Parameterized functions that return values, While loops (chp 8)

1. [~5 minutes] Show an instructor or a TA that your warm-up 10 program (GUIstrings.py, reviewed at the start of class today) is working properly. Show them your code and how the string functions smush() and reverse() are coded up as separate functions, and how they are each called from the main function where the GUI code is. If you already tried adding a while loop to the GUI (as we were doing at the end of class today) so that it could take multiple clicks, don’t show them that version yet. Take that code out and show them the version that only takes a single button click.

   ☺ Get check 1 ☺

2. [~15 minutes] Your third check for this lab (below) will have you add a final button to your warm-up 10 program (GUIstrings.py) that’s labeled “Palindrome?” The functionality of this button will just be a combination of the functionalities of the three you’ve already built. It will check whether or not the user-inputted text is indeed a palindrome. If you don’t know/remember what a palindrome is, you should google it. Some examples of palindromes include:

mom
dad
radar
Race fast, safe car.
Rats live on no evil star.
Madam, in Eden I’m Adam!
A man, a plan, a canal – Panama

Display a message telling the user whether or not his/her input is a palindrome.

For this (your second) check, start by writing a separate function that gets called to do this check for palindromicity. This function will be a boolean function, simply returning True if the string parameter is palindromic, and False otherwise. Name your function isPalindrome(phrase). Don’t write this code from scratch. Call on the other functions we’ve already written (namely, smush() and reverse()) to compactly and elegantly accomplish this for you.

Take a few minutes to brainstorm first. If you can’t figure it out, or if you want to check your solution, read on for the details on how to code it!

To determine whether or not the inputted text is palindromic, first smush() the text and save the result into a variable. Then, reverse() the smushed text into another variable, and compare it with the originally smushed text. If they are the same, then you have a
palindrome, otherwise, you don’t. Test your function separately by passing it a string and printing the return value to the shell: e.g., `print(isPalindrome("mom"))`.

**Get check 2 (for the isPalindrome() function itself, without any GUI integration yet)**

2. [~10 minutes] Now add the palindrome button to the GUI and when the user clicks it, call your isPalindrome() function from Check 1 to determine whether the user-inputted string is a palindrome. Write an if statement that outputs text to the GUI saying either "Your phrase is a palindrome!" or "Sorry, not a palindrome."

**Get check 3**

3. [~15 minutes] Event-driven random color. (Still no looping. Just a single mouse click.)
   a. Save and open the program funsquare.py. It creates and draws a Rectangle object, takes a mouse click, and then closes the window. (You’ll need the usual graphics module graphics.py too.) Read the code, test it, and make sure it behaves as advertised.
   b. Add code so that if the user clicks the rectangle, it is filled with a random color. Recall that to generate a random color, use the `color_rgb(r,g,b)` function, where each argument is an integer between 0 and 255. The first argument `r` stands for the amount of red in your color, `g` for green, and `b` for blue. So, for example, `.setFill(color_rgb(255,0,0))` colors your shape red, `.setFill(color_rgb(0,0,0))` colors it black, and `.setFill(color_rgb(255,255,255))` colors it white.
   If you use the `randrange(start,end)` function to generate a random integer between 0 and 255 for each of the three arguments `r`, `g`, and `b`, then `.setFill(color_rgb(r,g,b))` will color your shape a random color.
   Remember, to use the `randrange()` function, you will need to first import it, like this: `from random import randrange`

**Get check 4**

4. [~15 minutes] A “definite” loop. Make a copy of your code for the previous exercise. Now modify the code to make the shape flash through 50 different random colors if the user clicks the square. (Put the code that sets your shape a random color into a loop --- which kind of loop would make the most sense here? --- and `sleep()` for about .02 seconds between each iteration.)

**Get check 5**

5. [~15 minutes] An indefinite loop. Modify the previous exercise so that rather than rapidly changing the colors of the shape, the color changes once each time the shape is clicked. If and only if the user clicks outside the shape, make the window close.

