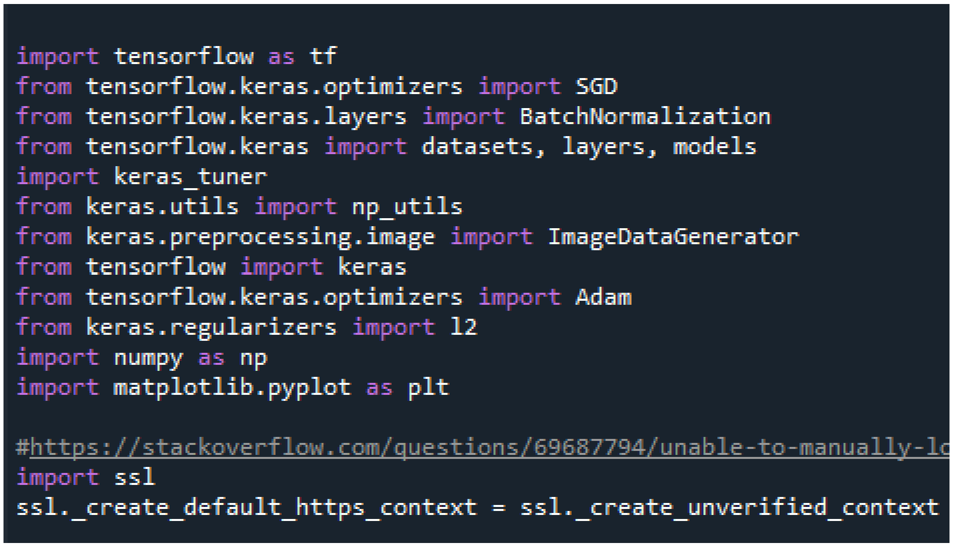
Lab7 EE104 Documentation

Names: Grant von Pingel

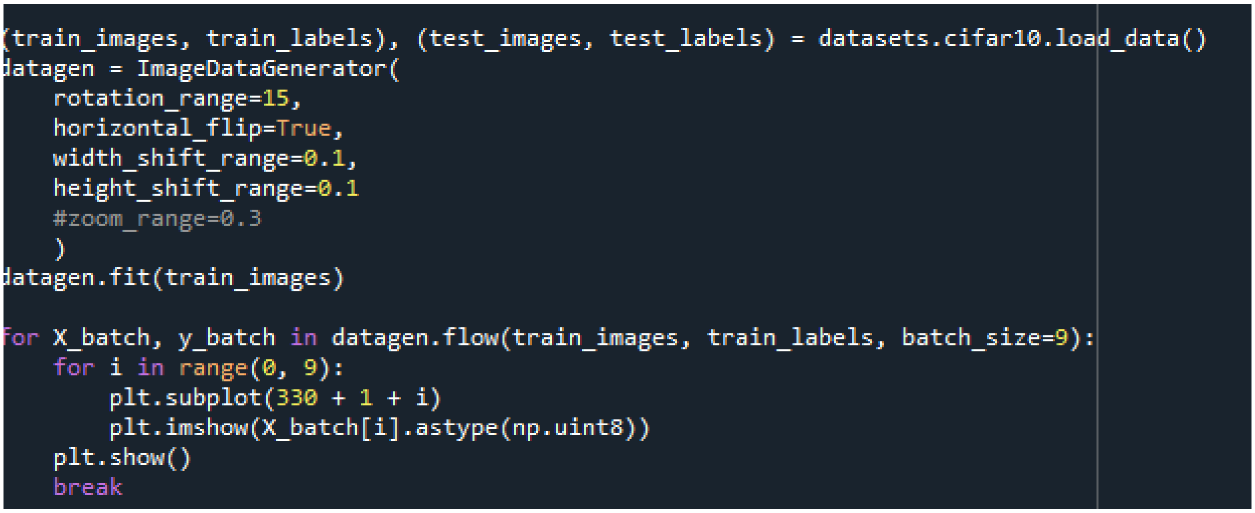
Kerolles Hermina

Part 1: CNN - Baseline



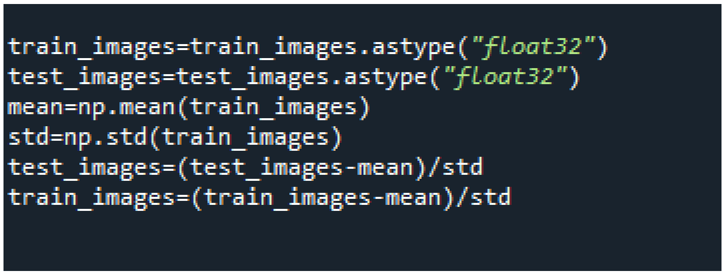
```
import tensorflow as tf
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras import datasets, layers, models
import keras_tuner
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras
from tensorflow.keras.optimizers import Adam
from keras.regularizers import l2
import numpy as np
import matplotlib.pyplot as plt

#https://stackoverflow.com/questions/69687794/unable-to-manually-l
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

- Importing all the necessary libraries and setting up the GPU to be utilized



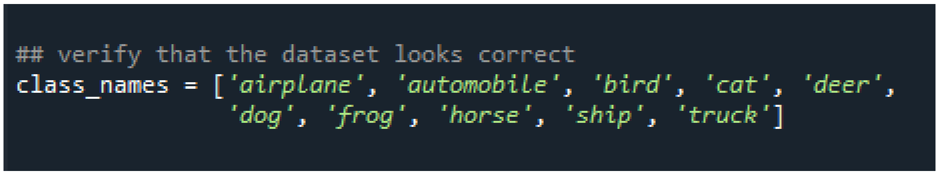
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
datagen = ImageDataGenerator(
    rotation_range=15,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
    #zoom_range=0.3
    )
datagen.fit(train_images)

for X_batch, y_batch in datagen.flow(train_images, train_labels, batch_size=9):
    for i in range(0, 9):
        plt.subplot(330 + 1 + i)
        plt.imshow(X_batch[i].astype(np.uint8))
    plt.show()
    break
```

- This sets up training images and sets up the range for the test images and data sets



