EXPERIMENT NO. 8 (Group B)

- **Aim:** Write a program that asks the user for a number and outputs the number squared that is entered

- **Outcome:** Connectivity, configuration and serial communication with Arduino.

- **Hardware Requirement:** Arduino, USB Cable etc.

- **Software Requirement**: Arduino IDE

Theory:

- **Arduino Serial Monitor for Beginners**

Arduino serial monitor for beginners in electronics. Send and receive data between the serial monitor window on a computer and an Arduino. The serial monitor is a utility that is part of the Arduino IDE. Send text from an Arduino board to the serial monitor window on a computer. In addition, send text from the serial monitor window to an Arduino board. Communications between the serial monitor and Arduino board takes place over the USB connection between the computer and Arduino.

- **Demonstration of the Arduino Serial Monitor for Beginners**

Part 2 of this Arduino tutorial for beginners shows how to install the Arduino IDE. In addition, it shows how to load an example sketch to an Arduino. It is necessary to know how to load a sketch to an Arduino board in this part of the tutorial. Therefore, first finish the previous parts of this tutorial before continuing with this part. A sketch loaded to an Arduino board demonstrates how the serial monitor works in the sub-sections that follow.

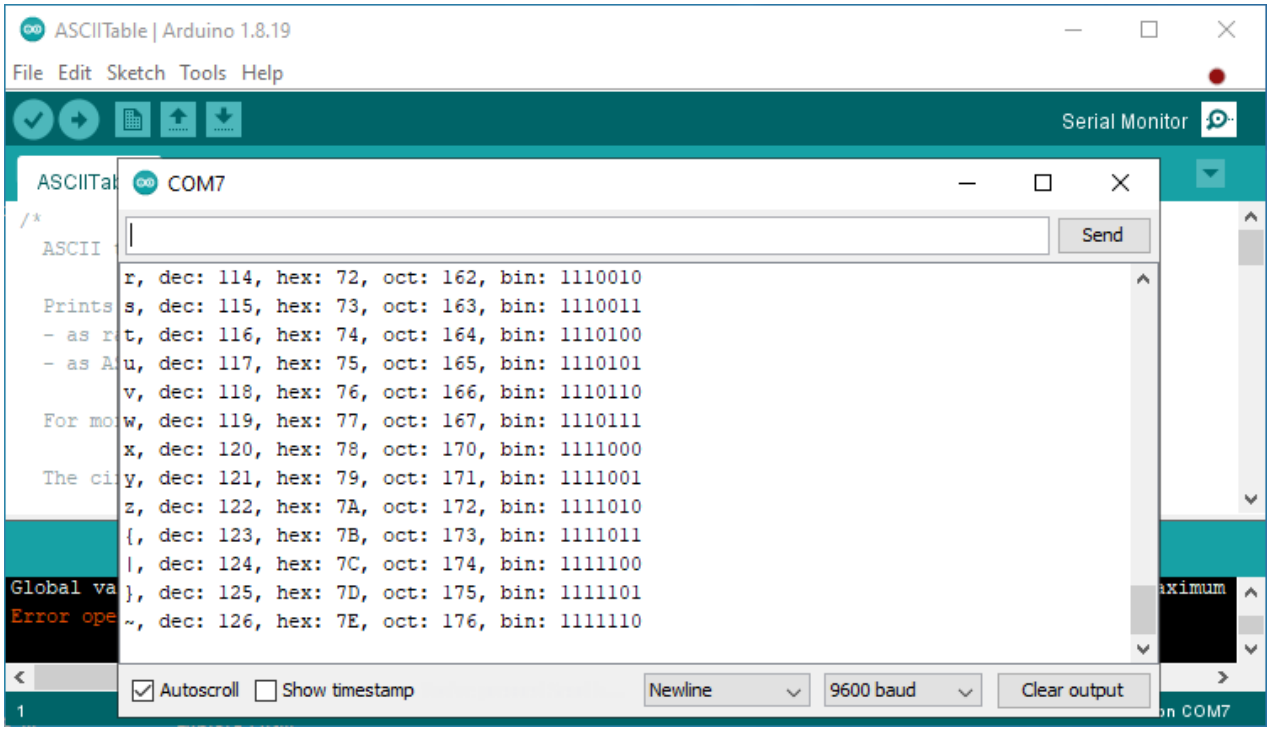
- **Load an Example Sketch that uses the Serial Monitor to an Arduino Board**

Start the Arduino IDE application. Select File → Examples → 04.Communication → ASCIITable from the top Arduino IDE menu bar. As a result, the ASCIITable example sketch opens in a new Arduino IDE window. Upload the ASCIITable example sketch to the Arduino Uno or MEGA 2560 board.

After the ASCIITable sketch is uploaded, nothing is seen to happen. This is because this example sketch sends text out of the USB port of the Arduino board. Because there is nothing running on the computer to receive this text, nothing is seen.

- **How to Open the Arduino Serial Monitor Window for Beginners**

The following image shows the location of the serial monitor window icon on the Arduino IDE toolbar. A red dot near the top right of the image shows the serial monitor toolbar icon location.



Click the Serial Monitor icon near the top right of the Arduino IDE to open the serial monitor window. The above image shows the serial monitor window opened, and on top of the Arduino IDE window. Because the ASCIITable example is loaded on the Arduino board, when the serial monitor window opens, the Arduino sends text to the serial monitor window. This is also because opening the serial monitor window resets the Arduino board, causing the ASCIITable sketch to run from the beginning again.

The ASCIITable sketch sends text out of the USB port of the Arduino. Because the serial monitor is connected to the USB port, it receives the text and displays it in the big receive area of the window. As a result, text scrolls on the serial monitor window for a while. The text then stops because the Arduino has finished sending text. Use the right scrollbar in the serial monitor window to scroll up. Scrolling up reveals all of the text that the Arduino sent.

- **What to do When Junk Characters are Displayed**

When junk, or garbage characters, or even nothing is displayed in the serial monitor, it is usually because of an incorrect baud rate setting. Look at the bottom of the serial monitor in the above image. Notice the value 9600 baud in a box. This is the baud setting of communications between the Arduino and serial monitor. The ASCIITable, and most other built-in example sketches, set the Arduino to communicate at 9600 baud. If your serial monitor window shows a different baud rate, change it to 9600 baud. Do this by clicking the baud drop-down list. Select 9600 baud on the list that drops down.

- **Reset the Arduino Board with the RESET Button**

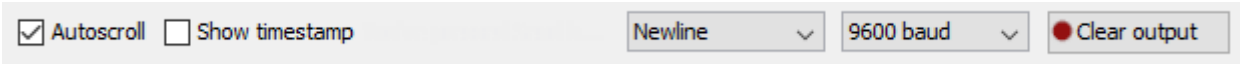
Press and release the RESET button on the Arduino board and the ASCIITable sketch runs from the beginning again. As a result of the reset, the same text scrolls down the serial monitor window and then stops again. The RESET button is the only push button on the Arduino Uno or MEGA 2560.

