COUNTING MACHINE

MARCH 4, 2023

Table of Contents

Introduction

This project is basically about making a secured counting machine which asked for a password using the Arduino 4x4 keypad then displaying a message on LCD display. The operation is:

- The user is asked to enter a password followed by '=' through the keypad.
- Arduino UNO verifies the password (123456 by default) and sends UART frame to raspberry pi to start operation.
- The user puts the money in front of the camera and pushes the push button to start operation.
- The raspberry pi captures an image through a camera, runs an AI image classification model and calculates the amount of money.
- A UART frame is sent back to Arduino UNO to display the amount of money on the LCD.

Required Hardware:

- Arduino UNO
- Raspberry Pi 3
- Pi Camera
- Push Button
- 20x4 LCD
- Keypad

Required Modules:

- argparse
- pathlib
- openCV
- torch
- numpy

Part 1: Arduino:

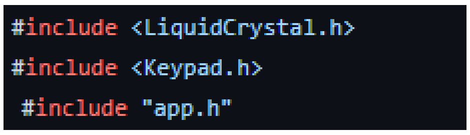
In repo the Arduino folder which contains files:

- Demo.mp4
- main.ino
- app.h
- app.cpp

Libraries, Functions and Constants

The libraries used in project are:

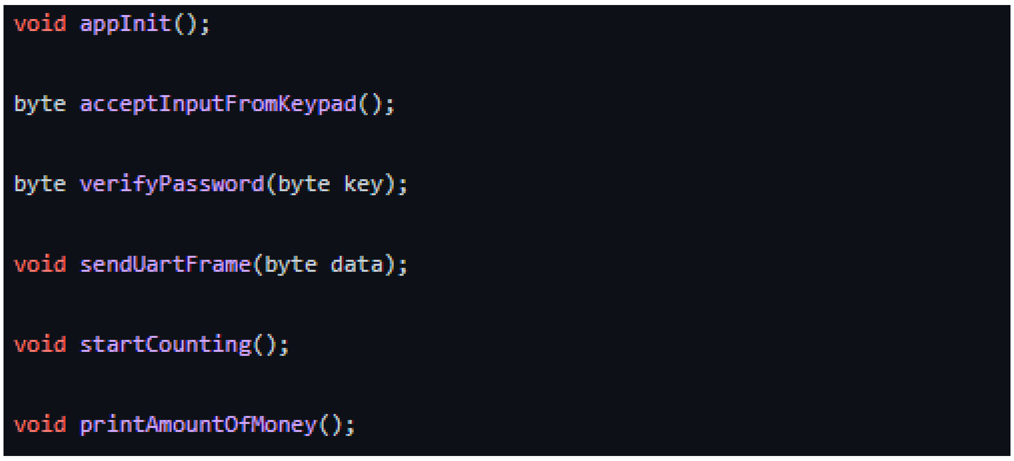
1. Liquid Crystal: allows you to control LCD displays.
2. Keypad: is a library for using matrix style keypads with the Arduino.



```
#include <LiquidCrystal.h>
#include <Keypad.h>
 #include "app.h"
```

Figure 1

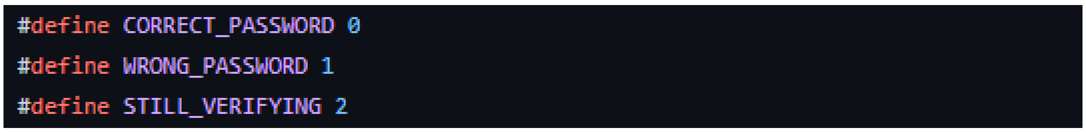
Declaring the functions:



```
void appInit();

byte acceptInputFromKeypad();

byte verifyPassword(byte key);

void sendUartFrame(byte data);

void startCounting();

void printAmountOfMoney();
```

Figure 2

Declaring some constants along the code:



