IOT DEVELOPMENT, APPLICATIONS AND PRACTICE LAB MANUAL

SYLLABUS

20CIC05

IOT DEVELOPMENT, APPLICATIONS AND PRACTICE LAB

Instruction	3 Hours per week
Duration of End Examination	3 Hours
Semester End Examination	50 Marks
Continuous Internal Evaluation	50 Marks
Credits	1.5

Pre-requisites: CAMP, Programming Basics

Course Objectives: The objectives of this course are to

1. Understand the basics of IoT.
2. Impart necessary and practical Skills using components of Internet of Things.
3. Develop skills required to build real-time IoT based projects.

Course Outcomes: On successful completion of the course, students will be able to

1. Use of various hardware and software components related to Internet of Things.
2. Interface I/O devices, sensors to Raspberry Pi/Arduino.
3. Implement various communication protocols in IoT.
4. Monitoring remote system using IoT.
5. Hypothesizing Real time IoT based projects.
6. Develop real life IoT based projects.

LIST OF EXPERIMENTS:

1. Introduction to IoT equipment's and Perform necessary software installation.
2. Write a program to interface PIR sensor with Arduino and turn ON LED when motion is detected.
3. Write a program to interface DHT22 sensor with Arduino and display temperature and humidity readings.
4. Write a program to interface motor with Raspberry Pi. Turn ON motor when the temperature is high.
5. Write a program to interface LCD with Raspberry Pi and print temperature and humidity readings on it.
6. Write a program to send sensor data to smart phone using Bluetooth.
7. Write a program to interface flame/smoke sensor with Arduino /Raspberry Pi and give an alert message when flame/smoke is detected.
8. Perform experiment with GPS module by interfacing with Arduino/Raspberry Pi.
9. Write a program to send and receive messages using MQTT protocol.
10. Write a program to upload sensor data to local/cloud server using Wi-Fi.
11. Write a program to retrieve sensor data from local/cloud server.
12. Implement any case study using Arduino/Raspberry Pi.

Text Books:

1. ArshdeepBahga and Vijay Madisetti, "Internet of Things: A Hands-on Approach", Universities Press, 2014.

Suggested Reading:

1. Dr. SRN Reddy, RachitTirnkral and Manasi Mishra, "Introduction to Internet of Things: A practical Approach", ETI Labs, 2018.
2. Adrian McEwen, "Designing the Internet of Things", Wiley, 2013.
3. Raj Kamal, "Internet of Things: Architecture and Design", McGraw Hill, 2017.
4. CunoPfister, "Getting Started with the Internet of Things", 0 Reilly Media, 2011.
5. O. Vermesan, P. Friess, "Internet of Things – Converging Technologies for Smart Environments and Integrated Ecosystems", River Publishers, Series in Communications, 2013.

Online Resources:

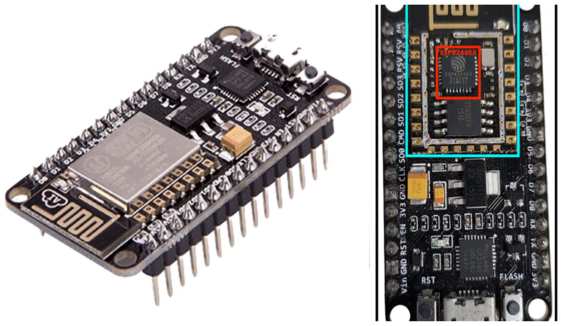
1. Li Da Xu, Wu He, and Shancang Li, "Internet of Things in Industries: A Survey ", IEEE Transactions on Industrial Informatics, Vol. 10, No. 4, Nov. 2014.
2. T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", IETF, Standards Track, Mar. 2012.
3. Z. Shelby, K. Hartke, C. Bormann, "The Constrained Application Protocol (CoAP)", Internet Engineering Task Force (IETF), Standards Track, 2014.
4. L.Fenzel, "What's The Difference between IEEE 802.15.4 And ZigBee Wireless?" Electronic Design (Online), Mar. 2013.
5. S. N. Das and S. Misra, "Information theoretic self-management of Wireless Sensor Networks", Proceedings of NCC 2013.
6. F. Luo *et al*., "A Distributed Gateway Selection Algorithm for UAV Networks," in IEEE Transactions on Emerging Topics in Computing, vol. 3, no. 1, pp. 22-33, March 2015.

Mapping of Course Outcomes with Program Outcomes and Program Specific Outcomes:

PO/PSO / *CO*	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO1	3	-	-	2	2	-	1	-	3	1	1	3	1	2	-
CO2	3	3	2	3	3	3	3	-	3	1	2	3	2	2	-
CO3	3	3	2	2	3	3	3	-	3	1	2	3	1	1	2
CO4	3	3	2	3	3	2	3	-	3	1	2	3	2	2	2
CO5	3	3	2	3	3	3	3	-	3	3	3	3	2	2	1
CO6	3	3	2	3	3	3	3	3	3	3	3	3	3	3	3

I. Introduction to IoT equipment and necessary software installation

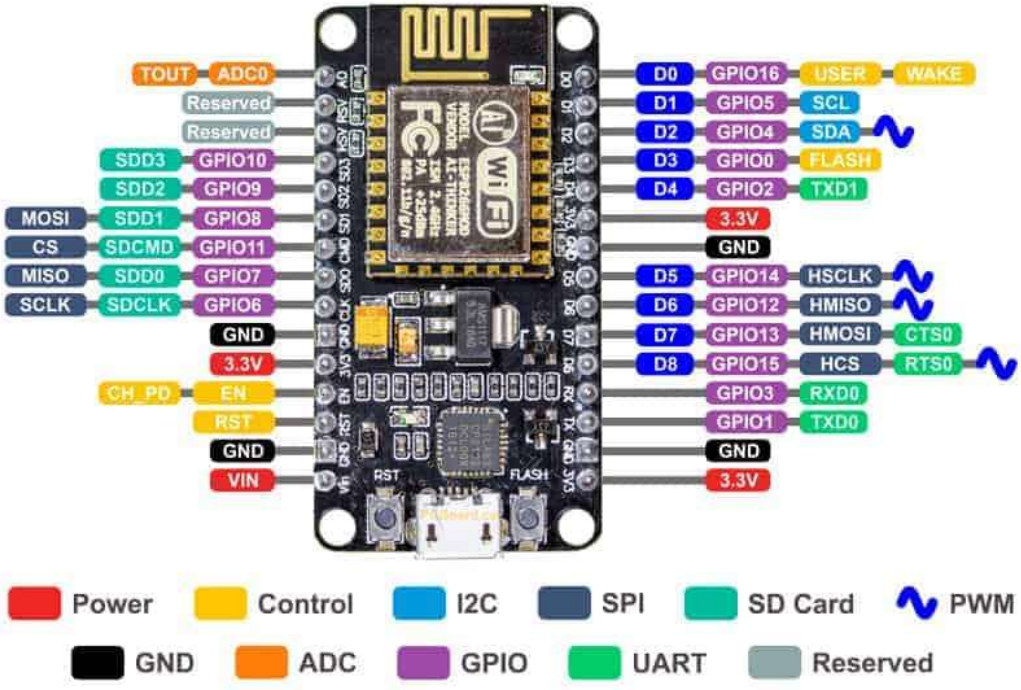
Step 1: NodeMCU Devkit 1.0



NodeMCU (Node MicroController Unit) 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.

But, what about Arduino? The Arduino project created an open-source hardware design and software SDK for their versatile IoT controller. Similar to NodeMCU, the Arduino hardware is a microcontroller board with a USB connector, LED lights, and standard data pins. It also defines standard interfaces to interact with sensors or other boards. But unlike NodeMCU, the Arduino board can have different types of CPU chips (typically an ARM or Intel x 86 chips) with memory chips, and a variety of programming environments. There is an Arduino reference design for the ESP8266 chip as well. However, the flexibility of Arduino also means significant variations across different vendors. For example, most Arduino boards do not have WiFi capabilities, and some even have a serial data port instead of a USB port. The term NodeMCU usually refers to the firmware, while the board is called Devkit. NodeMCU Devkit 1.0 consists of an ESP-12E on a board, which facilitates its use. It also has a voltage regulator, a USB interface.

- **NodeMCU Pinout and Functions Explained**



Power Pins There are four power pins. **VIN** pin and three **3.3V** pins.

VIN can be used to directly supply the NodeMCU/ESP8266 and its peripherals. Power delivered on **VIN** is regulated through the onboard regulator on the NodeMCU module – you can also supply 5V regulated to the **VIN** pin **3.3V** pins are the output of the onboard voltage regulator and can be used to supply power to external components.

GND are the ground pins of NodeMCU/ESP8266

I2C Pins are used to connect I2C sensors and peripherals. Both I2C Master and I2C Slave are supported. I2C interface functionality can be realized programmatically, and the clock frequency is 100 kHz at a maximum. It should be noted that I2C clock frequency should be higher than the slowest clock frequency of the slave device.

GPIO Pins NodeMCU/ESP8266 has 17 GPIO pins which can be assigned to functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light and Button programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts.

ADC Channel the NodeMCU is embedded with a 10-bit precision SAR ADC. The two functions can be implemented using ADC. Testing power supply voltage of VDD3P3 pin and testing input voltage of TOUT pin. However, they cannot be implemented at the same time.

UART Pins NodeMCU/ESP8266 has 2 UART interfaces (UART0 and UART1) which provide asynchronous communication (RS232 and RS485), and can communicate at up to 4.5 Mbps. UART0 (TXD0, RXD0, RST0 & CTS0 pins) can be used for communication. However, UART1 (TXD1 pin) features only data transmit signal so; it is usually used for printing log.

SPI Pins NodeMCU/ESP8266 features two SPIs (SPI and HSPI) in slave and master modes. These SPIs also support the following general-purpose SPI features:

- 4 timing modes of the SPI format transfer
- Up to 80 MHz and the divided clocks of 80 MHz
- Up to 64-Byte FIFO

SDIO Pins NodeMCU/ESP8266 features Secure Digital Input/Output Interface (SDIO) which is used to directly interface SD cards. 4-bit 25 MHz SDIO v1.1 and 4-bit 50 MHz SDIO v2.0 are supported.

PWM Pins The board has 4 channels of Pulse Width Modulation (PWM). The PWM output can be implemented programmatically and used for driving digital motors and LEDs. PWM frequency range is adjustable from 1000 µs to 10000 µs (100 Hz and 1 kHz).

Control Pins are used to control the NodeMCU/ESP8266. These pins include Chip Enable pin (EN), Reset pin (RST) and WAKE pin.

EN: The ESP8266 chip is enabled when EN pin is pulled HIGH. When pulled LOW the chip works at minimum power.

RST: RST pin is used to reset the ESP8266 chip.

WAKE: Wake pin is used to wake the chip from deep-sleep.

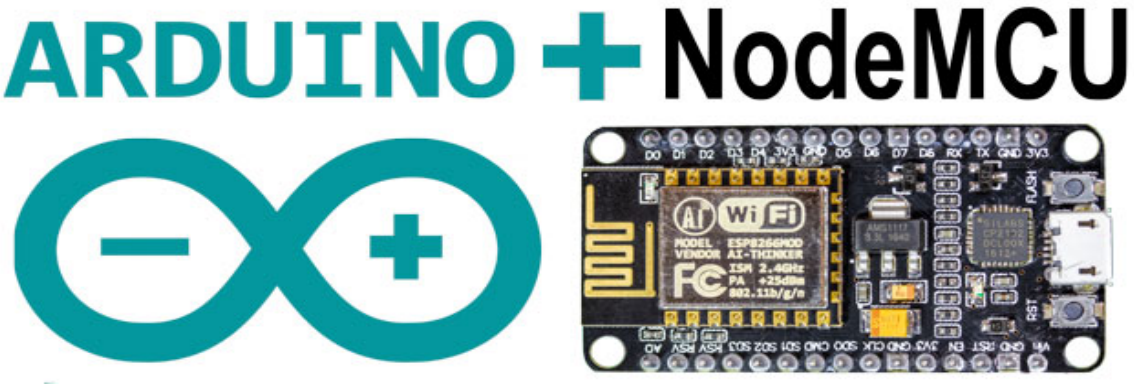
 Control Pins are used to control the NodeMCU/ESP8266. These pins include Chip Enable pin (EN), Reset pin (RST) and WAKE pin.

