Working with Lists (Ch. 7)

CSC110: INTRO TO COMPUTER PROGRAMMING WITH PYTHON

INSTRUCTOR: BILL BARRY

1

R o a d m a p

☞ Lists

List Algorithms

Dictionaries & Sets
All work due Fri 11:59pm

NSC finals Mon-Wed
Final Exam Tues, 8–10am
Last Day of Quarter: Wed

2

This Week: Lists

- ✓ Intro, Common Operations, Searching with in
- ✓ List Methods, Useful Commands & Functions
- ✓ List/File Exercise
- More on Passing Data to Functions
- List Exercise, Intro to Tuples

More on Passing Data to Functions

3

Proj07 Notes/Requests

If you are writing PyUnit tests, replace your plain call to main() with this code. It will then only run when you are running directly from *your program*, not when the test harness uses it:

```
if __name__ == '__main__':
    main()
```

To make it possible for me to run my test code on your code, please…

- Please pass to fix functions *only* what they need to operate on, e.g., don't pass the part number and the description together to fixPartNumber, just the part number portion
- Return numbers where requested, not strings
- Use these function names:

fixCost	fixModel	fixType
fixDescription	fixPartNumber	fixYearEnd
fixLocation	fixPrice	fixYearStart
fixMake	fixQuantity	

4

Introduction, Common Operations, Using `in` for Searching

5

List Analogy

When you write a grocery list, you often consider it to be **one** thing

- "What are you writing?" "My grocery list"
- "I forgot my grocery list. Would you please grab it for me?"

But when you get to the store, it also serves a **collection of individual items**:

- "What's the first item on the list?"
- "Is butter on the list?" (you look at each item to see if butter is listed there)

You can process the list using **iteration**. Shopping becomes:

- For each item on the list: find it, grab it, put it in your basket, check it off the list

6

Introduction to Sequences and Lists

Sequences are objects that contain multiple items of data, one after another

Python offers several of these including **Strings**, **Lists**, and **Tuples**

Sequences support these operations, at least:
- Indexing, slicing, membership, concatenation, length, iteration

Lists and Tuples have many of the same methods

List elements are *mutable*; their contents can be changed during program execution
- We'll spend most of our time here

Tuple elements are *immutable*; their contents may not be changed once created
- We'll do a quick look at these

7

Typical List Operations #1

Lists can be created using square bracket notation:
```
myFriends = ['Dana', 'Chris', 'Lupe', 'Pat']
busExpenses = []          # empty list
```
Lists can contain items of disparate data types:
```
clientData = ['John Doe', 432.94, 43]
```
You can create lists by converting ranges using the list function:
```
myNums = list(range(3, 7)) # yields: [3, 4, 5, 6]
```
…and there are other things that can be converted similarly, as we'll see later

8

Typical List Operations #2

You can display lists:
```
print(myNums)                # result:   [3, 4, 5, 6]
```
You can access list items by index:
```
print(myFriends[1])          # result:  'Chris'
```
You can iterate through lists using a for loop:
```
myFriends = ['Dana', 'Chris', 'Lupe', 'Pat']
for oneFriend in myFriends:
    print(oneFriend)
```
```
Output
Dana
Chris
Lupe
Pat
```

9

Typical List Operations #3

You can easily determine the count of elements in a list:
```
print len(myFriends)              # result:  4
```
You can change elements (they are mutable):
```
myFriends[2] = 'Alex'             # 'Lupe' → 'Alex'
```
You can concatenate them (returns a new list):
```
friendList2 = ['Corey','Parker']
myFriends += friendList2          # now superset
```
You can slice them (returns a new list):
```
print(myFriends[0:2])             # ['Dana', 'Chris']
```
Note: lists are **themselves** immutable, but their **elements** are mutable

10

Pause & Practice #1

Create lists containing names from each of these groups:
- Best friends
- Good friends
- Work friends

Create a list of **all** friends, a superset of all of these

Display the all-friends list

Display a count of friends on the all-friends list

11

Pause & Practice #1: Code

```
bestFriends = ['Taylor', 'Morgan', 'Hayden']
goodFriends = ['Tracy', 'Peyton']
workFriends = ['Jamie', 'Alex']
allFriends = bestFriends + goodFriends + workFriends
print(len(allFriends), 'friends total')
print(allFriends)
```

```
Output
7 friends total
['Taylor', 'Morgan', 'Hayden', 'Tracy', 'Peyton', 'Jamie', 'Alex']
```

12

Finding Items with **in**/**not** in Operators

As you might expect, it's easy to find out whether items are in a list:

```
myFriends = ['Dana', 'Chris', 'Lupe', 'Pat']
print('Lupe' in myFriends)       # result:  True
print('Alex' in myFriends)       # result:  False
print('Alex' not in myFriends)   # result:  True
```

