FUNDAMENTALS OF PROGRAMMING COMP1005 SEMESTER 1,2023 - ASSIGNMENT-1

Submitted by:
SANJANA JAYARAM MOTTEMMAL
CURTIN ID:21317503
MASTER OF COMPUTING (CYBER SECURITY)

CONTENTS

OVERVIEW

This program simulates a concert stage for the mythical rock band Spinal Tap. The overall program can be divided into different parts. They are lights, a smoke machine, props/band, backdrop, and choreography. The objective of this task is to provide a visual representation of the stage setup of the rock band's performance and allow users to visualize the stage experience in different scenarios by changing input parameters.

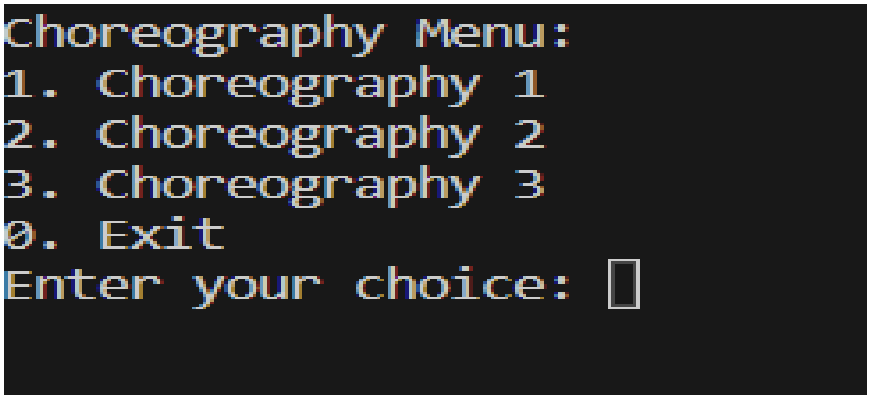
Developing this program was motivated by a desire to gain a better understanding of Python programming language's real-world applications and how to simulate a real-world scenario using Python, Matplotlab, and other dependencies. Furthermore, it provides practice for theoretical concepts in order to integrate the study portions in a more effective manner, and it provides a visual experience to a layperson other than a programmer.

Every programmer involved in this task faces a creative challenge. Dealing with the colour arrangement and integration is not only a programming task, but it is also a creative task. Nature always offers captivating visual experiences through diffusion. This program will also aim to demonstrate how smoke diffusion can be implemented in a more practical manner. It will also demonstrate how it can alter the surrounding lights

USER GUIDE

Run the program from the command line for all files other than spinal_tap.py using the "python "filename" command

After running Spinal_tap.py, a menu will appear. Select the desired option from the menu. Once you have made your selection, the program will execute the corresponding action. Upon completion of the action, it will return to the user.



TRACEABILITY MATRIX

No:	Feature	Code/Line of code	Testing/setup	Reference
1	**lights.py**			
i	Light Class-(colour, position, direction, and intensity (0-11)	Light class – attributes Lines-12-32	'lights list' creates a list of Light class's attributes, tested with different values- • Moving lights • Changing colour,position, • intensity,spread_angle(random)	Lec6: Modelling world with objects(page 29) Lec3:Arrays and Plotting Colours: (page 52) Random:(page 54)
ii	Spread of light as a cone-	Line -110	Setting different values to parameters	Matplotlib-documentation
2	**light_group.py**			
i	Grouping of lights and modifying it	LightGroup class Line -14	Group 1 with even number lights and group 2 with odd number lights	
3	**smoke_machine.py**			
i	SmokeMachine-class-with position,direction,intensity	Line 10	Tested with each neighbourhoods separately and combinely	Lec5:Files and Grids-Heat Diffusion Formula,Research articles
ii	Diffuse method			Practest3-BubbleMachine
iii	Update method	Line 34-update smoke	Tested with different colours,different values of alpha and size	
4	**Props.py**			
i	Props class definition	Line 8	Defined for position,shape,size and color	
ii	Create_patch()	Line 21	Setting circles and rectangles for creating props	Practest2
iii	Event handlining	Line 94	Used on_click event to play the music	Web development reference
iii	Animation	Line 72	Update method ,tested with True/False values to halo	
iv	Background sound	Line 33	Pygame.mixer () used:tested with audio files	Online resources

5	**backdrop.py**		Using scipy.ndimage different modification on image tried.	Lec2:Multidimensional arrays and functions(page 44
i	Grayscale/Toggle grayscle	Line 12,23		
ii	Sub-sample image	Lines 15,16		
iii	Image rotation	Lines 76-79		
6.	**choreography.py**	Different settings	Explained in showcase	
7	**Spinal_tap.py**		Try with different input values	
i	Display and handle choreography menu	Lines 7-22		
i	Call animation object	Line 60		

DISCUSSION

Plot

Created the same size plot for all the independent files and the main **spinal_tap.py**. The plot consists of **two subplots**. Plot **ax0** represents the stage view, while plot **ax1** represents the main features of the visualisation.