EN: The ESP8266 chip is enabled when EN pin is pulled HIGH. When pulled LOW the chip works at minimum power.

RST: RST pin is used to reset the ESP8266 chip.

WAKE: Wake pin is used to wake the chip from deep-sleep.

❖ NodeMCU Compatibility with Arduino IDE



The NodeMCU offers a variety of development environments, including compatibility with the Arduino IDE (Integrated Development Environment). The NodeMCU/ESP8266 community took the IDE selection a step further by creating an Arduino add-on. If you're just getting started programming the ESP8266 or even an established developer, this is the highly recommended environment. Setting up and configuring the Arduino IDE for a NodeMCU ESP8266.

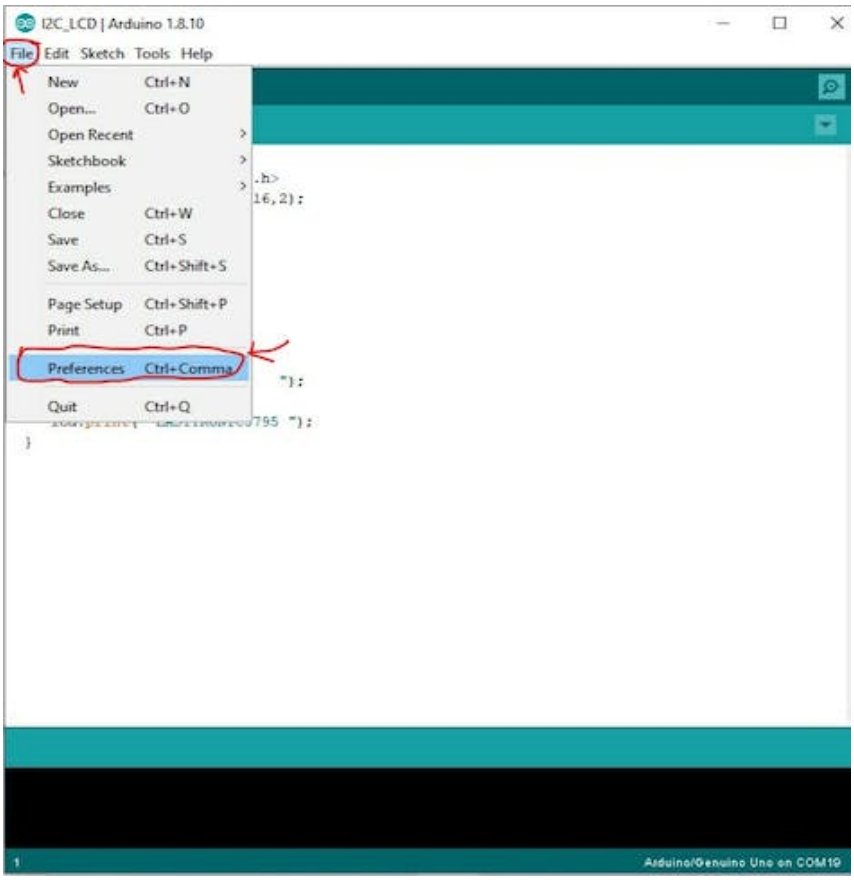
Step 1: Installing Arduino IDE Software

Install Arduino IDE software from the link http://www.arduino.cc/en/main/software

After installing Arduino IDE icon is created on the Desktop as show in the figure.

Step 2:

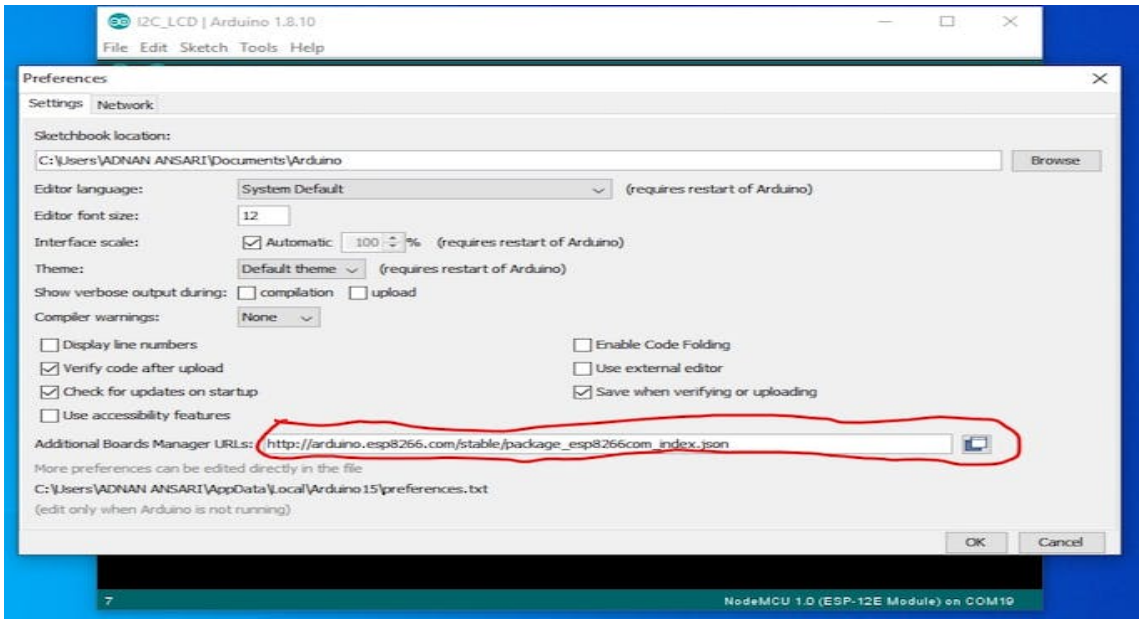
Open the software and click on the left upper corner " file" option then click on the preference. Click on the preference and paste the link that we are sharing in the next step.



Step 3:-

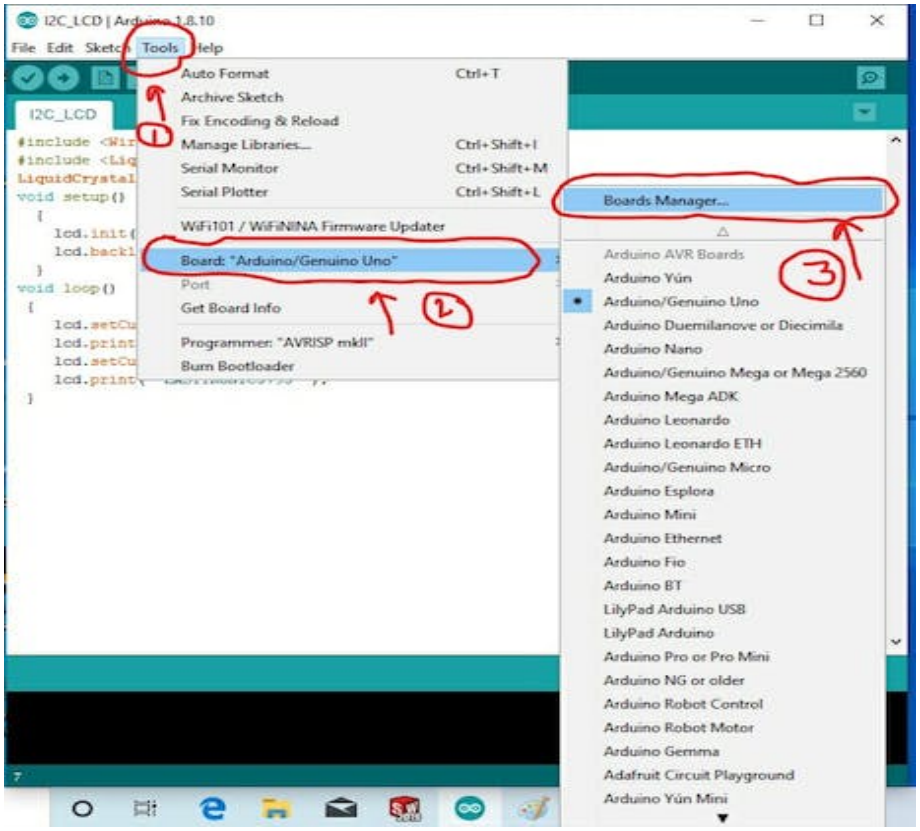
Paste the given link into the additional board manager URL.

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



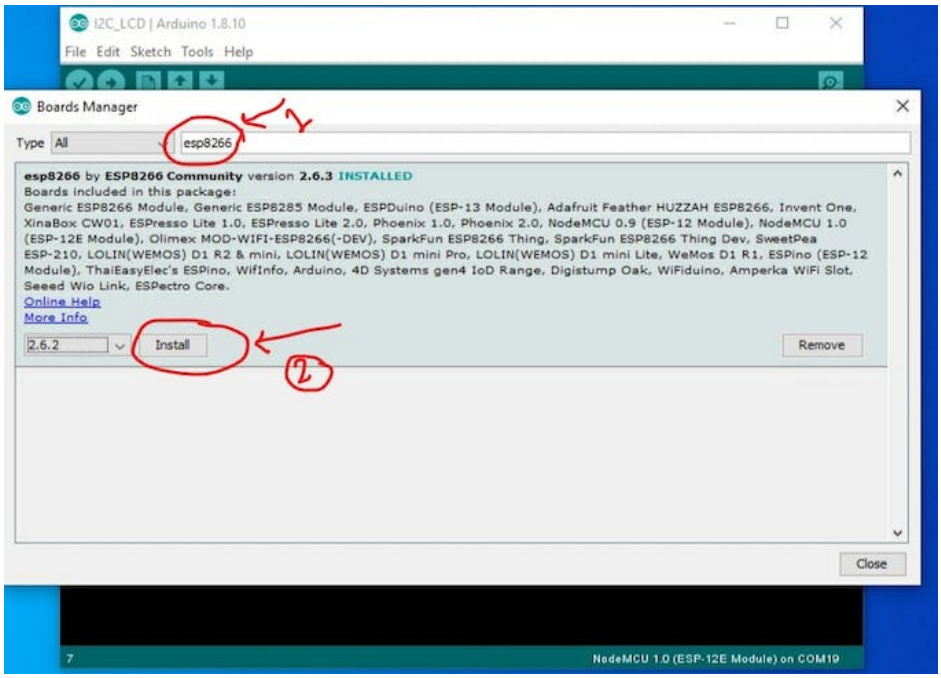
Step 4:-

This is the last step in which you have to go to the tools from the given option. Click on the board where arduino/genuino uno and select the board manager to add the board in App.



