Homework 2:
A Dog, a Panic, in a Pagoda

Due date: March 8 at 3:30pm.

You will not demo this homework to a lab leader; instead, you will upload it to Dropbox when you are *complete*, **even if it is after the due date**. You must also turn in your handwritten answers to the **Discussion Questions** (at the end of this document) before the start of lecture on **March 8**. You can (and should) turn in the Discussion Questions even if you aren't finished with the code.

Overview

You will implement a program to identify **palindromes**. Recall that a palindrome is a phrase that – ignoring casing and punctuation – is spelled the same forwards as backwards. An example of a palindrome is the title of this lab. You will also implement a function to **construct** a palindrome.

Functions

You will write the following functions:

1. `def print_menu()`: prints the main menu to the screen.
   (a) Function behavior:
      i. Prints a menu with three options: "Check a palindrome"; "Make a palindrome"; and "Quit".
   (b) Restrictions:
      i. Do **not** use `input` to read a menu choice from the user.
2. `def get_menu_choice()`:
   (a) Function behavior:
      i. Read the user's selected menu option using `input()`, and validate it[1].
      ii. The validated selected option is **returned**.
   (b) Restrictions:
      i. No other functions are called from `get_menu_choice`.
3. `def get_phrase()`:
   (a) Function behavior:
      i. Ask the user to input an English phrase as a string.
      ii. Validate that at least one character is entered.
      iii. Return the validated phrase.
   (b) Restrictions:
      i. No other functions are called from `get_phrase`.
4. `def is_palindrome(phrase)`: given a string named `phrase`, this function returns `True` if the string is a palindrome, and `False` otherwise.
   (a) Function behavior:
      i. Given `phrase`, you must determine if the string is identical forwards and backwards, *if we ignore upper/lowercase and any non-letter characters* like spaces and punctuation.
      ii. Adapt the loop from lecture, which assumes that the entered string is lower-case and does **not** contain non-letter characters.

[1] From now on, *validate* always implies a loop that repeatedly asks for input until the input is valid.

iii. In each iteration of the loop you are adapting, you will need to write two more loops. The first loop should increment `i` as long as index `i` in `phrase` is not a letter; we can determine if a single character is a letter by taking that letter and calling `.isalpha()` on it, which will be `True` only if the character is a letter.

iv. The second loop should decrement (decrease) `j` as long as index `j` in `phrase` is not a letter.

v. Once the two loops have finished, you know that index `i` is a letter and index `j` is a letter. You can now compare them and perhaps make a decision, or perhaps not.

vi. Once you are sure the string is not a palindrome, you should return `False`. Once you are sure the string *is* a palindrome, return `True`.

(b) Restrictions:

i. You cannot use any function that removes or replaces characters from a string. We haven't learned these yet; do not use solutions from your friends or the Web, because they probably break this rule.

ii. You cannot remove or replace characters from the phrase by hand, without using a function.

iii. You cannot use any function that reverses a string for you.

iv. You cannot reverse the string without using a function.

v. **In general**, you can only make one "pass" through the string. This means that once you examine a particular character in the string, you can never examine that character again. All the restrictions above would violate this rule if ignored.

Example: **if** we wrote a loop to go through the phrase and remove any non-letter character, and then go through the modified phrase to see if it is a palindrome, we would be making **two** passes through the string... so we can't do that.

vi. You should not *assume* that `phrase` is all lowercase. You can use the `.lower()` function to create a copy of `phrase` that is converted to all lowercase.

5. `def menu_check_palindrome()`: this function implements Menu Option 1, "check a phrase to see if it is a palindrome."

(a) Function behavior:

i. Call `get_phrase` to read the user's selected phrase.

ii. Call `is_palindrome` with the phrase you saved from `get_phrase`, and save the resulting Boolean value.

iii. Use the Boolean value from `is_palindrome` to print either:

A. "X is a palindome!"; or

B. "X is not a palindrome"

where X is the phrase entered by the user.

(b) Restrictions:

i. Do not use `input` in this function.

ii. Do not perform any of the logic to detect a palindrome in this function.

iii. Do not convert the phrase to lowercase or remove any characters from it.

iv. When you print the final response message, the "X" must appear **exactly as the user entered it**.

6. `def make_palindrome(phrase, skip_last)`: given a string named `phrase` and a Boolean named `skip_last`, this function constructs a new palindrome by reversing `phrase` and adding it to itself.

(a) Function behavior:

i. Using a loop, iterate backwards through `phrase`, appending one letter at a time to the end of the original value of `phrase`.

ii. If `skip_last` is true, then you must start your loop with the **second to last** character, instead of the last character.

iii. Return the final string after appending the additional letters.

iv. Example: given a `phrase` of "`warsaw`" and a `skip_last` of `False`, you would create and return "`warsawwasraw`"; given "`lonelyt`" and `True`, you return "`lonelytylenol`".

(b) Restrictions:

i. Same as `is_palindome`.

ii. Do not insert spaces, commas, or any other character into the new palindrome string; only use the exact characters from the given phrase.

7. `def get_repeat_last()`: ask the user if they want the last letter of their phrase repeated when making a new palindrome.

(a) Function behavior:

i. Ask the user if they would like the last letter of their phrase repeated.

ii. Validate that they enter either "`y`" or "`n`".

iii. If "`y`" is entered, return `True`; if "`n`" is entered, return `False`.