Light

The Light class represents a single light source with **color, position, intensity, direction, spread_angle, and speed attributes**. Each light is represented as an object of a Light class and with all the objects created a list called "lights".

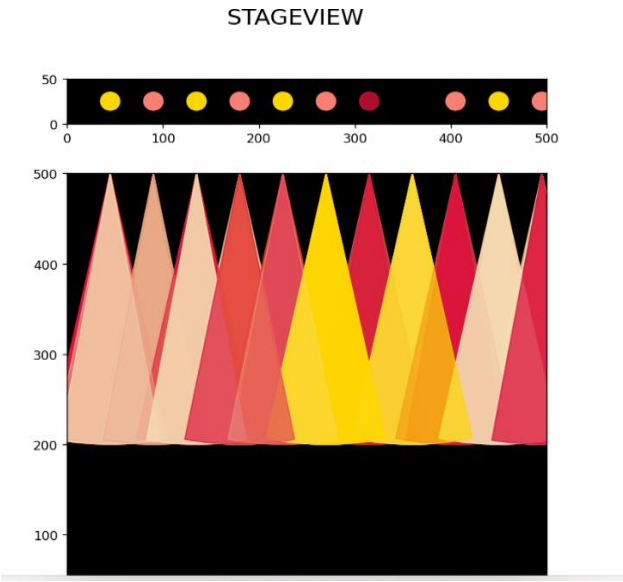
'Color' is represented as a string and 'position' is represented as a tuple of two values[x,y], which defines the light's coordinates in the plot. Intensity determines the brightness of the light and its value is kept between (0-11) using 'random' and spread_angle is used to represent light spread using cones. Each light shines through a cone with a specified spread angle, which determines the coverage area. Define the beam width of the light and speed is a numerical value to set the speed of light movement.

Light Group

It is an extension of lights.py. It also contains the **Lights class**. It was created to group lights for simulation. It created 2 groups with ***odd and even number lights***, changed their properties, and verified that they worked.

Light.py and light_group.py use a while loop to iterate and simulate changes in light's position, colours, spread angle, intensity, etc. The plot is set to be updated with each iteration and

iteration numbers can be changed with the conditions. these iterations and randomisation of intensity, position ,spread angle create an animation effect on the output.



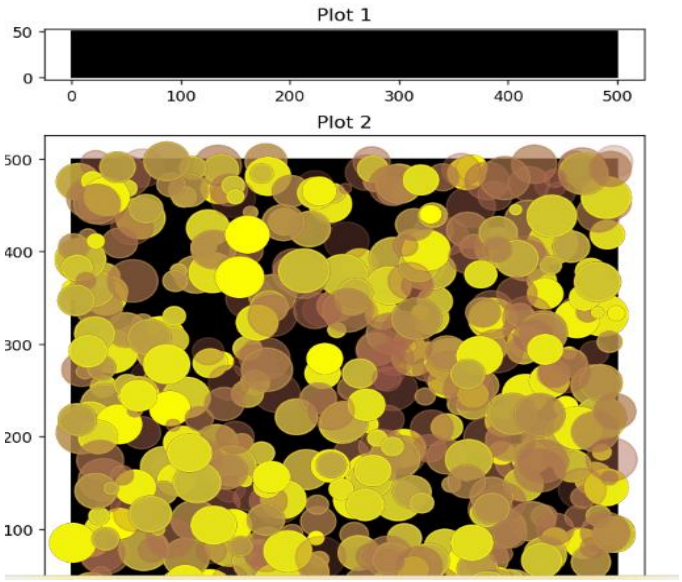
Smoke Machine

The smokeMachine class represents properties such as **position, direction, and intensity**. It also defines two **neighborhood schemes**. Moor and Von Neuman. The diffuse method diffuses smoke in the grid based on the chosen neighborhood. Random values of x and y are generated to simulate the machine's movement. Using the update method inside the SmokeMAchine class, x, and y values are updated for each instance. This method diffuses smoke in the grid using a neighborhood scheme.