Step 5:-

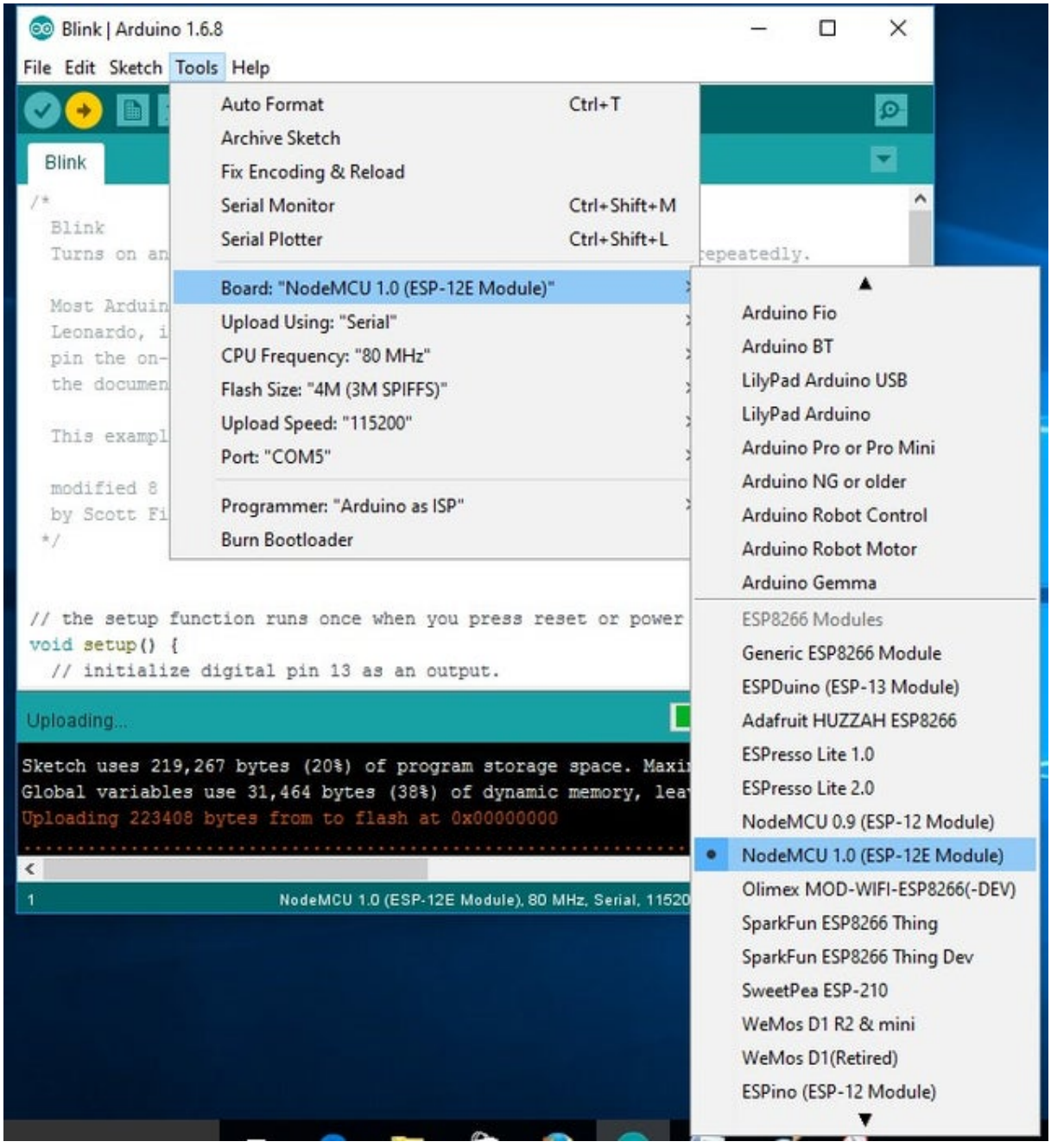
Write esp8266 in the search and install the board to the software.



Step 6:-

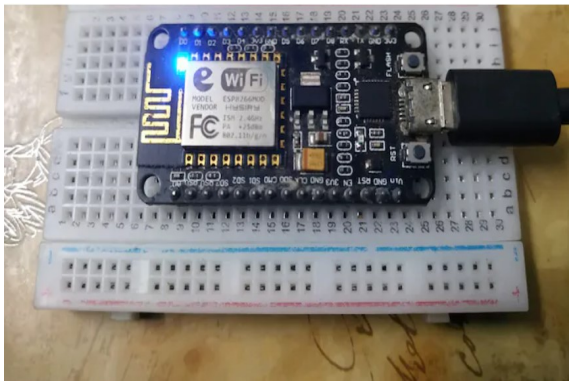
Selecting ESP8266 Arduino Board

To run the esp8266 with Arduino we have to select the **Board: "Arduino/Genuino Uno"** and then change it to **NodeMCU 1.0 (ESP-12E Module)** or other esp8266 modules depending on what you have .This can be done by scrolling down, as shown in the figure

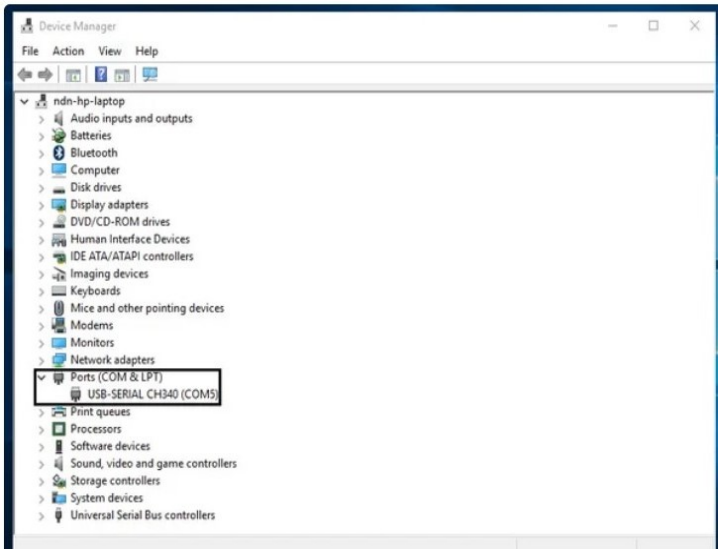


Step 7:-

Connecting ESP8266 to the PC

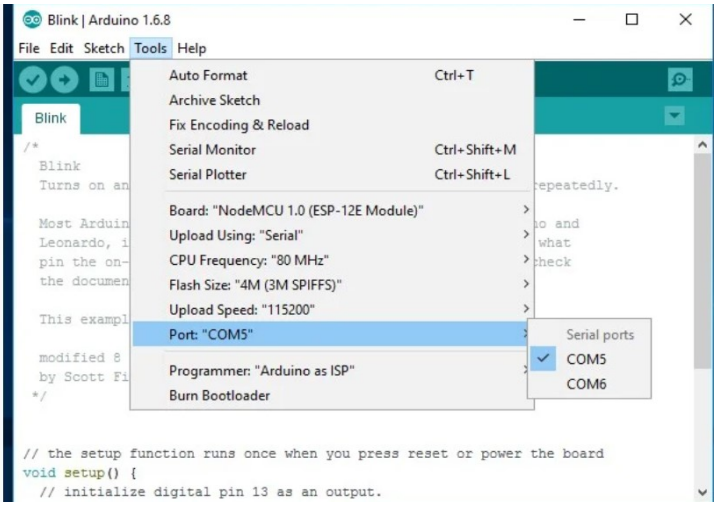


Now let's connect the ESP8266 module to your computer through USB cable as shown in the figure. When module is connected to the USB, COM port is detected eg: here COM5 is shown in the figure.



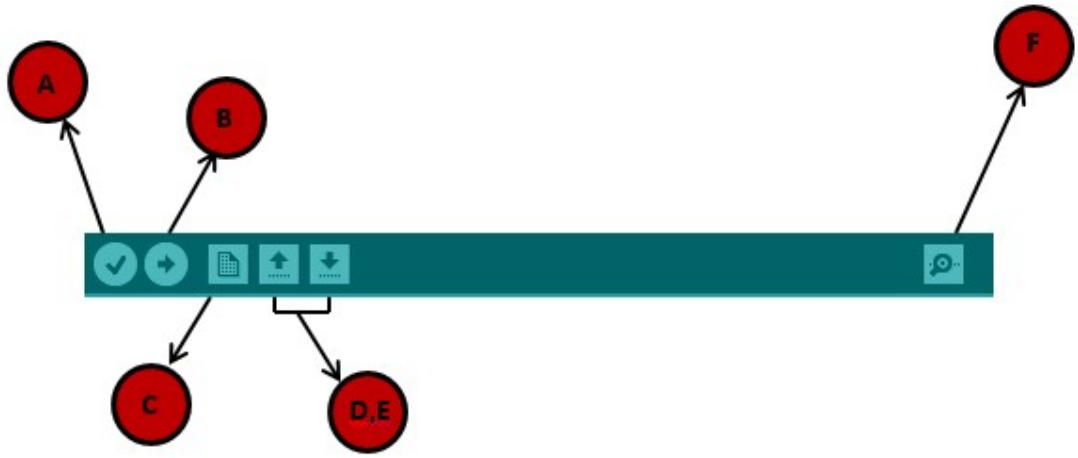
Step 8:- Selecting COM Port

Click on tools to select the port: "COM" based on which esp8266 module is connected to your respected COM port of the computer. To select COM ports refer previous steps.



Step 9:-
Upload the program to your board.

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.



A − Used to check if there is any compilation error.

B − Used to upload a program to the Arduino board.

C − Shortcut used to create a new sketch.

D − Used to directly open one of the example sketch.

E − Used to save your sketch.

F − Serial monitor used to receive serial data from the board and send the serial data to the Board.

Now, simply 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.

Note − If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

We will study in depth, the Arduino program structure and we will learn more new terminologies used in the Arduino world. The Arduino software is open-source. The source code for the Java environment is released under the GPL and the C/C++ microcontroller libraries are under the LGPL.

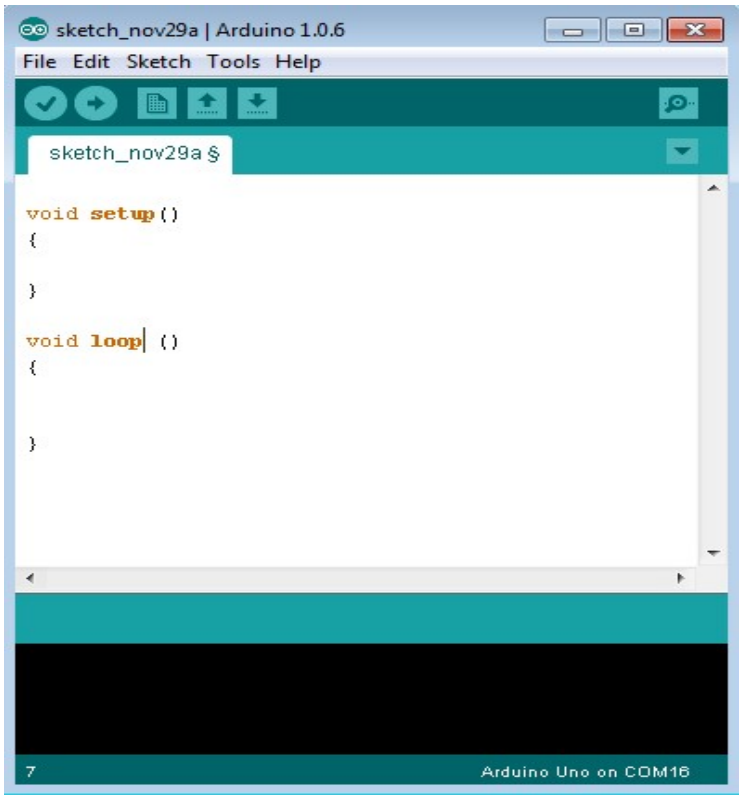
Sketch − the first new terminology is the Arduino program called “sketch”.