(b) Restrictions:

i. No other functions are called from `get_repeat_last`.

8. `def menu_make_palindrome()`: this function implements Menu Option 2, "create a new palindrome."

(a) Function behavior:

i. Call `get_phrase` to read the user's selected phrase.

ii. Call `get_repeat_last` to learn if the user wants the last letter of their phrase repeated.

iii. Call `make_palindrome` with the entered phrase. `make_palindrome` also needs a parameter `skip_last`, which you must note is the **opposite** of "repeat the last letter?".

iv. Print the original phrase, and the resulting palindrome phrase to the user.

(b) Restrictions:

i. Same as `menu_check_palindrome`.

9. `def main()`: implement the "main" of this program.

(a) Function behavior:

i. Display the menu.

ii. Read the user's menu choice.

iii. Execute the function corresponding to the menu choice.

iv. Repeat until option 3 (Quit) is selected – **including** printing the menu on each loop.

(b) Restrictions:

i. `main()` cannot call `main()`, and neither can any of your other functions. Use a loop.

Program Flow

1. Display the main menu, allowing the user to select among three operations. Read an integer from the user indicating the desired operation. Verify that the operation is a valid value. If the user does not input a valid selection, print a message and try again.

2. Use `if` and `elif` statements to select the correct operation according to the user's request. Each operation will be implemented as a function, and you will call the appropriate function depending on the selected menu option.

3. Request and read any additional parameters needed for the operation. **Validate all user input** according to the notes in the Operations section below, looping until a value in the correct range is entered. You may assume the user will always enter the correct **type** of data, but might enter data that is not in the correct **range** of values.

4. Perform the desired operation using the provided input values, and output the formatted result.
5. Return to the menu and ask for the next selection. Repeat this until the user decides to Quit.
6. **Use constant variables to represent any "magic numbers" in your calculation.** There should be no constant literals in your code unless they are being saved to a constant variable, with the exception of main menu identifiers.

Example Output

User input is in *italics*. These do not cover every possible test case. It is up to you to thoroughly test your program before submitting it.

```
Main menu:
1. Is it a palindrome?
2. Make a palindrome!
3. Quit
Choose a function:
1
Enter a phrase:
[PRETEND THIS LINE IS EMPTY]
Enter a phrase:
Go hang a salami, I'm a lasagna hog!
"Go hang a salami, I'm a lasagna hog!" is a palindrome!
Main menu:
1. Is it a palindrome?
2. Make a palindrome!
3. Quit
Choose a function:
1
Please enter a phrase:
The times, they are a changin'
"The times, they are a changin'" is not a palindrome.
Main menu:
1. Is it a palindrome?
2. Make me a palindrome!
3. Quit.
Choose a function:
2

Please enter a phrase:
Senile F
Should the last letter be repeated? y/n
n
"Senile F" made into a palindrome is "Senile F elineS"
Main menu:
1. Is it a palindrome?
2. Make me a palindrome!
3. Quit.
Choose a function:
2

Please enter a phrase:
No lemon
Should the last letter be repeated? y/n
```

```
y
“No lemon” made into a palindrome is “No lemonnomel oN”
Main menu:
1. Is it a palindrome?
2. Make a palindrome!
3. Quit
Choose a function:
3

Bye!
```

Discussion Questions

In addition to uploading your code to Dropbox, you must also turn in **handwritten** answers to the following questions.

1. A **bit string** is a string that consists entirely of 0's and 1's, as in **0011101**. Bit strings can be palindromes the same as any other string, if they are "spelled" the same forwards as backwards.
   (a) List all bit strings of length 1 that are palindromes. (There are two.)
   (b) List all bit strings of length 2 that are palindromes.
   (c) List all bit strings of length 3 that are palindromes. Hint: for each palindrome of length 2, insert either a 0 or a 1 in the middle of the string to make a new string that is a palindrome.
   (d) List all bit strings of length 4 that are palindromes.
   (e) List all bit strings of length 5 that are palindromes using the same procedure as in (c).
   (f) How many bit strings of length n are palindromes? Your answer should be in terms of n, and will probably need different equations/results depending on whether n is even or odd; it should also be consistent with your answers to (a) through (e).
2. Consider the Latin sentence "Sator Arepo Tenet Opera Rotas", one of the first palindromes in human record. An amazing fact about this phrase is revealed when we write the words in a "word square", one word written per line of the square, as such:

   ```
   S A T O R
   A R E P O
   T E N E T
   O P E R A
   R O T A S
   ```

   Read the horizontal lines of the square... then read the vertical columns top to bottom, left to right. Notice anything cool? Indeed, the first word "Sator" is spelled with the first letter of each of the five words. The second word "Arepo" is spelled with the second letter of each of the five words... etc., etc., until "Rotas", spelled with the fifth letter of each of the five words. Amazing.

   Invent a 4x4 word square with English words of 4 letters each, choosing whatever words fit into the same pattern as above. Your four words do not need to form a palindrome (that's... *really hard*), but brownie points will be awarded if your phrase sounds like a real-ish sentence. Example: Then Hera Errs Nasa.
3. Watch the following music video: `https://youtu.be/JUQDzj6R3p4`. What is notable about each of the phrases in the song's lyrics? Please share your favorite phrase from the song.