CS134:

Sequences and Loops

Announcements & Logistics

- **Homework 3** is out on GLOW, due Monday @ 10 pm
  - Covers materials through last lecture (conditionals)
- **Lab 1** graded feedback will be released at noon today
  - Instructions on how to view feedback on course webpage under Labs
- **Lab 2** due today 10pm / ***tomorrow 9 pm (due to power outage)***
  - Make sure to sign honorcode.txt
- Slight change to Jeannie's office hours today: **1:30-3:30pm**

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** on Friday
- Discuss *slicing* and *indexing* of strings
- Introduce **for loops** as a mechanism to iterate over sequences

Sequences in Python: Strings

- **Sequences** are an abstract type in Python that represent ordered collections of elements: e.g., strings, lists, ranges, etc.
- Today we will focus on **strings** which are an ordered sequence of individual characters (also of type `str`)
    - Consider for example: `word = "Hello"`
        - `'H'` is the first character of word, `'e'` is the second character, and so on
        - In Computer Science, it is convention to use zero-indexing, so we say that `'H'` is the zeroth character of word, `'e'` is the first character, and so on
- We can access each character of a string using **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.

0	1	2	3	4	5	6	7
'W	i	l	l	i	a	m	s'
-8	-7	-6	-5	-4	-3	-2	-1

```
word = 'Williams'
```

Accessing Elements of Sequences

```
In [1]: word = 'Williams'

In [2]: word[0]  # character at 0th index?
Out[2]: 'W'

In [3]: word[3]  # character at 3rd index?
Out[3]: 'l'

In [4]: word[7] # character at 7th index?
Out[4]: 's'

In [5]: word[8] # will this work?
---------------------------------------------------------------------------
IndexError
```