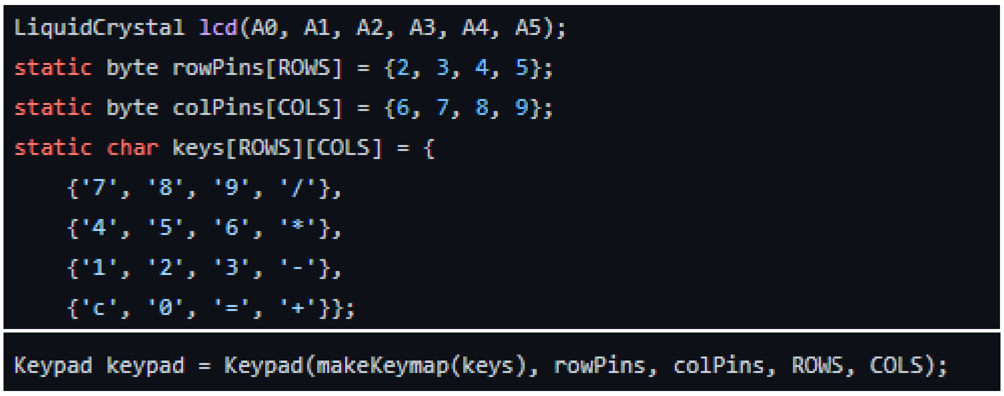
```
#define CORRECT_PASSWORD 0
#define WRONG_PASSWORD 1
#define STILL_VERIFYING 2
```

Figure 3

LCD and Keypad

Declaring LCD and The Keypad

1. 20x4 LCD attached to pins (A0, A1, A2, A3, A4, A5).
2. 4x4 keypad whose row pins are attached to Pins [Rows] = {2, 3, 4, 5}, Pins [Columns] = {6, 7, 8, 9}.



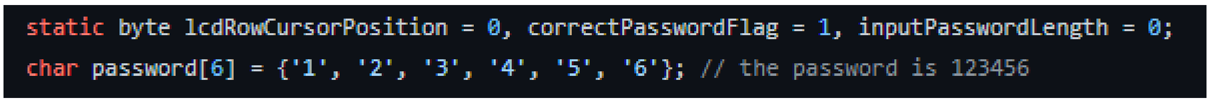
```
LiquidCrystal lcd(A0, A1, A2, A3, A4, A5);
static byte rowPins[ROWS] = {2, 3, 4, 5};
static byte colPins[COLS] = {6, 7, 8, 9};
static char keys[ROWS][COLS] = {
    {'7', '8', '9', '/'},
    {'4', '5', '6', '*'},
    {'1', '2', '3', '-'},
    {'c', '0', '=', '+'}};
```

```
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
```

Figure 4

Password

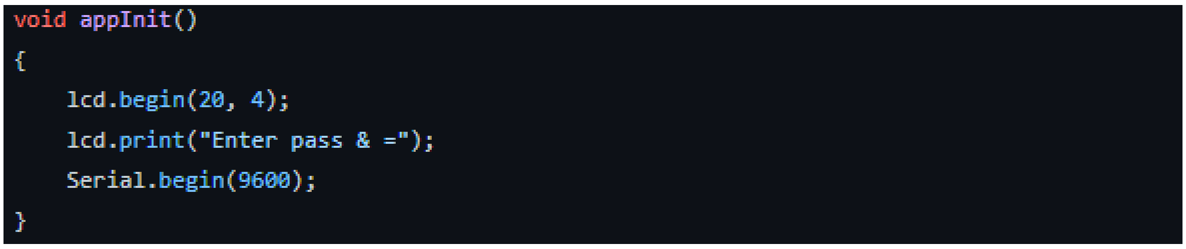
Setting the password in the project



```
static byte lcdRowCursorPosition = 0, correctPasswordFlag = 1, inputPasswordLength = 0;
char password[6] = {'1', '2', '3', '4', '5', '6'}; // the password is 123456
```

Figure 5

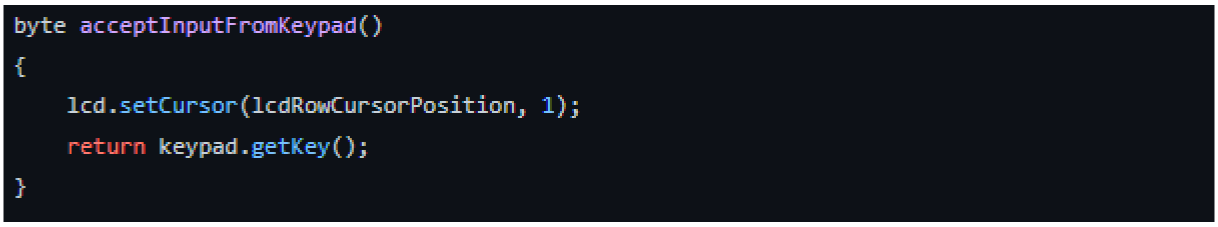
LCD have been started then it has been used for displaying a message “Enter pass” to get the password from the user.



```
void appInit()
{
    lcd.begin(20, 4);
    lcd.print("Enter pass & =");
    Serial.begin(9600);
}
```

