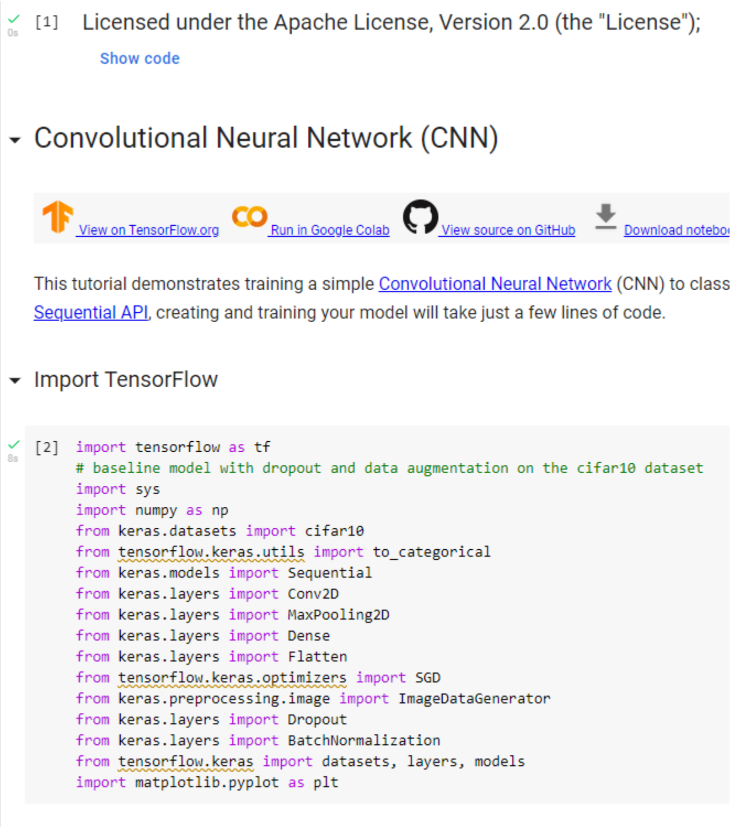
(1) CNN



[1] Licensed under the Apache License, Version 2.0 (the "License");

Convolutional Neural Network (CNN)

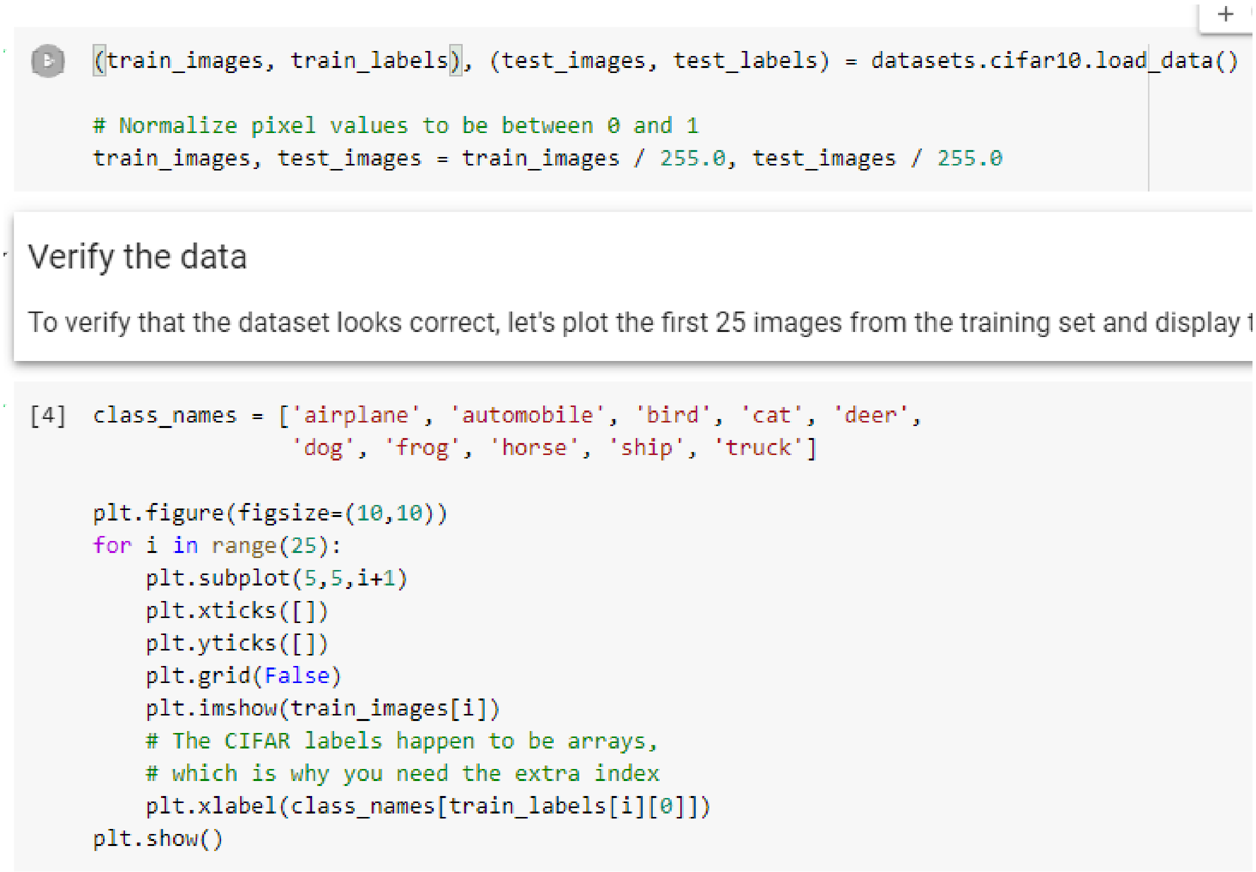
This tutorial demonstrates training a simple Convolutional Neural Network (CNN) to class
Sequential API, creating and training your model will take just a few lines of code.

