ATME College of Engineering

Department of Electronics and Communication Engineering

(ACADEMIC YEAR 2025-26)

LABORATORY MANUAL

Course: IOT (INTERNET OT THINGS) LABORATORY MANUAL

Course Code: BEC657C

SEMESTER: VI

Prepared by

Nagesh M S
Assistant Professor
Dept. of ECE

INSTITUTION VISION AND MISSION

VISION:

Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

MISSION:

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as tourch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence

DEPARTMENT VISION AND MISSION

VISION

To develop highly skilled and globally competent professionals in the field of Electronics and Communication Engineering to meet industrial and social requirements with ethical responsibility.

MISSION

- To provide State-of-art technical education in Electronics and Communication at undergraduate and post-graduate levels, to meet the needs of the profession and society and achieve excellence in teaching-learning and research.
- To develop talented and committed human resource, by providing an opportunity for innovation, creativity and entrepreneurial leadership with high standards of professional ethics, transparency and accountability.
- To function collaboratively with technical Institutes/Universities/Industries, offer opportunities for interaction among faculty-students and promote networking with alumni, industries and other stake-holders.

Program Outcomes (POs)

PO1: Engineering Knowledge: Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development.

PO3: Design/Development of Solutions: Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required.

PO4: Conduct Investigations of Complex Problems: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions.

PO5: Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems.

PO6: The Engineer and The World: Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment.

PO7: Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws.

PO8: Individual and Collaborative Team work: Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

PO9: Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences

PO10: Project Management and Finance: Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.

PO11: Life-Long Learning: Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO1: To have the capability to understand and adopt the technological advancements with the usage of modern tool to analyze and design embedded system or processes for variety of applications.

PSO2: To work effectively in a group as an independent visionary, team member and leader having the ability to understand the requirement and develop feasible solutions to emerge as potential core or electronic engineer

Program Educational Objectives (PEOs)

PEO1: To produce graduates to excel in the profession, higher education and pursue research exercises in Electronics and Communication Engineering.

PEO2: To create technically able alumni with the capacity to examine, plan, to create and execute Electronics and Communication frameworks thereby involving in deep routed learning.

IoT (Internet of Things) Lab		Semester	**6**
Course Code	**BEC657C**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Credits	01	Exam Hours	3
Examination type (SEE)	Practical		

Course Objectives:

This course will enable students to

- To impart necessary and practical knowledge of components of the Internet of Things
- To develop skills required to build real-life IoT-based projects.

Sl.No.	Experiments
1(i)	To interface LED/Buzzer with Arduino /Raspberry Pi and write a program to 'turn ON' LED for 1 sec after every 2 seconds.
1(ii)	To interface the Push button/Digital sensor (IR/LDR) with Arduino /Raspberry Pi and write a program to 'turn ON' LED when a push button is pressed or at sensor detection.
2 (i)	To interface the DHT11 sensor with Arduino /Raspberry Pi and write a program to print temperature and humidity readings.
2(ii)	To interface OLED with Arduino /Raspberry Pi and write a program to print its temperature and humidity readings.
3	To interface the motor using a relay with Arduino /Raspberry Pi and write a program to 'turn ON' the motor when a push button is pressed.
4(i)	Write an Arduino/Raspberry Pi program to interface the Soil Moisture Sensor.
4(ii)	Write an Arduino/Raspberry Pi program to interface the LDR/Photo Sensor.
5	Write a program to interface an Ultrasonic Sensor with Arduino /Raspberry Pi.
6	Write a program on Arduino/Raspberry Pi to upload temperature and humidity data to thingspeak cloud.
7	Write a program on Arduino/Raspberry Pi to retrieve temperature and humidity data from thingspeak cloud.
8	Write a program to interface LED using Telegram App.
9	Write a program on Arduino/Raspberry Pi to publish temperature data to the MQTT broker.
10	Write a program to create a UDP server on Arduino/Raspberry Pi and respond with humidity data to the UDP client when requested.
11	Write a program to create a TCP server on Arduino /Raspberry Pi and respond with humidity data to the TCP client when requested.
12	Write a program on Arduino / Raspberry Pi to subscribe to the MQTT broker for temperature data and print it.

Course outcomes (Course Skill Set):

At the end of the course, the student will be able to:

- Explain the Internet of Things and its hardware and software components.
- Interface I/O devices, sensors & communication modules.
- Remotely monitor data and control devices.
- Develop real-life IoT-based projects.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment will be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- The record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- The total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightage is to be given for neatness and submission of record/write-up on time.
- The department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to **20 marks** (40% of the maximum marks).

The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.
- **SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute**.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted within the schedule mentioned in the university's academic calendar.
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners. General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.The minimum duration of SEE is 02 hours

Suggested Learning Resources:

- Vijay Madisetti, Arshdeep Bahga, Internet of Things. "A Hands-on Approach", University Press
- Dr. SRN Reddy, Rachit Thukral, and Manasi Mishra, "Introduction to Internet of Things: A Practical Approach", ETI Labs
- Pethuru Raj and Anupama C Raman, "The Internet of Things: Enabling Technologies, Platforms, and Use Cases", CRC Press
- Jeeva Jose, "Internet of Things", Khanna Publishing House, Delhi
- Adrian McEwen, "Designing the Internet of Things", Wiley

Raj Kamal, "Internet of Things: Architecture and Design", McGraw Hill

CYCLE OF EXPERIMENTS

Subject: Internet of Things Lab **Subject Code: BECL657C**

CYCLE-I

1.(a) To interface LED/Buzzer with Arduino /Raspberry Pi and write a program to 'turn ON' LED for 1 sec after every 2 seconds.

1.(b) To interface the Push button/Digital sensor (IR/LDR) with Arduino /Raspberry Pi and write a program to 'turn ON' LED when a push button is pressed or at sensor detection.

2.(a) To interface the DHT11 sensor with Arduino /Raspberry Pi and write a program to print temperature and humidity readings.

2.(b) To interface OLED with Arduino /Raspberry Pi and write a program to print its temperature and humidity readings.

3. To interface the motor using a relay with Arduino /Raspberry Pi and write a program to 'turn ON' the motor when a push button is pressed.

4.(a) Write an Arduino/Raspberry Pi program to interface the Soil Moisture Sensor.

4.(b) Write an Arduino/Raspberry Pi program to interface the LDR/Photo Sensor.

5. Write a program to interface an Ultrasonic Sensor with Arduino /Raspberry Pi.

Cycle-II

6. Write a program on Arduino/Raspberry Pi to upload temperature and humidity data to thing speak cloud.

7. Write a program on Arduino/Raspberry Pi to retrieve temperature and humidity data from thing speak cloud

8. Write a program to interface LED using Telegram App.

9. Write a program on Arduino/Raspberry Pi to publish temperature data to the MQTT broker.

10. Write a program to create a UDP server on Arduino/Raspberry Pi and respond with humidity data to the UDP client when requested.

11. Write a program to create a TCP server on Arduino /Raspberry Pi and respond with humidity data to the TCP client when requested.

12. Write a program on Arduino / Raspberry Pi to subscribe to the MQTT broker for temperature data and print it.

IOT LAB - QUESTION BANK – 2025-2026

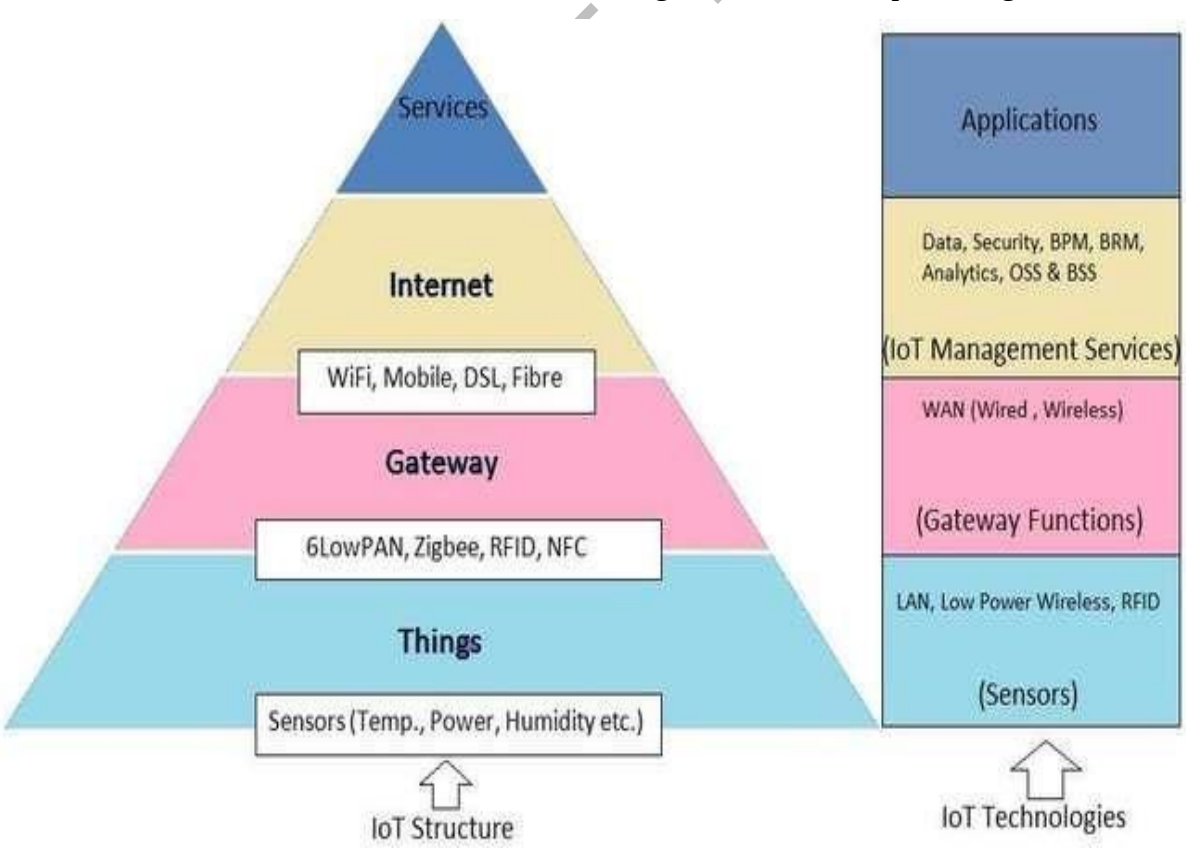
Program No.	List of Programs
1.	To interface LED with Arduino /Raspberry Pi and write a program to 'turn ON' LED for 1 sec after every 2 seconds.
2.	To interface Buzzer with Arduino /Raspberry Pi and write a program to 'turn ON' Buzzer for 1 sec after every 2 seconds.
3.	To interface the DHT11 sensor with Arduino /Raspberry Pi and write a program to print temperature and humidity readings.
4.	To interface OLED with Arduino /Raspberry Pi and write a program to print its temperature and humidity readings.
5.	To interface the motor using a relay with Arduino /Raspberry Pi and write a program to 'turn ON' the motor when a push button is pressed.
6.	Write an Arduino/Raspberry Pi program to interface the Soil Moisture Sensor.
7.	Write an Arduino/Raspberry Pi program to interface the LDR.
8.	Write a program to interface an Ultrasonic Sensor with Arduino /Raspberry Pi.
9.	Write a program on Arduino/Raspberry Pi to upload temperature and humidity data to thingspeak cloud.
10.	Write a program on Arduino/Raspberry Pi to retrieve temperature and humidity data from thingspeak cloud.
11.	Write a program to interface LED using Telegram App
12.	Write a program on Arduino/Raspberry Pi to publish temperature data to the MQTT broker.
13.	Write a program to create a UDP client on Arduino/Raspberry Pi and respond with humidity data to the UDP server when requested.
14.	Write a program to create a TCP client on Arduino /Raspberry Pi and respond with humidity data to the TCP server when requested.
15.	Write a program on Arduino / Raspberry Pi to subscribe to the MQTT broker for temperature data and print it

Introduction to Internet of Things (IoT)

Introduction: IOT stands for "Internet of Things". The IOT is a name for the vast collection of "things" that are being networked together in the home and workplace (up to 20 billion by 2020 according to Gardner, a technology consulting firm).

Characteristics of the IOT

Networking	These IOT devices talk to one another (M2M communication) or to servers located in the local network or on the Internet. Being on the network allows the device the common ability to consume and produce data.
Sensing	IOT devices sense something about their environment.
Actuators	IOT devices that do something. Lock doors, beep, turn lights on, or turn the TV on



Communications in IoT

Communications are important to IOT projects. In fact, communications are core to the whole genre. There is a trade-off for IOT devices. The more complex the protocols and higher the data rates, the more powerful processor needed and the more electrical power the IOT device will consume.

TCP/IP base communications (think web servers; HTTP-based commutation (like REST servers); streams of data; UDP) provide the most flexibility and functionality at a cost of processor and electrical power.

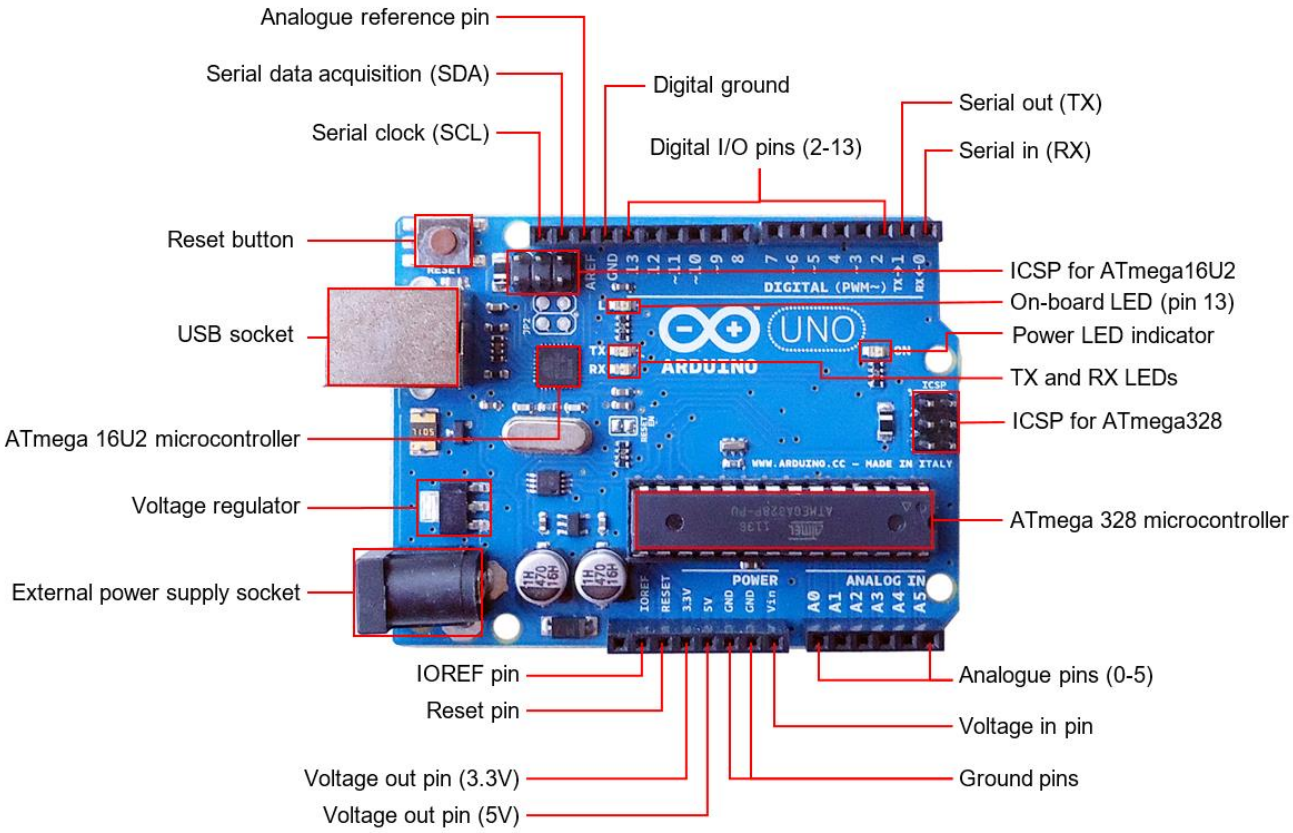
Low-power Bluetooth and Zigbee types of connections allow much lower power for connections with the corresponding decrease in bandwidth and functionality. IOT projects can be all over the map with requirements for communication flexibility and data bandwidth requirements.

Arduino in IoT

In IoT applications the Arduino is used to collect the data from the sensors/devices to send it to the internet and receives data for purpose of control of actuators.

Arduino Uno

Introduction: 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. The word "uno" means "one" in Italian and was chosen to mark the initial release of Arduino Software.



Features of the Arduino

1. Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.

2. The board functions can be controlled by sending a set of instructions to the microcontroller on the board via Arduino IDE.
3. Arduino IDE uses a simplified version of C++, making it easier to learn to program.
4. Arduino provides a standard form factor that breaks the functions of the micro- controller into a more accessible package.

