Introduction

Today in studio, you will complete several tasks that will help you get closer to completing your assignment. You should be prepared to engage with your classmates for the entire studio time. The main goals for today are:

1. To practice getting input and working with variables.
2. To practice evaluating and writing mathematical expressions.
3. To practice writing if statements.

You will submit your completed assignment as a .pdf at the end of the studio session.

Deliverables:

1. A table of expressions and results from Part 1
2. Your python code -- just the function "Plot_Paintings"
3. The calculated spacing between paintings for case 1.
4. A screenshot of your arrangement on the wall for each case, without edits to the colors and/or borders
5. A screenshot of any arrangement with edits to the colors and/or borders

Part 1 (~30 min)

Using what you have learned about how expressions are evaluated based on types, evaluate the following expressions by hand and provide your expected results from the computer.

1. Fill out the following table below with the **expected result** by hand calculating the expression.

Expression	Expected Result	Actual Result
20 / 2 + 4 + 5.5 + 3 * 4		
20 / (2 + 4 + 5.5) + 2 / 4		
20 / 2 + 4 + 5.5 + 2 / 4		
2 * (45.0 + 10) / 3		
2 * 45.0 + 10.0 / 12		
2 * 45.0 + 10 / 12		
28 / (6 + 12) * 2**		

2. Using the python console in Pycharm, evaluate each of these expressions (copy paste the expression next to the >> and hit enter). Record the **actual result** in the table above.

Part 2 (~60 min)

Write a small program to help plan how to arrange artwork in a gallery.

An art gallery needs your help arranging the artwork in its next exhibit! The director explains to you that all of the artwork is of the same size, and there is only one wall upon which to place it. The paintings must be placed in a single row along the wall, with equal spacing between each. Additionally, the spacing between the edge of the wall and the closest painting must be equal to the spacing between paintings. See the diagram below.

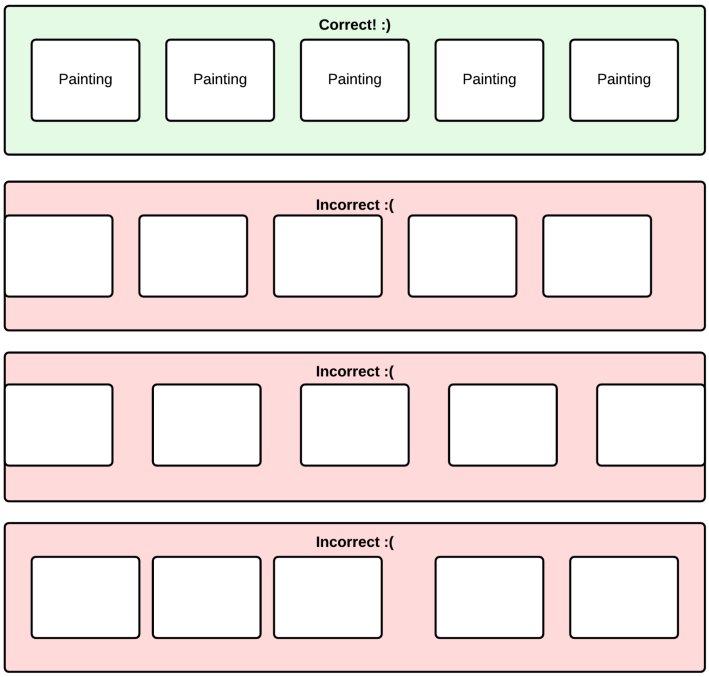

Figure 1. Correct and incorrect painting arrangements

Given inputs of the wall length, painting length, and number of paintings, determine whether the paintings will fit on the wall and calculate how far apart to place them so that they are evenly spaced. Then use the Python functions provided by the TAs to display your arrangement. If the wall cannot fit all of the paintings, a custom error message should be displayed. See the flowchart below.

