COM 110 Fall 2019 - Lab 5
Graphics and Functions

1. **Drawing an Entry object.** Write a module that creates a graphics window (say around 600 x 600 pixels large). In this window, create an `Entry` object (a text field for user input) with a width of 50 characters. As usual, have the window close upon mouse click.

☺ **Get check 1** ☺

2. **Adding a button.** Add a "button" (similar to our buttons from class) labeled "Lower Case!" Don't write the code for this from scratch! You should do this by calling our parameterized function we wrote in class that will create the button for you. (You can just copy the the code for the drawButton() function definition into your module. If your button.py code from class is not working, you should fix it first. Let us help you!)

☺ **Get check 2** ☺

3. **Taking user input and displaying output.** Add code so that if the user inputs text and clicks the button, the inputted text is displayed to the user (using a Text object) in all lower case. Upon a second mouse click, the window should close, as before.

☺ **Get check 3** ☺

4. **Defining a parameterized function that returns a value.** In the next two checkpoints, we will be adding a second button to the GUI that's labeled "Smush!" When the user clicks this button, the input string will be displayed, but this time in lowercase *and* with any non-alphabet characters removed. For example, the input

COM110 is the greatest! Programming is fun.

will yield

comisthegreatestprogrammingisfun

To keep our code organized and modular (just as you should for Programming Assignment 3), we will ***follow the steps below*** to create a *separate parameterized string function* that will actually take a string and smush it, **`return`**-ing the smush()ed string. Your main GUI function will then call this string function, passing the user-inputted text in for the function's parameter value. Let's take this one step at a time!

a) First, let's write the smush() function. Your smush() function should take a string parameter (the original string to be smush()ed) and return a string value. It will look like this:

```
def smush(originalStr):
      <code for generating smushed version of originalStr>
return <variable that holds resulting smushed string>
```

You should find a way to code this without using any built-in string functions to help you. It is only a few lines of code, and they are provided/explained in the paragraphs to follow.

If you are short on time, read on for the details on how to code it! Otherwise take a few minutes to brainstorm first.

i. loop through each character in the input string `originalStr`. For each character, check **`if`** its ASCII value falls within the range of lowercase alphabet characters. You can do this by using direct string comparison. So the boolean expression

```
"a" <= char and char <= "z"
```

evaluates to `True` if the character `char` is between the letters "`a`" and "`z`", but evaluates to `False` otherwise.

ii. Then, to construct your new string, simply *accumulate* the characters you want to keep one-by-one onto a string variable using repeated string concatenation:

```
newstring = newstring + char
```

But, don't forget to give an initial value to the `newstring` variable first, lest you ask the computer to evaluate it without giving it any definition/value first! (Analogous to initializing an integer accumulator variable at 0 so that it starts "empty," you should start the `newstring` variable out as an *empty string*: `newstring = ""`).

For now, just test your `smush()` function in isolation from the GUI, by hard-coding some test inputs and printing the resulting `smush()`ed text to the Shell. E.g.,

```
print(smush("COM110 is the greatest!"))
```

☺ **Get check 4** ☺

b) Now create the "Smush!" button in your GUI, and when the user clicks it, have your program call the `smush`() function you wrote in part a), displaying the returned value in your `Text` object.

☺ **Get check 5** ☺

5. In this part we will add a third button to the GUI that's labeled "Reverse!" When the user clicks this button, the input string will be displayed, but this time with all the characters reversed. For example, the input

Comp Sci

will yield

icS pmoC

a) Before you create the "Reverse!" button in your GUI, write a separate string function called `reverse()` (*similar in structure* to the `smush()` function above) that will get called to do the reversing. Again, you should find a way to do this without using any built-in string functions to help you.

If you are short on time, or can't figure it out, read on for the details on how to code it! Otherwise take a few minutes to brainstorm first.

Loop through the characters of the string parameter and simply concatenate them onto a new string, but add each character to the *front* of the string rather than the *back*:

```
newstring = char + newstring
```

Again, test your function via the Shell first, without worrying about integrating it into the GUI yet. (Keeping these functions separate from the GUI is not only good programming style, but doing so will come in handy for the next checkpoint!)

☺Get check 6☺

b) Now create the "Reverse!" button in your graphical window, having it invoke the `reverse()` string function you created in part a) above.

☺Get check 7☺

6. Add a final button to the GUI that's labeled "Palindrome?" The functionality of this button is just a combination of the functionalities of the previous three. 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.

Again, first write a separate function that gets called to do this check for palindromicity. This function will be a **boolean** function, simply returning `True` if the parameter is palindromic, and `False` otherwise. Name your function `isPalindrome(phrase)`.

If you are short on time, read on for the details on how to code it! Otherwise take a few minutes to brainstorm first.

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.

Then, to integrate your isPalindrome() function with your GUI, from the main method, write an additional **if** statement that calls isPalindrome to check if the user input is palindromic and then display a sentence like "That's a palindrome!" if it is and "Sorry, not a palindrome" if it's not.

☺Get checks 8 (for the function itself) and 9 (for integrating it with the GUI)☺

Extra 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…

i. a series of concentric circles (like a target) [elegantly done with clever use of the loop counter]

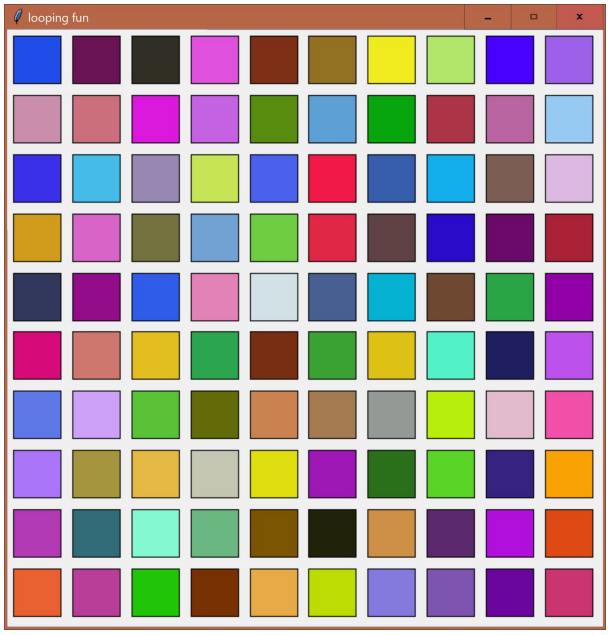
☺Bonus A1☺

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

☺Bonus A2☺

iii. a 10x10 grid of randomly-colored rectangles, like below. It's surprisingly simple. Try `win.setCoords(0,0,10,10)` to set up the window, then use nested for loops like this.

```
for row in range(10):
    for col in range(10):
        ...
```



You can make these appear to change colors by re-drawing the rectangles 20 times. (Just put your nested loops in yet another loop that makes them execute 20 times!)

☺Bonus A3☺

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. For each `color_rgb()` argument, you can try the `randrange()` function, which you can look up in your text.

☺Bonus B☺