REVA UNIVERSITY
Bengaluru, India

School of Computer Science and Applications

LAB MANUAL

Academic Year-2025-26

Programme: B.SC- CS

Semester: V Sem

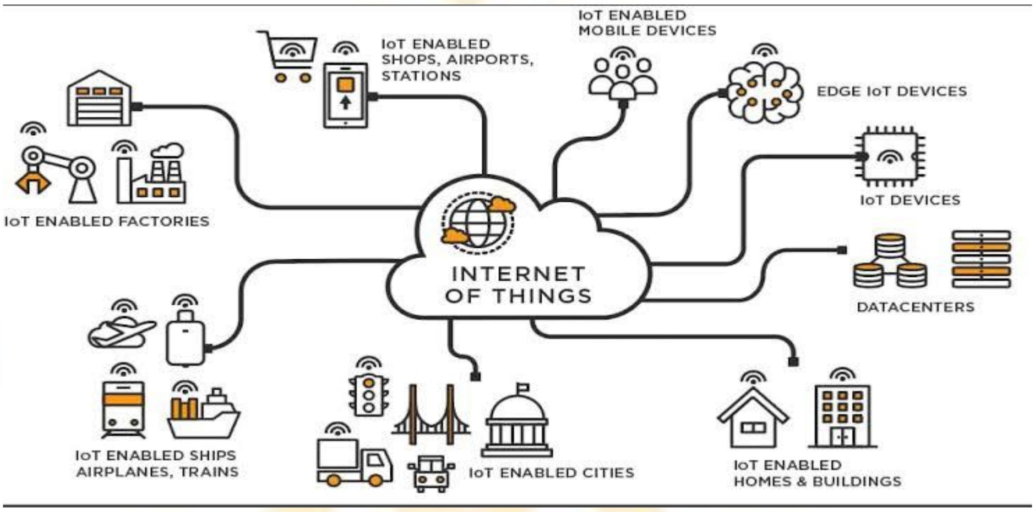
Course Title: IoT Security Lab

Course Code: B23DC0506

INDEX

Introduction to IoT:

The **Internet of things** (**IoT**) describes physical objects (or groups of such objects) with sensors, processing ability, software, and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks. Internet of things has been considered a misnomer because devices do not need to be connected to the public internet, they only need to be connected to a network and be individually addressable.


IoT ENABLED MOBILE DEVICES
IoT ENABLED SHOPS, AIRPORTS, STATIONS
EDGE IoT DEVICES
IoT DEVICES
IoT ENABLED FACTORIES
INTERNET OF THINGS
DATACENTERS
IoT ENABLED SHIPS AIRPLANES, TRAINS
IoT ENABLED CITIES
IoT ENABLED HOMES & BUILDINGS

Introduction to Arduino:

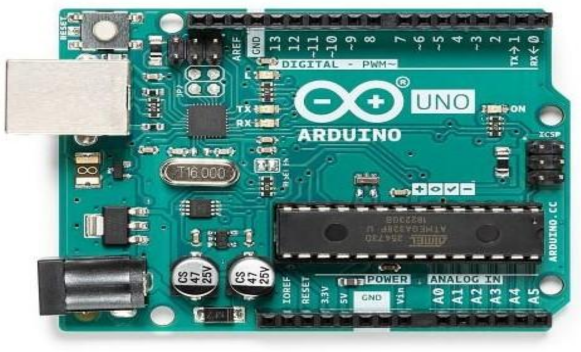


Figure 1: **Arduino Uno** **Figure 2: Type B USB**

The **Arduino Uno** is an open-source microcontroller board based on the Microchip

ATmega328P microcontroller and developed by Arduino.cc The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The board has 14 digital I/O pins (six capable of PWM output), 6 analog I/O pins, and is programmable with the Arduino IDE (Integrated Development Environment), via a type B USB cable.

It can be powered by the USB cable or by an external 9-volt battery, though it accepts voltages between 7 and 20 volts.

TECHNICAL SPECIFICATIONS:

Sl.No.	**NAME**	**DESCRIPTION / RANGE**
1	Microcontroller	Microchip ATmega328P
2	Operating Voltage	5 Volts
3	Input Voltage	7 to 20 Volts
4	Digital I/O Pins	14 (of which 6 can provide PWM output)
5	UART	**1**
6	I2C	**1**
7	SPPI	**1**
8	Analog Input Pins	6
9	DC Current per I/O Pin	20 mA
10	DC Current for 3.3V Pin	50 mA
11	Flash Memory	32 KB of which 0.5 KB used by bootloader
12	SRAM	2 KB
13	EEPROM	1 KB
14	Clock Speed	16 MHz
15	Length	68.6 mm
16	Width	53.4 mm
17	Weight	25 g

General pin functions:

- **LED**: There is a built-in LED driven by digital pin 13. When the pin is high value, theLED is on, when the pin is low, it is off.
- **VIN**: The input voltage to the Arduino/Genuino board when it is using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). Youcan supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V**: This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 20V), the USB connector (5V), orthe VIN pin of the board (7-20V). Supplying voltage via the 5V or 3.3V pins bypasses theregulator, and can damage the board.
- **V3**: A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50mA.
- **GND**: Ground pins.
- **IOREF**: This pin on the Arduino/Genuino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source, or enable voltage translators on the outputsto work with the 5V or 3.3V.
- **Reset**: Typically used to add a reset button to shields that block the one on the board.

Special pin functions:

- **Serial / UART:** pins 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL serial chip.
- External **interrupts**: pins 2 and 3. These pins can be configured to trigger an interrupt on alow value, a rising or falling edge, or a change in value.
- **PWM** (pulse-width modulation): pins 3, 5, 6, 9, 10, and 11. Can provide 8-bit PWMoutput with the analogWrite() function.
- **SPI** (Serial Peripheral Interface): pins 10 (SS), 11 (MOSI), 12 (MISO), and 13 (SCK).

These pins support SPI communication using the SPI library.

- **TWI** (two-wire interface) / I²C: pin SDA (A4) and pin SCL (A5). Support TWIcommunication using the Wire library.
- **AREF** (analog reference): Reference voltage for the analog inputs.

Working with Arduino IDE (Integrated Development Environment)

An **integrated development environment** (**IDE**) is a software application that providescomprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools and a debugger.

Link to download and install arduino IDE software: https://www.arduino.cc/en/Main/Software

Arduino IDE

The Arduino is a fantastic single-board microcontroller solution for many DIY projects, and, we will look at the Integrated Development Environment, or IDE, that is used to program it.

Download the IDE

First, you must download the IDE and install it. Start by isiting Arduino’s software page (https://www.arduino.cc/en/Main/software). The IDE is available for most common operating systems, including Windows, Mac OS X, and Linux, so be sure to download the correct version for your OS. If you are using Windows 7 or older, do not download the Windows app version, as this requires Windows 8.1 or Windows 10. Once the installer has downloaded, go ahead and install the IDE. Chances are you will want to enable all options on the installer, including any USB drivers and libraries, but do make sure to readthe EULA!

The Arduino IDE

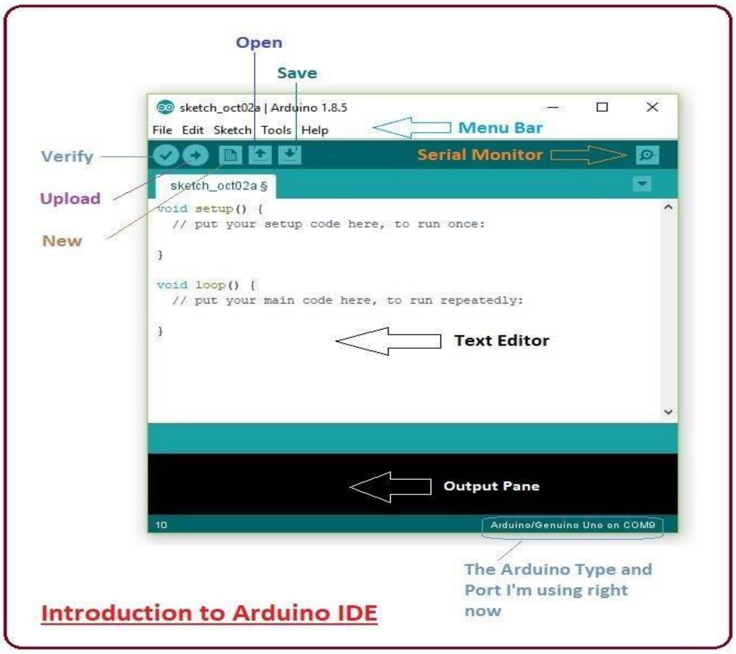
The Arduino IDE is incredibly minimalistic, yet it provides a near-complete environment for most Arduino-based projects. The top menu bar has the standard options, including “File” (new, load save, etc.), “Edit” (font, copy, paste, etc.), “Sketch” (for compiling and programming), “Tools” (useful options for testing projects), and “Help”. The middle section of the IDE is a simple text editor that where you can enter the program code. The bottom section of the IDE is dedicated to an output window that is used to see the status
of the compilation, how much memory has been used, any errors that were found in the program, and various other useful messages.

The IDE environment is mainly distributed into three sections

- ➢ Menu Bar
- ➢ Text Editor

- Output Pane

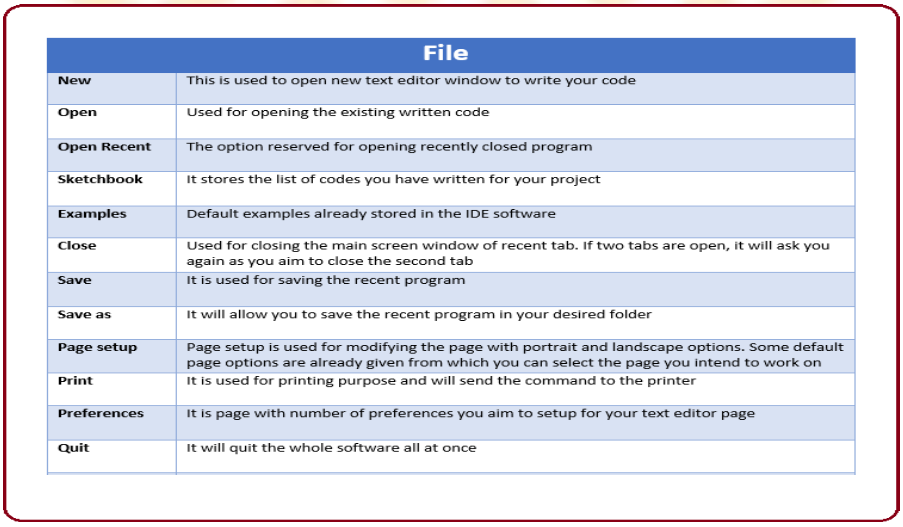
As you download and open the IDE software, it will appear like an image below.


