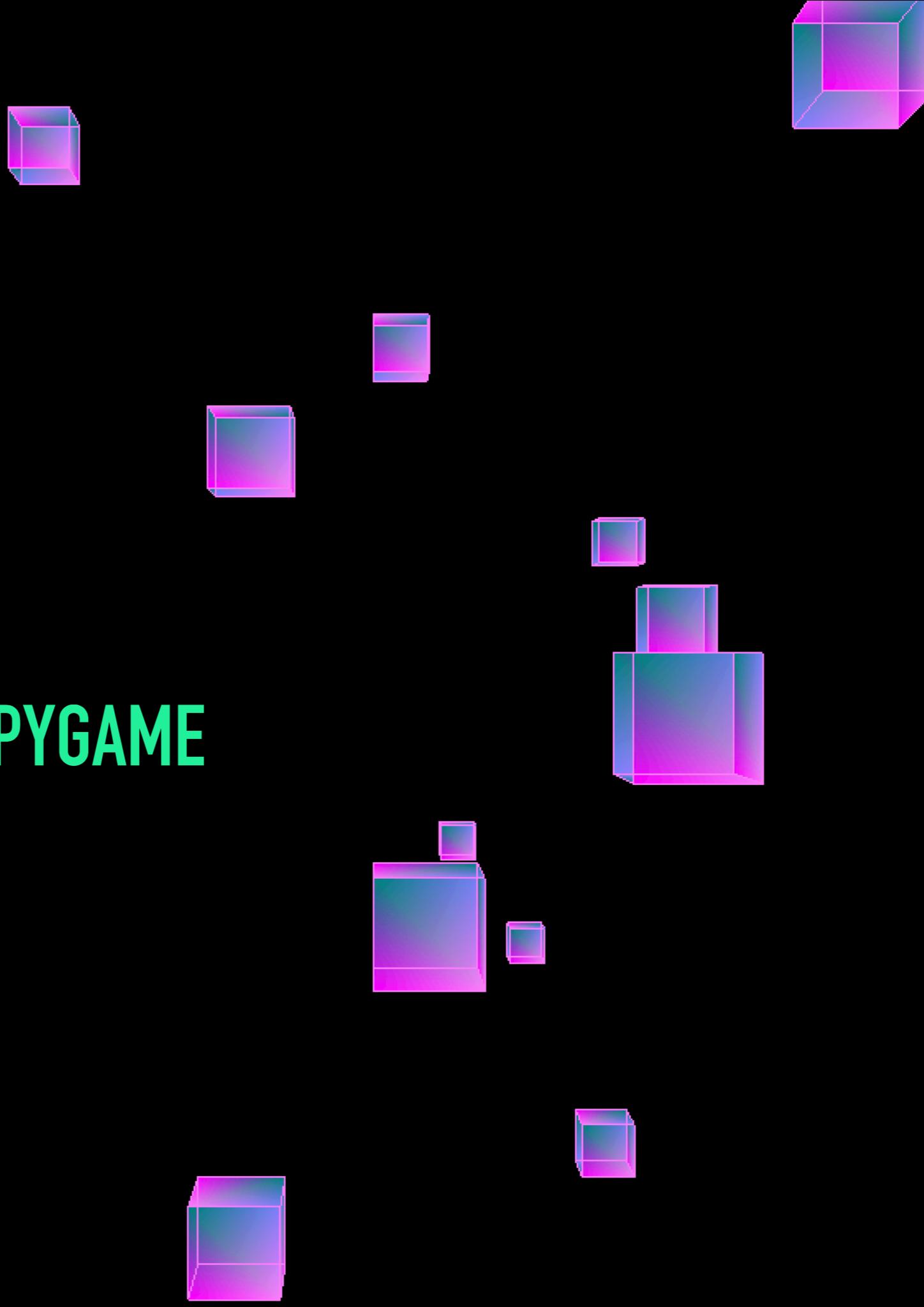


# INTRO TO PYGAME



- ▶ Pygame is an open source game development library to create real-time graphics-based games.
- ▶ You can find the standard documentation here <https://www.pygame.org/>
- ▶ Pygame comes preinstalled on the Raspberry Pi!

## Install Pygame

- ▶ type this in your Terminal / Powershell (not the interactive Python shell)

```
pip install pygame
```

## Import Pygame

```
import pygame
```

## Initialise Pygame

```
pygame.init()
```

## Create a window screen

```
width = 1000  
height = 1000
```

```
screen = pygame.display.set_mode((width,height))
```

## The main game loop

```
import pygame
from pygame.locals import QUIT ← import the QUIT constant
from sys import exit ← import the exit function from the sys module

pygame.init()

width = 1000
height = 1000
screen = pygame.display.set_mode((width,height))

running = True
while running:
    for e in pygame.event.get():
        if e.type == pygame.QUIT:
            running = False

pygame.quit() ← uninitialized all Pygame modules, release all resources
exit() ← exit the program
```

## Controlled frame rate (recommended in case of high CPU load)

```
import pygame
from pygame.locals import QUIT
from sys import exit

pygame.init()

clock = pygame.time.Clock()

width = 1000
height = 1000
screen = pygame.display.set_mode((width,height))

running = True
while running:
    for e in pygame.event.get():
        if e.type == QUIT:
            running = False
    clock.tick(30)

    pygame.quit()
    exit()

    # amount of maximum frames per second
```

Read:

<https://www.pygame.org/docs/ref/time.html>

Find the code here:

[https://github.com/hylynyiv/AlgorithmicArt/  
blob/main/Boilerplate.py](https://github.com/hylynyiv/AlgorithmicArt/blob/main/Boilerplate.py)

## Inserting an image

```
import pygame
from pygame.locals import QUIT
from sys import exit

pygame.init()
clock = pygame.time.Clock()
width = 1000
height = 1000
screen = pygame.display.set_mode((width,height))

img = pygame.image.load("Images/cat.png")

running = True
while running:
    for e in pygame.event.get():
        if e.type == QUIT:
            running = False
    screen.blit(img, (0,0))
    pygame.display.update()
    clock.tick(30)

pygame.quit()
exit()
```

**Note:** You can technically place the instructions to render the image *outside* the game loop. This is because a static image doesn't need to be updated in every frame. However, for typical game development everything should go inside the game loop.

```
import pygame
from pygame.locals import QUIT
from sys import exit

pygame.init()
clock = pygame.time.Clock()
width = 1000
height = 1000
screen = pygame.display.set_mode((width,height))

img = pygame.image.load("Images/cat.png")
screen.blit(img, (0,0))
pygame.display.update()

running = True
while running:
    for e in pygame.event.get():
        if e.type == QUIT:
            running = False
    clock.tick(30)

pygame.quit()
exit()
```

Read:

<https://www.pygame.org/docs/ref/image.html>

Find the code here:

