Sequences and Loops
Check-in and Reminders

- Reminder: **pick up Homework 2** from up front, due Monday
- Lab 2 due Wed 11 pm (Mon labs), Thurs 11 pm (Tues labs)
- Can always work on lab machines after 4 pm
- Keep your work consistent with what is on **evolene**
- Always push to **evolene** when done with a work session
- If restarting work on a different machine:
  - If working **on that lab on that machine for the 1st time**: clone the repository just like you would in lab
  - Otherwise, make sure to **Fetch -> Pull** in Atom first!

Do You Have Any Questions?
Leftovers: Simplifying Boolean Expressions

- There are several code patterns involving booleans and conditionals that can be simplified as good coding style

```python
if BE:
    return True
else:
    return False
```

```python
if BE1:
    return BE2
else:
    return False
return BE1 and BE2
```

BE: Boolean expression, e.g. `num % 2 == 0`, `char in word`

Many more examples!
Motivation: Iteration

• Given a word like 'Boston', or 'Williams', how many vowels does it have?

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

    # body ?
```

• Helper function we can use?
Old Friend: isVowel

- Simple predicate to check if a letter is a vowel

```python
def isVowel1(char):
    """determines whether a character is a vowel""
    c = char.lower()
    return (c == 'a' or c == 'e' or c == 'i' or c == 'e' or c == 'o' or c == 'u')

def isVowel2(char):
    """determines whether a character is a vowel""
    # assume c is not an empty string
    c = char.lower()
    return c in 'aeiou'
```

Can we chain and say `c == 'a' or 'e' or 'i' or 'e' or 'u'?`

Built in method to convert char to lower case

Simplified check using in!
Indexing: Accessing Characters

- Can access elements of a sequence (such as a string or list) using its indices.
- Indices start at 0 and go on to \texttt{length(word) - 1}.
- We read \texttt{word[0]} as word sub 0.

\texttt{In [1]: word = 'Boston'}

\texttt{In [2]: word[0]}  
\texttt{Out [2]: 'B'}

\texttt{In [3]: word[1]}  
\texttt{Out [3]: 'o'}

We need to check characters at all indices starting from 0, then 1, 2, ..., up to \texttt{len(word)-1}.
How Do Indices Work?

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

```python
word = 'Boston'
```

```
0  1  2  3  4  5
'B' 'o' 's' 't' 'o' 'n'
-6 -5 -4 -3 -2 -1
```
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.

```python
# Generic form of a for loop
for var in seq:
    # body of loop
    # statements involving var
```

**Note.** *(for loop syntax)* Indentation defines the loop body and colon `:` after name of sequence.
Counting Vowels

• Coming back to our motivating example

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. char above is the loop variable that takes on the values of each character in word
Counting Vowels: Tracing the Loop

- How the local variables are updated as the loop runs

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

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

---

**Loop variable**
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.

```python
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
```
New Sequence: Lists

• A list is a comma separated sequence of values
  
  In [1]: phrase = ['A', 'lovely', 'spring', 'day']
  
  In [2]: type(phrase)
  
  Out [2]: <class 'list'>
  
  In [3]: numseq = [3, 4, 5, 6]
  
  In [4]: alsoAList = ['1', '3', '4', 'CS']
  
  In [5]: list('Shikha')
  
  Out [5]: ['S', 'h', 'i', 'k', 'h', 'a']

• We will study lists in more detail in coming lectures

• Example of 'mutable' objects in Pythons.

• In contrast, strings are immutable
Looping over Lists

• We can loop over lists the same way we loop over strings.
• The loop variable iteratively takes on the values of each item in the list, starting with the 1st item, then 2nd, and finally the last item of the list.
• The following loop iterates over the list, printing each item in it

```python
phrase = ["A", "lovely", "Fall", "day"]
for word in phrase:
    print(word)
```
Exercise: WordStartEnd

• Let's count the number of words in the given list that start and end with the same letter. See Jupyter Notebook for testing this function.

def wordStartEnd(wordList):
    '''Takes a list of words and counts the # of words it that start and end with the same letter'''
    count = 0  #initialize counter
    for word in wordList:
        if len(word):  #why do we need this?
            if word[0].lower() == word[-1].lower():
                # print(word) debugging print here perhaps
                count += 1
    return count
Range Function

• When the `range` function is given two integer arguments, it returns a range object of all integers starting at the first and up to, *but not including*, the second.

• To see the list included in the range, we can pass it to the list function which returns a list of numbers.

• A list is a new Python type: stores a sequence of any values, delimited by square brackets, and separated by commas.

In [1]: range(0, 10)
In [2]: range(0, 10)
Out [2]: list(range(0,3))
In [3]: list(range(3))  #missing first arg defaults to 0
Out [3]: [0,1,2]
Loops to Repeat Tasks

- Sometimes we might use a loop, not to iterate over a sequence but just to repeat a task over and over. The following loops print a pattern to the screen.