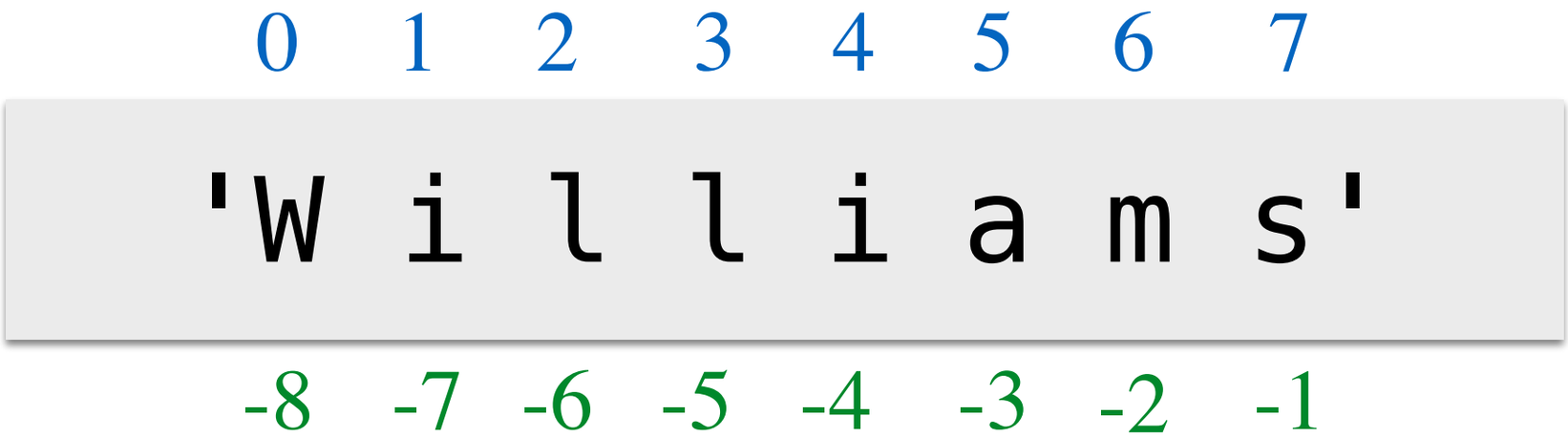
Length of a Sequence

- 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)
- Thus, a string `word` has (positive) indices
  `0, 1, 2, ..., len(word)-1`

```
In [6]: len("Williams")

Out[6]: 8

In [7]: len("pneumonoultramicroscopicsilicovolcanoconiosis")

Out[7]: 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




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

Note: Most other languages do not support negative indexing!

Slicing Sequences

- Python allows us to extract subsequences of a sequence using the slicing operator `[:]`.
  - e.g., suppose we want to extract the substring `'Williams'` from `'Williamstown'`
  - We can use the starting and ending indices of the substring and the slicing operator `[:]`
  - More examples in Jupyter notebook

```
In [15]: place = "Williamstown"

In [19]: # return the sequence from 0th index up to (not including) 8th
         place[0:8]

Out[19]: 'Williams'
```

Slicing Sequences: Optional Step

- The slicing operator `[:]` optionally takes a third step parameter that determines in what direction to traverse, and whether to skip any elements while traversing and creating the subsequence
- By default the step is set to **+1** (which means move left to right in increments of one)
- Default starting index is 0, ending index is end of string
- We can pass other step parameters to obtain new sliced sequences; see examples in Jupyter notebook.

```
In [20]: place = "Williamstown"

In [21]: place[:8:1] # 1 is default
Out[21]: 'Williams'
```

Slicing Sequences: Optional Step

- When the optional step parameter is set to -1 it gives a nifty way to reverse sequences as well

```
In [20]: place = "Williamstown"
```

```
In [22]: place[:8:2] # go left to right in increments of 2
```

```
Out[22]: 'Wlim'
```

```
In [23]: place[::2] # can you guess the answer?
```

```
Out[23]: 'Wlimtw'
```

```
In [24]: place[::-1] # reverse the sequence
```

```
Out[24]: 'nwotsmailliW'
```

Testing Membership: `in` Operator

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

```
In [25]: 'Williams' in 'Williamstown'
```

```
Out[25]: True
```

```
In [26]: 'W' in 'Williams'
```

```
Out[26]: True
```

```
In [27]: 'w' in 'Williams' # capitization matters
```

```
Out[27]: False
```

```
In [28]: 'liam' in 'WiLLiams' # will this work?
```

```
Out[28]: False
```

String Methods: upper(), lower()

- The **`upper()`** and **`lower()`** string methods in Python convert a string to upper or lowercase respectively; returns a new string

```
In [29]: message = "HELLLOOOO...!!!"

In [30]: message.lower() # leaves non-alphabets the same
Out[30]: 'hellloooo...!!!'

In [31]: song = "$$ la la la laaa la $$..."

In [32]: song.upper()
Out[32]: '$$ LA LA LA LAAA LA $$...'
```

New `isVowel()` function

- We can write an improved `isVowel()` function that takes a character as input and returns whether or not it is a vowel
    - Ignore case by converting to lower case
    - Use `in` operator

In [33]:
```
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')
```

In [34]:
```
def isVowel(char):
    """Simpler isVowel function"""
    c = char.lower() # convert to lower case first
    return c in 'aeiou'
```

Iteration Motivation: countVowels

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

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