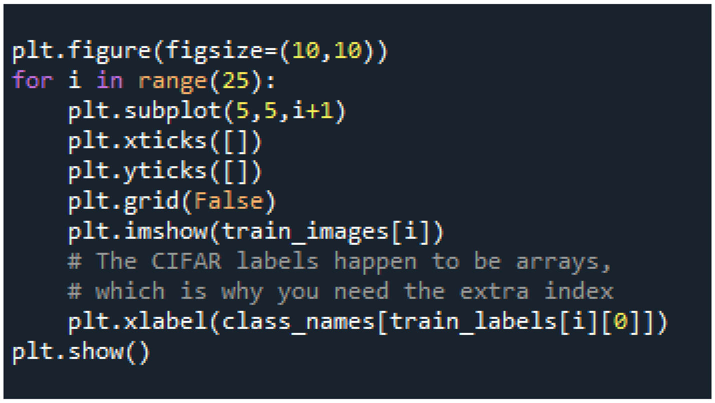
```
train_images=train_images.astype("float32")
test_images=test_images.astype("float32")
mean=np.mean(train_images)
std=np.std(train_images)
test_images=(test_images-mean)/std
train_images=(train_images-mean)/std
```

- Sets the images as a type of float and creates test equations that are used to filter the test images



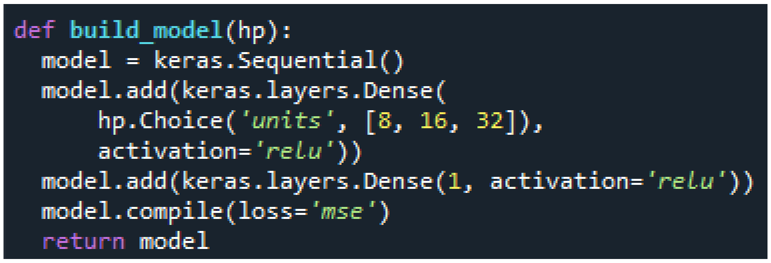
```
## verify that the dataset looks correct
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

- All of the different classifications that the images can be put under



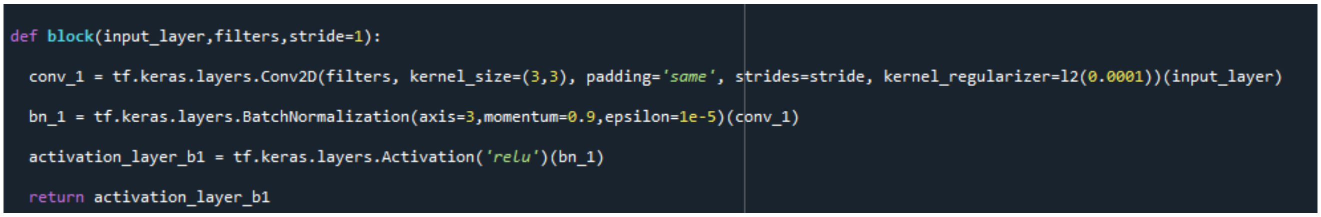
```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

- Plot the subplot and setting up the coordinates and then plotting the training images with labels



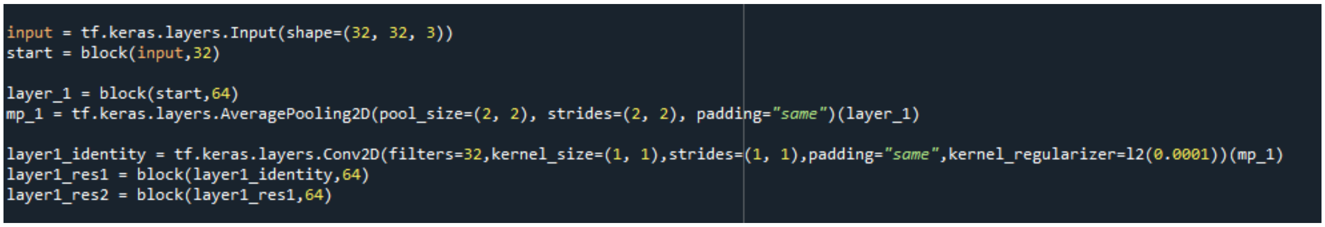
```
def build_model(hp):
  model = keras.Sequential()
  model.add(keras.layers.Dense(
      hp.Choice('units', [8, 16, 32]),
      activation='relu'))
  model.add(keras.layers.Dense(1, activation='relu'))
  model.compile(loss='mse')
  return model
```

- Setting up the different training labels and test labels and then setting up the build model



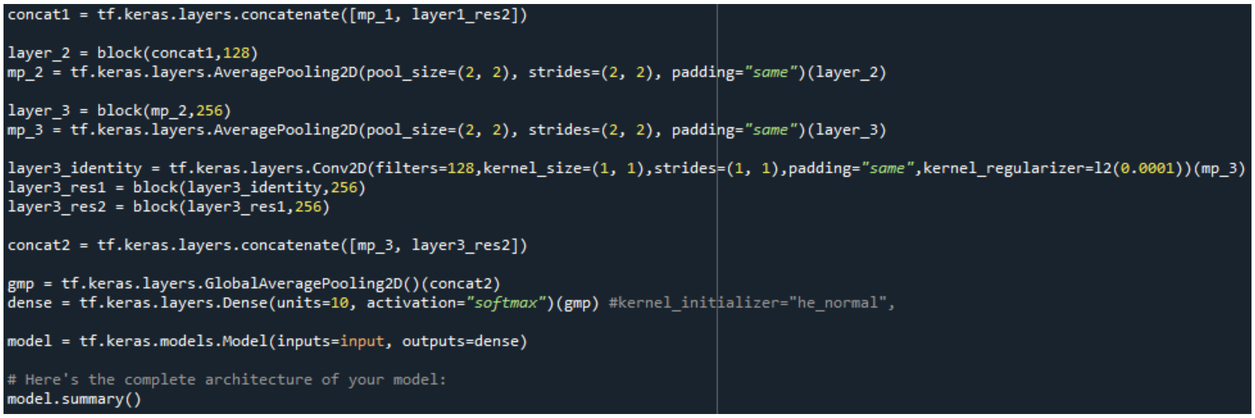
```
def block(input_layer,filters,stride=1):

  conv_1 = tf.keras.layers.Conv2D(filters, kernel_size=(3,3), padding='same', strides=stride, kernel_regularizer=l2(0.0001))(input_layer)

  bn_1 = tf.keras.layers.BatchNormalization(axis=3,momentum=0.9,epsilon=1e-5)(conv_1)

  activation_layer_b1 = tf.keras.layers.Activation('relu')(bn_1)

  return activation_layer_b1
```

