Lab Overview

In this lab, you will write a series of short Python programs to manipulate strings, read input from the user, and display output. Start by making a folder for Lab 4 in your Dropbox where you keep your Computer Science 1 material, then start working on the following three checkpoints.

Checkpoint 1: Functions and Framing Four-Letter Words

Create a new program, check1.py, and open it in the WingIDE. There are two parts to this checkpoint:

- Write a function called framed() that creates a text frame around a given word. This function should accept a single string as an argument. Add code to call your function and test it, e.g.

  \[
  \text{framed( 'CSCI' )}
  \]

  or

  \[
  \text{framed( 'darn' )}
  \]

- Next, add code to use the raw_input() function to read a four-letter word into a string. Verify that the string contains exactly four characters, then pass this string to the framed() function you just wrote. The output when you run your program should look like this:

  \[
  \text{Enter a four-letter word: CSCI}
  \]
  \[
  \text{**********}
  \]
  \[
  \text{** CSCI **}
  \]
  \[
  \text{*******}
  \]

  When the user enters invalid input, your program should look like this:

  \[
  \text{Enter a four-letter word: bananas}
  \text{ERROR: 'bananas' is not a four-letter word.}
  \]

When you have this working, show it to the TA or a mentor. Make sure your function follows the program structure we discussed and required in the homeworks. Congratulations, you have completed Checkpoint 1.
Checkpoint 2: Framing Other Words

Be sure you save check1.py and make a copy of it called check2.py. You will modify this for Checkpoint 2.

Requiring the user to type a word that is exactly four letters is limiting, so let’s expand your program to accept strings of different lengths. More specifically, allow the user to enter a string containing at least four characters and at most twelve characters (otherwise, display the error message shown below).

Also, have the user input the character to use for the border. If the user enters multiple characters, use the first character entered (and display a warning message). Example program runs are shown below:

Enter a word: CSCI
Enter border character: *
**********
** CSCI **
**********

or

Enter a word: fiddlesticks
Enter border character: #
#################################
## fiddlesticks ##
#################################

or

Enter a word: huh
ERROR: 'huh' is too short.

or

Enter a word: arghhhhhhhhhhhhhhhhh
ERROR: 'arghhhhhhhhhhhhhhhh' is too long.

or

Enter a word: fiddlesticks
Enter border character: $*
WARNING: '$*#' is too long, using '$' for border
$$$$$$$$$$$$$$$$$$$$$ $ fiddlesticks $$ $$$$$$$$$$$$$$$$$$ 

When you have this working, show it to a TA or mentor. Congratulations, you have completed Checkpoint 2.
Checkpoint 3: Letter Counting

In this last checkpoint, you will write a new program that determines the number of occurrences of a given letter in a string. To do so, you are only allowed to use the `replace()` and `len()` functions.

Example program runs are shown below:

Enter a sentence: the quick brown fox jumped over the lazy dog
Enter character: t
The letter 't' appears in the sentence exactly 2 times

or

Enter a sentence: the quick brown fox jumped over the lazy dog
Enter character: x
The letter 'x' appears in the sentence exactly 1 time

or

Enter a sentence: the quick brown fox jumped over the lazy dog
Enter character: s
The letter 's' appears in the sentence exactly 0 times

or

Enter a sentence: the quick brown fox jumped over the lazy dog
Enter character: ?
The letter '?' appears in the sentence exactly 0 times

When you have your program fully working, save your code, and then show the result to a mentor or a TA. Congratulations, you have finished Checkpoint 3 and are all done with Lab 4.

Incremental Development and Testing

At several points during the lab, we asked you to write and test code that only completed part of the requirements for a checkpoint. You should adopt this approach of “incremental” development and testing in all of your programming. Do not expect to be able to write a complete working program from scratch without testing parts of it first.