Import TensorFlow

[2]
```
import tensorflow as tf
# baseline model with dropout and data augmentation on the cifar10 dataset
import sys
import numpy as np
from keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout
from keras.layers import BatchNormalization
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

[1] - License

[2] - Import libraries



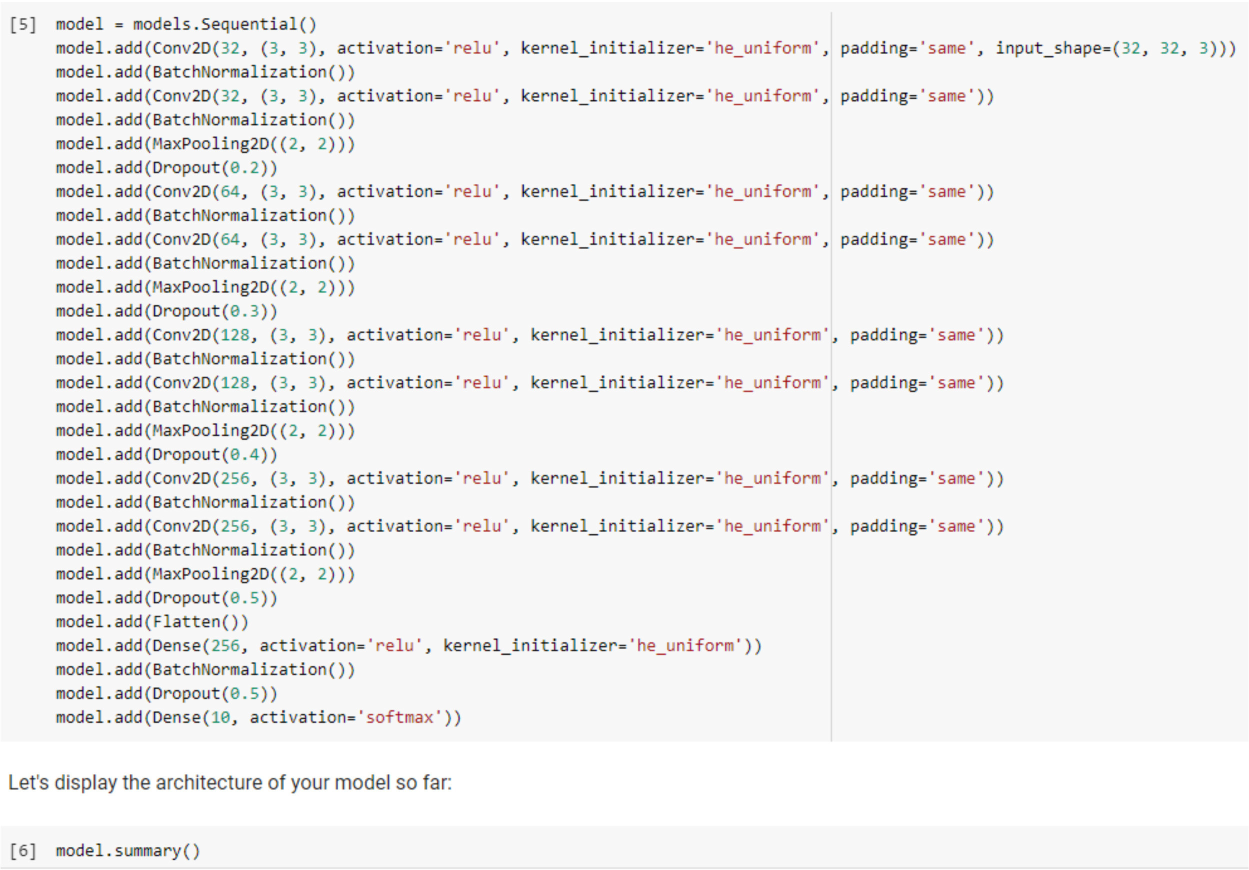
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

Verify the data

To verify that the dataset looks correct, let's plot the first 25 images from the training set and display t

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

     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()
```