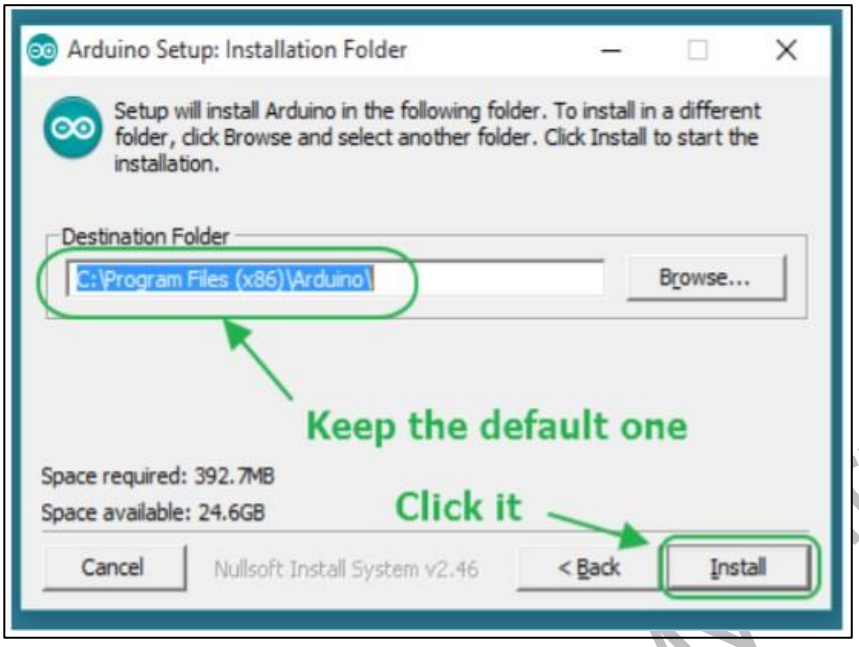
Arduino IDE(Integrated Development Environment)

Introduction: The Arduino Software (IDE) is easy-to-use and is based on the Processing programming environment. The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++. The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

The Arduino Software (IDE) – contains:

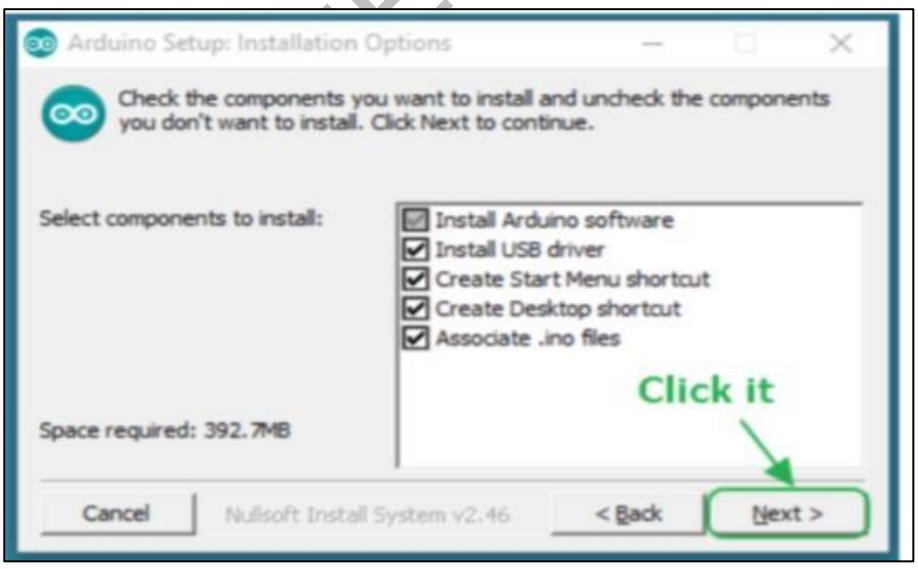
- A text editor for writing code
- A message area
- A text consoles
- A toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

Installation of Arduino Software (IDE)



Step1: Downloading

- To install the Arduino software, download this page: http://arduino.cc/en/Main/Software and proceed with the installation by allowing the driver installation process.

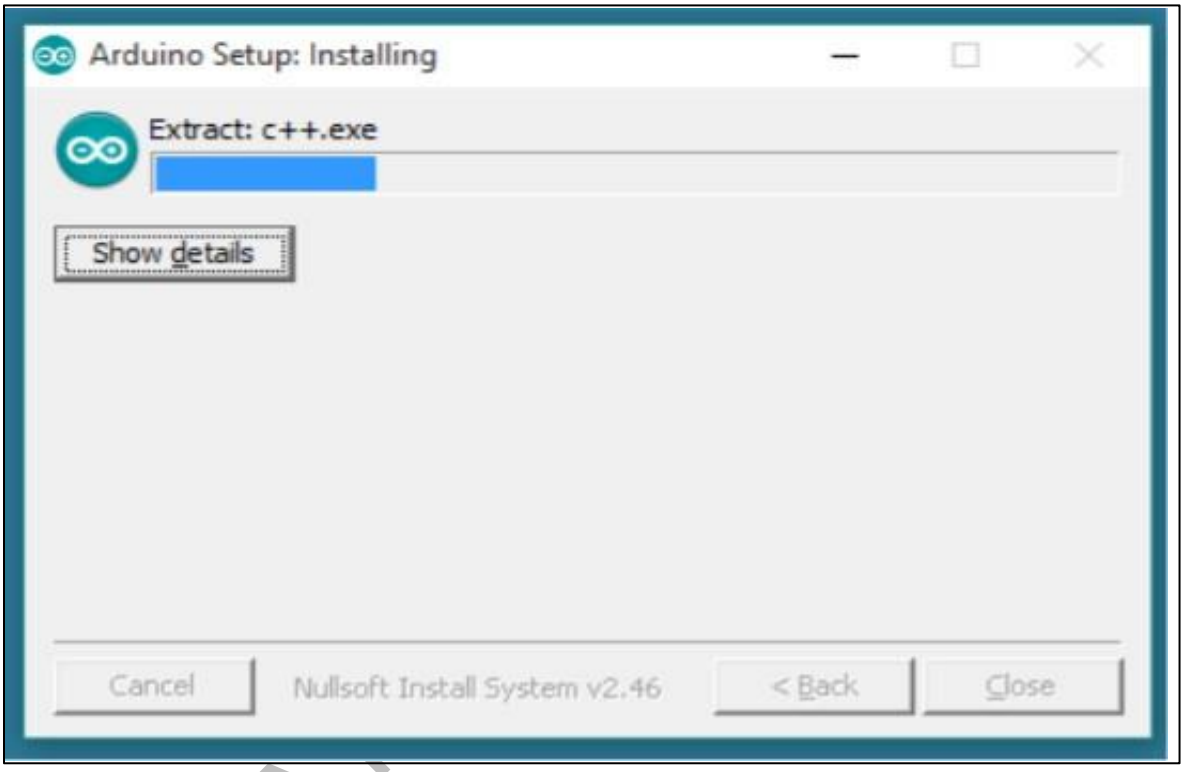


Step 2: Directory Installation

- Choose the installation directory.

Step 3: Extraction of Files

- The process will extract and install all the required files to execute properly the Arduino Software (IDE)

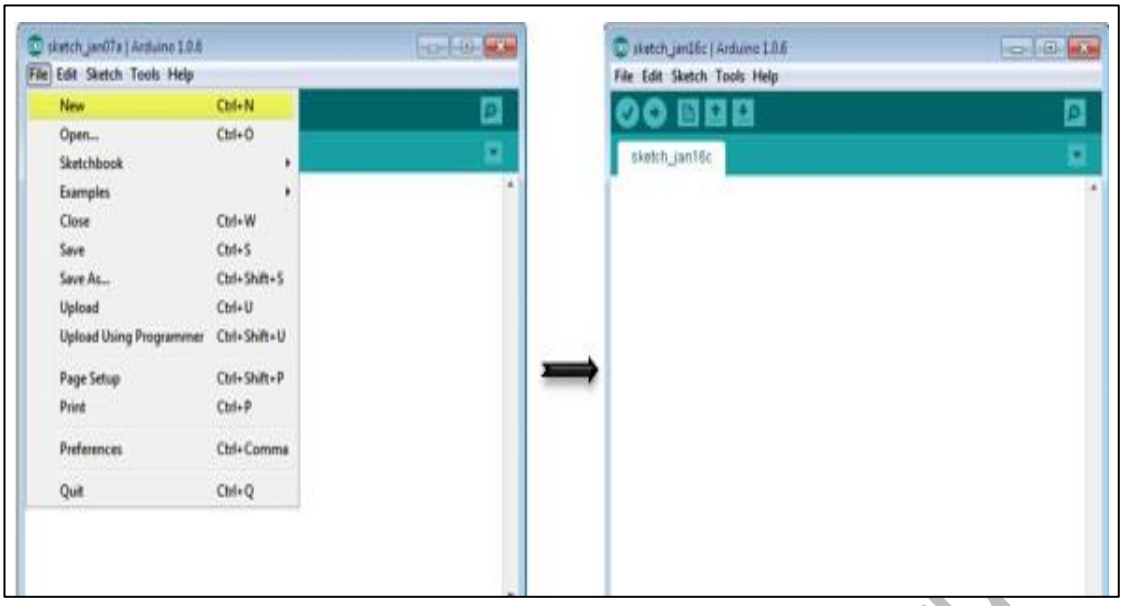


Step 4: Connecting the board

- The USB connection with the PC is necessary to program the board and not just to power it up. The Uno and Mega automatically draw power from either the USB or an external power supply. Connect the board to the computer using the USB cable. The green power LED (labelled PWR) should go on.

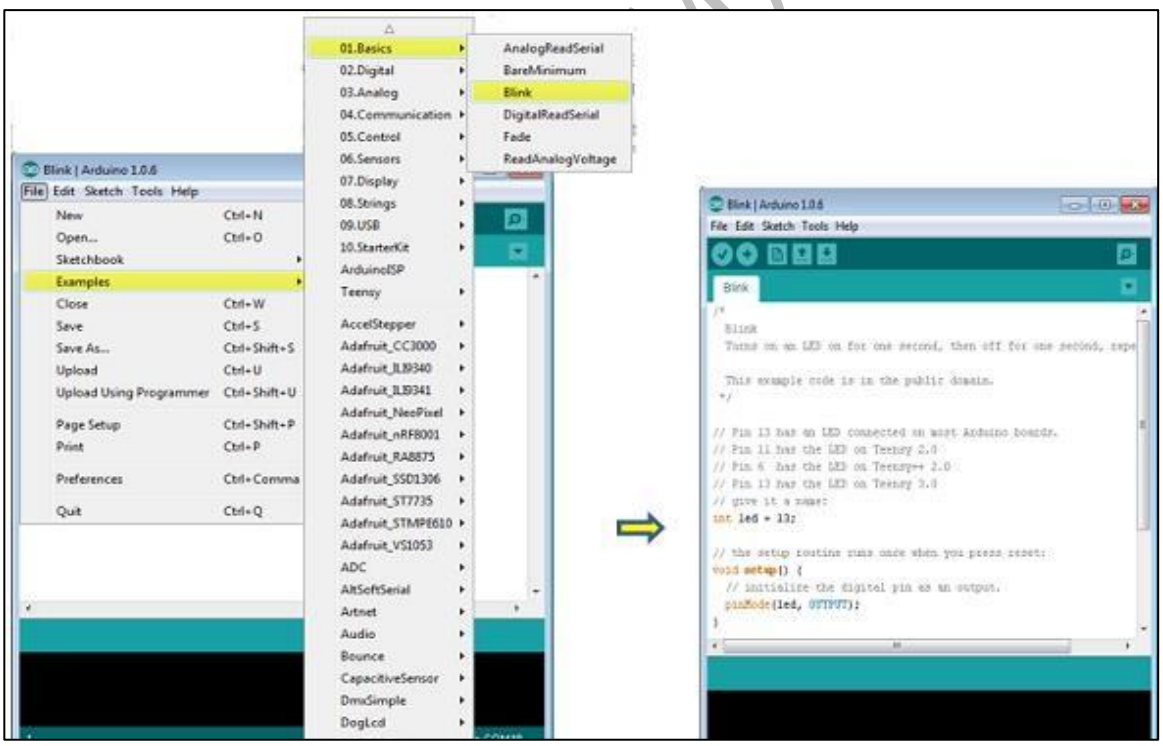
Step 5: Working on the new project

- Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit.
- Open a new sketch File by clicking on New.



Step 6: Working on an existing project

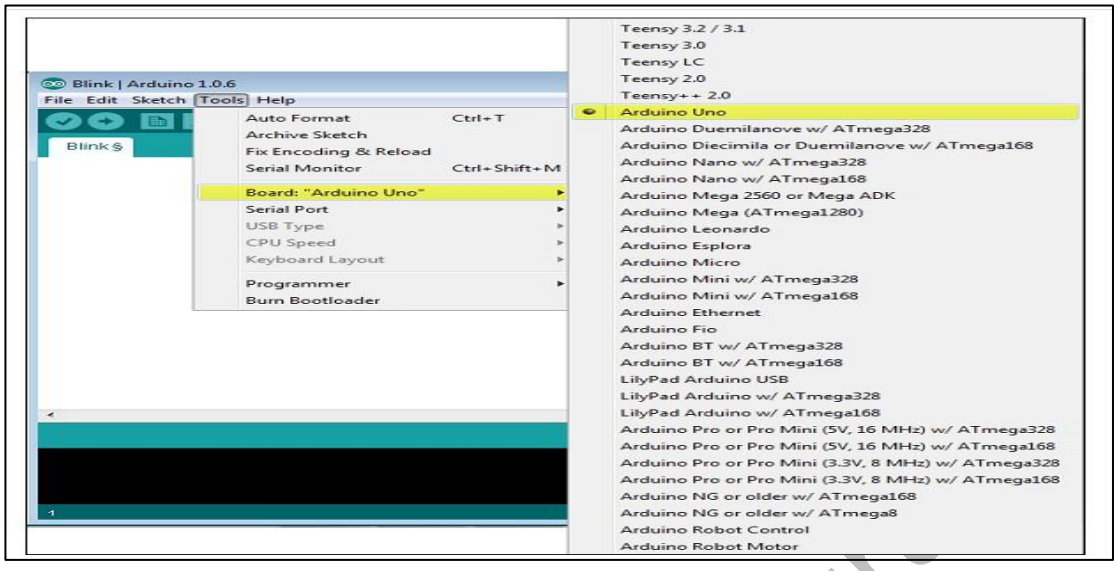
- To open an existing project example, select File → Example → Basics → Blink.



Step 7: **Select your Arduino board.**

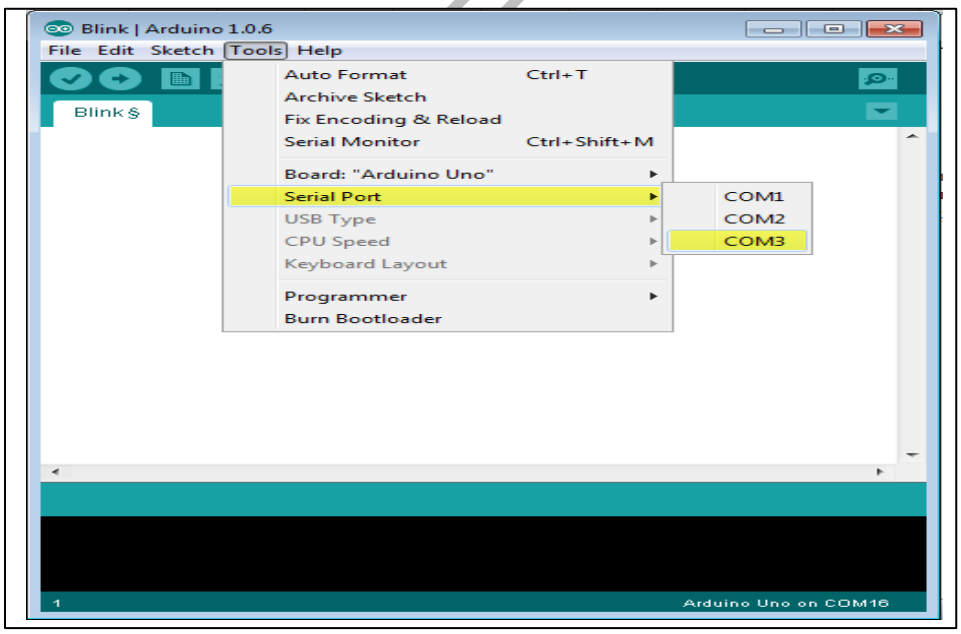
- To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

- Go to Tools → Board and select your board.



Step 8: Select your serial port

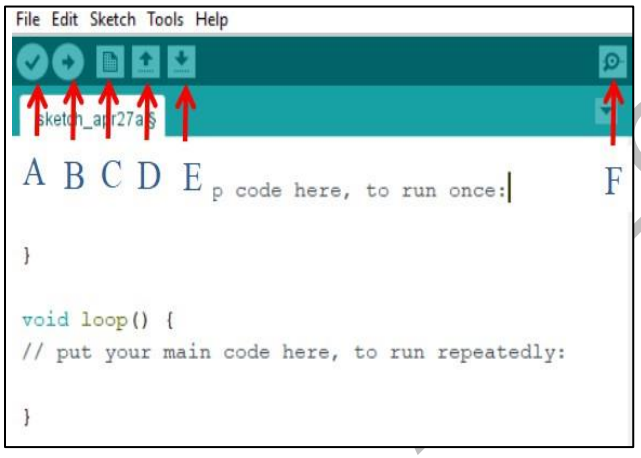
- Select the serial device of the Arduino board.
- Go to Tools → Serial Port menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports).
- To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.



Step 9: Upload the program to your board.