Open
Save
sketch_oct02a | Arduino 1.8.5
File Edit Sketch Tools Help
Menu Bar
Verify
Upload
New
Serial Monitor
sketch_oct02a §
Text Editor
Output Pane
10
Arduino/Genuino Uno on COM9
The Arduino Type and Port I'm using right now

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Introduction to Arduino IDE

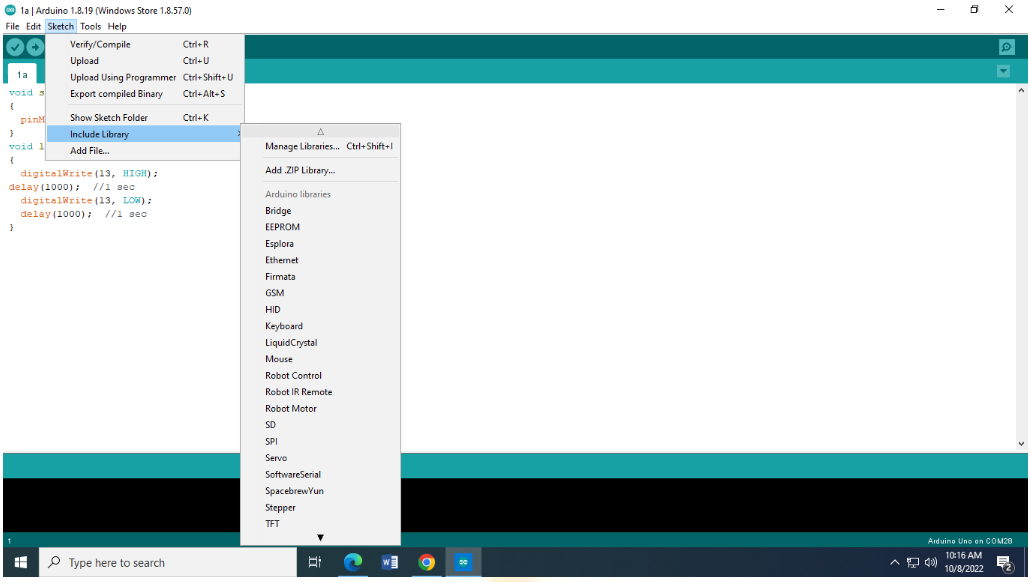
File - You can open a new window for writing the code or open an existing one. Followingtable shows the number of further subdivisions the file option is categorized into



File	
New	This is used to open new text editor window to write your code
Open	Used for opening the existing written code
Open Recent	The option reserved for opening recently closed program
Sketchbook	It stores the list of codes you have written for your project
Examples	Default examples already stored in the IDE software
Close	Used for closing the main screen window of recent tab. If two tabs are open, it will ask you again as you aim to close the second tab
Save	It is used for saving the recent program
Save as	It will allow you to save the recent program in your desired folder
Page setup	Page setup is used for modifying the page with portrait and landscape options. Some default page options are already given from which you can select the page you intend to work on
Print	It is used for printing purpose and will send the command to the printer
Preferences	It is page with number of preferences you aim to setup for your text editor page
Quit	It will quit the whole software all at once

Libraries

Libraries are very useful for adding the extra functionality into the Arduino Module. Thereis a list of libraries you can add by clicking the Sketch button in the menu bar and going to Include Library.



```
void s
{
  pinM
}
void l
{
  digitalWrite(13, HIGH);
delay(1000);  //1 sec
  digitalWrite(13, LOW);
  delay(1000);  //1 sec
}
```

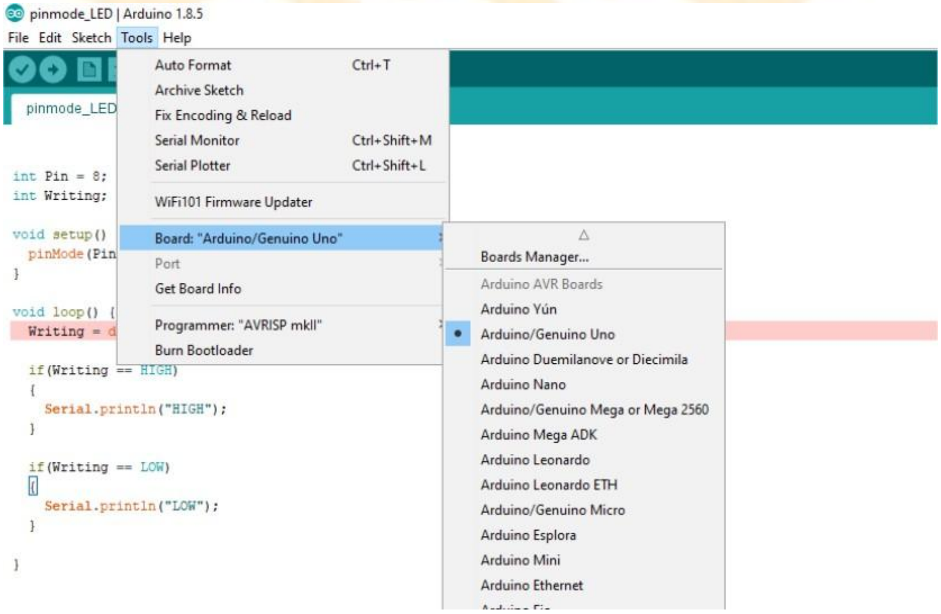
As you click the Include Library and Add the respective library it will on the top of the sketch with a #include sign. Suppose, I Include the EEPROM library, it will appear on thetext editor as

#include <EEPROM.h>.

Most of the libraries are preinstalled and come with the Arduino software. However, youcan also download them from the external sources.

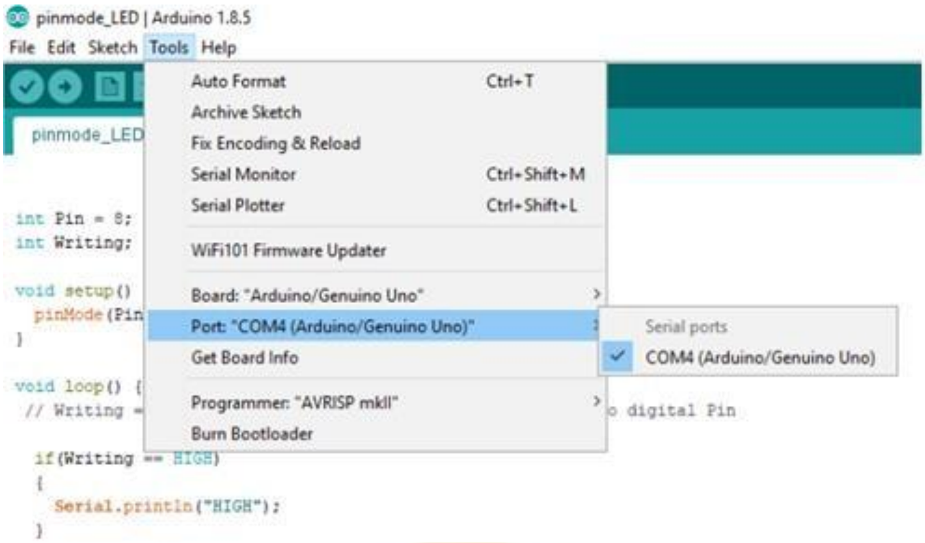
How to select the board

In order to upload the sketch, you need to select the relevant board you are using and the ports for that operating system. As you click the Tools on the Menu, it will open like the figure below:



```
int Pin = 8;
int Writing;

void setup()
  pinMode(Pin
}

void loop() {
  Writing = d

  if(Writing == HIGH)
  {
    Serial.println("HIGH");
  }

  if(Writing == LOW)
  {
    Serial.println("LOW");
  }

}
```

Just go to the "Board" section and select the board you aim to work on. Similarly, COM1, COM2, COM4, COM5, COM7 or higher are reserved for the serial and USB board. You can look for the USB serial device in the ports section of the Windows Device Manager. Following figure shows the COM4 that I have used for my project, indicating the Arduino Uno with COM4 port at the right bottom corner of the screen.


pinmode_LED | Arduino 1.8.5
File Edit Sketch Tools Help
pinmode_LED
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino/Genuino Uno"
Port: "COM4 (Arduino/Genuino Uno)"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
✓ COM4 (Arduino/Genuino Uno)

```
int Pin = 8;
int Writing;

void setup()
  pinMode(Pin
}

void loop() {
 // Writing =                              o digital Pin

  if(Writing == HIGH)
  {
    Serial.println("HIGH");
  }
```

A1

Problem Statement:

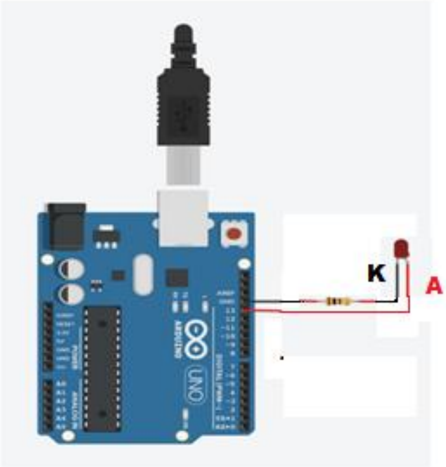
AIM: Write a program to blink an LED.

a) Infinite number of times with ON & OFF duration of 1 sec.

b) Only three times with ON and OFF duration 2 sec.

