CS 153 and 453
Lab 4 - Loops

Python For Loop

A for loop is one method of repeating a statement or block of statements. The for loop structure looks like this

```
for target in object :

   statements               # body of the loop

else:                       # optional else part

  statements                # executed if the loop didn't hit a break
```

Python assigns the items in the iterable object to target, one by one, and executes the loop body for each item.

The optional else block is executed if the loop exits without running into a break statement.

The object can be any sequence object. Remember that a sequence type is any ordered group of items. Strings, lists, and tuples are sequences.

Examples:

```
# iterate over all characters in a string
for ch in "hello":
   print (ch, end = ' ')
print()
print()

# iterate over all items in a list
for item in [2, 8, 10, 14]:
   print (item)
print()

# iterate over all values in a tuple
for value in ("cat", "dog", "rabbit"):
   print(len(value))
```

The object can also be the result of calling a function, such as the range function. The range function is unique, though, because it generates the items of a list on demand. It doesn't build the entire list in memory all at once.

Python While Loop

The structure of a while loop is

```
initialization

while test:

   statements

else:                 # optional else block

   statements         # executed if the loop didn't hit a break
```

Every for loop has an equivalent while loop.

Nested Loops

A loop can be nested within another loop. Nested means that one loop is placed inside another loop. It's possible for any combination of while loops and for loops to be nested.

Nested loops are very common in two-dimensional problems. Here is a short example with two nested for loops:

```
for i in range(1, 4):
   for k in range(1, 3):
         print(i * k, end=" ");
   print( )
```

The outer for loop (controlled by i) repeats 3 times. Every time the outer loop repeats, the inner loop (controlled by k) repeats 2 times. What is the output of this code segment?

Output
1 2
2 4
3 6

There are 7 problems in this assignment. All of the code for the 7 problems goes into one file.

- Put a header with comments just like you did in the previous lab assignments
- Print the problem number before you write the code for a problem. Print a blank line before and a blank line after the output of each problem.
- Put the code for the problems in order.
- Put a comment before each problem's code with a brief description of problem.
- Any time you get input from the user, you must print a meaningful prompt.

Problem 1 - output the lowercase letters

Two functions that may help you with this problem:

```
ord ( character ) - returns the ASCII number of a character

chr ( number ) - returns the character given by the ASCII number
```

Write a **for** loop to print the lowercase letters 'a' through 'z', one character per line.

There is no input from the user on this problem.

Problem 2 - triangle of asterisks

Input a number from the user. Then, write **nested for loops** to display a triangle made of asterisks. The number the user types determines the number of lines in the triangle. For example, if the user typed 5, the triangle should look like:

```
*****
 ****
  ***
   **
    *
```

The minimum number that the user can enter is 1. The maximum number is 50. Put an if statement to check for this. If the user enters a number less than 1 or greater than 50, print an error message "Invalid input". Don't print any asterisks if the input is invalid.

Problem 3 - prime test

Input an integer from the user and use a while loop to ensure that the user has entered a positive number greater than 1.

Then, use another while loop with an else block to determine whether the number is prime or not and print an appropriate message.

Problem 4 - Counting vowels

Input a string from the user. Convert the string to lowercase.

Write a for loop that counts how many of each vowel (a, e, i, o, u) are found in the string. After the loop is over, display the number of times each vowel was found.

Example Output:

```
# of a's:   3
# of e's:  10
# of i's:   5
.
.
```

The numbers must be right aligned.

Problem 5 - simple substitution cipher

A substitution cipher is a way of encrypting text by replacing every character in a message or document with a different character. Most commonly, each letter is replaced by another letter of the alphabet.

Input a string from the user. (This is the plaintext.) You may assume that all characters in the plaintext are either letters or spaces. Spaces are NOT replaced.

Create a new string in which every letter in the plaintext string has been replaced by the letter 4 characters forward in the alphabet. (The new string is the ciphertext.) Hint: Use modulus to assist with wrapping from 'z' to 'a'.

Examples:

'a' would be replaced by 'e'
'z' would be replaced by 'd'
'm' would be replaced by 'q'

Output the ciphertext string.

CS 453 students must make the following modification to problem 5.

Before inputting the string, input an integer from the user. This integer will be used as the number of letters to shift forward. (In problem 5 above, the value was fixed at 4.)

Use a loop to ensure that the integer is a number from 1 to 25, inclusive.

Replace each letter in the plaintext string by shifting forward the number of letters of the alphabet based on this integer.

```
Example:
Enter the shift value:    2
Enter the plain text:     hello
The ciphertext is:        jgnnq
```

Problem 6 - table of squares

Using two **nested for loops**, display a table of numbers and their squares from 1 to 80 in a table like this:

```
Num Square Num Square Num Square Num Square
  1      1  21    441  41   1681  61   3721
  2      4  22    484  42   1764  62   3844
.
.
.
```

The numbers must be right-aligned in each column.

Problem 7 - inputting words and sorting them

Python has a list data type. A list is a sequence of items enclosed in square brackets. For example,

```
numbers = [2, 6, 4, 3]
```

There is a sort method that can be used on lists. We could sort the above list.

```
numbers.sort( )
```

After sorting, the list would contain [2, 3, 4, 6].

Use a sentinel loop to input words from the user. An inner while loop must be used to verify that each input string is only one word.

Add the words to a list. Stop the sentinel loop when the word "quit" (case doesn't matter) is encountered.

After the sentinel loop ends, sort the list.

Finally, using a **for** loop, print the sorted list, one word per line.

Submit one Python program with the solutions to all 7 problems.