Pushing the RESET button in holds the board in reset. This means that the sketch currently loaded on the board stops running. Releasing the RESET button takes the board out of reset. As a result, the

sketch currently loaded on the Arduino starts running from the beginning again.

- **Clear the Serial Monitor Window Receive Area**

The red dot in the image below shows the location of the Clear output button at the bottom of the serial monitor window. Click the Clear output button and text is cleared from the receive area of the serial monitor window. Reset the Arduino, and the receive area fills with text from the ASCIITable sketch again.



Serial Monitor Window Clear Output Button

- **What the ASCIITable Sketch Does**

ASCII stands for American Standard Code for Information Interchange. ASCII is a standard way that uses numbers to represent various characters. For example, the decimal number 65 represents the letter A. Another example is the decimal number 125 represents a closing brace: }. This allows computers to send and receive text by sending and receiving numbers. For example when a computer receives the number 65, it knows to display the letter A.

The ASCIITable sketch sends the numbers 33 through to 126 out of the USB port. This results in the printable text characters from the ASCII table displayed in the serial monitor window. In addition to the ASCII characters, the number that represents each character is displayed. Each number is shown in four different numbering systems. These are the decimal, hexadecimal, octal and binary number systems. In the serial monitor window, these number systems are abbreviated to dec, hex, oct and bin.

Conclusion: -

_____

_____

_____

_____

_____

_____

Assignment Questions

1. What is an LCD interface, and how does it work with an Arduino board?
2. What are the different types of LCD interfaces that can be used with an Arduino board?
3. How do you connect an LCD to an Arduino board, and what are the steps involved in setting it up?
4. How do you use the LiquidCrystal library in Arduino to control the LCD interface?
5. How do you display text on an LCD screen using an Arduino board?
6. Explain Serial.readString() and string to interger function ?

EXPERIMENT NO. 9 (Group B)

- **Aim:** Write a program to control the color of the LED by turning potentiometers.
- **Outcome:** Connectivity, configuration and control of LED using Arduino circuit under different conditions.
- **Hardware Requirement:** Arduino, LED, 220 ohm resistor etc.
- **Software Requirement**: Arduino IDE
- **Theory:**

The problem statement is like Arduino traffic light, a fun little project that you can build in under an hour. Here's how to build your own using an Arduino, and how to change the circuit for an advanced variation.

What You Need to Build an Arduino Traffic Light Controller

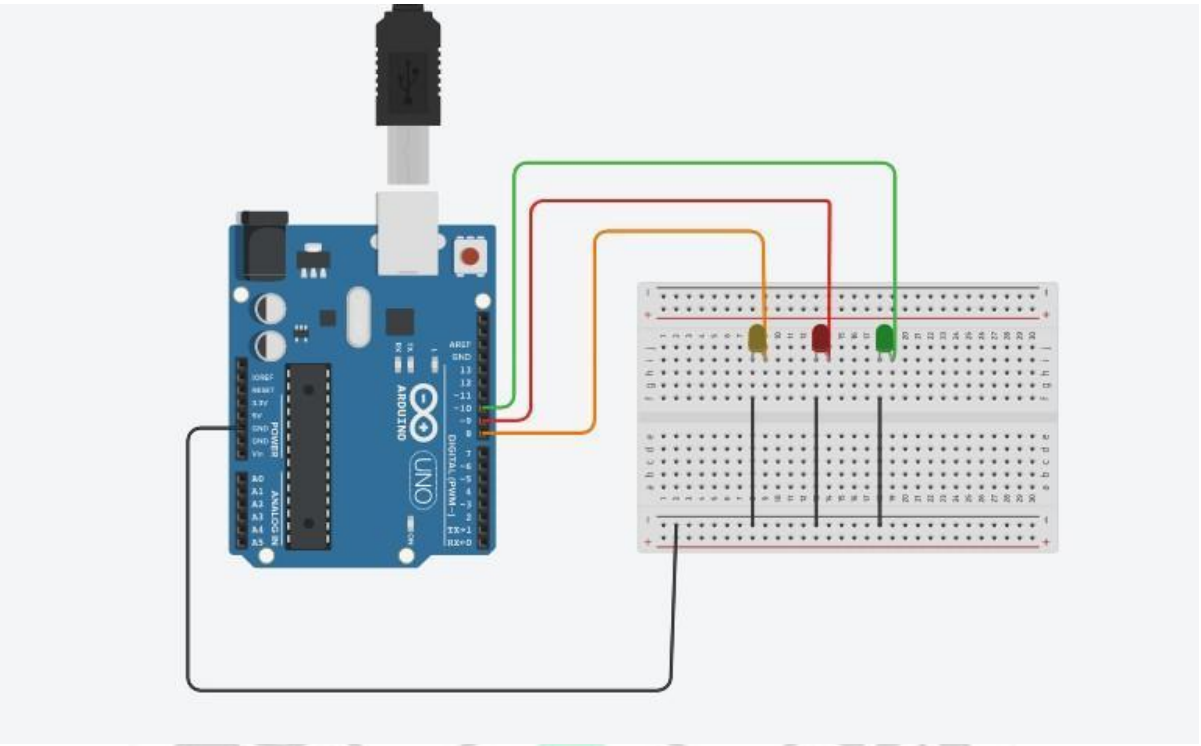
Apart from the basic Arduino, you'll need:

- 1 x 10k-ohm resistor
- 1 x pushbutton switch
- 6 x 220-ohm resistors
- A breadboard
- Connecting wires
- Red, yellow and green LEDs

Arduino Traffic Light: The Basics

Let's start small. A basic, single traffic light is a good place to start. Here's the circuit:

Connect the anode (long leg) of each LED to digital pins eight, nine, and ten (via a 220-ohm resistor). Connect the cathodes (short leg) to the Arduino's ground.



Code for the Arduino Traffic Light

Start by defining variables so that you can address the lights by name rather than a number. Start a new Arduino project, and begin with these lines:

```
int red =
10; int
```

Next, let's add the setup function, where you'll configure the red, yellow and green LEDs to be outputs. Since you have created variables to represent the pin numbers, you can now refer to the pins by name instead:

```
   void setup(){
pinMode(red,
OUTPUT);
  pinMode(yellow,
```

The **pinMode** function configures the Arduino to use a given pin as an output. You have to do this for your LEDs to work at all. Now for the actual logic of the traffic light. Here's the code you need. Add this below your variable definitions and setup function:

```
  void
loop(){
changeLights(
);
delay(15000)
;
```

```
    // turn off yellow, then turn red on
for 5 seconds digitalWrite(yellow, LOW);
    digitalWrite(red,
HIGH); delay(5000);
    // red and yellow on for 2 seconds (red is already
on though) digitalWrite(yellow, HIGH);
    delay(2000);
    // turn off red and yellow, then
turn on green digitalWrite(yellow,
```

Upload this code to your Arduino, and run (make sure to select the correct board and port from the **Tools** > Board and **Tools** > **Port** menus). You should have a working traffic light that changes every 15 seconds, like this (sped up):

Let's break down this code. The **changeLights** function performs all the hard work. This rotates the traffic light through yellow and red, then back to green. As this gets called inside the **loop** function, the Arduino will run this code forever, with a 15-second pause every time.

