CS134: Sequences & Loops
Announcements & Logistics

• **Homework 3** is out on GLOW, due next Monday @ 10 pm
  • Covers materials through last lecture (conditionals)
• **Lab 1** graded feedback will be released today
  • Instructions on how to view feedback on course webpage under Resources
• **Lab 2** due today 10pm / tomorrow 10pm
• Rohit will be in Jeannie’s class on Friday

Do You Have Any Questions?
Last Time

• Looked at more complex decisions in Python
  • Used Boolean expressions with and, or, not
• Chose between many different options in our code
  • If elif else chained conditionals
Today’s Plan

• Start discussing sequences in Python
  • Focus on strings today
  • Move on to lists next Lecture
  • Lab 3 covers both!
• Discuss basic strings operators: slicing [: :], indexing [], in
• Learn about simple string methods
• Introduce for loops as a mechanism to iterate over sequences
Cover LOTS of new material today (and Friday)!

DRINKING FROM

THE FIREHOSE

Don’t be afraid to ask for help!
Sequences in Python: Strings

- **Sequences** in Python represent **ordered collections of elements**: e.g., strings, lists, ranges, etc.

- Today we will focus on **strings** (type `str`) which are ordered sequences of individual characters
  - Example: `word = "Hello"
    - 'H' is the first character of word, 'e' is the second character, and so on
  - In CS, we use **zero-indexing**, so we say that 'H' is at index 0, 'e' is at index 1, and so on
  - We can access each character of a string using these **indices**
How Do Indices Work?

• Can access elements of a sequence (such as a string) using its *index*
• Indices in Python are both positive and negative
• Everything outside of these values will cause an *IndexError*.

```
word = "Williams"
```
Accessing Elements of Sequences

>>> word = "Williams"
>>> word[0] # character at 0th index?
'W'

>>> word[3] # character at 3rd index?
'l'

>>> word[7] # character at 7th index?
's'

>>> word[8] # will this work?
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
Length of a Sequence

- Python has a built-in `len()` function that computes the length of a sequence such as a string (or a list, which we will see in next lecture).
- For a string, `len()` simply returns the number of characters.
- Thus, a string `word` has (positive) indices $0, 1, 2, \ldots, \text{len(word)}-1$.

```python
>>> len("Williams")
8
>>> len("pneumonoultramicroscopicsilicovolcanoconiosis")
45
```
Negative Indexing

- Negative indexing starts from -1, and provides a handy way to access the last character of a non-empty sequence without knowing its length.

```python
>>> word = "Williams"
>>> word[-1]
's'
```

Note: Most other languages do not support negative indexing!
Slicing Sequences

- We can **extract subsequences** of a sequence using the **slicing** operator `[:]`

- For a given sequence `var`, `var[start:end:step]` returns a new sequence starting at index ‘start’ (inclusive), ending at index ‘end’ (exclusive), using an increment of ‘step’

- Example: Suppose we want to extract the substring "Williams" from "Williamstown" using slicing operator `[:]`

- Note: Many more examples in Jupyter notebook!

```python
>>> place = "Williamstown"
>>> # return the sequence from 0th index up to 8th
>>> # (not including) 8th
>>> place[0:8:1]
'Williams'
```
Slicing Sequences: Using Step

- The (optional) third `step` parameter to the slicing operator determines in what direction to traverse, and whether to skip any elements while traversing and creating the subsequence.

- By default, `start = 0, end = len(), step = +1` (which means move left to right in increments of one).

- We can use other `step` parameters to obtain new sliced sequences.

```python
>>> place = "Williamstown"
>>> place[:8:1]  # start is 0, end is 8, step is +1
'Williams'
>>> place[:8:2]  # start is 0, end is 8, step is +2
'Wlim'
>>> place[::2]   # start is 0, end is 12, step is +2
'Wlimtwn'
```
Slicing Sequences: Optional Step

• When the step parameter is set to a negative value it gives a nifty way to reverse sequences

• Note: `start` and `end` are interpreted “backwards” when using a negative step!

```python
>>> place = "Williamstown"
>>> place[::-1] # reverse the sequence
'nwotsmailliW'
>>> place[::-2]
'nosali'
>>> place[8:0:-1]
'tsmailli'
```
Testing Membership: \texttt{in} Operator

- The \texttt{in} operator in Python is used to test if a given sequence is a subsequence of another sequence; returns \texttt{True} or \texttt{False}

```python
>>> "Williams" \texttt{in} "Williamstown"
True

>>> "W" \texttt{in} "Williams"
True

>>> "w" \texttt{in} "Williams" \# capitalization matters
False

>>> "liam" \texttt{in} "WiLLiams" \# will this work?
False
```
String Methods: upper(), lower()

• Python provides several convenient **methods** for manipulating **strings**

• Methods are like functions, but are applied to specific variables using **dot notation**: `var.method()` (more info on methods coming soon!)