- Defining the block layer, setting up the different layers, and getting the equations for the activation later



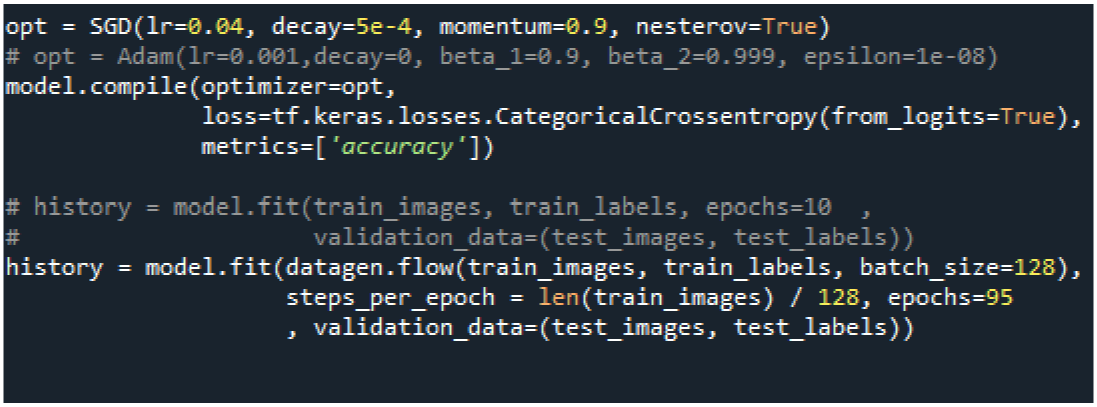
```
input = tf.keras.layers.Input(shape=(32, 32, 3))
start = block(input,32)

layer_1 = block(start,64)
mp_1 = tf.keras.layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding="same")(layer_1)

layer1_identity = tf.keras.layers.Conv2D(filters=32,kernel_size=(1, 1),strides=(1, 1),padding="same",kernel_regularizer=l2(0.0001))(mp_1)
layer1_res1 = block(layer1_identity,64)
layer1_res2 = block(layer1_res1,64)
```

- Setting up Layer 1, setting up the input layers, res1, and res2



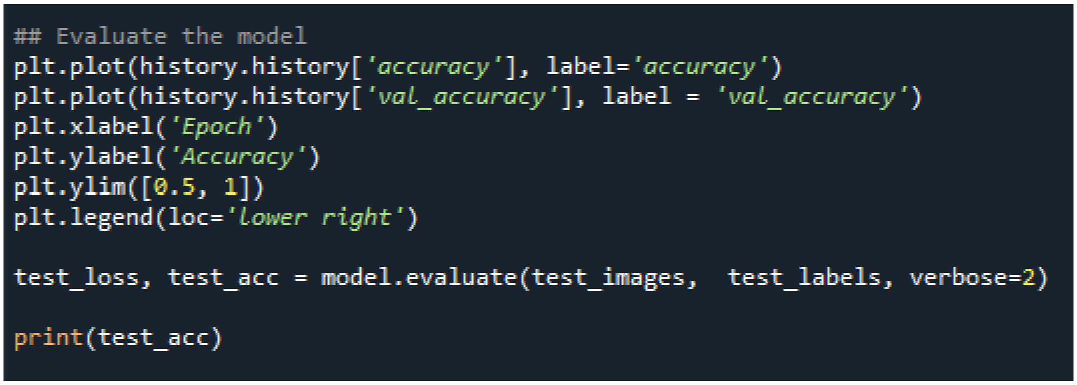
```
concat1 = tf.keras.layers.concatenate([mp_1, layer1_res2])

layer_2 = block(concat1,128)
mp_2 = tf.keras.layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding="same")(layer_2)

layer_3 = block(mp_2,256)
mp_3 = tf.keras.layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding="same")(layer_3)

layer3_identity = tf.keras.layers.Conv2D(filters=128,kernel_size=(1, 1),strides=(1, 1),padding="same",kernel_regularizer=l2(0.0001))(mp_3)
layer3_res1 = block(layer3_identity,256)
layer3_res2 = block(layer3_res1,256)

concat2 = tf.keras.layers.concatenate([mp_3, layer3_res2])

gmp = tf.keras.layers.GlobalAveragePooling2D()(concat2)
dense = tf.keras.layers.Dense(units=10, activation="softmax")(gmp) #kernel_initializer="he_normal",

model = tf.keras.models.Model(inputs=input, outputs=dense)

# Here's the complete architecture of your model:
model.summary()
```

- Concatenating the layers together and then doing the same for layers 2, 3, and 4, while changing the amount of pictures



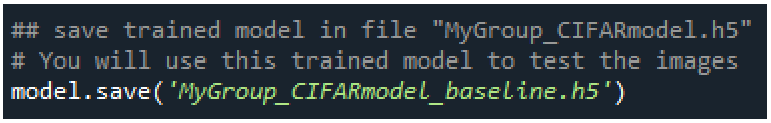
```
opt = SGD(lr=0.04, decay=5e-4, momentum=0.9, nesterov=True)
# opt = Adam(lr=0.001,decay=0, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model.compile(optimizer=opt,
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# history = model.fit(train_images, train_labels, epochs=10  ,
#                     validation_data=(test_images, test_labels))
history = model.fit(datagen.flow(train_images, train_labels, batch_size=128),
                    steps_per_epoch = len(train_images) / 128, epochs=95
                   , validation_data=(test_images, test_labels))
```

- Compiling the neural network, calculating the accuracy on how well the program was able to guess the image, and then using the history to create another model



```
## Evaluate the model
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print(test_acc)
```

