ICS3U: Searching and Sorting Lists

Searching

When data are stored in a list, we can access them through indexing or slicing. Both of these methods specify one or more positions (*indices*) for values; however, sometimes we want to access data by their *values* instead.

We have already seen how to check whether or not an element is present in a list by using `in`.

```
>>> L = [3, 8, 1, 5, 1]
>>> 8 in L
True
>>> 9 in L
False
```

To determine the position of an element in a list, use the `index` method. There is no `find` method like there is with strings, nor is there an `rindex` method.

```
>>> L.index(1)
2
```

Attempting to obtain the index of a non-existent value will result in a run-time error. Always use `in` before attempting to determine an element's index, unless you are certain that it is in the list.

```
>>> L.index(9)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: 9 is not in list
```

To count the number of occurrences of an element in a list, use the `count` method. It behaves in the same manner as it does for strings.

```
>>> L = [15, 23, 8, 15, 47]
>>> L.count(15)
2
```

Sorting

The easiest way to sort an existing list is to use the `sort` method. This sorts the list **in-place**, meaning that the original ordering will be lost. Using sort with no arguments sorts a list in ascending order.

```
>>> L = [5, 7, 1]
>>> L.sort()
>>> L
[1, 5, 7]
```

To sort a list in descending order, you can use the keyword argument `reverse=True`.

```
>>> L.sort(reverse=True)
>>> L
[7, 5, 1]
```

Note that if you simply wish to reverse a list, without putting all elements in descending order, you can use the `reverse` method or slicing.

```
>>> L = [7, 0, 2, 1, 5]
>>> L.reverse()
>>> L
[5, 1, 2, 0, 7]
>>> L = L[::-1]
>>> L
[7, 0, 2, 1, 5]
```

If you want to create a new list instead of modifying an existing one, the built-in function `sorted` can be used instead. It uses the same keyword arguments as `sort`.

```
>>> L = [9, 18, 2, 7]
>>> S = sorted(L, reverse=True)
>>> S
[18, 9, 7, 2]
>>> L
[9, 18, 2, 7]
```

Note that `sorted` can be used on any sequence, including a string. This makes it possible to sort the characters in a string into a list, which is not possible using `sort`.

```
>>> word = "python"
>>> letters = sorted(word)
>>> letters
['h', 'n', 'o', 'p', 't', 'y']
```

An additional parameter, `key`, can be used to specify a function on which to base the sorting criteria. For example, to sort a list of strings based on their length, rather than alphabetically, we can use the built-in function `len` as a parameter.

```
>>> animals = ["donkey", "pony", "Horse"]
>>> animals.sort(key=len)
>>> animals
['pony', 'Horse', 'donkey']
```

Methods for built-in types (`int`, `float`, `str`, and so on) can also be used as the `key`. The code below performs a case-insensitive sort. Recall that by default, Python places all capital letters before lowercase ones, so the element `Horse` would normally precede both `donkey` and `pony` in the list.

```
>>> animals.sort(key=str.lower)
>>> animals
['donkey', 'Horse', 'pony']
```

It is also possible to use one of our own functions, as long as it returns a value. Here is some code that sorts a list of integers based on their last digits.

```
def last_digit(n):
    last = str(n)[-1]
    return last

L = [12, 38, 15, 79, 30]
L.sort(key=last_digit)
print(L)
```

Note that there are no brackets after `last_digit` in the `key`. This is because we are not actually "calling" the function directly here, but rather passing the memory location of our function to the `sort` method.