<https://github.com/hylynyiv/AlgorithmicArt/blob/main/InsertImage.py>

## Drawing a line

```
screen = pygame.display.set_mode((800, 600))
```

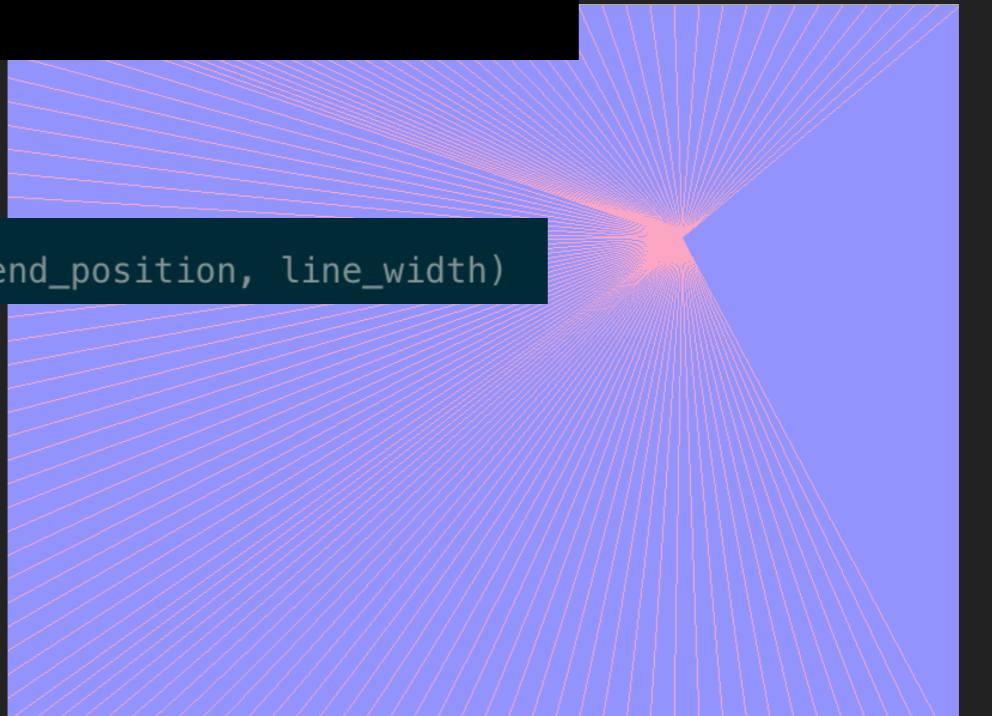
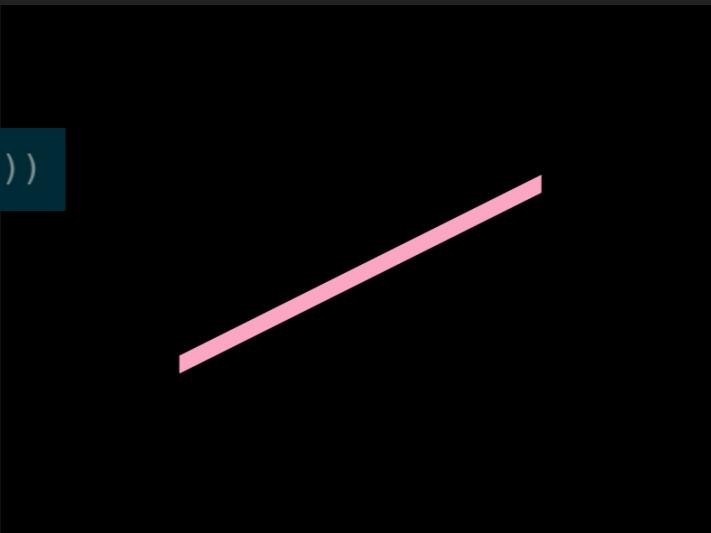
```
color = (255, 170, 200)
```

```
start_position = (200, 400)  
end_position = (600, 200)
```

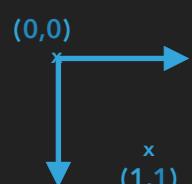
```
line_width = 20
```

```
pygame.draw.line(screen, color, start_position, end_position, line_width)
```

```
pygame.display.update()
```



The origin of the coordinates system is in the upper left corner extending downwards to the right



Note: Use draw.aaline to draw an antialiased line.

## Drawing a rectangle

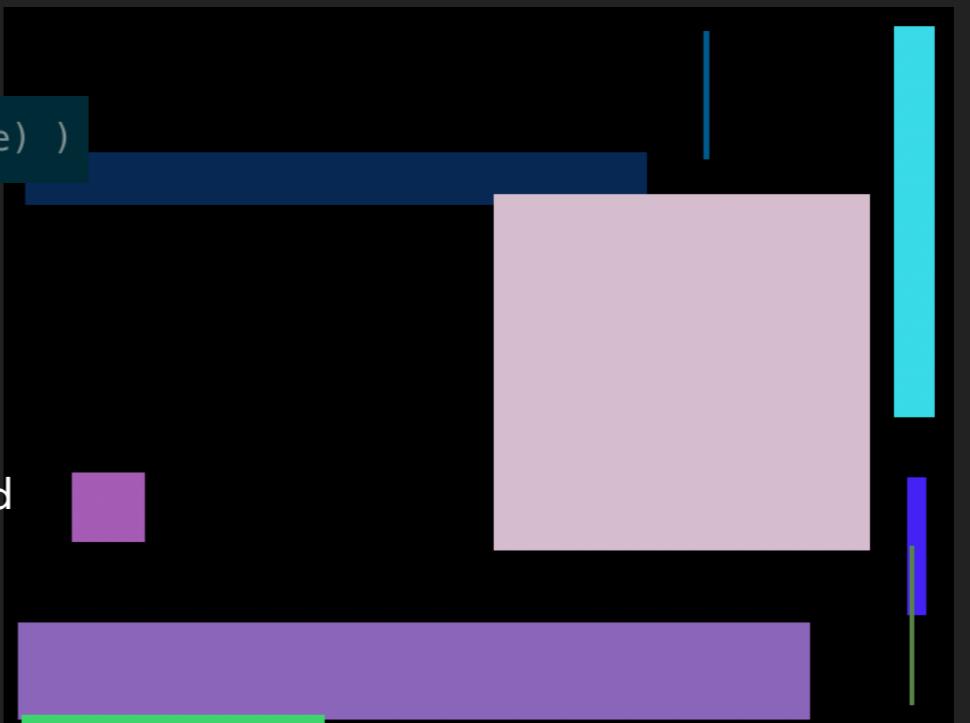
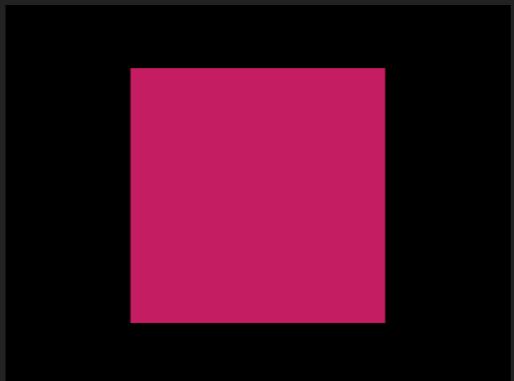
```
screen = pygame.display.set_mode((800, 600))
```

```
color = (200,30,100)
x,y = 200, 100
position = (x,y)
width = 400
height = 400
size = (width, height)
```

```
pygame.draw.rect(screen, color, (position, size) )
```

```
pygame.display.update()
```

Note: You can also use `surface.fill()` to create filled rectangles. This will take less computational resources and can be hardware accelerated.