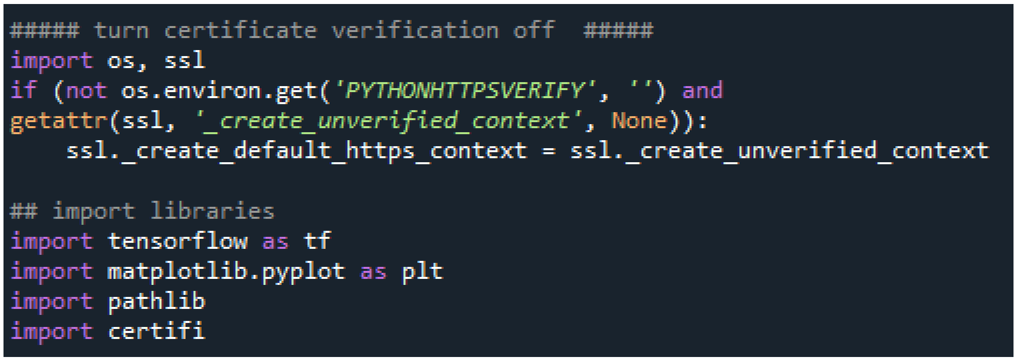
- Plotting the accuracy, val_accuracy, and which epoch we are in all in the terminal
- Finally printing the test accuracy of the runs



```
## save trained model in file "MyGroup_CIFARmodel.h5"
# You will use this trained model to test the images
model.save('MyGroup_CIFARmodel_baseline.h5')
```

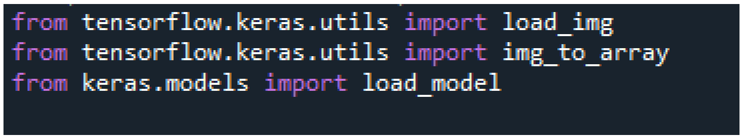
- Save the trained model into an h5 file to be used later

Part 2: CNN - Test



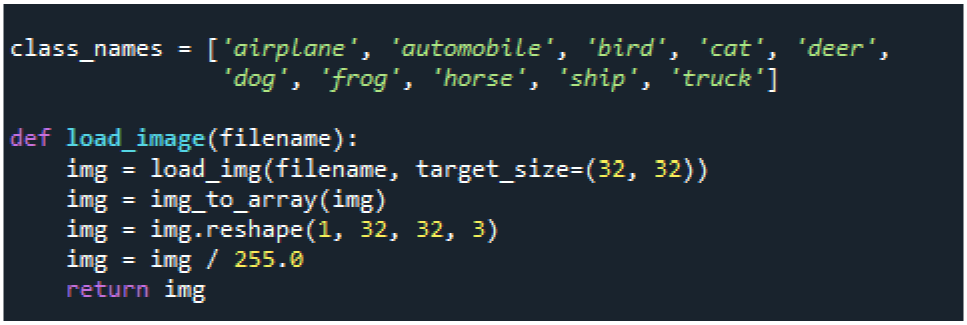
```
##### turn certificate verification off  #####
import os, ssl
if (not os.environ.get('PYTHONHTTPSVERIFY', '') and
getattr(ssl, '_create_unverified_context', None)):
    ssl._create_default_https_context = ssl._create_unverified_context

## import libraries
import tensorflow as tf
import matplotlib.pyplot as plt
import pathlib
import certifi
```

- Import the different libraries needed for program to function



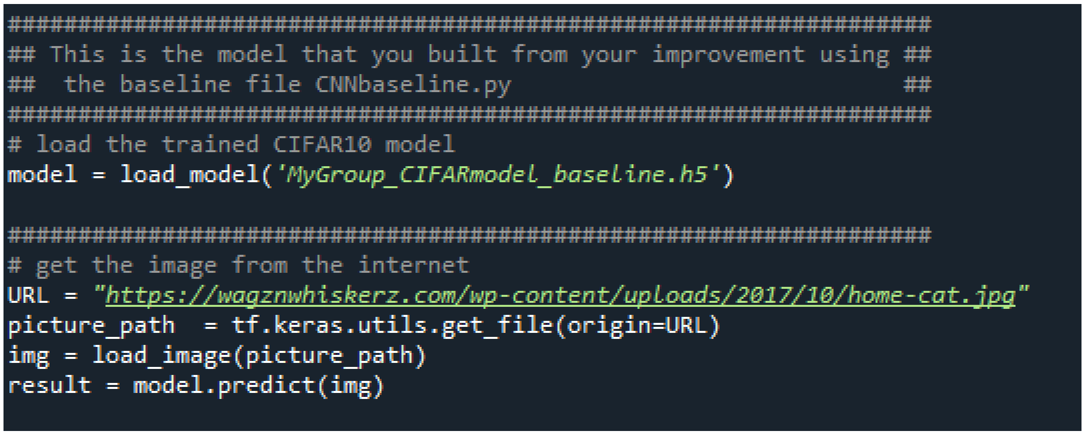
```
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.models import load_model
```

- Import all the TensorFlow functions



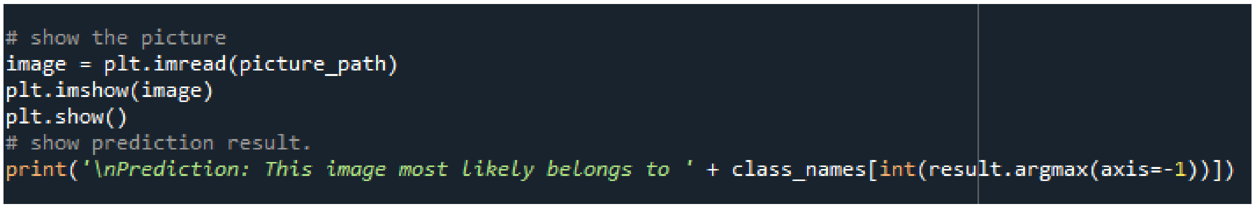
```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

def load_image(filename):
    img = load_img(filename, target_size=(32, 32))
    img = img_to_array(img)
    img = img.reshape(1, 32, 32, 3)
    img = img / 255.0
    return img
```