Components and Software Required

Sl.No.	COMPONENTS	QUANTITY
01	Arduino UNO	01
02	USB cable	01
03	LED	01
04	560Ω Resistor	01
05	Connecting wires	03



Connection:

LED 3 PINS
A node – 5V
K node - GND
Output: 13

Program

a) Infinite number of times with ON & OFF duration of 1 sec.

Program:

```
void setup()
{
  pinMode(13, OUTPUT);
}
void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);    //1 sec
  digitalWrite(13, LOW);
  delay(1000);    //1 sec
}
```

b) Only three times with ON and OFF duration 2 sec.

Program:

```
#define ledPin 13
void setup()
{
  pinMode(ledPin , OUTPUT);

  digitalWrite(ledPin, HIGH);
  delay(2000);    //2 sec
  digitalWrite(ledPin, LOW);
  delay(2000);    //2 sec

  digitalWrite(ledPin, HIGH);
  delay(2000);    //2 sec
  digitalWrite(ledPin, LOW);
  delay(2000);    //2 sec

  digitalWrite(ledPin, HIGH);
  delay(2000);    //2 sec
  digitalWrite(ledPin, LOW);
  delay(2000);    //2 sec

}
void loop()
{
}
```

Output:

Results:
The Code to turn ON/OFF the LED has been written and executed successfully using Arduino UNO

A2

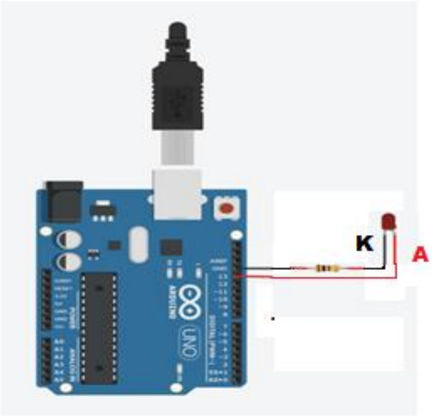
Problem Statement:

Write a program to increase and decrease the brightness of the LED.
Write a program to control the brightness of LED using a Potentiometer.

Components and Software Required

(a)

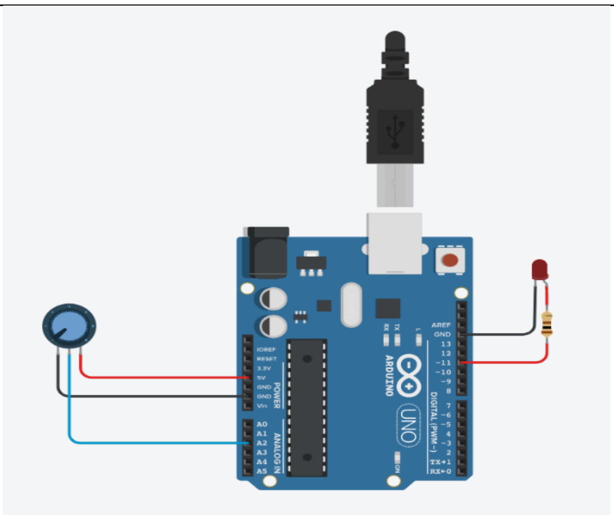
Sl.No.	COMPONENTS	QUANTITY
01	Arduino UNO	01
02	USB cable	01
03	LED	01
04	560Ω Resistor	01
05	Connecting wires	03



LED 3 PINS
A node PIN – 5V
K node PIN- GND
Out PIN : 11

(b)

Sl.No.	COMPONENTS	QUANTITY
01	Arduino UNO	01
02	USB cable	01
03	LED	01
04	560Ω Resistor	01
05	10KΩ Potentiometer	01
06	Bread Board	01
07	Connecting wires	03



Connection
LED 3 PINS – Arduino UNO
A node PIN – 5V
K node PIN- GND
Out PIN : 11

Potentiometer - Arduino UNO
GND – GND
VCC- 5V
Out – A2

Program

Program to increase the brightness of LED:

```
int ledPin = 11;
void setup()
{
pinMode(ledPin, OUTPUT);
Serial.begin(9600);
}
void loop()
{
for(int i=0; i<255; i=i+50)
{
analogWrite(ledPin, i);
delay (500);
}
}
```

Program to decrease the brightness of LED:

```
int ledPin = 11;
void setup()
{
pinMode(ledPin, OUTPUT);
Serial.begin(9600);
}
void loop()
```

```
{
for(int i=255; i>0; i=i-50)
{
analogWrite(ledPin, i);
delay (500);
}
}
```

Program of potentiometer:

```
int analogPin = A2;           // wiper
int ledPin = 11;
int val = 0;

void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
void loop()
 {
  val = analogRead(analogPin);
  int level = map(val, 0, 1023, 0, 255);
  analogWrite(ledPin, level);

  Serial.print("Pot = ");
  Serial.print(val);
  Serial.print("\t brightness = ");
  Serial.println(level);
  delay(1000);
}
```

Results:

	Output:	Increasing of the LED light Decreasing of the LED Light

Note:

- The brightness of an LED can be controlled by adjusting the amount of power supplied to it. The most common method for this is **Pulse Width Modulation (PWM)**, which rapidly switches the LED **on and off** at a high frequency, adjusting the ratio of **on-time to off-time** to control brightness.
- A potentiometer, sometimes known as a “pot,” is an electrical component used to manually alter the resistance in a circuit. It has three terminals and is a form of variable resistor. Potentiometer has many applications as it is used to control the volume of audio devices and change the brightness of screens or LEDs. We can regulate a parameter by varying the resistance over a wide range.

A3

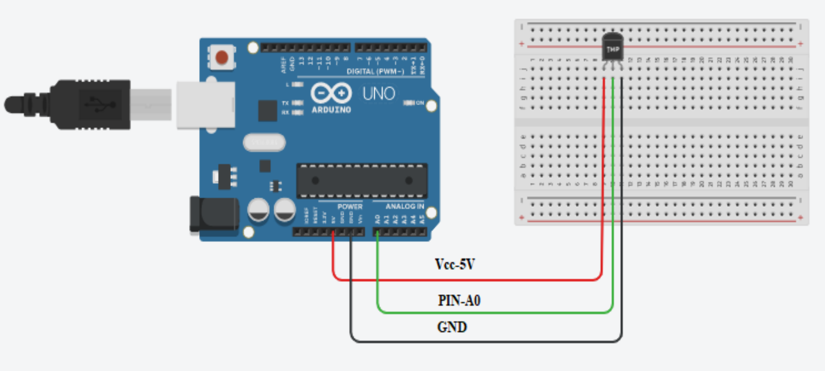
Problem Statement:

a) Write a program to interface temperature sensor and display the values on the serial monitor.
b) Write a program to display the range of temperature on LCD.

Components and Software Required

Components for (a)

Sl.No.	COMPONENTS	QUANTITY
01	Arduino UNO	01
02	USB cable	01
03	Temperature Sensor (LM35)	01
04	Jumper wires	03

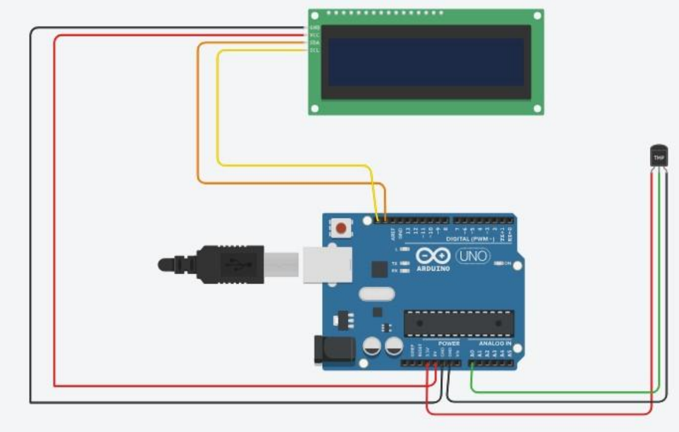


1. Connect temperature sensor LM35 Out pin to A0 of Arduino UNO.
2. Connect temperature sensor LM35 Vcc pin to 3.3V of Arduino UNO.
3. Connect temperature sensor LM35 Gnd pin to Gnd of Arduino UNO.

Components for (b)

Sl.No.	COMPONENTS	QUANTITY
01	Arduino UNO	01
02	USB cable	01
03	Temperature Sensor (LM35)	01
05	LCD I2C	01
06	Bread Board	01
07	Connecting / Jumper wires	07

Circuit Diagram:



Procedure:

1. Connect temperature sensor LM35 Out pin to A0 of Arduino UNO.
2. Connect temperature sensor LM35 Vcc pin to 3.3V of Arduino UNO.
3. Connect temperature sensor LM35 Gnd pin to Gnd of Arduino UNO.
4. Connect pin of LCD_I2C Gnd to Gnd of Arduino UNO.
5. Connect pin of LCD_I2C Vcc to 5V of Arduino UNO.
6. Connect pin of LCD_I2C SDA to SDA of Arduino UNO.
7. Connect pin of LCD_I2C SCL to SCL of Arduino UNO.

