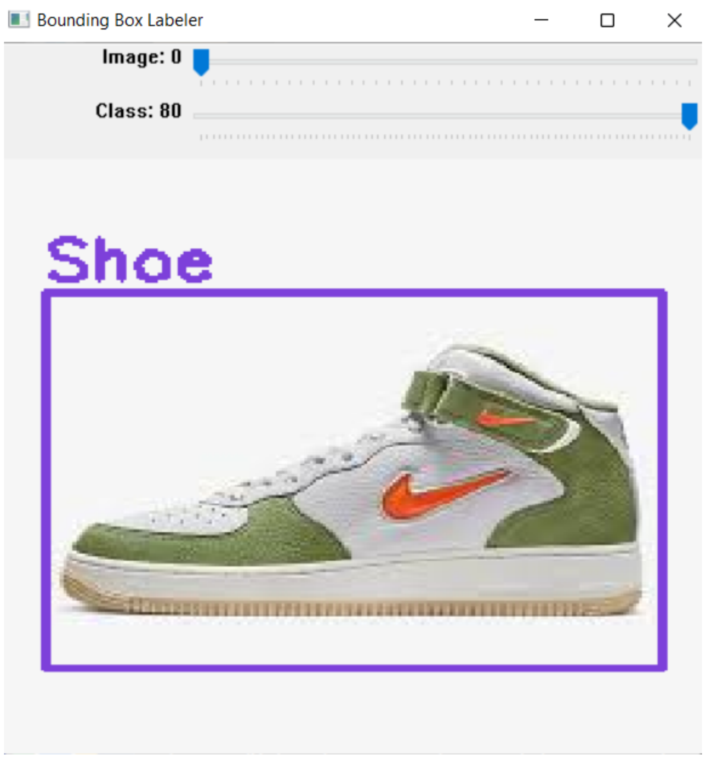
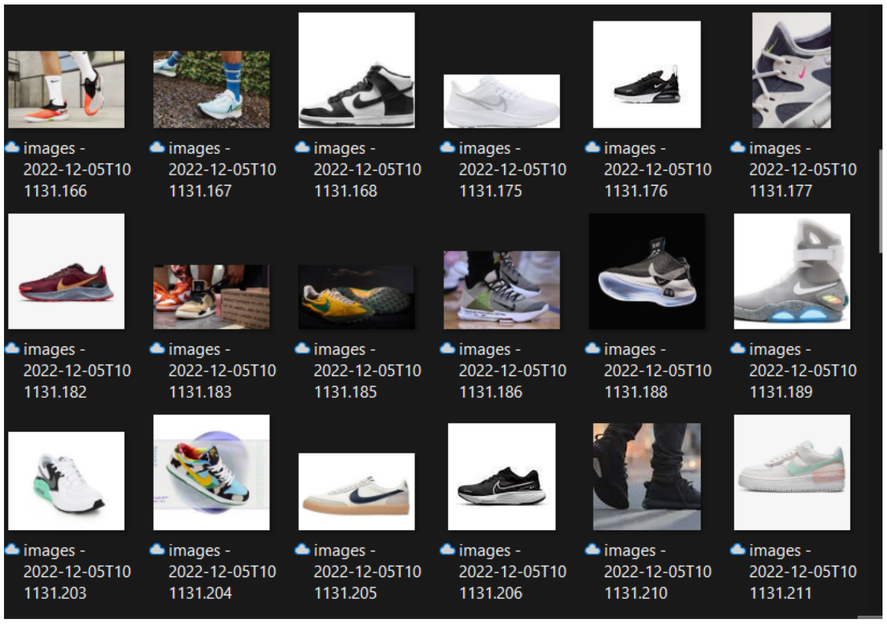
Lab8 EE104 Documentation