- To code the smoke machine used two neighbourhood options for smoke diffusion: Moore and Von-Neumann. Depending on the neighbourhood choice, neighbouring cells are considered for smoke diffusion. The neighbourhood type is passed as an argument to the update method of the SmokeMachine class.
- Light and smoke interaction is not provided since it is independently implemented. However, the smock machine will diffuse when it interacts with lights.
- modeled smoke diffusion in a grid-based system. The diffuse method of the SmokeMachine class implements smoke diffusion. Using a grid as input, the method updates the smoke concentration in neighbouring cells according to the neighborhood type (Moore or Von-Neumann). The diffusion process is governed by the intensity of the smoke machine.
- In the code, smoke diffusion and visualisation occur simultaneously within the main loop. Each iteration of the loop updates the smoke machine and scatters points representing smoke particles onto the plot. To visualize the simulation in real time, plt.pause(0.001) introduces a small delay between each iteration.
- Timesteps for the smoke simulation are determined by the desired behaviour and speed of the diffusion process. The simulation speed can be controlled by adjusting the pause duration (plt. pause()).

Another implementation of smoke machine

In the final plot, smoke machine was plotted using the idea from the Practest3.



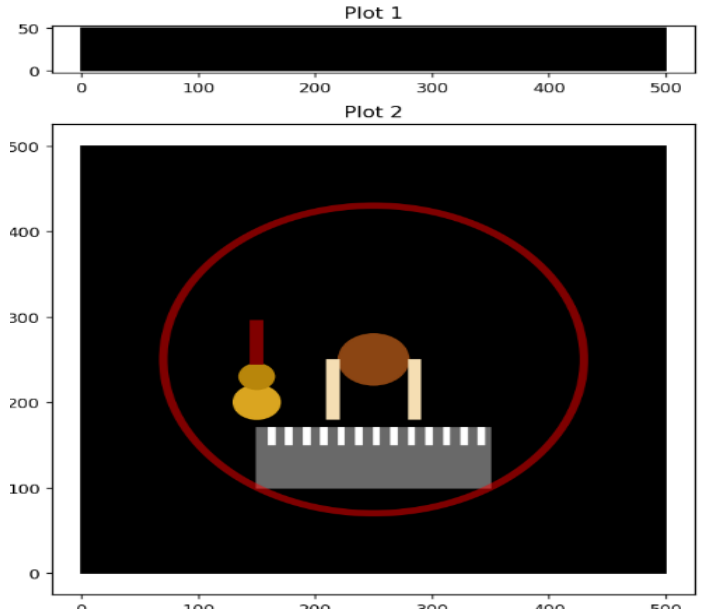
Props

The props code implements a dynamic and interactive animation using the **Pygame** and Matplotlib libraries. It introduces the concept of "props," which are objects displayed on the plot with attributes such as position, shape, size, and color. The Prop class creates these props using circles and rectangles, initializing them with their respective attributes. The **create_patch()** method generates the graphical representation of each prop on the plot.

Additionally, the code incorporates a halo effect represented by a circle_patch. Initially, the halo is visible and added to the plot. The update() function plays a crucial role in animating the props. It iterates through each prop, calling their update() method to modify their positions or implement other animations. Furthermore, every 20 frames, the halo's visibility is toggled, and its position is set to a specific prop. This combination of prop movements and the dynamic halo effect contributes to engaging and visually appealing animation.

To enhance the overall experience, background music is included. The Pygame library is initialized, and a sound file is loaded using Pygame.mixer.music.load(). This enables the **playback of music** in the background while interacting with the plot. Specifically, the **on_click() function** serves as a **callback for mouse click events**. It checks if a click occurs within the plot area and responds by playing the associated music with the clicked prop using pygame.mixer.music.play().

To create the animation, the code employs the FuncAnimation class. This class takes the figure, the update() function, a sequence of frames (ranging from 0 to 360 with steps of 5), and an interval of 50 milliseconds between frames. The animation is automatically updated and displayed using plt.show(). This combination of Pygame and Matplotlib functionalities provides a captivating and interactive animation experience .