1. a.) Create a program that generates rectangles of random colors, sizes and positions.  
Make sure the rectangles are never cropped but contained entirely within the limits of the screen.
1. b.) Make this an animation where rectangles of different colors, sizes and positions are created newly in every frame.
1. c.) Research about the various additional "flags" of the `display.set_mode()` function. Remove the border and title bar from the display. Find out how to close your program with the key "q".

## Drawing a polygon

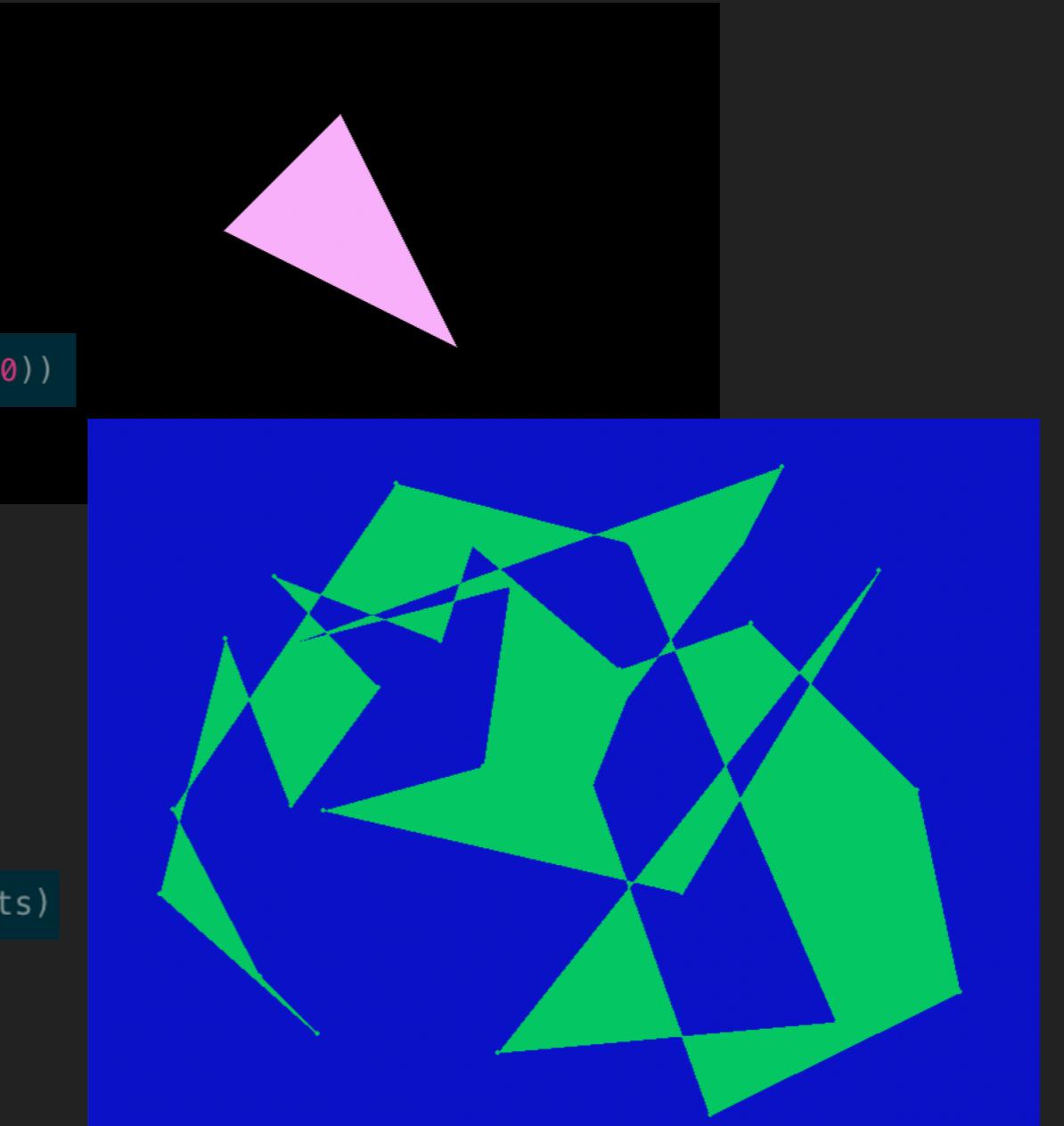
```
screen = pygame.display.set_mode((800, 600))
```

```
color = (250, 180, 250)
```

```
pos1 = (200, 200)
pos2 = (300, 100)
pos3 = (400, 300)
points = [pos1, pos2, pos3]
```

```
pygame.draw.polygon(screen, color, points)
```

```
pygame.display.update()
```



