CS 134:
Lists and Loops
Announcements & Logistics

- **Homework 3** is due tonight @ 11 pm
- **Lab 1** graded feedback was released on Wed
  - Any problems?
- **Lab 3** is today/tomorrow in lab
  - A collection of word puzzles: can use your newly acquired knowledge of strings, lists (today), functions and loops to solve them

Do You Have Any Questions?
Last Time

- Started discussing sequences in Python
- Focused on strings (sequences of characters)
- Discussed slicing and indexing of strings
- Learned about in operator to test membership:
  - Note: There is also a not in operator
- Also learned about string methods .lower() and .upper()
  - Note: There are also string methods .islower() and .isupper() that return True if string is in lowercase/uppercase, else return False
- Introduced for loops as a mechanism to iterate over sequences
Today’s Plan

- Discuss **for loops** in more detail
- Introduce a new sequence: **Lists**
  - Apply indexing [], slicing [:], in, + operators to lists
- Continue building a collection of functions that iterate over sequences (lists and strings)
Recap: Iterating with *for* Loops

- The **loop variable** (char and var in the examples below) takes on the value of each of the elements of the sequence one by one.

```python
for var in seq:
    # loop body
    (do something)
```

```python
# simple example of for loop
word = "Williams"
for char in word:
    print(char)
```

```plaintext
Williams
Williams
Williams
```
Counting Vowels Revisited

• We used a for loop to iterate over the characters in a string (word) and look for vowels (using `isVowel()` from last class)

```python
def isVowel(char):
    """Simpler isVowel function""
    c = char.lower()  # convert to lower case first
    return c in 'aeiou'

def countVowels(word):
    """Takes a string as input and returns the number of vowels in it""

    count = 0  # initialize the counter

    # iterate over the word one character at a time
    for char in word:
        if isVowel(char):  # call helper function
            count += 1

    return count
```

Count is an **accumulation** variable, since we accumulate the count (int) as we go through the loop.
Vowel Sequences Revisited

- We defined a function `vowelSeq()` that takes a string `word` as input and returns a string containing all the vowels in `word` in the same order as they appear. (using `isVowel()` from last class)

```python
def vowelSeq(word):
    '''Returns the vowel subsequence in given word'''
    vowels = ""  # accumulation variable
    for char in word:
        if isVowel(char):  # if vowel
            vowels += char  # accumulate characters
    return vowels
```

`vowels` is an **accumulation** variable, since we accumulate characters (strings) as we go through the loop.
Moving on: Lists

- **Lists** are another type of **sequence** in Python
- **Definition:** *A list is a comma separated sequence of values*
- Unlike strings, which can only contain characters, lists can be collections of **heterogenous objects** (strings, ints, floats, etc)
- Today we’ll focus on **iterating** over lists (i.e., looking at the elements sequentially) using for loops
- In upcoming lectures we’ll focus on manipulating and using lists to store dynamic sequences of objects
Lists

- Lists are:
  - **Comma separated sequences** of values
  - **Heterogenous** collections of objects
  - **Mutable** (or “changeable”) objects in Python. In contrast, strings are **immutable** (they cannot be changed).
  - We will discuss **mutability** in more detail soon!

```python
In [1]: # Examples of various lists:

wordList = ['What', 'a', 'beautiful', 'day']
numList = [1, 5, 8, 9, 15, 27]
charList = ['a', 'e', 'i', 'o', 'u']
mixedList = [3.145, 'hello', 13, True] # lists can be heterogeous

In [2]: type(numList)
Out[2]: list
```
Operations on Sequences

• We already saw several **sequence operators** and functions last time
  • We looked at **strings** last time
  • These apply to **lists** as well!
• We can do the following operations on lists:
  • Indexing elements of lists using `[ ]`
  • Using `len()` function to find length of list
  • Slicing lists using `[:]`
  • Testing membership using `in/not in` operators
  • Concatenation using `+`
Operations on Sequences

