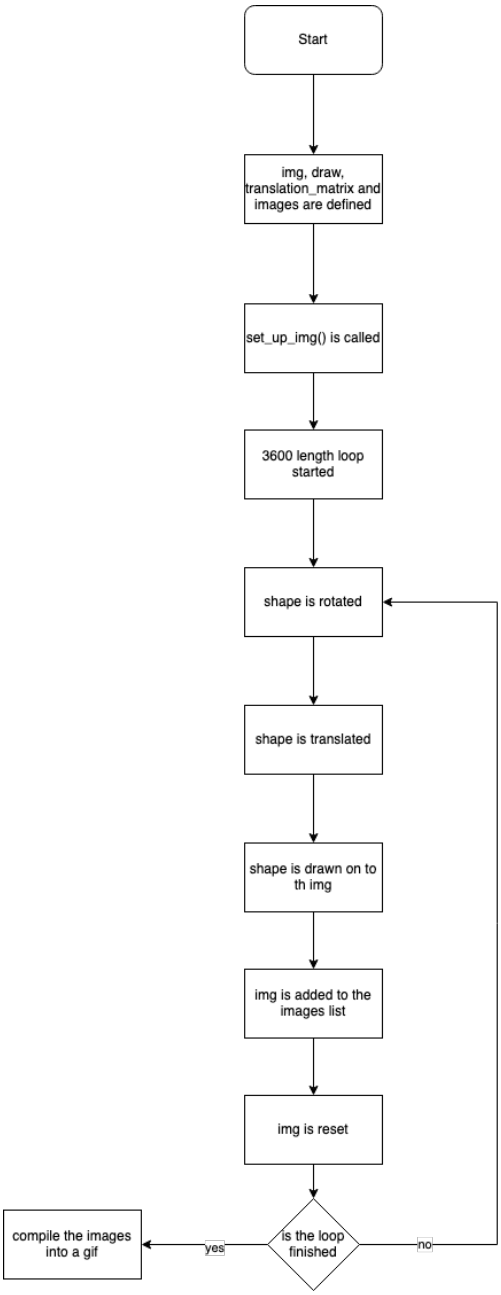
Rotating Square Documentation

Success Criteria

- Generate a gif as an output
- Create a rotating Square that infinitely rotates around the origin of the axis

Algorithms

This algorithm is the run-through of the program, to create the gif.



Mathematics

This project uses matrices as key aspect of its function. They are used to store the position of each of the vertices on the square. The key use of matrices in the project is to perform a rotation. The rotation matrix is shown here: (where θ is the angle to rotate)

$$\begin{pmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{pmatrix}$$

This matrix is applied to the matrix containing the four points of the shape iteratively to create a rotation of 0.1° per frame. By the doing this 3600 times, it creates a rotation of 360°, allowing the gif to smoothly loop. In addition to this, the program includes manual matrix addition and multiplication functions to handle both translations and transformations.

Annotated Development

Imports:

PIL (Pillow) - Python Image Library allows the user to create, manipulate, and draw on images. In this project, it is used to create the blank image, and draw the axis on. It the draws the square onto the screen in each frame and clears it afterwards. It is also used to Compile the images into the gif format.

Numpy - Numpy is used in this project for its array function. This allows for the creation, reshaping and indexing of the matrices.

Math - This library is used for the sine and cosine functions that it provides. These are needed for essential calculations to rotate the square for each frame.

Alive Progress - This library is used in this project to provide a command line progress bar when generating the image. It has a wide range of styles which can help the aesthetic of the program.

```
from PIL import ImageDraw, Image # python image libray which allows the creation of and drawing on images
import numpy as np # numpy stores each of the matricies as arrays
from math import cos, sin # math allows the use of the sine and cosine function
from alive_progress import alive_bar # alive_bar creates a nice progress bar when images are being generated
```