Structure

Arduino programs can be divided in three main parts: Structure, Values (variables and constants), and Functions. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Let us start with the Structure. Software structure consists of two main functions −

- Setup( ) function
- Loop( ) function



Void setup ( ) {

}

- **PURPOSE**_– the setup () function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.

Void Loop ( ) {

}

- **PURPOSE**_– after creating a setup () function, which initializes and sets the initial values, the loop () function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

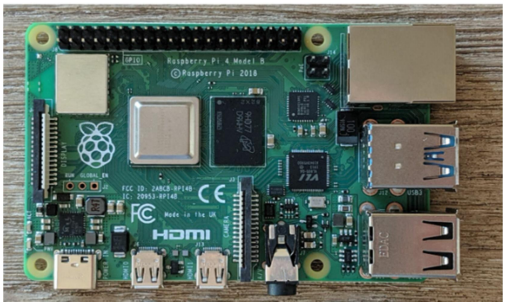
========== ** ==========

Started With Raspberry Pi:-

Six years ago, a single-board computer came on the market and changed the game for tinkerers and DIYers. The Raspberry Pi is a dream machine for all kinds of projects—gaming consoles, home streaming, VPN servers, and beyond—but the first step is gathering up your supplies and learning the basics. If you're building something with the Pi, start here.

What Is the Raspberry Pi?

The Raspberry Pi(Opens in a new window) is a tiny computer about the size of a deck of cards. It uses what's called a system on a chip(Opens in a new window), which integrates the CPU and GPU in a single integrated circuit, with the RAM, USB ports, and other components soldered onto the board for an all-in-one package.



It doesn't have onboard storage, but it has an SD card slot you can use to house your operating system and files.

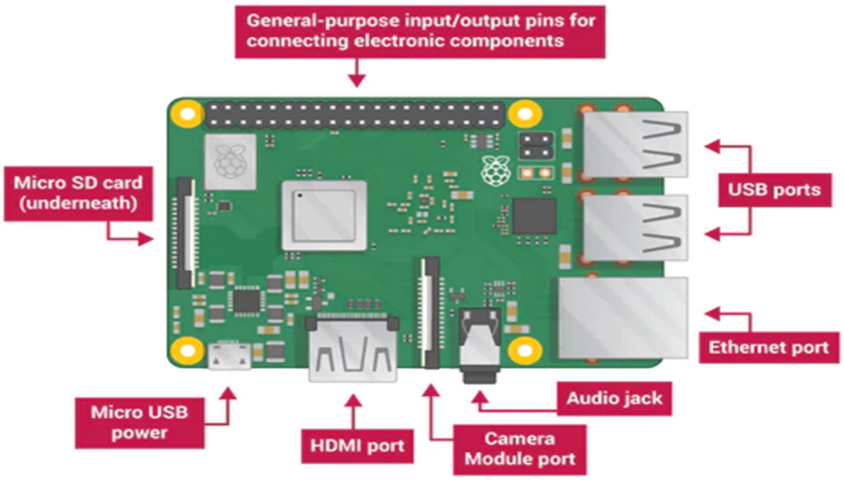
The nonprofit Raspberry Pi Foundation originally designed the Pi as an inexpensive computer for teaching programming, but it quickly became popular among DIYers looking for a more powerful brain in their electronics projects.

Since its inception, many models of the Pi have been released, some with multiple revisions (like the Raspberry Pi 3 Model B+, which improved the previous Model B's networking capabilities).

The latest Pi, at the time of this writing, is the Raspberry Pi 4 , which rocks a 1.5GHz quad-core ARM CPU, a 500MHz Video Core VI GPU, and 1GB of RAM—though you can step up to 4GB of RAM for a bit more money.

The Raspberry Pi has a number of ports which you will use to control the Raspberry Pi, and it can use to control other devices. Your Raspberry Pi will have the following ports:

- **USB –** USB ports are used to connect a wide variety of components, most commonly a mouse and keyboard.
- **HDMI** – The HDMI port outputs video and audio to your monitor.
- **Audio** – The audio jack allows you to connect standard headphones and speakers.
- **Micro USB** – The Micro USB port is only for power, do not connect anything else to this port. Always connect the power after you have already connected everything else.
- **GPIO** – The GPIO ports allow the Raspberry Pi to control and take input from any electronic component.
- **SD card slot** – The Raspberry Pi uses SD cards the same way a full-size computer uses a hard drive. The SD card provides the Raspberry Pi with internal memory, and stores the hard drive.



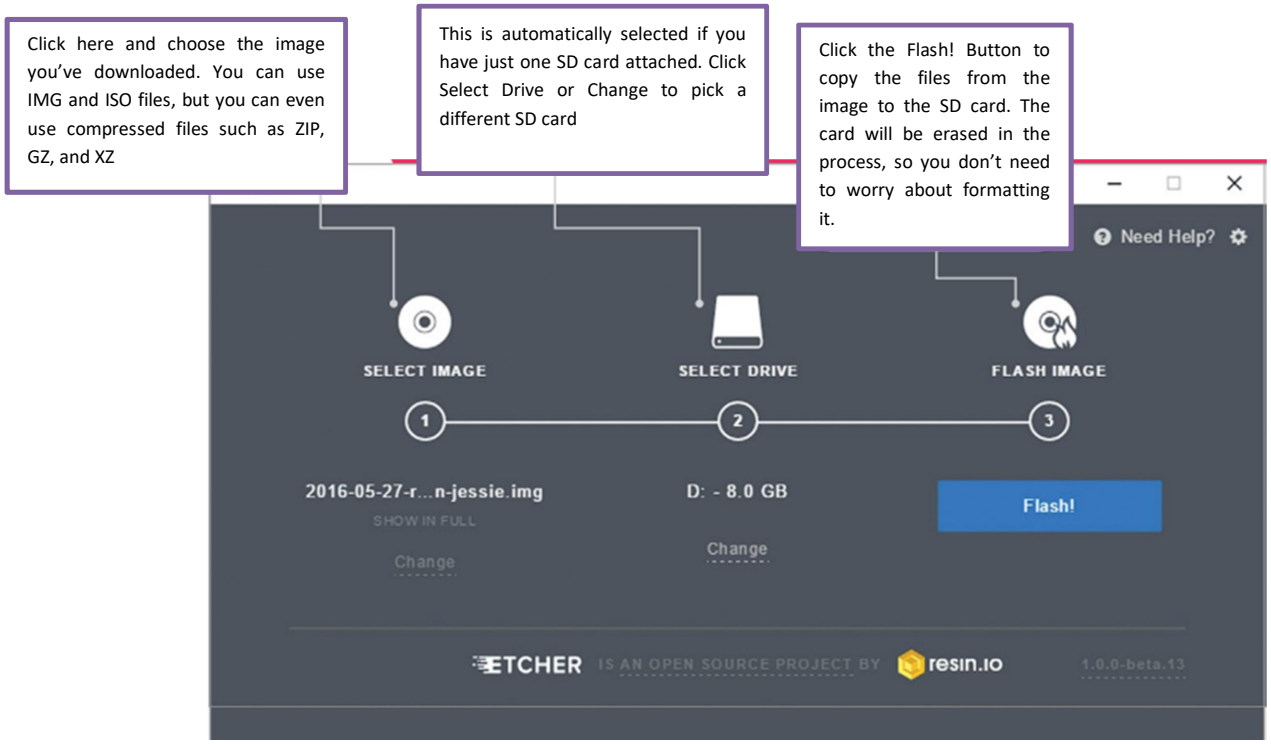
➢ **Setting up your Raspberry Pi for the first time:-**

Getting your Raspberry Pi up and running is a very easy process, just follow these simple steps:

1. Insert an SD card with a Linux operating system on it. We strongly recommend using Raspbian or another OS designed specifically for the Raspberry Pi, at least until you are more familiar with its capabilities.

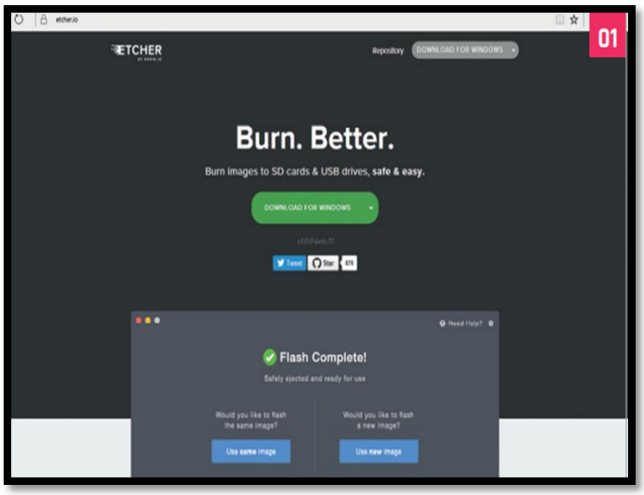
CREATE SD CARDS WITH ETCHER

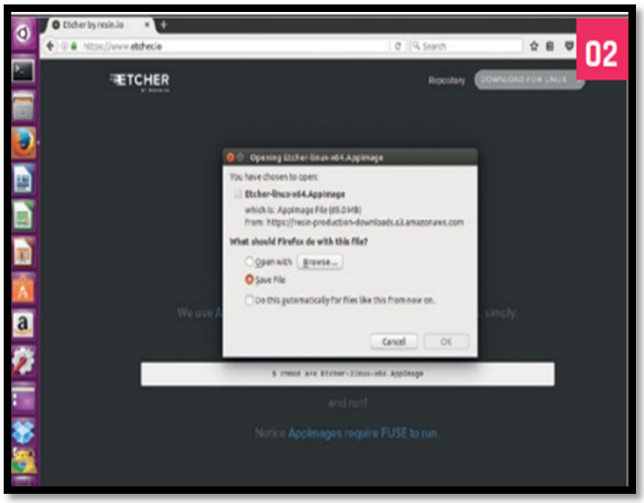
Etcher takes a lot of the stress out of flashing a drive. Etcher won't write to your hard drive volumes unless you check Unsafe Mode in Settings. Unsafe Mode is handy if you want to flash a USB thumb drive or other internal drive, but it's disabled by default, making the process safer for newcomers. We like Etcher so much, we thought we'd create this guide to installing and using it. Follow these steps for hassle-free SD card flashing.

