Code description

```
import numpy as np
import cv2
```

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

```
cascade = cv2.CascadeClassifier("haarcascades/haarcascade_frontalface_default.xml")
camera = cv2.VideoCapture(0)
```

Then we load a cascade to recognise objects. Classifiers are essentially serialised neural networks.

```
while True:
    ret, image = camera.read()
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Now we enter an infinite loop. In the loop we take an actual image from camera along with return code (whether the capture was successful) and convert the image 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 neighbours 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.

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

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

```
camera.release()
cv2.destroyAllWindows()
```

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