13

Pause & Practice #2

Create lists for each of these recipe ingredient lists:

- Cake batter: butter, sugar, baking powder, vanilla, egg, flour, milk
- Pie dough: flour, salt, water, butter
- Pasta dough: flour, egg, olive oil

Ask the user for an ingredient they are allergic/sensitive to, and display a message indicating all the recipes that are safe. Example run:

```
Enter an ingredient you are sensitive to: egg
Pie dough is safe
```

Cake batter and pasta dough aren't safe, so they aren't listed

14

Pause & Practice #2: Code

```
cakeBatterIngred = ['butter', 'sugar', 'baking powder', \
                    'vanilla', 'egg', 'flour', 'milk']
pieDoughIngred   = ['flour', 'salt', 'butter', 'water']
pastaDoughIngred = ['flour', 'egg', 'olive oil']
ingredient = input('Enter an ingredient: ')
if ingredient not in cakeBatterIngred:
    print('Cake batter is safe')
if ingredient not in pieDoughIngred:
    print('Pie dough is safe')
if ingredient not in pastaDoughIngred:
    print('Pasta dough is safe')
```

15

Brain Stretch #1

What would this mean?

```
cakeBatterIngred = ['butter', 'sugar', 'baking powder', \
                    'vanilla', 'egg', 'flour', 'milk']
pieDoughIngred   = ['flour', 'salt', 'butter', 'water']
pastaDoughIngred = ['flour', 'egg', 'olive oil']
allIngredLists   = [cakeBatterIngred, pieDoughIngred, \
                    pastaDoughIngred]
listOrder        = ['Cake batter', 'Pie dough', \
                    'Pasta dough']
```

16

Brain Stretch #1, continued

```
ingredient = input('Enter an ingredient: ')
for recipeIdx in range(0, len(allIngredLists)):
    if ingredient not in allIngredLists[recipeIdx]:
        print(listOrder[recipeIdx], 'is safe')
```

Because we have parallel arrays, we need the loop variable for an index into both arrays. We can't just iterate through allIngredLists as we typically would

17

More List Methods

18

List Methods: `index`

To find the first occurrence of an item in the list, use the index method:

```
myFriends = ['Dana', 'Chris', 'Lupe']
print(myFriends.index('Lupe'))       # 2
```

Note: if you search for an item that doesn't exist, you'll get a ValueError exception; use `in` or `count` to make sure ahead of time

19

List Methods: `count`

To count how many times an item appears in the list, use the count method:

```
myFriends = ['Dana', 'Chris', 'Lupe', 'Dana']
print(myFriends.count('Dana'))       # 2
print(myFriends.count('Pat'))        # 0
```

20

List Methods: append

To add items to the end of a list, use the append method:

```
myFriends = ['Dana', 'Chris', 'Lupe']
notherOne = 'Pat'
myFriends.append(notherOne)
print(myFriends)     # ['Dana', 'Chris', 'Lupe', 'Pat']
```

21

List Methods: insert

To add items in a specific position, use the insert method:

```
myFriends = ['Dana', 'Chris', 'Lupe']
myFriends.insert(1, 'Pat')
print(myFriends)       # ['Dana', 'Pat', 'Chris', 'Lupe']
```

22

List Methods: `remove`

To remove the first occurrence of a specific item, use the remove method:

```
myFriends = ['Dana', 'Chris', 'Lupe', 'Dana']
noMoFriend = 'Dana'
myFriends.remove(noMoFriend)
print(myFriends)     # ['Chris', 'Lupe', 'Dana']
```

To remove all occurrences, use a loop based on `count` results

23

List Methods: `sort`

To sort the contents of the list (ascending order), use the sort method:

```
myFriends = ['Pat', 'Chris', 'Lupe', 'Dana']
myFriends.sort()
print(myFriends)   # ['Chris', 'Dana', 'Lupe', 'Pat']
```

Numbers will be sorted from smallest to largest; strings will be sorted alphabetically. To sort in reverse order, use this form:

```
myFriends.sort(reverse = True)
print(myFriends)   # ['Pat', 'Lupe', 'Dana', 'Chris']
```

24

List Methods: **reverse**

To reverse the order of the list, use the reverse method:

```
myFriends = ['Pat', 'Chris', 'Lupe', 'Dana']
myFriends.reverse()
print(myFriends)     # ['Dana', 'Lupe', 'Chris', 'Pat']
```

25

Other Useful Functions & Commands

A FEW MORE THINGS YOU MIGHT NEED

26

Useful Commands: `del`

To delete a specific list item by index, use the del command:

```
myFriends = ['Pat', 'Chris', 'Lupe', 'Dana']
del myFriends[2]
print(myFriends)     # ['Pat', 'Chris', 'Dana']
```