Figure 6

Reading the input from keypad

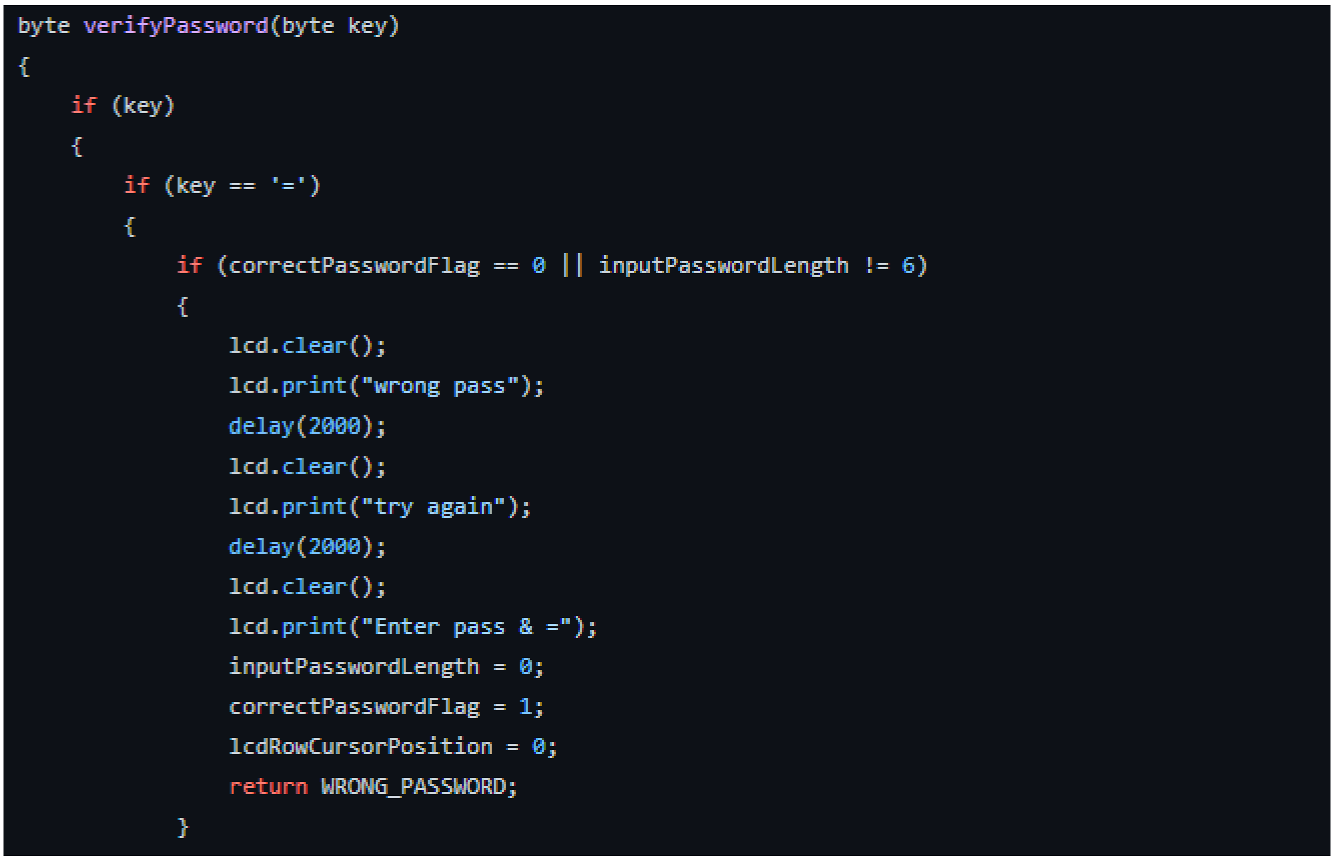


```
byte acceptInputFromKeypad()
{
    lcd.setCursor(lcdRowCursorPosition, 1);
    return keypad.getKey();
}
```

