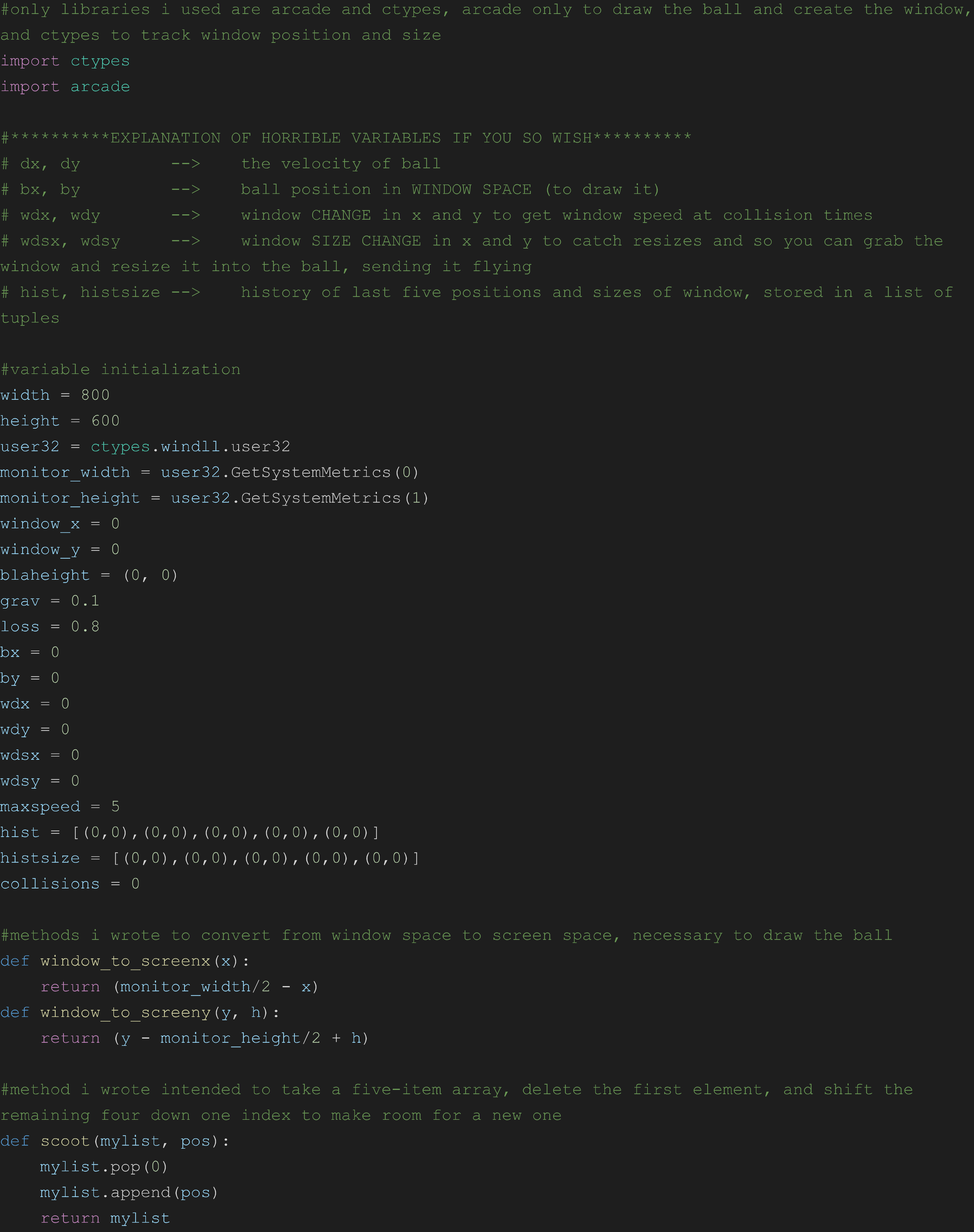
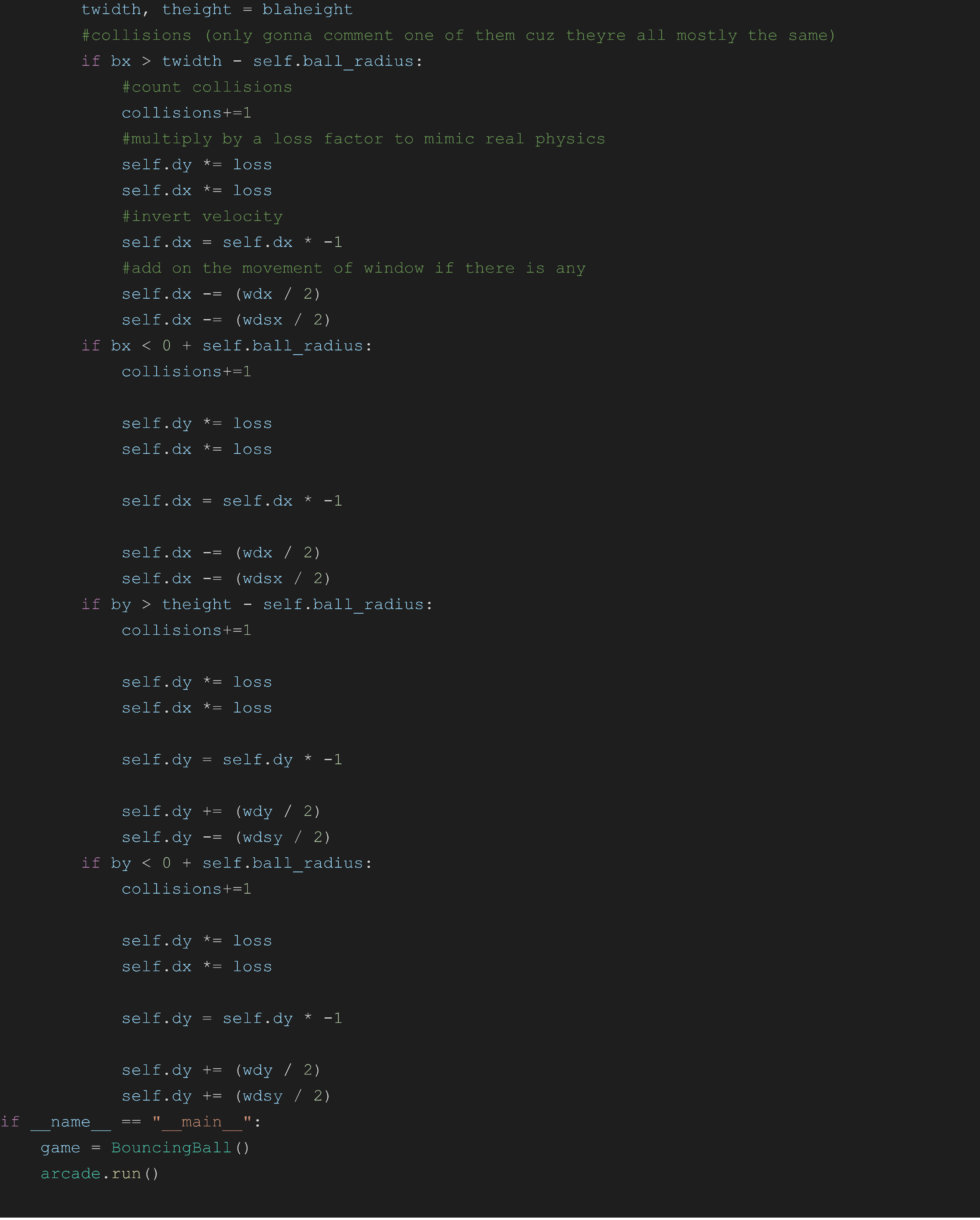


```
#only libraries i used are arcade and ctypes, arcade only to draw the ball and create the window, and ctypes to track window position and size
import ctypes
import arcade

#**********EXPLANATION OF HORRIBLE VARIABLES IF YOU SO WISH**********
# dx, dy         -->    the velocity of ball
# bx, by         -->    ball position in WINDOW SPACE (to draw it)
# wdx, wdy       -->    window CHANGE in x and y to get window speed at collision times
# wdsx, wdsy     -->    window SIZE CHANGE in x and y to catch resizes and so you can grab the window and resize it into the ball, sending it flying
# hist, histsize -->    history of last five positions and sizes of window, stored in a list of tuples

#variable initialization
width = 800
height = 600
user32 = ctypes.windll.user32
monitor_width = user32.GetSystemMetrics(0)
monitor_height = user32.GetSystemMetrics(1)
window_x = 0
window_y = 0
blaheight = (0, 0)
grav = 0.1
loss = 0.8
bx = 0
by = 0
wdx = 0
wdy = 0
wdsx = 0
wdsy = 0
maxspeed = 5
hist = [(0,0),(0,0),(0,0),(0,0),(0,0)]
histsize = [(0,0),(0,0),(0,0),(0,0),(0,0)]
collisions = 0

#methods i wrote to convert from window space to screen space, necessary to draw the ball
def window_to_screenx(x):
    return (monitor_width/2 - x)
def window_to_screeny(y, h):
    return (y - monitor_height/2 + h)

#method i wrote intended to take a five-item array, delete the first element, and shift the remaining four down one index to make room for a new one
def scoot(mylist, pos):
    mylist.pop(0)
    mylist.append(pos)
    return mylist
```

```python
