDGE]</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>[BEL]</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>[BACKSPACE]</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>[HORIZONTAL TAB]</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>[VERTICAL TAB]</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>[FORM FEED]</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>[CARRIAGE RETURN]</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>[SHIFT OUT]</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>[SHIFT IN]</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>[DATA LINK ESCAPE]</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>[DEVICE CONTROL 1]</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>[DEVICE CONTROL 2]</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>[DEVICE CONTROL 3]</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>[DEVICE CONTROL 4]</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>[NEGATIVE ACKNOWLEDGE]</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>[SYNCHRONOUS IDLE]</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>[ENG OF TRANS. BLOCK]</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>[CANCEL]</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>[END OF MEDIUM]</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>[SUBSTITUTE]</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>[ESCAPE]</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>[FILE SEPARATOR]</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>[GROUP SEPARATOR]</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>[RECORD SEPARATOR]</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>[UNIT SEPARATOR]</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>\</td>
<td>127</td>
<td>7F</td>
<td>[DEL]</td>
</tr>
</tbody>
</table>
sorted() returns a list

- **Note:** sorted(string) returns a sorted list (not string!)

```python
In [1]: sorted('shikha')
```

```
Out[1]: ['a', 'h', 'h', 'i', 'k', 's']
```

```python
In [2]: sorted('jeannie')
```

```
Out[2]: ['a', 'e', 'e', 'i', 'j', 'n', 'n']
```

```python
In [3]: sorted('Hello World')
```

```
Out[3]: [' ', 'H', 'W', 'd', 'e', 'l', 'l', 'l', 'o', 'o', 'r']
```
5. Data Structures

This chapter describes some things you’ve read about already in more detail, and adds some new things as well.

5.1. More on Lists

The list data type has some more methods. Here are all of the methods:

`list.append(x)`

Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`

`list.extend(iterable)`

Extend the list by appending all the items from the iterable. Equivalent to `a[len(a):] = iterable`.

`list.insert(i, x)`

Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.remove(x)`

Remove the first item from the list whose value is equal to `x`. It raises a `ValueError` if there is no such item.

`list.pop([i])`

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the `i` in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

`list.clear()`

Remove all items from the list. Equivalent to `del a[:]`

`list.index(x[, start[, end]])`

Return zero-based index in the list of the first item whose value is equal to `x`. Raises a `ValueError` if there is no such item.

The optional arguments `start` and `end` are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the `start` argument.

`list.count(x)`

Return the number of times `x` appears in the list.
Mutability and Aliasing
Value vs Identity

- Python is an **object oriented language**

- An **object’s identity** never changes in Python once it has been created; think of it as the object’s address in memory

- On the other hand, an **object’s value** can change
  - Objects whose values can change are called **mutable**; objects whose values cannot change are called **immutable**

- The `is` operator compares the identity of two objects (i.e., do they have the same memory address?)

- The `==` operator compares the value of an object (i.e., are the contents of the objects the same?)

- The `id()` function returns an integer representing an object’s identity (or address)

- **Question.** Which mutable objects have you encountered so far?
Mutability in Python

Strings, Ints, Floats are Immutable

- Once you create them, their value cannot be changed!
- All functions that we have seen on these objects return a new object and do not modify the original object

Lists are Mutable

- List values can be changed
- We have seen how we can mutate/change what’s in a list in many ways
- The mutability of lists has many implications such as aliasing, which can cause more trouble if we are not careful!
Mutability in Python

```python
>>> num = 5

>>> num = num + 1
```

**Ints and Floats are Immutable**
Mutability in Python

```python
>>> word = 'Williams'
>>> college = word
>>> word == college
True
>>> word is college
True
```

Even though `word` and `college` have the same identity and value now, if we update one of them it would just assume a new identity!

**Strings are Immutable**
Mutability in Python

```python
>>> word = 'Williams'
>>> college = word
>>> word == college
True
>>> word is college
True
>>> word = 'College'
>>> word == college
False
>>> word is college
False

Strings are Immutable
```
Mutability in Python

Lists are Mutable

```python
>>> myList = [1, 2, 3]
>>> newList = [1, 2, 3]
>>> list2 = myList
>>> myList == newList
True  # True; values are the same
>>> myList is newList
False  # False; identities are different
>>> myList == list2
True
>>> myList is list2
True
```

Lists are Mutable
Mutability in Python

Lists are Mutable

```python
>>> myList = [1, 2, 3]
>>> newList = [1, 2, 3]
>>> list2 = myList
>>> myList == newList
True
>>> myList is newList
False
>>> myList == list2
True
>>> myList is list2
True
>>> myList.append(4)
```

Lists are Mutable
Summary: Mutability in Python Sequences

Strings are Immutable

>>> word = 'Williams'
>>> college = word
>>> word == college
True
>>> word is college
True

Lists are Mutable

>>> myList = [1, 2, 3]
>>> newList = [1, 2, 3]
>>> list2 = myList
>>> myList == newList
True
>>> myList is newList
False
>>> myList == list2
True
>>> myList is list2
True

>>> myList.append(4)

Even though word and college have the same identity and value now, if we update one of them it would just assume a new identity!

>>> myList = [1, 2, 3]
>>> newList = [1, 2, 3]
>>> list2 = myList
>>> myList == newList
True
>>> myList is newList
False
>>> myList == list2
True
>>> myList is list2
True
>>> myList.append(4)
Seq Operations: Return a new Seq!

