CSCE 1035 Lab 05 Fall 2021

1. zyBooks Labs

Please follow the link on Canvas to complete the following zyBooks labs:

- 5.14 *LAB: Convert to binary*
- 5.15 *LAB: Count input length without spaces, periods, or commas*
- 5.16 *LAB: Password modifier*
- 5.17 *LAB: Output range with increment of 5*
- *5.18 LAB: Print string in reverse*
- *5.21 LAB: Smallest and largest numbers in a list*
- *5.24 LAB: Warm up: Drawing a right triangle*

This portion of the lab will be worth one-third of your Lab 05 grade.

2. Encipher a Plaintext Message Using the Rail-Fence Cipher

This exercise will introduce the user to a simple transposition cipher called the rail-fence, or zig-zag, cipher. To encipher a plaintext (i.e., readable) message using the rail-fence cipher, you can write your message in zig-zag lines across the page, and then read off each row in the following manner:

- Plaintext message: `hello world`
- Written in zig-zag form across two lines:

```
h l o o l
 e l w r d
```

Then, reading the first line followed by the second line, we have our ciphertext (now unreadable message): `HLOOLELWRD`

Notice that we cleaned things up by removing the spaces and changed all of the letters to capitals.

Write a small, but complete Python 3 program called **Lab5A.py** that enciphers a string entered by the user using the rail-fence cipher as follows:

a. Prompt the user for and read in a message that may include spaces (>1 character) to be enciphered. You may assume the user enters more than one character.

b. Strings support a method called `upper()` that converts all of the letters to uppercase. Use `upper()` to convert all the letters in the message to uppercase.

c. Strings also contain a method called `replace()` that can be used to substitute or replace characters in a string with other characters. To remove spaces, we can use the `replace()` method to replace a string with a single space `" "` with an empty string `""`.

d. Initialize two strings, called `top` and `bottom`, to an empty string `""`. These strings will eventually hold the top and bottom rows of the rail-fence cipher.

e. Initialize a counter to 0 that will be used to keep track of the position of the current character from the original plaintext message.

f. Now, use a `for` loop to iterate through all the characters in the plaintext message. Inside the `for` loop, you will:

- Check to see if the counter is an even or odd number, which can be done using the modulus operator `%` and comparing the result to 0 (for an even number). If the counter is even, concatenate the `top` string with the current character and assign it to the `top` string. Similarly, if the counter is odd, concatenate the `bottom` string with the current character and assign it to the `bottom` string (remember, strings are immutable, so we have to essentially redefine the strings).
- Inside the `for` loop, but not inside the `if-else` statement, increment the counter by one.

g. Outside the loop, concatenate the `top` and `bottom` strings and assign to a new string called `cipher`.

h. Finally, print out the enciphered rail-fence string!

For example, the output might look like this (input shown in **bold**):

```
$ python3 Lab5A.py
Type a message to encode: We are discovered, flee at once!
WAEICVRDFEAOC!ERDSOEE,LETNE
```

Note that you will submit this file to Canvas.

3. Looping Through the Hi/Lo Game

In this lab component, you will implement a simple Hi/Lo guessing game. Your program will randomly pick a secret number between 1 and 50, inclusively, while the user will be allowed 5 chances to guess the number. Your program will provide feedback on each guess.

Write a small, but complete Python3 program called **Lab5B.py** that implements the Hi/Lo guessing game as follows:

a. Generate a random number between 1 and 50, inclusively using the `randint()` function in the `random` module.

b. Print an introductory message about the game (see the sample output).

c. Initialize a counter to 1 that will keep track of the number of guesses.

d. Use a `while` loop that will iterate 5 times using the counter as the control variable. Inside the `while` loop, you will:

- Prompt for and read in the user's guess as an integer, providing information about which guess the user is on.
- If the guess is too low, print out a hint that the guess is too low and to try a higher number. Else if the guess is too high, print out a hint that the guess is

too high and to try a lower number. Otherwise, print a congratulatory message with the number of guesses it took.

- Increment the counter by 1.

e. Outside the loop, if the counter is equal to 6 (i.e., beyond the maximum number of guesses) and the guess is not equal to the secret number, print out a message saying that you have run out of guesses, revealing the secret number.

For example, the output might look like this (input shown in **bold**):

```
$ python3 Lab5B.py
Try to guess the number that I am thinking of between 1 and 50.
You have 5 chances to guess the number!
Guess #1: 25
Too high. Try a lower number.
Guess #2: 12
Too high. Try a lower number.
Guess #3: 6
Too high. Try a lower number.
Guess #4: 3
Too high. Try a lower number.
Guess #5: 1
Too low. Try a higher number.
Sorry, you have run out of guesses. The secret number was 2
$ python3 Lab5B.py
Try to guess the number that I am thinking of between 1 and 50.
You have 5 chances to guess the number!
Guess #1: 22
Too high. Try a lower number.
Guess #2: 14
Too low. Try a higher number.
Guess #3: 17
Too low. Try a higher number.
Guess #4: 19
Too low. Try a higher number.
Guess #5: 20
Nice job! You guessed my secret number in 5 guesses!
```

Note that you will submit this file to Canvas.

Now that you have completed this lab, it's time to turn in your results. Once you've moved the files to your windows machine (using **WinSCP**), you may use the browser to submit them to Canvas for the **Lab 05** dropbox.

You should submit the following files:

- **Lab5A.py**
- **Lab5B.py**
- **(Note that the zyBooks labs are submitted separately through Canvas.)**

Ask your TA to check your results before submission.

Now that you've finished the lab, use any additional time to practice writing simple programs out of the textbook, lectures, or even ones you come up with on your own to gain some more experience.