CS 2316

Individual Homework – Enigma Fun

Due: Thursday May 22nd, before 11:55 PM

Out of 100 points

Files to submit: **HW3.py**

For Help:

- TA Helpdesk Schedule posted on class website
- Use Piazzza or e-mail TA's

Notes:

- **Don't forget to include the required comments and collaboration statement (as outlined on the course syllabus).**
- **Do not wait until the last minute** to do this assignment in case you run into problems

User Interaction

You will write a few python functions for practice with user interaction and string and list processing. In your HW3.py file, include a comment at the top with your names, section, GTID/Emails, and your collaboration statement. Also include each of the following functions:

1. getNumber()
2. getYesNo()
3. getDirection()
4. doShift()
5. doEncryption()
6. encryptor()

Function Name: **getNumber**

Parameters:

- `promptMessage` — The message that you will display to the user when asking the user for input

Return Value:

- An integer.

Description:

This function should request that the user enter a number using the message passed into the function through the parameter `promptMessage`. Once the user enters the a value, you should ensure three things:

1. The user entered an integer.
2. The user entered a number greater than or equal to zero.
3. The user entered a number less than 26.

If all three conditions are satisfied, you should return the integer that the user entered (as an integer). If one or more of the conditions are not satisfied, you should continue to ask the user for input using `promptMessage` until he or she enters an integer that satisfies the three conditions, at which point you should return that integer.

Test Cases:

- `getNumber("Enter a number:")`⇒

```
Enter a number: 15
```

  15 is returned.

- `getNumber("Enter a valid integer:")`⇒

```
Enter a valid integer: blah
Enter a valid integer: -4
Enter a valid integer: 50
Enter a valid integer: 5
```

  5 is returned.

Function Name: **getYesNo**

Parameters:

- `promptMessage` — The message that you will display to the user when asking the user for input

Return Value:

- A boolean:
  - True if the user entered Y/yes/Yes/YES/y etc...
  - False if the user entered N/NO/No/nO/n, etc...

Description:

Write a function that takes in a string as a prompt to the user for input. It should display that string and get input from the user. If the user enters Y, or y, or YES, or Yes, or yeS, or YeS or yEs, or any other permutation of lower and upper case letters that spells out a yes, return true. If they enter N, or n, or NO, no, or any permutation of lower and upper case letters that spells out a no, return False. If they enter anything else, give them the error "Not a valid answer, enter Y or N." and then repeat the question until they give a valid answer.

Your function should be able to tolerate whitespace at the beginning end of a line; any number of tabs, spaces, or other whitespace character may be present at the beginning or end of the line; this should not affect whether your function works.

Test Cases:

- `getYesNo("Do you want to continue?")`⇒

```
Do you want to continue? I don't know
Not a valid answer, enter Y or N.
Do you want to continue? Yes
```

  True is returned.

- `getYesNo("Encrypt again?")`⇒

```
Encrypt again? nO
```

  False is returned.

Function Name: **getDirection**

Parameters:

- `promptMessage` — The message that you will display to the user when asking the user for input

Return Value:

- A string, either `"->"` or `"<-"`

Description:

Write a function that takes in a string as a prompt to the user for input. It should display that string and get input from the user. If the user enters ->(a right arrow) or <- (a left arrow), you should return the string containing the arrow (i.e. "->" or "<-"). If the user enters anything other than one of the two arrows, you should display, "Not a valid direction." and prompt the user for input again using `promptMessage`, continuing to ask for a valid input until one of the two arrows is entered.

Your function should be able to tolerate whitespace at the beginning end of a line; any number of tabs, spaces, or other whitespace character may be present at the beginning or end of the line; this should not affect whether your function works.

Test Cases:

- `getDirection("Shift direction?  ")`⇒

```
Shift direction? left
Not a valid direction.
Shift direction? <-
```

  `"<-"` *is returned.*

- `getDirection("Enter the shift direction:  ")`⇒

```
Enter the shift direction: ->
```

  `"->"` *is returned.*

Function Name: **doShift**

Parameters:

- `character` — The character you want to shift in a particular direction.
- `direction` — A string, either `"<-"` or `"->"` which indicates whether you need to shift the character forward or backward.
- `shiftAmt` — The number of characters to shift `"character"` by, either left or right.

Return Value:

- The character after it has had the shift applied to it.

Description:

The doShift function takes in one individual lowercase character and then will convert it into the character that is either `shiftAmt` number of characters to the right or left (What we are trying to implement here is a Caesar Cipher). So, if we shift the character ‘a’ right 3 characters, the resulting value is ‘d’. If we shift the character ‘d’ left 3 characters, we would wind up with ‘a’ again (Note that a character shift is reversable if you shift the same number of characters in the opposite direction).