The **changeLights** function consists of four distinct steps:

- Green on, yellow off
- Yellow off, red on
- Yellow on, red on
- Green on, red off, yellow off

These four steps replicate the process used in real traffic lights. For each step, the code is very similar. The appropriate LED gets turned on or off using **digitalWrite**. This is an Arduino function used to set output pins to HIGH (for on), or LOW (for off).

After enabling or disabling the required LEDs, the **delay** makes the Arduino wait for a given amount of time. Three seconds in this case.

Going Deeper: Arduino Pedestrian Crossing

Now that you know the basics, let's improve it. Add in a pushbutton for pedestrians to change the light whenever they like:

Notice how the traffic light is exactly the same as the previous example. Connect the button to digital pin 12. You'll notice that the switch has a high-impedance 10k-ohm resistor attached to it, and you may be wondering why. This is a pull-down resistor.

A switch either lets the current flow or doesn't. This seems simple enough, but in a logic circuit, the current should be always flowing in either a high or low state (remember, 1 or 0, HIGH or LOW). You might assume that a pushbutton switch that isn't actually pressed would be in a LOW state, but in fact, it's said to be 'floating', because no current gets drawn at all.In this floating state, it's possible that a false reading will occur as it fluctuates with electrical interference. In other words, a floating switch is giving neither a reliable HIGH nor LOW reading. A pull-down resistor keeps a small amount of current flowing when the switch gets closed, thereby ensuring an accurate low state reading.

In other logic circuits, you may find a pull-up resistor instead, and this works on the same principle, but in reverse, making sure that particular logic gate defaults to high.

Now, in the loop part of the code, instead of changing the lights every 15 seconds, you're going to read the state of the pushbutton switch instead, and only change the lights when it's activated.

Code for the Arduino Pedestrian Crossing

Start by adding a new variable to store your button pin:

```
int button = 12; // switch is on pin 12
```

Now, in the setup function, add a new line to declare the switch as an input. Add a line to set the traffic lights to the green stage. Without this initial setting, they would off until the first time **changeLights** runs.

```
pinMode(button,
INPUT)
```

Change the entire loop function to the following instead:

```
void loop() {
        if (digitalRead(button)
   == HIGH){ delay(15); //
   software debounce if
   (digitalRead(button) ==
   HIGH) {
        // if the switch is HIGH, ie· pushed down -
```

That should do it. You may be wondering why the button checking happens twice (**digitalRead(button)**), separated by a small delay. This is debouncing. Much like the pull-down resistor for the button, this simple check stops the code detecting minor interference as a button press.

By waiting inside the **if** statement for 15 seconds, the traffic lights can't change for at least that duration. Once 15 seconds is over the loop restarts. Each restart of the loop, it reads

the state of the button again, but if it isn't pressed, the **if** statement never activates, the lights never change, and the program restarts again.

Here's how this looks (sped up):

Arduino Traffic Light with Junction

Let's try a more advanced model.

Instead of a pedestrian crossing,

change your circuit to have two

traffic lights:

Connect the second traffic light to digital pins 11, 12, and 13.

Code for the Arduino Traffic Light with Junction

First, assign your new traffic light pins to variables, and configure them as outputs, like in the first example:

```
// light one
int red1 =
10;
int yellow1 =
9; int green1
= 8;
// light two
int red2 =
13;
int yellow2 =
12; int green2
= 11; void
setup(){
```

Now, update your loop to use the code from the first example (instead of the pedestrian crossing):

```
   void loop(){
 changeLights()
 ;
```

Once again, all the work is carried out in the **changeLights** function. Rather than going **red** > **red** & **yellow** > **green**, this code will alternate the traffic lights. When one is on green, the other is on red. Here's the code:

```
void changeLights(){
      // turn both yellows
 on digitalWrite(green1,
 LOW);
 digitalWrite(yellow1,
 HIGH);
 digitalWrite(yellow2,
 HIGH); delay(5000);
      // turn both yellows off, and opposite green and
```

```
    delay(5000);
    // both yellows on
again digitalWrite(yellow1,
HIGH);
digitalWrite(yellow2,
HIGH);
digitalWrite(green2,
LOW); delay(3000);
    // turn both yellows off, and opposite green and
red digitalWrite(green1, HIGH);
```

Conclusion: -

_____

_____

_____

_____

_____

_____

Assignment Questions:

Q1. Explain formula of calculating analog write values?

Q2. What is PWM ?

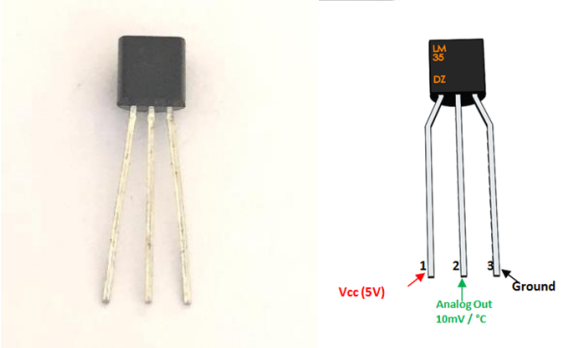
Q3. Explain any 3 Analog Sensors ?

EXPERIMENT NO. 10 (Group B)

Aim: Write a program so it displays the temperature in Fahrenheit as well as the maximum and minimum temperatures it has seen.

Outcome: Understanding working principle of DHT11, LM35 temperature sensor.

- **Hardware Requirement:** Arduino, LED, LM35, DHT11, etc
- **Software Requirement**: Arduino IDE
- **Theory:** **LM35 Temperature Sensor**



LM35 Temperature Sensor Pinout
LM35 Sensor Pinout Configuration

Pin Number	Pin Name	Description
1	Vcc	Input voltage is +5V for typical applications
2	Analog Out	There will be increase in 10mV for raise of every 1°C. Can range from -1V(-55°C) to 6V(150°C)
3	Ground	Connected to ground of circuit

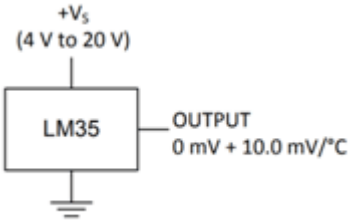
- **LM35 Sensor Features**
  - Minimum and Maximum Input Voltage is 35V and -2V respectively. Typically 5V.
  - Can measure temperature ranging from -55°C to 150°C
  - Output voltage is directly proportional (Linear) to temperature (i.e.) there will be a rise of 10mV (0.01V) for every 1°C rise in temperature.
  - ±0.5°C Accuracy
  - Drain current is less than 60uA
  - Low cost temperature sensor
  - Small and hence suitable for remote applications
  - Available in TO-92, TO-220, TO-CAN and SOIC package

- **LM35 Temperature Sensor Equivalent**
  LM34, DS18B20, DS1620, LM94022

How to use LM35 Temperature Sensor:

LM35 is a precession Integrated circuit Temperature sensor, whose output voltage varies, based on the temperature around it. It is a small and cheap IC which can be used to measure temperature anywhere between -55°C to 150°C. It can easily be interfaced with any Microcontroller that has ADC function or any development platform like Arduino.