Figure 7

In this step when the user enters “=” the code start checking the password:

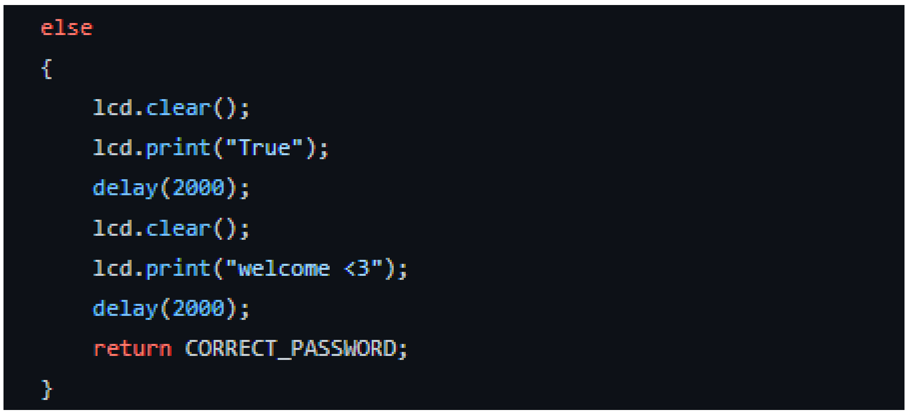
1. If it isn’t correct, it displays messages “wrong pass”, “try again” and returns “WRONG_PASSWORD” which returns “1”.



```
byte verifyPassword(byte key)
{
    if (key)
    {
        if (key == '=')
        {
            if (correctPasswordFlag == 0 || inputPasswordLength != 6)
            {
                lcd.clear();
                lcd.print("wrong pass");
                delay(2000);
                lcd.clear();
                lcd.print("try again");
                delay(2000);
                lcd.clear();
                lcd.print("Enter pass & =");
                inputPasswordLength = 0;
                correctPasswordFlag = 1;
                lcdRowCursorPosition = 0;
                return WRONG_PASSWORD;
            }
```

Figure 8

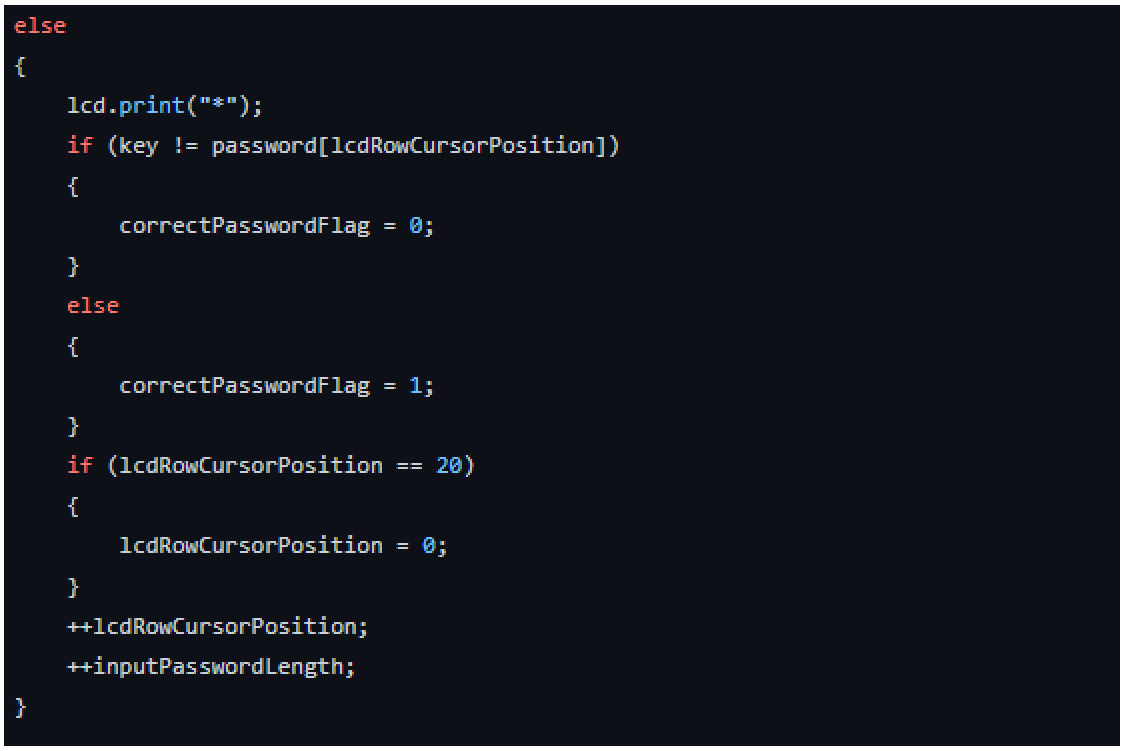
2. If it is correct, it displays messages “True”, “welcome <3” and returns “CORRECT_PASSWORD” which returns “0”.



```
else
{
    lcd.clear();
    lcd.print("True");
    delay(2000);
    lcd.clear();
    lcd.print("welcome <3");
    delay(2000);
    return CORRECT_PASSWORD;
}
```