[3] - Import CIFAR-10 dataset
[4] - Verify data by plotting 25 images



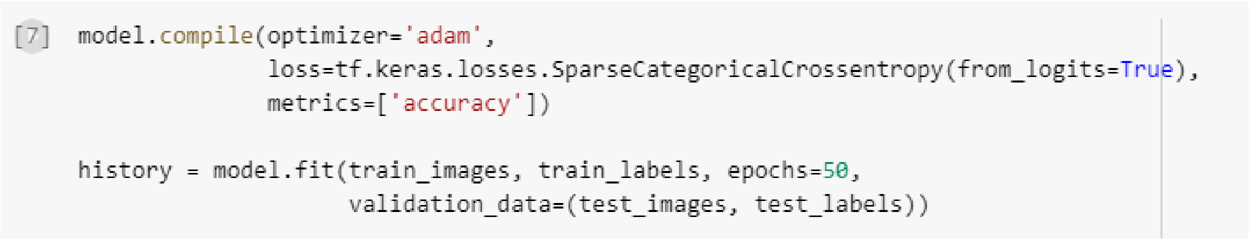
```
[5] model = models.Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.4))
    model.add(Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))
```

Let's display the architecture of your model so far:

```
[6] model.summary()
```

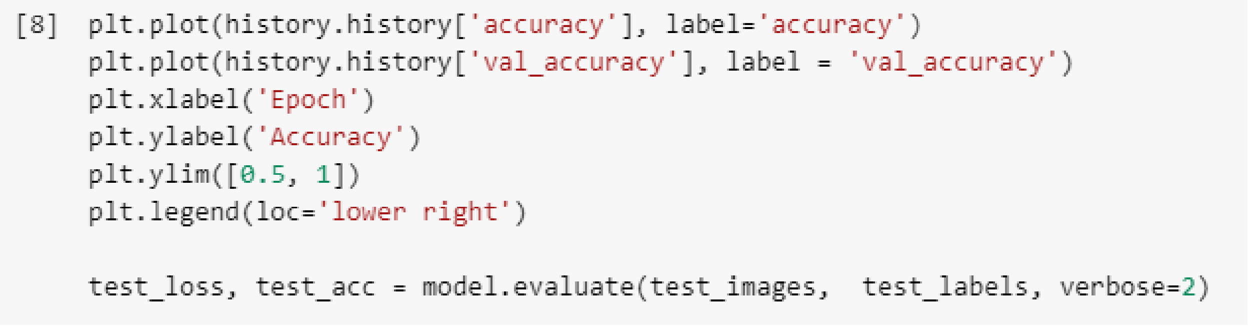
[5] - Create model

[6] - Output model summary



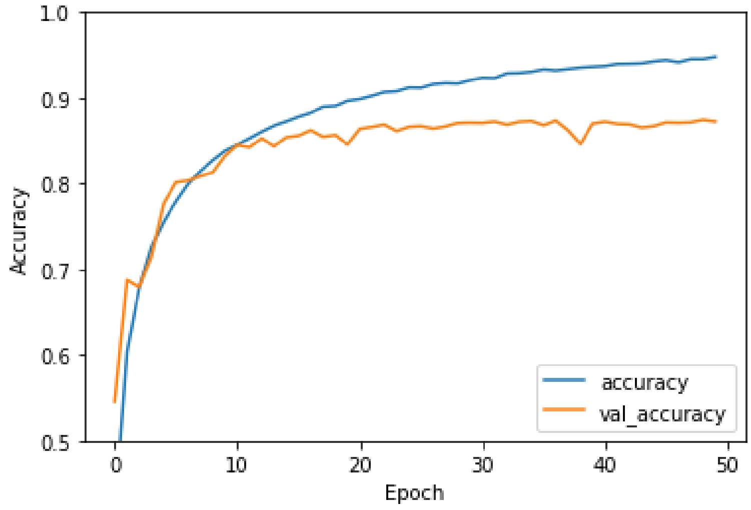
```
[7] model.compile(optimizer='adam',
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])

    history = model.fit(train_images, train_labels, epochs=50,
                        validation_data=(test_images, test_labels))
```

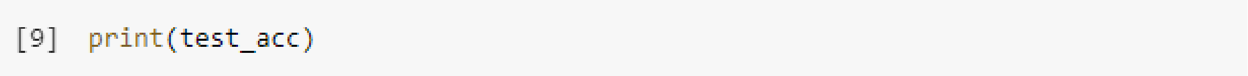
[7] - Compile and train model



```
[8]  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)
```

```
313/313 - 1s - loss: 0.4849 - accuracy: 0.8722 - 1s/epoch - 3ms/step
```