```python
def print_pattern():
    for i in range(5):
        print('$' * i)
    for j in range(5):
        print('*' * j)
    for _ in range(10):
        print('Hello World!')
```

Try this out in interactive python! When loop variable is not needed in body, can use _ as variable.
What If We Don’t Know When to Stop?

• Stopping condition of for loop: **no more elements in sequence**

```python
"A", "lovely", "Fall", "day"
```

• What if we don’t know when to stop?

```
Please enter your name: Ted
Hi, Ted
Please enter your name: Marshall
Hi, Marshall
Please enter your name: Lily
Hi, Lily
Please enter your name: quit
Goodbye
```

In this example, we don’t know how many users will be responding. We need to keep asking.
While Loops

- **for** loops iterate over a pre-determined sequence and stop at the end of the sequence.
- **while** loops are useful when we don't know in advance when to stop.
- A **while** loop will keep iterating until the condition in the parenthesis is satisfied and will halt if the condition fails to hold.
- A generic example of a while loop looks like this:

```python
while (continuation condition is true):
    # keep repeating the following
    # statements in loop body
```

**Note.** (while loop syntax) Indentation defines the loop body and colon `:` after continuation condition.
**While Loops**

*While* loops are a fundamental mechanism for expressing iteration. The keyword indicating the while loop is followed by a boolean expression denoting whether to iterate through the body of the loop one more time.

A generic example of a while loop looks like this:

```
while (some condition is true):
    # keep repeating the following statements in loop body
```

**Body** of loop = actions to perform if the continuation condition is true.

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

• Example of a while loop that depends on user input

```python
prompt = 'Please enter your name (type quit to exit): ' 
name = input(prompt)

while (name.lower() != 'quit'):
    print('Hi,', name)
    name = input(prompt)
print('Goodbye')
```

• See notebook for example tests of this piece of code.
While Loop to Print Halves

• Given a number, keep dividing it until it becomes smaller than 0 and print all the “halves”

```python
def printHalves(n):
    while n > 0:
        print(n)
        n = n//2

printHalves(100)
```

Infinite loop! Indentation matters!
Modules and Scripts

- **Script** is generally any piece of code saved in a file, e.g., `phase.py`
- Scripts are meant to be directly executed with: `python3 phase.py`
- A module are generally collection of statements and definitions (a sort of a library) that is meant to be imported and used by a different program
- Within a module, the module’s name is available in a variable called `__name__`
- When a module is executed to be run directed as a script (as opposed to being imported), the `__name__` variable is set to `main`
- Why does this matter? **Importing a module runs it**, and we often want different behavior when the code is run as script vs when its imported as a module
if `__name__` == `__main__`

- We can place code that we want to run when our module is executed as a script inside the `if `__name__` == `__main__`:` block.

- This is usually testing code and we do not want it run when we are importing functions from the file.

- For example, all the definition functions we have design on sequences and loops are now in the file `sequenceTools.py`.

- Notice the code at the bottom of the file under `if `__name__` == `__main__`:` block.
  - This code block will be run when we execute `python3 sequenceTools.py`.
  - This code block will not be run when we import functions from this module.
Testing Functions: Doctests

- Python's `doctest` module allows you to embed test cases and expected output directly into a function's docstring.
- To use the `doctest` module, you must import it using `import doctest`.
- To make sure the test cases are run when the program is run as a script from the terminal, you need to call `doctest.testmod()`.
- To make sure that the tests are not run in an interactive shell or when the functions from the module are imported, you should place the command within a guarded `if `__name__` == "__main__":` block, e.g.

```python
if __name__ == "__main__":
    import doctest
doctest.testmod()
```
Acknowledgments

These slides have been adapted from:

- [http://cs111.wellesley.edu/spring19](http://cs111.wellesley.edu/spring19) and