Code description

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import numpy as np
import cv2
import time
```

Firstly, we import some libraries to use later in code.

```
cascade_classifiers = [("Frontal face",
"haarcascades/haarcascade_frontalface_default.xml"),
                       ("Full body", "haarcascades/haarcascade_fullbody.xml"),
                       ("Profile face", "haarcascades/haarcascade_profileface.xml")]
```

Then we declare an array of tuples, which contain description and path to classifier. Classifiers are essentially serialized neural networks.

```
print("Select classifier:")

for i, t in enumerate(cascade_classifiers):
    print("["+str(i)+"]: " + t[0])

try:
    ind = int(input('Choose a number: \r\n'))
except ValueError:
    print("Error: Not a number")
    exit()

try:
    cascade = cascade_classifiers[ind][1]
except IndexError:
    print("Error: Value out of range")
    exit()

cascade = cv2.CascadeClassifier(cascade)
```

Here we prompt user to select desired classifier, perform some basic validation of input and select this classifier.

```
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)
```

Now we use `PiCamera` library for creating object representation of camera. We set the camera to use resolution 640x480px with framerate of 32 FPS. Then we wait for camera to initialize with sleep(0.1).

```
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
```

Now we enter a `for` loop, which loops through frames from camera. Frames are in BGR format, which is RGB with swapped blue and red channels. Later we will convert the image from BGR to grayscale for processing, so we would like the most to get a frame in grayscale in the first place. Unfortunately, the camera doesn't support a grayscale mode.

```
image = frame.array
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Here we take the actual image matrix from frame and convert it to grayscale.

```
objects = cascade.detectMultiScale(gray, 1.3, 5)
```

Now we use the neural network for detecting objects, passing the frame and additional parameters - scaleFactor of 1.3 and minNeighbors of 5. Scale factor specifies how much the image size is reduced at each image scale, because every frame is scaled down in steps and then each scaled image is an input to the neural network. This process increases the chance of a match. Min neighbours parameter specifies how many neighbors each candidate rectangle should have to retain it. This parameter will affect the quality of the detected faces: higher value results in less detections but with higher quality.

That's how we detect objects: with just one line of code!

```
for (x,y,w,h) in objects:
        image = cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = image[y:y+h, x:x+w]

    cv2.imshow('video', image)
```

The detectMultiScale() function returns coordinates of rectangle for each detected object, so now we draw these rectangles on the image and display it on screen.

```
rawCapture.truncate(0)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

truncate(0) function flushes the image stream from camera, so that we process only frames that we are able to process in real time. If we omitted this call, we would process each frame from camera, so the displayed video would be delayed (because the Raspberry Pi can process images at about 2 FPS and the input stream has a framerate of 32 FPS) and finally, out frame buffer would overflow.

Then we just check if the user pressed 'q' key on the keyboard and if so, we break from the loop.

```
camera.close()
cv2.destroyAllWindows()
```

Finally, we release the camera and destroy the window in which processed frames were displayed.