Names: Grant von Pingel
Kerolles Hermina

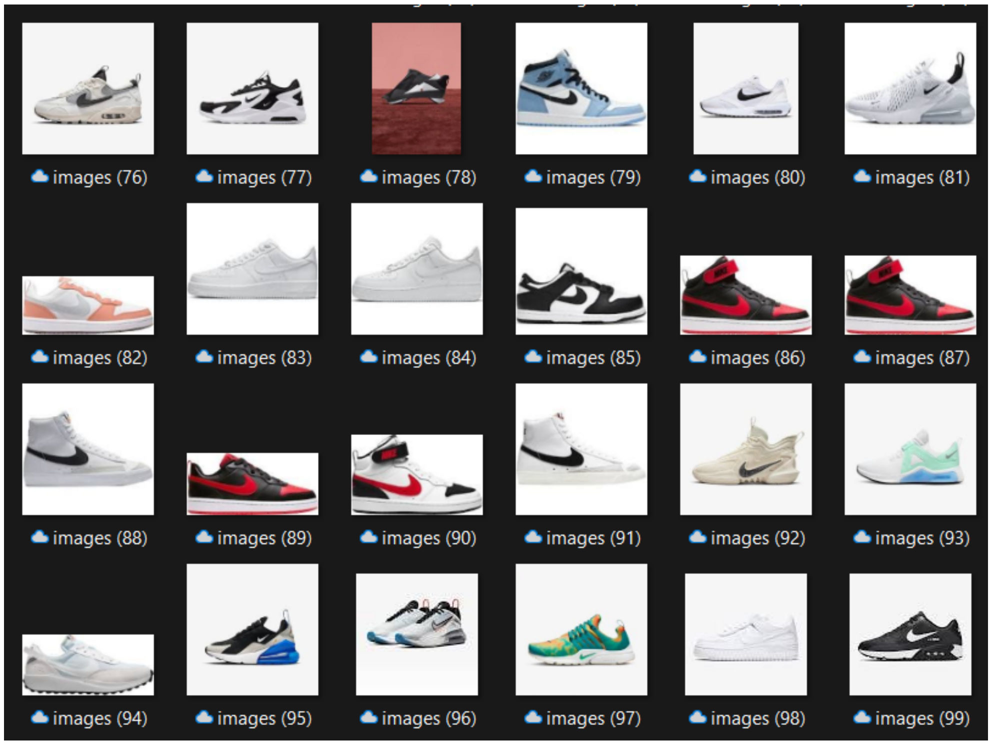
Part 1: Training New Class



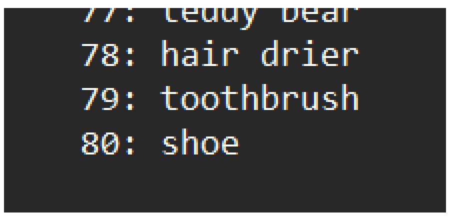
- Go through and label all new images for training as the designated class (shoe) and then take all of these files and transfer them to the YoloV5 folder


images - 2022-12-05T10 1131.166
images - 2022-12-05T10 1131.167
images - 2022-12-05T10 1131.168
images - 2022-12-05T10 1131.175
images - 2022-12-05T10 1131.176
images - 2022-12-05T10 1131.177
images - 2022-12-05T10 1131.182
images - 2022-12-05T10 1131.183
images - 2022-12-05T10 1131.185
images - 2022-12-05T10 1131.186
images - 2022-12-05T10 1131.188
images - 2022-12-05T10 1131.189
images - 2022-12-05T10 1131.203
images - 2022-12-05T10 1131.204
images - 2022-12-05T10 1131.205
images - 2022-12-05T10 1131.206
images - 2022-12-05T10 1131.210
images - 2022-12-05T10 1131.211

- Add around 200 new images to the training image folder

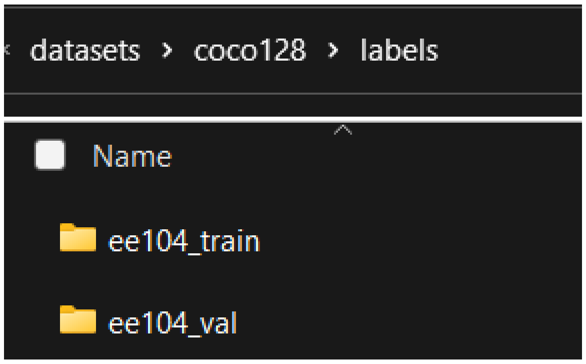


- Add some more images to the validation folder

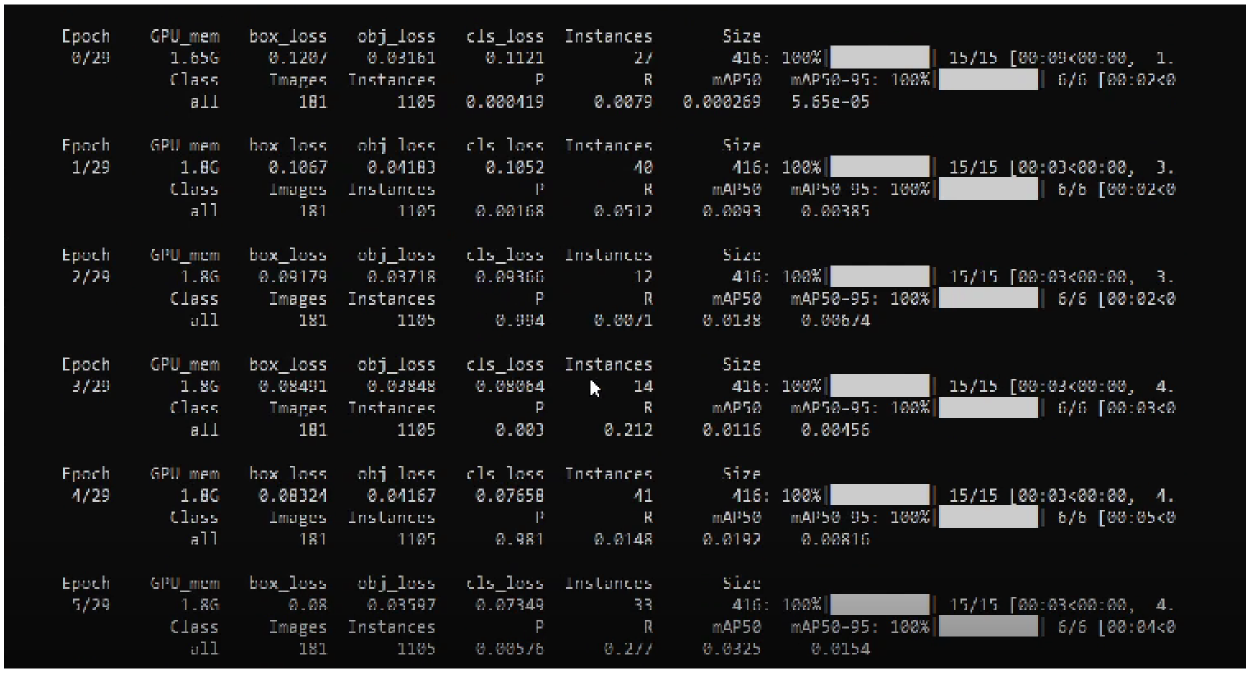


```
77: teddy bear
78: hair drier
79: toothbrush
80: shoe
```