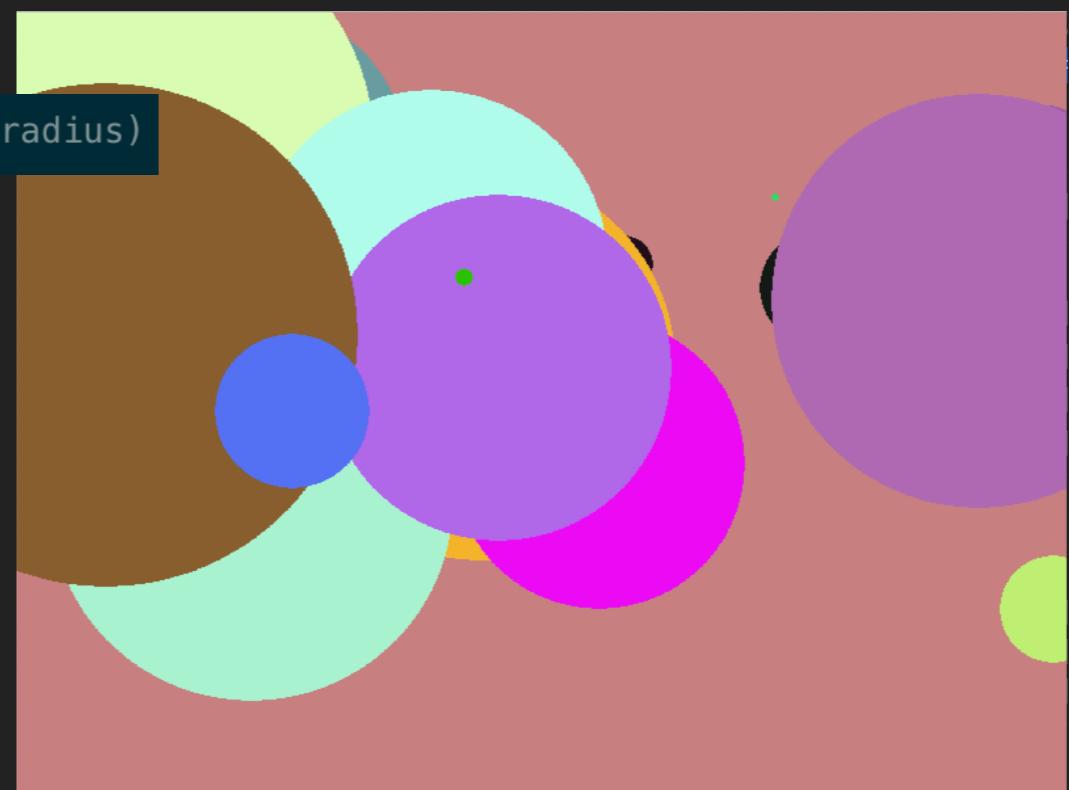
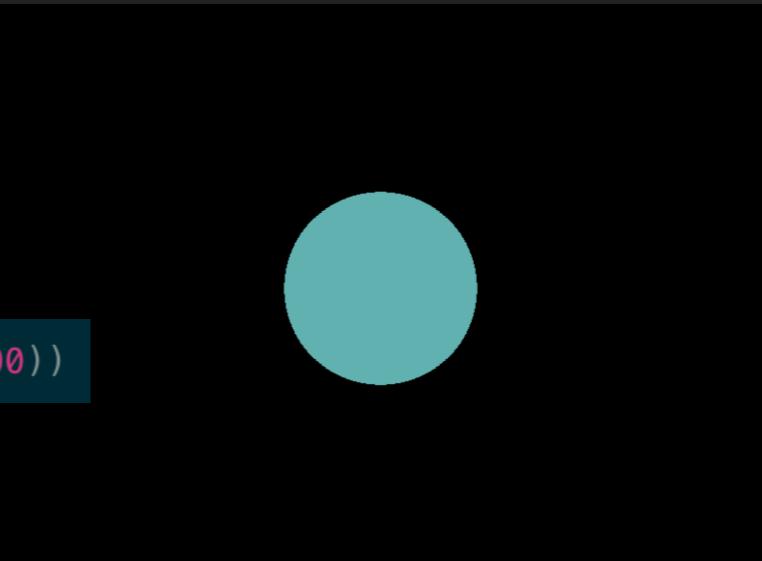
## Drawing a circle

```
screen = pygame.display.set_mode((800, 600))
```

```
color = (100, 180, 180)  
position = (400, 300)  
radius = 100
```

```
pygame.draw.circle(screen, color, position, radius)
```

```
pygame.display.update()
```



## Drawing an ellipse

```
screen = pygame.display.set_mode((800, 600))

color = (180, 180, 110)

position = (200, 250)
width = 400
height = 100
ellipse_rectangle = ( position, (width, height) )

pygame.draw.ellipse(screen, color, ellipse_rectangle)

pygame.display.update()
```



## Drawing an arc

```
from math import pi
```

```
screen = pygame.display.set_mode((800, 600))
```

```
color = (180, 180, 110)
rectangle = (200, 200, 400, 200)
start_angle = 0
stop_angle = pi
```

```
pygame.draw.arc(screen, color, rectangle, start_angle, stop_angle)
```

```
pygame.display.update()
```



## Drawing multiple lines

```
screen = pygame.display.set_mode((800, 600))

color = (255, 170, 200)

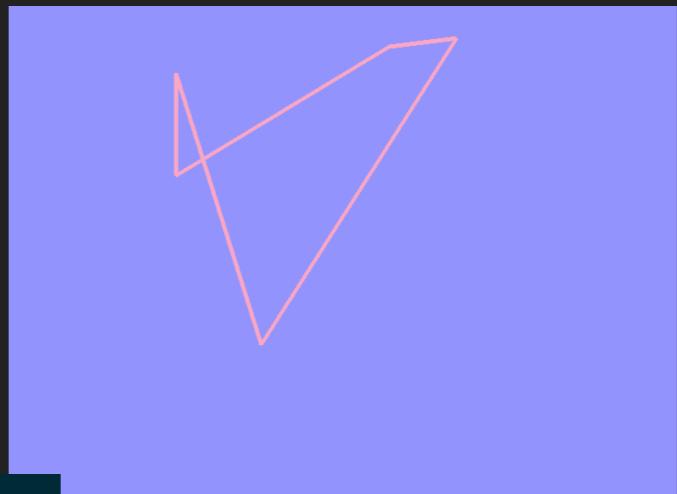
connect_points = True

points = [(200,200), (452,48), (530,39),(300,400), (200,80)]

line_width = 20

pygame.draw.lines(screen, color, connect_points, points, line_width)

pygame.display.update()
```



Note: Use `draw.aalines` to draw antialiased lines.

2. Research about the mouse module in the Pygame documentation.  
Write a program that picks up the cursor's current position and  
draws an object at these coordinates.

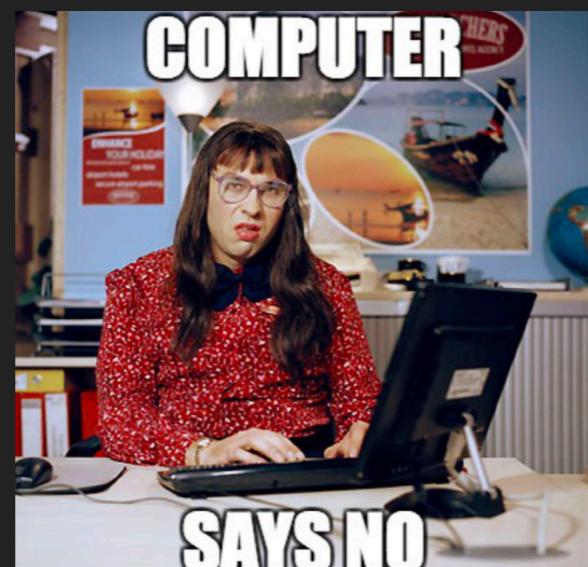
## Setting and getting pixel values

```
image = pygame.image.load("images/tobe1.jpg")
image_size = image.get_rect().size #get image size of the loaded image

screen = pygame.display.set_mode((image_size))

for y in range(image_size[1]):
    for x in range(image_size[0]):
        pix_color = image.get_at((x,y))
        screen.set_at((y,x), (pix_color))
```

3. Load an image of your choice. Invert the colors. Save it as a new picture.
4. Load an image of your choice. Find a pattern of your choice to move pixels regularly from one position to another in the whole image.
5. Use `get_at()` and `set_at()` to move pixels of an image to another position during the game loop. Use random values and reposition the pixels in a way so the original structure is destroyed.



result of exercise 3

## Move with keys

```
x, y = 0, 0
move_x, move_y = 0, 0

while True:

    if event.type == KEYDOWN:
        if event.key == K_LEFT:
            move_x = -1
        elif event.key == K_RIGHT:
            move_x = +1
        elif event.key == K_UP:
            move_y = -1
        elif event.key == K_DOWN:
            move_y = +1

    elif event.type == KEYUP:
        if event.key == K_LEFT:
            move_x = 0
        elif event.key == K_RIGHT:
            move_x = 0
        elif event.key == K_UP:
            move_y = 0
        elif event.key == K_DOWN:
            move_y = 0

    x+= move_x
    y+= move_y

    screen.fill((0,0,0))
    screen.blit(background, (x,y))

    pygame.display.update()
```

# Adding text

## Defining parameters

```
myfont = pygame.font.SysFont("futurattc", 19)

mytext = "Hello TechBasics Fans"
aliasing = False
fontcolor = (0,0,255)
bgcolor = (0,255,0)
```



```
myfont = pygame.font.Font("OpenSans-Regular.ttf", 19)
```

Use  
**font.Font()**  
for custom fonts. (Save the .ttf-file in the same folder as your script.)