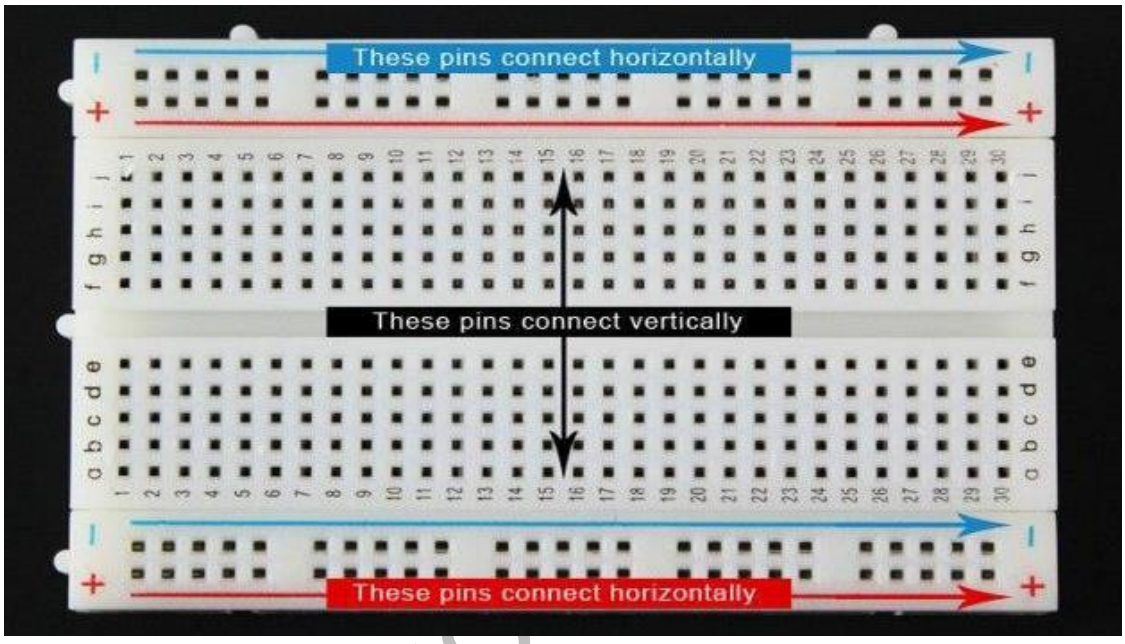
- Click the "Upload" button in the environment.
- Wait a few seconds; you will see the RX and TX LEDs on the board, flashing.
- If the upload is successful, the message "Done uploading" will appear in the status bar.

A	Verify
B	Upload
C	New
D	Open
E	Save
F	Serial Motor

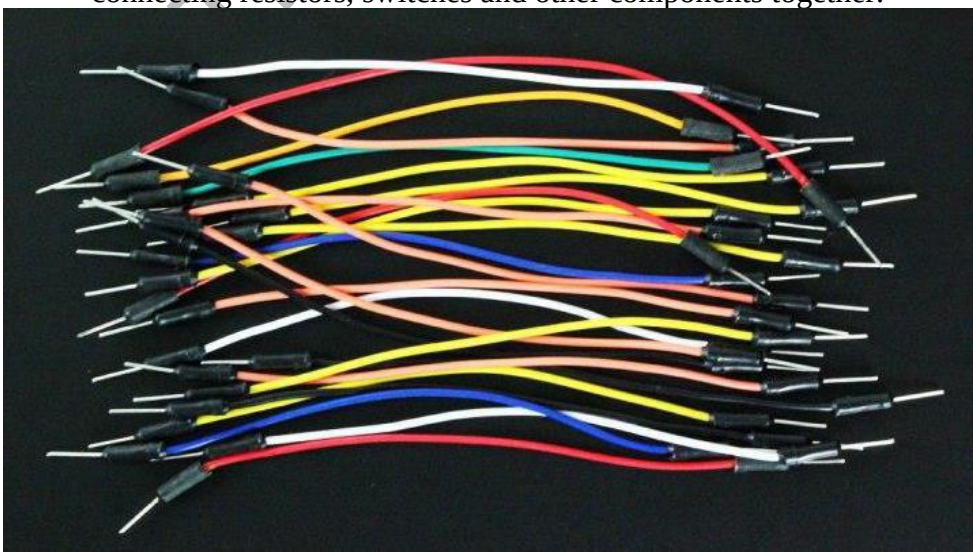


Arduino Breadboard

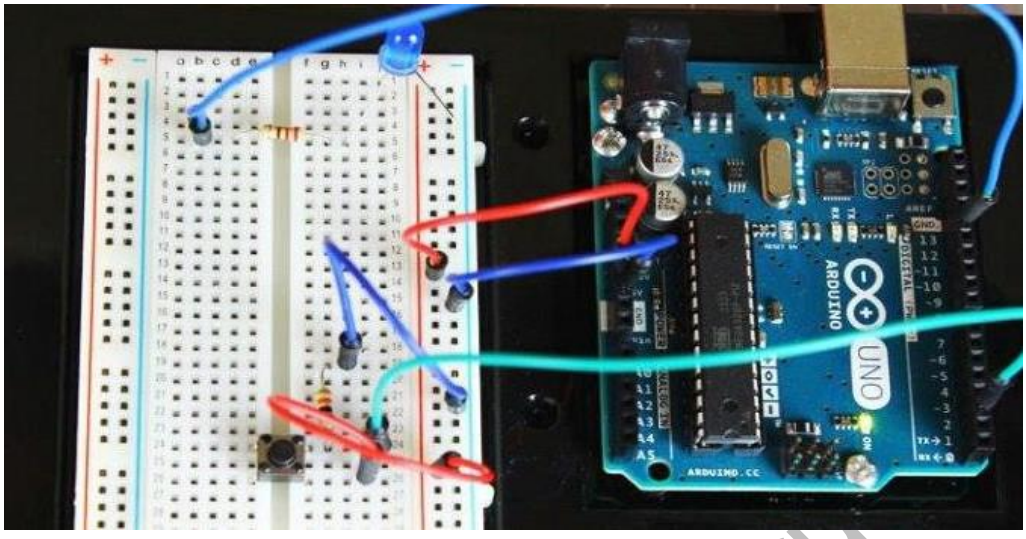
Another very important item when working with Arduino is a solderless breadboard. This device allows you to prototype your Arduino project without having to permanently solder the circuit together. Using a breadboard allows you to create temporary prototypes and experiment with different circuit designs. Inside the holes (tie points) of the plastic housing, are metal clips which are connected to each other by strips of conductive material.



On a side note, the breadboard is not powered on its own and needs power brought to it from the Arduino board using jumper wires. These wires are also used to form the circuit by connecting resistors, switches and other components together.



Here is a visual of what a completed Arduino circuit looks like when connected to a breadboard.



How To Program Arduino

Once the circuit has been created on the breadboard, you'll need to upload the program (known as a sketch) to the Arduino. Steps to execute program.

1. Install the Arduino IDE:

Download and install the Arduino IDE (Integrated Development Environment) from the official Arduino website: Arduino Software.

2.Connect Arduino Uno to Your Computer:

Use a USB cable to connect your Arduino Uno to

your computer. 3.Open Arduino IDE:

Launch the Arduino IDE on your computer.

4. Select Board and Port:

Go to "Tools" and select the appropriate board model (e.g., Arduino Uno) from the "Board" menu.

Also, select the correct port your Arduino Uno is connected to from

the "Port" menu. 5.Write or Open a Program:

Write your program in the Arduino IDE or open an existing one.

If you're new to Arduino, you might start with a simple example from the "File" -> "Examples" menu.

6.Verify and Compile:

Click the "Verify" (checkmark) button to check for any syntax

errors in your code. If there are no errors, the code will be compiled.

7.Upload the Program:

Click the "Upload" (right arrow) button to upload the compiled code to your Arduino Uno. 8.Monitor Serial Output (Optional):

If your program uses Serial communication, you can open the Serial Monitor by clicking "Tools"

-> "Serial Monitor."

9. Power Your Arduino Uno (if disconnected):

If your Arduino Uno is not externally powered and you've disconnected it during the programming process, ensure it has a power source before expecting your program to run.

10. Observe Results:

Once the upload is complete, your Arduino Uno will run the program. Observe the results, which might include LEDs blinking, motors moving, or any other actions based on your code.

11.Troubleshooting:

If you encounter any issues, check the error messages in the Arduino IDE. Common issues include incorrect board selection, port selection, or syntax errors in the code.

Aim:

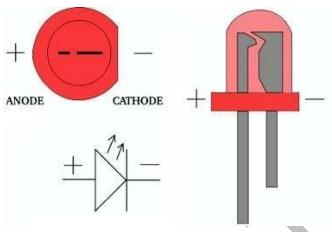
1a. To interface LED/Buzzer with Arduino /Raspberry Pi and write a program to 'turn ON' LED for 1 sec after every 2 seconds.

Hardware Required:

Sl No	Components	Quantity
1.	AURDUINO UNO	1
2.	LED or Buzzer	1
3.	USB cable for AURDUINO UNO	1
4.	Connecting wires	--
5.	Bread Board	1

Theory:

LED Stands for Light Emitting Diode is a semiconductor device that emits light when an electric current passes through it. LEDs are widely used for various applications due to their energy efficiency, long lifespan, and versatility. Here are some key points about LEDs. Basic Operation: LEDs work on the principle of electroluminescence. When electrons and holes (positive counterparts of electrons) recombine within the semiconductor material, they release energy in the form of photons, which produces light. Pin Connection Arduino UNO LED GND Cathode D13 Anode Item Mi n Max Unit Forward Current 20 30 mA Forward Voltage 1.8 2.2 V

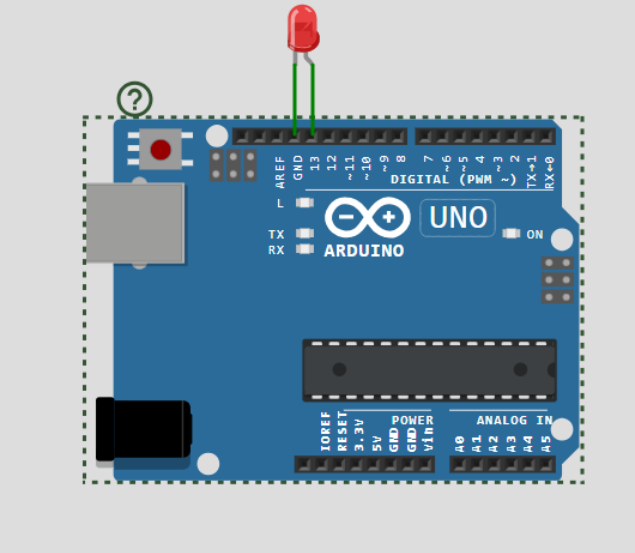


Item	Min	Max	Unit
Forward Current	20	30	mA
Forward Voltage	1.8	2.2	V

Code:

```
void setup()
{
// initialize digital pin LED_BUILTIN as an output.
pinMode(13, OUTPUT);
}
// the loop function runs over and over again forever
void loop()
{
digitalWrite(13, HIGH);//turn the LED on
delay(1000); // wait for a second (1 sec = 1000ms)
digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
delay(2000); // wait for a second (2 sec = 2000ms)
}
```

Circuit Diagram:



Pin Connection	
Arduino UNO	**LED**
GND	Cathode
D13	Anode

Procedure:

1. **Connect the Arduino Uno:**

- Connect the Arduino Uno to your computer using a USB cable (USB A to USB B).
- The green power LED (PWR) on the Arduino Uno should light up.
- Connect the LED to pin 13 and ground

2. **Open the Arduino IDE:**

- Launch the Arduino IDE application on your computer.

3. **Configure the Board and Port:**

- Go to Tools > Board.
- Select "Arduino Uno" (or the specific board version you have).
- Go to Tools > Port.
- Select the COM port associated with your Arduino Uno (e.g., COM3).

4. **Load and Upload a Sketch:**

- Open a sketch (program) in the Arduino IDE
- Click the "Upload" button (or press Ctrl + U).
- The IDE will compile the code and upload it to the Arduino Uno.
- Observ the RX and TX LEDs on the Arduino Uno flashing during the upload process.

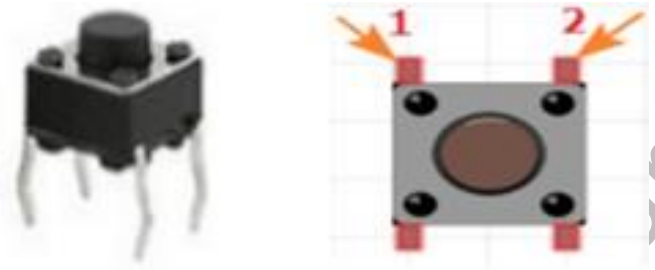
RESULT: LED is successfully turned on for 2 seconds using Arduino microcontroller Board.

1b. To interface the Push button/Digital sensor (IR/LDR) with Arduino /Raspberry Pi and write a program to 'turn ON' LED when a push button is pressed or at sensor detection.

Introduction:

Push-button is a very simple mechanism which is used to control electronic signal either by blocking it or allowing it to pass. This happens when mechanical pressure is applied to connect two points of the switch together. Push buttons or switches connect two points in a circuit when pressed. When the push-button is released, there is no connection between the two legs of the push-button. Here it turns on the built-in LED on pin 11 when the button is pressed. The LED stays ON as long as the button is being pressed.

Push Button



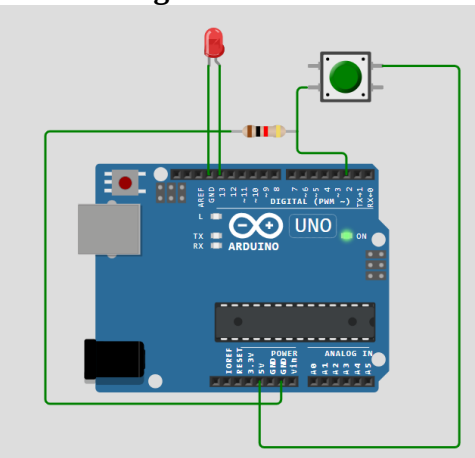
Specifications:

Size	6 x 6 x 5mm
Temperature	-30 ~ +70 Centigrade

Hardware Required:

Sl No	Components	Quantity
1.	AURDUINO UNO	1
2.	LED or Buzzer	1
3.	USB cable for AURDUINO UNO	1
4.	Connecting wires	--
5.	Bread Board	1

Circuit Diagram:



Code:

```
// Define constants for pin numbers
const int pushPin = 2;// Pin number for the push button
const int ledPin = 13;// Pin number for the LED

// Variable to hold the state of the button
int buttonState = 0;// Initial state of the button (LOW)

// The setup function runs once when you press reset or power the board
void setup() {
  pinMode(ledPin, OUTPUT);// Set the LED pin as an output
  pinMode(pushPin, INPUT);// Set the push button pin as an input
}

// The loop function runs over and over again forever
void loop() {
  // Read the state of the push button
    buttonState = digitalRead(pushPin);

  // Check if the button is pressed (HIGH)
  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);// Turn on the LED
  }
  else {
    digitalWrite(ledPin, LOW);// Turn off the LED
  }
}
```

Steps of working

1. **Connect the Arduino Uno:**

   Insert the push button into your breadboard and connect it to the digital pin 7(D7) which act as INPUT.

   Insert the LED into the breadboard. Attach the positive leg (the longer leg) to

digital pin 11 of the Arduino Uno, and the negative leg via the 220-ohm resistor to GND. The pin D11 is taken as OUTPUT.

The 10kΩ resistor used as PULL-UP resistor and 220 Ω resistors is used to limit the current through the LED.

2. Open the Arduino IDE:

Launch the Arduino IDE application on your computer.

3. **Configure the Board and Port:**

a. Go to Tools > Board.

b. Select "Arduino Uno" (or the specific board version you have).

c. Go to Tools > Port.

d. Select the COM port associated with your Arduino Uno (e.g., COM3).

4. **Load and Upload a Sketch:**

a. Open a sketch (program) in the Arduino IDE

b. Click the "Upload" button (or press Ctrl + U).

c. The IDE will compile the code and upload it to the Arduino Uno.

5. The initial state of the button is set to OFF.

6. After that the run a loop that continually reads the state from the pushbutton and sends that value as voltage to the LED. The LED will be ON accordingly.

2(a)To interface the DHT11 sensor with Arduino /Raspberry Pi and write a program to print temperature and humidity readings.

Component Required:

Sl No	Components	Quantity
1	ARDUINO UNO	1
2	DHT11 Temperature sensor	1
3	USB cable A to B	1
4	Connecting wires	--

Theory

The **DHT11 sensor** is a **low-cost** digital temperature and humidity sensor. It operates ata **voltage of 3.3V to 5V** and can measure temperatures rangingfrom **0°C to 50°C** with an accuracy of ±2°C. Additionally, it can measure relative humidity ranging from **20% to 90%** with an accuracy of ±5% .

The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measure, process this changed resistance values and change them into digital form. For measuring temperature this sensor uses a Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with increase in temperature.

DHT11 Specifications

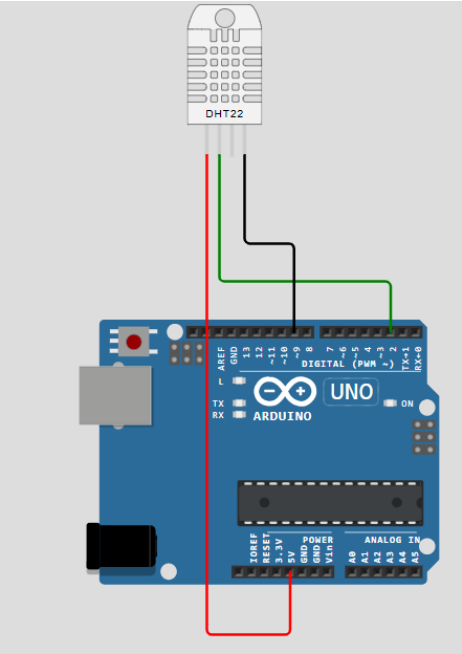
- Operating Voltage: 3.5V to 5.5V
- Operating current: 0.3mA (measuring) 60uA (standby)
- Output: Serial data
- Temperature Range: 0°C to 50°C
- Humidity Range: 20% to 90%
- Resolution: Temperature and Humidity both are 16-bit
- Accuracy: ±1°C and ±1%

Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc. There are thousands of libraries available for download directly through the Arduino IDE, and you can find all of them listed at the Arduino Library

Reference.