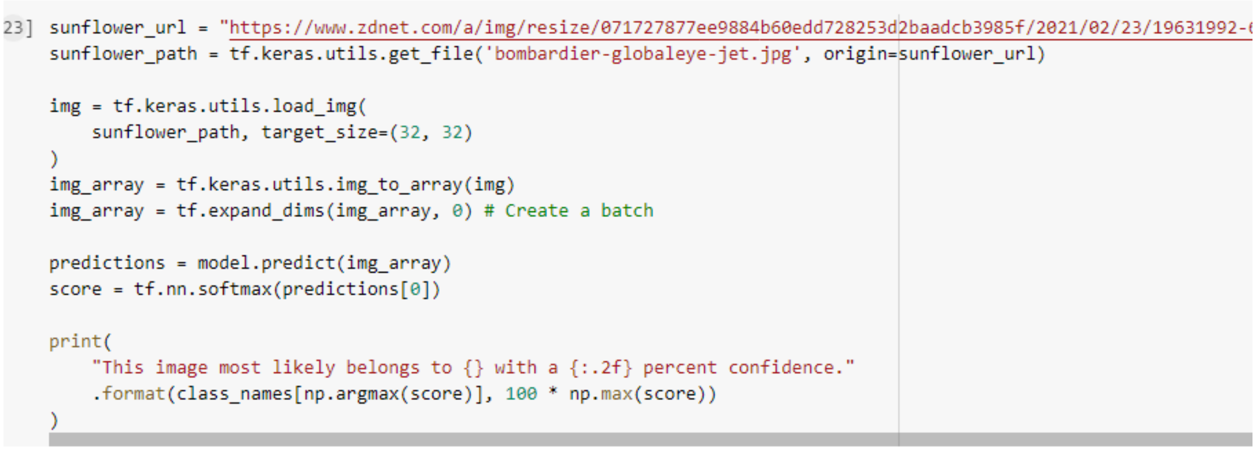




```
[9]  print(test_acc)
```

[8] - Print model plot

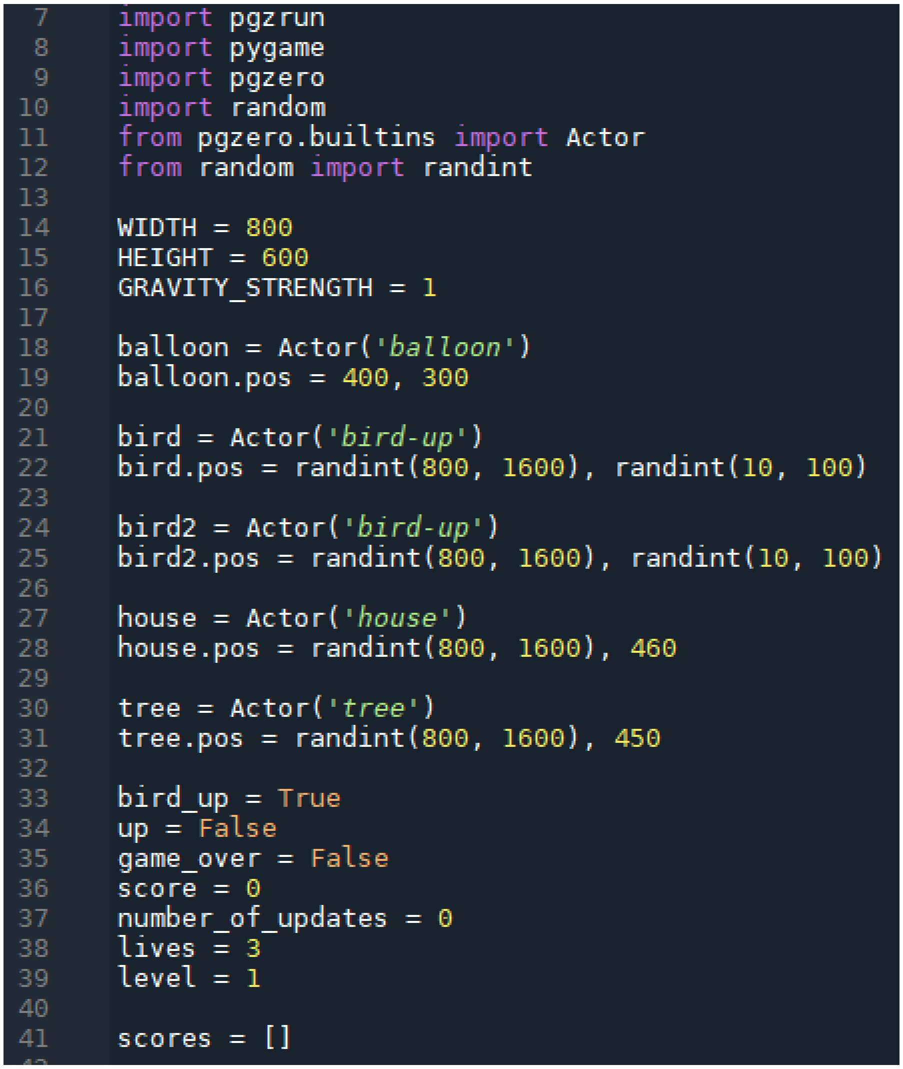
[9] - Print test accuracy



```
23] sunflower_url = "https://www.zdnet.com/a/img/resize/071727877ee9884b60edd728253d2baadcb3985f/2021/02/23/19631992-
    sunflower_path = tf.keras.utils.get_file('bombardier-globaleye-jet.jpg', origin=sunflower_url)

    img = tf.keras.utils.load_img(
        sunflower_path, target_size=(32, 32)
    )
    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) # Create a batch

    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])

    print(
        "This image most likely belongs to {} with a {:.2f} percent confidence."
        .format(class_names[np.argmax(score)], 100 * np.max(score))
    )
```

Use model to recognize images

(2) Balloon Flight [GAME]



```
 7    import pgzrun
 8    import pygame
 9    import pgzero
10    import random
11    from pgzero.builtins import Actor
12    from random import randint
13
14    WIDTH = 800
15    HEIGHT = 600
16    GRAVITY_STRENGTH = 1
17
18    balloon = Actor('balloon')
19    balloon.pos = 400, 300
20
21    bird = Actor('bird-up')
22    bird.pos = randint(800, 1600), randint(10, 100)
23
24    bird2 = Actor('bird-up')
25    bird2.pos = randint(800, 1600), randint(10, 100)
26
27    house = Actor('house')
28    house.pos = randint(800, 1600), 460
29
30    tree = Actor('tree')
31    tree.pos = randint(800, 1600), 450
32
33    bird_up = True
34    up = False
35    game_over = False
36    score = 0
37    number_of_updates = 0
38    lives = 3
39    level = 1
40
41    scores = []
```