Note that this is a *command*, not a *method*. It's also useful elsewhere, if you want to delete a variable entirely from memory

27

Useful Functions: `min, max, sum`

To find the smallest or largest items in the list, use the min or max function:

```
myFriends = ['Pat', 'Chris', 'Lupe', 'Dana']
print(min(myFriends), max(myFriends))    # Chris Pat
```

Note: works with numbers as well as text; strings will be handled alphabetically

To add the numbers in a list, use the sum function

```
myBills = [103.84, 75.23, 18.54, 39.73]
print(sum(myBills))  # 237.33999999999997
```

28

Summary

List Method	Description
`.index(item)`	Returns first index at which *item* exists in list
`.count(item)`	Counts the number of times the item occurs
`.append(item)`	Adds *item* to the end of the list
`.insert(index, item)`	Inserts *item* at position *index* in the list
`.remove(item)`	Removes first occurrence of *item* from the list
`.sort()`	Sorts the list in ascending order
`.reverse()`	Reverses the order of the items in the list

Note: List methods don't tend to throw exceptions. Many are forgiving about indexes, i.e., you can specify indexes before and after the list and Python does something reasonable

Useful Function/Command	Description
`del list(index)`	Deletes the item at position *index* from *list*
`min(list)`	Returns the item with the lowest value in *list*
`max(list)`	Returns the item with the highest value in *list*
`sum(list)`	Returns the sum of the items in *list*

Link to good web page on list methods

29

Pause & Practice: Grocery List Working with Files and Lists

Provided
- Code for a small menu-drive program to manage a grocery list. The working code displays the menu and handles common issues like out-of-range menu choices and "press enter to continue" functionality. It also loops so the menu is displayed after each choice, until the user chooses to exit
- A starter grocery list, GroceryList.txt – a file used for reading and writing the grocery list. I recommend you save a backup copy

Task
- Your job: work in groups to make menu choices 1 through 6 work. As always, please follow "we all rise together" protocols; make sure no one dominates the conversation, everyone is participating, and everyone understands choices being made/implemented

```
Grocery List Helper

1. Read list from file
2. Display list
3. Add item to list
4. Remove item from list
5. Sort list
6. Write list to file
0. Exit

Enter choice:
```

30

More on Passing Data to Functions

31

Data Passing Analogy

Consider these two scenarios:

You meet a new friend and want to share your phone number. You text them, and they store your phone number in their phone. The next day, they edit your phone number in their phone

- Q: Does that change *your* phone number?
- A: No, your phone number is **immutable**; it stays the same regardless of their edits

You have a get-well card for a sick coworker; you tell everyone at work, "If you'd like to sign the card for Chris, it's on the conference room table."

- Q: Do coworker well wishes affect *your* version of the card?
- A: Yes, the card's notes are **mutable**; you point to *the* card, they can modify *the* card's contents

In this analogy, **typical variables** passed to functions is like the phone number; it's **immutable** and can't be changed by the function. **List** variables passed to functions are like the card; the list's elements are **mutable** and can be changed by the function

32

Examples: Passing Mutable vs. Immutable Data

```
def changeImmutableType(origValue):
    origValue = origValue * 2

def changeMutableType(origList):
    origList.append('Goodbye')     # note: list not returned

def main():
    myNum = 5
    changeValueType(myNum)
    print(myNum)                   # 5 (unchanged)

    myList = ['Hello']
    changeRefType(myList)
    print(myList)                  # ['Hello', 'Goodbye']
```

33

Lists and `for` Loops

Looping through a list with a for loop works great for display, but not for updating list elements

Consider this code that tries to account for birthdays:

```
friendAges = [29, 36, 32, 27]
for friendAge in friendAges:
    friendAge += 1
print(friendAges)   # result: [29, 36, 32, 27]
```

If you want to *update* contents, access elements by index instead:

```
friendAges = [29, 36, 32, 27]
for friendPos in range(0, len(friendAges)):
    friendAges[friendPos] += 1
print(friendAges)   # result: [30, 37, 33, 28]
```

34

Copying Lists

Don't forget that lists are mutable objects:

```
friends1 = ['Chris', 'Dana']
friends2 = friends1          # variables point to same list in memory
friends1.append('Alex')
print(friends2)              # result:   ['Chris', 'Dana', 'Alex’]
```

If you really want to *copy* the entire list, you must *copy its elements* instead:

```
friends1 = ['Chris', 'Dana']
friends2 = []
for friend in friends1:
    friends2.append(friend)
friends1.append('Alex')
print(friends2)               # result:   ['Chris', 'Dana']
```

35

Pause & Practice #3

Basic Project

Write a function called sortList. It should take one parameter, the list to be sorted. It should sort the list. Consider whether the function needs a return. Test the function.