Power the IC by applying a regulated voltage like +5V (VS) to the input pin and connected the ground pin to the ground of the circuit. Now, you can measure the temperature in form of voltage as shown below.



If the temperature is 0°C, then the output voltage will also be 0V. There will be a rise of 0.01V (10mV) for every degree Celsius rise in temperature. The voltage can be converted into temperature using the below formulae.

$$V_{OUT} = 10\ mv/°C \times T$$

where

- V_{OUT} is the LM35 output voltage
- T is the temperature in °C

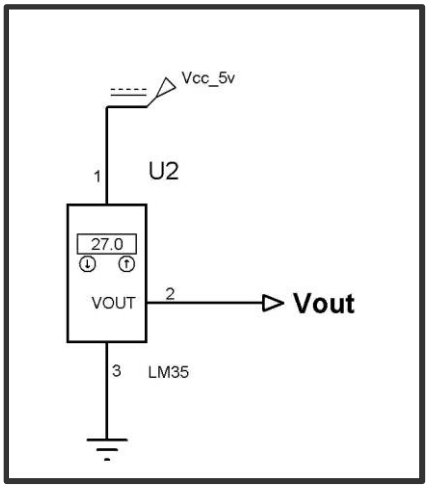
LM35 Temperature Sensor Applications

- Measuring temperature of a particular environment
- Providing thermal shutdown for a circuit/component
- Monitoring Battery Temperature
- Measuring Temperatures for HVAC applications.

- **How Does LM35 Sensor Work?**

  Main advantage of LM35 is that it is linear i.e. 10mv/°C which means for every degree rise in temperature the output of LM35 will rise by 10mv. So if the output of LM35 is 220 mv/0.22V the temperature will be 22°C. So if room temperature is 32°C then the output of LM35 will be 320mv i.e. 0.32V.

- **LM35 Interfacing Circuit**



As such no extra components are required to interface LM35 to ADC as the output of LM35 is linear with 10mv/degree scale. It can be directly interfaced to any 10 or 12

bit ADC. But if you are using an 8-bit ADC like ADC0808 or ADC0804 an amplifier section will be needed if you require to measure 1°C change.

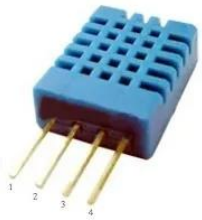
LM35 can also be directly connected to Arduino. The output of LM35 temperature can also be given to comparator circuit and can be used for over temperature indication or by using a simple relay can be used as a temperature controller.

- **DHT11 interfacing with arduino and weather station**

  DHT11 sensor is used to measure the **temperature** and **humidity**. It has a resistive humidity sensing component and a negative temperature coefficient (NTC). An 8 bit MCU is also connected in it which is responsible for its fast response. It is very inexpensive but it gives values of both temperature and humidity at a time.

- **Specification of DHT11**
  - It has humidity range from 20 to 90% RH
  - It has temperature range from 0 – 50 C
  - It has signal transmission range of 20 m
  - It is inexpensive
  - It has fast response and it is also durable

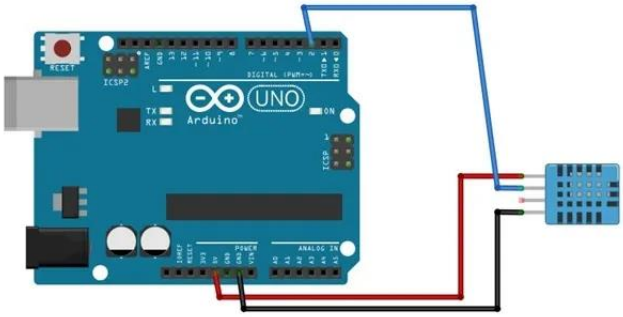
- **DHT11 Pin out**



  - The first pin of the DHT11 is vcc pin.
  - The second pin of the DHT is Data pin.
  - The third pin is not used.
  - The fourth pin of the DHT sensor is ground.

- **DHT11 interfacing with arduino**

  First of all connect the ground and the VCC of the DHT11 temperature and humidity sensor to the ground and 5v of the **Arduino**. Then connect the data pin of the DHT11 sensor to the pin 2 of the Arduino.



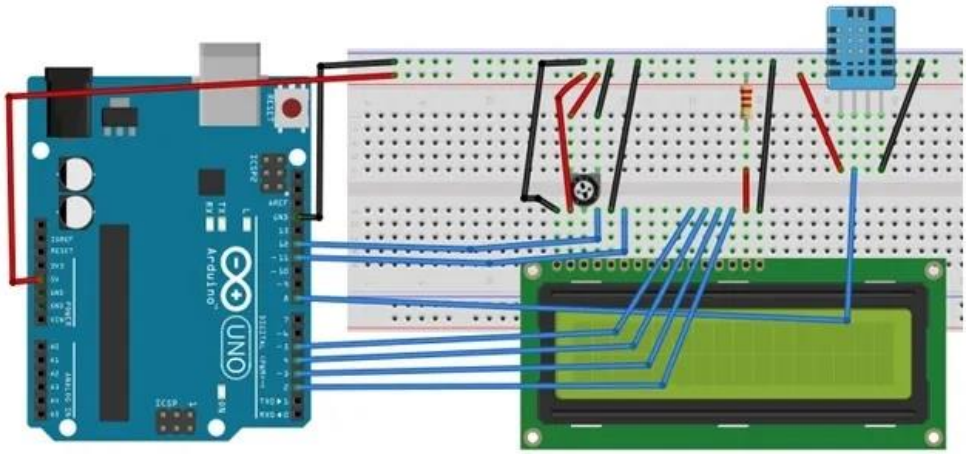
- **Installing the DHT11 Library**
  To run the following code in Arduino IDE you will first have to install the DHT library in you Arduino directory.
  Download the zip file from here and place it in your Arduino library folder. The path to Arduino library folder for my computer is
  Documents/ Arduino/ Libraries
  Unzip the downloaded file and place it in this folder.
  After copying the files, the Arduino library folder should have a new folder named DHT containing the dht.h and dht.cpp. After that copy the following code in the Arduino IDE and upload the code.