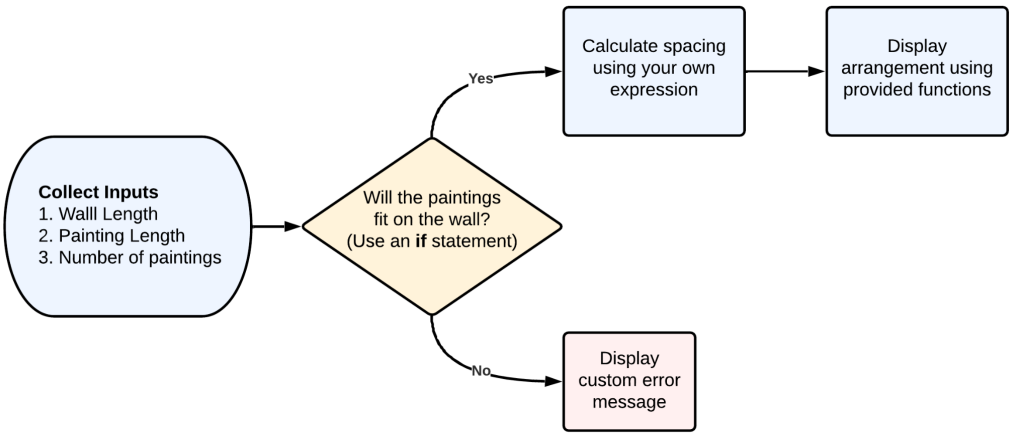

Figure 2. Flowchart illustrating program to be written

In order to determine whether the paintings fit on the wall, you will need to calculate the total length of the paintings and compare it to the length of the wall. Depending on the result of the comparison, one of two things will happen. In the case that the paintings do not fit, a statement should be printed that says something like "Oh no! These paintings will not fit on the wall!" In the case that the paintings do fit, the spacing between paintings (and edges of the wall) should be calculated and the resulting arrangement displayed. A good approach is to calculate the length of empty space and divide it by the number of empty spaces.

The TAs will provide one function that accepts inputs of wall length, painting length, and number of paintings, and outputs a diagram of the paintings on the wall. This function contains an `### EDIT HERE ###` section on line 38 that requires you to write an expression to calculate the spacing between paintings, and assign it to the variable "`Spacing`". Also, write your if statements to tell the user whether the paintings fit on the wall.

When the code runs, it will expect you to click within the plot several times to advance the loop. It should begin with a blank wall and, as you click repeatedly within the plot, paintings should fill the wall.

CASE #1:

- Wall length = 10 meters
- Painting Length = 2.5 meters
- Number of paintings = 3

CASE #2:

- Wall length = 15 meters
- Painting Length = 3 meters
- Number of paintings = 6

CASE #3:

- Wall length = 7 meters
- Painting Length = 2.5 meters
- Number of paintings = 1

Part 3

Now that you've successfully placed the paintings on the wall, try to make your arrangement look prettier! You can change the inputs to the functions "plt.plot()" and "plt.fill()" to show different border widths and styles, different fill colors, etc. For optimum brownie points, see if you can replicate this figure:

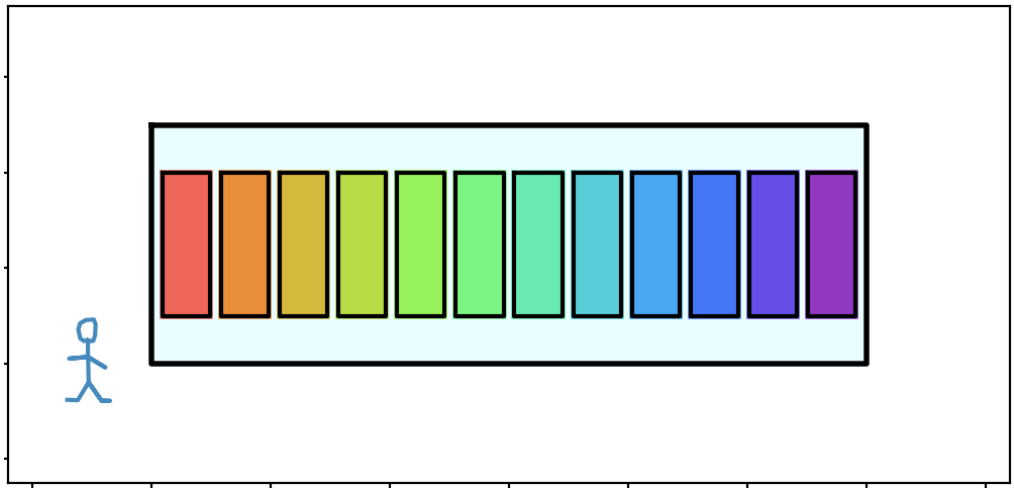

Figure 3. Rainbow arrangement. There are twelve paintings. From left to right, their colors follow the rainbow pattern [red,orange,yellow,green,blue,indigo,violet].

Rather than inputting colors as strings like 'red' or 'green', try inputting RBG triplets like [1,0,0] or [0,1,0]. Your program should present a rainbow pattern regardless of how many paintings are on the wall.

All of the possible inputs to the function "plt.plot()" are shown here in the matplotlib API: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot
An API (Application Programming Interface) is the usual place to go when you want to know more about the code that you're already using. For example, when you plot something with plt.plot(), you can input the bare minimum (x and y data) or you can input all these other arguments! To change the filled rectangle properties, find the page for matplotlib.pyplot.fill() and adjust the inputs for that function as well.