Backdrop

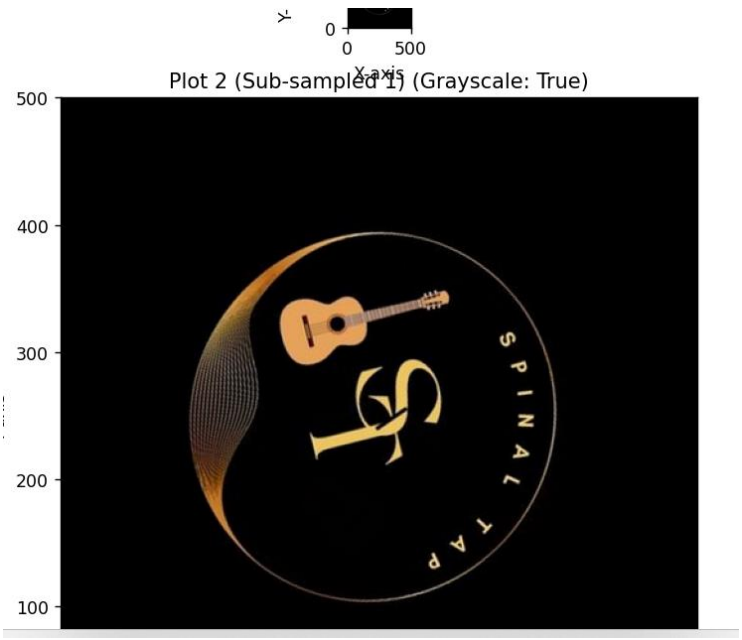
In this **scipy.ndimage used** for image manipulation. The plt.imread() function is used to load an image file called "logojpg.". The dimensions of the image are stored in **stage_height** and **stage_width**.

By using the dot product of the RGB values with specific coefficients, the loaded image is converted to a grayscale. The image is subsampled into two versions, pixel_face, and pixel_face2, by selecting every 10th and 50th pixel.

Initial parameters are set: **is_grayscale** is set to **False** to indicate that the image is initially displayed in color, and **alpha** is set to 0.5 as the initial transparency value. Two functions are defined: **toggle_grayscale()** toggles the **is_grayscale** flag between **True** and **False**, and **update_alpha()** increments the **alpha** value by 0.1 and resets it to 0.0 if it exceeds 1.0.

The **update_figure()** function updates the backdrop image, grayscale/color, and alpha values in each iteration. It selects the appropriate image and colormap based on the **is_grayscale** flag, and updates the subplots accordingly.

A loop is executed for a specified number of iterations (in this case, 10). In each iteration, the **toggle_grayscale()** function is called to switch between grayscale and color display. The **update_alpha()** function is called to increment the transparency value. The image is rotated by a random angle using **scipy.ndimage.rotate()**. The rotated image is then converted to grayscale using the same method as before.The **update_figure()** function is called again to update the figure with the revised settings and rotated image. The figure is redrawn, and there is a pause of 0.5 seconds between iterations with **plt.pause()**.



SHOWCASE

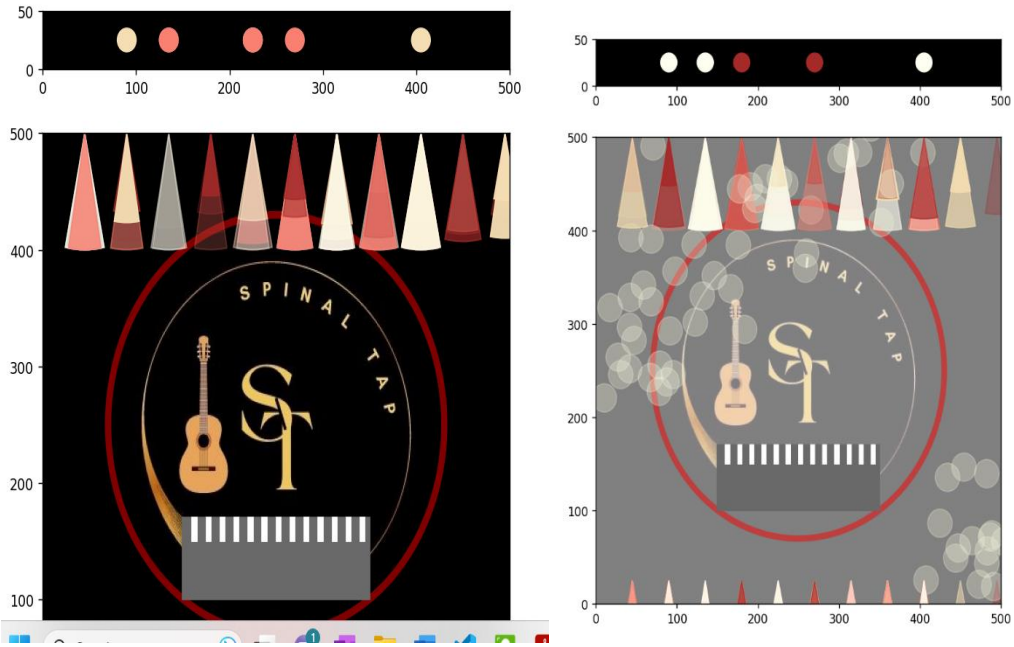
When the 'spinal_tap.py' file is implemented, a menu will appear. The user can choose the choreography number from that menu.