Steps to Add DHT11 Sensor Library

1) Open your Arduino IDE and go to **Sketch** > **Include Library** > **Manage Libraries**. The Library Manager should open.
2) Search DHT then find the DHT Sensor library by Adafruit
3) Click install button to install library
4) If ask click on install all button to install library dependencies



ARDUINO UNO	**DHT11**
GND	GND
5v	VCC
D2	DATA

Program

```
//Libraries
#include <DHT.h>

//Constants
#define DHTPIN 2     // what pin we're connected to
#define DHTTYPE DHT11   // DHT 11  (AM2302)
// Initialize DHT sensor for normal 16mhz Arduino
DHT dht(DHTPIN, DHTTYPE);

//Variables
int led=13;
int chk;
float hum;  //Stores humidity value
float temp; //Stores temperature value

void setup()
{
```

```
  Serial.begin(9600);
 dht.begin();
 pinMode(led, OUTPUT);

}

void loop()
{
  //Read data and store it to variables hum and temp
  hum = dht.readHumidity();
  temp= dht.readTemperature();
  //Print temp and humidity values to serial monitor
  Serial.print("Humidity: ");
  Serial.print(hum);
  Serial.print(" %, Temp: ");
  Serial.print(temp);
  Serial.println(" Celsius");
  digitalWrite(led, LOW);
  if (temp>2)
  {
    digitalWrite(led, HIGH);
    Serial.print("");
    Serial.print(" HOT CLIMATE");
  }

  delay(2000); //Delay 2 sec.
}
```

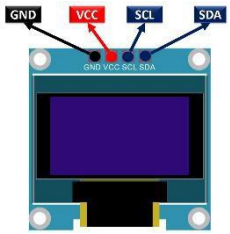
2 (b) To interface OLED with Arduino /Raspberry Pi and write a program to print its temperature and humidity readings.

Component Required:

Sl No	Components	Quanti ty
1.	AURDUINO UNO	1
2.	LED or Buzzer	1
3.	USB cable for AURDUINO UNO	1
4.	DHT11 Sensor	1
5.	OLED	1
6.	Connecting wires	--
7.	Bread Board	1
8.	AURDUINO UNO	1

Theory

The SSD1306 OLED I2C 128X64 OLED Display module is a small monochrome rganic light- emitting diode (OLED) display that is controlled through an I2C interface. It has a display resolution of 128×64 pixels, and the SSD1306 is the controller chip that manages the display. It‘s commonly used for display purposes in various electronics projects and is compact low power, and easily readable in low light conditions.



Specification of OLED	
Size	0.96 inch
Terminals	4
Pixels or Resolution	128×64
Communication	I2C only
VCC	3.3V-5V

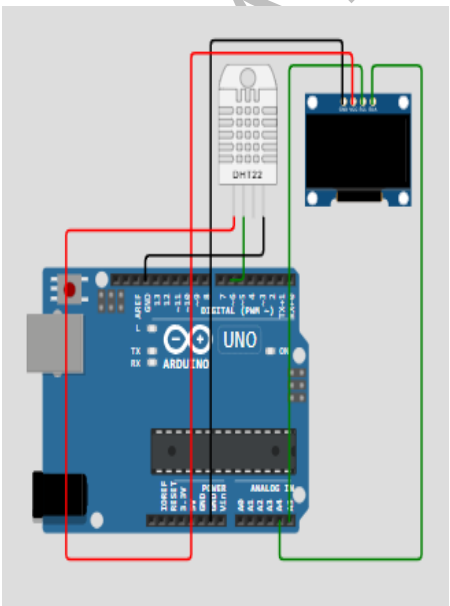
To control the OLED display libraries. Follow the next instructions to install those libraries.

1. Open Arduino IDE & navigate to the **Sketch > Include Library > Manage Libraries**

   Search by typing _**SD1306**‘.

2. Look and Search for **ESP8266 and ESP32 OLED driver for SSD1306 displays**. Click on that entry, and then select Install.

3.After successfully installing the library Add library to program by typing: #include "SSD1306Wire.h‖ and #include <Wire.h>



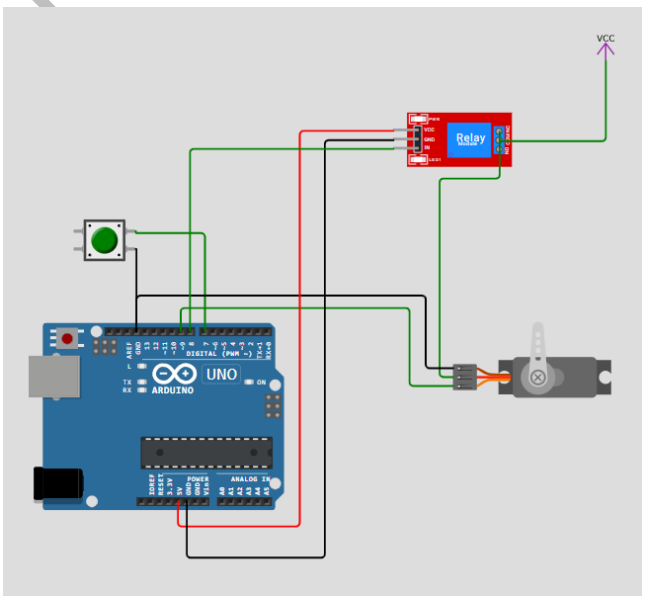
```
#include <Wire.h> // Include the Wire library for I2C communication
#include <Adafruit_GFX.h> // Include the Adafruit Graphics library for display functions
#include <Adafruit_SSD1306.h> // Include the Adafruit SSD1306 library for OLED display control
#include "DHT.h" // Include the DHT library for temperature and humidity sensor

#define DHTPIN 6 // Define the pin where the DHT sensor is connected
#define DHTTYPE DHT22 // Define the type of DHT sensor being used (DHT22)
#define OLED_RESET -1 // Define OLED reset pin (-1 indicates no reset pin)
#define SCREEN_ADDRESS 0x3C // Define the I2C address for the OLED display
#define SCREEN_WIDTH 128 // Define the width of the OLED display in pixels
#define SCREEN_HEIGHT 64 // Define the height of the OLED display in pixels

// Create an instance of the SSD1306 display with specified width, height, and I2C connection
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
// Create an instance of the DHT sensor with specified pin and type
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  dht.begin(); // Initialize the DHT sensor
  oled.begin(SSD1306_EXTERNALVCC, SCREEN_ADDRESS); // Initialize the OLED display with
external VCC

  delay(2000); // Wait for 2 seconds to allow setup to complete

  oled.clearDisplay(); // Clear any previous display content
  oled.setTextSize(1); // Set text size for display
  oled.setTextColor(WHITE); // Set text color to white
  oled.setCursor(10, 20); // Set cursor position on display
  oled.println("TEMP and HUMIDITY"); // Display title
  oled.setCursor(10, 30); // Move cursor down for subtitle
  oled.println("MONITORING DEVICE"); // Display subtitle
  oled.display(); // Update the display to show current content
  delay(2000); // Wait for another 2 seconds before entering loop
}

void loop() {
  float temperature = dht.readTemperature(); // Read temperature from DHT sensor
  float humidity = dht.readHumidity(); // Read humidity from DHT sensor

  oled.clearDisplay(); // Clear previous readings from display
  oled.setTextSize(1.8); // Set larger text size for readings
  oled.setTextColor(WHITE); // Set text color to white again
  oled.setCursor(0, 0); // Reset cursor position to top left of display
  oled.print("Temperature = "); // Print temperature label
  oled.println(temperature); // Print temperature value
  oled.print("Humidity = "); // Print humidity label
  oled.println(humidity); // Print humidity value
  oled.display(); // Update the display with new readings
  delay(2000); // Wait for 2 seconds before next loop iteration}
```

3. To interface the motor using a relay with Arduino /Raspberry Pi and write a program to 'turn ON' the motor when a push button is pressed.

Sl No	Components	Quantity
1	AURDUINO UNO	1
2	Relay 2 Channel	1
3	USB cable for NODE MCU	1
4	Connecting wires	--
5	Push Button	1
6	Power Supply	1
7	DC motor	1

Theory:

Servo motors are high torque motors which are commonly used in robotics and several other applications due to the fact that it's easy to control their rotation. Servo motors have a geared output shaft which can be electrically controlled to turn one (1) degree at a time. For the sake of control, unlike normal DC motors, servo motors usually have an additional pin besides the two power pins (Vcc and GND) which is the signal pin. The signal pin is used to control the servo motor, turning its shaft to any desired angle.



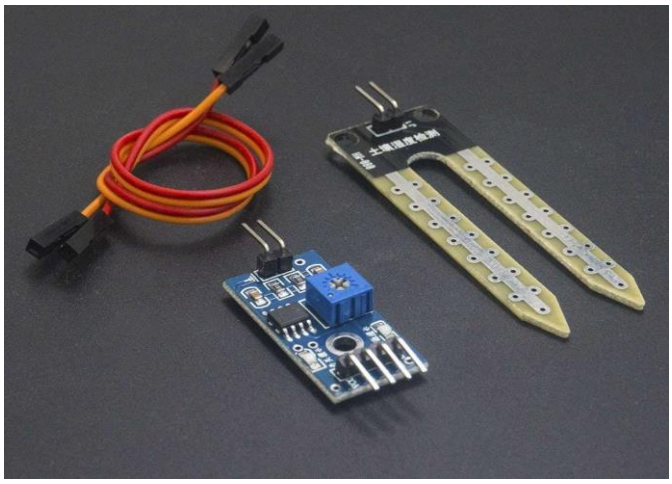
Program

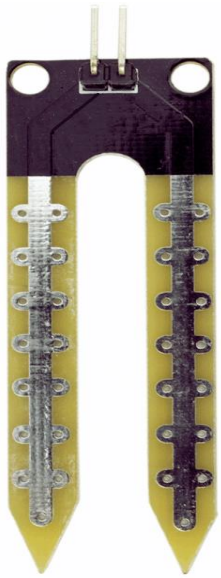
```
// Include the Servo library for servo motor control
#include <Servo.h>

// Define the pin for the button input
int buttonpin = 7;

// Store the last state of the button (HIGH or LOW)
int last_state = HIGH;

// Define the pin for the relay output
int relaypin = 8;

// Create a Servo object
Servo myservo;

// Initialize the servo position variable
int pos = 0;

void setup() {
  // Initialize serial communication at a baud rate of 115200
  Serial.begin(115200);

  // Set the button pin as an input with internal pull-up resistor
  pinMode(buttonpin, INPUT_PULLUP);

  // Set the relay pin as an output
  pinMode(relaypin, OUTPUT);

  // Attach the servo to pin 9
  myservo.attach(9);
}

void loop() {
  // Read the current state of the button
  int value = digitalRead(buttonpin);

  // Check if the button state has changed
  if (last_state != value) {
    // Update the last state
    last_state = value;

    // If the button is released (HIGH state due to pull-up resistor)
    if (value == HIGH) {
      // Turn off the relay
      digitalWrite(relaypin, LOW);
      Serial.println("released");
```

```
    // Move the servo from 0 to 180 degrees
    for (pos = 0; pos <= 180; pos += 1) {
      // Set the servo position
      myservo.write(pos);
      // Wait for 15 milliseconds before moving to the next position
      delay(15);
    }
  } else {
    // If the button is pressed (LOW state)
    // Turn on the relay
    digitalWrite(relaypin, HIGH);
    Serial.println("pressed");
  }
 }
}
```

4 (a) Write an Arduino/Raspberry Pi program to interface the Soil Moisture Sensor.

Overview of Soil Moisture



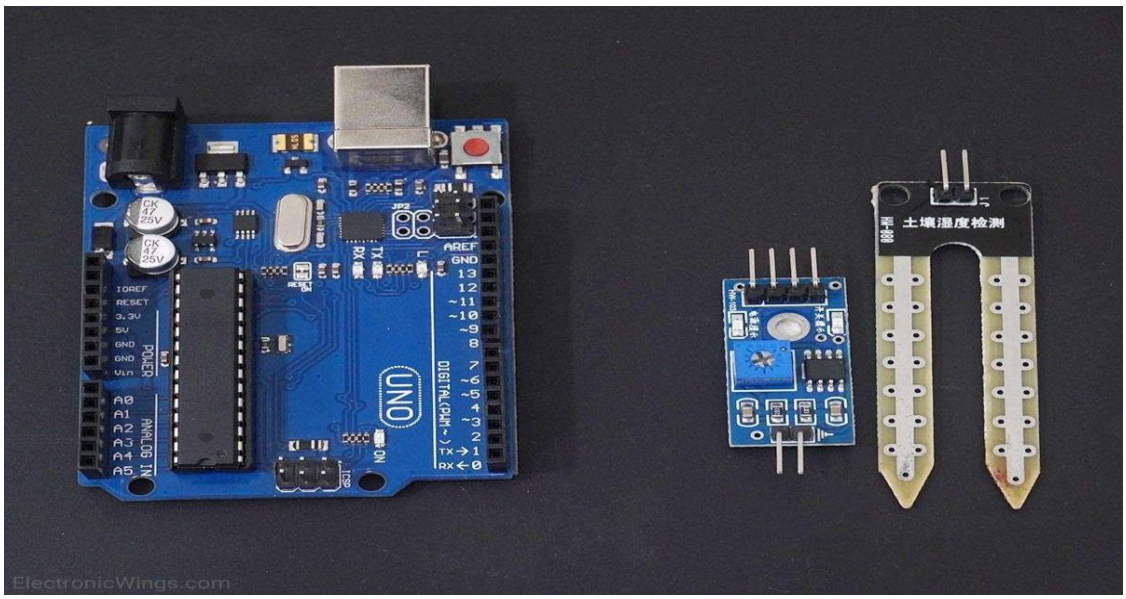


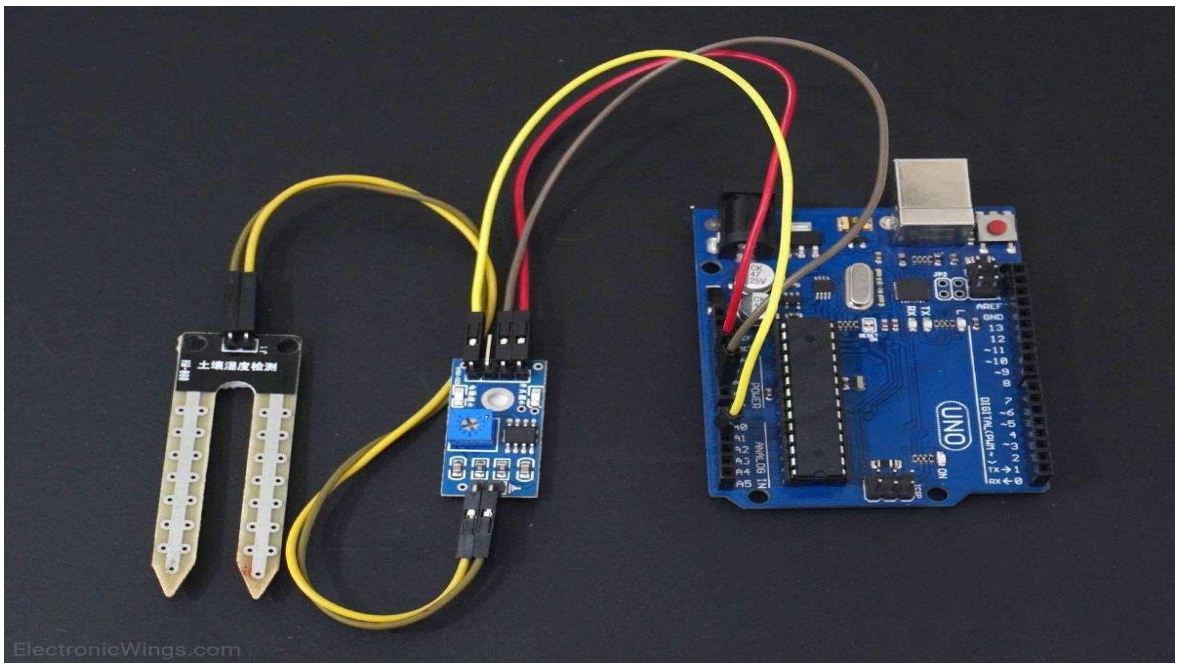
Soil Moisture Sensor

Soil moisture is basically the content of water present in the soil. This can be measured using a soil moisture sensor which consists of two conducting probes that act as a probe. It can measure the moisture content in the soil based on the change in resistance between the two conducting plates.

The resistance between the two conducting plates varies in an inverse manner with the amount of moisture present in the soil.

Interfacing Soil Moisture Sensor With Arduino UNO





Measure soil moisture using Arduino Uno

Here, the analog output of soil moisture sensor is processed using ADC. The moisture content in terms of percentage is displayed on the serial monitor.

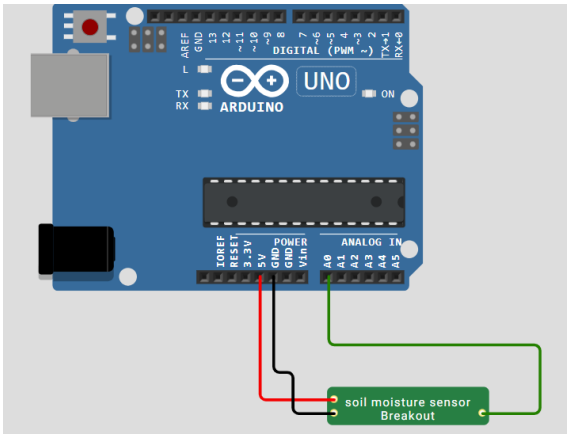
The output of the soil moisture sensor changes in the range of ADC value from 0 to 1023.

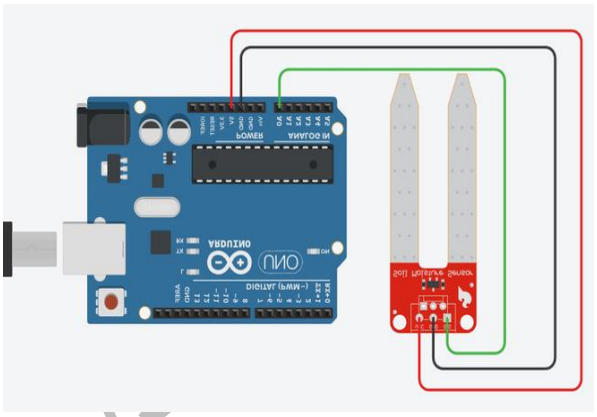
This can be represented as moisture value in terms of percentage using formula given below.

$$AnalogOutput = \frac{ADCValue}{1023}$$

Moisture in percentage = 100 – (Analog output * 100)

For zero moisture, we get maximum value of 10-bit ADC, i.e. 1023. This, in turn, gives 0% moisture.





