CS134 Lecture:
Sequences and Loops
Announcements & Logistics

• **Homework 3** will be posted to GLOW, due next Monday @ 10 pm

• **Lab 1** graded feedback will be released today
  • Instructions on how to view feedback on course webpage
    • It may seem like an odd procedure, but we're using real-world software development practices

• **Lab 2** due today 10pm / tomorrow 10pm

• No class on Friday: **Winter Carnival**

• Lab 3 (with a [prelab](#)) will be released 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

• Introduce iteration using **for loops** to iterate over **sequences**
• Introduce a new data type which is also a sequence:
  • Lists
• We will discuss sequences more on Monday
Sequences in Python: Strings

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

- **Strings** (type `str`) 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 list) 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
"Williams"
-8 -7 -6 -5 -4 -3 -2 -1
```

**Note:** Most other languages do not support negative indexing!
Accessing Elements of Sequences

```python
>>> 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
```
Sequence Length

- The `len(seq)` function returns the length of the sequence `seq`.
- Even though we zero-index, we still include the total number of elements in the length.

```python
>>> word = "Williams"
>>> len(word)  # total number of characters
8

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

>>> word[len(word)-1]  # what about this?
's'
```
Iteration Motivation: Counting Vowels

- **Problem:** Write a function `count_vowels(word)` that takes a `string` `word` as input and returns the number of vowels in the string (an `int`).
- We'll create a function `is_vowel()` to help us:

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

```python
>>> countVowels("Williamstown")
4

>>> countVowels("Ephelia")
4
```
def is_vowel(char):
    """Takes a char (str) returns True if char is a vowel otherwise False."""

    l_case = (char == 'a' or char == 'e' or char == 'i' or char == 'o' or char == 'u')
    u_case = (char == 'A' or char == 'E' or char == 'I' or char == 'O' or char == 'U')

    return l_case or u_case
First Attempt with Conditionals

• **Note:** counter += 1 is shorthand for
  
  counter = counter + 1

• Any downsides to this approach?

• What if I change word to "Williamstown"?

```python
word = 'Williams'
counter = 0
if is_vowel(word[0]):
    counter += 1
if is_vowel(word[1]):
    counter += 1
if is_vowel(word[2]):
    counter += 1
if is_vowel(word[3]):
    counter += 1
if is_vowel(word[4]):
    counter += 1
if is_vowel(word[5]):
    counter += 1
if is_vowel(word[6]):
    counter += 1
if is_vowel(word[7]):
    counter += 1
print(counter)
```
First Attempt with Conditionals

- Using conditionals as shown is repetitive and does not generalize to arbitrary length words.
- We need something else that allows us to “loop” over the characters in an arbitrary input string.

```python
counter = 0
word = 'Williamstown'
if is_vowel(word[0]):
    counter += 1
if is_vowel(word[1]):
    counter += 1
if is_vowel(word[2]):
    counter += 1
if is_vowel(word[3]):
    counter += 1
if is_vowel(word[4]):
    counter += 1
if is_vowel(word[5]):
    counter += 1
if is_vowel(word[6]):
    counter += 1
if is_vowel(word[7]):
    counter += 1
print(counter)
```
For Loops
Iterating with **for** Loops

- One of the most common ways to traverse or 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

```python
>>> # simple example of for loop
>>> word = "Williams"

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

Williams
```

**Note.** Python for loops are meant *specifically* for iterating over sequences and are also called a "for each" loop.

Why might we call it that?
Counting Vowels

• Let us use a for loop to implement `count_vowels()` function

• What do we need to keep track of as we iterate over `word`?

```python
def count_vowels(word):
    '''Takes word (str) as argument and returns the number of vowels in it (as int)'''
    pass
```
Counting Vowels

• Notice how count “accumulates” values in the loop

• We call count an accumulation variable

```python
def count_vowels(word):
    '''Takes word (str) as argument and returns the number of vowels in it (as int)'''
    count = 0 # initialize counter
    # iterate over word one character at a time
    for char in word:
        if is_vowel(char):
            count += 1 # increment counter
    return count
```
def count_vowels(word):
    '''Takes word (str) as argument and returns the number of vowels in it (as int)'''
    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count

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

def count_vowels(word):
    '''Takes word (str) as argument and returns the number of vowels in it (as int)'''

    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count
def count_vowels(word):
    '''Takes word (str) as argument and returns the number of vowels in it (as int)'''

    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count
def count_vowels(word):
    '''Takes word (str) as argument and returns
    the number of vowels in it (as int)'''

count = 0
for char in word:
    if is_vowel(char):
        count += 1
return count
def count_vowels(word):
    '''Takes word (str) as argument and returns the number of vowels in it (as int)'''
    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count

countVowels('Boston')
def count_vowels(word):
    '''Takes word (str) as argument and returns the number of vowels in it (as int)'''

count = 0
for char in word:
    if is_vowel(char):
        count += 1
return count
Exercise:
Vowel Sequences
Exercise: Vowel Sequences

- Define a function `vowel_seq(word)` that takes a string `word` and returns a string containing all the vowels in `word` in the order they appear.

```python
>>> vowel_seq("Chicago")
'iao'
>>> vowels_seq("protein")
'oei'
>>> vowel_seq("rhythm")
''
```

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

- Accumulation variables don’t have to be counters!
- Can accumulate strings as well: initialize to " instead of zero

```python
def vowel_seq(word):
    '''Takes word (str) as input and returns the vowel subsequence in given word (str)'''
    vowels = ""  # initialize accumulation var
    for char in word:
        if is_vowel(char):  # if vowel
            vowels += char  # accumulate characters
    return vowels
```
Lists
A New Sequence: Lists

- A list is a comma separated, ordered sequence of values.
- These values can be **heterogenous** (strings, ints, floats, etc)
  - Example: `my_list = ['Hello', 42, 23.5, True]`
  - In CS, we use **zero-indexing**, so we say that 'Hello' is at **index 0**, 42 is at **index 1**, and so on
- We can access each element of a list using these **indices**
How Do Indices Work?

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

```python
vowels = ['a', 'e', 'i', 'o', 'u']
```

```
0 1 2 3 4
['a', 'e', 'i', 'o', 'u']
-5 -4 -3 -2 -1
```
Lists

• Lists are:
  
  • **Comma separated, ordered 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!

# Examples of various lists:

```python
>>> wordList = ["What", "a", "beautiful", "day"]
```

```python
>>> numList = [1, 5, 8, 9, 15, 27]
```

```python
>>> charList = ['a', 'e', 'i', 'o', 'u']
```

```python
>>> mixedList = [3.14, 'e', 13, True]
```

```python
>>> type(numList)
list
```

Lists can be heterogeneous (mixed)!
How Do Indices Work?

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

```python
vowels = ['a', 'e', 'i', 'o', 'u']
```
Accessing Elements of Sequences

```python
>>> vowels = ['a', 'e', 'i', 'o', 'u']
>>> vowels[0]  # character at 0th index?
'a'

>>> vowels[3]  # character at 3rd index?
'o'

>>> vowels[4]  # character at 4th index?
'u'

>>> vowels[5]  # will this work?
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```
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
>>> vowels = ['a', 'e', 'i', 'o', 'u']
>>> vowels[-1]
'u'
```

Note: Most other languages do not support negative indexing!
Next time:
Sequence Slicing & Operators