## Render surface

```
mysurface = myfont.render(mytext, aliasing, fontcolor, bgcolor)
```

## Save to image

```
pygame.image.save(mysurface, "pygameTextToImage.png")
```

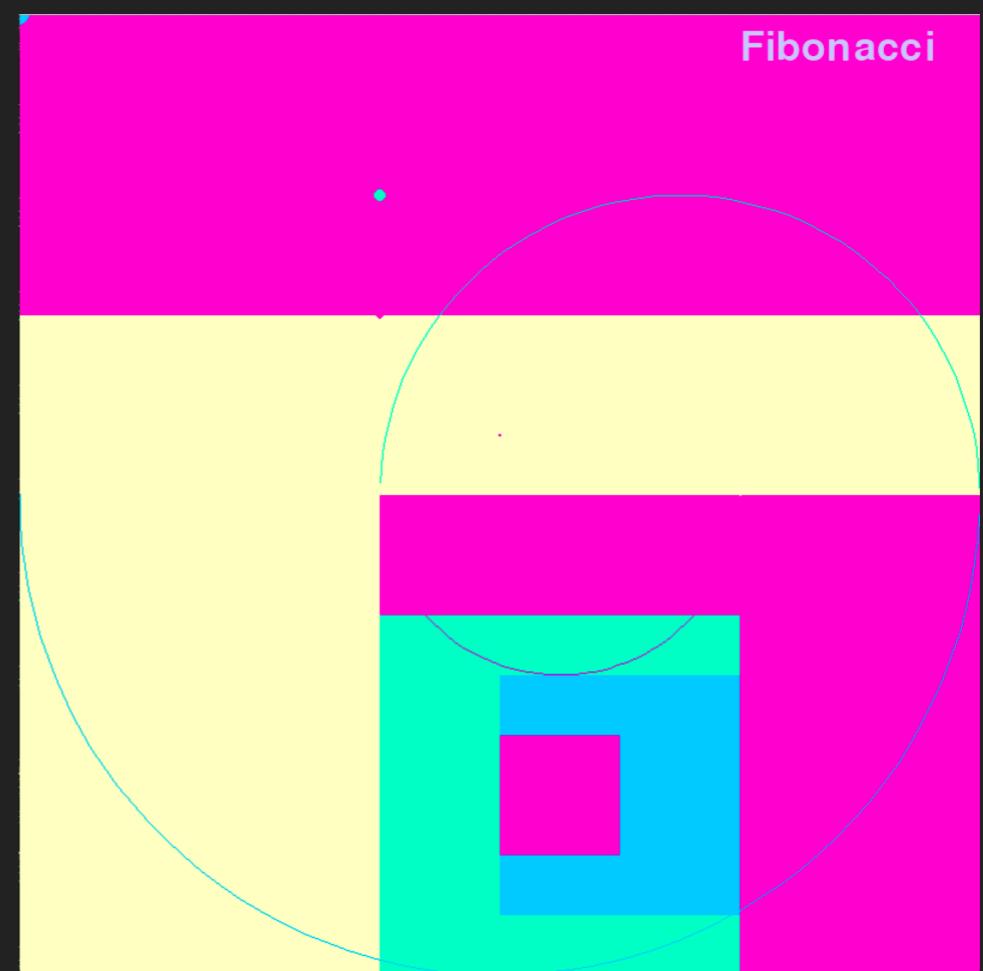
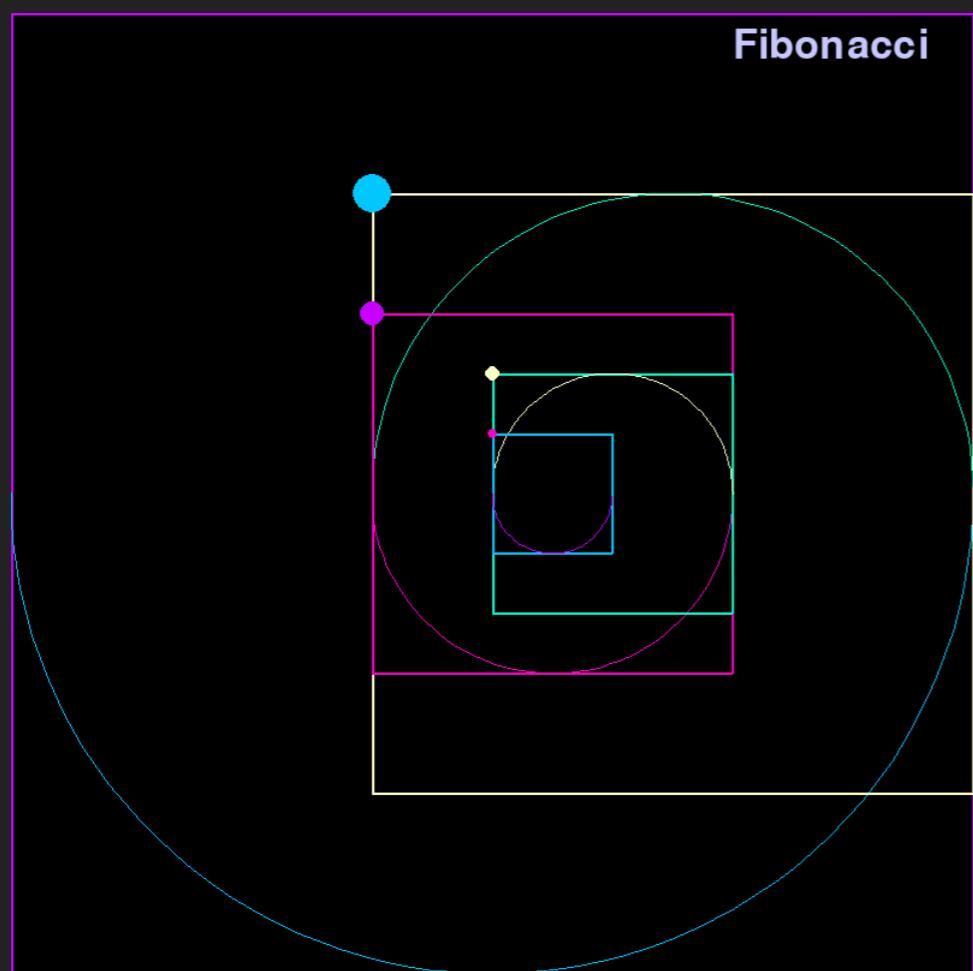
## Display on screen

```
gamewindow = pygame.display.set_mode((800,600))

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            exit()

    gamewindow.blit(mysurface, (0,0))
    pygame.display.update()
```

6. Research about the Fibonacci series and use it to draw geometrical forms. Use the ratios for the shapes of circles and rectangles.
7. Use another number sequence and create an artwork with it.  
Research on <http://oeis.org>.