Figure 9

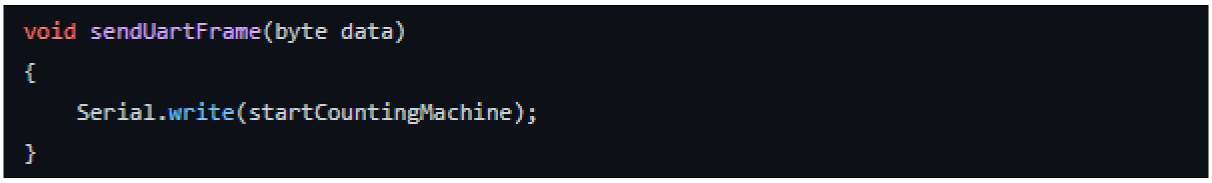
While the user entering the password, LCD will display "*" instead of the characters and passing the cursor position.



```
else
{
    lcd.print("*");
    if (key != password[lcdRowCursorPosition])
    {
        correctPasswordFlag = 0;
    }
    else
    {
        correctPasswordFlag = 1;
    }
    if (lcdRowCursorPosition == 20)
    {
        lcdRowCursorPosition = 0;
    }
    ++lcdRowCursorPosition;
    ++inputPasswordLength;
}
```

Figure 10

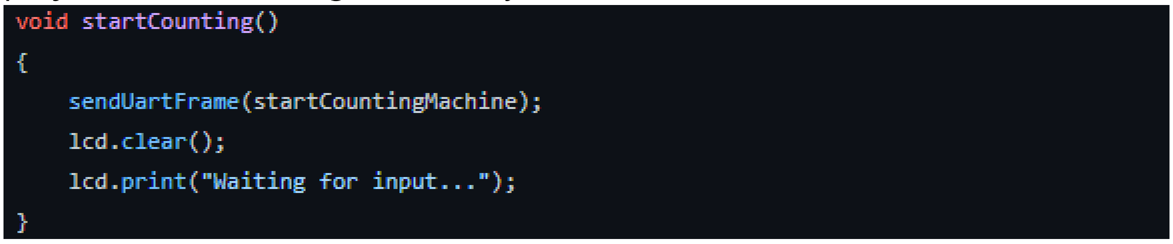
Sending data frames to the Arduino.



```
void sendUartFrame(byte data)
{
    Serial.write(startCountingMachine);
}
```

Figure 11

The password is correct, and the user starts showing money to the camera, the project starts counting the money.