1st choreography

In the first scenario, the **logo image** is used as the stage backdrop. Two sets of lights one with even number lights and the other with **odd number lights** and the **intensity** of these groups is set in a range of **(0-11)** using a random function and **spread_angle** set between (**10,12**). The **x-axis** of the Group -1 lights will change continuously with **45 units**, and when it reaches 495, it will return to the (45,0) position. **The y-axis** also changes between **500 and 25**, which produces reflection lights on the bottom of the screen. As for the **cone** property, the radius is set between **(50,100**) using random variables. Group-1 lights alternate between wheat and brown colors.

For **Group-2 lights**, the x-axis will change. However, the **y-axis** will not change. The cone radius is fixed at 100. The effect will help to create another moving effect for the lights.

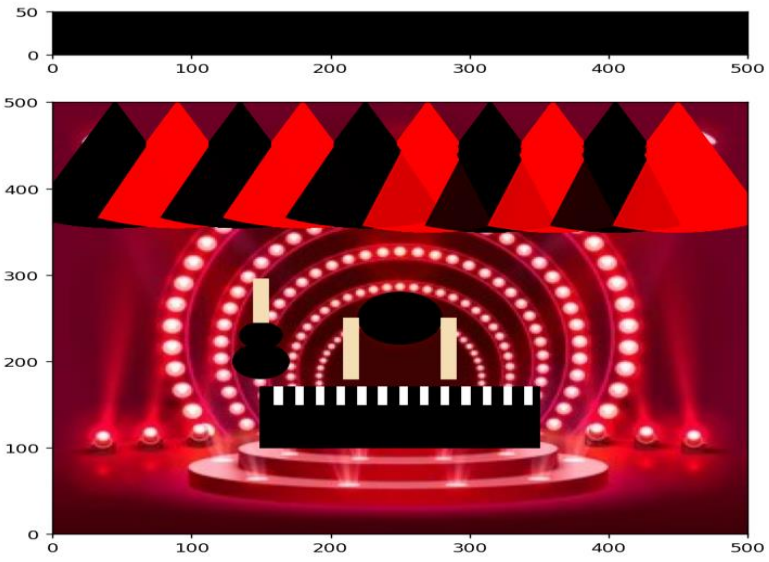
In respect of **props**, the **halo** is set to **TRUE** value and the **piano** has also been added. And **two smoke machines** are included in this; they are starting at (**0,200)and (400,00**), which indicates two corners of the stage.



2nd Choreography

Groups 1 and 2 are rearranged in the second scenario. Group 1 consists of the first five lights and group 2 consists of the remaining five lights. Set the intensity of the lights between (8-11) and set the spread_angle (10-25). The color of the lights are set to red and black. The cone radius of the first set of lights is 145 whereas the cone radius of the second set is 150.

There is no smoke machine in this choreography, and for the backdrop, one image is added without a halo effect. For the props, guitar and drums have been added, and piano has been retained. Halo effect has been removed from the props, and background sound has been added using Pygame.