Program

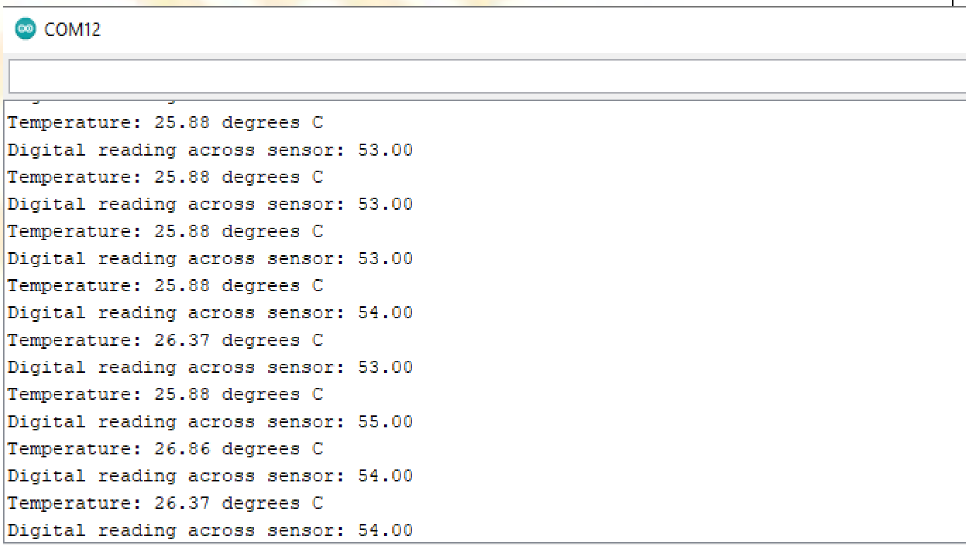
(a)

```
float sensor = 0;                    // Initialize the variables
float voltage = 0;
float celsius = 0;

void setup()
{
  Serial.begin(9600);             // Start serial comm @ baud rate 9600bps
}

void loop()
{
  sensor = analogRead(A0);    // Reading analog data from LM35 (tempsensor)
                                   // convert raw sensor value to millivolts
  voltage = (sensor*5000)/1024;
                                   // convert millivolts to Celsius
  celsius = voltage/10;
  Serial.print("Digital reading across sensor: ");
  Serial.print(sensor);
  Serial.println();
  Serial.print("Temperature: ");
  Serial.print(celsius,2);
  Serial.println(" degrees C");
  delay (1000);
}
```

(b)

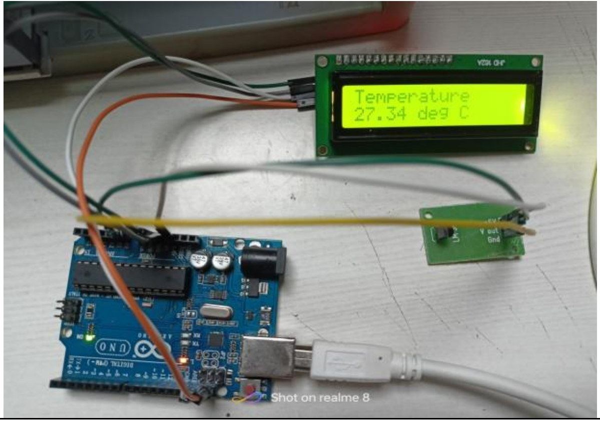
```
#include <LiquidCrystal_I2C.h>         // Initialize Liquid Crystal library
addressing I2C
LiquidCrystal_I2C lcd(0x27,20,4);      // I2C Addressing of the display device
float sensor = 0;                          *Most commonly used addresses
0x27 0x3F
float voltage = 0;                         *20 Character in 4line=80 character
float celsius = 0;
void setup()
{
  Serial.begin(9600);                      // Start serial comm @ baud rate
9600bps
  lcd.init();                              // LCD Initiation
  lcd.clear();                            // Initially clearing content in display
buffer
  lcd.backlight();                        // backlight enabled
  lcd.begin(16, 2);                       // 16 Character in 2 lines display=32
character
}
void loop()
{
  sensor = analogRead(A0);
  // convert raw sensor value to millivolts
  voltage = (sensor*5000)/1024;
   // convert millivolts to Celsius
  celsius = voltage/10;
  Serial.print("Temperature: ");
  Serial.print(celsius,2);
  Serial.println(" degrees C");
  lcd.setCursor(0, 0);                    // Set cursor (column, Row)
  lcd.print("Temperature");               // print text where cursor is set
  lcd.setCursor(0, 1);
  lcd.print(celsius,2);
  lcd.print(" deg C");
  delay (1000);
  lcd.clear();
}
```

Output:

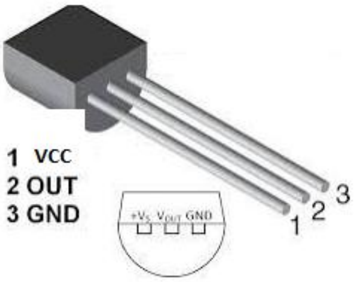


```
Temperature: 25.88 degrees C
Digital reading across sensor: 53.00
Temperature: 25.88 degrees C
Digital reading across sensor: 53.00
Temperature: 25.88 degrees C
Digital reading across sensor: 53.00
Temperature: 25.88 degrees C
Digital reading across sensor: 54.00
Temperature: 26.37 degrees C
Digital reading across sensor: 53.00
Temperature: 25.88 degrees C
Digital reading across sensor: 55.00
Temperature: 26.86 degrees C
Digital reading across sensor: 54.00
Temperature: 26.37 degrees C
Digital reading across sensor: 54.00
```

(b)



Note:



LM35

LM35 is **a temperature measuring device having an analog output voltage proportional to the temperature**. It provides output voltage in Centigrade (Celsius). It does not require any external calibration circuitry. The sensitivity of LM35 is 10 mV/degree Celsius.

Calculations:

$$\mathbf{{}^{\circ}C = \frac{Ref.voltage*Digital\ count}{ADC*10mv}}$$

Where,

Reference voltage = 5v
Digital count = Sensor output reading
ADC = 1023 ***∵ In built 8-bit ADC***
10mv = for every 1°C the voltage change is 10mv.

$$\mathbf{{}^{\circ}C = \frac{5*54}{1023*10*10^{-3}} \Rightarrow \frac{5*54*10^{+3}}{1023*10} \Rightarrow \frac{54*5000}{1023*10} \Rightarrow \frac{54*500}{1023} \Rightarrow \mathit{26.39{}^{\circ}C}}$$

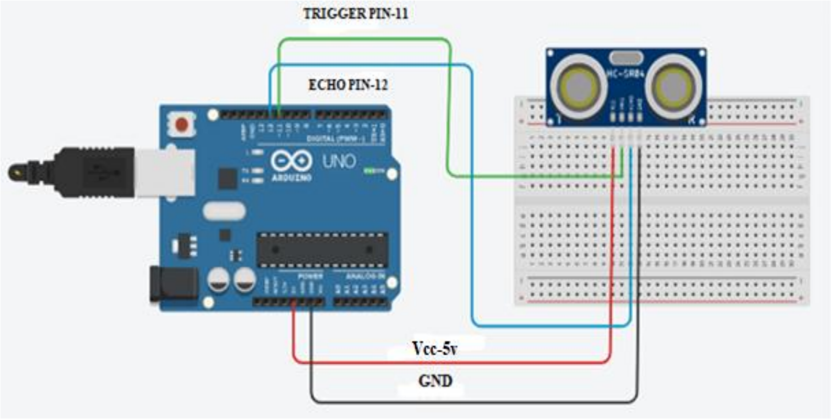
To convert degree Celsius to degree Fahrenheit the formula is:

°F = ***°C *(9/5) +32*** **=** ____*°F*

A4	**Problem Statement:**	**Write a program to interface ultrasonic sensor and display the distance from an object.**

Components and Software Required

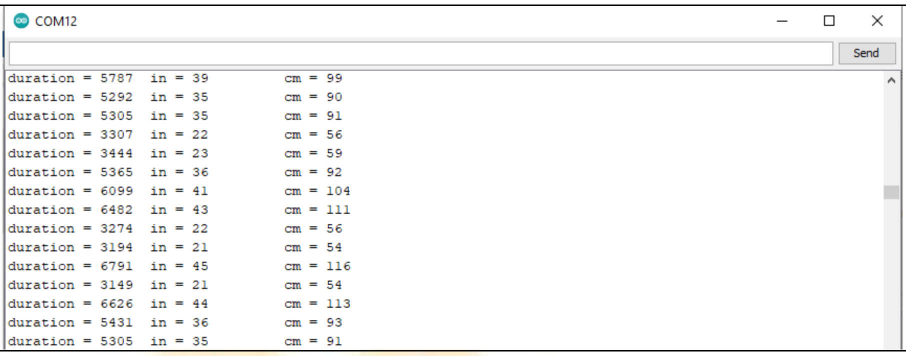
Sl.No.	COMPONENTS	QUANTITY
01	Arduino UNO	01
02	USB cable	01
03	Ultrasonic Sensor (HC-SR04)	01
04	Jumper wires	04



Program

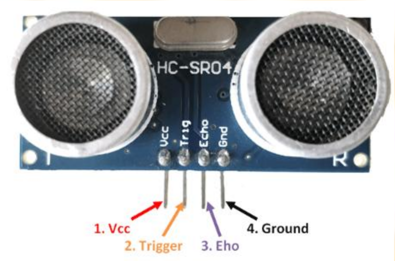
```
#define trigPin 11                        // define the constants & variables
#define echoPin 12
long duration, cm, inches;               // length of sound wave & how far away the
object is present
 //double,
void setup() {
  Serial.begin (9600);
                                              //Define inputs and outputs
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);     // turn on echo pin high for however
long the waves
                                                 Travelling for.
                                              //pulseIn output will be in µs
  cm = (0.0343*duration)/2;              // distance in cm

  inches= cm / 2.54;

  Serial.print("duration = ");
  Serial.print(duration);

  Serial.print("\t in = ");
```

		Serial.print(inches);
		
		Serial.print("\t cm = ");
		Serial.println(cm);
		
		delay(1000);
		}

	Output:	 COM12 Send
		duration = 5787 in = 39 cm = 99
		duration = 5292 in = 35 cm = 90
		duration = 5305 in = 35 cm = 91
		duration = 3307 in = 22 cm = 56
		duration = 3444 in = 23 cm = 59
		duration = 5365 in = 36 cm = 92
		duration = 6099 in = 41 cm = 104
		duration = 6482 in = 43 cm = 111
		duration = 3274 in = 22 cm = 56
		duration = 3194 in = 21 cm = 54
		duration = 6791 in = 45 cm = 116
		duration = 3149 in = 21 cm = 54
		duration = 6626 in = 44 cm = 113
		duration = 5431 in = 36 cm = 93
		duration = 5305 in = 35 cm = 91

More Info:

HC-SR04 Ultrasonic sensor



Pin Description:

- **Vcc** - The Vcc pin powers the sensor, typically with +5V
- **Trigger** - Trigger pin is an Input pin. This pin has to be kept high for 10µs to initialize measurement by sending US wave.
- **Echo** - Echo pin is an Output pin. This pin goes high for a period of time which will be equal to the time taken for the US wave to return back to the sensor.
- **Ground** - This pin is connected to the Ground of the system.