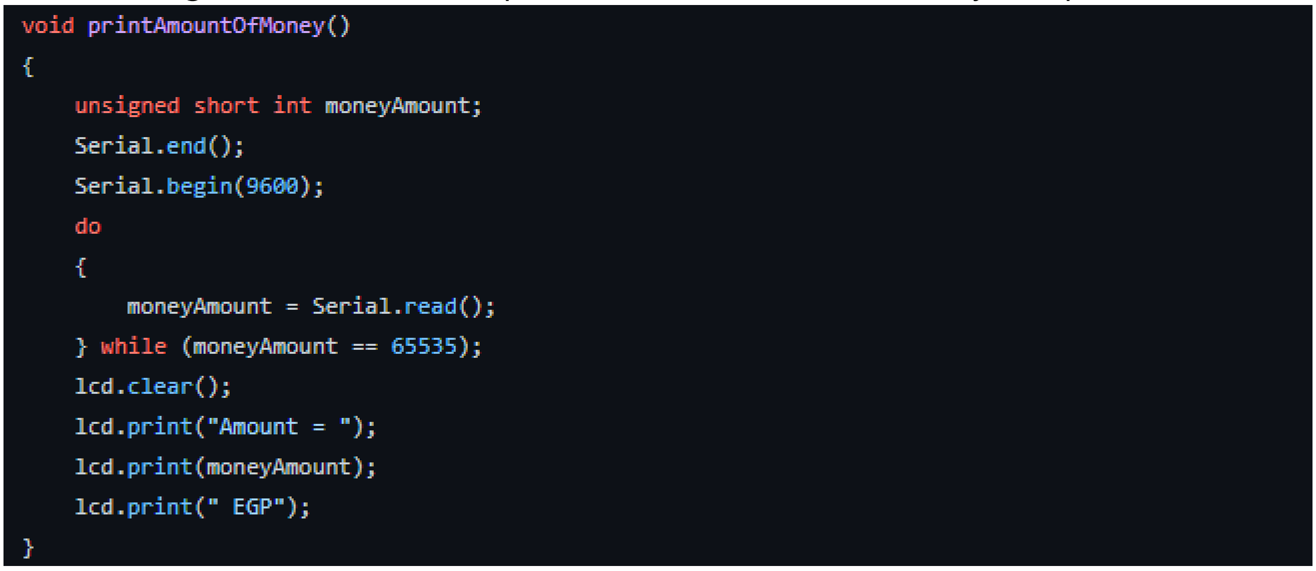


```
void startCounting()
{
    sendUartFrame(startCountingMachine);
    lcd.clear();
    lcd.print("Waiting for input...");
}
```

Figure 12

The output

The counting is done then it will print the “amount of the money” output.



```
void printAmountOfMoney()
{
    unsigned short int moneyAmount;
    Serial.end();
    Serial.begin(9600);
    do
    {
        moneyAmount = Serial.read();
    } while (moneyAmount == 65535);
    lcd.clear();
    lcd.print("Amount = ");
    lcd.print(moneyAmount);
    lcd.print(" EGP");
}
```