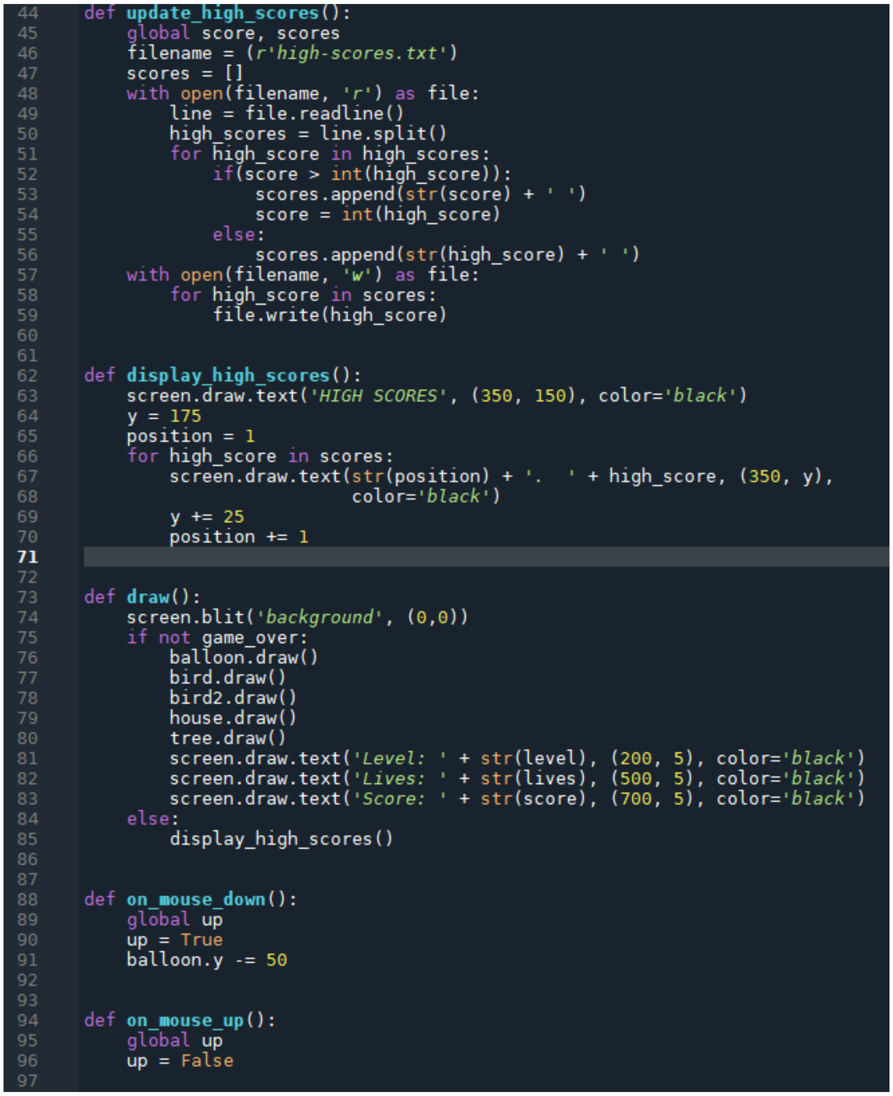
[7-12] - import libraries

[14-16] - set variables

[18-31] - set and place actors

[33-39] - set variables

[41] - set scores array



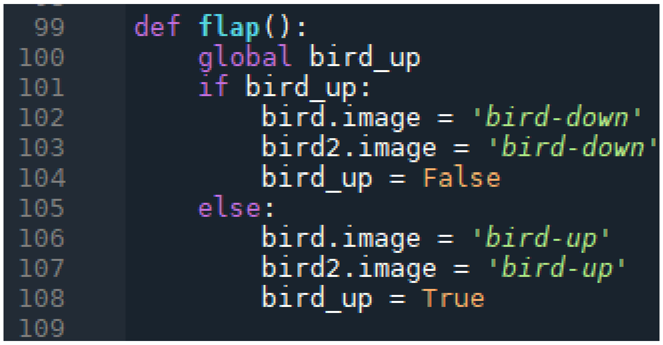
```
44  def update_high_scores():
45      global score, scores
46      filename = (r'high-scores.txt')
47      scores = []
48      with open(filename, 'r') as file:
49          line = file.readline()
50          high_scores = line.split()
51          for high_score in high_scores:
52              if(score > int(high_score)):
53                  scores.append(str(score) + ' ')
54                  score = int(high_score)
55              else:
56                  scores.append(str(high_score) + ' ')
57      with open(filename, 'w') as file:
58          for high_score in scores:
59              file.write(high_score)
60
61
62  def display_high_scores():
63      screen.draw.text('HIGH SCORES', (350, 150), color='black')
64      y = 175
65      position = 1
66      for high_score in scores:
67          screen.draw.text(str(position) + '.  ' + high_score, (350, y),
68                           color='black')
69          y += 25
70          position += 1
71
72
73  def draw():
74      screen.blit('background', (0,0))
75      if not game_over:
76          balloon.draw()
77          bird.draw()
78          bird2.draw()
79          house.draw()
80          tree.draw()
81          screen.draw.text('Level: ' + str(level), (200, 5), color='black')
82          screen.draw.text('Lives: ' + str(lives), (500, 5), color='black')
83          screen.draw.text('Score: ' + str(score), (700, 5), color='black')
84      else:
85          display_high_scores()
86
87
88  def on_mouse_down():
89      global up
90      up = True
91      balloon.y -= 50
92
93
94  def on_mouse_up():
95      global up
96      up = False
97
```

