CSCI 134 Fall 2021:
Lab 3: Building a Python Toolbox
(Word Puzzle Lab)

Jeannie Albrecht, Kelly Shaw, Shikha Singh
Lab 3: Goals

• In this lab, you will accomplish two tasks.

• Construct a toolbox or module of tools for manipulating words and word lists

• Use your toolbox to answer some trivia questions.

• In doing this lab, you will gain experience with the following:
  • Sequences (lists and strings), and associated operators and methods
  • Writing simple and nested loops (will be covered on Monday)
  • Writing doctests to test your functions
  • Creating a module and using the __all__ special variable.
Building a Module

• In this lab, you will build several useful functions on strings, that you will collect in a module `wordTools`
  • Any script (e.g. `wordTools.py`) can serve as a module: the module name does not have the extension `.py`
  • Intention finding building a module is for another user to import the functions in the module to use them
  • The user of the module does not need to know the implementation details of the function
  • User should know what the function does, and what the input/output format is: which is why it is important to document your functions!

• `pydoc3 wordTools` gives an overview of all functions in the module `wordTools`
Importing a Module

• If the variable starts/ends with “__” it is a special variable in Python
  • Where have we already seen this?
• __all__ is another special variable
  • A list of strings of function names (or other public objects) that are intended to be imported when the user types:
    ```python
    from moduleName import *
    ```
• Note: Any specific function/variable/etc. in the module can also be explicitly imported as:
  ```python
  from moduleName import explicitVariableName
  ```
Testing Functions: Doctests

• We have already seen two ways to test a function (what were they??)

• Python's `doctest` module allows you to embed test cases and expected output directly into a function's docstring

• To use the doctest module, we must import it using:
  ```python
  from doctest import testmod
  ```

• To make sure the test cases are run when the program is run as a script from the terminal, we then need to call `testmod()`.

• To ensure that the tests are not run in interactive Python or when the module is imported, we place the command within a guarded if block:
  ```python
  if __name__ == '__main__':
  ```
def isVowel(char):
    """Takes a letter as input and returns true if and only if it is a vowel."
    >>> isVowel('e')
    True
    >>> isVowel('U')
    True
    >>> isVowel('t')
    False
    >>> isVowel('Z')
    False
    """
    return char.lower() in 'aeiou'

if __name__ == '__main__':
    # the following code tests the tests in the docstrings ('doctests').
    # as you add tests, re-run this as a script to test your work
    from doctest import testmod
    # this import is necessary when testing
    testmod()  # test this module, according to the doctests

Run the doctests only when file is executed as a script
Look at the Starter Code
Review: `wordStartEnd`

- Write a function that iterates over a given list of words `wordList`, returns a (new) list containing all the words in `wordList` that start and end with the same letter (ignoring case).

```python
def wordStartEnd(wordList):
    ''' Takes a list of words wordList and returns a list of all words in wordList that start and end with the same letter'''
    pass
```

```python
>>> wordStartEnd([ 'Anna', 'banana', 'salad', 'Rigor', 'tacit', 'hope' ])
[ 'Anna', 'Rigor', 'tacit' ]
>>> wordStartEnd([ 'New York', 'Tokyo', 'Paris' ])
[]
>>> wordStartEnd([ '*Hello*', '', 'nope' ])
[ '*Hello*' ]
```
Exercise: `wordStartEnd`

- **Step by step approach (organize your work):**
  - Go through every word in `wordList`
  - Check if **word starts and ends at same letter**
  - If true, we need to “collect” this word (remember it for later!)
    - Else, just go on to next word
  - Takeaway: need a new list to **accumulate** desirable words

- **Break down bigger steps (decomposition!)**
  - If word starts and ends at same letter:
    - Can do this using string **indexing**
    - Think about **corner cases:** what if string is empty? what about case?
• Write a function that iterates over a given list of words `wordList`, returns a (new) list containing all the words in `wordList` that start and end with the same letter (ignoring case).

```python
def wordStartEnd(wordList):
    ''' Takes a list of words and returns a list of words in it that start and end with the same letter'''
    # initialize accumulation variable (of type list)
    result = []
    for word in wordList:  # iterate over list

        # check for empty strings before indexing
        if len(word) != 0:
            if word[0].lower() == word[-1].lower():
                result += [word]  # concatenate to resulting list
    return result  # notice the indentation of return
```