- Add a new class to the class list in this case shoe



- Add all of the bounding box files into the label files for the training and validation



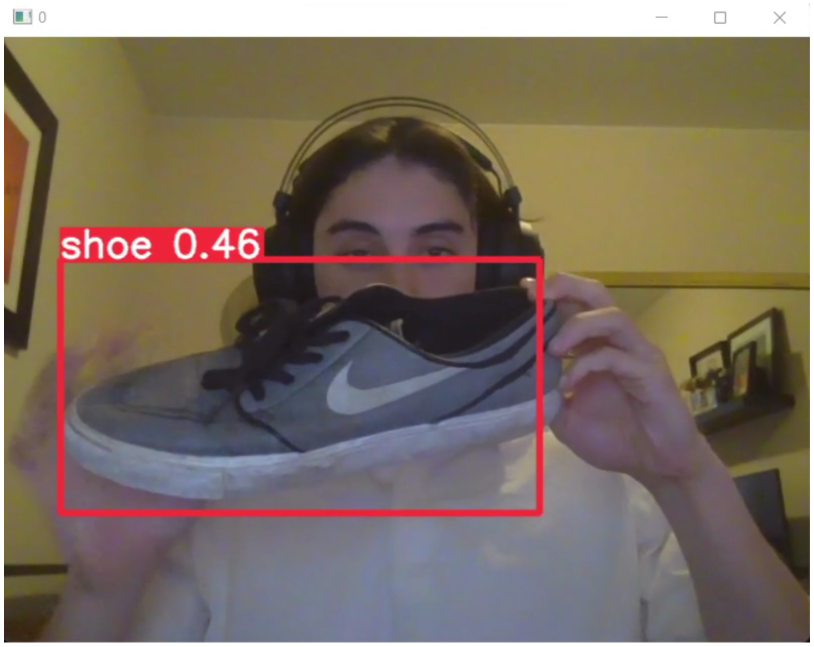
```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       0/29      1.65G     0.1207    0.03161     0.1121         27        416: 100%|██████████| 15/15 [00:05<00:00,  1.
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 6/6 [00:02<0
                   all        181       1105   0.000419     0.0079   0.000269   5.65e-05

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       1/29       1.8G     0.1067    0.04183     0.1052         40        416: 100%|██████████| 15/15 [00:03<00:00,  3.
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 6/6 [00:02<0
                   all        181       1105    0.00168     0.0512     0.0093    0.00385

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       2/29       1.8G    0.09179    0.03718    0.09366         12        416: 100%|██████████| 15/15 [00:03<00:00,  3.
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 6/6 [00:02<0
                   all        181       1105      0.994     0.0071     0.0138    0.00674

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       3/29       1.8G    0.08491    0.03848    0.08064         14        416: 100%|██████████| 15/15 [00:03<00:00,  4.
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 6/6 [00:03<0
                   all        181       1105      0.003      0.212     0.0116    0.00456

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       4/29       1.8G    0.08324    0.04167    0.07658         41        416: 100%|██████████| 15/15 [00:03<00:00,  4.
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 6/6 [00:05<0
                   all        181       1105      0.981     0.0148     0.0192    0.00816

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       5/29       1.8G       0.08    0.03597    0.07349         33        416: 100%|██████████| 15/15 [00:03<00:00,  4.
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 6/6 [00:04<0
                   all        181       1105    0.00576      0.277     0.0325     0.0154
```

- Training the model through 29 epochs to have the camera recognize the new class (shoe)
- Next, you can go to the camera where it can detect the new class

