CS 115, Autumn 2021
Programming Project #6: Progress Visualization (20 points)
Due: Tuesday, November 15, 2021, 11:30 PM

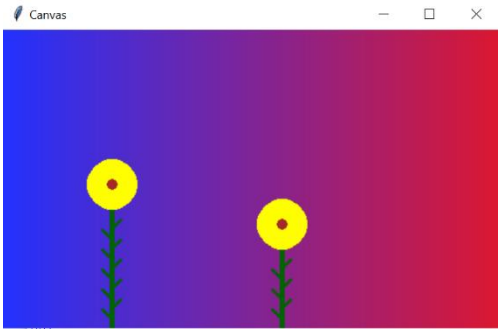
Program Description:

This project covers parameters and graphics. Turn in a Python file named `fitness_visualization.py`.

To run this project, you must download the file `ECGUI.py` from the Project section of the class web page and save it in the same folder as your code. Do not turn in `ECGUI.py`.

Many programmers who specialize in the subfield of Human Computer Interaction (HCI) work on designs to help users visualize progress and stay motivated. One really common form is flower(s) or tree(s) that grow as the user completes a desired activity. You may have seen these in fitness apps, like the Fitbit on the left, cars, and other places.

This week you will write a program that asks the user for a number of steps and then outputs flowers whose height and number depends on that number. The more a user has walked, the nicer their flower garden will be. Your program should produce an image similar to the one on the right. The background should be a gradient between two colors, either left to right or top to bottom.



If there are no visible differences to the naked eye between the expected output and your output, we will consider it correct. Assume the user always types in a valid value.

Background:

```
Preferred width? 500
Preferred height? 300

Start red? 35
Start green? 50
Start blue? 255

Stop red? 220
Stop green? 24
Stop blue? 50

Gradient direction (left/right or up/down? left

How many steps have you walked? 8767
```

The background should be a gradient between two colors the user specifies. To create a gradient between any two colors that continues across the panel, compute the amount you would need to add to the red each time to reach the stop color. For example, if the start was `35` and stop was `220` you could compute the amount by finding the difference, `220 - 35 = 185` and then dividing this number by the width of the panel to create a left to right gradient. If you are creating a top to bottom gradient divide by the height instead of the width. Repeat the process for the green and blue values. Then, to draw the gradient, draw lines from one side of the drawing panel to the other, each one pixel more to the right (if a left to right gradient) or the bottom (if a top to bottom gradient), each time increasing the red, green and blue by the numbers computed above. For example, given the input displayed above, your calculations would look like the following:

Red: `(220 - 35) / 500 = 185 / 500 = 0.37`
Green: `(24 - 50) / 500 = -26 / 500 = -0.052`
Blue: `(50 - 255) / 500 = -205 / 500 = -0.42`

Now, repeat drawing vertical lines that stretch from top to bottom. The first should be at an `x` of `0`, the second an `x` of `1`, the third an `x` of 2, etc. Each time you draw a line, add the values calculated above to its red, green

and blue. Here are the values for red, green and blue that would be used for the first 5 lines in the proceeding example:

```
red: 35.0    green: 50.0            blue: 255.0
red: 35.37   green: 49.948          blue: 254.59
red: 35.74   green: 49.896          blue: 254.18
red: 36.11   green: 49.844          blue: 253.77
red: 36.48   green: 49.792          blue: 253.36
```

Create a color from red, green and blue values by writing: `Color(`**`red_value`**`,` **`green_value`**`,` **`blue_value`**`)`

Required Behavior:

When your program starts it should output the introductory message, prompt the user for the window size, prompt the user for red, green and blue start and stop values, prompt the user for the gradient direction and prompt the user for the number of steps they have walked. An example of this interaction is shown above. You can assume that the user will input an integer value greater than 0.

Once the user has typed in their data, your program should open a `drawing_panel` with your gradient background displayed on it. This background should match the values the user input. It should then animate drawing flowers on the `drawing_panel` based on the number of steps the user input. Your program should draw one flower with 10 leaves for every 5000 steps a user has walked. If a user has walked a number of steps that has a remainder when divided by 5000 then the program should draw one more flower. This flower should have two leaves for every 1000 additional steps the user has walked. Therefore, if the user walked 7809 steps, the program should draw one flower with 10 leaves and one flower with 4 leaves.

Flowers should all start at the bottom of the screen. They should be placed at random `x` locations. Flowers should never appear partially off the screen; make sure you choose random locations where the whole flower can fit on the screen.

The stalk should grow `20` pixel taller every time two more leaves are added. Both the stalk and leaves are lines that have a stroke weight of `5` pixels. The leaves are positioned at intervals of `10` on the stem. Their `x` starts at the stem and ends `10` pixels away from it. Their `y` also ends `10` pixels from where it started.

The flower petals are a single `50 x 50` oval centered above the stem. The eye of the flower is a `10 x 10` circle centered inside this one.

You may choose any colors you like for the flowers.

Implementation Guidelines:

To receive full credit, you are required to have the three particular functions described below. These functions use parameter passing and perform some of the program's complex computations. One of them also uses returns.

Function to draw a gradient

This function should draw the gradient background. It should not ask for any user input. Instead, it should take all data it needs for parameters. It should be able to draw a gradient between any two colors and, depending on the user input, either left to right or top to bottom.

Function to compute the change in color

This function should take necessary information about one component of a color (reds, greens or blues) for two colors and compute the amount to add to the start color each time to reach the stop color smoothly. This computation is described in detail in the *Background* section of this document.

Function to draw a flower

Your function should draw a single flower. Notice that we specify each flower in terms of its location and how many leaves it has. Different flowers have different positions, and numbers of leaves. Therefore, your function should accept several parameters so that it is possible to call it many times to draw the many different flowers potentially on the screen.

You should also create other functions to capture structure and redundancy. It is up to you to figure out which other functions are necessary.

Development Strategy (How to Get Started):

This program does not require as many lines of code as some past ones. However, the numeric computations and parameters are not simple. You might be overwhelmed with the amount of detail you have to handle all at once. As famous computer scientist Brian Kernighan once said, "Controlling complexity is the essence of computer programming." To make things easier, begin with a smaller piece of the problem.

Write your code in stages, repeatedly making small improvements. Start by having your background increase each shade by 1 each time. Then, compute the actual change and add it in.

For the flowers, start by having your flower function draw only one flower of a set height at a set location, then generalize it by **adding one parameter at a time**. For example, add a parameter to change the `x` position. Test the parameter by passing different values. Once it works, move on to the next.

Style Guidelines:

For this assignment you are limited to the language features discussed in lectures 1 - 9.

We require at least the three functions named previously. You should use additional functions when they would improve your code. You may receive a deduction if your functions accept too many parameters or unnecessary parameters. An "unnecessary" parameter in this case is one whose value is redundant with another's, could be computed using others' values or simply isn't used.

Use `panel.sleep(100)` to animate your leaf drawing. You may not use `panel.animate()`.

Give meaningful names to functions, variables, and parameters. Follow Python's naming standards as specified in lecture. Limit the lengths of your lines to fewer than 80 characters. Include meaningful comment headers at the top of your program and at the start of each function.