HC-SR04 Ultrasonic sensor Description:

HC-SR04 Ultrasonic (US) sensor is a 4 pin module, whose pin names are Vcc, Trigger, Echo and Ground respectively. This sensor is a very popular sensor used in many applications where measuring distance or sensing objects are required. The module has two eyes like projects in the front which forms the Ultrasonic transmitter and Receiver.

Time taken by pulse is actually for to and fro the sensor, while we need only half of this therefore time is taken as Time/2:

∴ Distance = (Speed * Time) /2

Calculations: -

Distance = (Speed * Time) /2 cm
Distance= (0.034300 * Time) /2 cm

Inches = Centimeter / 2.54

A5	Problem Statement:	Write a program to interface motion sensor and display its status using LED. If motion is detected, turn on LED otherwise keeps the LED off.

Sl.No.	COMPONENTS	QUANTITY
01	Aurdino	01
02	Micro USB cable	01
03	Motion sensor	01
04	LED	01
05	560Ω Resistor	01
06	Connecting/Jumper wires	05

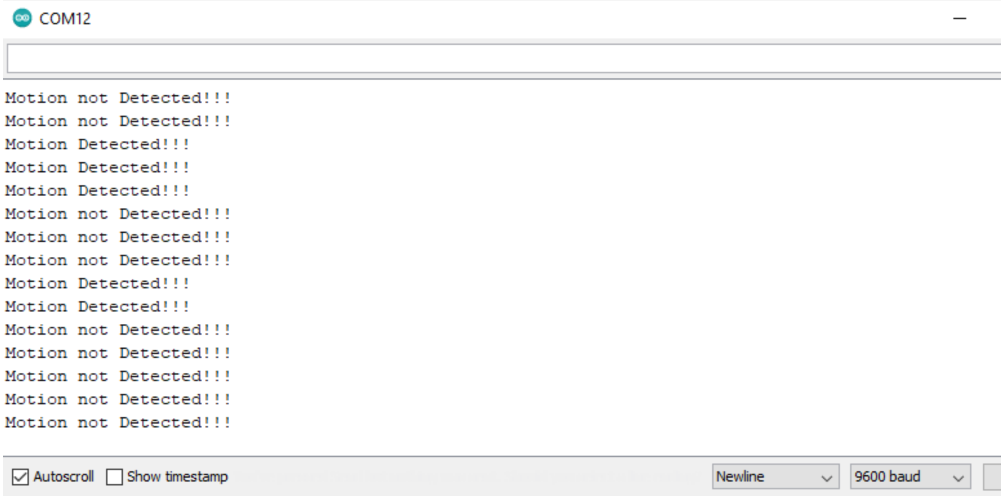
PIR Sensor to arduino
OUT –PIN 5
VCC – 5volts
GND –GND

LED to Arduino
Anode – PIN4
Kathode -GND

Program

```
// -----------------------------
// Motion Detection with LED
// -----------------------------

#define ledPin 4      // LED connected to digital pin 4
#define pirPin 5      // PIR sensor output connected to digital pin 5

int pirStat = 0;      // Variable to store sensor status

void setup() {
  pinMode(ledPin, OUTPUT);   // Set LED pin as output
  pinMode(pirPin, INPUT);    // Set PIR sensor pin as input
  Serial.begin(9600);        // Initialize serial monitor
  Serial.println("PIR Motion Sensor Test Initialized...");
}

void loop() {
  pirStat = digitalRead(pirPin);   // Read sensor value

  if (pirStat == HIGH) {           // If motion detected
    digitalWrite(ledPin, HIGH);    // Turn ON LED
    Serial.println("MOTION DETECTED");
    delay(1000);                   // Delay for stability
  }
  else {                         // If no motion detected
    digitalWrite(ledPin, LOW);     // Turn OFF LED
    Serial.println("MOTION NOT DETECTED");
    delay(1000);                  // Delay for stability
  }
}
```

Output:

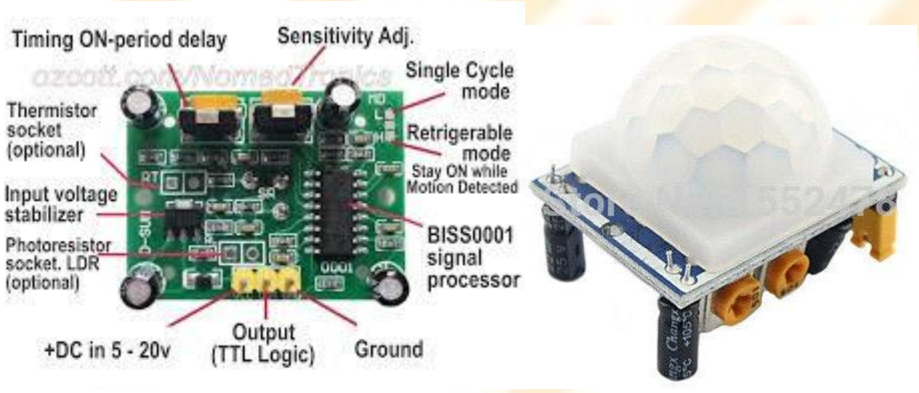


```
Motion not Detected!!!
Motion not Detected!!!
Motion Detected!!!
Motion Detected!!!
Motion Detected!!!
Motion not Detected!!!
Motion not Detected!!!
Motion not Detected!!!
Motion Detected!!!
Motion Detected!!!
Motion not Detected!!!
Motion not Detected!!!
Motion not Detected!!!
Motion not Detected!!!
Motion not Detected!!!
```

☑ Autoscroll □ Show timestamp Newline 9600 baud

Note:

HC-SR501 Passive Infrared (PIR) Motion Sensor

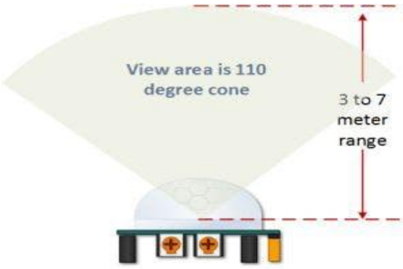


PIR Motion Sensor description:-

The Passive Infrared Sensor (PIR) sensor module is used for motion detection. It is often referred to used "PIR", "Pyroelectric", "Passive Infrared" and "IR Motion" sensor. The module has an on-board pyroelectric sensor, conditioning circuitry and a dome shaped Fresnel lens. It is used to sense movement of people, animals, or other objects.

This motion sensor module uses the LHI778 Passive Infrared Sensor and the BISS0001 IC to control how motion is detected.

The module features adjustable sensitivity that allows for a motion detection range f**rom 3 meters to 7 meters.**

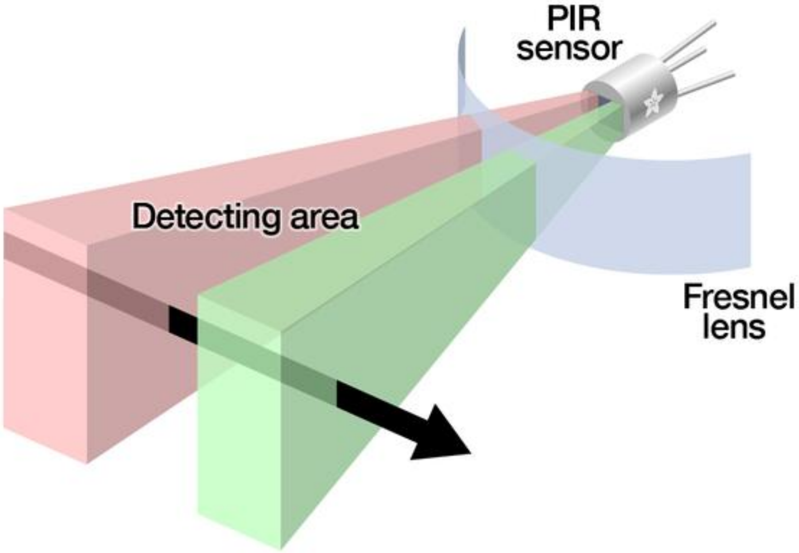


Applications:

They are commonly used in burglar alarms and automatically-activated lighting systems.

PIR Motion sensor working principle:-

The PIR sensor itself has two slots in it, each slot is made of a special material that is sensitive to IR. The lens used here is not really doing much and so we see that the two slots can 'see' out past some distance (basically the sensitivity of the sensor). When the sensor is idle, both slots detect the same amount of IR, the ambient amount radiated from the room or walls or outdoors. When a warm body like a human or animal passes by, it first intercepts one half of the PIR sensor, which causes a *positive differential* change between the two halves. When the warm body leaves the sensing area, the reverse happens, whereby the sensor generates a negative differential change. These change pulses are what is detected.



Specifications:-

- Color: White + Green OR White + Blue
- Infrared Sensor with Control Circuit Board
- The Sensitivity and Holding Time Can be Adjusted
- Working Voltage Range: DC 4.5V- 20V
- Current Drain: <60uA
- Detection Range: <140°
- Voltage Output: High/Low level Signal: 3.3V TTL output

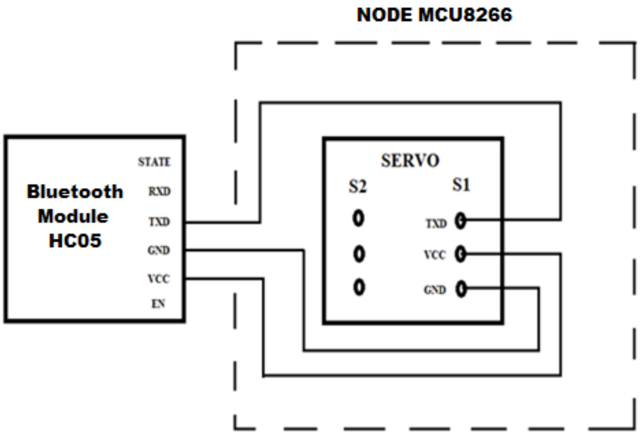
- Detection Distance: 3 to 7m (can be adjusted)
- Delay Time: 5 to 200s (Can be Adjusted, Default 5s +/- 3%)
- Blockade time: 2.5s (Default)
- Work temperature: -20-+80°C
- Dimension: 3.2cm x 2.4cm x 1.8cm (Approx.)
- Sensitive Setting: Turn to Right, Distance Increases (About 7M); Turn to Left, Distance Reduce (About 3M)

Time Setting: Turn to Right, Time Increases (About 200S); Turn to Left, Time Reduce (About 5S).