```
>>> countVowels('Williamstown')
4
>>> countVowels('Ephilia')
4
```

Attempts 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`

In [35]:
```
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
```

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:

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

`for` loop Flow Chart

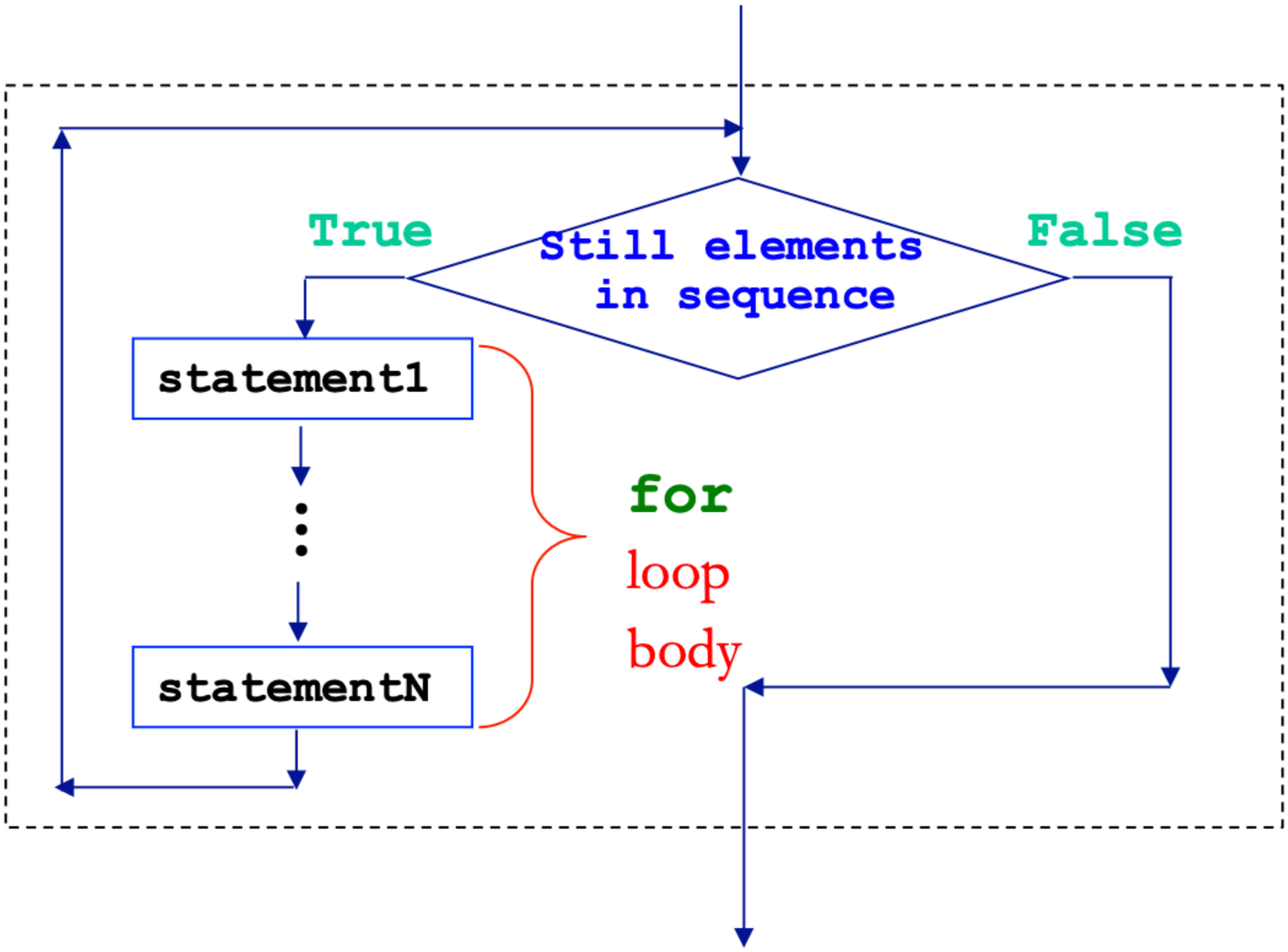


Image Source: (http://cs111.wellesley.edu/spring19)

Iterating with `for` Loops

- The loop variable (char in this example) takes on the value of each of the elements of the sequence one by one

```
In [37]: # simple example of for loop

         word = "Williams"

         for char in word:
             print(char)
```

```
W
i
l
l
i
a
m
s
```

Counting Vowels

- We can now use a for loop to finish implementing our `countVowels()` function