Advanced Project

Make the function take an additional parameter, a Boolean value indicating whether they want the list in reverse order. Sort accordingly

Super Advanced Project

Make the second parameter optional

36

Pause & Practice #3: Code (Basic)

```
def sortList(theList):
    theList.sort()

myFriends = ['Taylor', 'Hayden', 'Morgan']
sortList(myFriends)
print(myFriends)      # ['Hayden', 'Morgan', 'Taylor']
```

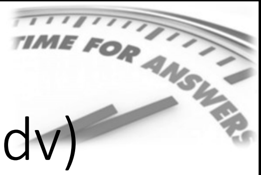
37

Pause & Practice #3: Code (Adv)

```
def sortList(theList, wantReversed):
    if wantReversed:
        theList.sort(reverse = True)
    else:
        theList.sort()

myFriends = ['Taylor', 'Hayden', 'Morgan']
sortList(myFriends, True)
print(myFriends)     # ['Taylor', 'Morgan', 'Hayden']
sortList(myFriends, False)
print(myFriends)     # ['Hayden', 'Morgan', 'Taylor']
```

38



Pause & Practice #3: Code (Super Adv)

```
def sortList(theList, wantReversed = False):
    if wantReversed:
        theList.sort(reverse = True)
    else:
        theList.sort()

myFriends = ['Taylor', 'Hayden', 'Morgan']
sortList(myFriends, True)
print(myFriends)     # ['Taylor', 'Morgan', 'Hayden']
sortList(myFriends)
print(myFriends)     # ['Hayden', 'Morgan', 'Taylor']
```

39

Brain Stretch #2

What would this mean?

```
cakeBatterIngred = ['butter', 'sugar', 'baking powder', \
                    'vanilla', 'egg', 'flour', 'milk']
pieDoughIngred   = ['flour', 'salt', 'butter', 'water']
pastaDoughIngred = ['flour', 'egg', 'olive oil']
allIngredLists   = [cakeBatterIngred, pieDoughIngred, \
                    pastaDoughIngred]
del cakeBatterIngred, pieDoughIngred, pastaDoughIngred
```

40

List Exercise, Intro to Tuples

LET'S USE WHAT WE'VE LEARNED

41

List Exercise

You'll be provided a file that contains the text of Robert Frost's poem "Stopping by Woods on a Snowy Evening"

Task:

Write a function called findUniqueWordsInFile that…

- Takes one parameter, a string representing the name of the file to analyze
- Removes punctuation: . , ;
- Normalizes casing (make all words lower case)
- Returns a *sorted* list of *unique* words (case insensitive)

Call the function with the file name (RobertFrostStoppingByWoods.txt). Print the word count and the resulting word list

42

Unique Words Function: Pseudocode

```
Accept the file name parameter
Create unique word list (empty)
Open file for reading
For each line in the file:
    Get rid of punctuation: . , ;
    Lower case the text
    Split the line into words
    For each word in the split-up list:
        If the word is not already in the unique word list:
            Add the word to the unique word list
Close the file
Sort the unique word list
Return the unique word list
```

43

Lab Exercise: Code, Part 1

```
def findUniqueWordsInFile(fileName):
    uniqueWordList = []
    textFile = open(fileName, 'r')
    for line in textFile:
        line = line.replace('.', '').replace(',', '').replace(';', '').lower()
        words = line.split()
        for word in words:
            if word not in uniqueWordList:
                uniqueWordList.append(word)
    textFile.close()
    uniqueWordList.sort()
    return uniqueWordList
```

44

Lab Exercise: Code, Part 2

```
def main():
    poemWordList = findUniqueWordsInFile( \
                    'RobertFrostStoppingByWoods.txt')
    print('The poem has', len(poemWordList), 'unique words\n')
    print(poemWordList)

main()
```

45

Tuples: Introduction

Python's tuples are like Python lists, except elements are immutable, meaning that once they are created, they can't be changed

The syntactic difference: when creating a tuple, use **parentheses** instead of square brackets around the list of elements. For element access, [] is still used

Sequence methods that **don't** attempt to change the tuple will succeed, including those that generate a new sequence (tuple, in this case)

Sequence methods that **do** attempt to change the tuple will fail

When you return multiple items from a function, Python creates a tuple and passes it back to the caller; its elements are then unpacked into variables

46

Tuples: Code

```
myFriends = ('Dana', 'Chris', 'Lupe', 'Pat')
# these will work just fine
print(myFriends[1])
print(myFriends[0:2])
print(myFriends.index('Lupe'))
# these will fail
myFriends[2] = 'Alex'           # can't change an element
myFriends.append('Alex')        # can't add an element
```

47

The End

48