A6	Problem Statement:	Write a program to interface Single LED blinking using Node MCU and Bluetooth module.

Components required:

Sl.No.	COMPONENTS	QUANTITY
01	NODE MCU(ESP8266)	01
02	Micro USB cable	01
03	Bluetooth Module	01
04	LED	01
05	560Ω Resistor	01
06	Connecting/Jumper wires	05



Program

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(5,4);                    // RX(D5),TX(D6)
void setup()
{
// Open serial communications and wait for port to open:
Serial.begin(9600);
mySerial.begin(9600);

pinMode(15,OUTPUT);
}
void loop()
{                                         // run over and over
```

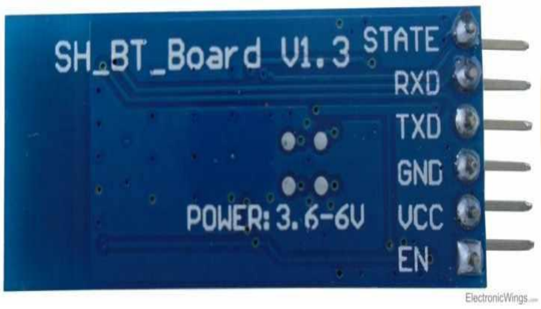
		char val;
		if (mySerial.available())
		{
		val=mySerial.read();
		if(val=='1')
		digitalWrite(15,HIGH);
		if(val=='0')
		digitalWrite(15,LOW);
		}
		}

	Output:	

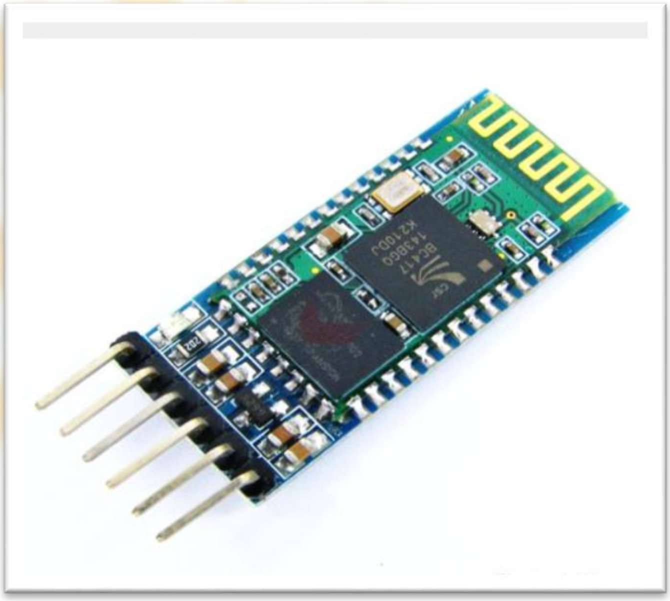
More Info:

Introduction:
Controlling an LED using a **NodeMCU (ESP8266)** and a **Bluetooth module (HC-05 or HC-06)** allows wireless operation from a smartphone or another Bluetooth-enabled device. This setup is commonly used in IoT applications, home automation, and wireless lighting control.

HC-05 Bluetooth Module Pin Diagram



HC-05 Bluetooth Module



PART B

B1	**Problem Statement:**	Write a Python Program to address data privacy for Patient Data Anonymization using Generalization(Age& Heart Rate)

Procedure:

Procedure:

1. Create a sample dataset containing PatientID, Age, HeartRate, Weight, and Location.
2. Identify sensitive fields to generalize (Age, HeartRate, Location).
3. Apply generalization rules:
   - Age → ranges (20-25, 26-30.)
   - HeartRate → categories (<80, 80-82, >82)
   - Location → group into regions
4. Display the anonymized dataset.

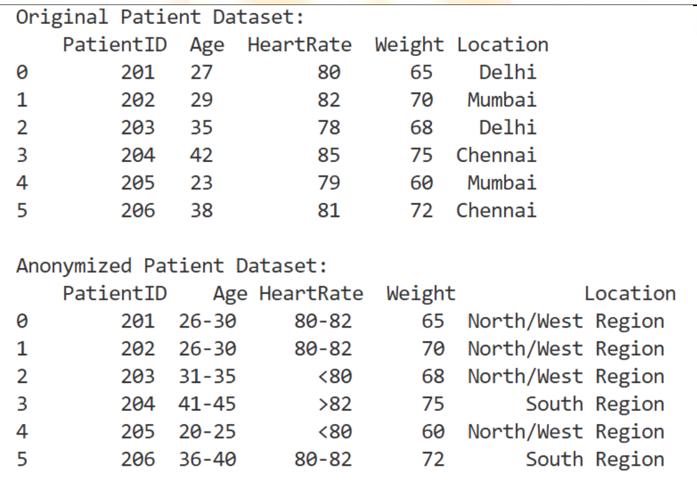
Library – install pandas, pip install pandas
# Import pandas for data handling

Program

```
import pandas as pd

data = {
    'PatientID': [201, 202, 203, 204, 205, 206],  # Unique patient ID
    'Age': [27, 29, 35, 42, 23, 38],              # Patient ages
    'HeartRate': [80, 82, 78, 85, 79, 81],        # Patient heart rates
    'Weight': [65, 70, 68, 75, 60, 72],           # Patient weight in kg
    'Location': ['Delhi', 'Mumbai', 'Delhi', 'Chennai', 'Mumbai', 'Chennai']
# Patient city
}   # Step 1: Simulate Patient Dataset


# Create DataFrame from data
df = pd.DataFrame(data)
print("Original Patient Dataset:\n", df)  # Display original data

# Step 2: Generalize Age - Function to convert age into ranges
def generalize_age(age):
    if age <= 25:
        return '20-25'
    elif age <= 30:
        return '26-30'
    elif age <= 35:
        return '31-35'
    elif age <= 40:
        return '36-40'
    else:
        return '41-45'

# Apply generalization to Age column
df['Age'] = df['Age'].apply(generalize_age)
```

```
# Step 3: Generalize HeartRate- Function to convert heart rate into categories

def generalize_hr(hr):
    if hr < 80:
        return '<80'
    elif hr <= 82:
        return '80-82'
    else:
        return '>82'

# Apply generalization to HeartRate column
df['HeartRate'] = df['HeartRate'].apply(generalize_hr)

# Step 4: Generalize Location- Function to group locations into regions
def generalize_location(loc):
    if loc in ['Delhi', 'Mumbai']:
        return 'North/West Region'
    else:
        return 'South Region'

# Apply generalization to Location column
df['Location'] = df['Location'].apply(generalize_location)

# Step 5: Display Anonymized Dataset
print("\nAnonymized Patient Dataset:\n", df)  # Show anonymized data
```

Output:



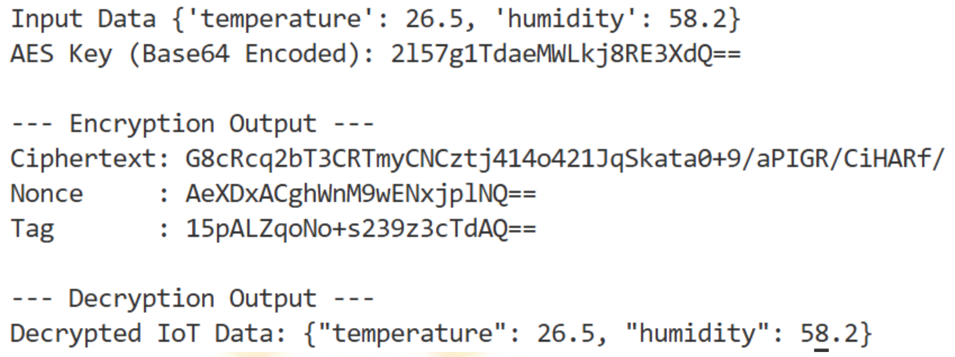
```
Original Patient Dataset:
    PatientID  Age  HeartRate  Weight Location
0        201   27         80      65    Delhi
1        202   29         82      70   Mumbai
2        203   35         78      68    Delhi
3        204   42         85      75  Chennai
4        205   23         79      60   Mumbai
5        206   38         81      72  Chennai

Anonymized Patient Dataset:
    PatientID    Age HeartRate  Weight           Location
0        201  26-30     80-82      65  North/West Region
1        202  26-30     80-82      70  North/West Region
2        203  31-35       <80      68  North/West Region
3        204  41-45       >82      75       South Region
4        205  20-25       <80      60  North/West Region
5        206  36-40     80-82      72       South Region
```

B2	**Problem Statement:**	Write a Python Program to address Symmetric Cryptography - Secure IoT Sensor Data of Agriculture Greenhouse using AES Encryption and Decryption
	Procedure:	1. Import the necessary libraries. 2. Create sample IoT sensor data. 3. Generate a random AES key. 4. Encrypt the data using AES in **EAX mode** (which provides confidentiality and integrity). 5. Decrypt the encrypted message to verify correctness. 6. Print both encrypted and decrypted results. Library: `pycryptodome` (install using `pip install pycryptodome`) • Modules used: o `Crypto.Cipher.AES` for AES encryption/decryption o `Crypto.Random` for generating a random key o `base64` for encoding binary data o `json` for handling sensor data

	Program	# Program: AES Encryption and Decryption for IoT Sensor Data to
		
		# Import required libraries - AES encryption/decryption
		
		from Crypto.Cipher import AES
		from Crypto.Random import get_random_bytes
		# To generate random AES key
		import base64 # To encode binary data to text
		import json # To handle JSON data
		
		# Step 1: Initialize IoT Sensor Data
		sensor_data = {
		"temperature": 26.5, # Temperature in Celsius
		"humidity": 58.2 # Humidity percentage
		}
		
		# Convert sensor data dictionary into string
		data_str = json.dumps(sensor_data)
		
		# Step 2: Generate AES Key (128-bit)
		key = get_random_bytes(16) # Generate 16 bytes = 128-bit key
		print("Input Data",sensor_data)
		print("AES Key :", base64.b64encode(key).decode())
		
		# Step 3: Encrypt the Data
		cipher = AES.new(key, AES.MODE_EAX) # Create AES cipher in EAX mode
		ciphertext, tag = cipher.encrypt_and_digest(data_str.encode()) # Encrypt data