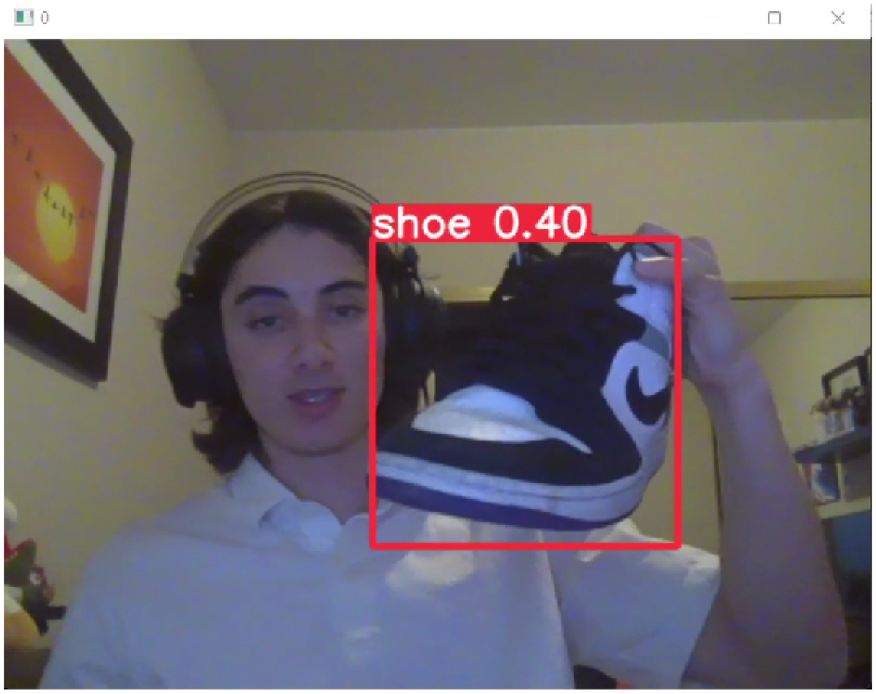
Part 2: Recognizing New Class

```
C:\yolov5>python detect.py --source 0 --weights C:\yolov5\runs\train\exp2\weights\best.pt
```

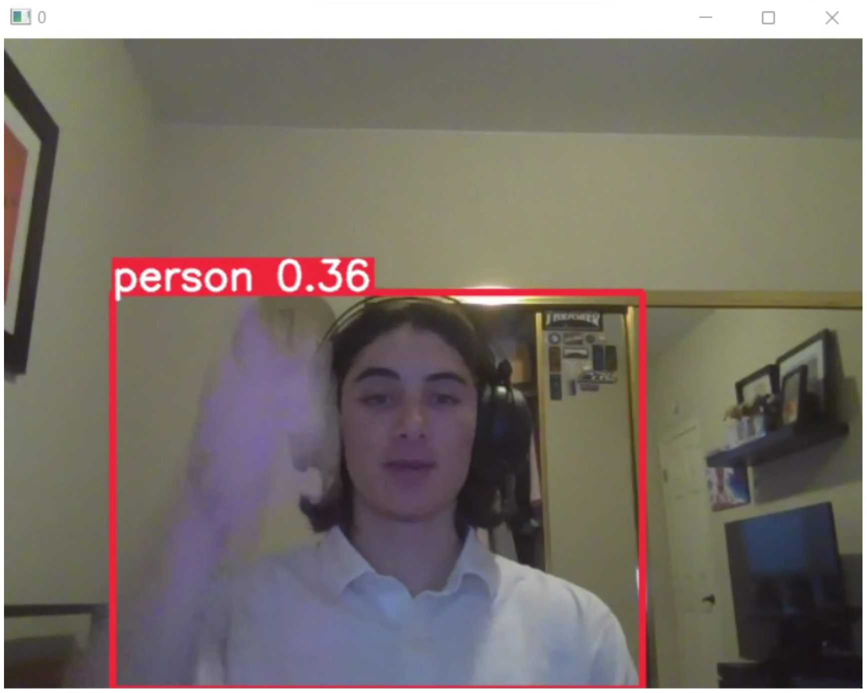
- Use this python code in the terminal for the Yolo folder to detect the camera and use the newly trained (shoe) class recognition.



- Showing the first instance of a shoe and having the program recognize it as the new class (shoe)

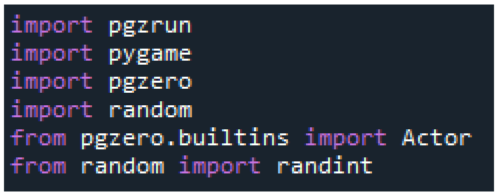


- Showing a second instance to show the program works on multiple types of shoes and it also shows how accurate it thinks its prediction is



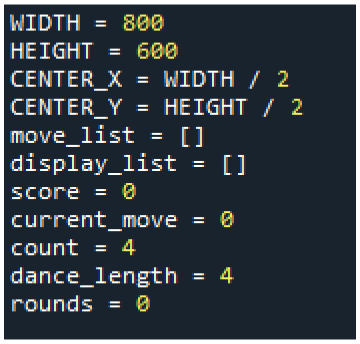
- Showing it still recognizes other classes even when the new class (shoe) is not present

