Functions Exercises

#1 Moving Trucks (10 points)

Experienced programmers are very good at 'thinking in functions' but even still they often realize that the first implementation is not perfect. Through a process known as refactoring, a programmer can improve their code to centralize common code into a function and make that function customizable using cleverly selected parameters. This week's playlist includes an example of this in an animated graphic exercise that draws trucks speeding down a highway. The first implementation did not make very good use of functions, but I walk through the process and thinking involved in refactoring the code to not only be easier to read, easier to manage, but more functional for future applications as well.

You can earn points by following along with this demo and mimicking the changes I make. Your code should be functional and organized using the same functions as mine, but feel free to change or add to the visual display to make it your own. This assignment has no autograder, so your task is to replicate the functions and if you can, make me smile when I see your display!

https://www.youtube.com/playlist?list=PLgvXUCQxGQrznKW4EoDk3vf5jzK5Cy8Zk

#2 Lessons on Impressions (10 points)

I wanted to build an app that helps me to learn different impressions of accents and celebrities. Give it a quote, and it returns a script customed to using that quick of speaking. I got it working but then realized the code is not very flexible or reusable, and I need to refactor it. Check out the starter video on my thinking of how I want the program to work, and then complete my refactoring job!

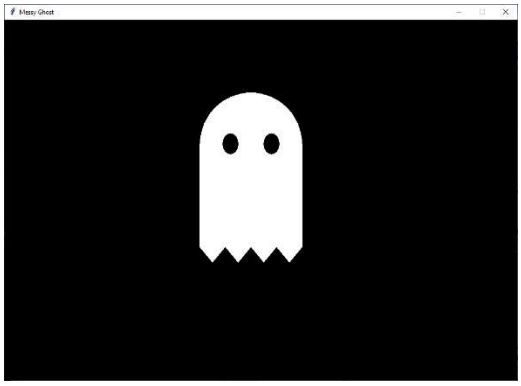
- Add a comment block[1] that explains the current behavior of each 'section' of code. A section may be one line, a handful, or many that make up some specific goal. You are not explaining each line of code, but the larger purpose of reaching the goal
- Once you have identified the bigger pieces and steps of the algorithm, consider how you can break them into functions. Create those functions and refactor your code so it executes exactly as it did before.

[1] In Python you can create a comment block by either putting a # at the beginning of each line or use the triple quotes (''') around the text you want to act as a comment. Just remember that the triple quotes must conform to the indentation rules. You can use a single comment block at the top of the file laying out the entire logic, but you can also comment each section as you go. Your goal is not to describe literally what the code is doing (e.g., the code adds one to the value count) but the purpose of that line of code (e.g., the code adds one to the counter that tracks the number of characters in the name).

#3 Messy Ghost (30 points)

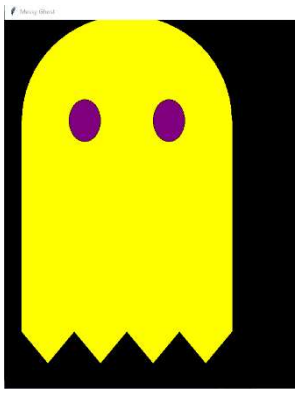
I started a screensaver app to draw a friendly ghost that floats back and forth across the screen. My initial code worked but was not very flexible or reusable. I have a few ideas that might improve the code, which you might take one.

Refactor into functions ***(5 points)***



My code puts everything into one main method, which is terribly inflexible. Refactor (i.e., redesign and rewrite) my code by creating a function that just draws a ghost. That function must let the caller choose the ghost's position on the screen by providing an initial x and y location as parameters. The final product after your changes should draw the ghost exactly as before you started making changes. Test your change by moving the ghost to different positions on the screen. If you did not refactor everything to conform to the new x/y position your ghost drawing may go crazy!

Enhance your function ***(5-15 points)***

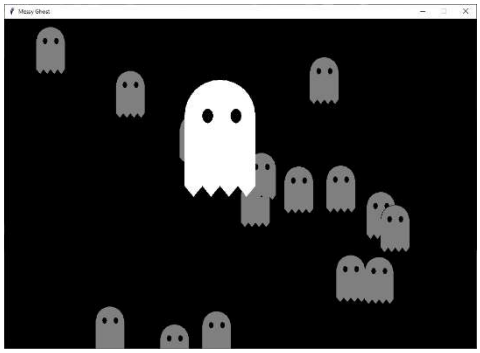


I have a few small changes I want to add to the basic ghost drawing. I want to be able to change the color of my ghost. Allow the caller a parameter to determine the ghost's body color and another for the eye color. If you want to be fancy, look up how Python allows you to set a default color for these parameters, so they are note required on every call.

I also want to make my ghost's size dynamic. Instead of setting a fixed height and width, I want to scale the height using a single parameter. For example, if passed `0.5` the ghost would be half the width and height, or `2` would make the ghost twice as big.

To move or not to move ***(10 points)***

It would be nice to separate the drawing and animation functions such that I can add ghosts to the scene as background that do not move. Split your function into two where one draws the ghost and the other makes the ghost move. Your animation function will need to access the individual 'parts' of your ghost so you will probably need to add a return value from the drawing function that passes a parameter to the animation. Consider what data structure might work best for this (e.g., a list, tuple, dictionary, class… any can work). Modify the main function to save the parts from the draw function and pass them to your new move function. Remember, you do not need to create new ideas, but merely move existing code. You need to identify the code required for animation, move it to the new function, and finally ensure that the new parameters to your function align with the copied code. Again, you know it works if the screensaver's behavior does not change after your changes. Once you are sure it works, feel free to add whatever flourishes you want to create a new ghost-filled display.



#4 Tougher Refactoring Problem (25 points)

Since Python code is stored in files, we can even write Python code to analyze Python code. Another bit of code I started analyzes how many functions and other elements of Python you are using in your source code. Once again, my code seems to be a mess, as I never managed to build more than one function. You must refactor the "refactorUsefulFunctions.py" code to create helpful functions.

The input file for this program is the code you are writing itself. Your program counts the number of total lines, blank lines, and coded lines in your code. It also counts how many functions and comments you use. One of the issues in my code is an overreliance on copy/paste, which means there are many redundant copied of logic that would be better organized as a function on its own. Tackle this task in two parts:

- Use comment blocks (see the first problem) to document the current behavior of each 'section' of code. A section may be one line, a handful, or many that make up some specific goal. You are not explaining each line of code, but the larger purpose of reaching the goal
- Once you have identified the bigger pieces and steps of the algorithm, consider how you can break them into functions and make those changes. Your program functionality will not change, but the output will to reflect your changes in code!