{

"name": "soil moisture sensor",

"

```
void setup()
{
  // Set the serial monitor baudrate to 9600
  Serial.begin(9600);
}

void loop()
{
  // Variable to store ADC value ( 0 to 1023 )
  int level;
  // analogRead function returns the integer 10 bit integer (0 to 1023)
  level = analogRead(0);

  // Print text in serial monitor
  Serial.println("Analog value:");
  // Print analog value in serial monitor
  Serial.println(level);
}
```

4 (b) Write an Arduino/Raspberry Pi program to interface the LDR/Photo Sensor.

LDR SENSOR is a light-dependent resistor, its resistance changes according to changes in light intensity. A light-dependent resister is also known as LDR SENSOR, which is used to detect

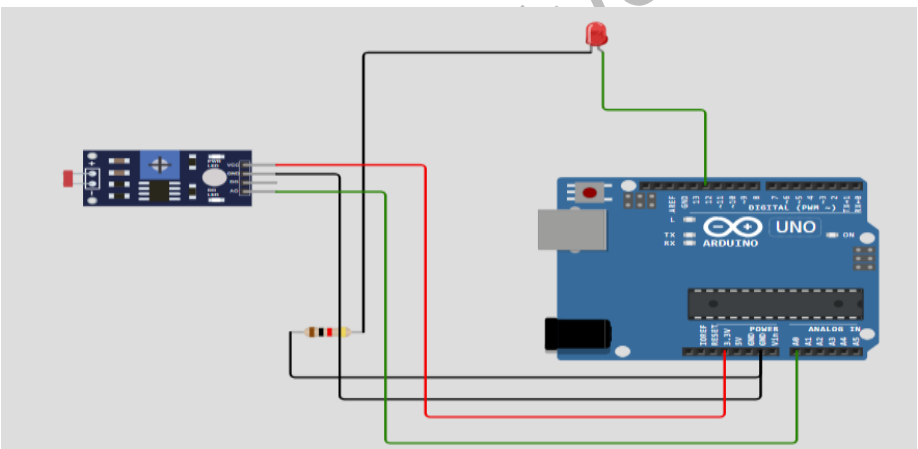
the intensity of light. In this blog we will discuss what LDR SENSOR is and how it can be used with the Arduino Development Board.

stud lights are LED that are mounted on the road and used as indicators. These indicators only turn on during the night and are off during the day And the reason behind this was LDR SENSOR.

Those stud lights have LDR SENSOR, battery, controller unit and solar panel inbuilt. In the daytime, when there will be sufficient sunlight, the controller unit receives input from the LDR sensor and according to the received input the control unit either charges the battery or turns on the light.

Component Required:

Sl No	Components	Quantity
1	Arduino Uno	1
2	Bluetooth Module HC-05	1
3	USB cable A to B	1
4	Connecting wires	--
5	LED	1
6	Smart Phone with Arduino Bluetooth Controller app	1



LDR interfacing With Arduino

LDR Working Principle

LDR SENSOR is nothing but a light-dependent resistor, its resistance changes according to changes in light intensity.

The LDR SENSOR is made of photosensitive material. The zig-zag lines you see on the sensor are nothing but a photosensitive material.

When light falls on this material the resistance of the material changes and hence conductivity. This was the original operation of LDR SENSOR

Connections

the LDR sensor is only a resistor and can be directly connected to any GPIO pin. So for this application we are connecting this LDR SENSOR to A0 pin of Arduino and connecting the second pin of LDR SENSOR to the 5v pin of Arduino.

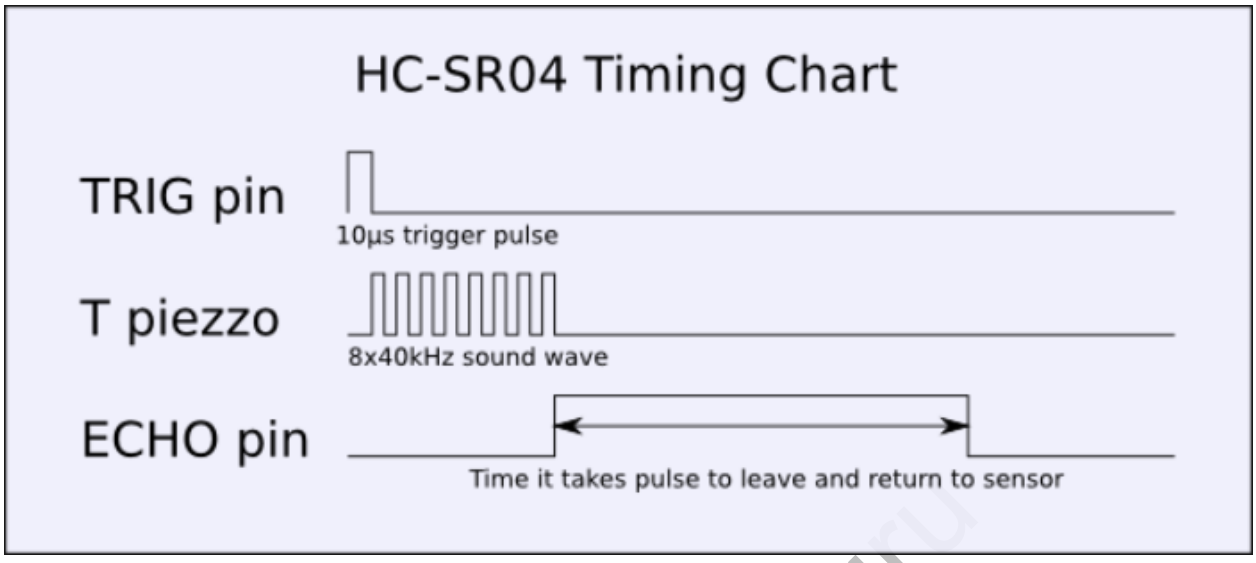
```
// Define constants for the LED pin and LDR pin
const int ledPin = 12;  // Pin for the LED
const int ldrPin = A0;  // Pin for the Light Dependent Resistor (LDR)

// Define the threshold value for determining brightness
int threshold = 600;  // Value below which it is considered dark

void setup() {
  // Initialize serial communication at a baud rate of 9600
  Serial.begin(9600);

  // Set the LED pin as an output
  pinMode(ledPin, OUTPUT);

  // Set the LDR pin as an input
  pinMode(ldrPin, INPUT);
}

void loop() {
  // Read the analog value from the LDR
  int ldrStatus = analogRead(ldrPin);

  // Check if it's dark based on the threshold value
  if (ldrStatus <= threshold) {
    // Turn on the LED
    digitalWrite(ledPin, HIGH);

    // Print a message indicating it's dark and the LED is on
    Serial.print("It's dark, turn on LED: ");
    Serial.println(ldrStatus);

  } else {
    // Turn off the LED
    digitalWrite(ledPin, LOW);

    // Print a message indicating it's bright and the LED is off
    Serial.print("It's bright, turn off LED: ");
    Serial.println(ldrStatus);
  }
}
```

5. Write a program to interface an Ultrasonic Sensor with Arduino /Raspberry Pi.

Working of HC-SR04

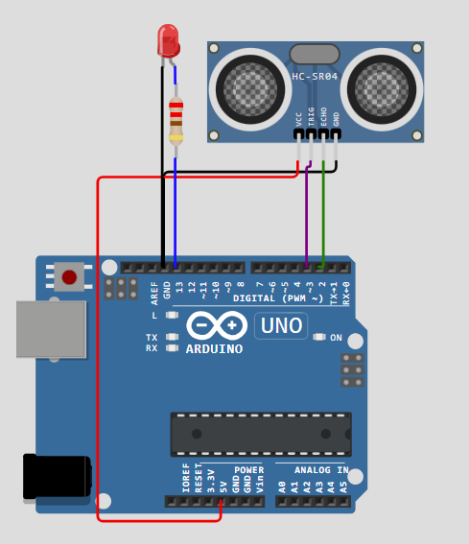
The HC-SR04 emits ultrasonic waves at 40,000 Hz. In order to make it emit waves, we need to give a 10 microseconds HIGH pulse at the Trigger pin. The module responds by emitting a sonic burst of 8 pulses. This 8-pulse pattern helps differentiate the pulses emitted by the module from the ambient noise. As soon as the pulses are transmitted, the ECHO pin goes HIGH, and

stays HIGH till all the reflected pulses are received. The module times out after 38ms, if all the reflected pulses are not received in this duration.



The timing diagram explains the behavior of the module –

The time for which the Echo pin remains HIGH can help determine the distance of the sensor from the reflecting surface. The speed of sound in air is 340 m/s, or 0.034 cm/ microsecond. If the ECHO pin stays HIGH for, say 100 microseconds, then the distance travelled by the waves is: 100*0.034 = 3.4cm. Therefore, the distance from the surface is 3.4/2 = 1.7 cm (since the waves reflect back from the surface and cover the same distance again)




```
// Define pins for the ultrasonic sensor
#define ECHO_PIN 2  // Pin connected to the echo output of the sensor
#define TRIG_PIN 3  // Pin connected to the trigger input of the sensor
```

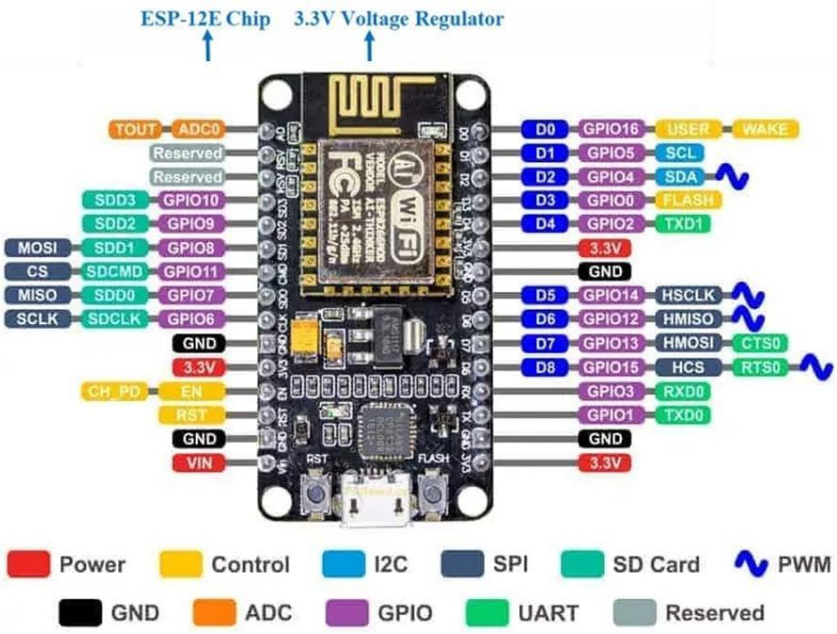
```
void setup() {
  // Initialize serial communication at a baud rate of 115200
  Serial.begin(115200);

  // Set the built-in LED pin as an output
  pinMode(LED_BUILTIN, OUTPUT);

  // Set the trigger pin as an output and the echo pin as an input
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}

// Function to read the distance in centimeters using the ultrasonic sensor
float readDistanceCM() {
  // Set the trigger pin low for a short duration to ensure a clean start
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);

  // Send a 10 microsecond pulse to the trigger pin to initiate measurement
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Measure the time it takes for the echo signal to return
  int duration = pulseIn(ECHO_PIN, HIGH);

  // Calculate the distance using the speed of sound (approximately 0.034 cm/us)
  return duration * 0.034 / 2; // Divide by 2 because the sound travels to the object and back
}

void loop() {
  // Read the current distance from the sensor
  float distance = readDistanceCM();

  // Determine if an object is nearby (within 100 cm)
  bool isNearby = distance < 100;

  // Turn the LED on if an object is nearby, otherwise turn it off
  digitalWrite(LED_BUILTIN, isNearby);

  // Print the measured distance to the serial console
  Serial.print("Measured distance: ");
  Serial.println(readDistanceCM()); // Note: This reads the distance again, consider storing the first reading

  // Wait for 100 milliseconds
```

Introduction to Node MCU ESP8266 module

The NodeMCU (*N*ode *M*icro-Controller *U*nit) is an open-source software and hardware development environment built around an inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains the crucial elements of a computer: CPU, RAM, networking (WiFi), and even a modern operating system and SDK. That makes it an excellent choice for Internet of Things (IoT) projects of all kinds.

NodeMCU ESP8266 Specifications & Features and Pin out

- Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106
- Operating Voltage: 3.3V
- Input Voltage: 7-12V
- Digital I/O Pins (DIO): 16
- Analog Input Pins (ADC): 1
- UARTs: 1
- SPIs: 1
- I2Cs: 1
- Flash Memory: 4 MB
- SRAM: 64 KB
- Clock Speed: 80 MHz
- USB-TTL based on CP2102 is included onboard, Enabling Plug n Play
- PCB Antenna
- Small Sized module to fit smartly inside your IoT projects



PIN	CODE
A0	A0
GPIO 16	D0
GPIO 5	D1
GPIO 4	D2
GPIO 0	D3
GPIO 2	D4
GPIO14	D5
GPIO 12	D6
GPIO 13	D7
GPIO 15	D8
GPIO 9	SD2
GPIO10	SD3
GPIO3	Rx
GPIO1	Tx

Steps to install Node MCU

1. Download and install Arduino IDE
2. Open the IDE and follow this path. **File -> preferences -> Additional board manager URL.**
3. Now paste the URL in the dialog box **:**

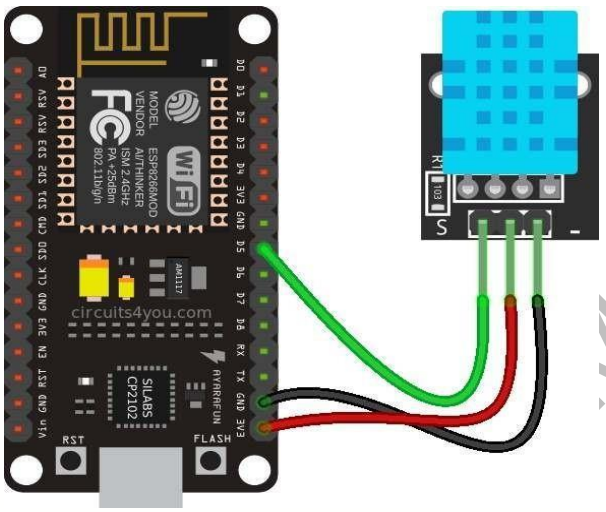
Pin Connection details	
NODE MCU	**DHT11**
GND	GND
3V3	VCC
D5/D4	DATA

http://arduino.esp8266.com/stable/package_esp8266com_index.json

4. Then, click the "**OK**" button.
5. Now follow this path**. Tools -> Board -> Boards Manager**
6. Search for **ESP8266** and install the "**ESP8266 by ESP8266 Community**"
7. After this, restart your Arduino IDE.
8. Then, go to **Tools** > **Board** and check that you have ESP8266 boards available.

First, make sure you have an ESP8266 selected in **Tools** > **Board**. If you're using the ESP8266-12E NodeMCU Kit as shown in previous pictures, select the **NodeMCU 1.0 (ESP-12E Module)** option.

6 Write a program on Arduino/Raspberry Pi to upload temperature and humidity data to thingspeak cloud.



Theory:

ThingSpeak is an open-source Internet of Things (IoT) application and API that allows users to collect and store sensor data in the cloud and perform analytics on that data. It allows users to create "channels" to collect data from multiple sensors, and also has built-in support for visualizing and analyzing the data. ThingSpeak can be used for a variety of applications, such as monitoring environmental conditions, tracking the location of assets, and controlling devices remotely. It is available for free and also has paid subscription plans for additional features and support. The device that sends the data must be configured with the correct channel information, such as the channel ID and write API key.

- ThingSpeak is a platform providing various services exclusively targeted for building IoT applications.
- It offers the capabilities of real-time data collection, visualizing the collected data in the form of charts, ability to create plugins and apps for collaborating with web services, social network and other APIs.

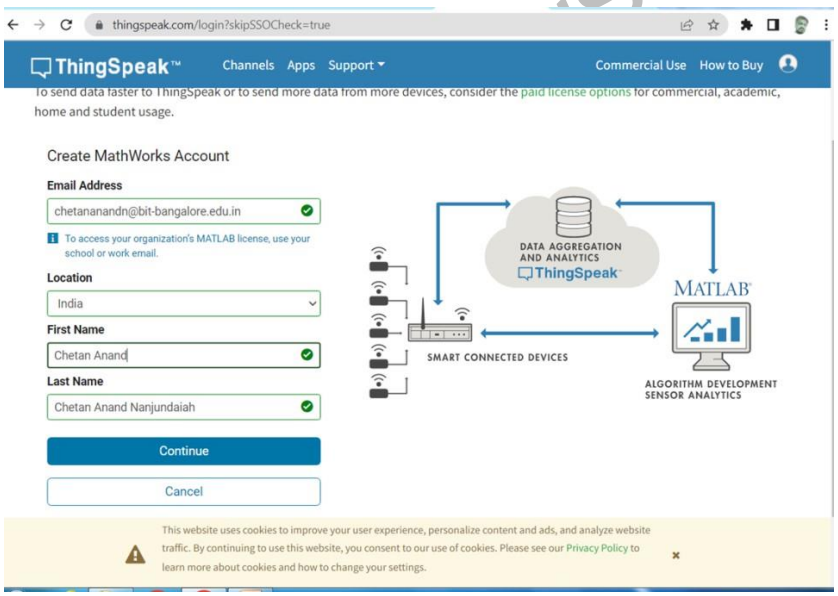
The core element of ThingSpeak is a 'ThingSpeak Channel'.
A channel stores the data that we send to ThingSpeak and comprises of the below elements:

8 fields for storing data of any type - These can be used to store the data from a sensor or from an embedded device.
3 location fields - Can be used to store the latitude, longitude and the elevation. These are very useful for tracking a moving device.
1 status field - A short message to describe the data stored in the channel.

