CSCI 134 Fall 2021:
Lists and Loops

September 24, 2021

Shikha Singh, 9AM
Jeannie Albrecht, 10AM
Announcements & Logistics

• **Homework 3** is out on GLOW, due Monday 10 pm

• **Lab 1** graded feedback was released on Wed
  • Any problems?

• **Lab 3** will be released today at noon
  • Watch pre-lab video with your herd and discuss before lab
  • Lab 3 is a collection of word puzzles: can use our newly acquired knowledge of strings, 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()
  • There are also string methods .islower() and .isupper() that return True if string is in lowercase/uppercase, else False
• (Briefly) 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 operator to lists

• Build a collection of functions that iterate over lists and strings

• Build a module for working with sequences
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)
```

```python
W
i
l
l
i
a
m
s
```
Recap: count Vowels

- **Problem:** Write a function `countVowels()` that takes a string `word` as input, counts and returns the number of vowels in the string.

```python
def countVowels(word):
    '''Returns number of vowels in the word'''
    pass

>>> countVowels('Willamstown')
4

>>> countVowels('Ephelia')
4
```
(Bad) Attempt with Conditionals

- Using conditionals as shown is repetitive and does not generalize to arbitrary length words.

- Note that `val += 1` is shorthand for `val = val + 1`.

```python
word = 'Williams'
counter = 0
if isVowel(word[0]):
    counter += 1
if isVowel(word[1]):
    counter += 1
if isVowel(word[2]):
    counter += 1
if isVowel(word[3]):
    counter += 1
if isVowel(word[4]):
    counter += 1
if isVowel(word[5]):
    counter += 1
if isVowel(word[6]):
    counter += 1
if isVowel(word[7]):
    counter += 1
print(counter)
```
Counting Vowels Revisited

• Let’s use a for loop to finish implementing our `countVowels()` function correctly

```python
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 **accumulator** variable, since we accumulate the value as we go through the loop.
Counting Vowels: Tracing the Loop

• How are the local variables updated as the loop runs?

def countVowels(word):
    '''Takes a string as input and returns the number of vowels in it'''
    count = 0
    for char in word:
        if isVowel(char):
            count += 1
    return count

countAllVowels('Boston')
Exercise: vowelSeq

• Define 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. (Hint: we can use `isVowel()` from last class)

```python
def vowelSeq(word):
    '''returns the vowel subsequence in word'''
    pass

>>> vowelSeq("Chicago")
"iao"

>>> vowelSeq("protein")
"oei"

>>> vowelSeq("rhythm")
""
```
Exercise: vowelSeq

• Define 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. (Hint: we can use `isVowel()` from last class)

```python
def vowelSeq(word):
    # returns the vowel subsequence in word''
    vowels = ""  # accumulation variable
    for char in word:
        return vowels
```
Exercise: vowelSeq

- Define 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. (Hint: we can use `isVowel()` from last class)

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

- Next week 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!

```
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 heterogeneous

In [2]: type(numList)
```

Out[2]: list
Operations on Sequences

- We already saw several string operators and functions last time
- Most of these apply to lists as well
- We can do the following on lists:
  - Indexing elements of lists using \([\]\)
  - Using \texttt{len()}\ function to find length
  - Slicing lists using \([::]\)
  - Testing membership using \texttt{in/not in} operators
  - Concatenation using \(+\)
Operations on Sequences

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

    wordList[3]

Out[1]: 'day'

In [2]: wordList[-1]

Out[2]: 'day'

In [3]: len(wordList)

Out[3]: 4

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

In [5]: nameList[2:4]

Out[5]: ['Chris', 'Daxi']
```
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
```
Sequences: not in 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
Strings to Lists: \texttt{split()}

- It is often useful to be able to convert strings to lists, and lists to strings.
- The \texttt{split()} method splits strings at “spaces” (the default separator) and returns a list of (sub)strings.
- Can optionally specify other \texttt{delimiters} as well.

\begin{verbatim}
In [5]: phrase = "What a lovely day"

In [6]: phrase.split()

Out[6]: ['What', 'a', 'lovely', 'day']

In [7]: newPhrase = "What a *lovely* day!"  # multiple spaces or punctuations don't matter

In [8]: newPhrase.split()

Out[8]: ['What', 'a', '*lovely*', 'day!']

In [9]: commaSepSpells = "Impervius, Portus, Lumos, Reducio, Protego"  # comma separated strings

In [10]: commaSepSpells.split(',')

Out[10]: ['Impervius', 'Portus', 'Lumos', 'Reducio', 'Protego']
\end{verbatim}
List to Strings: `join()`

- Given a list of strings, the `join()` string method, when applied to a character `char`, concatenates the strings together with the character `char` between them.
Looping over Lists

• We can loop over lists the same way we loop 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, 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 accumulator variable!
Exercise: `wordStartEnd`

- Write a function that iterates over a given list of words `wordList`, returns a (new) list containing all the words in `wordList` that start and end with the same letter (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 words `wordList`, returns a (new) list containing all the words in `wordList` that start and end with the same letter (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:  # iterate over list
        # 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
```