## Simple Music Player

```
from pygame import *
from pygame.locals import *
from sys import exit

init()
clock = time.Clock()

screen = display.set_mode((300,100), NOFRAME)
display.set_caption("Music Player")
screen.fill((150,150,250))

while True:
    for e in event.get():
        if e.type == QUIT:
            quit()
            exit()
        if e.type == KEYDOWN:
            if e.key == K_q:
                quit()
                exit()

    #The actual audio player
    mixer.music.load("Sound/NyanCat.mp3")
    mixer.music.play()

    #Button to pause
    draw.rect( screen, (0,100,200), ((30, 30),( 50, 50)) )
    #Button to play
    triangle = [ (110, 30), (110,80), (160, 55) ]
    draw.polygon( screen, (0,200,100), triangle )

    display.update()

    #wait for mouse to get clicked over button location
    x,y = mouse.get_pos()
    if mouse.get_pressed()[0]:
        if (30 < x < 80) and (30 < y < 80):
            mixer.music.pause()
        elif (110 < x < 160) and (30 < y < 80):
            mixer.music.play()

    clock.tick(60)
```

## Installing MoviePy for Conda:

```
conda install -c conda-forge moviepy
```

alternative installation modes:

<https://anaconda.org/conda-forge/moviepy>

## Minimal Video Player:

```
from pygame import *
from moviepy.editor import *

clip = VideoFileClip("Videos/raccoon1.mp4")
clip.preview()

quit()
```

## Installation of OpenCV for Anaconda

```
conda install -c conda-forge opencv
```

instructions for alternative installation methods:

<https://anaconda.org/conda-forge/opencv>

## Documentation and useful links

<https://docs.opencv.org/>

<https://opencv-python-tutroals.readthedocs.io/en/latest/index.html>

<https://www.geeksforgeeks.org/python-play-a-video-using-opencv/>

```
import cv2

cap = cv2.VideoCapture("Videos/raccoon1.mp4")

if cap.isOpened() == False:
    print("Error opening file.")

cap.set(cv2.CAP_PROP_FPS, 25)

while cap.isOpened():

    ret, frame = cap.read()
    if ret == True:
        cv2.imshow("Frame", frame)

        if cv2.waitKey(10) & 0xFF == ord("q"):
            break

    else:
        break

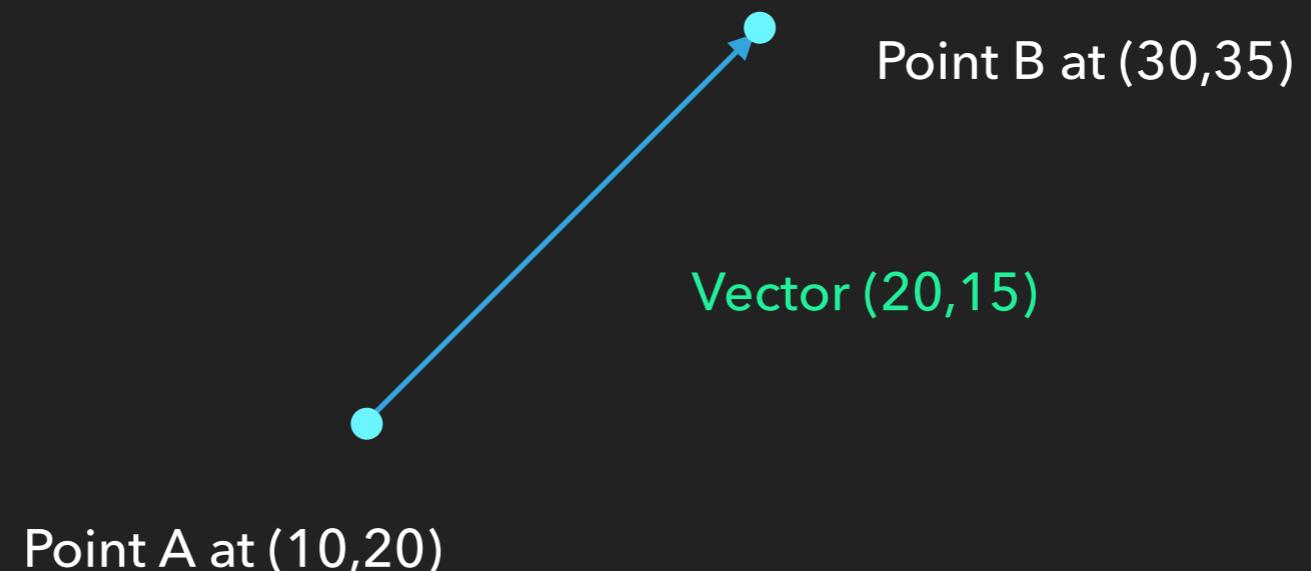
cap.release()

cv2.destroyAllWindows()
```

## Vectors:

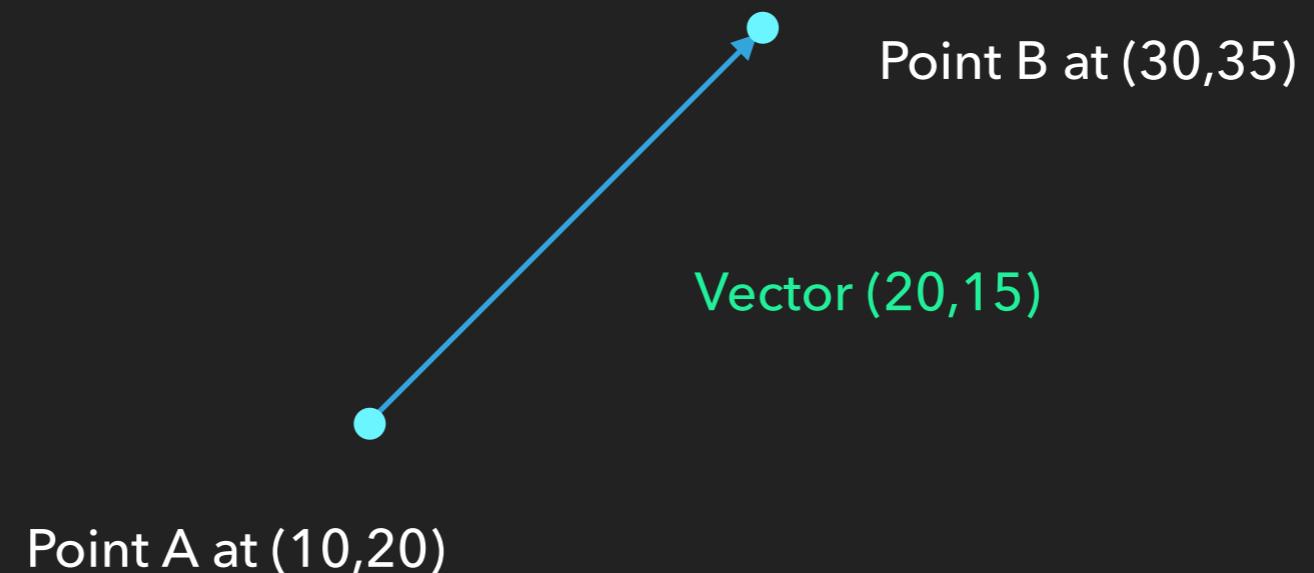
are mathematical representations of a **directional** change.