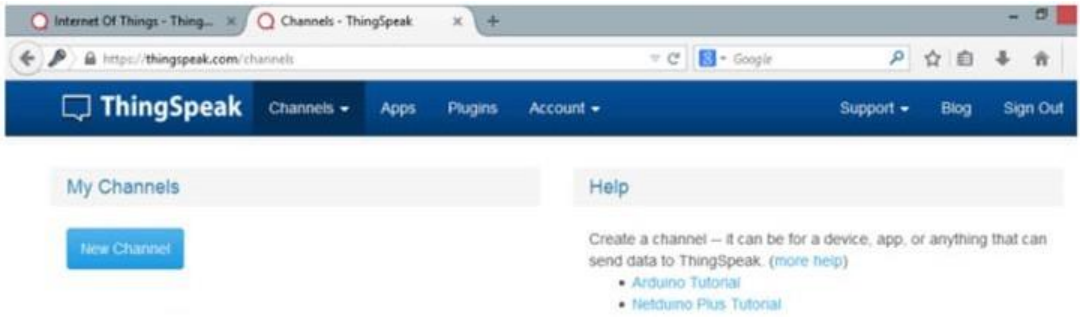
- To use ThingSpeak, we need to sign up and create a channel.
- Once we have a channel, we can send the data, allow ThingSpeak to process it and also retrieve the same.

Creating a ThingSpeak Channel

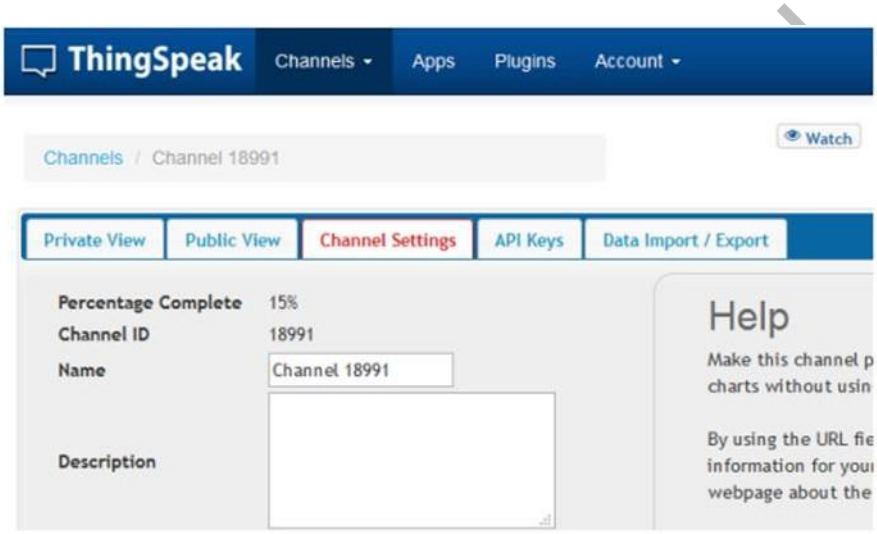
Step 1: Open https://thingspeak.com/ and click on the 'Get Started Now' button on the center of the page and you will be redirected to the sign-up page(you will reach the same page when you click the 'Sign Up' button on the extreme right).
Fill out the required details and click on the 'Create Account' button



Now you should see a page with a confirmation that the account was successfully created. The confirmation message disappears after a few seconds and the final page should look as in the below screen:

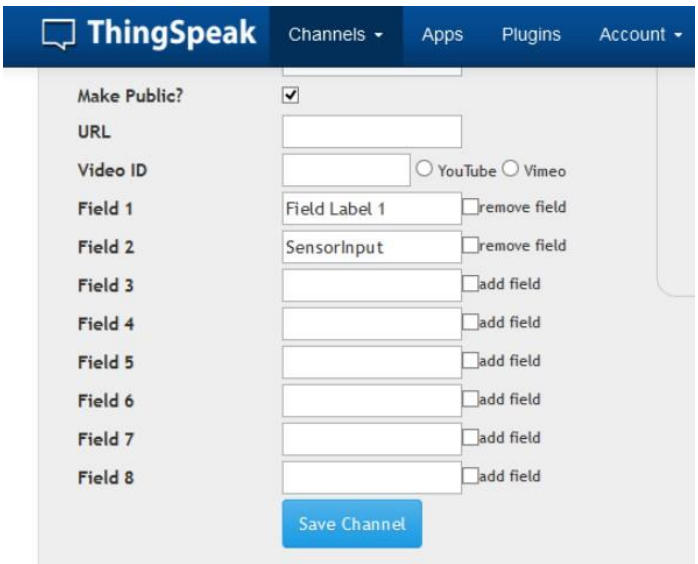


Step 2: Go ahead and click on 'New Channel'. You should see a page like the below:



Change the name to fit your need

Add a description corresponding to the channel



Fields 1 to 8 - These are the fields which correspond to the data sent by a sensor or a 'thing'. A field has to be added before it can be used to store data. By default, Field 1 is added.Once you have edited the fields, click on '**Save Channel**' button.

Latitude, longitude and elevation:

These fields correspond to the location of a 'thing' and are especially significant for moving things.

Make Public?

- If the channel is made public, anyone can view the channel's data feed and the corresponding charts. If this check box is not checked, the channel is private, which means for every read or write operation, the user has to pass a corresponding API key.

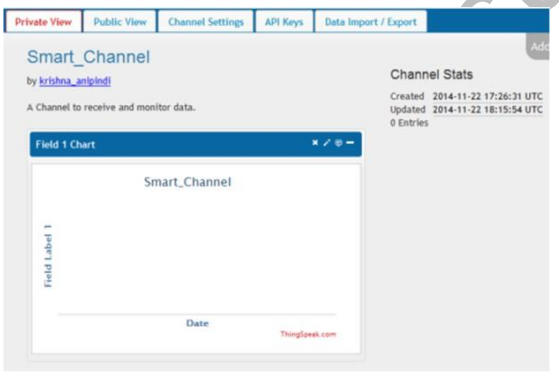
URL :

This can be the URL of your blog or website and if specified, will appear on the public view of the channel

Video ID:

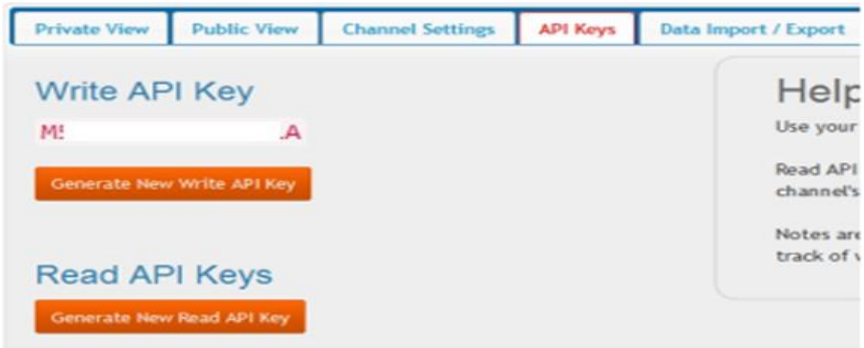
This is the ID corresponding to your YouTube or Vimeo ID. If specified, the video appears on the public view of the channel.

Step 3: 'Private View' tab is defaulted:



The Private View shows a chart corresponding to each of the fields that we have added. Now click on the 'Public View' tab. This should look exactly similar to the what we see in the 'Private View' tab since our channel is public.

Step 4: click on the 'API Keys' tab



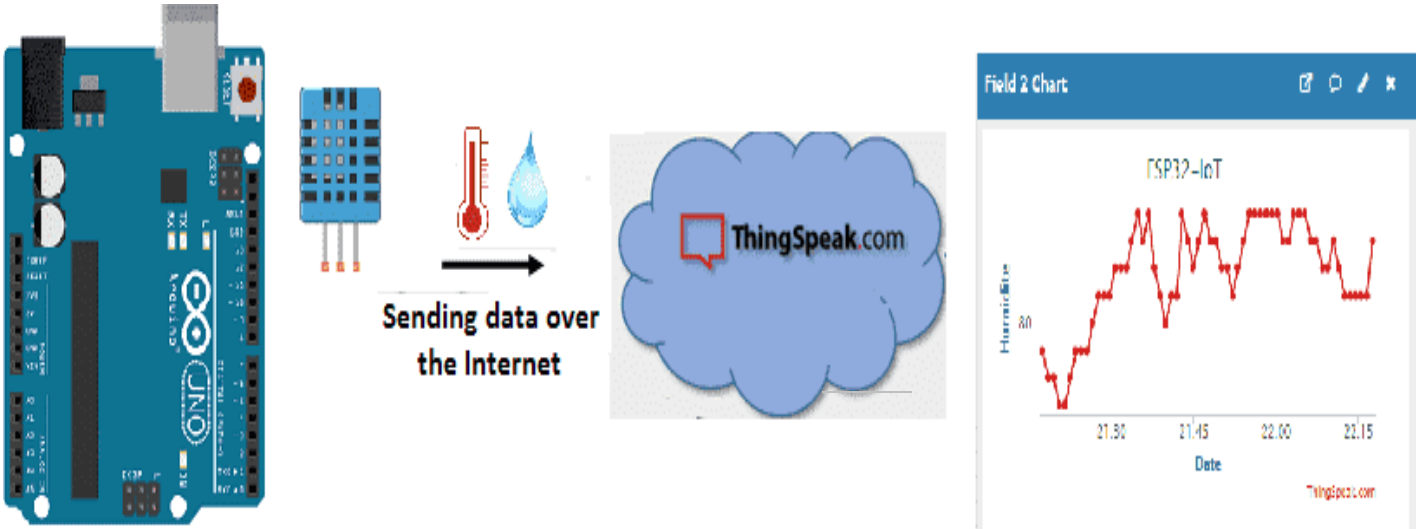
The write API key is used for sending data to the channel
The read API key(s) is used to read the channel data

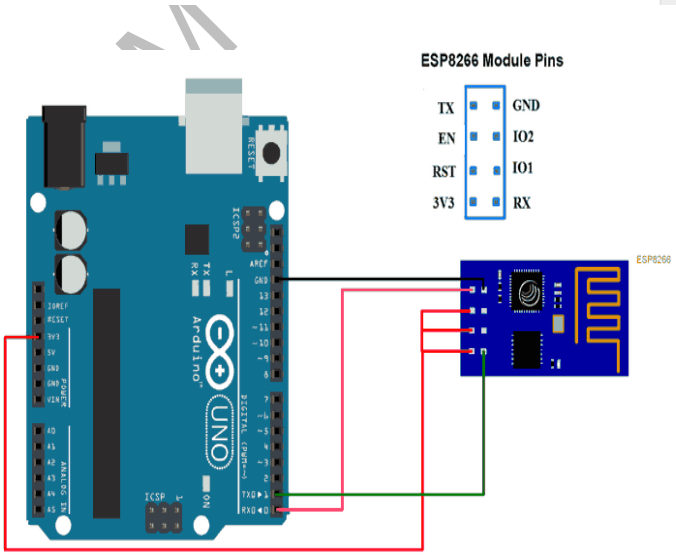
Share the Read API keys with people who are approved and authorized to view your channel.

Step 5: Installing the ThingSpeak

Library

To send or receive sensor readings to ThingSpeak, we'll use the ThingSpeak Arduino library. Go to Sketch > Include Library > Manage Libraries… and search for "ThingSpeak in Library Manager. Install the ThingSpeak library by **MathWorks**






```
// Include the ESP8266 WiFi library
#include <ESP8266WiFi.h>

// Include the ThingSpeak library for sending data to the ThingSpeak server
```

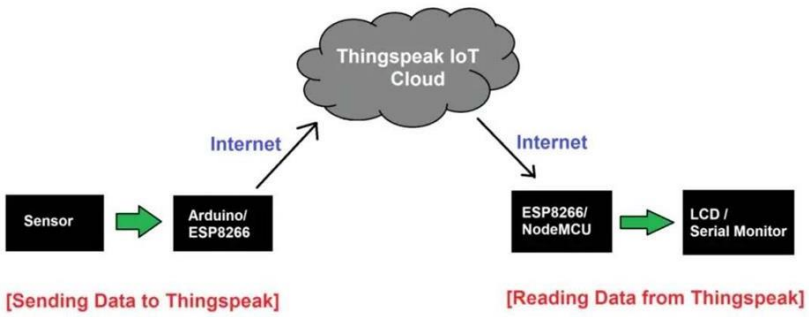
```
#include <ThingSpeak.h>

//Include the DHT sensor library
#include <DHT.h>

// Define the GPIO pin connected to the DHT11 sensor
#define DHTPIN 4

// Define the type of DHT sensor being used
#define DHTTYPE DHT11

// Create a DHT object named 'dht'
DHT dht(DHTPIN, DHTTYPE);

// Variables to hold sensor readings
int chk;
float hum;    // Humidity
float temp;   // Temperature
// WiFi client used by ThingSpeak
WiFiClient client;

// Your ThingSpeak channel number and API key
long myChannelNumber = 2887065;
const char myWriteAPIKey[] = "7GVIQLJ2TG2UARNI";

void setup() {
  // Start the serial communication
  Serial.begin(9600);

  // Connect to WiFi with SSID, password, and channel number
  WiFi.begin("TECMCA", "tec&2020mca", 6);

  // Wait until WiFi is connected
  while(WiFi.status() != WL_CONNECTED)
  {
    delay(200);
    Serial.print("..");  // Indicate ongoing connection attempt
  }

  // Once connected, print the IP address
  Serial.println();
  Serial.println("Esp32 is connected!");
  Serial.println(WiFi.localIP());

  // Initialize the DHT sensor
dht.begin();

  // Initialize ThingSpeak with the WiFi client
  ThingSpeak.begin(client);
```

```
}

void loop() {
  // Read humidity and temperature values from the DHT sensor
  hum = dht.readHumidity();
  temp = dht.readTemperature();

  // Print the temperature and humidity to the Serial Monitor
  Serial.println("Temperature: " + (String) temp);
  Serial.println("Humidity: " + (String) hum);

  // Write the temperature value to field 1 of the ThingSpeak channel
  ThingSpeak.writeField(myChannelNumber, 1, temp, myWriteAPIKey);

  // Write the humidity value to field 2 of the ThingSpeak channel
  ThingSpeak.writeField(myChannelNumber, 2, hum, myWriteAPIKey);

  // Wait for 2 seconds before sending the next set of data
  delay(2000);
}
```

7 Write a program on Arduino/Raspberry Pi to retrieve temperature and humidity data from thingspeak cloud

Theory:

To read values from Thingspeak we need to upload some data in real time, to do this, first upload temperature and humidity data to Thingspeak using previous experiment using NodeMCU 8266



Channel Settings for reading data

1. Go to your Thingspeak account and do the following setting to receive temperature and humidity data.
2. Go to channel setting put 'tick' mark for both filed 1 and filed 2 and scroll down to bottom and save it.
3. You need your channel ID to read the fields on your channel you wish to read so that copy your channel id and paste in the code
4. You need your **Read API key** from your channel and copy **Read API key**.
5. Use this **Read API key** in our code.