Part 3: Game Development Dance Challenge



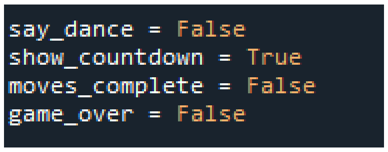
```
import pgzrun
import pygame
import pgzero
import random
from pgzero.builtins import Actor
from random import randint
```

- Import the different libraries for pygames and random



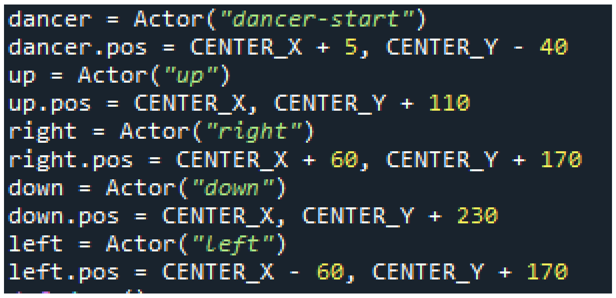
```
WIDTH = 800
HEIGHT = 600
CENTER_X = WIDTH / 2
CENTER_Y = HEIGHT / 2
move_list = []
display_list = []
score = 0
current_move = 0
count = 4
dance_length = 4
rounds = 0
```

- This dest the width and height of the screen while also calculating the center for x and y. It also sets the initial conditions for the different variables like rounds, count, and dance length



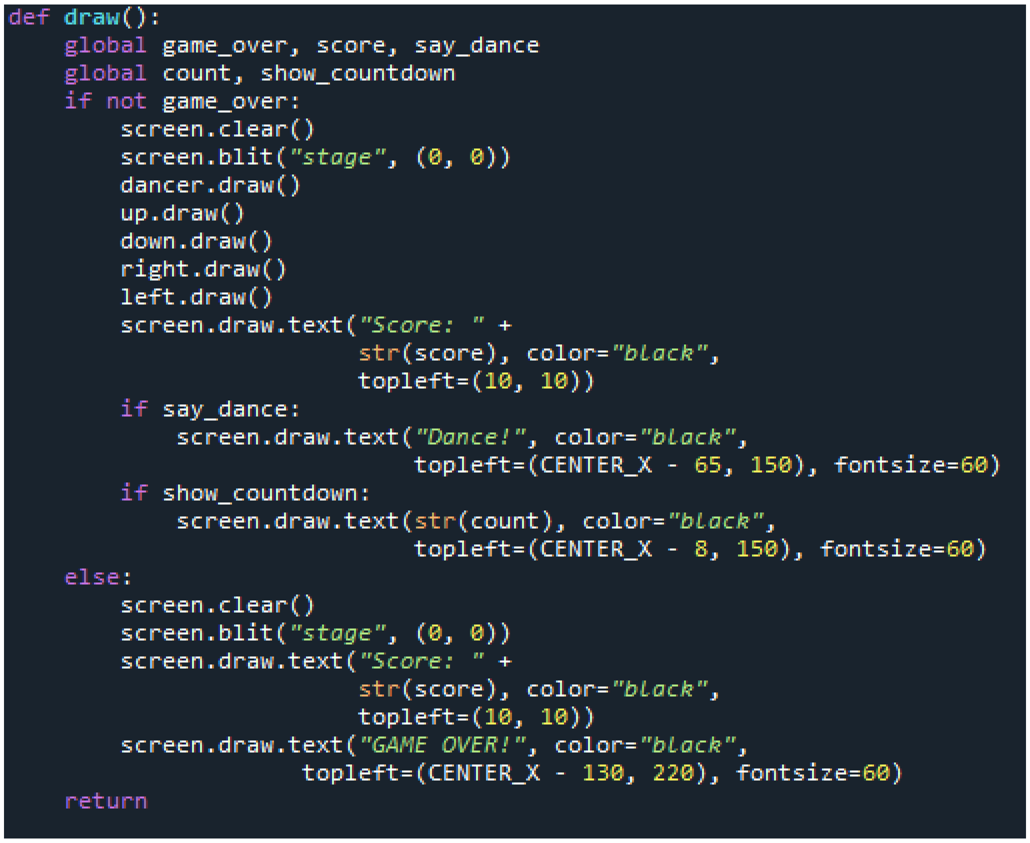
```
say_dance = False
show_countdown = True
moves_complete = False
game_over = False
```

- Sets up the initial states of the events like game over and moves completely that later will get triggered to become true or false which then runs another program.