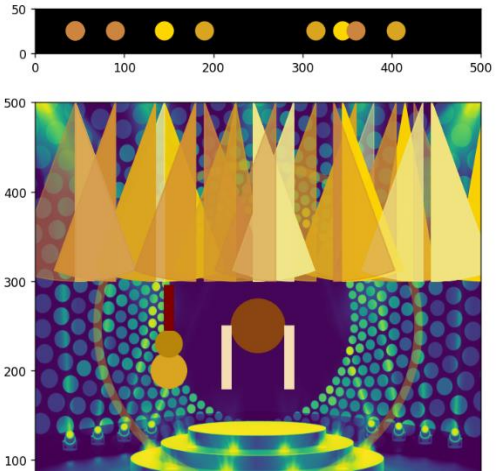


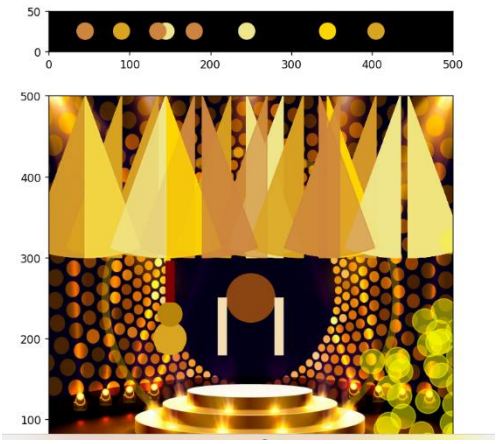
3rd Choreography

The lights are divided into two groups, **odd and even**. Set the **intensity** of lights in the range (0-11) using random numbers and set the spread_angle to a common value of 10 for all. The intensity value here is used as the alpha value when creating the **Wedge** patches in the **update_frame** function. This affects the patches' transparency, making them more or less opaque based on the intensity value. The lights are set to a similar **colour scheme of yellow and gold**. Even lights have been redirected from **270 degrees to 280 degrees**. For odd lights, the angle is set at **260 degrees**.

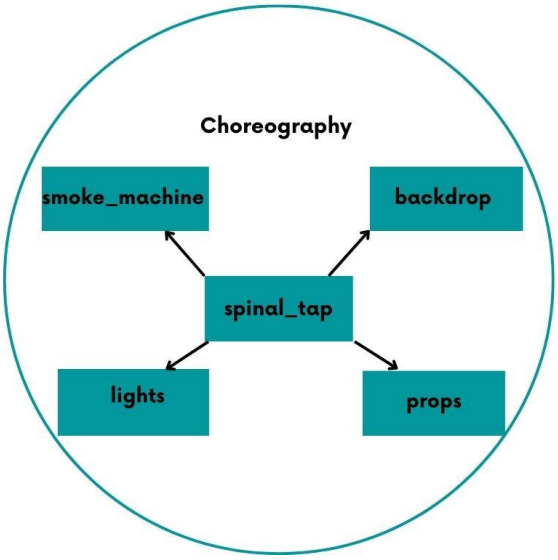
A smoke machine is added to the choreography, starting at position (0,400), and its colour is changed to yellow, and the **alpha value has been changed from 0.3 to 0.5** in order to make it more visible. When it comes to props, The piano was not included and the halo's colour was changed to gold. Its **line width** was increased to 7 from 5 and its Alpha value was reduced to 0.3 from 0.5.

For the backdrop first set the lights in the greyscale image then switch to the normal image.





UML USE CASE DIAFRAM



CONCLUSION

This program successfully created a concert stage for Spinal Tap. Among its main components are lights, smoke machines, props/bands/backdrops that help to create a variety of functionalities on stage. It provides the user with a visually refreshing experience. It can be considered one of the best examples of how Python can be applied to real-life scenarios and solve real-life problems, as well as how to simplify large problems into smaller ones. As a student, I revised the portions I studied in the first semester and acquire solid Python skills through this course. The knowledge I gained from this assessment will help to achieve future milestones in the course.

FUTURE WORK

This assessment leaves many possibilities open. Limited knowledge and time were a huddle to achieve it fully. It has been tried several times to read the choreography from a CSV or JSON file but it did not work in the end. As for the simulation part, if I have a chance in the future, I will try to implement it in 3D and will add more functionalities and animations.

REFERENCES

1.Classes-Python 3.8.4rc1 documentation. (n.d.). Docs.python.org.

https://docs.python.org/3/tutorial/classes.html

2. Lecture 6: Modelling world with objects (page 29)

3. Lecture 3: Arrays and Plotting
Colours: (page 52)
Random:(page 54)

4.Play sound in Python. (2021a, January 13). GeeksforGeeks.

https://www.geeksforgeeks.org/play-sound-in-python/

5. Wedges
https://matplotlib.org/stable/api/_as_gen/matplotlib.patches.Wedge.html

6.Smoke Machine:
Practical Test3-Bubble Machine
Lec5:Files and Grids-Heat Diffusion Formula,Research articles

7.Backdrop
Lec2:Multidimensional arrays and functions(page 44)