CS134: Sequences and Loops
Announcements & Logistics

- **Homework 3** is out on GLOW, due next Monday @ 11 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 11pm / tomorrow 11pm
- No class on Friday!
- **Lab 3** starter code pushed on Friday
  - Try to spend 30-60 minutes on it before your scheduled lab
  - Should be able to do #1-3 in Part 1 after today’s class
  - Tuesday late lab starts at 2:35 (not 2:25)

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 Monday
  - Lab 3 covers both!
- 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** (type `str`) which are ordered sequences of individual characters
  - 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 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.

```python
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).
- For a string, `len()` simply returns the number of characters.
- Thus, a string `word` has (positive) indices $0, 1, 2, \ldots, \text{len}(\text{word})-1$.

```
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.

```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
In [1]: place = "Williamstown"

In [2]: # return the sequence from 0th index up to (not including) 8th place[0:8:1]
   place[0:8:1]

Out[2]: '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 pass other `step` parameters to obtain new sliced sequences.

```python
In [3]: place = "Williamstown"

In [4]: place[8:1]  # start is 0, end is 8, step is +1
Out[4]: 'Williams'

In [5]: place[8:2]  # start is 0, end is 8, step is +2
Out[5]: 'Wlim'

In [6]: place[::2]  # start is 0, end is 12, step is +2
Out[6]: 'Wlimtw'
```
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
In [15]: place[:,-1] # reverse the sequence
Out[15]: 'nwotsmailliW'
```

```python
In [16]: place[:,-2]
Out[16]: 'nosali'
```

```python
In [17]: place[8:0:-1]
Out[17]: 'tismailli'
```
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()

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

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

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

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

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

In [34]:
```python
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
```

```python
>>> countVowels('Williamstown')
4
>>> countVowels('Ephelia')
4
```
First 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`

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

```python
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)
```
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)
  ```

  var is called the loop variable.

  seq is a sequence (for example, a string).
Iterating with **for** Loops

- As the loop executes, 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 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
```
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?

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')

Loop variable

word | 'Boston'
count | 1
char | 'B' 'o' 's' 't' 'o' 'n'
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
```

```python
countVowels('Boston')
```

```
word: 'Boston'

char: 'B'  'o'  's'  't'  'o'  'n'

count: 2

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

Counting Vowels: Tracing the Loop

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

```
def countVowels('Boston'):
    word = 'Boston'
    count = 0
    for char in word:
        if isVowel(char):
            count += 1
    return count
```
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(char, word)` 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

• 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")
"
```
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
```