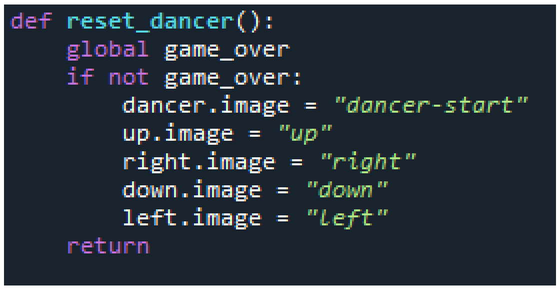
```
dancer = Actor("dancer-start")
dancer.pos = CENTER_X + 5, CENTER_Y - 40
up = Actor("up")
up.pos = CENTER_X, CENTER_Y + 110
right = Actor("right")
right.pos = CENTER_X + 60, CENTER_Y + 170
down = Actor("down")
down.pos = CENTER_X, CENTER_Y + 230
left = Actor("left")
left.pos = CENTER_X - 60, CENTER_Y + 170
```

- This sets the different positions for the different dance moves and calls the actors



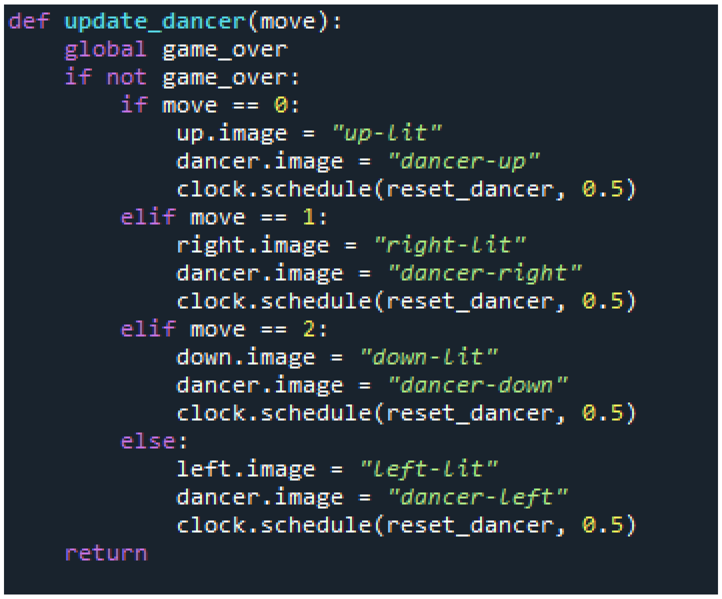
```python
def draw():
    global game_over, score, say_dance
    global count, show_countdown
    if not game_over:
        screen.clear()
        screen.blit("stage", (0, 0))
        dancer.draw()
        up.draw()
        down.draw()
        right.draw()
        left.draw()
        screen.draw.text("Score: " +
                         str(score), color="black",
                         topleft=(10, 10))
        if say_dance:
            screen.draw.text("Dance!", color="black",
                             topleft=(CENTER_X - 65, 150), fontsize=60)
        if show_countdown:
            screen.draw.text(str(count), color="black",
                             topleft=(CENTER_X - 8, 150), fontsize=60)
    else:
        screen.clear()
        screen.blit("stage", (0, 0))
        screen.draw.text("Score: " +
                         str(score), color="black",
                         topleft=(10, 10))
        screen.draw.text("GAME OVER!", color="black",
                         topleft=(CENTER_X - 130, 220), fontsize=60)
    return
```

- This function sets up all of the actors and objects that get drawn onto the screen and their position in relation to the coordinates from the box. It also sets up the game-over text to appear if game over happens



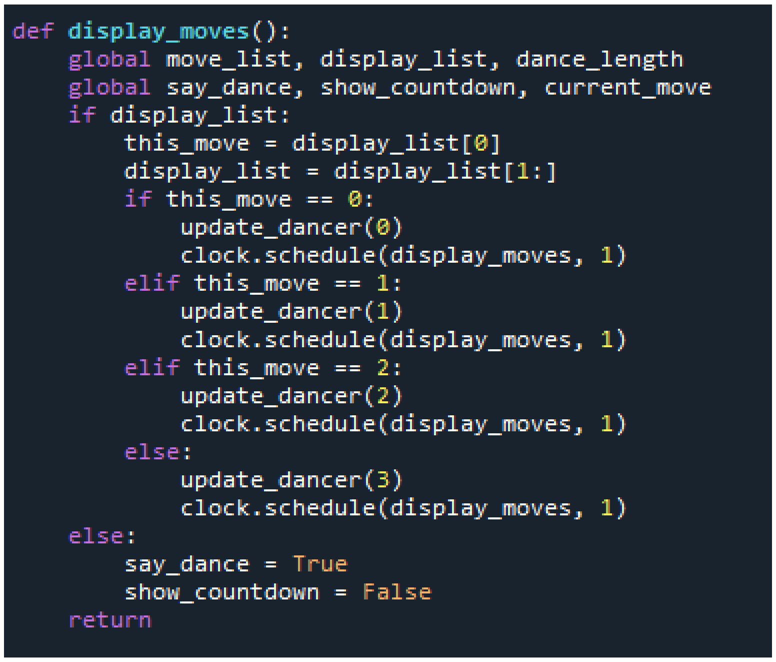
```python
def reset_dancer():
    global game_over
    if not game_over:
        dancer.image = "dancer-start"
        up.image = "up"
        right.image = "right"
        down.image = "down"
        left.image = "left"
    return
```

- The reset dancer function makes sure if not game over then when it shows an image it is corresponding to the right file.