- **Code of DHT11 interfacing with arduino**

```
// Code for DHT11 Temperature and humidity sensor.
 #include " DHT.h "
 #define DHTPIN 2
 #define DHTTYPE DHT11
 DHT dht ( DHTPIN, DHTTYPE ) ;
 void setup ()
{
   Serial.begin ( 9600 ) ;
   dht.begin ( ) ;
}
 void loop ( ) {
 float humidity = dht.readHumidity ( ) ;
 float temp = dht.readTemperature ( ) ;
 if ( isnan ( t ) || isnan ( h ) ) {
     Serial.println ( " Sensor not working " ) ;
   }
 else
{

     Serial.print ( " Temp is " ) ;
     Serial.print ( temp ) ;
     Serial.println ( " *C " ) ;
     Serial.print ( " Humidity in % is : " ) ;
     Serial.print ( humidity ) ;
     Serial.print ( " % \t " ) ;

   }
 }
```

- **Weather Station using DHT11 and arduino**

In this example we will make a weather station that will sense the humidity and temperature and will show it on the lcd attached to the Arduino. Make the circuit as shown in the diagram. The resistor in the circuit will make the black light darker. We have used the 220 ohm resistor but you can use any resistor having value near to that. The potentiometer we used in the circuit is used to set the screen contrast. We have used the 10 K ohm value but you can choose any value relative to that one.



- **Components Required**
  - Arduino Uno (you can use any)
  - 16 x 2 LCD
  - DHT11 Temperature and humidity sensor
  - 10 K ohm potentiometer
  - 220 ohm resistor

- **Code of weather station using arduino and DHT11**

```
// This code is for the weather station using the DHT11 humidity and temperature sensor.
// Install the library of the DHT before uploading the code in the Arduino IDE
#include < dht.h >
#include < LiquidCrystal.h >
LiquidCrystal lcd ( 12, 11, 5, 4, 3, 2 ) ;
dht DHT ;                    // declaring dht a variable
#define DHT11_PIN 8          // initializing pin 8 for dht
void setup ( ) {
  lcd.begin ( 16, 2 ) ;      // starting the 16 x 2 lcd
}
void loop ( )
{
int chk = DHT.read11(DHT11_PIN );
lcd.setCursor ( 0, 0 ) ;
lcd.print ( " Temperature is : " ) ;
  lcd.print ( DHT.temperature ) ;
  lcd.print ( ( char ) 223 ) ;
  lcd.print ( " C " ) ;
  lcd.setCursor ( 0 , 1 );
  lcd.print ( " Humidity is : " ) ;
  lcd.print ( DHT.humidity ) ;
  lcd.print ( " % " ) ;
  delay ( 1000 ) ;
}
```

Conclusion: -

_____

_____

_____

_____

_____

Assignment Questions:

Q1. Explain the difference between DHT11 and DHT22 ?

Q2. Draw an interfacing diagram of DHT11 with arduino ?

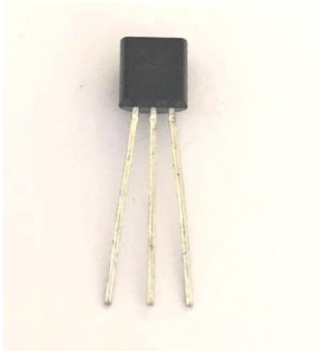
Q3. What are the ways of adding libraries in arduino IDE ?

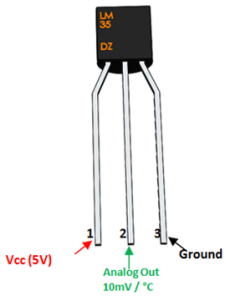
Q4. What is a positive temperature coefficient and Negative Temperature Coefficient ?

EXPERIMENT NO. 11 (Group B)

- **Aim:** Write a program read the temperature sensor and send the values to the serial monitor on the computer
- **Outcome:** Understanding working principle of DHT11, LM35 temperature sensor, Relationship between different temperature scales
- **Hardware Requirement:** Arduino, LED, LM35, DHT11, etc
- **Software Requirement**: Arduino IDE
- **Theory:**

LM35 Temperature Sensor





LM35 Temperature Sensor Pinout
LM35 Sensor Pinout Configuration

Pin Number	Pin Name	Description
1	Vcc	Input voltage is +5V for typical applications
2	Analog Out	There will be increase in 10mV for raise of every 1°C. Can range from -1V(-55°C) to 6V(150°C)
3	Ground	Connected to ground of circuit

- **LM35 Sensor Features**
  - Minimum and Maximum Input Voltage is 35V and -2V respectively. Typically 5V.
  - Can measure temperature ranging from -55°C to 150°C
  - Output voltage is directly proportional (Linear) to temperature (i.e.) there will be a rise of 10mV (0.01V) for every 1°C rise in temperature.
  - ±0.5°C Accuracy
  - Drain current is less than 60uA
  - Low cost temperature sensor
  - Small and hence suitable for remote applications
  - Available in TO-92, TO-220, TO-CAN and SOIC package

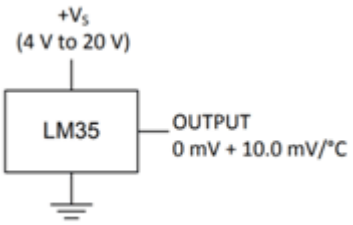
- **LM35 Temperature Sensor Equivalent**

LM34, DS18B20, DS1620, LM94022

How to use LM35 Temperature Sensor:

LM35 is a precession Integrated circuit Temperature sensor, whose output voltage varies, based on the temperature around it. It is a small and cheap IC which can be used to measure temperature anywhere between -55°C to 150°C. It can easily be interfaced with any Microcontroller that has ADC function or any development platform like Arduino.

Power the IC by applying a regulated voltage like +5V (VS) to the input pin and connected the ground pin to the ground of the circuit. Now, you can measure the temperature in form of voltage as shown below.



If the temperature is 0°C, then the output voltage will also be 0V. There will be rise of 0.01V (10mV) for every degree Celsius rise in temperature. The voltage can converted into temperature using the below formulae.

$$V_{OUT} = 10\ mv/°C \times T$$

where

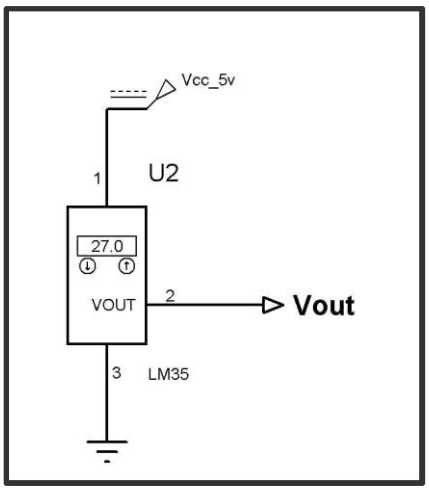
- V_{OUT} is the LM35 output voltage
- T is the temperature in °C

- **LM35 Temperature Sensor Applications**
  - Measuring temperature of a particular environment
  - Providing thermal shutdown for a circuit/component
  - Monitoring Battery Temperature
  - Measuring Temperatures for HVAC applications.