• The following operations, that can be performed on both lists and strings, and always return a new list/string
  • \texttt{sorted(sequence)}: returns a new sorted sequence
  • \texttt{[::]} slicing operator: returns a new sliced sequence
  • assignment of a new sequence to a variable
    • \texttt{names = 'Shikha and Jeannie'}
    • \texttt{myList = [1, 2, 3]}
  • concatenation always creates a new sequence
  • functions like \texttt{len()}, accessing an element using an index, etc do not modify the sequence
List Aliasing

• Any assignment or operation that “points” to a list implicitly creates an alias.

```python
>>> myList = [1, 2, 3]
>>> list2 = myList  # creates an alias!
>>> newList = [1, 2, 3]
>>> list2 is myList                        True
>>> myList is newList                      False
```
List Aliasing

• Any assignment or operation that “points” to a list implicitly creates an alias

```python
>>> myList = [1, 2, 3]
>>> list2 = myList  # creates an alias!
>>> newList = [1, 2, 3]
>>> list2 is myList                        True
>>> myList is newList                      False
>>> myList.append(4)                       # also changes list2!
[1, 2, 3, 4]
```
ints, floats, and strings are NOT mutable

- `int`, `str` and `float` are immutable
- Once created they can never be changed.
- Any operation on them always creates a new object.
- Aliases aren’t a problem!

```python
name = 'gryffindor'
```
ints, floats, and strings are NOT mutable

- **int**, **str** and **float** are immutable
- Once created they can never be changed.
- Any operation on them always creates a new object.
- Aliases aren’t a problem

```python
name = 'gryffindor'
name = name[4:8]  # returns new string, value gets assigned to name

'gryffindor'  # 'find'
```

Aliasing Examples

In [1]:
   
   nums = [23, 19]
   words = ['hello', 'world']
   mixed = [12, nums, 'nice', words]

In [2]:
   
   words.append('sky')

In [3]:
   
   mixed

Out[3]:
   [12, [23, 19], 'nice', ['hello', 'world', 'sky']]

In [4]:
   
   mixed[1].append(27)

In [5]:
   
   nums

Out[5]:
   [23, 19, 27]

In [6]:
   
   mixed

Out[6]:
   [12, [23, 19, 27], 'nice', ['hello', 'world', 'sky']]
Aliasing Examples

In [9]:
   def foo(someList):
      someList.append('*')

   newList = ['#']
   bar = foo(newList)

In [10]:
   newList

Out[10]: ['#', '*']

In [12]:
   def foo(someList):
      print(id(someList))
      someList.append('*')

   newList = ['#']
   print(id(newList))
   bar = foo(newList)

4462680064
4462680064
Moving on...
Tuples: New Immutable Sequence

• Tuples are an immutable sequence of values separated by commas and enclosed within parentheses ( )

In [1]:
# string tuple
names = ('Shikha', 'Jeannie', 'Kelly', 'Lida')

# num tuple
primes = (2, 3, 5, 7, 11)

# singleton
num = (5,)

# parens are optional
values = 5, 6

# empty tuple
emp = ()
Tuples as an Immutable Sequence

- Tuples, like strings, support any sequence operation that does not involve mutation: e.g.,
  - \texttt{len()} function
  - indexing into tuples
  - concatenation: +, *
  - slicing operator [::]
  - \texttt{in} and \texttt{not in} operators to check membership
  - iteration over tuples
- See examples of these in today's Jupyter notebook
Multiple Assignment and Unpacking

• Tuples support simple and nifty assignment for assigning multiple values at once, and for also "unpacking" sequence values

```python
>>> a, b = 4, 7

>>> b, a = a, b  # reverses the values

>>> studentInfo = ['Harry Potter', 11, True]

>>> name, age, glasses = harryInfo  # tuple assignment to unpack lists!

# is just better/ more concise way of writing:
# name = studentInfo[0]
# age = studentInfo[1]
# glasses = studentInfo[2])
```
Multiple Return from Functions

• When returning multiple values from functions, tuples can come in handy as well

```python
In [1]: # multiple return values as a tuple
def arithmetic(num1, num2):
    ''' Takes two numbers and returns the sum and product'''
    return num1 + num2, num1 * num2

In [2]: arithmetic(10, 2)
Out[2]: (12, 20)

In [3]: type(arithmetic(3, 4))
Out[3]: tuple
```
Tuples and Enumerate

• Python's built-in function `enumerate` takes an iterable sequence `seq` as input and returns an enumerate object (essentially a tuple of the values `(index, seq[index])` for that sequence.

• Typically, `enumerate` is used with to iterate over a sequence and its index directly in a for loop

In [1]: seasons = ['Spring', 'Summer', 'Fall', 'Winter']

In [2]: for index, word in enumerate(seasons):
   print(index, word)

0 Spring
1 Summer
2 Fall
3 Winter
Conversion between Sequences

- The functions `tuple`, `list`, and `str` let us convert between sequences

```
In [4]: word = "Williamstown"

In [5]: charList = list(word)

In [6]: charList

Out[6]: ['W', 'i', 'l', 'l', 'i', 'a', 'm', 's', 't', 'o', 'w', 'n']

In [7]: charTuple = tuple(charList)

In [8]: charTuple

Out[8]: ('W', 'i', 'l', 'l', 'i', 'a', 'm', 's', 't', 'o', 'w', 'n')

In [9]: list((1, 2, 3, 4, 5))  # tuple to list

Out[9]: [1, 2, 3, 4, 5]
```
Next Time: While loops