Figure 13

Part 2: Raspberry Pi

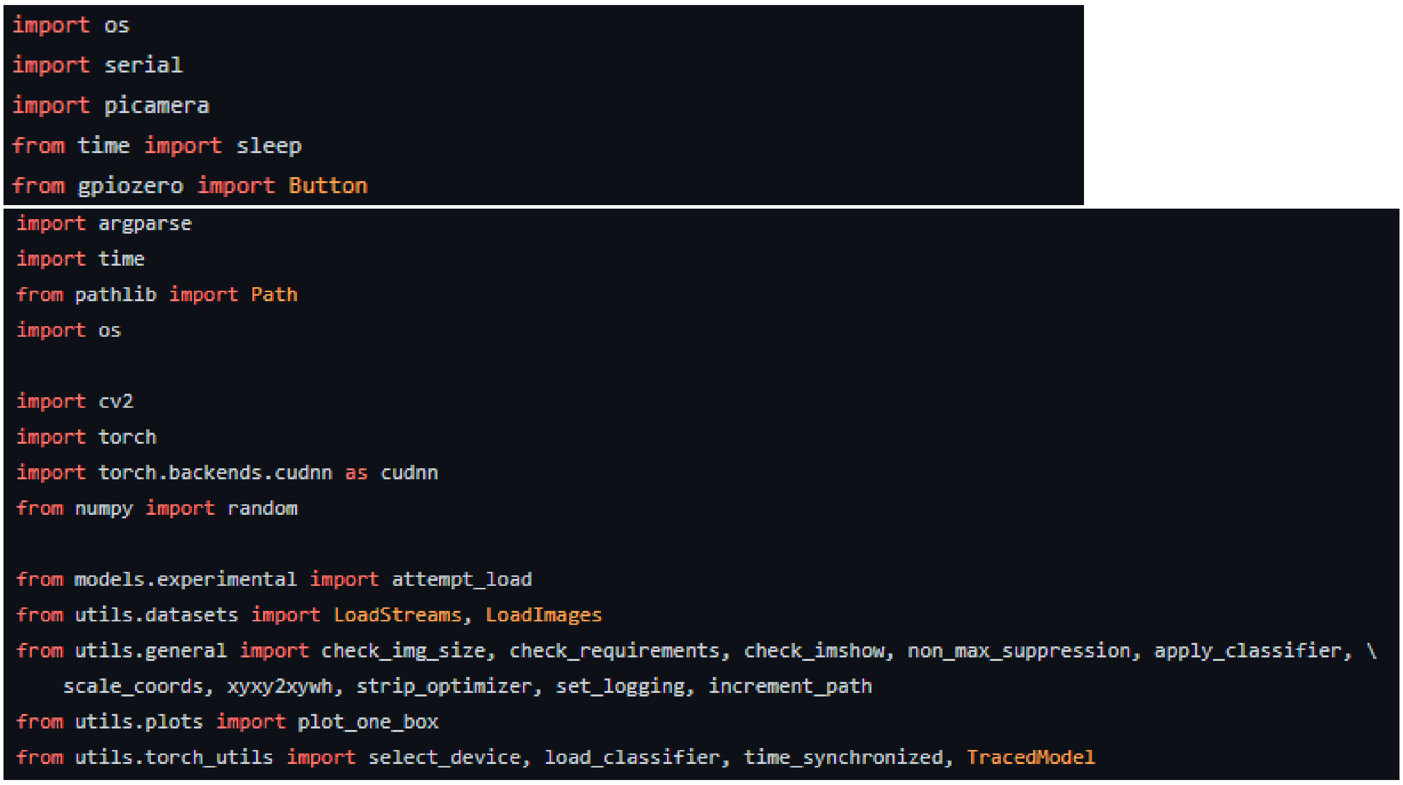
Raspberry Pi which contains files

- detect.py (it represents the image with a rectangular with its value)
- custom_detect.py (it prints the notes in list and makes txt file with predicted labels)
- best.pt (the model file)
- app.py (waits the frame to start and the button to be pushed then the camera starts capturing images and the model will run and sends frame back.)

Libraries and Modules

- The Python serial library allows Raspberry Pi to communicate with UART.
- The picamera library allows to control Camera.
- Reading a button input device with gpiozero. It is essentially a sensor with only two possible states: “pressed” and “not pressed”. If True, the GPIO pin will be pulled high by default. If False, the GPIO pin will be pulled low by default.
- The argparse module makes it easy to write user-friendly command-line interfaces.
- The standard library os and pathlib modules to get timestamps such as the creation, modification, and access date and times of files.
- Path helps creating a nicer API for paths.
- CV2 is the module import name for OpenCV-python.

- torch.backends controls the behavior of various backends that PyTorch supports.



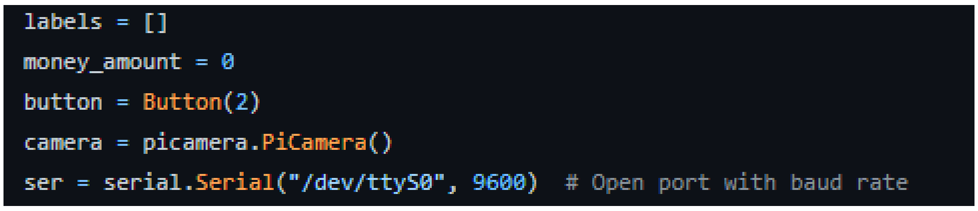
```
import os
import serial
import picamera
from time import sleep
from gpiozero import Button
```

```
import argparse
import time
from pathlib import Path
import os

import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow, non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized, TracedModel
```