#get average rate of change of list, to get average velocity of window when hit ball
def avglist(mylist):
    x_start, y_start = mylist[0]
    x_end, y_end = mylist[-1]
    return x_end - x_start, y_end - y_start

class BouncingBall(arcade.Window):
    def __init__(self):
        super().__init__(width, height, "ball", resizable=True)
        self.ball_screen_x = window_to_screenx(width // 2)
        self.ball_screen_y = window_to_screeny(height // 2, blaheight[1])
        self.ball_radius = 25
        self.dx = -1
        self.dy = -1
        #THE BELOW MAY NEED CHANGING DEPENDING ON YOUR MONITOR'S REFRESH RATE!!! dont worry tho itll work either way
        self.set_update_rate(1/144)

    #method only to draw the ball
    def on_draw(self):
        arcade.Window.clear(self)
        global bx, by
        self.ball = arcade.draw_circle_filled(bx, by, self.ball_radius, arcade.color.RED)

    #calculations and heavy lifting here
    def on_update(self, _delta_time):
        global window_x, window_y, blaheight
        global bx, by
        global wdx, wdy, hist, histsize
        global wdsx, wdsy, maxspeed, collisions
        #gravity of ball
        self.dy += grav
        #update position of ball
        self.ball_screen_x += self.dx
        self.ball_screen_y += self.dy
        #get window thingamajigs for drawing
        blaheight = self.get_size()
        window_x, window_y = self.get_location()
        #get bx (ball x) and by (ball y) relative to screen to draw them
        bx, by = window_to_screenx(window_x + self.ball_screen_x), window_to_screeny(window_y - self.ball_screen_y, blaheight[1])
        #this records last five positions and sizes of windows to do math