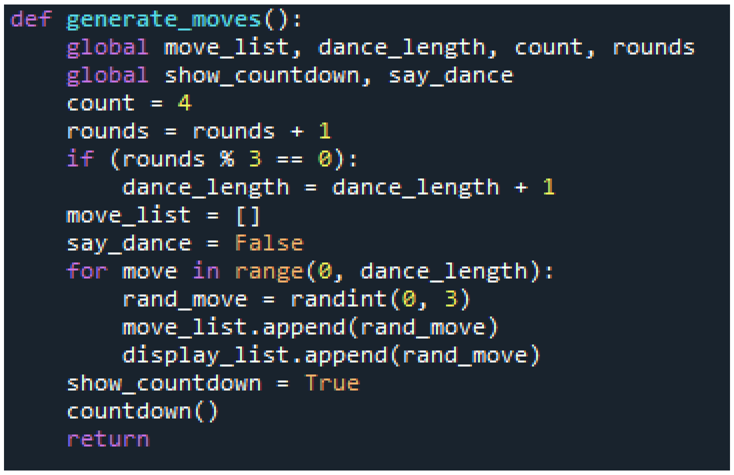
```python
def update_dancer(move):
    global game_over
    if not game_over:
        if move == 0:
            up.image = "up-lit"
            dancer.image = "dancer-up"
            clock.schedule(reset_dancer, 0.5)
        elif move == 1:
            right.image = "right-lit"
            dancer.image = "dancer-right"
            clock.schedule(reset_dancer, 0.5)
        elif move == 2:
            down.image = "down-lit"
            dancer.image = "dancer-down"
            clock.schedule(reset_dancer, 0.5)
        else:
            left.image = "left-lit"
            dancer.image = "dancer-left"
            clock.schedule(reset_dancer, 0.5)
    return
```

- This function updates the move so when a certain move is used it can then be copied and set equal to when the user inputs a value



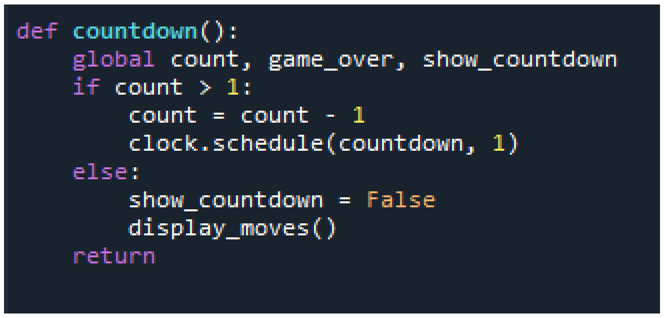
```python
def display_moves():
    global move_list, display_list, dance_length
    global say_dance, show_countdown, current_move
    if display_list:
        this_move = display_list[0]
        display_list = display_list[1:]
        if this_move == 0:
            update_dancer(0)
            clock.schedule(display_moves, 1)
        elif this_move == 1:
            update_dancer(1)
            clock.schedule(display_moves, 1)
        elif this_move == 2:
            update_dancer(2)
            clock.schedule(display_moves, 1)
        else:
            update_dancer(3)
            clock.schedule(display_moves, 1)
    else:
        say_dance = True
        show_countdown = False
    return
```

- This displays the move on the screen and updates from the update function. So when the screen is active you can see the figure moving.



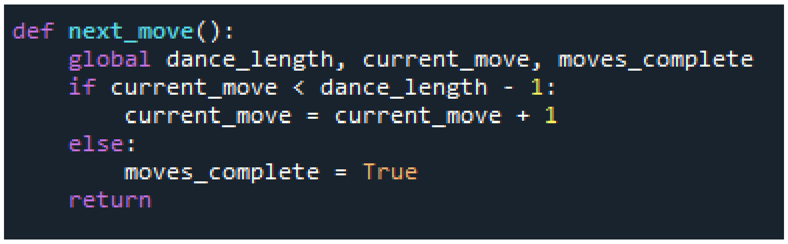
```python
def generate_moves():
    global move_list, dance_length, count, rounds
    global show_countdown, say_dance
    count = 4
    rounds = rounds + 1
    if (rounds % 3 == 0):
        dance_length = dance_length + 1
    move_list = []
    say_dance = False
    for move in range(0, dance_length):
        rand_move = randint(0, 3)
        move_list.append(rand_move)
        display_list.append(rand_move)
    show_countdown = True
    countdown()
    return
```

- This function generates the moves for each level and the new thing that was added is that every 3 levels the dancing length would increase thus increasing the difficulty.