Figure 14

Receive Images and transmit data

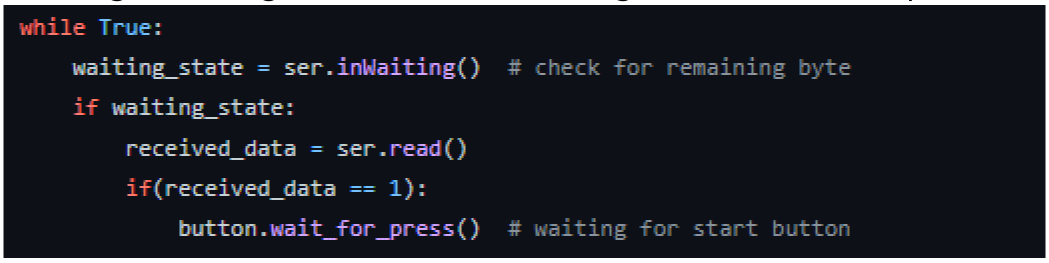
Declaring the GPIO pin which the button is connected to. PiCamera ( ) package provides a pure Python interface to the Raspberry Pi camera module.



```
labels = []
money_amount = 0
button = Button(2)
camera = picamera.PiCamera()
ser = serial.Serial("/dev/ttyS0", 9600)  # Open port with baud rate
```

Figure 15

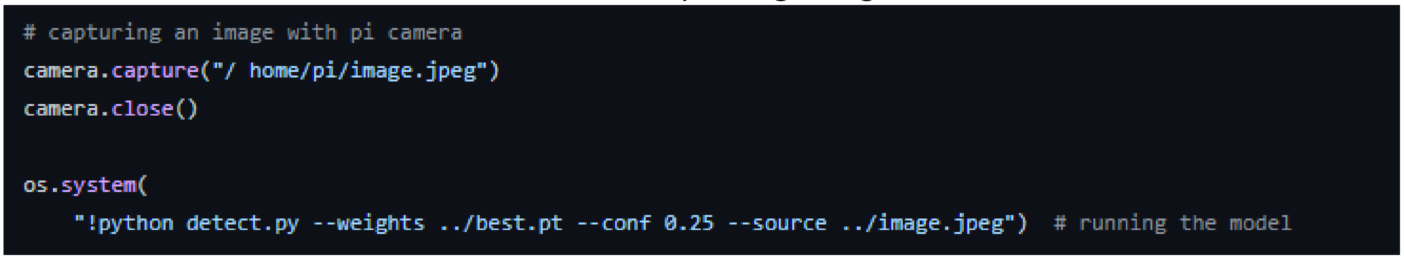
Reading incoming serial data and waiting the button to be pressed.



```
while True:
    waiting_state = ser.inWaiting()  # check for remaining byte
    if waiting_state:
        received_data = ser.read()
        if(received_data == 1):
            button.wait_for_press()  # waiting for start button
```

Figure 16

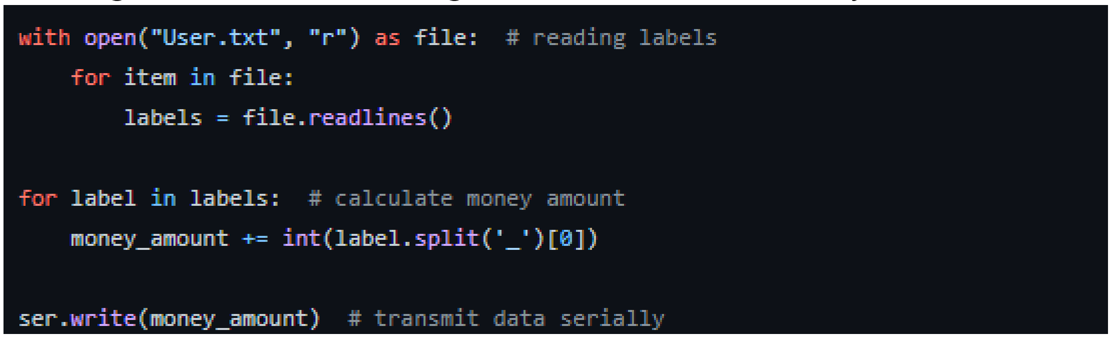
When the button starts, the camera starts capturing images and the model will run.



```python
# capturing an image with pi camera
camera.capture("/ home/pi/image.jpeg")
camera.close()

os.system(
    "!python detect.py --weights ../best.pt --conf 0.25 --source ../image.jpeg")  # running the model
```

Figure 17

Reading labels and calculating the amount of the money then transmitting data



```python
with open("User.txt", "r") as file:  # reading labels
    for item in file:
        labels = file.readlines()

for label in labels:  # calculate money amount
    money_amount += int(label.split('_')[0])

ser.write(money_amount)  # transmit data serially
```

Figure 18