[44-59] - updates high scores

[62-70] - displays high scores on game over

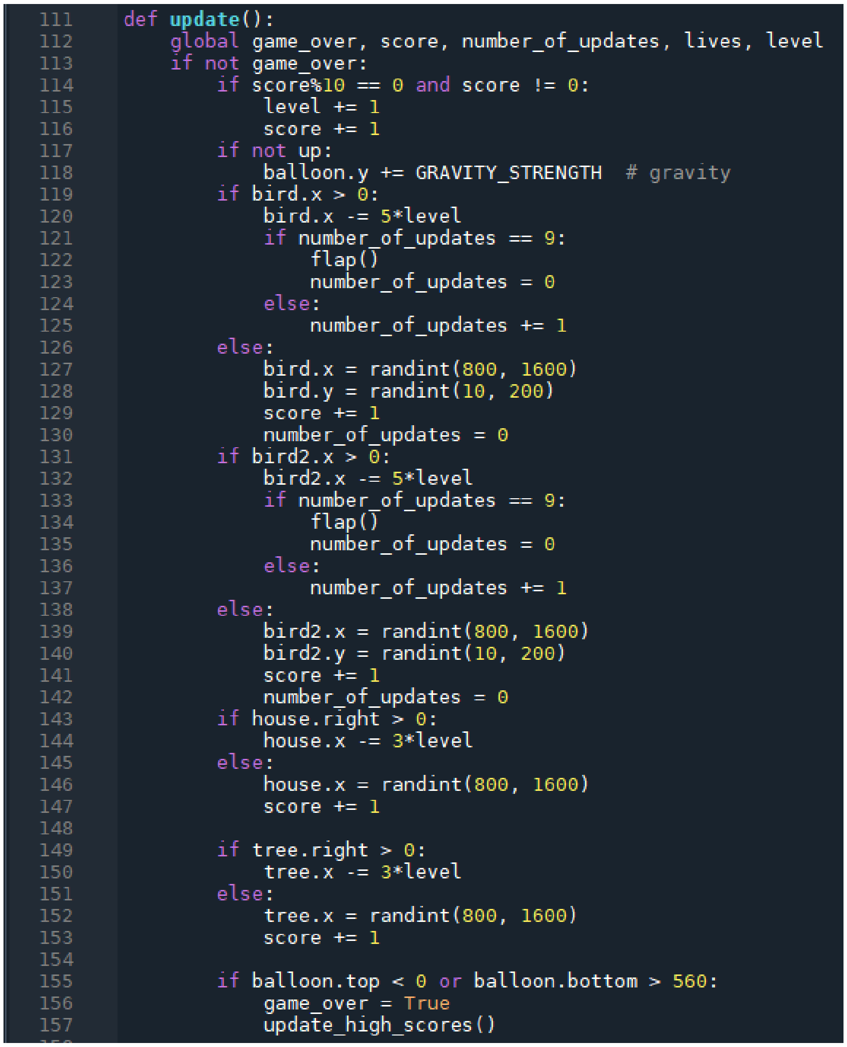
[73] - draws game graphics

[88-96] - functions that are called whenever the mouse is clicked

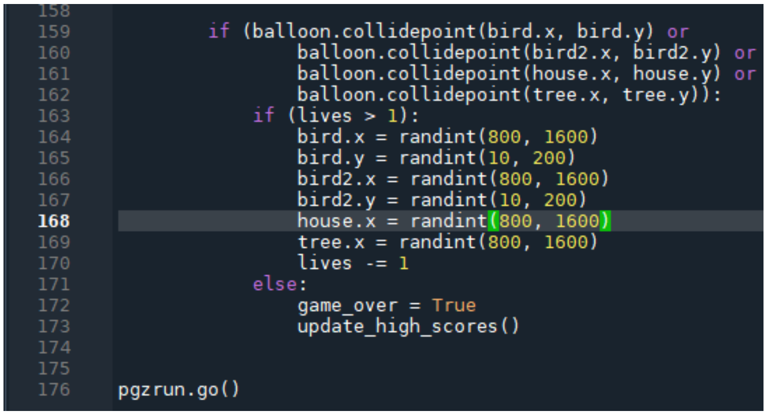


```
 99     def flap():
100         global bird_up
101         if bird_up:
102             bird.image = 'bird-down'
103             bird2.image = 'bird-down'
104             bird_up = False
105         else:
106             bird.image = 'bird-up'
107             bird2.image = 'bird-up'
108             bird_up = True
109
```

[99-108] - animates bird sprite



```
111 def update():
112     global game_over, score, number_of_updates, lives, level
113     if not game_over:
114         if score%10 == 0 and score != 0:
115             level += 1
116             score += 1
117         if not up:
118             balloon.y += GRAVITY_STRENGTH  # gravity
119         if bird.x > 0:
120             bird.x -= 5*level
121             if number_of_updates == 9:
122                 flap()
123                 number_of_updates = 0
124             else:
125                 number_of_updates += 1
126         else:
127             bird.x = randint(800, 1600)
128             bird.y = randint(10, 200)
129             score += 1
130             number_of_updates = 0
131         if bird2.x > 0:
132             bird2.x -= 5*level
133             if number_of_updates == 9:
134                 flap()
135                 number_of_updates = 0
136             else:
137                 number_of_updates += 1
138         else:
139             bird2.x = randint(800, 1600)