- Load up the different categories the pictures can be sorted into
- reshape the images to specified sizes



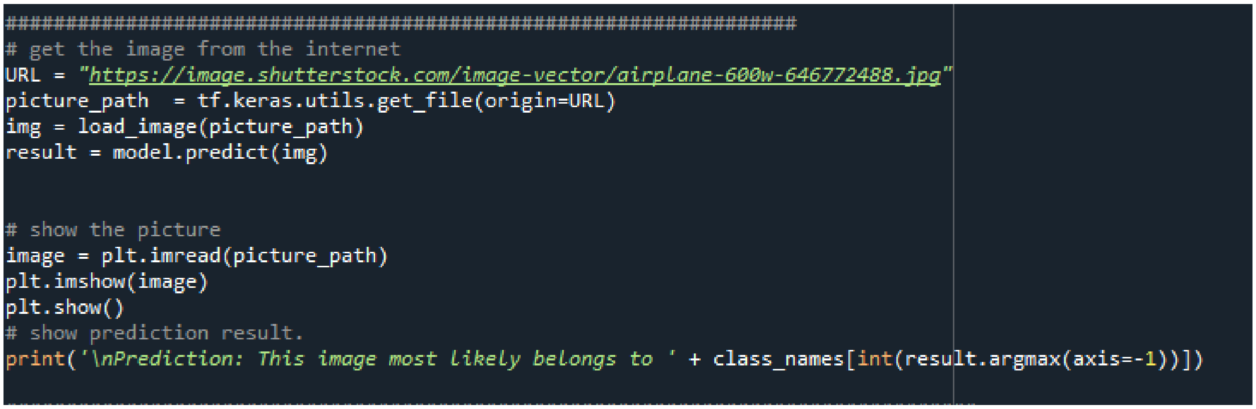
```
##################################################################
## This is the model that you built from your improvement using ##
##  the baseline file CNNbaseline.py                            ##
##################################################################
# load the trained CIFAR10 model
model = load_model('MyGroup_CIFARmodel_baseline.h5')

##################################################################
# get the image from the internet
URL = "https://wagznwhiskerz.com/wp-content/uploads/2017/10/home-cat.jpg"
picture_path  = tf.keras.utils.get_file(origin=URL)
img = load_image(picture_path)
result = model.predict(img)
```

- Our first test image, which was a cat, and loading image into neural network created



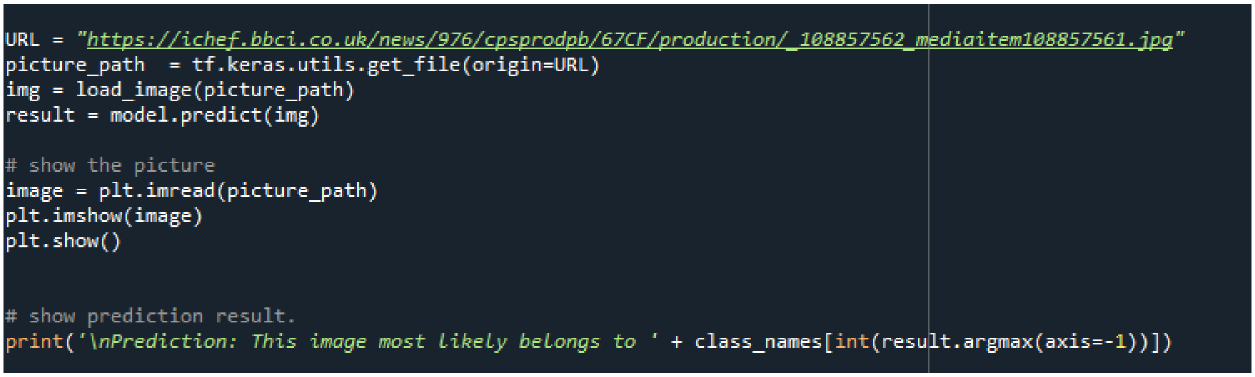
```
# show the picture
image = plt.imread(picture_path)
plt.imshow(image)
plt.show()
# show prediction result.
print('\nPrediction: This image most likely belongs to ' + class_names[int(result.argmax(axis=-1))])
```

- Plot the image and show the image, while also printing the prediction of what the neural network thought the image was



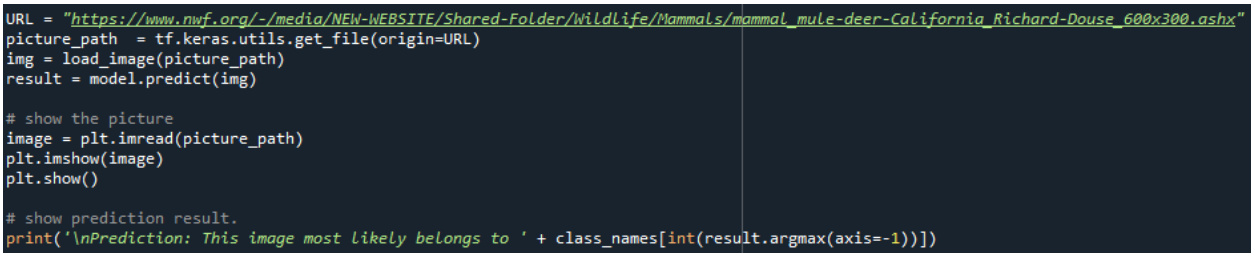
```
##################################################################
# get the image from the internet
URL = "https://image.shutterstock.com/image-vector/airplane-600w-646772488.jpg"
picture_path  = tf.keras.utils.get_file(origin=URL)
img = load_image(picture_path)
result = model.predict(img)


# show the picture
image = plt.imread(picture_path)
plt.imshow(image)
plt.show()
# show prediction result.
print('\nPrediction: This image most likely belongs to ' + class_names[int(result.argmax(axis=-1))])
```