Main Functions

The Matrix multiplication function is designed to multiply 2x2 matrices with 2x4 matrices. It does this by manually indexing both matrices to create the product of the multiplication. It will then return a 2x4 matrix which is the product of matrixA and matrixB

```
def matrix_multiplication(matrixA, matrixB):
    """
    - this function does matrix multiplication
    - it is built to multiply 2x2 with a 2x4 matrix
    - incorrect shapes will yield an error
    - it returns a 2x4 matrix
    """
    a = (matrixA[0, 0] * matrixB[0, 0]) + (matrixA[0, 1] * matrixB[1, 0])
    b = (matrixA[0, 0] * matrixB[0, 1]) + (matrixA[0, 1] * matrixB[1, 1])
    c = (matrixA[0, 0] * matrixB[0, 2]) + (matrixA[0, 1] * matrixB[1, 2])
    d = (matrixA[0, 0] * matrixB[0, 3]) + (matrixA[0, 1] * matrixB[1, 3])
    e = (matrixA[1, 0] * matrixB[0, 0]) + (matrixA[1, 1] * matrixB[1, 0])
    f = (matrixA[1, 0] * matrixB[0, 1]) + (matrixA[1, 1] * matrixB[1, 1])
    g = (matrixA[1, 0] * matrixB[0, 2]) + (matrixA[1, 1] * matrixB[1, 2])
    h = (matrixA[1, 0] * matrixB[0, 3]) + (matrixA[1, 1] * matrixB[1, 3])
    return np.array([[a, b, c, d], [e, f, g, h]]).reshape(2, 4)
```

The matrix addition function allows for the addition of 2 2x2 matrices. It returns a 2x2 matrix which is the sum of matrixA and matrixB. Its purpose in the program is to translate the points so that they are rotating around the origin of the graph (the centre).

```
def matrix_addition(matrixA, matrixB):
    """
    - this function performs matrix addition
    - it takes in 2 parameters - matrixA and matrixB
    - both must be 2x2 matricies
    - it returns a 2x2 matrix
    """
    a = matrixA[0, 0] + matrixB[0, 0]
    b = matrixA[0, 1] + matrixB[0, 1]
    c = matrixA[0, 2] + matrixB[0, 1]
    d = matrixA[0, 3] + matrixB[0, 1]
    e = matrixA[1, 0] + matrixB[1, 0]
    f = matrixA[1, 1] + matrixB[1, 1]
    g = matrixA[1, 2] + matrixB[1, 2]
    h = matrixA[1, 3] + matrixB[1, 3]
    return np.array([[a, b, c, d], [e, f, g, h]]).reshape(2, 4)
```

The draw square function is responsible for drawing the square on the screen. It takes in the 4 points of the square as parameters of the function.

```
def draw_shape(A_img, B_img, C_img, D_img):
    """
    - this function draws the sqaure every time based on the 4 points provided to it
    - the outline of the square is blue
    """
    draw.polygon([A_img, B_img, C_img, D_img], outline="blue")
```

The set up img function will clear the screen by drawing a white rectangle over it. Then It will draw both the x and y axis on the page. This function is used to reset the img for each frame of the gif.

```
def set_up_img():
    """
    - this function resets the image by drawing a white rectange over it
    - the axis are also drawn with both lines
    """
    draw.rectangle(((0,0),(500, 500)), fill="white")
    draw.line(((250, 0), (250, 500)), fill="black")
    draw.line(((0, 250), (500, 250)), fill="black")
```

Setup Variables

These variable are key variables in the program. The “img” variable creates an image object from the PIL library. Then, the “draw” variable is initialised, which allows shapes to be drawn onto the image. Next, the key “shape” variable is initialised. This stores the position of the four vertices of the square. The “translation_matrix” is used to centre the square on the screen. The “images” variable is an empty list, which was made to store all the generated images so that they can be compiled into a gif.

```
# this creates a nes 500x500 blank image
img = Image.new("RGB", (500, 500), "white")
# this creates a draw object that can draw on the image just created
draw = ImageDraw.Draw(img)
# shape is the original shape and its coordinates are (0, 0), (100, 0), (100, 100), (0, 100)
# it is reshaped to 2x4 to allow foir both matrix addition and multiplication
# it is initialised around the origin to make the rotaion work (see mathematics section in documentation)
shape = np.array([[0, 100, 100, 0], [0, 0, 100, 100]]).reshape(2, 4)
# the translation matix translates the square to the centre of the screen
translation_matrix = np.array([[250, 250, 250, 250], [250, 250, 250, 250]])
# this is the list that will contain each individual image
images = []
```

The Main Loop and GIF compilation

Firstly, a context manager is made to initialise the progress bar. Then the loop is started. The shape is rotated and stored in itself using the rotate shape function. The shape is rotated 3600 times at 0.1° to create an overall 360° rotation over 3600 frames. This is done to add smoothness to the animation. Then, each vertex is retrieved from the shape matrix and passed into the draw shape function. Then, the image is appended to a list and converted to HSV format as that is what the GIF compiler requires. Then the image is cleared, ready for the next image to be drawn. The Bar() function moves the progress bar on by one. Once the loop is finished, I use the PIL save() function to compile the list of images into a GIF. The duration 0.1ms per frame and it is set to look infinitely.