- **How Does LM35 Sensor Work?**

Main advantage of LM35 is that it is linear i.e. 10mv/°C which means for every degree rise in temperature the output of LM35 will rise by 10mv. So if the output of LM35 is 220mv/0.22V the temperature will be 22°C. So if room temperature is 32°C then the output of LM35 will be 320mv i.e. 0.32V.

- **LM35 Interfacing Circuit**



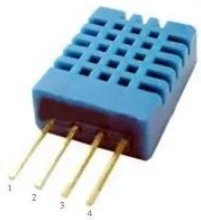
As such no extra components required to interface LM35 to ADC as the output of LM35 is linear with 10mv/degree scale. It can be directly interfaced to any 10 or 12 bit ADC. But if you are using an 8-bit ADC like ADC0808 or ADC0804 an amplifier section will be needed if you require to measure 1°C change.
LM35 can also be directly connected to Arduino. The output of LM35 temperature can also be given to comparator circuit and can be used for over temperature indication or by using a simple relay can be used as a temperature controller.

- **DHT11 interfacing with arduino and weather station**

DHT11 sensor is used to measure the **temperature** and **humidity**. It has a resistive humidity sensing component and a negative temperature coefficient (NTC). An 8 bit MCU is also connected in it which is responsible for its fast response. It is very inexpensive but it gives values of both temperature and humidity at a time.

- **Specification of DHT11**
  - It has humidity range from 20 to 90% RH
  - It has temperature range from 0 – 50 C
  - It has signal transmission range of 20 m
  - It is inexpensive
  - It has fast response and it is also durable

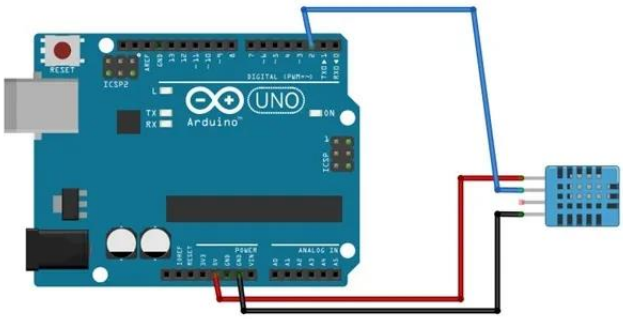
- **DHT11 Pin out**



  - The first pin of the DHT11 is vcc pin.
  - The second pin of the DHT is Data pin.
  - The third pin is not used.
  - The fourth pin of the DHT sensor is ground.

- **DHT11 interfacing with arduino**

First of all connect the ground and the VCC of the DHT11 temperature and humidity sensor to the ground and 5v of the **Arduino**. Then connect the data pin of the DHT11 sensor to the pin 2 of the Arduino.



- **Installing the DHT11 Library**

  To run the following code in Arduino IDE you will first have to install the DHT library in you Arduino directory.

  Download the zip file from github and place it in your Arduino library folder.

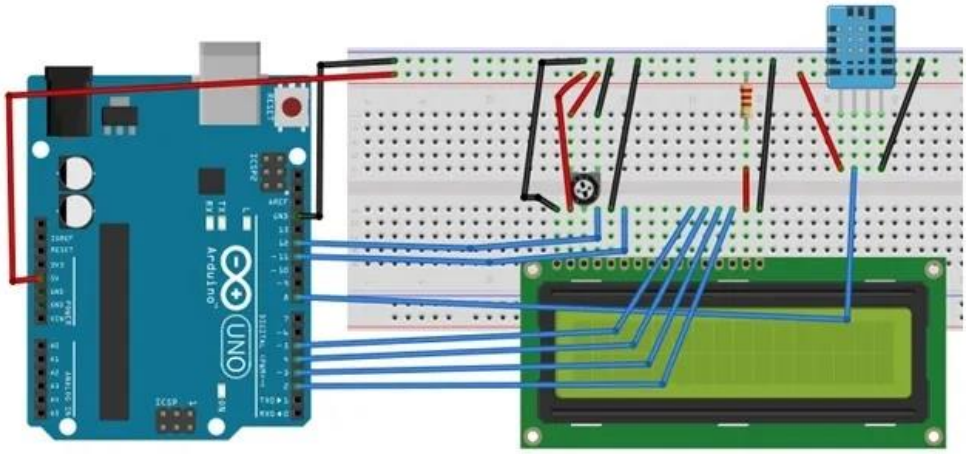
- **Code of DHT11 interfacing with Arduino**

```
// Code for DHT11 Temperature and humidity sensor.
 #include " DHT.h "
 #define DHTPIN 2
 #define DHTTYPE DHT11          // Selecting the type of DHT sensors
 DHT dht ( DHTPIN, DHTTYPE ) ;
 void setup ( ) {
   Serial.begin ( 9600 ) ;
   dht.begin (  ) ;    // The sensor will start working
 }
 void loop ( ) {
   float humidity = dht.readHumidity ( ) ;
  float temp = dht.readTemperature ( ) ;
     if ( isnan ( t ) || isnan ( h ) ) {
      Serial.println ( " Sensor not working " ) ;
   }
 else
{

      Serial.print ( " Temp is " ) ;
      Serial.print ( temp )
      Serial.println ( " *C " ) ;     // Printing " *C "  on display.
      Serial.print ( " Humidity in % is : " ) ;
      Serial.print ( humidity ) ;     // Printing the humidity on display
      Serial.print ( " % \t " ) ;     // Printing "%" on display

   }
 }
```

- **Weather Station using DHT11 and arduino**

In this example we will make a weather station that will sense the humidity and temperature and will show

it on the lcd attached to the Arduino. Make the circuit as shown in the diagram. The resistor in the circuit will make the black light darker. We have used the 220 ohm resistor but you can use any resistor having value near to that. The potentiometer we used in the circuit is used to set the screen contrast. We have used the 10 K ohm value but you can choose any value relative to that one.



- **Components Required**
  - Arduino Uno (you can use any)
  - 16 x 2 LCD
  - DHT11 Temperature and humidity sensor
  - 10 K ohm potentiometer
  - 220 ohm resistor

- **Code of weather station using arduino and DHT11**

```
// This code is for the weather station using the DHT11 humidity and temperature
sensor.
// Install the library of the DHT before uploading the code in the Arduino IDE
#include < dht.h >
#include < LiquidCrystal.h >
LiquidCrystal lcd (12, 11, 5, 4, 3, 2 ) ;
dht DHT ;
#define DHT11_PIN 8
void setup () {
  lcd.begin ( 16, 2 ) ;
}
void loop ()
{
  int chk = DHT.read11(DHT11_PIN ) ;
  lcd.setCursor ( 0, 0 ) ;
  lcd.print ( " Temperature is : " ) ;
  lcd.print ( DHT.temperature ) ;
  lcd.print ( ( char ) 223 ) ;
  lcd.print ( " C " ) ;
  lcd.setCursor ( 0 , 1 );
  lcd.print ( " Humidity is : " ) ;
  lcd.print ( DHT.humidity ) ;
  lcd.print ( " % " ) ;
  delay ( 1000 ) ;

}
```

Temperature Scales

Thermometers measure temperature according to well-defined scales of measurement. The three most common temperature scales are the Fahrenheit, Celsius, and Kelvin scales.