140             bird2.y = randint(10, 200)
141             score += 1
142             number_of_updates = 0
143         if house.right > 0:
144             house.x -= 3*level
145         else:
146             house.x = randint(800, 1600)
147             score += 1
148 
149         if tree.right > 0:
150             tree.x -= 3*level
151         else:
152             tree.x = randint(800, 1600)
153             score += 1
154 
155         if balloon.top < 0 or balloon.bottom > 560:
156             game_over = True
157             update_high_scores()
```



```
158
159         if (balloon.collidepoint(bird.x, bird.y) or
160                 balloon.collidepoint(bird2.x, bird2.y) or
161                 balloon.collidepoint(house.x, house.y) or
162                 balloon.collidepoint(tree.x, tree.y)):
163             if (lives > 1):
164                 bird.x = randint(800, 1600)
165                 bird.y = randint(10, 200)
166                 bird2.x = randint(800, 1600)
167                 bird2.y = randint(10, 200)
168                 house.x = randint(800, 1600)
169                 tree.x = randint(800, 1600)
170                 lives -= 1
171             else:
172                 game_over = True
173                 update_high_scores()
174 
175 
176 pgzrun.go()
```

[114-116] - Increments level when score is a multiple of 10

[117-118] - applies gravity to balloon

[119-153] - sets obstacle positions & increases score whenever object touches the left of the screen

[155-157] - game over when balloon touches top or bottom of screen

[159-173] - Lose a life whenever an obstacle is touched

[176] - Run game