Project Description [5 pts]:
Name: Fourier Drawing Machine
A program in which users can make or customize variants of a drawing machine to create spirograph drawings. The machine will be customizable in several ways: the radius, positioning, and number of wheels, etc. The user will be able to save and export drawings.

Competitive Analysis [5 pts]:
After some preliminary research, I have found that there are several kinds of spirography/drawing machines. There is a simulation of a simple toy spirograph with gears and holes (https://nathanfriend.io/inspirograph/). I might add a similar feature should I have time, but I want to focus on more advanced drawing machines. I also found spirography simulators with a nice stator-motor interface with adjustable radii (https://seedcode.com/SpirographN/sgn.html). I think I will also try to implement this feature to simplify user interaction. Another machine I found utilized drawing arms connected to the spinning wheels (htmlspirograph.com). I want to implement this feature in addition to having several ways of manipulating the drawing paper as well.

	Interface	Customizability	Playback	Draw Shaft
Competitor 1	Boring	Standard	No	No
Competitor 2	Elegant	Extremely	Yes	No
Competitor 3	Cool use of color/gradients	Too much -- overwhelming	Yes	Yes

Structural Plan [5 pts]:
The project will be organized into several manageable files: one main file, which runs the app, several repeatedly used classes (ie. Button and Slider classes) in another file, and most likely another file which stores any saved drawings via a string of the __dict__ of the specific instance of the Paper object. In terms of a drawing machine, there is a wheel object and an assembly object, which stores multiple wheels. In the drawing mode with a shaft, there will be a shaft object which controls the pen. Most likely, I will also need to make a Paper object to store and manipulate the drawings.

Algorithmic Plan [5 pts]:
There is a main wheel in an assembly with a given dtheta. This given attribute determines how all other wheels in an assembly behave via rotational velocity equations. The next trickiest parts will be connecting a draw shaft to several wheels, creating a Paper object which can be rotated and translated while the machine is drawing.

Approach Below:

To connect a drawing shaft to the existing drawing machine, I can define a fixed point and define a shaft, which is a list containing three tuples of (x,y) coordinates: the point on the wheel in the assembly, the fixed point, and the calculated point where the pen draws from.

A Paper object which stores a path, must have several attributes to behave correctly. In order to store points and animate them moving, I will end up creating a rotation matrix to store points in a regular 2d list multiplied by an appropriate rotation factor. Then, when viewing the Paper object, I can present it with zero rotation. The same is true of a translating Paper, expect instead of multiplying, we add by a translation factor.

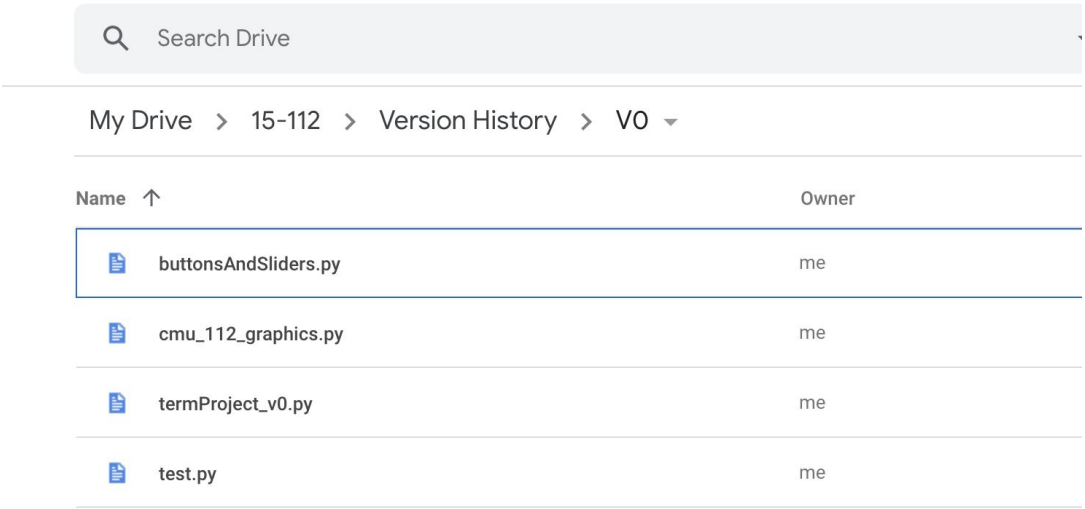
For exporting images I will be using file I/O to save a str(__dict__) of a Paper object to a locally created file which stores all previously created drawings. Then, in the View Mode, the drawing will playback to file.

Timeline Plan [5 pts]:
A timeline for when you intend to complete the major features of the project.

11/21 ⇒ Draw Shaft, Pen Customization completed
11/23 ⇒ exporting images completed
11/24 ⇒ Interface between apps completed or at satisfactory level

11/25 - 11/31 ⇒ Math Mode, Paper Object completed, Implement importing svgs and Fast Fourier Transform

Version Control Plan [3 pts]:
Uploading code to Google Drive



Final Version will be uploaded to GitHub

Module List [2 pts]: Not applicable

TP2 Updates:

Currently, the program can draw and has a satisfactory level of user interface. The user can save and view files. The pen can be customized. There is also a wave feature.

To do:

- Fix the draw Shaft
- Implement FFT by importing svgs
- Make color further customization

TP3 Updates:

Currently, the program can successfully draw, use an FFT with imported .svg files. The user can change the pen color in the middle of a drawing to create a multi-colored drawing. The draw shaft idea was scrapped due to time spent implementing FFT. Additionally, there is a drawSpeed variable so that the user can edit the playback speed of a current drawing or a saved drawing.

Advice from the TP-user-study-a-thon:

- Altering playback speed ⇒ implemented
- Saving drawings with multiple colors ⇒ implemented
- Use a textbox for the color editor
    - ⇒ I chose to ignore this advice because I personally like the RGB slider interface