**Get check 6**

6. [~15 minutes] Password protection. What is the underlying algorithm behind any password prompt? E.g., this all-too familiar one:
Turns out: just a little while loop! (And I do mean little. As in, two lines.)

Edit your funsquare.py program so that it’s password protected. (To keep things simple we’ll just prompt the user for the password in the Shell terminal for now, not in the GUI.) Before the graphical window for your fun square even opens, tell the user that in order to use this program they need to type in the right password.

You should hard-code the correct password as a string constant at the top of your program. E.g.,

```
PASSWORD = "camels"
```

If they guess the password wrong, then make them keep trying, *indefinitely*. Only if they get it correct should the rest of the program resume.

😊 Get check 7 😊

7. [~15 minutes] Write a function that asks a user for an indefinite number of positive input values and averages the values. The user will be instructed to enter a negative number to indicate that s/he has no more input values. A sample run of the program is shown here:

```
Enter numbers to be averaged. (Entering a negative number ends the input.)

Number: 4
Number: 5
Number: 10
Number: -1

The average of your numbers is 6.33333333333
```
8. [~15 minutes] Go back to your GUIstrings.py program and save it as GUIpalindrome.py. Add an Exit button. Then add a single while loop to it the way we started to at the end of class today so that it will take as many clicks as the user wants until the user clicks Exit. Then the window should close and the program should end. Keep in mind that the program should respond as expected to various types of clicks:
- if the user clicks inside a button, just as before, it should take the string from the Entry box and execute that action on it.
- if the user does not click on a button it should do nothing or perhaps say to the user “you didn’t click on a button”
- [optional] it would also be nice if the user hasn’t entered any text for it to say “you didn’t enter any text”

Note the user should be able to go back and forth between clicking buttons and clicking outside the buttons, and changing the text string in between button clicks, and the program should continue to behave as expected. The structure of your program will be:

```python
pt = win.getMouse()
while not <quitButtonClicked>:
    <your big multi-way if statement here>
    pt = win.getMouse()
win.close()
```

You need to replace <quitButtonClicked> with a boolean expression that checks if the Point pt lies within the boundaries your quit button.

Extra time? (Complete these in any order, getting checks one at a time.)

- These are essentially some relatively simple ideas for the animation part of Programming Assignment 3, so completing them now can serve two purposes (point-wise)... **in any order, one at a time:**

  A. Build upon the button.py program we wrote in class, which has two buttons, one that draws a circle, and one that draws a rectangle. Recall that we created a parameterized drawButton() function that draws buttons according to the arguments being passed to it. We also had drawCircle() and drawRectangle() functions that we called for actually drawing the circle and rectangle. Modify the drawCircle() function so that it is more flexible. Add parameters for color, location (of center point), and size (radius) of the circle. Now, use your “enhanced” drawCircle() function to create...
    1. a series of concentric circles (like a target) [with careful use of the loop counter]
Bonus A1

ii. a window full of colorful, randomly placed “bubbles” of all shapes and sizes [using the random number function randrange()]

Bonus A2

B. Build on the palindrome program from lab today: if the user input is a palindrome, respond with a fun animation in celebration. For the animation, besides looping the .move() function as we’ve already done, you can also play around with looping some of the Text Methods which can be found in the Zelle text, like .setFace(), .setSize(), .setStyle(). You can also try things like having the color of an object flash quickly through many random colors. To do this: loop the .setFill(color) command and for the color, use the color_rgb(red, green, blue) function (see top of page 156 for explanation). For each color_rgb() argument, you can try the randrange() function, which you can look up in your text.

Bonus B

- **Bonus C**: Add to the password program of check 8 so that it only gives the user 3 total tries at guessing the password. If they fail three times the program should exit.
- **Bonus D**: Look at our program primes.py, which prints out all primes between 1 and a user-inputted value. Notice that for the function isPrime(num), we check all possible divisors between 1 and num. In fact, we can save the computer some work here. To determine whether a number is prime, we don’t need to check all divisors all the way up to num. We can actually stop much earlier than that. Change your code so it checks the minimum number of divisors it needs to determine whether a number is prime.