CSC 110 Week 7, Lecture 1

Dr. Bodine

Last week…

- Loops, loops and more loops!
- Definite loops
- Indefinite loops
- Sentinel loops
- Nested loops
- Boolean algebra

This week…

- Data collections
- Lists/arrays
- Functions and methods
- Uses for storing complex data

Basics of storing data

- Do we always need to store the individual data?
- Consider a program that finds the sum…
- What if we then want to go back and do something else with it?
  - Variance/standard deviation
  - Sum of the squares
- Storing sequential data in a list allows us to go back and use the numbers to do something else

Basic characteristics of Lists or Arrays

- Ordered sequence of items
  - We still have to consider type!
- Single name for the list
  - Example: mylist refers to the entire collection
- Indexed
  - Zero-based numbering
  - In math, denote with subscript: $score_0$
  - In many languages, denote with brackets: mylist[0]

More detailed information

- Length of list/array
  - Static vs dynamic
  - Static: Most languages require you to specify this at the start
  - Dynamic: python allows us to add or subtract from length

- Type inside the list
  - Homogeneous vs Hetergeneous
  - Homogeneous: Most languages require all elements to be of the same type*
  - Heterogeneous: Python allows arbitrary mix of data types

*Other languages have other ways of handling mixed-type data structures

Discussion

- Consider the following sets. Are they valid lists in python? Are they valid arrays in general?
  - [0, 1, 1, 3, 5, 8, 13, 21,…]
  - [142, 13, 7, 6, 34]
  - [‘142’, ‘13’, ‘7’, ‘6’, ’34’]
  - [‘142’, ‘13’, ‘7’, ‘6’, ’34’, ‘test’]
  - [‘142’, ‘13’, ‘7’, ‘6’, ’34’, 172]
  - [‘this’, ‘is’, ‘a’, ‘test’]

How have we used them so far?

- Indexed them:
  - mylist[i]
- Found the length of them:
  - len(mylist)
- Iterated over them (looped):
  - for element in mylist
  - for i in range(len(mylist))

Sequence operations: what else can we do?

- What happens when we try to add lists?
- What happens when we try to repeat lists?
- What happens when we try to cut/slice the list?
- What happens when we try to ask if an expression is in the list?

Sequence operations: what else can we do?

- Can't directly add, per se, but we can **concatenate** using + sign
  - Just like we did with strings!
  - Smashes the second entry onto the end of the first
  - Example: test1, test2 = [1,2,3,4],[10,20,30,40]
  - test1+test2 = [1,2,3,4,10,20,30,40]

- Can also **repeat**, just like strings
  - 3*test1 = [1,2,3,4,1,2,3,4,1,2,3,4]

- What if we actually want to add or multiply the elements?

Sequence operations: what else can we do?

- We can **cut/slice** the list, just like for strings
  - mylist = [1,2,3,4]
  - mylist[1:3] = [2,3]
  - Remember, the second entry is strictly greater than the index of the last element in the resulting sliced list.

- What happens when you slice and get only a single element? Is that different than indexing? Why or why not?

Sequence operations: what else can we do?

- We can **ask about membership**
  - <expr> in <seq>
  - Example: mylist = [1,2,3,4]
  - 1 in mylist returns a Boolean (in this case True)

- How could we use this? Can you think of a way to combine this with our previous studies to do something helpful?

List Methods: what else can we do?

- What happens if we try to **add to the end** of the list?
- What happens if we try to **sort** the list?
- What happens if we try to **find** where an entry occurs in the list?
- What happens if try to **insert** an element somewhere in the middle?
- What happens if we try to **remove** elements?
- What happens if we try **extract** an element?

List methods

- <list>.append(<element>)
- <list>.sort()
- <list>.reverse()
- <list>.index(<element>)
- <list>.insert(<index>,<element>)
- <list>.count(<element>)
- <list>.remove(<element>)
- <list>.pop(<index>)

- Remember, these are **methods** so we use the **.** to access them:
  - mylist.<method>(<argument>)
  - What happens if we try to reassign mylist with a method?
- Replace <list> with our list name
- Replace <element> with the element of the list we would like to add or look for
- Replace <index> with the index

Quick check: True or False

- In python lists are mutable but strings are not.
- The size of a list is set at initialization and remains fixed
- In general, arrays are homogeneous. In python, lists are heterogeneous.
- A list must contain at least one item.