- Same as previous, except this time the url was a plane



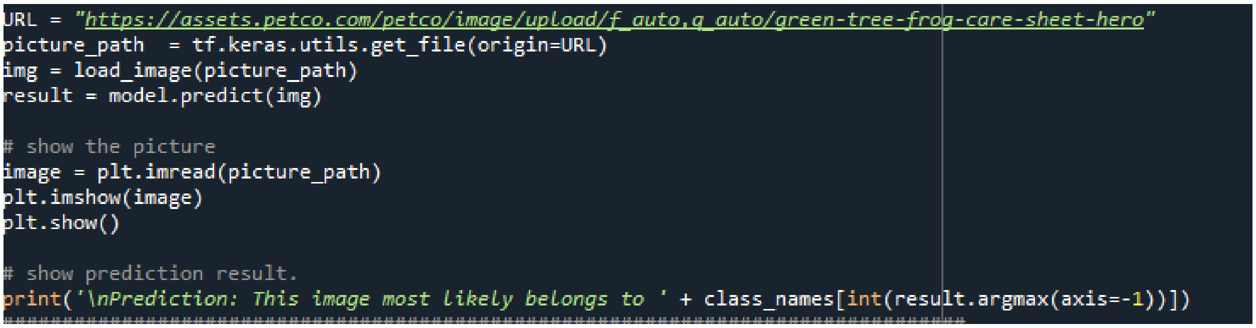
```
URL = "https://ichef.bbci.co.uk/news/976/cpsprodpb/67CF/production/_108857562_mediaitem108857561.jpg"
picture_path  = tf.keras.utils.get_file(origin=URL)
img = load_image(picture_path)
result = model.predict(img)

# show the picture
image = plt.imread(picture_path)
plt.imshow(image)
plt.show()


# show prediction result.
print('\nPrediction: This image most likely belongs to ' + class_names[int(result.argmax(axis=-1))])
```

- Same as previous, except this time the url was a bird



```
URL = "https://www.nwf.org/-/media/NEW-WEBSITE/Shared-Folder/Wildlife/Mammals/mammal_mule-deer-California_Richard-Douse_600x300.ashx"
picture_path  = tf.keras.utils.get_file(origin=URL)
img = load_image(picture_path)
result = model.predict(img)

# show the picture
image = plt.imread(picture_path)
plt.imshow(image)
plt.show()

# show prediction result.
print('\nPrediction: This image most likely belongs to ' + class_names[int(result.argmax(axis=-1))])
```

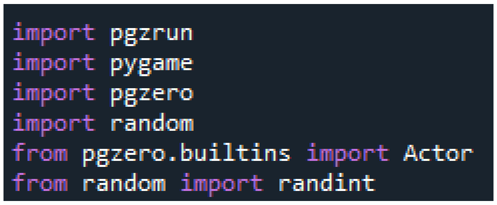
- Same as previous, except this time the url was a deer



```
URL = "https://assets.petco.com/petco/image/upload/f_auto,q_auto/green-tree-frog-care-sheet-hero"
picture_path  = tf.keras.utils.get_file(origin=URL)
img = load_image(picture_path)
result = model.predict(img)

# show the picture
image = plt.imread(picture_path)
plt.imshow(image)
plt.show()

# show prediction result.
print('\nPrediction: This image most likely belongs to ' + class_names[int(result.argmax(axis=-1))])
```

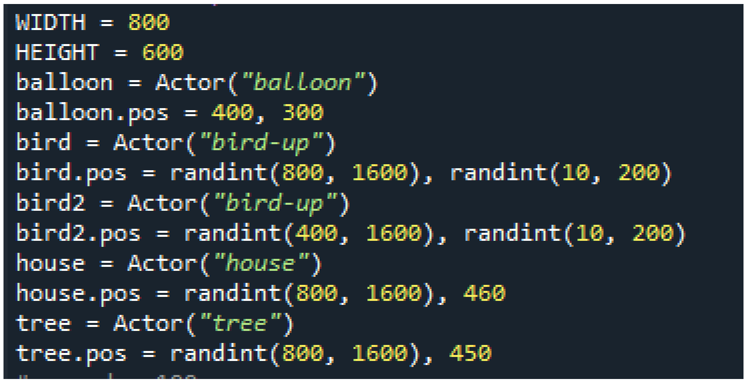
- Same as previous, except this time the url was a frog

Part 3: Balloon Flight Game



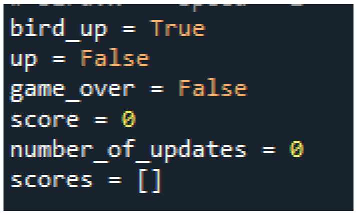
```
import pgzrun
import pygame
import pgzero
import random
from pgzero.builtins import Actor
from random import randint
```