- **Celsius Scale & Fahrenheit Scale**

The Celsius scale has a freezing point of water as 0ºC and the boiling point of water as 100ºC. On the Fahrenheit scale, the freezing point of water is at 32ºF and the boiling point is at 212ºF. The temperature difference of one degree Celsius is greater than a temperature difference of one degree Fahrenheit. One degree on the Celsius scale is 1.8 times larger than one degree on the Fahrenheit scale 180/100=9/5.

- **Kelvin Scale**

Kelvin scale is the most commonly used temperature scale in science. It is an absolute temperature scale defined to have 0 K at the lowest possible temperature, called absolute zero. The freezing and boiling points of water on this scale are 273.15 K and 373.15 K, respectively. Unlike other temperature scales, the Kelvin scale is an absolute scale. It is extensively used in scientific work. The Kelvin temperature scale possesses a true zero with no negative temperatures. It is the lowest temperature theoretically achievable and is the temperature at which the particles in a perfect crystal would become motionless.

- **Relationship Between Different Temperature Scales**

The relationship between three temperature scales is given in the table below:

Relationship between different Temperature Scales

Conversion	Equation
Celsius to Fahrenheit	$T_{F^o} = \frac{9}{5}T_{c^o} + 32$
Fahrenheit to Celsius	$T_{C^o} = \frac{5}{9}T_{F^o} - 32$
Celsius to Kelvin	$T_K = T_{C^o} + 273.15$
Kelvin to Celsius	$T_{C^o} = T_K - 273.15$
Fahrenheit to Kelvin	$T_K = \frac{5}{9}(T(F^0) - 32) + 273.15$
Kelvin to Fahrenheit	$T_{F^0} = \frac{9}{5}(T(K) - 273.15) + 32$

Conclusion: -

_____

_____

_____

_____

_____

Assignment Questions:

Q1. What is the DHT11 temperature sensor and how does it work?

Q2. Explain the process of connecting it to a microcontroller and reading temperature and humidity data from it.

Q3. What are some common applications of the DHT11 sensor in real-world scenarios?

Q4. Write a program in Arduino to convert temperature from Celsius to Fahrenheit. The user should be able to input the temperature in Celsius and the program should output the corresponding temperature in Fahrenheit. Additionally, explain the formula used for conversion and the difference between the Celsius and Fahrenheit temperature scales.

EXPERIMENT NO. 12 (Group B)

- **Aim:** Understanding the connectivity of Raspberry Pi with LED Blinking
- **Outcome:** understanding GPIO, LED, BCM Mode, Board Mode
- **Hardware Requirement:** Raspberry Pi, LED etc
- **Software Requirement**: Python IDE
- **Theory:**

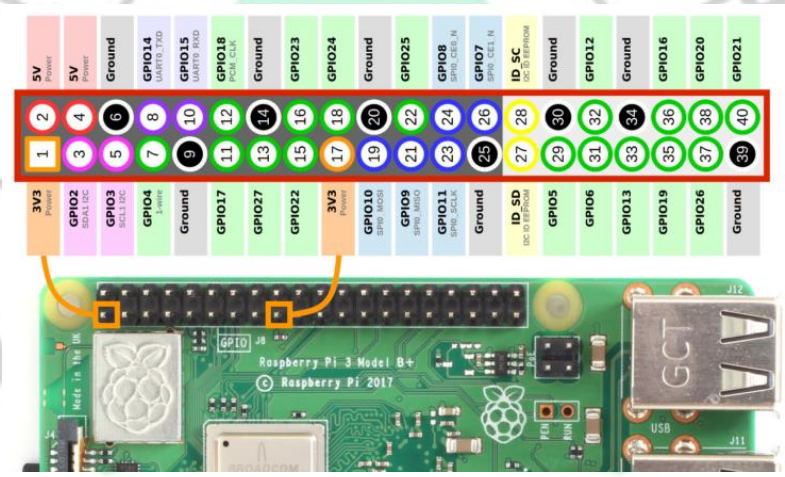
Understanding GPIO pins on Raspberry Pi board and its use in program

Introduction:

- Raspberry Pi 3 Model B is the latest version of raspberry pi board.
- It is released on 29 February.

The above figure shows the Raspberry Pi 3 Model B and It's GPIO pins

General-purpose input/output (GPIO) is a generic pin on an integrated circuit or computer board whose behavior—including whether it is an input or output pin—is controllable by the user at run time.



There are 40 pins available on board of Raspberry pi 3 model B.
The Pins are arranged in a 2×20 fashion as shown in the figure above
Out of these, 26 pins are GPIO pins
As you can observe, the numbers to the pins are given in zigzag manner.

The first (bottom) row starts with number '1'. So the pins in this row have odd numbers i.e. from 1 to 39.

The 2nd (Top) row starts with number '2'. So the pins in this row have even numbers i.e. from 2 to 40.

Out of 40 pins,

a. 26 pins are GPIO pins,
b. 8 pins are Ground (GND) pins,
c. 2 pins are 5V power supply pins
d. 2 pins are 3.3V power supply pins
e. 2 pins are not used

Now if you're coming to the Raspberry Pi as an Arduino user, you're probably used to referencing pins with a single, unique number.

In Raspberry Pi there are two different numbering schemes for referencing Pi pin numbers:

1. Broadcom chip-specific pin numbers (BCM)
2. Physical pin numbers (BOARD)

You're free to use either number-system.

The programs require that you declare which scheme you're using at the very beginning of your program.

In a program, at a time, you can use **only one** number scheme.

Broadcom chip-specific pin numbers (BCM)

BCM - Broadcom pin number, commonly called "GPIO", these are the ones you probably want to use with RPi.GPIO

The parameter used for this system is (GPIO.BCM).

This is a lower level way of working - it refers to the channel numbers on the Broadcom SOC.

To use this system, you have to always work with a diagram describing which channel number goes to which pin on the RPi board.

Your script could break between revisions of Raspberry Pi boards.

In this system

a. 26 GPIO pins are named as GPIO 01 to GPIO 26

Physical Numbering System (BOARD)

This system uses physical - Numbers corresponding to the pin's physical location on the header

The numbers printed on the board are physical numbering system.

The parameter used for this system is (GPIO.BOARD).