Practice working with lists

- Start with an empty list called mylist
  - Try to add the list of numbers from 0 to 10 to the list
  - Try to add an extra element 0 at the start
  - Try to add an element 11 at the end
  - Try to remove the 5^{th} element
  - Get the value of the 5^{th} element

Practice working with lists

- Start with an empty list called mylist
  - Try to add the list of numbers from 0 to 10 to the list
  - Try to add an extra element 0 at the start
  - Try to add an element 11 at the end
  - Try to remove the 5th element
  - Get the value of the 5th element

```
>>> mylist = []
>>> mylist = mylist+list(range(11))
>>> mylist
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> mylist.insert(0,0)
>>> mylist.append(11)
>>> mylist
[0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> mylist.pop(5)
4
>>> mylist
[0, 0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11]
>>> mylist[5]
5
>>>
```

How did we remove elements

- What if we hadn't added that extra 0 at the start of the list. Could we have used the mylist.remove(5) command to remove the 5th element?
  - Better to use the pop command which is mylist.pop(index). Since we were asking to remove the 5th element not the instances of 5.

Example of removing values from list

- What if we had a list that had multiple elements with the same value and we wanted to remove them.
  - Example: List of exam scores scores=[95,76,0,72,87,89,0,95,62,0,55]
  - Assume 0 scores are people who didn't take the exam. Want to remove them before computing the average.

Example of removing values from list

- What if we had a list that had multiple elements with the same value and we wanted to remove them.
  - Example: List of exam scores scores=[95,76,0,72,87,89,0,95,62,0,55]
  - Assume 0 scores are people who didn't take the exam. Want to remove them before computing the average.

```
>>> scores=[95,76,0,72,87,89,0,95,62,0,55]
>>> scores
[95, 76, 0, 72, 87, 89, 0, 95, 62, 0, 55]
>>> scores.remove(0)
>>> scores
[95, 76, 72, 87, 89, 0, 95, 62, 0, 55]
>>>
```

Example of removing values from list

- What if we had a list that had multiple elements with the same value and we wanted to remove them.
  - Example: List of exam scores scores=[95,76,0,72,87,89,0,95,62,0,55]
  - Assume 0 scores are people who didn't take the exam. Want to remove them before computing the average.

```
>>> scores=[95,76,0,72,87,89,0,95,62,0,55]
>>> scores
[95, 76, 0, 72, 87, 89, 0, 95, 62, 0, 55]
>>> while 0 in scores:
        scores.remove(0)


>>> scores
[95, 76, 72, 87, 89, 95, 62, 55]
>>>
```

A fairly big caveat…

- Lists/arrays are very important in computer science but python handles them in a non-universal way. Lists in python are more flexible than arrays in most languages and you need to be careful to make sure that you pay attention to type. When appending to a list, be careful that you understand the resulting impact on the list (does it become mixed type, would it be allowed in other languages?)

Using our lists to do some maths…

- Consider our list of student scores from assignment 3
  - Remember our file that had a list of <First Name> <Last Name> <score>
  - We output grades based on the score

- Now what if I wanted to know how high above or below the average each person scored?
  - Modify our program to compute the average and then print for each student how their score compared to the average
  - <First Name> <Last Name> scored <insert number> <above/below> average

How would we do this?

- First we have to go through the file to get the scores and store them in a list
- After we have the list we can get the average
- Once we have the average we can loop back through the scores and calculate the difference between the score and the average
  - We also need to check if the score is above or below average to be able to print the correct statement!

```
def main():
    #get filename from user and open file
    infilename = input("Enter the name of the file containing the scores: ")
    infile =open(infilename,"r")

    #open outputfile
    outfile = open("gradesif.txt","w")

    scores=[]
    entries=[]

    #read in lines and get scores
    for line in infile:
        entry = line.split()
        entry[2] = eval(entry[2])
        entries.append(entry)
   #turn score into a number from a string
        scores.append(entry[2])

    average = sum(scores)/len(scores)

    for i in entries:
        diff = i[2]-average
        if diff>0:
            print(i[0],i[1],"scored",diff, "points above average")
        elif diff<0:
            print(i[0],i[1],"scored",abs(diff), "points below average")
        elif diff ==0:
            print(i[0],i[1],"scored the same as the average")
```

Let's take a closer look at this code…

- I used multiple lists: entries and scores. Can you think of another way to do this?
- What type were the elements in entries? What type were the elements in scores? Why does it matter?
- What happens if I try to index entries[i]? What happens if I try to index entries[i][j]?