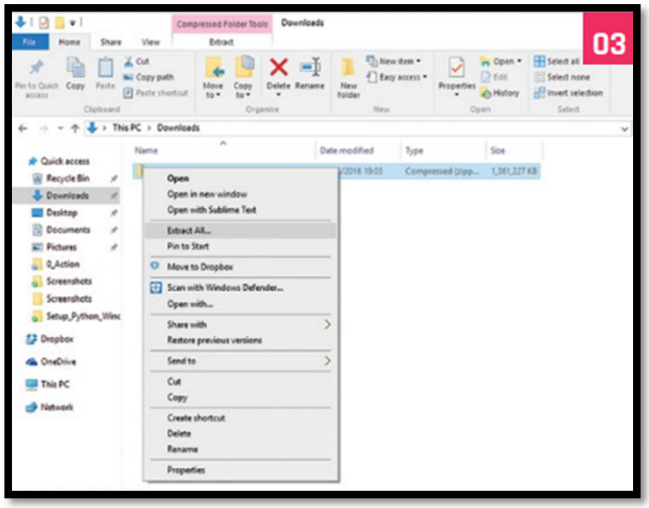


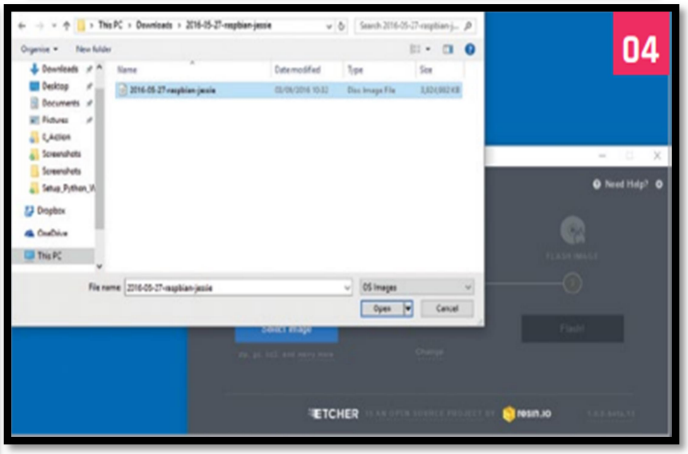
- ***STEP-01:*** Install in Windows or Mac Download and install Etcher from the **etcher.io** website.
- ***STEP-02:*** Install on Linux Download the AppImage file from the Etcher website. Open a terminal window and enter**: cd Downloads chmod a+x Etcher-linux-x64. AppImage**

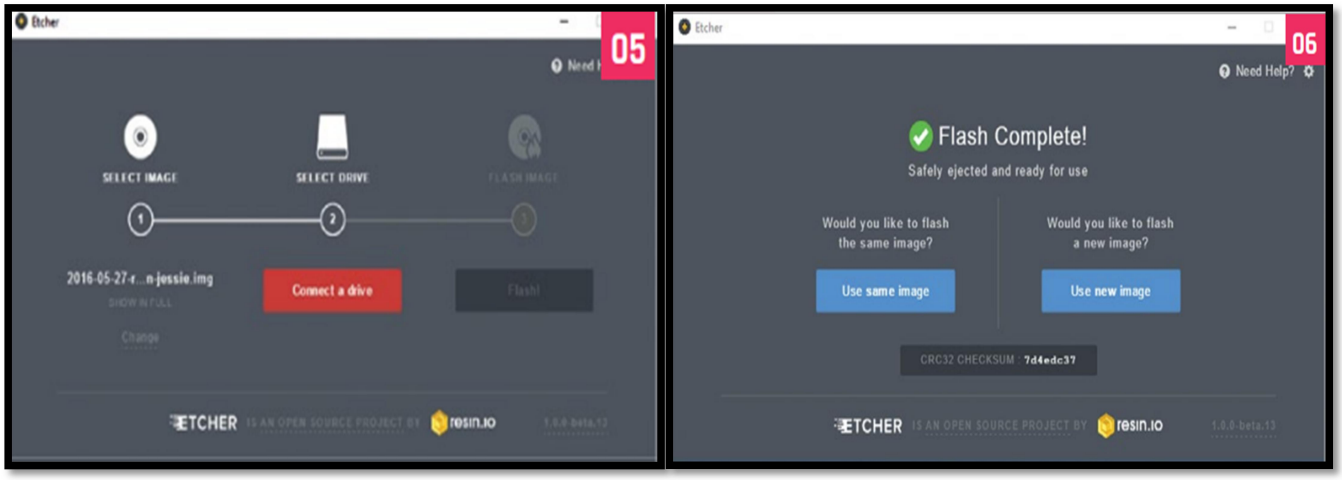
  ./Etcher-linux-x64.AppImage
- ***STEP-03:*** Download your OS image Download a copy of the latest Raspbian image from **raspberrypi.org/downloads** (or the OS image you want to install). Unzip the file after it has downloaded. Double-click the file in Mac or Linux (or use unzip in a terminal window). In Windows, right-click the file and choose Extract All. Etcher can install directly from a ZIP file, but the process takes a lot longer.
- ***STEP-04:*** Select the image Click Select Image in Etcher. Use the file manager window and locate the image you unzipped in the previous step. Click Open. The image will appear under Select Image, and Connect a drive will highlight red.
- ***STEP-05***: Insert your SD card Attach your SD card to the computer. Etcher will select it automatically. Etcher won't write to your hard drives by default, but check that the SD card is listed correctly. Now click Flash! to write the image file to the SD card.
- ***STEP-06:*** Writing the image Etcher will format the SD card, before writing and verifying the image; this is shown by a progress bar. When done, remove the SD card, insert it into your Raspberry Pi, and power it up. If you want to flash another SD card with the same image, insert it and click Use Same Image.







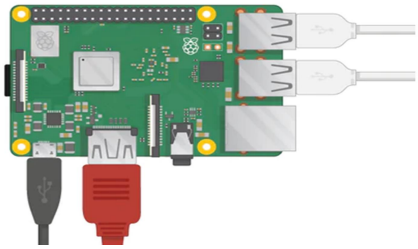




2. Connect a monitor to the HDMI port. Make sure the monitor is connected to a wall socket and is powered on. The Raspberry Pi is still off, so you won't see anything yet. If you are connecting a non-HDMI monitor, use an adapter that won't block access to the USB ports.

3. Connect a mouse and keyboard to the USB ports. High-end equipment such as 'gaming' mice can drain power, put unnecessary load on the processor and may need software and drivers to work properly. To avoid complications, stick to simple 'plug-and-play' devices.

   Your Raspberry Pi should now look like this:



4. If you want to connect to the Internet via Ethernet instead of WiFi, you should also plug in an Ethernet cable. If you want to hear sound through speaker or headphones, you should also plug those in now. Neither of these is necessary to start up the Raspberry Pi.

5. Finally, connect the power supply to a wall socket and plug in the micro USB cable. A red LED will light up on the Raspberry Pi, and on the monitor you will see the Raspberry Pi booting up. In a few moments you will see a desktop screen.

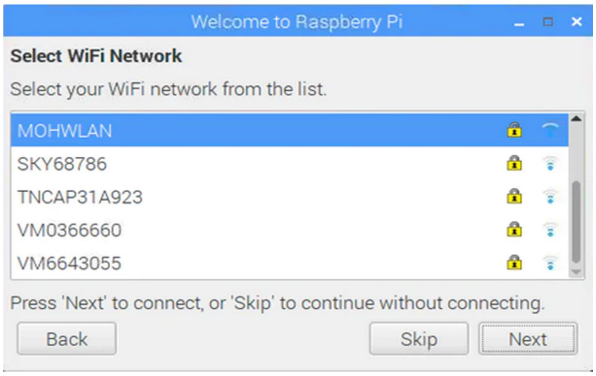
Final steps

If this is the first time you have loaded up your Raspberry Pi, you will see a welcome screen asking you to provide some basic settings.

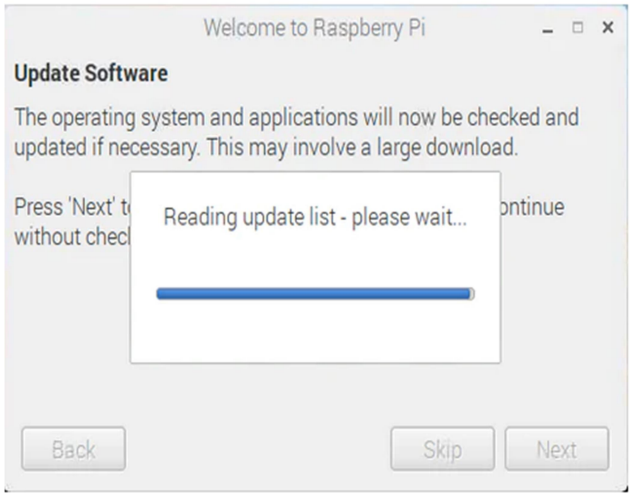
Click Next to begin:

- Choose your Country, Language and Time zone. You can also set whether you are using a US keyboard layout. Click **Next** to continue.
- Next you will be asked to create a password. The default password is 'raspberry' and unless you want anyone to be able to log in, you should definitely change this. Click **Next** to continue.

- On the next screen the Raspberry Pi will display a list of the wireless networks it can pick up. If your model does not have WiFi, this screen will not appear and you can skip this step. Select your network, enter your WiFi password and click **Next**.

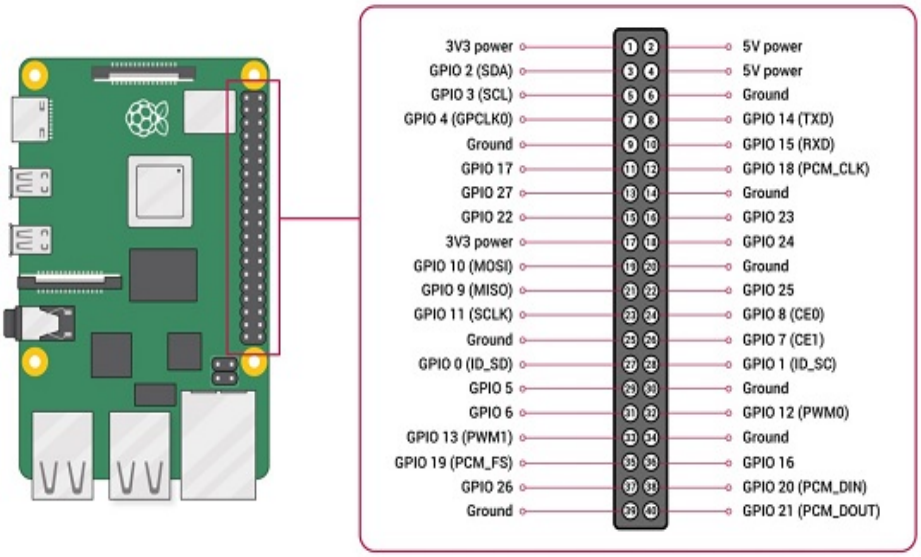


- The Raspberry Pi will now check for any software updates, and download newer versions if available. If you don't have WiFi or did not set up in the previous screen, but still want to check for updates, plug in an Ethernet cable before starting. The Raspberry Pi might prompt you to reboot it after this stage, in order to complete an update it installed.
- Raspberry Pi is now set up and ready to use!



Raspberry Pi 3 GPIO Pinout / Pin diagram:

The Processor used in Raspberry Pi 3:
Broadcom BCM2837: It is a 1.2GHz 64bit ARM quad-core Cortex A53 processor, with 512 KiB shared L2 cache, dual-core VideoCore IV GPU @ 400 MHz supporting OpenGL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode.



A powerful feature of the **Raspberry Pi** is the row of **GPIO** (general-purpose input/output) pins along the extreme right edge of the board. Like every Raspberry Pi chipset, it consists of a 40-pin GPIO. A standard interface for connecting a single-board computer or microprocessor to other devices is through General-Purpose Input/output (GPIO) pins. GPIO pins do not have a specific function and can be customized using the software.

Raspberry Pi 3 Power Pins:
The board consists of two 5V pins, two 3V3 pins, and 9 ground pins (0V), which are unconfigurable.

5V: The 5v pins directly deliver the 5v supply coming from the mains adaptor. This pin can use to power up the Raspberry Pi and it can also use to power up other 5v devices.

3.3V: The 3v pin is there to offer a stable 3.3v supply to power components and to test LEDs.

GND: Ground is commonly referred to as GND. All the voltages are measured with respect to the GND voltage.

Raspberry Pi 3 Input/Outputs pins:

A GPIO pin that is set as an **input** will allow a signal to be received by the Raspberry Pi that is sent by a device connected to this pin. A voltage between 1.8V and 3.3V will be read by the Raspberry Pi as HIGH and if the voltage is lower than 1.8V will be read as LOW.

Note: Do not connect a device with an input voltage above 3.3V to any of the GPIO pins, or else it will fry the Raspberry Pi.

A GPIO pin set as an **output** pin sends the voltage signal as high (3.3V) or low (0V). When this pin is set to HIGH, the voltage at the output is 3.3V and when set to LOW, the output voltage is 0V.