```
# Encode binary data to Base64 for printing
cipher_b64 = base64.b64encode(ciphertext).decode()
nonce_b64 = base64.b64encode(cipher.nonce).decode()
tag_b64 = base64.b64encode(tag).decode()

# Print each part clearly
print("\n--- Encryption Output ---")
print("Ciphertext:", cipher_b64)
print("Nonce     :", nonce_b64)
print("Tag       :", tag_b64)

# Step 4: Decrypt the Data
cipher2 = AES.new(key, AES.MODE_EAX,
nonce=base64.b64decode(nonce_b64))
decrypted = cipher2.decrypt_and_verify(
    base64.b64decode(cipher_b64),
    base64.b64decode(tag_b64)
)

# Step 5: Display Decrypted Data
print("\n--- Decryption Output ---")
print("Decrypted IoT Data:", decrypted.decode())
```

Output:



```
Input Data {'temperature': 26.5, 'humidity': 58.2}
AES Key (Base64 Encoded): 2l57g1TdaeMWLkj8RE3XdQ==

--- Encryption Output ---
Ciphertext: G8cRcq2bT3CRTmyCNCztj414o421JqSkata0+9/aPIGR/CiHARf/
Nonce      : AeXDxACghWnM9wENxjplNQ==
Tag        : 15pALZqoNo+s239z3cTdAQ==

--- Decryption Output ---
Decrypted IoT Data: {"temperature": 26.5, "humidity": 58.2}
```

B3	**Problem Statement:**	Write a Python Program to address Asymmetric Cryptography: Securing IoT Smart Home Sensor Data using RSA Public-Key Encryption

Procedure

To implement RSA encryption and decryption to secure data transmission from smart home IoT sensors using public-key cryptography.

- **Library:** pycryptodome
  Install using:
- pip install pycryptodome
- **Modules used:**
  - Crypto.PublicKey.RSA → for key generation
  - Crypto.Cipher.PKCS1_OAEP → for RSA encryption/decryption
  - base64 → for readable output
  - json → for data formatting

Procedure

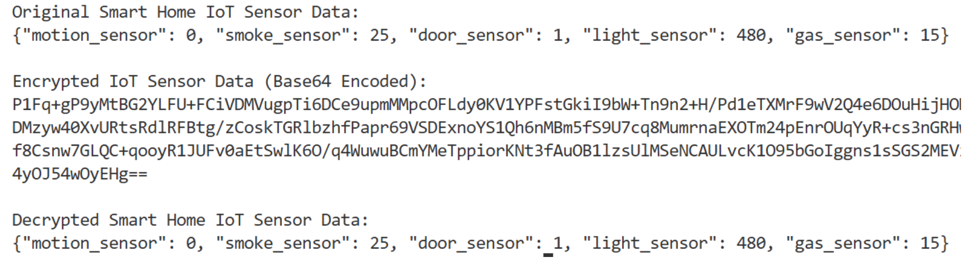
- Import all required modules.
- Create a simulated IoT sensor dataset (like temperature, humidity, light intensity).
- Generate a pair of RSA keys — **public key** for encryption and **private key** for decryption.
- Use the **public key** to encrypt sensor data using the **PKCS1_OAEP** scheme.
- Convert the encrypted data to **Base64** for easy viewing or transmission.
- Use the **private key** to decrypt and retrieve the original data.
- Print all intermediate and final results (keys, encrypted data, decrypted data).

Program

```
# Step 1: Import required libraries
from Crypto.PublicKey import RSA          # For RSA key pair generation
from Crypto.Cipher import PKCS1_OAEP      # For RSA encryption/decryption
import base64                             # For Base64 encoding
import json                               # For handling JSON data

# Step 2: Simulated Smart Home IoT Sensor Data (Numerical)

sensor_data = {
    "motion_sensor": 0,      # 0 = No motion, 1 = Motion detected
    "smoke_sensor": 25,      # Smoke level (0–100 scale)
    "door_sensor": 1,        # 1 = Door open, 0 = Closed
    "light_sensor": 480,     # Light intensity in lumens
    "gas_sensor": 15         # Gas concentration (ppm)
}
```

```
# Convert to JSON string for encryption
data_str = json.dumps(sensor_data)
print("Original Smart Home IoT Sensor Data:")
print(data_str)
print()

# Step 3: Generate RSA Key Pair
# RSA uses a public key for encryption and private key for decryption
key_pair = RSA.generate(2048)
public_key = key_pair.publickey()

# Step 4: Encrypt IoT Data using Public Key (PKCS1_OAEP)
# PKCS1_OAEP adds padding and randomness for stronger encryption
cipher_rsa = PKCS1_OAEP.new(public_key)        # Create cipher object
encrypted_data = cipher_rsa.encrypt(data_str.encode())  # Encrypt IoT
data
encrypted_b64 = base64.b64encode(encrypted_data).decode()  # Convert
to Base64

print("Encrypted IoT Sensor Data (Base64 Encoded):")
print(encrypted_b64)
print()

# Step 5: Decrypt Data using Private Key
cipher_rsa_dec = PKCS1_OAEP.new(key_pair)        # Create decryptor
decrypted_data =
cipher_rsa_dec.decrypt(base64.b64decode(encrypted_b64))

print("Decrypted Smart Home IoT Sensor Data:")
print(decrypted_data.decode())
```

Output:



```
Original Smart Home IoT Sensor Data:
{"motion_sensor": 0, "smoke_sensor": 25, "door_sensor": 1, "light_sensor": 480, "gas_sensor": 15}

Encrypted IoT Sensor Data (Base64 Encoded):
P1Fq+gP9yMtBG2YLFU+FCiVDMVugpTi6DCe9upmMMpcOFLdy0KV1YPFstGkiI9bW+Tn9n2+H/Pd1eTXMrF9wV2Q4e6DOuHijHO
DMzyw40XvURtsRdlRFBtg/zCoskTGRlbzhfPapr69VSDExnoYS1Qh6nMBm5fS9U7cq8MumrnaEXOTm24pEnrOUqYyR+cs3nGRH
f8Csnw7GLQC+qooyR1JUFv0aEtSwlK6O/q4WuwuBCmYMeTppiorKNt3fAuOB1lzsUlMSeNCAULvcK1O95bGoIggns1sSGS2MEV
4yOJ54wOyEHg==

Decrypted Smart Home IoT Sensor Data:
{"motion_sensor": 0, "smoke_sensor": 25, "door_sensor": 1, "light_sensor": 480, "gas_sensor": 15}
```

B4	**Problem Statement:**	**Write a Python Program to address Data Integrity: Ensuring Data Integrity in Traffic Monitoring IoT Systems using SHA-256 Hashing.**

Procedure

To verify the integrity of traffic IoT data (vehicle count, speed, and signal status) using the SHA-256 hash algorithm, ensuring that transmitted data remains unaltered.

- **Modules Used:**
  - `hashlib` → To compute SHA-256 hash values
  - `json` → To format structured traffic data

Procedure:

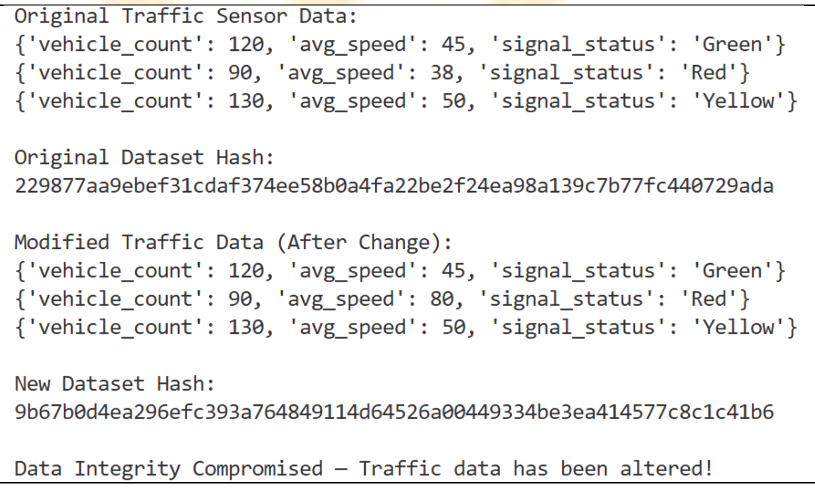
1. Import required libraries (`hashlib`, `json`).
2. Create a list containing **3 sets of traffic sensor readings** (vehicle count, average speed, signal status).
3. Convert the dataset into JSON format and compute its **original SHA-256 hash**.
4. Modify one record to simulate **data tampering**.
5. Recompute the hash value and compare it with the original hash.
6. Display whether data integrity is **verified or compromised**.

Program

```
# Aim: To check if traffic IoT data has been altered using hash
comparison

# Step 1: Import necessary libraries
import hashlib   # For computing SHA-256 hash
import json      # For JSON data formatting

# Step 2: Simulated Traffic IoT Sensor Data
traffic_data = [
    {"vehicle_count": 120, "avg_speed": 45, "signal_status": "Green"},
    {"vehicle_count": 90, "avg_speed": 38, "signal_status": "Red"},
    {"vehicle_count": 130, "avg_speed": 50, "signal_status": "Yellow"}
]

print("Original Traffic Sensor Data:")
for record in traffic_data:
    print(record)
print()

# Step 3: Function to Compute SHA-256 Hash
#Convert dataset to JSON string and generate SHA-256 hash"""

def compute_hash(data):
    json_data = json.dumps(data, sort_keys=True)  # Ensures consistent
```

```
format
    return hashlib.sha256(json_data.encode()).hexdigest()