```
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
```

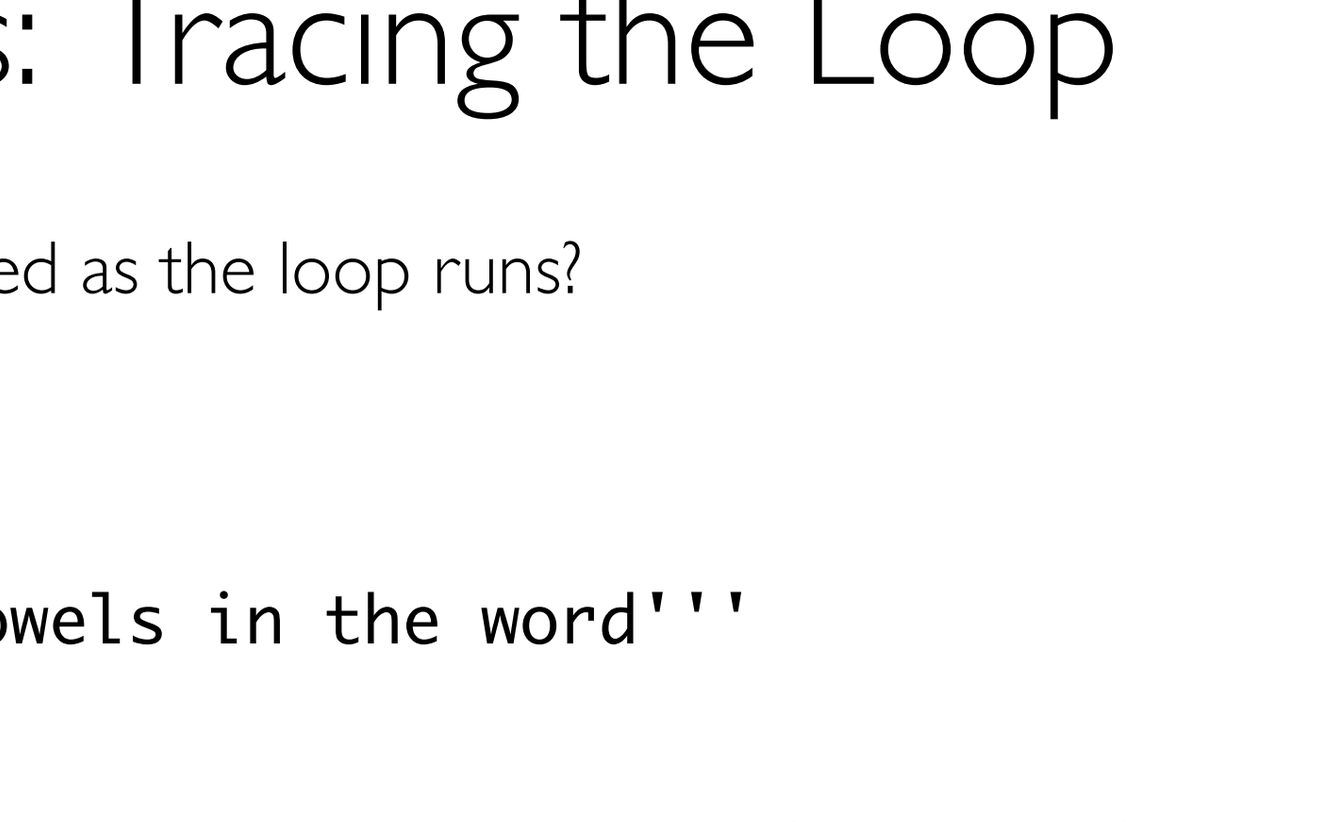
Counting Vowels: Tracing the Loop

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

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

Loop variable

```
countAllVowels('Boston')
```



Exercise: Count Characters

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

```
def countChar(char, word):
    '''Counts # of times a character appears in a 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, and returns the number of times that character appears in the word (ignoring case)

```
def countChar(char, word):
    '''Counts # of times a character appears in a word'''
    count = 0  # initialize count
    for letter in word:
        if char.lower() == letter.lower():
            count += 1 # update count
    return count
```

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.

```
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.

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

More next time!