Along with the simple function of input and output pins, the GPIO pins can also perform a variety of alternative functions. Some specific pins are:

PWM (pulse-width modulation) pins:

- Software PWM is available on all pins
- Hardware PWM is available on these pins only: GPIO12, GPIO13, GPIO18, GPIO19

SPI pins:

SPI (Serial Peripheral Interface) is another protocol used for master-slave communication. It is used by the Raspberry pi board to quickly communicate between one or more peripheral devices. Data is synchronized using a clock (SCLK at GPIO11) from the master (RPi) and the data is sent from the Pi to our SPI device using the MOSI (Master Out Slave In) pin. If the SPI device needs to communicate back to Raspberry Pi, then it will send data back using the MISO (Master In Slave Out) pin.
There are 5 pins involved in SPI communication:

- **GND**: Connect all GND pins from all the slave components and the Raspberry Pi 3 board together.
- **SCLK**: Clock of the SPI. Connect all SCLK pins together.
- **MOSI**: It stands for Master Out Slave In. This pin is used to send data from the master to a slave.
- **MISO**: It stands for Master In Slave Out. This pin is used to receive data from a slave to the master.
- **CE**: It stands for Chip Enable. We need to connect one CE pin per slave (or peripheral devices) in our circuit. By default, we have two CE pins but we can configure more CE pins from the other available GPIO pins.

SPI pins on board:
SPI0: GPIO9 (MISO), GPIO10 (MOSI), GPIO11 (SCLK), GPIO8 (CE0), GPIO7 (CE1)
SPI1: GPIO19 (MISO), GPIO20 (MOSI), GPIO21 (SCLK), GPIO18 (CE0), GPIO17 (CE1), GPIO16 (CE2)

I2C pins:
I2C is used by the Raspberry Pi board to communicate with devices that are compatible with Inter-Integrated Circuit (a low-speed two-wire serial communication protocol). This communication standard requires master-slave roles between both devices. I2C has two connections: **SDA (Serial Data)** and **SCL (Serial Clock)**. They work by sending data to and using the SDA connection and the speed of data transfer is controlled via the SCL pin.

- **Data**: (GPIO2), Clock (GPIO3)
- **EEPROM Data**: (GPIO0), EEPROM Clock (GPIO1)

UART Pins:
Serial communication or the UART (Universal Asynchronous Receiver / Transmitter) pins provide a way to communicate between two microcontrollers or the computers. TX pin is used to transmit the serial data and RX pin is used to receive serial data coming from a different serial device.

- TX (GPIO14)
- RX (GPIO15)

II. A program to interface PIR sensor with Arduino and turn ON LED when motion is detected.

PIR sensors allow you to sense motion. They are small, inexpensive, low-power, easy to use and don't wear out. For that reason they are commonly found in appliances and gadgets used in homes or businesses.

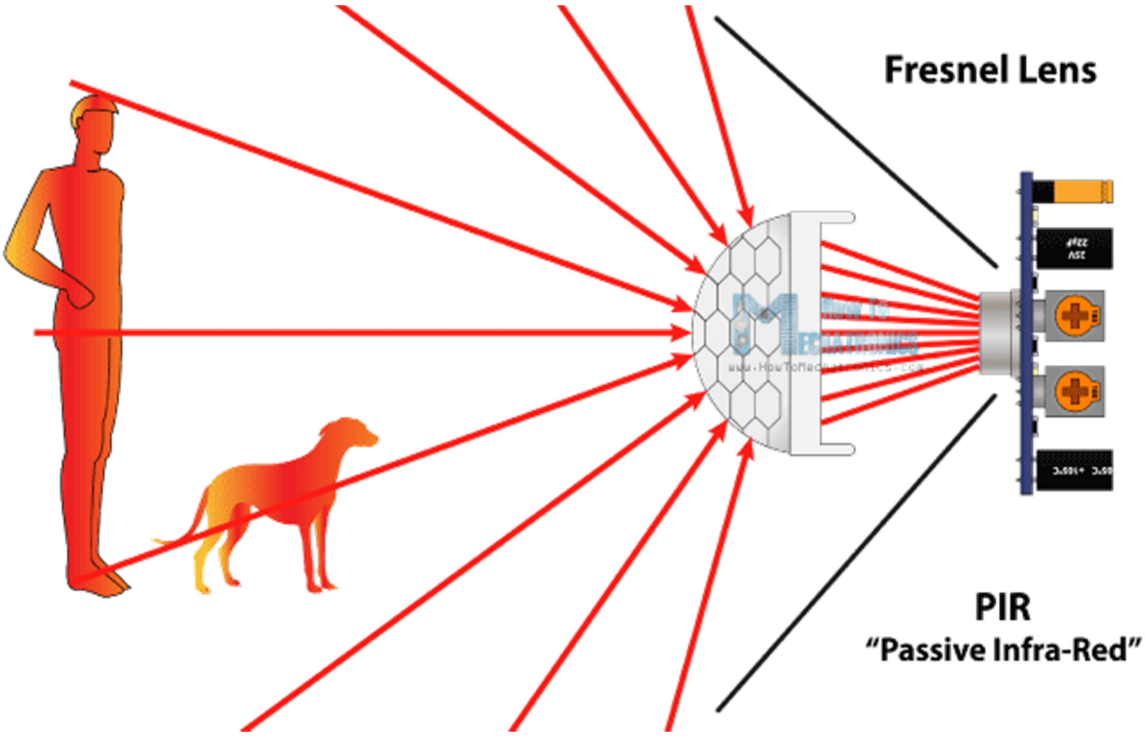
What is a PIR Sensor?

A **passive infrared sensor** is an electronic sensor that measures infrared light radiating from objects. PIR sensors mostly used in PIR-based motion detectors. Also, it used in security alarms and automatic lighting applications. The below image shows a typical pin configuration of the PIR sensor, which is quite simple to understand the pinouts. The PIR sensors consist of 3 pins.

- Pin1 corresponds to the drain terminal of the device, which connected to the positive supply 5V DC.
- Pin2 corresponds to the source terminal of the device, which connects to the ground terminal via a 100K or 47K resistor. The Pin2 is the output pin of the sensor. The pin 2 of the sensor carries the detected IR signal to an amplifier from the
- Pin3 of the sensor connected to the ground.

What does a PIR Sensor detect?

Generally, PIR sensor can detect animal/human movement in a requirement range. PIR is made of a pyroelectric sensor, which is able to detect different levels of infrared radiation. The detector itself does not emit any energy but passively receives it.

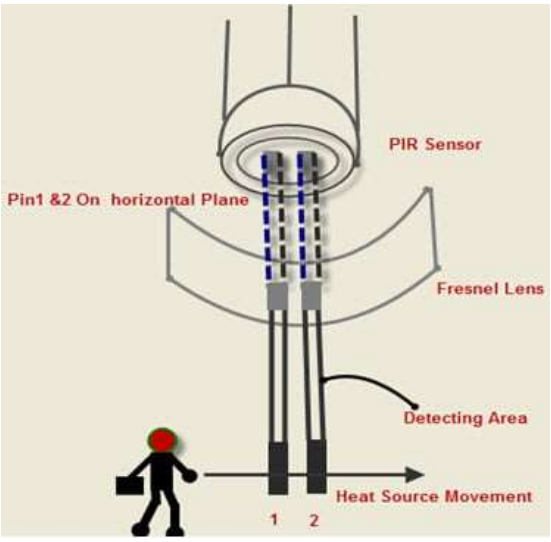


It detects infrared radiation from the environment. Once there is infrared radiation from the human body particle with temperature, focusing on the optical system causes the pyroelectric device to generate a sudden electrical signal.

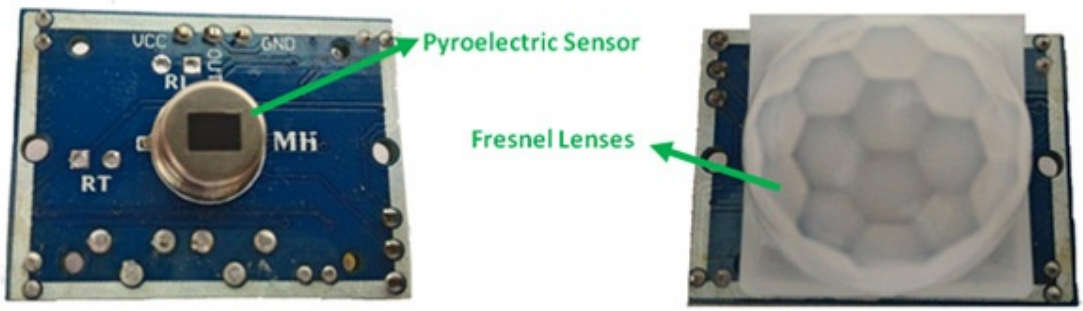
Simply, when a human body or any animal passes by, then it intercepts the first slot of the PIR sensor. This causes a positive differential change between the two bisects. When a human body leaves the sensing area, the sensor generates a negative differential change between the two bisects.

PIR Sensor Working Principle

The passive infrared sensor does not radiate energy to space. It receives the infrared radiation from the human body to make an alarm. Any object with temperature is constantly radiating infrared rays to the outside world. The surface temperature of the human body is between 36° C - 27 ° C and most of its radiant energy concentrated in the wavelength range of 8 um-12 um.



Passive infrared alarms classified into **infrared detectors** (infrared probes) and alarm control sections. The most widely used infrared detector is a pyroelectric detector. It uses as a sensor for converting human infrared radiation into electricity. If the human infrared radiation is directly irradiated on the detector, it will, of course, cause a temperature change to output a signal. But in doing all this, the detection distance will not be more. In order to lengthen the detection distance of the detector, an optical system must be added to collect the infrared radiation. Usually, plastic optical reflection system or plastic Fresnel lens used as a focusing system for infrared radiation.



In the detection area, the lens of the detector receives the infrared radiation energy of the human body through the clothing and focused on the pyroelectric sensor. When the human body moves in this surveillance mode, it enters a certain field of view in sequence and then walks out of the field of view. The pyroelectric sensor sees the moving human body for a while and then does not see it, so the infrared radiation of human body constantly changes the temperature of the pyroelectric material. So that it outputs a corresponding signal, which is the alarm signal.

What is the Range of PIR Sensor?

- Indoor passive infrared: Detection distances range from 25 cm to 20 m.
- Indoor curtain type: The detection distance ranges from 25 cm to 20 m.
- Outdoor passive infrared: The detection distance ranges from 10 meters to 150 meters.
- Outdoor passive infrared curtain detector: distance from 10 meters to 150 meters

Components Required

You will need the following components −

- 1 × Breadboard
- 1 × ESP8266 Board
- 1 × PIR Sensor
- 1 x Red LED

❖ **Arduino Code for PIR Sensor by Using ESp8266 Board**

```
int RedLed = 12;                    // Digital pin D6
int sensor = 5;                     // Digital pin D1

void setup() {                      // put your setup code here, to run once:
  pinMode(sensor, INPUT);           // declare sensor as input
  pinMode(RedLed, OUTPUT);          // declare LED as output
  Serial.begin(115200);
}

void loop()  {                              // put your main code here, to run repeatedly:
  long state = digitalRead(sensor);
    if(state == HIGH)   {
      digitalWrite (RedLed, HIGH);
      Serial.println("Motion detected, Led Glow ON Stage");
      delay(1000);
                          }
    else    {
      digitalWrite (RedLed, LOW);
      Serial.println("Motion absent! Led Glow OFF Stage");
      delay (1000);
            }
                  }
```

Result:-

You will see a message on your serial Monitor port if a motion is detected and another message when the motion stops.

Like's the following...