A vector is the difference between the coordinates of point A and B.



The **magnitude** of a vector is the distance between two points:

$$\text{Vector AB length} = \sqrt{20*20+5*5} = 25$$



A vector can be **normalized** to express its unit vector, which has no information about the magnitude and is often used to represent the heading.

A unit vector has always a length of 1. We obtain it by dividing the components by the magnitude.

Vector (20,15)

$$\rightarrow 20/25 = 0.8$$

$$\rightarrow 15/25 = 0.6$$

Unit Vector (0.8,0.6)

## Installation of OpenGL for Python

```
pip install PyOpenGL PyOpenGL_accelerate
```

instructions for alternative installation methods:  
<http://pyopengl.sourceforge.net/>

## Documentation

<https://www.khronos.org/registry/OpenGL-Refpages/gl4/>  
<https://www.opengl.org//documentation/>

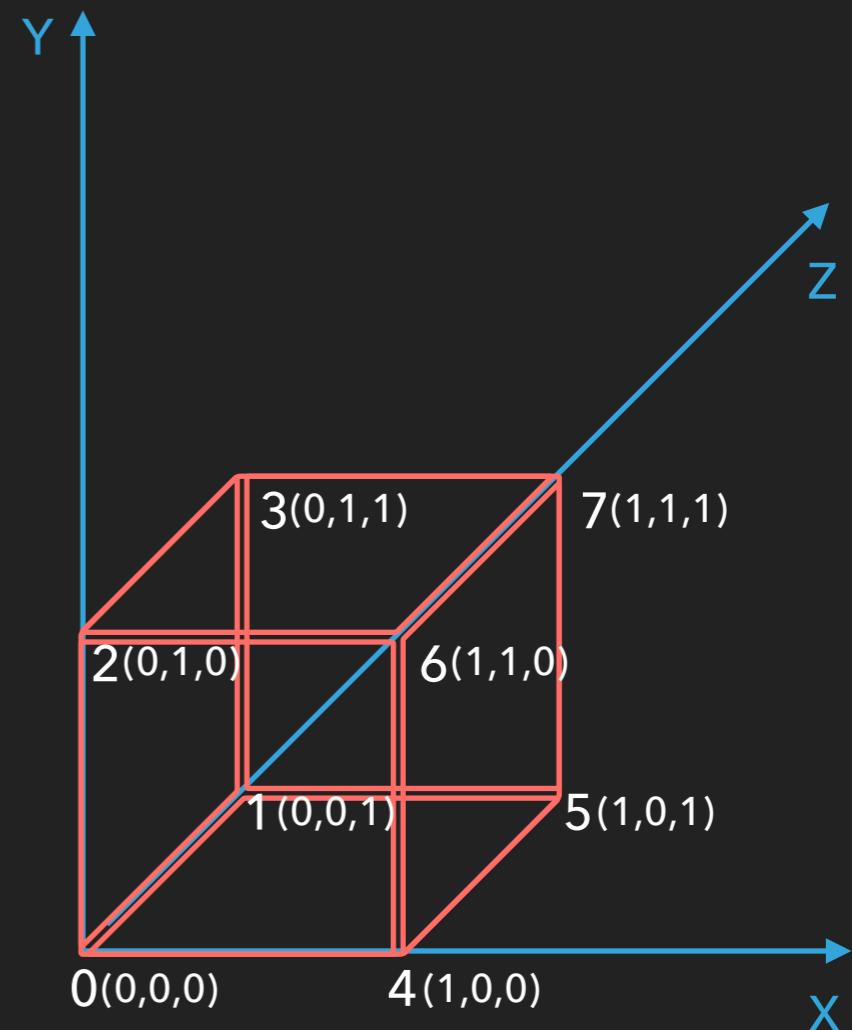
## Rotate a Cube

The nodes of 3-dimensional objects are also called "vertices" (singular: vertex).

Each vertex is defined by a tuple of 3 values, indicating their position in a 3D space.

A cube can be described by a set of 8 vertices, containing their coordinates on the X-, Y- and Z-axis, f.i.:

- 0 (0,0,0),
- 1 (0,0,1),
- 2 (0,1,0),
- 3 (0,1,1),
- 4 (1,0,0),
- 5 (1,0,1),
- 6 (1,1,0),
- 7 (1,1,1)

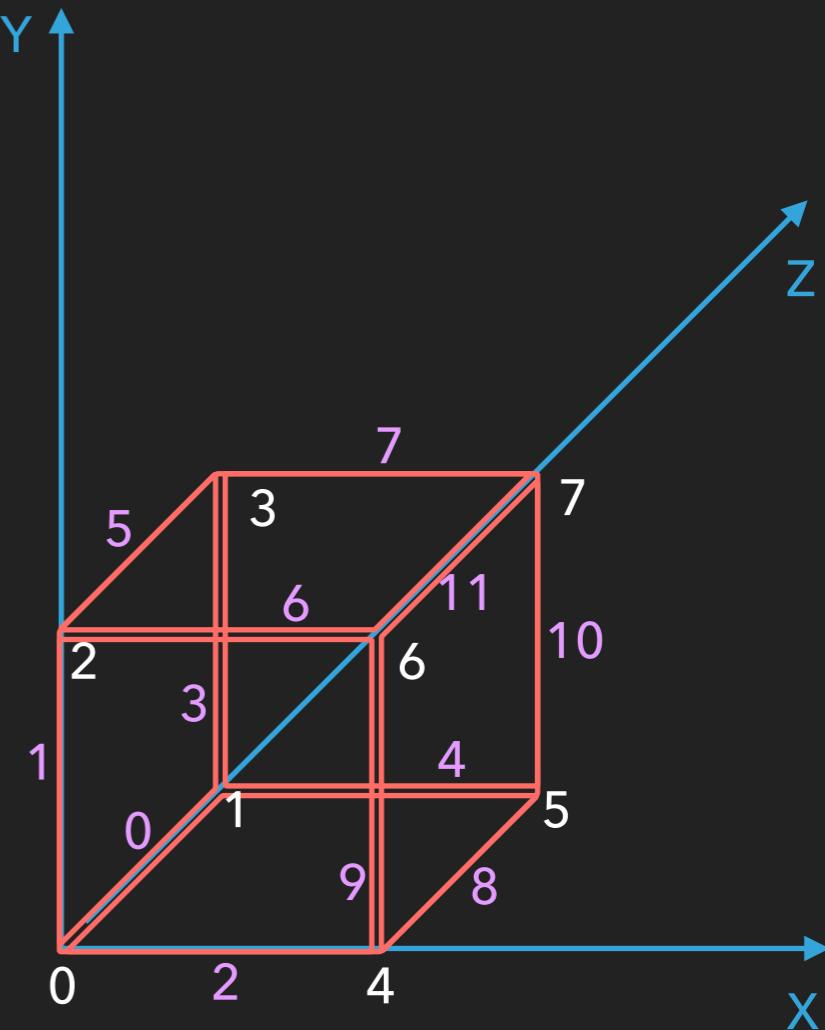


A 3D object is also defined by the way its nodes connect, i.e. by its edges. Every node connects to 3 other points.