        hist = scoot(hist, (window_x, window_y))
        histsize = scoot(histsize, blaheight)
        wdsx, wdsy = avglist(histsize)
        #wdx (window delta x) tracks the change in x
        wdx, wdy = avglist(hist)
        #temporary width and height of window in case of resizing, setting it to a tuple called blaheight (dont ask why)
```



```
        twidth, theight = blaheight
        #collisions (only gonna comment one of them cuz theyre all mostly the same)
        if bx > twidth - self.ball_radius:
            #count collisions
            collisions+=1
            #multiply by a loss factor to mimic real physics
            self.dy *= loss
            self.dx *= loss
            #invert velocity
            self.dx = self.dx * -1
            #add on the movement of window if there is any
            self.dx -= (wdx / 2)
            self.dx -= (wdsx / 2)
        if bx < 0 + self.ball_radius:
            collisions+=1

            self.dy *= loss
            self.dx *= loss

            self.dx = self.dx * -1

            self.dx -= (wdx / 2)
            self.dx -= (wdsx / 2)
        if by > theight - self.ball_radius:
            collisions+=1

            self.dy *= loss
            self.dx *= loss

            self.dy = self.dy * -1

            self.dy += (wdy / 2)
            self.dy -= (wdsy / 2)
        if by < 0 + self.ball_radius:
            collisions+=1

            self.dy *= loss
            self.dx *= loss

            self.dy = self.dy * -1

            self.dy += (wdy / 2)
            self.dy += (wdsy / 2)
if __name__ == "__main__":
    game = BouncingBall()
    arcade.run()
```