Motion detected, Led Glow ON Stage
Motion detected, Led Glow ON Stage
Motion absent! Led Glow OFF Stage

Code to Note:

PIR sensor has three terminals - V_{cc}, OUT and GND. Connect the sensor as follows: −

- Connect the $+V_{cc}$ to +5v on ESP8266 board.
- Connect OUT to digital pin 5 on ESP8266 board.
- Connect GND with GND on ESP8266.

III. Write a program to interface DHT22 sensor with Arduino and display temperature and humidity readings.

The DHT22 is a commonly used Temperature and humidity sensor. The sensor comes with a dedicated NTC to measure temperature and an internal circuit to output the values of temperature and humidity as serial data. The sensor is also factory calibrated and hence easy to interface with other microcontrollers.

The sensor can measure temperature from -40°C to 80°C and humidity from 0% to 100% with an accuracy of ±1°C and ±1%. So if you are looking to measure in this range then this sensor might be the right choice for you.

How does the DHT22 work?

The DHT22 consist of a humidity sensing component, a NTC temperature sensor (or thermistor) and an IC on the back side of the sensor.

For measuring humidity they use the humidity sensing component which has two electrodes with moisture holding substrate between them. So as the humidity changes, the conductivity of the substrate changes or the resistance between these electrodes changes. This change in resistance is measured and processed by the IC which makes it ready to be read by a microcontroller.

On the other hand, for measuring temperature these sensors use a NTC temperature sensor or a thermistor. A thermistor is actually a variable resistor that changes its resistance with change of the temperature. These sensors are made by sintering of semi conductive materials such as ceramics or polymers in order to provide larger changes in the resistance with just small changes in temperature. The term “NTC” means “Negative Temperature Coefficient”, which means that the resistance decreases with increase of the temperature.

Wiring

In this example we will have one ESP8266 connected to a DHT22 temperature sensor. The DHT22 has 4 pins: VCC, Data, NC (not connected, no function) and GND. The GND of both devices need to be wired together. VCC can be either the 3V3 pin (3.3V output) or Vin (5V output), both on the ESP8266 or an external voltage source. As the DHT22 can operate at voltages between 3.3-6V, we connect this sensor to the ESP8266's Vin pin in this example. The data pin of the DHT22 is connected to GPIO pin 5 of the ESP8266 in this example.

Connect your DHT22 to your Node MCU 1.0 as follows:

1. Connect 3V3 (3.3V pin) or Vin (5V pin) of the Node MCU to the Vcc pin of the DHT22
2. Digital pin D1 (GPIO 5) to the second pin of the DHT22
3. Connect a 5K resistor data pin and VCC pin of the DHT22.
4. Leave the third DHT22 pin unconnected
5. Connect the fourth pin of the DHT22 to ground of ESP8266.

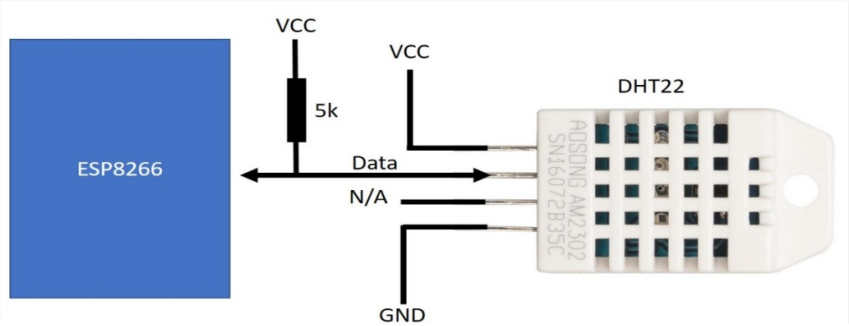
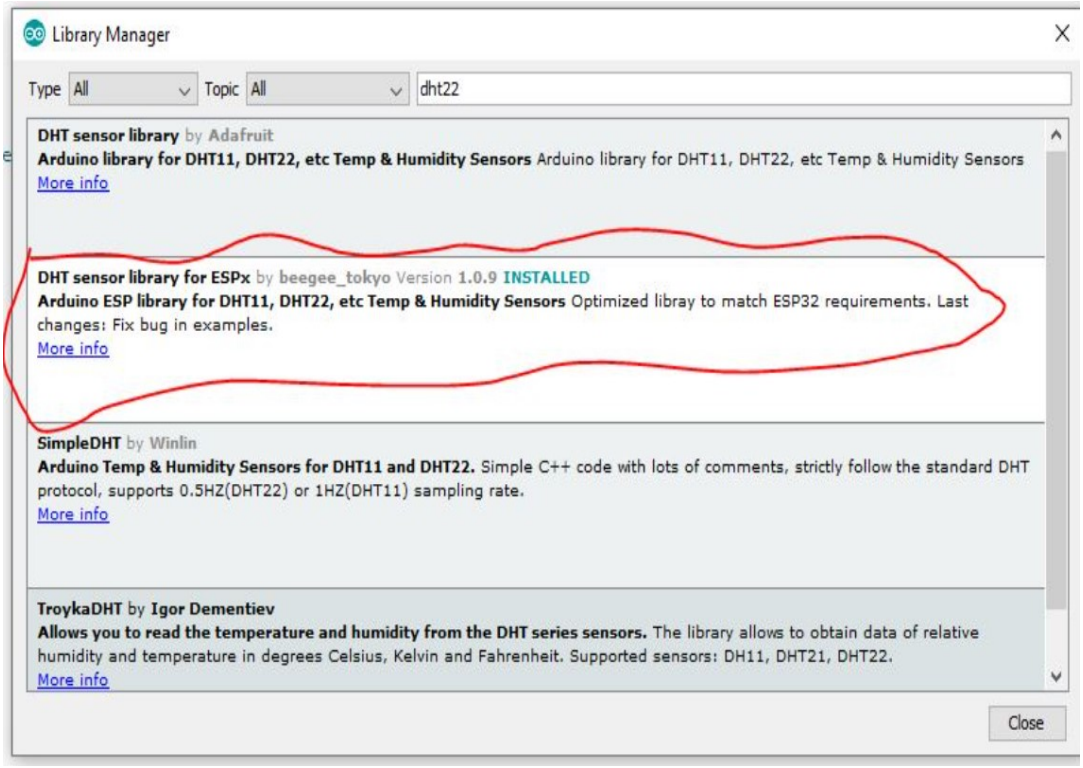


Image: Wiring of DHT22 with ESP8266. The Data line requires a 5k pull-up

ESP8266 Code

First, we start by including the libraries needed for all the functionality. We need the **DHT sensor library for ESPx**, in order to be able to get temperature and humidity data from the DHT22. This library can be installed via **Arduino IDE library manager**.



Components Required

You will need the following components −

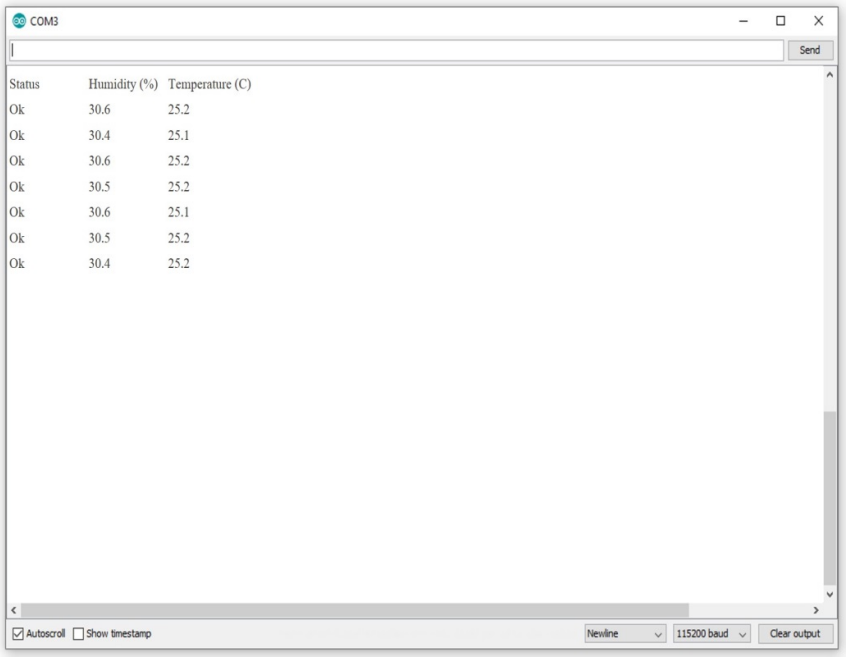
- 1 × Breadboard
- 1 × ESP8266 Board
- 1 × DHT22 Sensor
- 1 x 5k Resistor

❖ **Arduino Code for DHT22 Sensor by Using ESp8266 Board**

```
#include "DHTesp.h"
DHTesp dht;

void setup()
{
 Serial.begin(115200);
 Serial.println();
 Serial.println("Status\tHumidity (%)\tTemperature (C)");

 // Autodetect is not working reliable, don't use the following line
 // dht.setup(17);
 // use this instead:
 dht.setup(5, DHTesp::DHT22); // Connect DHT sensor to GPIO 5
}
void loop()
{
 delay(2000);
 float humidity = dht.getHumidity();
 float temperature = dht.getTemperature();
 Serial.print(dht.getStatusString());
 Serial.print("\t");
 Serial.print(humidity, 1);
 Serial.print("\t\t");
 Serial.println(temperature, 1);

}
```

Result:

After uploading the code to the board, open up your serial monitor, set the baud rate to 115200, and you should see something like the following:



IV. A program to interface motor with Raspberry Pi. Turn ON motor when the temperature is high.

PYTHON CODE FOR RASPBERRY PI DHT11/DHT22 & STEPPER MOTOR.

Install the DHT python library. This is done by entering the following command:

```
sudo pip3 install Adafruit_DHT
```

Note: If you run into problems with the above command, you may not have PIP installed on your Pi. You can fix that by running the following commands. These will install PIP and other utilities you may need.

```
sudo apt-get install python3-dev python3-pip
sudo python3 -m pip install --upgrade pip setuptools wheel
```

Now let's take a look at the code we're going to use. This is some very basic code written in Python. The first section of code imports the DHT library from Adafruit and the system time library.

```
import Adafruit_DHT
import RPi.GPIO as GPIO
import time

channel = (29,31,33,35)
DHT_SENSOR = Adafruit_DHT.DHT11
DHT_PIN = 4
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(channel, GPIO.OUT)

while True:
   try:
      humidity, temperature = Adafruit_DHT.read(DHT_SENSOR, DHT_PIN)

       if( temperature >=25) :
          print("Temp={0:0.1f}C ".format(temperature))
          GPIO.output(channel, (GPIO.HIGH,GPIO.LOW,GPIO.LOW,GPIO.HIGH))
          sleep(0.02)
          GPIO.output(channel, (GPIO.HIGH,GPIO.HIGH,GPIO.LOW,GPIO.LOW))
          sleep(0.02)
          GPIO.output(channel, (GPIO.LOW,GPIO.HIGH,GPIO.HIGH,GPIO.LOW))
          sleep(0.02)
          GPIO.output(channel, (GPIO.LOW,GPIO.LOW,GPIO.HIGH,GPIO.HIGH))
          sleep(0.02)

      else:
         print("Sensor failure. Check wiring...")
         time.sleep(3)

   except KeyboardInterrupt:
      GPIO.cleanup()
```

V. A program to interface LCD with Raspberry Pi and print temperature and humidity readings on it.

The DHT11 sensor can measure relative humidity and temperature with the following specifications.