In [1]: wordList = ['What', 'a', 'beautiful', 'day']

    wordList[3]

Out[1]: 'day'

Indexing lists using []

In [2]: wordList[-1]

Out[2]: 'day'

Finding length of list using len()

In [3]: len(wordList)

Out[3]: 4

In [4]: nameList = ['Aamir', 'Beth', 'Chris', 'Daxi', 'Emory']

In [5]: nameList[2:4]

Out[5]: ['Chris', 'Daxi']

Slicing lists using [:] (can also use optional step)
Membership in Sequences

- Recall: The `in` operator in Python is used to test if a given sequence is a subsequence of another sequence; returns True or False.

```python
In [20]: namelist = ['Anna', 'Beth', 'Chris', 'Daxi', 'Emory', 'Fatima']

In [28]: 'Anna' in namelist  # test membership
Out[28]: True

In [30]: 'Jeannie' in namelist
Out[30]: False
```
**not in** sequence operator

- The **not in** operator in Python returns True if and only if the given element is **not** in the sequence

```python
In [20]: nameList = ["Anna", "Beth", "Chris", "Daxi", "Emory", "Fatima"]

In [28]: "Anna" in nameList # test membership
Out[28]: True

In [30]: "Jeannie" in nameList
Out[30]: False

In [31]: "Jeannie" not in nameList # not in returns true if el not in seq
Out[31]: True

In [33]: "a" not in "Chris"
Out[33]: True
```

Note that **not in** also works for strings.
List Concatenation

- We can use the `+` operator to **concatenate** lists together.
- Creates a **new list** with the combined elements of the sublists.
  - *Does not modify original lists*

```python
aList = ['the', 'quick', 'brown', 'fox']

bList = ['jumped', 'over', 'the', 'dogs']

aList + bList  # concatenate lists
['the', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'dogs']

aList  # aList is unchanged
['the', 'quick', 'brown', 'fox']

bList = bList + ['back']  # add 'back' to bList

bList  # since we reassign result to bList, bList has changed
['jumped', 'over', 'the', 'dogs', 'back']
```

returns a new list with elements from `aList` and `bList`

`aList` is unchanged!

To change `bList`, we have to reassign `bList` to the new list.
Looping over Lists

• We can **loop** over **lists** the same way we looped over **strings**
• As before, the **loop variable** iteratively takes on the values of each item in the list, starting with the 0th item, then 1st, until the last item
• The following loop iterates over the list of ints, printing each item in it

```
In [15]: numList = [0, 2, 4, 6, 8, 10]

In [16]: for num in numList:
   print(num)

0
2
4
6
8
10
```
Exercise: countItem

- Let’s write a function `countItem()` that takes as input a sequence `seq` (can be a string or a list), and an element `el`, and returns the number of times `el` appears in the sequence `seq`.

```python
def countItem(seq, el):
    """Takes seq as input, and returns the number of times el appears in seq"""
    pass
```
Exercise: `countItem`

Let’s write a function `countItem()` that takes as input a sequence `seq` (can be a string or a list), and an element `el`, and returns the number of times `el` appears in the sequence `seq`.

```python
def countItem(seq, el):
    """Takes seq as input, and returns the number of times el appears in seq""
    count = 0  # initialize counter

    for item in seq:
        if item == el:  # if this item matches el
            count += 1  # increment counter
    # else do nothing, go to next item
    return count
```

Another accumulation variable!
Exercise: `wordStartEnd`

- Write a function that iterates over a given list of strings `wordList`, returns a (new) list containing all the strings in `wordList` that start and end with the same character (ignoring case).

```python
def wordStartEnd(wordList):
    """Takes a list of words wordList and returns a list of all words in wordList that start and end with the same letter""
    pass
```

```python
>>> wordStartEnd(["Anna", "banana", "salad", "Rigor", "tacit", "hope"])
["Anna", "Rigor", "tacit"]
>>> wordStartEnd(["New York", "Tokyo", "Paris"])
[]
>>> wordStartEnd(["*Hello*", ",", "nope"])
["*Hello*"]
```
Exercise: wordStartEnd