Write following program and upload in the Node MCU82666

After successful upload Open the serial monitor; you will be able to see the values read from your channel.

```
#include <ThingSpeak.h>      // Library to interact with ThingSpeak
#include <ESP8266WiFi.h>  // Library to connect ESP8266 to WiFi

// WiFi credentials
const char* ssid = "TECMCA";         // WiFi SSID
const char* password = "tec&2020mca"; // WiFi Password

// ThingSpeak channel details
unsigned long CHANNEL_ID = 2887065;          // Channel ID for ThingSpeak
const char* READ_API_KEY = "N1H6LJ392OT31I1V"; // API Key for reading data

// Fields in ThingSpeak channel
const int Field1 = 1; // Field 1 for Temperature
const int Field2 = 2; // Field 2 for Humidity

WiFiClient client; // Create WiFi client object

void setup()
{
  Serial.begin(9600); // Start Serial communication at 9600 baud rate

  WiFi.begin(ssid, password);  // Connect to WiFi network
  WiFi.mode(WIFI_STA);         // Set WiFi mode to Station (client mode)

  ThingSpeak.begin(client);    // Initialize ThingSpeak with WiFi client

  // Wait for WiFi to connect
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Wifi connecting ");
  }

  Serial.println("Wifi connected successfully ");
}

void loop() {
  // --------- Reading Field 1 (Temperature) ---------
  long temp = ThingSpeak.readLongField(CHANNEL_ID, Field1, READ_API_KEY); // Read temperature
value from Field 1
  int statusCode = ThingSpeak.getLastReadStatus(); // Get status of last read operation

  if (statusCode == 200) {
    Serial.print("Temperature: ");
    Serial.println(temp); // Print temperature if read was successful
  } else {
    Serial.println("Unable to read channel / No internet connection"); // Error message
  }
```

```
  delay(100); // Short delay

  // --------- Reading Field 2 (Humidity) ---------
  long humidity = ThingSpeak.readLongField(CHANNEL_ID, Field2, READ_API_KEY); // Read
humidity value from Field 2
  statusCode = ThingSpeak.getLastReadStatus(); // Update status code

  if (statusCode == 200) {
    Serial.print("Humidity: ");
    Serial.println(humidity); // Print humidity if read was successful
  } else {
    Serial.println("Unable to read channel / No internet connection"); // Error message
  }

  delay(100); // Short delay before next loop iteration
}
```

8.Write a program to interface LED using Telegram App.

```
// Include WiFi library for ESP32 WiFi functions
#include <ESP9266WiFi.h>
// Include WiFiClientSecure for secure (HTTPS) connections
#include <WiFiClientSecure.h>
// Include UniversalTelegramBot library to interact with Telegram Bot API
#include <UniversalTelegramBot.h>
// Include ArduinoJson for parsing JSON data from Telegram
#include <ArduinoJson.h>

// Define your Telegram Bot token (keep this private in real projects)
#define BOT_TOKEN "7742829309:AAFQSrRF45CTRv-AYqySdpynJ9TSGRlNByA"
// Define the chat ID to which the bot will respond
#define CHAT_ID "1032626606"
// Define the pin number for the LED
#define LED_PIN 15

// Create a secure WiFi client object
WiFiClientSecure secured_client;
// Create a Telegram bot object using the bot token and secure client
UniversalTelegramBot bot(BOT_TOKEN, secured_client);

// Variables to manage bot polling timing
unsigned long lastTimeBotRan;
const int botRequestDelay = 1000; // Delay between bot requests (in ms)

void setup() {
  Serial.begin(9600); // Start serial communication for debugging
```

```
  pinMode(LED_PIN, OUTPUT); // Set the LED pin as output
  WiFi.begin("", ""); // Connect to WiFi network (SSID, password)

  // Wait until WiFi is connected
  while (WiFi.status() != WL_CONNECTED) {
    delay(250);
  }

  // Set the root certificate for Telegram (required for HTTPS)
  secured_client.setCACert(TELEGRAM_CERTIFICATE_ROOT);
  Serial.println("WiFi connected"); // Print confirmation
}

void loop() {
  // Check if it's time to poll the bot for new messages
  if (millis() - lastTimeBotRan > botRequestDelay) {
    // Get new messages from Telegram
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1);

    // Process all new messages
    while (numNewMessages) {
      for (int i = 0; i < numNewMessages; i++) {
        String chat_id = bot.messages[i].chat_id; // Get sender's chat ID
        String text = bot.messages[i].text;       // Get message text

        // Only respond to messages from the authorized chat ID
        if (chat_id == CHAT_ID) {
          if (text == "/led_on") {
            digitalWrite(LED_PIN, HIGH); // Turn LED on
            bot.sendMessage(chat_id, "LED turned ON", ""); // Confirm action
          } else if (text == "/led_off") {
            digitalWrite(LED_PIN, LOW); // Turn LED off
            bot.sendMessage(chat_id, "LED turned OFF", ""); // Confirm action
          } else if (text == "/state") {
            // Report current LED state
            String state = (digitalRead(LED_PIN) == HIGH) ? "ON" : "OFF";
            bot.sendMessage(chat_id, "LED is " + state, "");
          }
        }
      }
      // Check for more new messages
      numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    }
    lastTimeBotRan = millis(); // Update the last bot polling time
  }
}
```

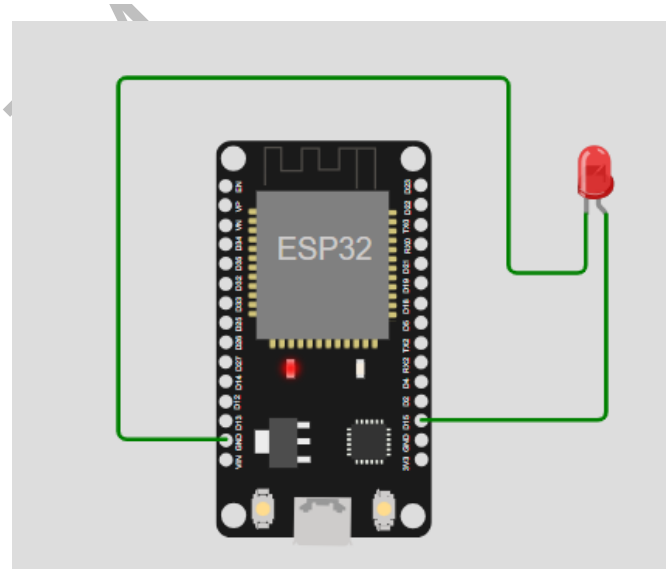
1. To **Start a Conversation:** Click the "Start" button to initiate a chat with BotFather.

4. **Create a Bot:** Type or select the "/newbot" command.
5. **Choose a Name:** BotFather will prompt you to enter a name for your bot.
6. **Choose a Username:** Enter a unique username for your bot, ensuring it ends with "bot".
7. **Receive the Token:** BotFather will provide you with a unique bot token, which you need to authenticate your bot.
8. **Save the Token:** Copy and save the bot token securely, as it is required to interact with your bot through the Telegram Bot API.

2. Obtaining the Chat ID:
    a. Add the newly created bot to the desired Telegram chat or group where you want to receive messages.
    b. Open a web browser and enter the following URL, replacing `<YourBotToken>` with the token you received from BotFather:

    `https://api.telegram.org/bot<YourBotToken>/getUpdates`

    c. JSON response that contains information about the most recent messages received by your bot.
    d. Check for the "chat" object in the response, which contains details about the chat your bot is part of.
    e. The "id" field within the "chat" object corresponds to the chat ID of the group or channel. Make note of this chat ID; you will need it to send messages to the chat.



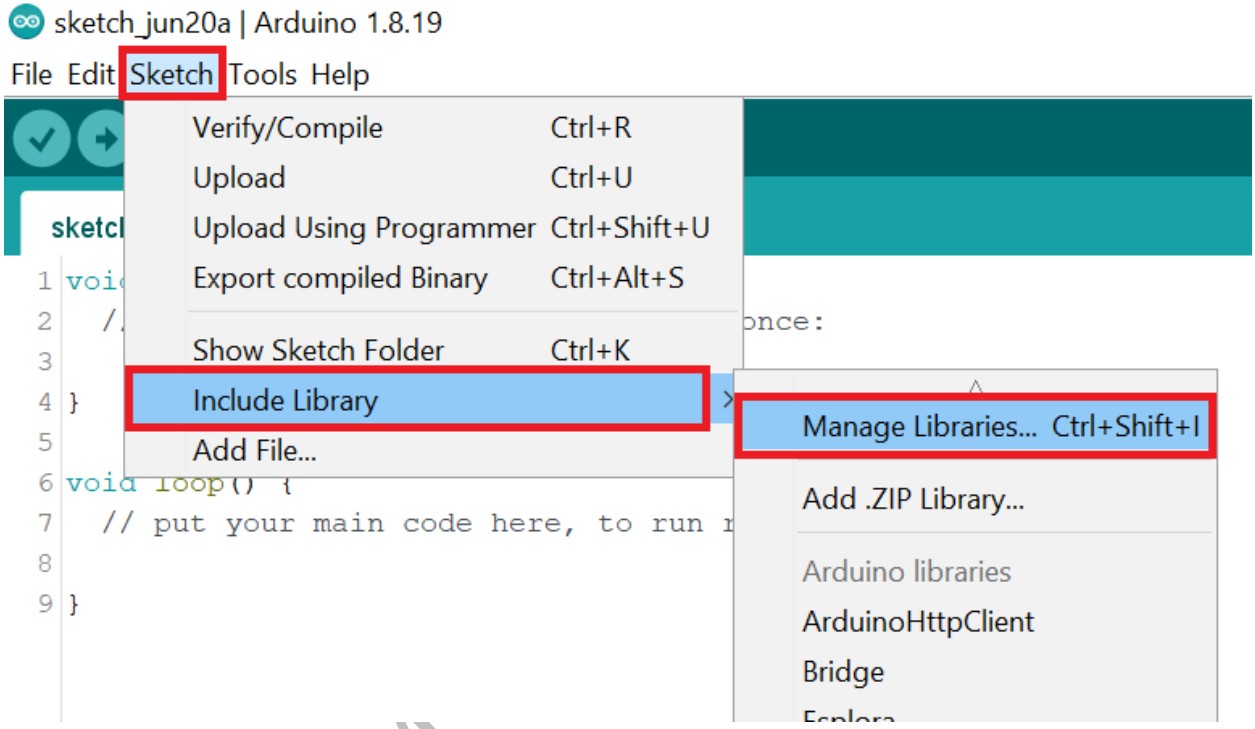
Sending the text message using Telegram to NodeMCU
https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot

Extract the library and add it to the libraries folder path of Arduino IDE.

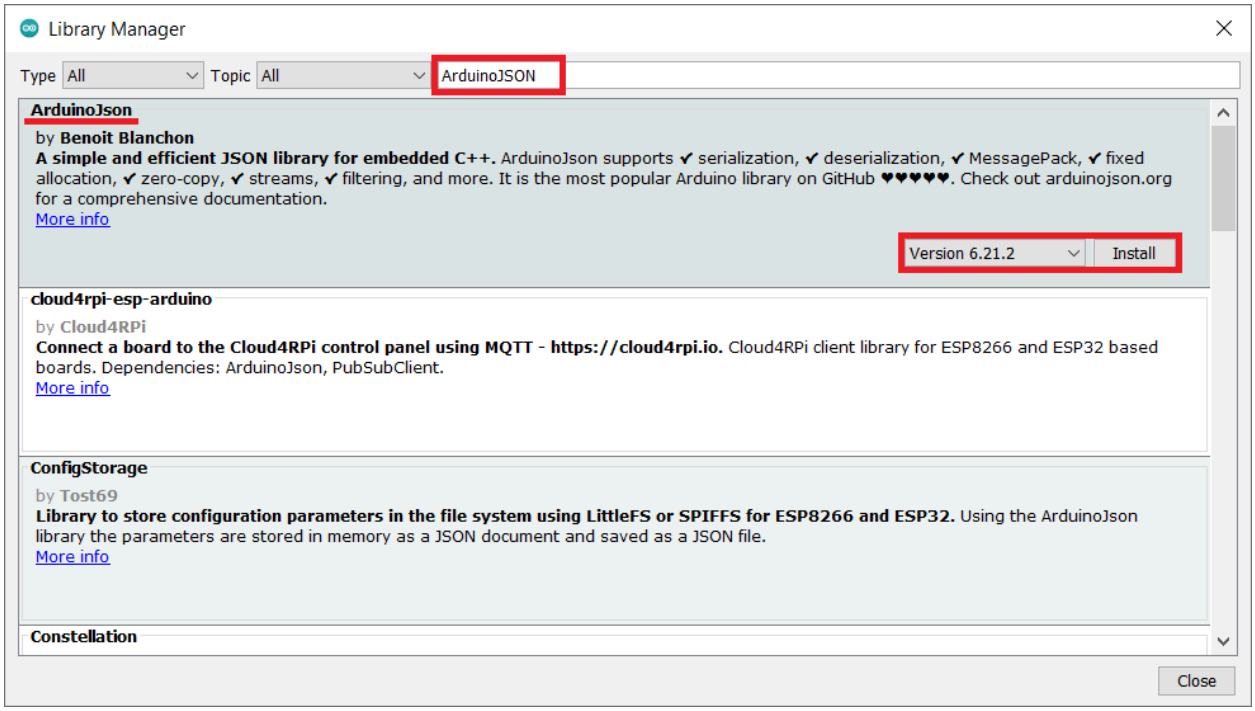
For information about how to add a custom library to the Arduino IDE and use examples from it, refer **Adding Library To Arduino IDE** in the Basics section.

Now install another library which is Arduinojson library for the above example. We need to install the Arduinojson library using the Arduino Library Manager.

- Open the Arduino IDE
- Navigate to **Sketch ► Include Library ► Manage Libraries…**



- The library Manager window will pop up. Now enter Arduinojson into the search box, and click Install on the Arduinojson option to install version 6.12.2 or higher. As shown below image.



Control the led using Telegram

Let's control the LED from telegram using the ESP32 and Arduino IDE.

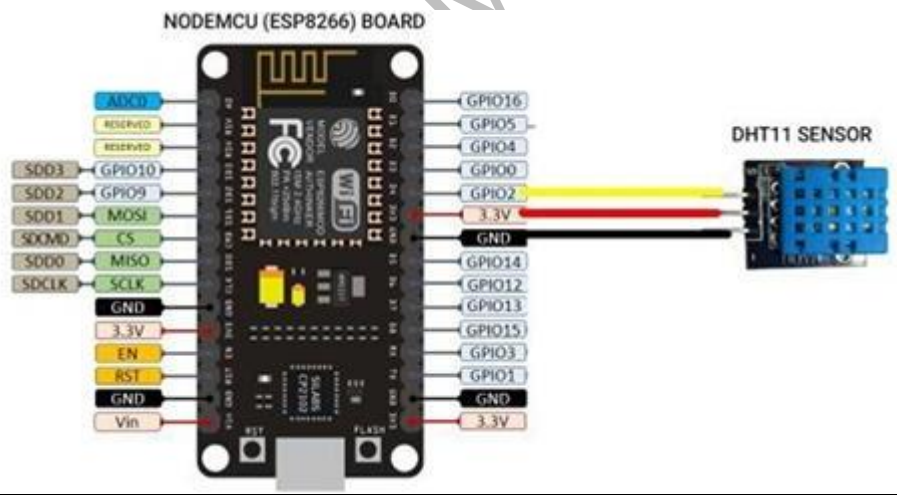
Before uploading the code make sure you have added your **SSID, Password, Token ID,** and **Chat ID**

9 Write a program on Arduino/Raspberry Pi to publish temperature data to the MQTT broker.

Component Required:

Sl No	Components	Quantity
1	Node MCU ESP 8266	1
2	Temperature sensor DHT11	1
3	USB cable	1
4	Connecting wires	--
6	Breadboard	1

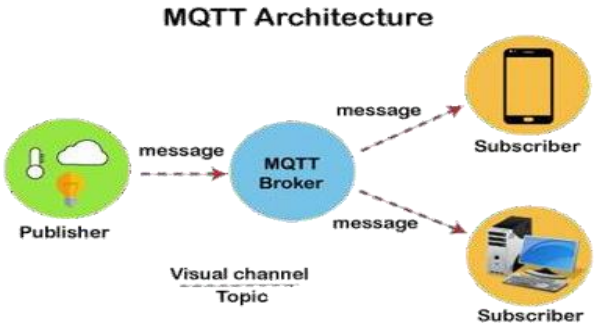
Circuit Diagram



PIN Configuration : NODMCU 3.3V to DHT VCC Pin
NODMCU GND to DHT GND pin
NODMCU GPIO2 to DHT DATA pin

Theory:

MQTT stands for Message Queuing Telemetry Transport. MQTT is a simple messaging protocol, designed for constrained devices with low bandwidth. So, it's the perfect solution to exchange data between multiple IoT devices.
Devices publish messages on a specific topic. All devices that are subscribed to that topic receive the message.



In a publish and subscribe system, a device can publish a message on a topic, or it can be subscribed to a particular topic to receive messages.
The MQTT broker is responsible for receiving all messages, filtering the messages, deciding who is interested in them, and then publishing the message to all subscribed clients.

The MQTT broker is the central point of communication, and it is in charge of dispatching all messages between the senders and the rightful receivers. A client is any device that connects to the broker and can publish or subscribe to topics to access the information. A topic contains the routing information for the broker. Each client that wants to send messages publishes them to a certain topic, and each client that wants to receive messages subscribes to a certain topic. The broker delivers all messages with the matching topic to the appropriate clients.

In this Experiment, we will create a setup that allows a NODE MCUESP8266 board to send data to another MCU ESP 8266, using MQTT (Message Queuing Telemetry Transport). The sender device simply publishes a message to a broker service, which then can be subscribed to by a receiver device.
The data we will send consists of readings from a DHT11 sensor, including temperature and humidity data, from a NODE MCU ESP8266 to another NODE MCU. This experiment utilizes the broker test.mosquitto.org, an open-source service that is free for anyone to use.

To view the data:

1. Go to http://www.hivemq.com/demos/websocket-client/
2. Click "Connect"
3. Under Subscriptions, click "Add New Topic Subscription"
4. In the Topic field, type "weather" then click "Subscribe"

change the temperature/humidity, and you should see
the message appear on the MQTT Broker, in the "Messages" pane.