• Example: The **upper()** and **lower()** string **methods** convert a string to upper or lowercase respectively; these methods **return a new string**

```python
>>> message = "HELLLOOOO...!!!"
>>> message.lower() # leaves non-alphabets the same
'hellooo...!!!'

>>> song = "$$ la la la laaa la $$...
>>> song.upper()
'$$ LA LA LA LAAA LA $$...'
```
isVowel() function

Consider two versions of an isVowel() function that takes a character (a string) as input and returns whether or not it is a vowel

- Ignore case by converting to lowercase using str.lower() method
- Use in operator to simplify code (fewer boolean expressions)

```python
def oldIsVowel(char):
    """Old isVowel function""
    c = char.lower()  # convert to lower case first
    return (c == 'a' or c == 'e' or
            c == 'i' or c == 'o' or c == 'u')

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

- **Problem:** Write a function `countVowels()` that takes a string `word` as input and returns the number of vowels in the string (an int)
- We can use our `isVowel()` function to help us

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

>>> countVowels("Williamstown")
4
>>> countVowels("Ephelia")
4
```
First Attempt with Conditionals

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

• Recall that `val += 1` is shorthand for `val = val + 1`.

• We need something else that allows us to “loop” over the characters in an arbitrary input string.

```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)  # 3
```
For Loops
Iterating with for Loops

- One of the most common ways to manipulate a sequence is to perform some action for each element in the sequence.
- This is called looping or iterating over the elements of a sequence.
- Syntax of a for loop:

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

var is called the loop variable.
seq is a sequence (for example, a string).
Iterating with \texttt{for} Loops

• As the loop executes, the loop variable (\texttt{char} in this example) takes on the value of each of the elements of the sequence one by one

\begin{verbatim}
>>> # simple example of for loop
>>> word = "Williams"

>>> for char in word:
...   print(char)

W
i
l
l
i
a
m
s
\end{verbatim}
Counting Vowels

- We can use a for loop to implement our `countVowels()` function.
- Notice how `count` “accumulates” values in the loop.
- We call `count` an accumulation variable.

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

def countVowels(word):
    '''Returns number of vowels in the word'''
    count = 0
    for char in word:
        if isVowel(char):
            count += 1
    return count

countVowels('Boston')
Counting Vowels: Tracing the Loop

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

def countVowels(word):
    '''Returns number of vowels in the word'''
    count = 0
    for char in word:
        if isVowel(char):
            count += 1
    return count

countVowels('Boston')
Counting Vowels: Tracing the Loop

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

```python
def countVowels(word):
    '''Returns number of vowels in the word'''
    count = 0
    for char in word:
        if isVowel(char):
            count += 1
    return count

countVowels('Boston')
```

- Loop variable
- `word`: 'Boston'
- `count`: 1
- `char`: 'B', 'o', 's', 't', 'o', 'n'
Counting Vowels: Tracing the Loop

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

def countVowels(word):
    '''Returns number of vowels in the word'''
    count = 0
    for char in word:
        if isVowel(char):
            count += 1
    return count

countVowels('Boston')

Counting Vowels: Tracing the Loop

<table>
<thead>
<tr>
<th>word</th>
<th>'Boston'</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>1</td>
</tr>
<tr>
<td>char</td>
<td>'B' 'o' 's' 't' 'o' 'n'</td>
</tr>
</tbody>
</table>
How are the local variables updated as the loop runs?

def countVowels(word):
    '''Returns number of vowels in the word'''
    count = 0
    for char in word:
        if isVowel(char):
            count += 1
    return count

countVowels('Boston')
How are the local variables updated as the loop runs?

def countVowels(word):
    '''Returns number of vowels in the word'''
    count = 0
    for char in word:
        if isVowel(char):
            count += 1
    return count

countVowels('Boston')
Exercise:
Count Characters
Exercise: Count Characters

- Define a function `countChar()` that takes two arguments, a character and a word (both strings), and returns the number of times (int) that character appears in the word (ignoring case).

```python
def countChar(char, word):
    '''Counts # of times char appears in word'''
    pass

>>> countChar('m', "ammonia")
2

>>> countChar('a', "Alabama")
4

>>> countChar('a', "rhythm")
0
```
Exercise: Count Characters

- Define a function `countChar()` that takes two arguments, a character and a word (both strings), and returns the number of times (int) that character appears in the word (ignoring case).

```python
def countChar(char, word):
    '''Counts # of times char appears in word'''
    count = 0  # initialize accumulation var
    for letter in word:  # letter is the loop variable
        if char.lower() == letter.lower():
            count += 1  # increment count (accumulate)
    return count
```
Exercise:
Vowel Sequences
Exercise: Vowel Sequences

- 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:

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

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

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

>>> vowelSeq("rhythm")
''
```

What might be other good values to test edge cases?
Exercise: Vowel Sequences

- 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.
- Accumulation variables don’t have to be counters! Can accumulate strings as well.

```python
def vowelSeq(word):
    '''returns the vowel subsequence in word'''
    vowels = ""  # accumulation variable
    for char in word:  # char is loop variable
        if isVowel(char):  # if char is a vowel
            vowels += char  # accumulate
    return vowels
```
Code from today can be found in sequenceTools.py