# Compute the original dataset hash
original_hash = compute_hash(traffic_data)
print("Original Dataset Hash:")
print(original_hash)
print()

# Step 4: Simulate Data Tampering
# Example: an attacker changes the average speed in one record
traffic_data[1]["avg_speed"] = 80  # Tampered data

print("Modified Traffic Data (After Change):")
for record in traffic_data:
    print(record)
print()

# Compute hash after modification
new_hash = compute_hash(traffic_data)
print("New Dataset Hash:")
print(new_hash)
print()

# Step 5: Compare Hashes to Verify Integrity
if original_hash == new_hash:
    print("Data Integrity Verified — No changes detected.")
else:
    print("Data Integrity Compromised — Traffic data has been altered!")
```

Output:

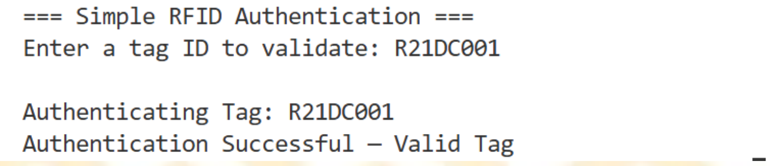


```
Original Traffic Sensor Data:
{'vehicle_count': 120, 'avg_speed': 45, 'signal_status': 'Green'}
{'vehicle_count': 90, 'avg_speed': 38, 'signal_status': 'Red'}
{'vehicle_count': 130, 'avg_speed': 50, 'signal_status': 'Yellow'}

Original Dataset Hash:
229877aa9ebef31cdaf374ee58b0a4fa22be2f24ea98a139c7b77fc440729ada

Modified Traffic Data (After Change):
{'vehicle_count': 120, 'avg_speed': 45, 'signal_status': 'Green'}
{'vehicle_count': 90, 'avg_speed': 80, 'signal_status': 'Red'}
{'vehicle_count': 130, 'avg_speed': 50, 'signal_status': 'Yellow'}

New Dataset Hash:
9b67b0d4ea296efc393a764849114d64526a00449334be3ea414577c8c1c41b6

Data Integrity Compromised — Traffic data has been altered!
```

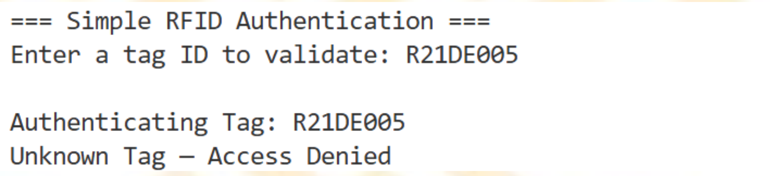
B5	Problem Statement:	Write a Python Program to address Data Authentication RFID Tag Authentication using Challenge–Response Mechanism with SHA-256
	Procedure and requirements	To implement a simple RFID-based authentication system using a **challenge–response mechanism** secured with the **SHA-256 hash function**, ensuring that only authorized RFID tags gain access. 1. Define a set of **virtual RFID tags** with their corresponding **secret keys**. 2. Generate a random **nonce** (challenge) each time a tag attempts authentication. 3. Compute the **SHA-256 hash** of the concatenation of tag ID, secret key, and nonce. 4. Compare the computed hash (response) with the expected hash. 5. If both match → Authentication successful; otherwise, access is denied. 6. Test the system with valid and invalid RFID tag IDs.
	Program	

```
# Simple RFID authentication: user enters a tag (valid or fake) and program
verifies it.

import hashlib
import random
import string

# Valid RFID tags with their secret keys
rfid_tags = {
    "R21DC001": "Abhi@123",
     "R21DC002": "Akash_456"
}

def generate_nonce(length=8):
    """Return a random alphanumeric nonce (challenge)."""
    return ''.join(random.choices(string.ascii_letters + string.digits,
k=length))

def compute_hash(tag_id, secret, nonce):
    """Compute SHA-256 hash of tag_id + secret + nonce (hex
string)."""
    return hashlib.sha256((tag_id + secret + nonce).encode()).hexdigest()

def authenticate(tag_id):
    """Authenticate a tag ID. Prints result and returns True/False."""
    print(f"\nAuthenticating Tag: {tag_id}")
    # If tag not registered, deny immediately

    if tag_id not in rfid_tags:
        print("Unknown Tag — Access Denied")
        return False

    # Registered tag: perform challenge-response (simulated)
    secret = rfid_tags[tag_id]            # stored secret for this tag
    nonce = generate_nonce()              # reader challenge
    response = compute_hash(tag_id, secret, nonce)   # what tag would send
    expected = compute_hash(tag_id, secret, nonce)   # what reader expects
    # Compare (they will match in this simulation)

    if response == expected:
        print("Authentication Successful — Valid Tag")
        return True
    else:
        print("Authentication Failed")
        return False

# ---- Main ----
print("=== Simple RFID Authentication ===")
user_tag = input("Enter a tag ID to validate: ").strip()  # prompt user
authenticate(user_tag)
```

Output:



```
=== Simple RFID Authentication ===
Enter a tag ID to validate: R21DC001

Authenticating Tag: R21DC001
Authentication Successful — Valid Tag
```



```
=== Simple RFID Authentication ===
Enter a tag ID to validate: R21DE005

Authenticating Tag: R21DE005
Unknown Tag — Access Denied
```

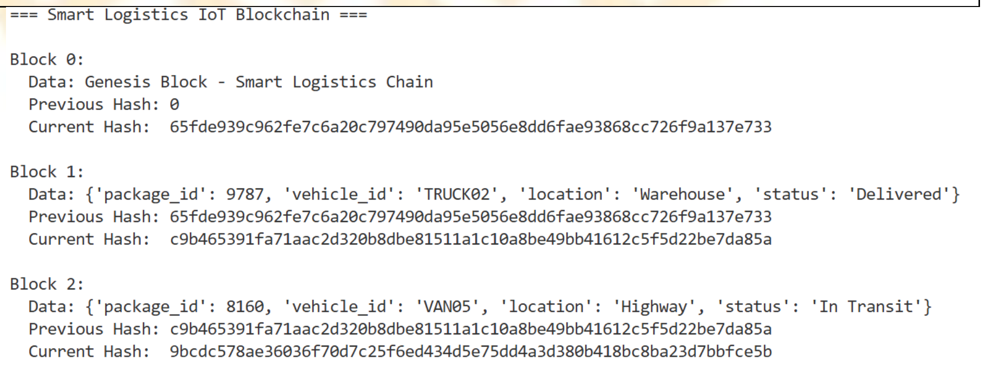
B6	**Problem**	**Write a Python Program to address Blockchain-based Secure**

	Statement:	Tracking of Logistics Data in IoT Systems
	Procedure	1. **Initialize** a blockchain with a **genesis block** (the first block). 2. **Define** a function to add new blocks with logistics data. 3. **Simulate** logistics data for multiple shipments — each containing package info, vehicle ID, location, and status. 4. **Hash** each block using SHA-256 and link it to the previous one for integrity. 5. **Display** the blockchain to show how data from IoT logistics sensors is securely chained.

Program

```
# Aim: Securely store logistics data using blockchain

import hashlib, json, time, random

# Step 1: Initialize blockchain
blockchain = []

# Step 2: Create genesis (first) block
first_block = {
    "index": 0,
    "data": "Genesis Block - Smart Logistics Chain",
    "previous_hash": "0"
}

# Compute hash for genesis block
first_block["hash"] =
hashlib.sha256(json.dumps(first_block).encode()).hexdigest()
blockchain.append(first_block)

# Step 3: Function to add a new block

def add_block(data):
    """Add new logistics record to the blockchain."""
    previous_block = blockchain[-1]
    block = {
        "index": len(blockchain),
        "data": data,
        "previous_hash": previous_block["hash"]
    }
    # Generate a hash for the current block
    block["hash"] =
hashlib.sha256(json.dumps(block).encode()).hexdigest()
    blockchain.append(block)

# --------------------------
# Step 4: Generate logistics data
# --------------------------
def generate_logistics_data():
    """Simulate logistics tracking data for IoT-enabled vehicles."""
    return {
        "package_id": random.randint(1000, 9999),
        "vehicle_id": random.choice(["TRUCK01", "TRUCK02", "VAN05", "VAN08"]),
        "location": random.choice(["Warehouse", "Highway", "City Hub", "Customer Site"]),
        "status": random.choice(["Loaded", "In Transit", "Delivered", "Pending"])
    }

# ---------------------------
# Step 5: Add 3 logistics data blocks(0,1,2)
# ---------------------------
for _ in range(2):
    data = generate_logistics_data()
    add_block(data)
    time.sleep(0.2)

# ---------------------------
# Step 6: Display entire blockchain
# ---------------------------
print("=== Smart Logistics IoT Blockchain ===\n")
for block in blockchain:
    print(f"Block {block['index']}:")
    print(f"  Data: {block['data']}")
    print(f"  Previous Hash: {block['previous_hash']}")
    print(f"  Current Hash:  {block['hash']}\n")
```

Output:



```
=== Smart Logistics IoT Blockchain ===

Block 0:
  Data: Genesis Block - Smart Logistics Chain
  Previous Hash: 0
  Current Hash:  65fde939c962fe7c6a20c797490da95e5056e8dd6fae93868cc726f9a137e733

Block 1:
  Data: {'package_id': 9787, 'vehicle_id': 'TRUCK02', 'location': 'Warehouse', 'status': 'Delivered'}
  Previous Hash: 65fde939c962fe7c6a20c797490da95e5056e8dd6fae93868cc726f9a137e733
  Current Hash:  c9b465391fa71aac2d320b8dbe81511a1c10a8be49bb41612c5f5d22be7da85a

Block 2:
  Data: {'package_id': 8160, 'vehicle_id': 'VAN05', 'location': 'Highway', 'status': 'In Transit'}
  Previous Hash: c9b465391fa71aac2d320b8dbe81511a1c10a8be49bb41612c5f5d22be7da85a
  Current Hash:  9bcdc578ae36036f70d7c25f6ed434d5e75dd4a3d380b418bc8ba23d7bbfce5b
```

Course Handled by: **HoD**

1. **A Section -**