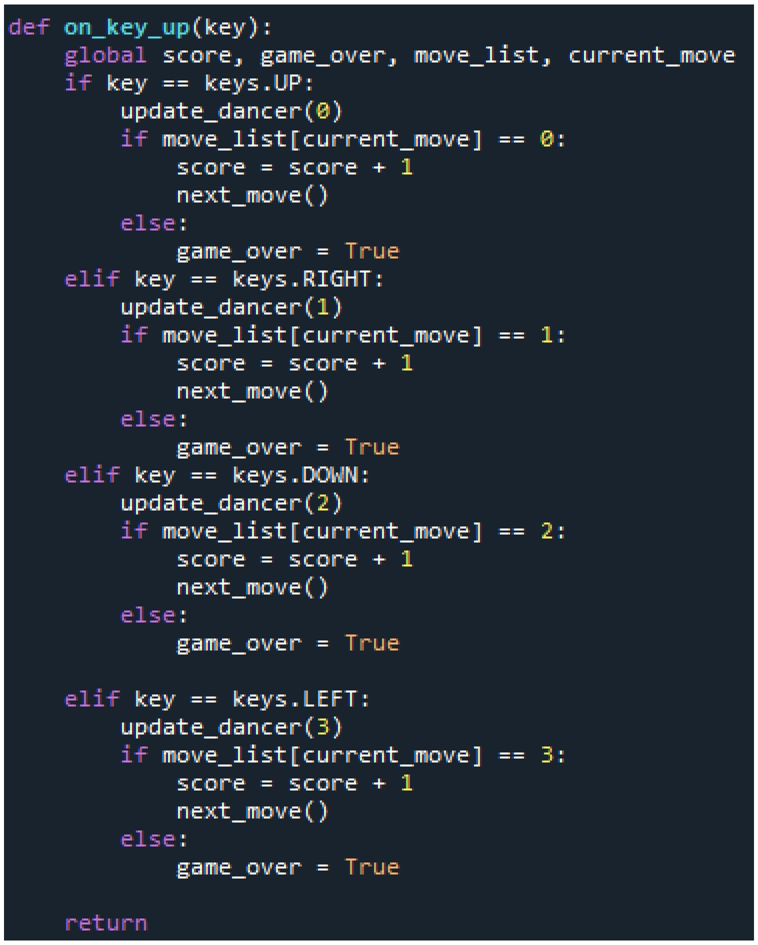
```python
def countdown():
    global count, game_over, show_countdown
    if count > 1:
        count = count - 1
        clock.schedule(countdown, 1)
    else:
        show_countdown = False
        display_moves()
    return
```

- This is for the countdown that you would see on the scree in order for you to dance



```python
def next_move():
    global dance_length, current_move, moves_complete
    if current_move < dance_length - 1:
        current_move = current_move + 1
    else:
        moves_complete = True
    return
```

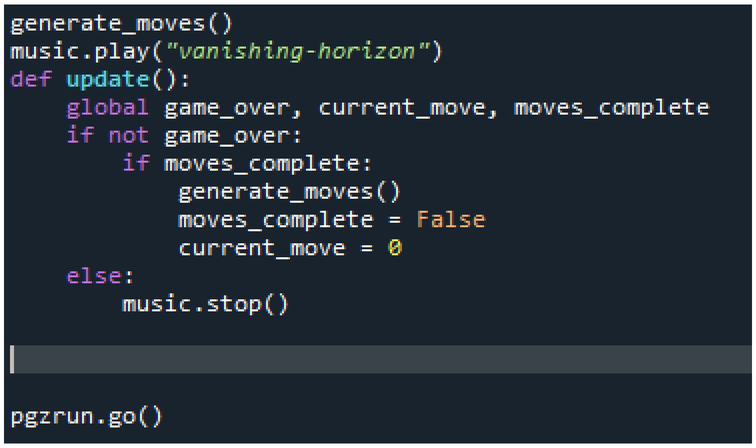
- This allows you to copy the dance moves and if you copy them correctly you can set moves to complete allowing you to move on to the next level



```python
def on_key_up(key):
    global score, game_over, move_list, current_move
    if key == keys.UP:
        update_dancer(0)
        if move_list[current_move] == 0:
            score = score + 1
            next_move()
        else:
            game_over = True
    elif key == keys.RIGHT:
        update_dancer(1)
        if move_list[current_move] == 1:
            score = score + 1
            next_move()
        else:
            game_over = True
    elif key == keys.DOWN:
        update_dancer(2)
        if move_list[current_move] == 2:
            score = score + 1
            next_move()
        else:
            game_over = True

    elif key == keys.LEFT:
        update_dancer(3)
        if move_list[current_move] == 3:
            score = score + 1
            next_move()
        else:
            game_over = True

    return
```

\

- When you copy the move correctly the score will increase. The dancer also moves corresponding to the user inputs



```python
generate_moves()
music.play("vanishing-horizon")
def update():
    global game_over, current_move, moves_complete
    if not game_over:
        if moves_complete:
            generate_moves()
            moves_complete = False
            current_move = 0
    else:
        music.stop()


pgzrun.go()
```

- This calls the music played in the background and generates the moves for the dance game. It also runs all of the programs with pgzrun.go(). I named the song the same but just replaced the .ogg file with a Japanese 80s city pop song.