- Import different libraries used for Py Games



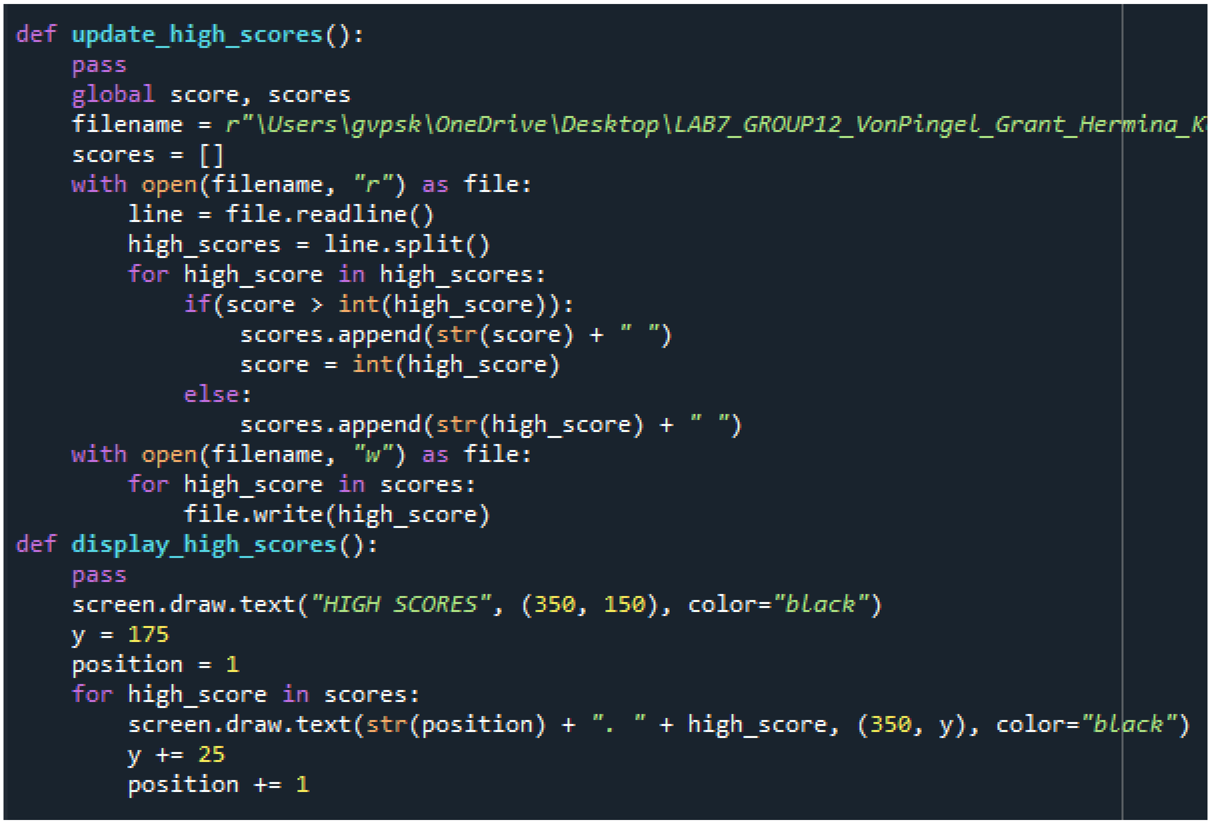
```
WIDTH = 800
HEIGHT = 600
balloon = Actor("balloon")
balloon.pos = 400, 300
bird = Actor("bird-up")
bird.pos = randint(800, 1600), randint(10, 200)
bird2 = Actor("bird-up")
bird2.pos = randint(400, 1600), randint(10, 200)
house = Actor("house")
house.pos = randint(800, 1600), 460
tree = Actor("tree")
tree.pos = randint(800, 1600), 450
```

- Setting up the dimensions of the screen and setting up the different actors and where they spawn on the screen



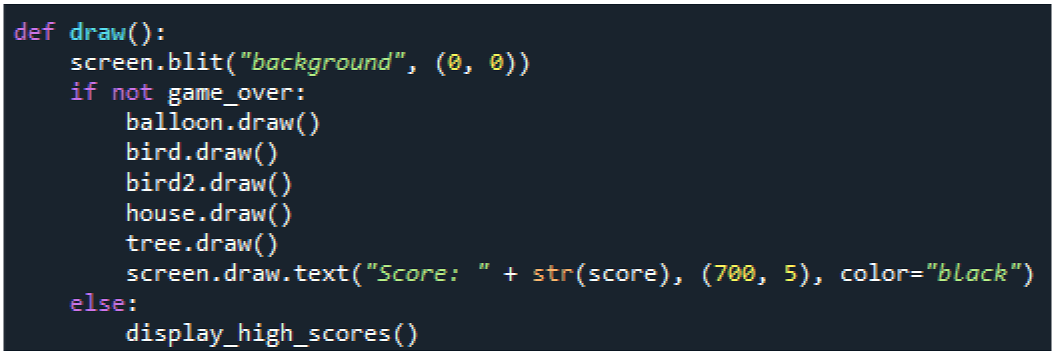
```
bird_up = True
up = False
game_over = False
score = 0
number_of_updates = 0
scores = []
```

- Setting up different callbacks that we could later flip to be true or false to go to a new function
- Setting score to be 0 initially



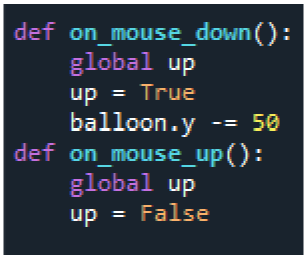
```python
def update_high_scores():
    pass
    global score, scores
    filename = r"\Users\gvpsk\OneDrive\Desktop\LAB7_GROUP12_VonPingel_Grant_Hermina_K
    scores = []
    with open(filename, "r") as file:
        line = file.readline()
        high_scores = line.split()
        for high_score in high_scores:
            if(score > int(high_score)):
                scores.append(str(score) + " ")
                score = int(high_score)
            else:
                scores.append(str(high_score) + " ")
    with open(filename, "w") as file:
        for high_score in scores:
            file.write(high_score)
def display_high_scores():
    pass
    screen.draw.text("HIGH SCORES", (350, 150), color="black")
    y = 175
    position = 1
    for high_score in scores:
        screen.draw.text(str(position) + ". " + high_score, (350, y), color="black")
        y += 25
        position += 1
```

- Setting up the high score of file and then replacing the high score if it was greater than any previous high score
- Displaying the high score when Game Over occurs



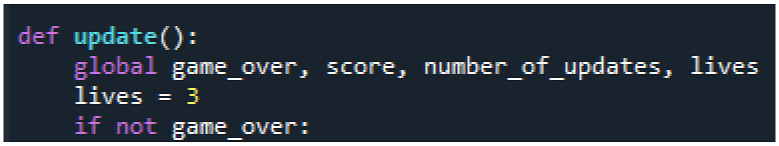
```python
def draw():
    screen.blit("background", (0, 0))
    if not game_over:
        balloon.draw()
        bird.draw()
        bird2.draw()
        house.draw()
        tree.draw()
        screen.draw.text("Score: " + str(score), (700, 5), color="black")
    else:
        display_high_scores()
```