The advantage of using this numbering system is that your hardware will always work, regardless of the board revision of the RPi.

You will not need to rewire your connector or change your code.

In this system

a. 26 GPIO pins are named between 0 to 40

The below table summarizes the pinout of Raspberry-Pi in both the number systems.

Sr No.	Pins		BOARD Pin No	BCM Pin No
1	3.3V		1, 17	1, 17
2	5V		2, 4	2, 4
3	Ground		6,9,14,20,25,30,34,39	6,9,14,20,25,30,34,39
4	UART	TXD	8	GPIO 14
		RXD	10	GPIO 15
	I2C1	SDA1	3	GPIO 2

5		SCL1	5	GPIO 3
	I2C0	SDA0	27	ID_SD
		SCL0	28	ID_SC
6	SPI0	MOSI 0	19	GPIO 10
		MISO 0	21	GPIO 9
		SCLK 0	23	GPIO 11
		CE 0	24	GPIO 8
	SPI1	MOSI 1	38	GPIO 20
		MISO 1	35	GPIO 19
		SCLK 1	40	GPIO 21
		CE 1	26	GPIO 7

The Python IDLE shell and command line

To use the Python IDLE IDE for programming in Raspberry-Pi use the following

Open **Python 3** from the main menu:

Or open terminal window and type the command *sudo idle 3.5* and press enter
Install all libraries required for Buzzer as given above.
Write the program as per algorithm given below
Save with **Ctrl + S** and run with **F5**.
See output on Python Shell or Terminal Window.

Raspberry Pi GPIO programming using Python

The Raspberry Pi is often used in conjunction with other hardware to create interesting electronic projects.
The Pi 3 comes with 40 GPIO pins that you can use to interface with various hardware devices—for both receiving data from them or for writing data to them.
To do this, we have to program the GPIO pins. To do this, special libraries in Python are used. To include these libraries in the program, the command used is 'import'
This way, we can write applications to both read and also to control devices, i.e., turn them on and off, etc.
The default operating system used in Raspberry-Pi is Raspbian.
The Python package used for Raspberry Pi GPIO programming is RPi.GPIO. It is already installed in

Raspbian.
If you are using any other operating system, the package can be installed by using the following command:

$ sudo pip install RPi.GPIO

There are **important 8 steps** in the programming of Raspberry-Pi using Python as follows

1. Import the RPi.GPIO library using the following command import RPi.GPIO as GPIO
2. Import the Time library using the following command

import time

3. Set numbering scheme to be used. The method used for this is GPIO.setmode(). We will use physical number scheme. So the method is written as

GPIO.setmode(GPIO.BOAD)

4. Set the pin mode as INPUT or OUTPUT using the commands GPIO.setup(channel, GPIO.IN) GPIO.setup(channel, GPIO.OUT)
5. Read input using following command

   GPIO.input(pin no)
6. Write output using following comman

   GPIO.output(pin no, state)
7. Give delay using command using following command

   time.sleep(1) # delay for 1 second
8. Clean up GPIO and exit using following commands

   GPIO.cleanup()

   print("Exiting...")

You must clean up the pin set-ups before your program exits otherwise those pin settings will persist, and that might cause trouble when you use the same pins in another program.
The Pi 'expresses its displeasure' with a warning.
To clean up the entire set of pins, invoke GPIO.cleanup().
If you want only a few pins to be cleaned up, then the pin numbers should be provided as GPIO.cleanup (channel_list).
Anyway, you can suppress the warning messages by calling GPIO.setwarnings (False).
Save the program with proper name. The file is saved with extension '.py'.
The IDE named 'IDLE' used for programming is an interpreter and not a compiler. So to run the python program, we need to give the super user permission as follows.

Studying Connectivity and Configuration of Raspberry Pi board with basic peripherals (LEDs)

Aim/Objectives:

To understand the concept of Led bar
To understand the common anode & common cathode configuration.
To interface LED bar with Raspberry Pi.
Generate various patterns on LED bar.

Software:

Raspbian OS
IDLE editor

Hardware Modules:

Raspberry Pi Board
LED bar
Monitor

Theory:

Introduction to "LED"

LED is a Light Emitting Diode.
Light emitting diode is a two lead semiconductor light source. It is a p-n junction diode, which emits light when it is activated.
When a suitable voltage is applied to the leads, electrons are able to recombine with electron holes within the device, and the color of light (corresponding to the energy of photon) is determined by the energy band gap of the semiconductor.
It has two terminals named as 'anode (+ve)' and 'cathode (-ve)'.

Battery is connected to these two terminals.
When LED is forward biased, it emits light.
In LED bar number of LEDs are connected in series (in our case 8 LEDs are connected)
LED bar has two configurations as Common Anode: In this, anode terminal of all the LEDs are made common and connected to the VCC (+5v). By controlling cathode terminal we can make LED ON or OFF (current sourcing).
Common Cathode: In this, cathode terminal of all the LEDs are made common and connected to the Ground (0v). By controlling anode terminal we can make LED ON or OFF (current sinking).

Safety precautions:

- Raspberry-Pi provides 3.3V and 5V VCC pins
- Raspberry-Pi operates on 3.3V.
- Various sensors and actuators operate on different voltages.
- Read datasheet of a given sensor or an actuator and then use appropriate VCC pin to connect a sensor or an actuator.
- Ensure that signal voltage coming to the Raspberry-Pi from any sensor or actuator does not exceed 3.3V.
- If signal/data coming to Raspberry-Pi is greater than 3.3V then use voltage level shifter module to decrease the incoming voltage.
- The Raspberry-Pi is a costly device, hence you should show the circuit connections to your instructor before starting your experiment.

Interface diagram:

Steps for assembling circuit:

Connect led bar module pins from D0- D3 to Raspberry Pi GPIO pins 7, 11, 13, 15 respectively.
Connect led bar module pin COM to the GND pin of Raspberry-Pi module.

Procedure:

Write the program as per the algorithm given below.
Save program.
Run code using Run module.

Algorithm:

Import GPIO and Time library
Set mode i.e. GPIO.BOARD
Set GPIO 8 pins as a Output pin
Print message “ON”
After 1 second time delay, Make all the leds ON one by one
Print message “OFF”
After 1 second time delay, Make all the leds OFF one by one

Observation:

See output on Command prompt or Python shell.

Assignment Questions:

Q1. What is an LED, and how does it work?

Q2. How can you connect an LED to a Raspberry Pi?

Q3. Are there any limitations on the number of LEDs you can connect to a Raspberry Pi?

Q4. Can you control multiple LEDs independently using a Raspberry Pi?

Q5. What are some common projects or applications involving LEDs and Raspberry Pi?

EXPERIMENT NO. 14/15/16/17/18 (Group C)

- **Aim:**
- **Outcome:**
- **Hardware Requirement:**
- **Software Requirement**:
- **Theory:**
- **Conclusion: -**