Program:

```
#include <ESP8266WiFi.h>       // Library to connect ESP8266 to WiFi
#include <PubSubClient.h>      // MQTT client library
#include <DHT.h>               // DHT sensor library
#include <ArduinoJson.h>       // JSON handling library
```

```
// WiFi credentials
const char* ssid = "Enter your Hot spot ID";   // Replace with your WiFi SSID
const char* password = "Password";             // Replace with your WiFi password

// MQTT broker details
const char* mqtt_server = "broker.mqttdashboard.com";  // Public MQTT broker
const int mqtt_port = 1883;                            // Default MQTT port
const char* mqtt_topic = "weather monitoring";         // Topic to publish data
const char* mqtt_client_id = "arduino-weather-demo";   // Unique MQTT client ID

// DHT Sensor setup
#define DHTPIN D4          // GPIO pin where DHT11 is connected
#define DHTTYPE DHT11      // Specify the type of DHT sensor used (DHT11)
DHT dht(DHTPIN, DHTTYPE);  // Create DHT object

// Create WiFi and MQTT client objects
WiFiClient espClient;
PubSubClient client(espClient);

// Variable to track previous payload to avoid redundant publishing
String prevPayload = "";

// Connect to WiFi
void connectToWiFi() {
  Serial.print("Connecting to WiFi");
  WiFi.begin(ssid, password);     // Start connecting to WiFi
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);                   // Wait until connected
  }
  Serial.println(" Connected!");
}

// Connect to MQTT broker
void connectToMQTT() {
  Serial.print("Connecting to MQTT... ");
  while (!client.connected()) {
    if (client.connect(mqtt_client_id)) {
      Serial.println("Connected!");
    } else {
      Serial.print("Failed. Retry in 5 seconds. State: ");
      Serial.println(client.state());
      delay(5000);                // Retry after delay
    }
  }
}

void setup() {
```

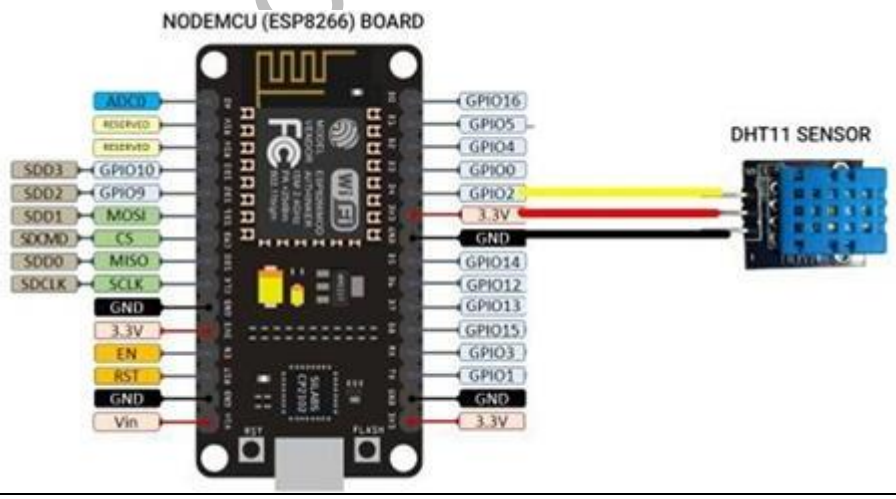
```
  Serial.begin(9600);           // Start serial communication for debugging
  dht.begin();                  // Initialize the DHT sensor
  connectToWiFi();              // Connect to WiFi
  client.setServer(mqtt_server, mqtt_port);  // Set MQTT broker details
  connectToMQTT();              // Connect to MQTT broker
}

void loop() {
  if (!client.connected()) {
    connectToMQTT();            // Reconnect if MQTT connection is lost
  }
  client.loop();                // Maintain MQTT connection

  Serial.print("Measuring weather conditions... ");
  float temperature = dht.readTemperature();   // Read temperature
  float humidity = dht.readHumidity();         // Read humidity

  // Check if readings are valid
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Prepare JSON payload
  StaticJsonDocument<100> jsonDoc;
  jsonDoc["temp"] = temperature;
  jsonDoc["humidity"] = humidity;

  String payload;
  serializeJson(jsonDoc, payload);  // Convert JSON object to string

  // Publish only if data has changed
  if (payload != prevPayload) {
    Serial.println("Updated!");
    Serial.print("Reporting to MQTT topic ");
    Serial.print(mqtt_topic);
    Serial.print(": ");
    Serial.println(payload);
    client.publish(mqtt_topic, payload.c_str());  // Publish to MQTT topic
    prevPayload = payload;                        // Update previous payload
  } else {
    Serial.println("No change");                  // Skip publish if no change
  }

  delay(1000);  // Delay before next reading (1 second)
}
```

10 Write a program to create a UDP server on Arduino/Raspberry Pi and respond with humidity data to the UDP client when requested.

Component Required:

Sl No	Components	Quantity
1	Node MCU ESP 8266	1
2	Temperature sensor DHT11	1
3	USB cable	1
4	Connecting wires	--
6	Breadboard	1

Circuit Diagram



PIN Configuration : NODMCU 3.3V to DHT VCC Pin
NODMCU GND to DHT GND pin
NODMCU D4 (GPIO2) to DHT DATA pin

Theory

User Datagram Protocol (UDP) is a network communication protocol that operates at the transport layer of the Internet Protocol (IP) suite. It is a connectionless and lightweight protocol designed for fast and efficient data transmission, but it does not provide the same level of reliability and error-checking as Transmission Control Protocol (TCP).

UDP is a lightweight, connectionless, and fast protocol that prioritizes low-latency data transmission over reliability. It is suitable for applications where occasional packet loss or out-of-order delivery can be tolerated, and real-time communication is essential. However, for applications that require guaranteed delivery and error recovery, TCP is a better choice.

```
#include <ESP8266WiFi.h>    // WiFi library for ESP8266
#include <WiFiUdp.h>        // UDP communication library
#include <DHT.h>            // DHT sensor library

#define DHTPIN D4           // Define the pin connected to the DHT sensor (GPIO2)
#define DHTTYPE DHT11       // Sensor type is DHT11; use DHT22 if applicable
DHT dht(DHTPIN, DHTTYPE);   // Create DHT sensor object

// WiFi credentials (change to match your network)
const char* ssid = "SYS123";
const char* password = "12345678";

// Server IP and port (must match the server configuration)
const char* udpServerIP = "192.168.0.168";  // Server IP address
const uint16_t udpPort = 1234;              // UDP port number

WiFiUDP udp;  // Create a UDP object

void setup() {
  Serial.begin(9600);  // Start serial communication for debugging
  delay(1000);         // Short delay before starting

  WiFi.begin(ssid, password);  // Connect to WiFi
  Serial.println("Connecting to WiFi...");

  // Wait until WiFi is connected
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // WiFi connected
  Serial.println("\nWiFi connected successfully");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());  // Print device's IP address
```

```
  dht.begin();  // Initialize the DHT sensor
}

void loop() {
  float temp = dht.readTemperature();  // Read temperature in Celsius

  // Check if reading is valid
  if (isnan(temp)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Print temperature to serial monitor
  Serial.print("Temperature in degree Celsius: ");
  Serial.println(temp);

  // Convert float temperature to a string
  String tempStr = String(temp);

  // Send temperature via UDP to the server
  udp.beginPacket(udpServerIP, udpPort);  // Begin UDP packet
  udp.write(tempStr.c_str());             // Write temperature string
  udp.endPacket();                        // Send the packet

  delay(2000);  // Wait for 2 seconds before sending again
}
```

```
# Python program for server

import socket  # Import socket module for network communication

host = '0.0.0.0'  # Listen on all network interfaces
port = 1234       # UDP port (must match ESP8266's target port)

# Create a UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the host and port
server_socket.bind((host, port))

print(f"UDP server on {host}:{port}")

# Continuously listen for incoming UDP data
while True:
    data, addr = server_socket.recvfrom(1024)  # Receive up to 1024 bytes
    print(f"Received from {addr}: {data.decode()} °C")  # Print received data
```

Procedure

To run Python code for UDP Server

1) Install python software
2) Open python IDLE
3) In python IDLE go to **File - New File** (it opens new script windows) -**Type the code**
4) Click on **Run** button and **Save** the program from script window
5) See the output from IDLE shell (command prompt)

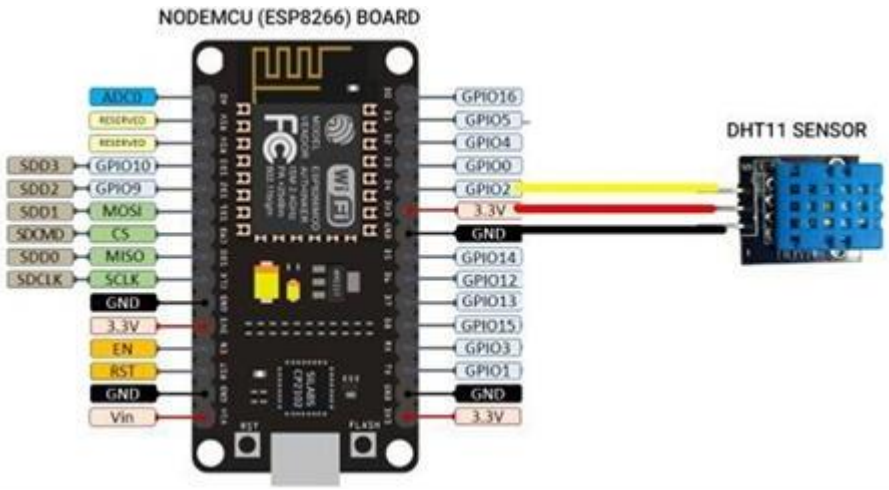
Result: Python program used to create UDP server and Arduino program written to respond with humidity data to UDP client when requested.

11 Write a program to create a TCP server on Arduino /Raspberry Pi and respond with humidity data to the TCP client when requested.

Component Required:

Sl No	Components	Quantity
1	Node MCU ESP 8266	1
2	Temperature sensor DHT11	1
3	USB cable	1
4	Connecting wires	--
6	Breadboard	1

Circuit Diagram



PIN Configuration : NODMCU 3.3V to DHT VCC Pin
NODMCU GND to DHT GND pin
NODMCU D4 (GPIO2) to DHT DATA pin

Theory

The Transmission Control Protocol (TCP) is a widely used protocol in the Internet Protocol (IP) suite. It is a connection-oriented protocol that provides reliable, ordered, and error-checked delivery of data between applications running on hosts1. In IoT, TCP is used to establish connections between clients and servers, allowing devices to interact with each other and resolve common problems.

A TCP server is a program that listens for incoming connections from clients and responds to their requests. When a client connects to the server, it sends a request for data. The server then sends back the requested data to the client. In IoT, TCP servers are used to provide access to data from sensors and other devices
In the context of the Internet of Things (IoT), a TCP server refers to a network service running on a device or gateway that listens for incoming TCP connections from other IoT devices or clients

In the context of IoT, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are two protocols of the Transport Layer that are used to transmit data between devices over a network.

TCP is a connection-oriented protocol that provides reliable delivery services by keeping track of the segments being transmitted or received by assigning numbers to every single one of them. It also implements an error control mechanism for reliable data transfer and takes into account the level of congestion in the network .

On the other hand, UDP is a connectionless protocol that is used for simple request-response communication when the size of data is less and hence there is lesser concern about flow and error control. It is a suitable protocol for multicasting as UDP supports packet switching. Normally used for real-time applications which cannot tolerate uneven delays between sections of a received message.

In summary, TCP is more reliable but slower than UDP, while UDP is faster but less reliable than TCP

Finding IP address of your computer

1) Search command Prompt from your PC
2) Type ***ipconfig*** in command prompt
3) Search ***IPv4 Adress …….: 192.168.137.1*** like this in prompt
4) Copy and paste the same in the program at ***tcpServerIP***

```
#include <ESP8266WiFi.h>     // Include ESP8266 WiFi library
#include <DHT.h>             // Include DHT sensor library
```

```
#define DHTPIN D4          // Define the GPIO pin connected to the DHT sensor (D4)
#define DHTTYPE DHT11      // Define sensor type (use DHT22 if using that instead)
DHT dht(DHTPIN, DHTTYPE);  // Create a DHT sensor object

const char* ssid = "SYS123";        // WiFi network SSID (replace with your network)
const char* password = "12345678";  // WiFi password

const char* tcpServerIP = "192.168.0.168";  // IP address of the server (e.g., your PC)
const uint16_t port = 1234;                 // TCP port to connect to the server

WiFiClient client;  // Create a TCP client object

void setup() {
  Serial.begin(9600);  // Start serial communication for debugging
  delay(1000);         // Short delay for stability

  WiFi.begin(ssid, password);       // Start WiFi connection
  Serial.println("Connecting to WiFi...");

  while (WiFi.status() != WL_CONNECTED) {  // Wait until connected
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nWiFi connected successfully");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());     // Print local IP for reference

  dht.begin();   // Initialize the DHT sensor
}

void loop() {
  if (!client.connect(tcpServerIP, port)) {  // Try connecting to the server
    Serial.println("Connection to server failed");
    delay(1000);
    return;   // Skip sending if connection fails
  }

  float temp = dht.readTemperature();  // Read temperature in Celsius

  if (isnan(temp)) {  // Check if reading failed
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  Serial.print("Temperature in degree Celsius: ");
  Serial.println(temp);

  client.print(temp);  // Send temperature as plain text over TCP
```

```
  client.stop();      // Close the TCP connection
  delay(2000);        // Wait 2 seconds before sending the next reading
}
```

```
//server program in python

import socket  # Import socket module for networking

host = '0.0.0.0'  # Listen on all available interfaces (any IP)
port = 1234       # Port to listen on (must match NodeMCU client)

# Create a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((host, port))       # Bind to the specified host and port
server_socket.listen(1)                # Start listening for incoming connections

print(f"Server listening on {host}:{port}")
while True:
    client_socket, addr = server_socket.accept()  # Accept new connection
    print(f"Connection from {addr}")
    try:
        while True:  # Keep receiving data until connection is closed
            data = client_socket.recv(1024)  # Receive up to 1024 bytes
            if not data: break               # If no data, client disconnected
            print(f"Received: {data.decode()} °C")  # Decode and print temperature
    finally:
        client_socket.close()  # Close client connection when done
```

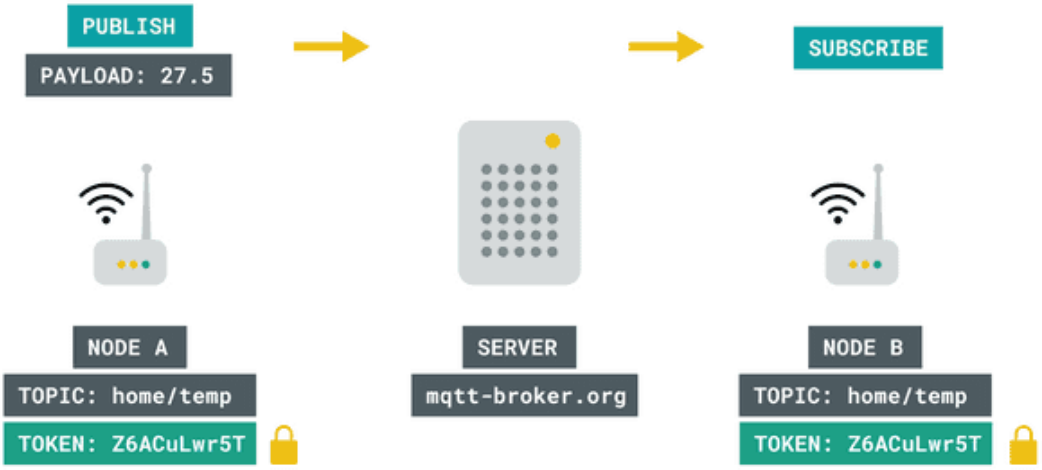
12 Write a program on Arduino / Raspberry Pi to subscribe to the MQTT broker for temperature data and print it.

Component Required:

Sl No	Components	Quantity
1	Node MCU ESP 8266	1
2	USB cable	1
3	Connecting wires	--
4	Breadboard	1

Theory

The MQTT protocol was first introduced in 1999, as a light-weight **publish** and **subscribe** system. It is particularly useful for devices with low-bandwidth, here we can send commands, sensor values or messages over the Internet with little effort.



A basic explanation on how it works is that a node, for example and Arduino with a Wi-Fi module, sends a payload to a broker. A broker is a kind of "middle-point" server, that essentially stores payloads sent to it, in something called **topics**. A topic, is a definition of what type of data it contains, it could for example be "humidity" or " temperature". Another node can then subscribe to this information, from the broker, and voilà, data has been moved from Node A to Node B over the Internet.

One way to protect the data is for example, by using a **token**, something that is quite common when working with various IoT services. For instance, if we are publishing something to a broker, anyone that has the URL, e.g. **randombroker.org/randomtopic** can subscribe to it. But if we add a unique token on both sides, they wouldn't be able to. These tokens could for example be **Z6ACuLwr5T**, which is not exactly something easy to guess


```
// Include the ESP8266 WiFi library to connect to WiFi networks
#include <ESP8266WiFi.h>

// Include the Arduino MQTT client library for MQTT communication
#include <ArduinoMqttClient.h>

// WiFi credentials (update with your own network's SSID and password)
const char *ssid = "SYS123"; // Replace with your WiFi SSID
const char *password = "12345678"; // Replace with your WiFi password

// MQTT broker information
const char broker[] = "test.mosquitto.org"; // Public test MQTT broker
int port = 1883; // Default MQTT port

// Topic to subscribe to (temperature topic)
const char topicT[] = "home/temp";
```

```
// Create a WiFi client to handle the network connection
WiFiClient wifiClient;

// Create an MQTT client using the WiFi client
MqttClient mqttClient(wifiClient);

void setup()
{
 Serial.begin(9600); // Start the serial monitor at 9600 baud

 // Connect to the WiFi network
 Serial.print("Wifi connecting to ");
 Serial.println(ssid);
 WiFi.begin(ssid, password);

 // Wait until the device is connected to WiFi
 while (WiFi.status() != WL_CONNECTED) {
  Serial.println("Connecting...");
  delay(500);
 }

 Serial.println("WiFi connected successfully");

 // Attempt to connect to the MQTT broker
 Serial.print("Attempting to connect to the MQTT broker: ");
 Serial.println(broker);
 if (!mqttClient.connect(broker, port)) {
  Serial.print("MQTT connection failed! Error code = ");
  Serial.println(mqttClient.connectError());
  while (1); // Halt the program if connection fails
 }

 Serial.println("You're connected to the MQTT broker!");

 // Set the callback function to handle incoming messages
 mqttClient.onMessage(onMqttMessage);

 // Subscribe to the specified topic
 Serial.print("Subscribing to topic: ");
 Serial.println(topicT);
 mqttClient.subscribe(topicT);

 Serial.print("Topic Temperature: ");
 Serial.println(topicT);
}

void loop()
{
```

```
  // Poll the MQTT client to handle incoming messages and keep the connection alive
  mqttClient.poll();
  delay(1000); // Wait for a second before the next poll
}

// Callback function that is called when a message is received
void onMqttMessage(int messageSize) {
  Serial.println("Received a message with topic '");
  Serial.print(mqttClient.messageTopic());
  Serial.print("', length ");
  Serial.print(messageSize);
  Serial.println(" bytes:");

  // Print the message contents
  while (mqttClient.available()) {
    Serial.print((char)mqttClient.read());
  }

  Serial.println();
  Serial.println();
  delay(1500); // Delay to help with readability of output
}
```