But what happens when we shift a character like ‘z’ to the right? Or the character ‘a’ to the left? We are forced to wrap-around to the other end of the alphabet and continue moving in the original direction of the shift. So, the character ‘z’ shifted right 1 character would result in the character ‘a’; ‘z’ shifted right 2 would be the character ‘b’ and so on. So, how do we go about doing this?

One way is to use `ascii_lowercase` and string indexing. As you’ve learned in class and on HW2, `ascii_lowercase` is a string containing all the lowercase letters of the alphabet. In this string, the letter ‘a’ is index 0, ‘b’ is index 1, all the way through ‘z’ being index 25. Therefore, there are 26 letters in the string. For a right shift, we know that if the index of the character we have in the `ascii_lowercase` plus the offset is greater than 25, we need to wrap around. To perform this wrap around, all we need to do is add the offset to the index of the original character and then subtract the total number of characters in `ascii_lowercase`; this will give you the index of the character you should return.

For a left shift, you should simply perform the normal subtraction between the index and the shift amount, then add the number of characters in `ascii_lowercase`. This will give you the index of the character you should return.

Test Cases:

- `doShift('d', "->", 5)`⇒ *`'i'` is returned.*
- `doShift('i', "<-", 5)`⇒ *`'d'` is returned.*

- `doShift('a', "<-", 10)`⇒ *`'q'` is returned.*
- `doShift('v', "->", 13)`⇒ *`'i'` is returned.*
- `doShift('a', "->", 25)`⇒ *`'z'` is returned.*

Function Name: **doEncryption**

Parameters:

- None

Return Value:

- A string which represents the message the user inputs which has had each character shifted in the string shifted by the shift amount the user enters.

Description:

Write a function that will ask the user to enter a string made solely of lowercase letters (no spaces, punctuation, etc). Then, using your getDirection function that you wrote earlier, ask the user to enter a direction. Then, using your getNumber function you wrote earlier, ask the user to enter a shift amount. Once you have stored all this information from the user, use your doShift function to encode each character in the string. Once you have encoded the entire message, return the encoded message. **You must call and use the output from the functions you wrote earlier to receive full credit.**

Test Cases:

- `doEncryption()`⇒

```
    Enter a valid string: asdf
    Enter a direction: ->
    Enter an integer: 7
```

  `"hzkm"` is returned.

- `doEncryption()`⇒

```
    Enter a valid string: cstwentythreesixteen
    Enter a direction: <-
    Enter an integer: 10
```

  `"sijmudjojxhuuiynjuud"` is returned.

Function Name: **encryptor**

Parameters:

- None

Return Value:

- None

Description:

Write a function that uses your other functions to allow the user to do multiple encryptions. Your function must deal properly with invalid inputs, which is most easily done by making use of the previous functions you have already written. After each calculation, ask the user if they want to do another encryption. Be sure to save the results of each calculation as they go!

Once the user says they do not want to do any more encryptions, print a final message that states the total number of encryptions they performed, and lists the results (final answer) of each encryption (each encryption on its own line) before saying "Goodbye!"

Test Cases:

- `encryptor()`⇒

```
Enter a valid string: asdf
Enter a direction: <-
Enter an integer: 8
Result is skvx
Encrypt another? yes

Enter a valid string: cstwentythreesixteen
Enter a direction: ->
Enter an integer: 5
Result is hxybjsydymwjjxncyjjs
Encrypt another? no

Your 2 Encrypted lines:
skvx
hxybjsydymwjjxncyjjs

Goodbye!
```

Grading:

You will earn points as follows for each function that works correctly according to the specifications.

getNumber		**20**
Properly shows prompt	5	
Properly returns answer	5	
Properly handles invalid input – “ten”	5	
Ensures number is between 0 and 25	5	
getYesNo		**20**
Properly shows prompt	3	
Properly handles leading spaces	3	
Properly handles trailing spaces	3	
Properly handles invalid input	11	
getDirection		**10**
Properly shows prompt	2	
Properly handles leading spaces	1	
Properly handles trailing spaces	1	
Properly handles invalid input	6	
doShift		**10**
Properly shifts characters up	3	
Properly shifts characters down	2	
Properly implements wraparound	5	
doEncryption		**20**
Uses previously written functions	5	
Does correct shift	5	
Returns the correct string	10	
encryptor		**20**
Allows multiple encryptions	5	
Correctly allows user to quit	3	
Prints correct messages	2	
Remembers/prints number and value of results	10	