- Drawing the background and the different actors onto the screen



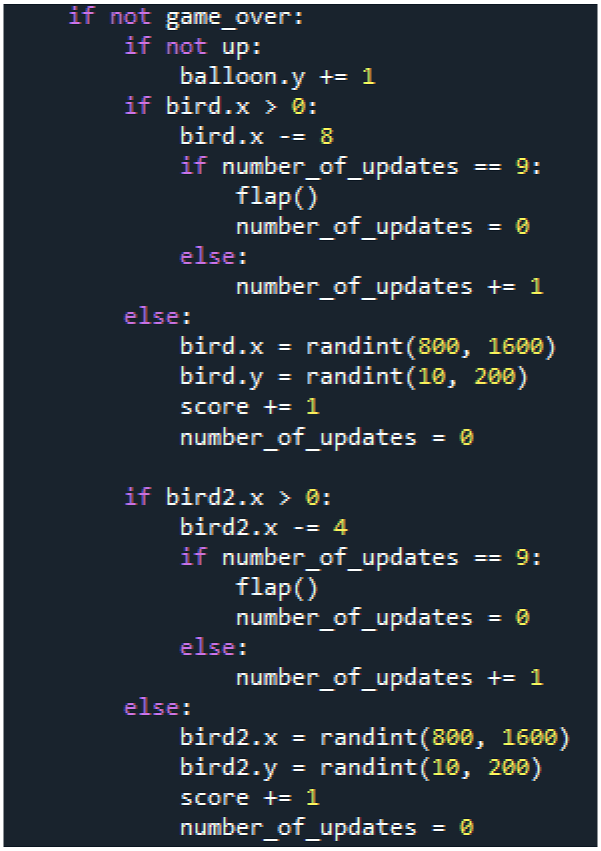
```python
def on_mouse_down():
    global up
    up = True
    balloon.y -= 50
def on_mouse_up():
    global up
    up = False
```

- When you click the balloon, the balloon goes up
- When you don’t click the balloon, the balloon goes down



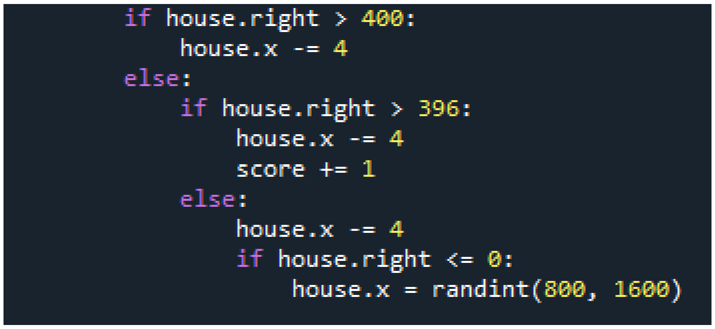
```
def update():
    global game_over, score, number_of_updates, lives
    lives = 3
    if not game_over:
```

- Defining the different global updates



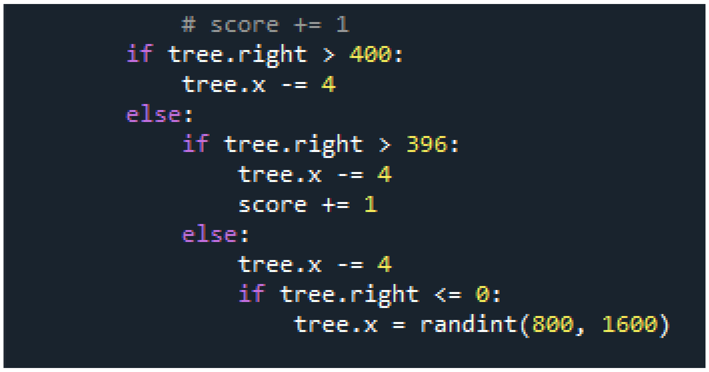
```
    if not game_over:
        if not up:
            balloon.y += 1
        if bird.x > 0:
            bird.x -= 8
            if number_of_updates == 9:
                flap()
                number_of_updates = 0
            else:
                number_of_updates += 1
        else:
            bird.x = randint(800, 1600)
            bird.y = randint(10, 200)
            score += 1
            number_of_updates = 0

        if bird2.x > 0:
            bird2.x -= 4
            if number_of_updates == 9:
                flap()
                number_of_updates = 0
            else:
                number_of_updates += 1
        else:
            bird2.x = randint(800, 1600)
            bird2.y = randint(10, 200)
            score += 1
            number_of_updates = 0
```

- Defining the speed of the birds and the positioning in which they spawn
- Having the birds move across the screen and give a point when they exit the screen



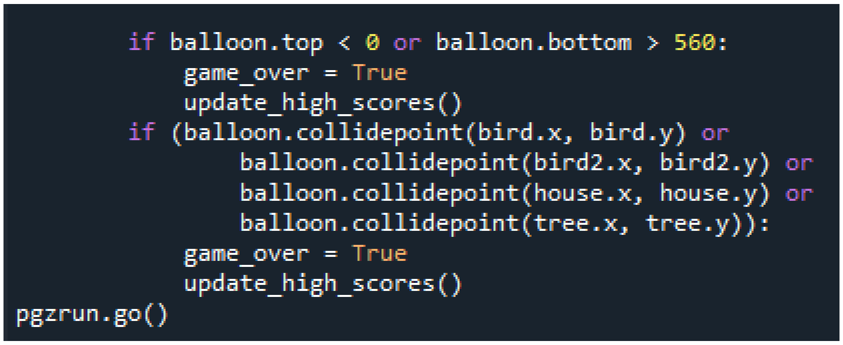
```
        if house.right > 400:
            house.x -= 4
        else:
            if house.right > 396:
                house.x -= 4
                score += 1
            else:
                house.x -= 4
                if house.right <= 0:
                    house.x = randint(800, 1600)
```

- Have the house move at a certain speed and give you a point when you pass over the house



```
            # score += 1
        if tree.right > 400:
            tree.x -= 4
        else:
            if tree.right > 396:
                tree.x -= 4
                score += 1
            else:
                tree.x -= 4
                if tree.right <= 0:
                    tree.x = randint(800, 1600)
```

- Have a tree move across the screen at a specific speed and gaining a point when you pass over a tree and spawn location



```
        if balloon.top < 0 or balloon.bottom > 560:
            game_over = True
            update_high_scores()
        if (balloon.collidepoint(bird.x, bird.y) or
                balloon.collidepoint(bird2.x, bird2.y) or
                balloon.collidepoint(house.x, house.y) or
                balloon.collidepoint(tree.x, tree.y)):
            game_over = True
            update_high_scores()
pgzrun.go()
```

- If the balloon touches the bottom or collides with any of the actors, it is game over
- Run the program