• **Step by step approach (organize your work):**
  • Go through every word in wordList
  • Check *if word starts and ends at same letter* *
  • If true, we need to “collect” this word (remember it for later!)
    • Else, just go on to next word
  • Takeaway: need a new list to **accumulate** desirable words

• *Break down bigger steps (decomposition!)*
  • If word starts and ends at same letter:
    • Can do this using string **indexing**
  • Think about **corner cases**: what if string is empty? what about case?
Exercise: wordStartEnd

- Write a function that iterates over a given list of strings `wordList`, returns a (new) list containing all the strings in `wordList` that start and end with the same character (ignoring case).

```python
def wordStartEnd(wordList):
    '''Takes a list of words and returns a list of words in it that start and end with the same letter'''
    result = []
    for word in wordList:
        # check for empty strings before indexing
        if len(word) != 0:
            if word[0].lower() == word[-1].lower():
                result += [word]  # concatenate to resulting list
    return result  # notice the indentation of return
```

Notice this syntax! We are adding word (a string) to result (a list).
Nested Loops

- A **for loop** body can contain one (or more!) additional **for loops**:
  - Called **nesting for loops**
  - Conceptually similar to nested conditionals
- Example: What do you think is printed by the following Python code?

```python
# What does this do?
def mysteryPrint(word1, word2):
    """Prints something""
    for char1 in word1:
        for char2 in word2:
            print(char1, char2)

mysteryPrint('123', 'abc')
```
```python
# What does this do?

def mysteryPrint(word1, word2):
    """Prints something""
    for char1 in word1:
        for char2 in word2:
            print(char1, char2)

mysteryPrint('123', 'abc')
```

Inner loop (w/ char2 and word2) runs to completion on each iteration of the outer loop
Nested Loops

• What is printed by the nested loop below?

```python
# What does this print?
for letter in ['b', 'd', 'r', 's']:
    for suffix in ['ad', 'ib', 'ump']:
        print(letter + suffix)
```
# What does this print?

```python
for letter in ['b', 'd', 'r', 's']:
    for suffix in ['ad', 'ib', 'ump']:
        print(letter + suffix)
```
Lab 3 Notes
Lab 3: Goals

• In this lab, you will accomplish two tasks:

  • Construct a module of tools for manipulating strings and lists of strings (in wordTools.py)

  • Use your toolbox to answer some (fun?) trivia questions (in puzzles.py)

• You will gain experience with the following:

  • Sequences (lists and strings), and associated operators/methods

  • Writing simple and nested for loops

  • Writing doctests to test your functions
Testing Functions: Doctests

• We have already seen two ways to test a function
  • You can run your code 1) interactively or 2) as a script
• Python’s **doctest** module allows you to embed test cases and expected output directly into a function’s docstring
• To use the doctest module, we must import it using:
  ```python
  from doctest import testmod
  ```
• To make sure the test cases are run when the program is run as a script from the terminal, we then need to call **testmod()**.
• To ensure that the tests are not run in interactive Python, we place this command within a “guarded” if block:
  ```python
  if __name__ == '__main__':
  ```
Testing Functions: Doctests

```python
def isVowel(char):
    """"""Takes a letter as input and returns true if and only if it is a vowel.
>>> isVowel('e')
True
>>> isVowel('U')
True
>>> isVowel('t')
False
>>> isVowel('Z')
False
    """""""
    return char.lower() in 'aeiou'
```

```python
if __name__ == '__main__':
    # the following code tests the tests in the docstrings ('doctests').
    # as you add tests, re-run this as a script to test your work
    from doctest import testmod
    # this import is necessary when testing
    testmod()  # test this module, according to the doctests
```