Lists

Kelly Rivers and Stephanie Rosenthal

15-110 Fall 2019

Announcements

- Exam 1 Last Friday

  How was it?

- Homework 3 Check-in 1 was due on Monday

  How was it?

- Homework 3 Check-in 2 is due Monday at 12-noon!

  Start Early!

Recap – Unit 1

Abstraction:

- Circuits/gates, bytecode, Python programming, algorithms are all levels of abstraction to understand computation
- Binary, interpreted as types and values, within files are all ways to represent data

Algorithms:

- Algorithms need to specify input, output and process
- Python code includes syntax for specifying variables, functions, if/else, while, for-loops, strings, etc
- We still need to debug our code by running/testing it to understand if it has runtime or logical errors.

Unit 2: Data Structures and Efficiency

Data Structures:

- Organize data in meaningful ways

Efficiency:

- Analyze our data structures and algorithms to understand how much memory they use and how fast they run, compare and contrast different algorithms, or improve an algorithm if possible

Lists

Lists represent sequences, in order, of data

Defining Lists

```
L = []                      # Empty list
L = ["a", "b", "c"]         # Lists of the same type
L = ["a", 1, True]          # Lists of different types
L = [0]*8                   # Repetition of a value
```

Functions for Lists

```
L = [1,2,3,4,5,6,7,8,9]

length = len(L)  #len gives you length of list
m = min(L)       #min finds the minimum value
m = max(L)       #max finds the maximum value
s = sum(L)       #sum finds the sum of a list of numbers
```

List Comparisons

```
L = [1,2,3,4,5,6,7,8,9]
M = [2,3,4,5,6,7,8,9]

L == M      # False - checks each item to see if they match
L < M       # True - if the first item is < than second - True
            #       if equal, check second item, and so on
            #       same as how you would compare strings
```

Indexing - Same as for strings

```
L = [1,2,3,4,5,6,7,8,9]
M = [2,3,4,5,6,7,8,9]

L[0]          # 1 - first item in L
M[3]          # 5 - fourth item in M
L[1:]         # Every item from index 1 on
M[:5]         # Every item up to and not including index 5
L[1:] == M # True
```

Modifying Lists

```
L = [1,2,3,4,5,6,7,8,9]
```

```
L = L+[11,12,13] # 11,12,13 get added to the end
L = [0]+L        # 0 is added at the beginning of the list
L = L[2:]        # the first two elements are removed
L = L[:4]        # all but the first four elements are removed
```

Finding Elements in Lists

```
L = [1,2,3,4,5,6,7,8,9]

5 in L          # returns a Boolean if 5 is in the list L
5 not in L      # returns a Boolean if 5 is not in the list L
L.count(5)      # counts the number of 5's in the list
L.index(5)      # returns the index of 5 in the list (4)
```

Swapping Elements in Lists

Like swapping variables, we need a temp variable

Note: Swapping edits the list.

Swap the first two elements:

```
L = [1,2,3,4,5,6,7,8,9]
temp = L[0]
L[0] = L[1]
L[1] = temp
```

Range

L = ["dog", "cat", "mouse"]

index into L: 0 1 2

We can generate all indexes of our list using:

```
range(len(L))          # all values 0 up to but not including len(L)
indexes = list(range(len(L))    # [0,1,2]
```

Note: list() converts a string or an iterable range into a list

Looping/Iterating through Lists

By iterating through the indexes of the list

```
total = 0
for i in range(len(L)):
    total += L[i]      # sum the elements in the list
print(total)
```

By iterating through the items in the list

```
total = 0
for i in L:
    total += i
print(total)
```

Tradeoffs of Iteration Types

You may want to use the iteration through the list when you are only going to use one element at a time and don't need to know index.

If you think you need to use other elements of the list beside the one or you need the order of the elements, you probably want to use the index to loop through instead

Activity

Write a loop that computes two sums:

one for the odd index elements and

one for the even index elements

Print both sums when it is done with the loop

Check your work!