A cube is defined by 12 edges.

We can specify which vertices connect with each other with a set of 12 tuples.

- 0 (0,1),
- 1 (0,2),
- 2 (0,4),
- 3 (1,3),
- 4 (1,5),
- 5 (2,3),
- 6 (2,6),
- 7 (3,7),
- 8 (4,5),
- 9 (4,6),
- 10 (5,7),
- 11 (6,7)



```
from pygame import *
from pygame.locals import *
from sys import exit

from OpenGL.GL import *
from OpenGL.GLU import *

init()
clock = time.Clock()
width = 1000
height = 800

screen = display.set_mode((width,height), DOUBLEBUF | OPENGL)

fieldOfView = 45
aspectRatio = (width/height)
clippingPlane_near = 0.1
clippingPlane_far = 50

gluPerspective(fieldOfView, aspectRatio, clippingPlane_near, clippingPlane_far)
```

```
vertices = (
    (0,0,0),
    (0,0,1),
    (0,1,0),
    (0,1,1),
    (1,0,0),
    (1,0,1),
    (1,1,0),
    (1,1,1)
)

edges = (
    (0,1),
    (0,2),
    (0,4),
    (1,3),
    (1,5),
    (2,3),
    (2,6),
    (3,7),
    (4,5),
    (4,6),
    (5,7),
    (6,7)
)
```

```
def Cube():
    glBegin(GL_LINES)
    for edge in edges:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()
```

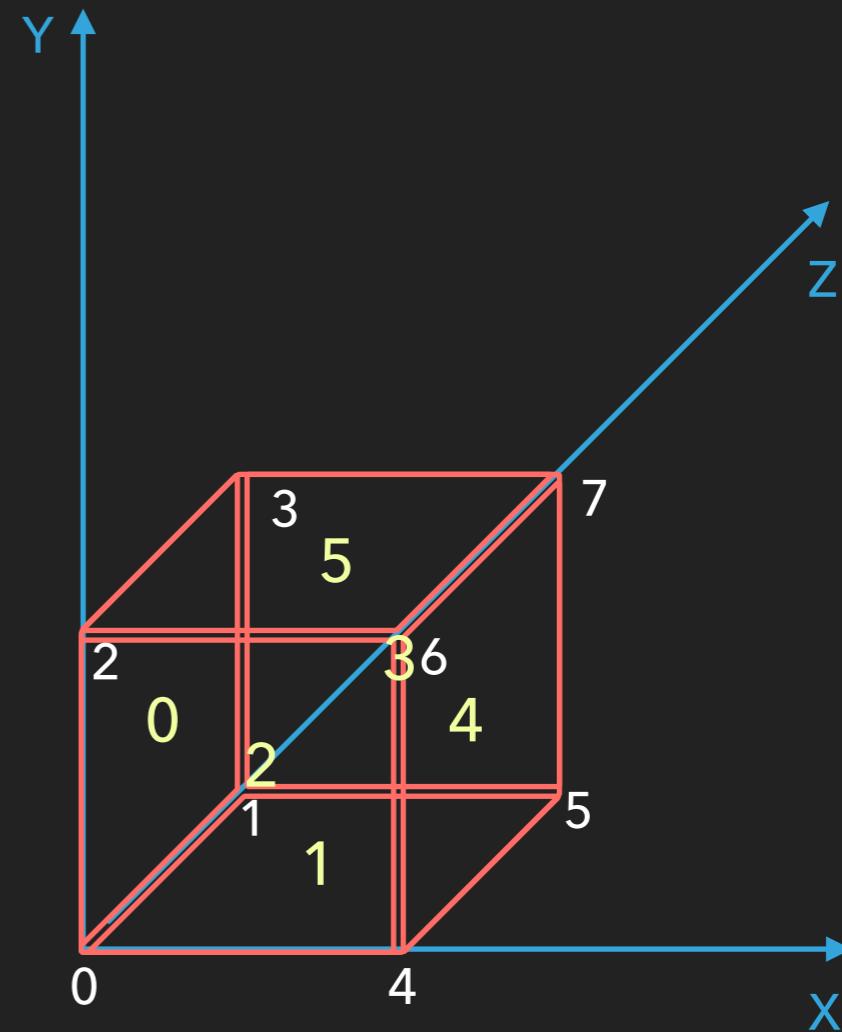
```
glTranslatef(0.0,0.0, -5)
```

```
while True:  
    for e in event.get():  
        if e.type == QUIT:  
            quit()  
            exit()  
  
        if e.type == KEYDOWN:  
            if e.key == K_q:  
                quit()  
                exit()  
  
    glRotatef(1,3,1,1)  
  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)  
  
    Cube()  
  
    display.flip()  
  
    clock.tick(60)
```

## Colouring the surfaces

A cube also consists of 6 surfaces. Every surface is defined by the 4 vertices it is composed of.

- 0 (0, 1, 3, 2),
- 1 (0, 1, 5, 4),
- 2 (0, 2, 6, 4),
- 3 (7, 3, 1, 5),
- 4 (7, 6, 4, 5),
- 5 (7, 6, 2, 3)

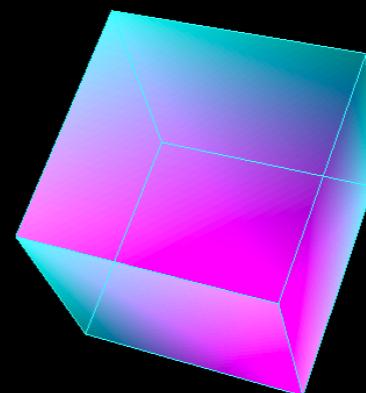


OpenGL colours are defined in between the range of 0 and 1.

```
colors = (
    (0.5,0.5,1),
    (0,0.5,0.5),
    (1,0,1),
    (1,0.5,1),
    (0.5,1,1),
    (0.5,1,1)
)
```

```
def Cube():
    glBegin(GL_QUADS)
    for surface in surfaces:
        x = 0
        for vertex in surface:
            x += 1
            glColor3fv(colors[x])
            glVertex3fv(vertices[vertex])
    glEnd()

    glBegin(GL_LINES)
    for edge in edges:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()
```



1. Experiment with the vertices to create various shapes.
2. Make an object move from left to right and reappear when it has passed over the right edge.