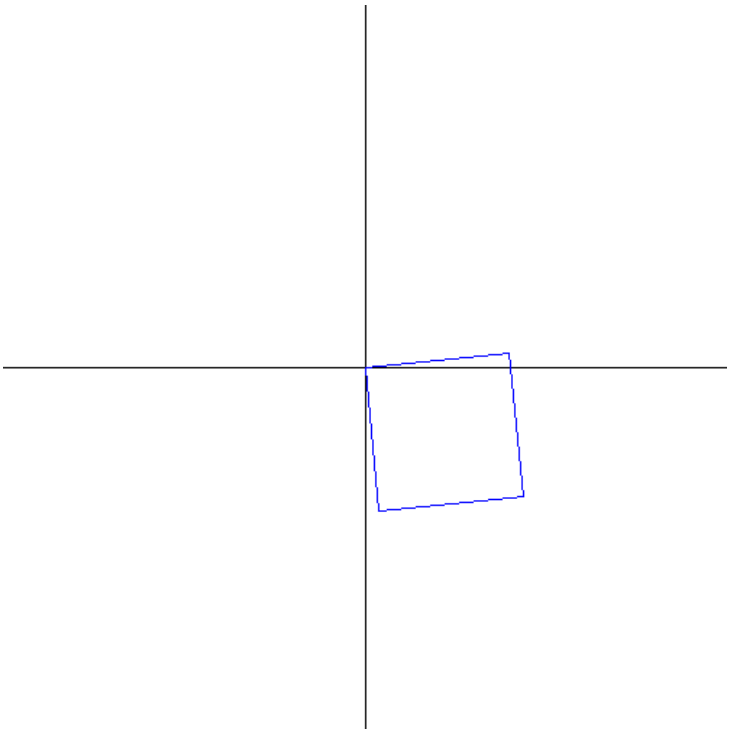
```
with alive_bar(3600, bar = 'smooth', spinner = 'waves') as bar:
    for i in range(3600):
        shape = rotate_shape(1/10, shape)
        shape_translated = matrix_addition(shape, translation_matrix)
        draw_shape((shape_translated[0, 0], shape_translated[1, 0]), (shape_translated[0, 1], shape_translated[1, 1]), (shape_translated[0, 2], shape_translated[1, 2]), (shape_translated[0, 3], shape_translated[1, 3]))
        images.append(img.convert("HSV"))
        set_up_img()
        bar()

images[0].save("Square_Rotation1.gif", append_images=images, save_all=True, duration=0.1, loop=0, optimize=True)
```

Images of the code running

```
(base) jamesrobertson@Jamess-MacBook-Pro Rotating Square % source /Users/jamesrobertson/opt/miniconda3/bin/activate
(base) jamesrobertson@Jamess-MacBook-Pro Rotating Square % conda activate pytorch_env
(pytorch_env) jamesrobertson@Jamess-MacBook-Pro Rotating Square % /Users/jamesrobertson/opt/miniconda3/envs/pytorch_env/bin
jamesrobertson/Desktop/Coding/VSCode/Rotating Square/rotating_square.py"
|                                        | 3600/3600 [100%] in 5.3s (684.83/s)
(pytorch_env) jamesrobertson@Jamess-MacBook-Pro Rotating Square %
```

The Output



Evaluation

Overall, this project was a success as it met the requirements set out in the success criteria. However, there are several things I could improve on in this project. I could first of all build the program around a class rather than having static functions. Secondly, I could have made the dimensions and colours of the squares variable. Finally, I could have improved my matrix multiplication and addition to be more dynamic - being able to multiply and divide matrices that are not just 2x2 and 2x4.