S.No	Parameter	Specification
1	Temperature Range	0-50°C
2	Temperature Accuracy	±2 °C
3	Humidity Range	20-90% RH
4	Humidity Accuracy	±5 %

➢ **Installing the Adafruit LCD Library on Raspberry Pi.**

Step 1: Install git on your Raspberry Pi by using the below line.

sudo apt-get install git

Step 2: The following line links to the GitHub page where the library is present just execute the line to clone the project file on Pi home directory

git clone git://github.com/adafruit/Adafruit_Python_CharLCD

Step 3: Use the below command to change directory line, to get into the project file that we just downloaded. The command line is given below

cd Adafruit_Python_CharLCD

Step 4: Inside the directory there will be a file called setup.py, we have to install it, to install the library. Use the following code to install the library

sudo python setup.py install

➢ **Installing the Adafruit DHT11 Library on Raspberry Pi.**

The DHT11 library provided by Adafruit can be used for DHT11, DHT22 and other one wire temperature sensors as well. The procedure to install the DHT11 library is also similar to the one followed for installing LCD library. The only line that would change is the link of the GitHub page on which the DHT library is saved.

Enter the four command lines one by one on the terminal to install the DHT library

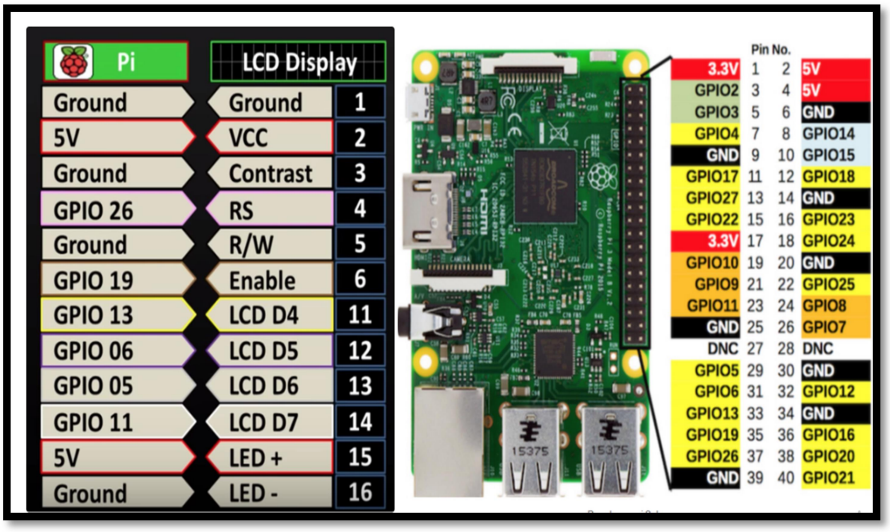
git clone https://github.com/adafruit/Adafruit_Python_DHT.git

cd Adafruit_Python_DHT

sudo apt-get install build-essential python-dev

sudo python setup.py install

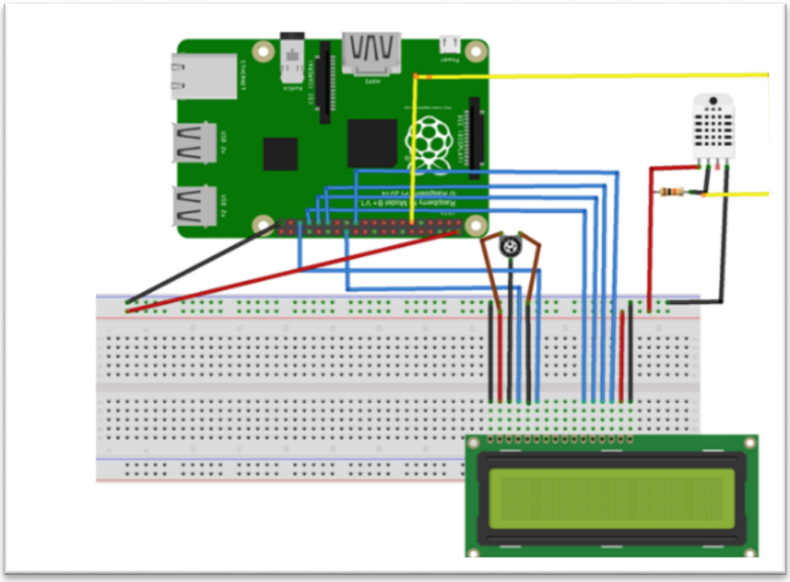
❖ **Circuit Diagram**



The DHT11 Module comes in 3 pins, Connect the Vcc to 5V on the pi, connect the ground pin to any ground pin on the pi and connect the data pin to your choice GPIO pin on the pi, in this tutorial we're using GPIO 17 which is pin number 11 on the pi.

NOTE: The DHT11 comes in Module or sensor type, the one shown in the schematic below is the sensor type that has 4 pins, a resistor is connected between the data pin and the Vcc, if you're using the module type with only 3 pins, there's no need for the resistor.

Reference the diagram below for the pinout of the raspberry pi pins.



Since the LCD will be using the two 5V available on the pi, we can use a breadboard to share the 5V between the LCD and the DHT11 Module. The LCD pins will be connected to the pi in the following order. Note that pin 7, 8, 9 and 10 of the LCD will not be used.

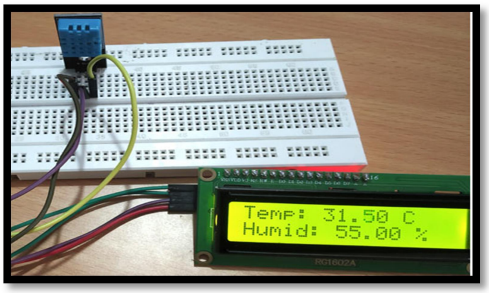
❖ **A Program for Raspberry Pi to get temperature/humidity from DHT11 sensor with LCD Display.**

```
import Adafruit_DHT

from Adafruit_CharLCD import Adafruit_CharLCD

sensor = Adafruit_DHT.DHT11

pin = 17

humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)

lcd = Adafruit_CharLCD(rs=26, en=19, d4=13, d5=6, d6=5, d7=11, cols=16, lines=2)


#DISPLAY A STATIC TEXT

lcd.clear()

if humidity is not None and temperature is not None:

    print('Temp={0:0.1f}*C  Humidity={1:0.1f}%'.format(temperature, humidity))

    lcd.message('Temp={0:0.1f}*C  \nHumidity={1:0.1f}%'.format(temperature, humidity))
```

```
else:

    print('Failed to get reading. Try again!')

    lcd.message('Failed to get reading. Try again!')
```

Output:

```
Temp: 31.50C
Humidity: 55.00%
Temp: 31.50C
Humidity: 55.00%
Temp: 31.50C
Humidity: 55.00%
```

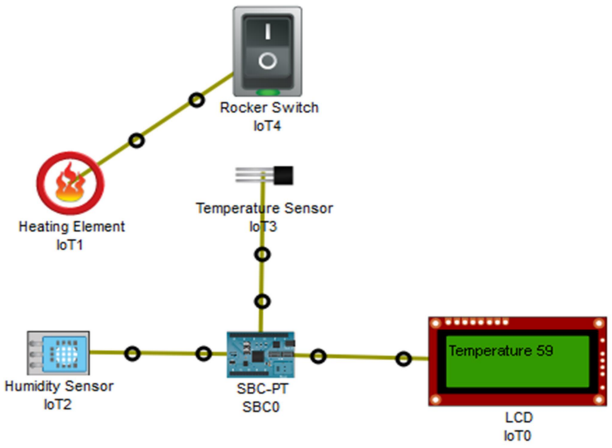


❖ **A program to interface LCD with Packet Tracer and print temperature and humidity readings on it.**

```
from gpio import *
from time import *

def main():
        pinMode(0,INPUT)
        pinMode(1, INPUT)
        pinMode(2, OUT)

        while True:
                hum=analogRead(1);
                temp=analogRead(0);
                delay(1000)
                customWrite(2,"Humidity "+str(hum)+" F")
                delay(2000)
                customWrite(2,"Temperature "+str(temp)+" C")
                delay(1000)

if __name__ == "__main__":
        main()
```

Output:

```
Humidity: 745 %
Temperature: 59 C
```



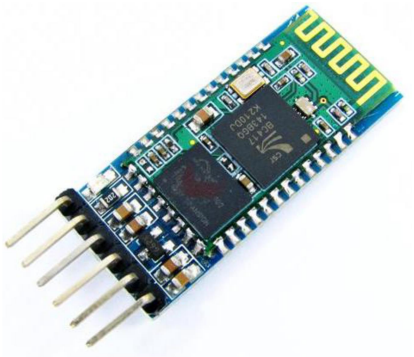
VI. To send sensor data to smart phone using Bluetooth.

***Bluetooth Module HC-05*:**

- It is used for many applications like wireless headset, game controllers, wireless mouse, wireless keyboard and many more consumer applications.
- It has range up to <100m which depends upon transmitter and receiver, atmosphere, geographic & urban conditions.
- It is IEEE 802.15.1 standardized protocol, through which one can build wireless Personal Area Network (PAN). It uses frequency-hopping spread spectrum (FHSS) radio technology to send data over air.
- It uses serial communication to communicate with devices. It communicates with microcontroller using serial port (USART).

HC-05 is a Bluetooth module which is designed for wireless communication. This module can be used in a master or slave configuration.

HC-05 Bluetooth Module



Bluetooth serial modules allow all serial enabled devices to communicate with each other
Using Bluetooth... It has 6 pins,

1. Key/EN: It is used to bring Bluetooth module in AT commands mode. If Key/EN pin is set to high, then this module will work in command mode. Otherwise by default it is in data mode. The default baud rate of HC-05 in command mode is 38400bps and 9600 in data mode.

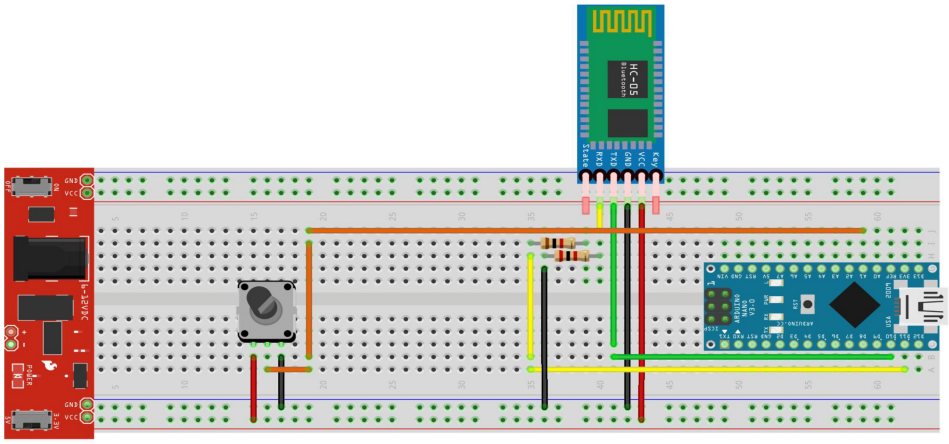
HC-05 module has two modes,

1. Data mode: Exchange of data between devices.
2. Command mode: It uses AT commands which are used to change setting of HC-05. To send these commands to module serial (USART) port is used.
3. VCC: Connect 5 V or 3.3 V to this Pin.
4. GND: Ground Pin of module.
5. TXD: Transmit Serial data (wirelessly received data by Bluetooth module transmitted out serially on TXD pin)
6. RXD: Receive data serially (received data will be transmitted wirelessly by Bluetooth module).
7. State: It tells whether module is connected or not.

HC-05 module Information

- HC-05 has red LED which indicates connection status, whether the Bluetooth is connected or not. Before connecting to HC-05 module this red LED blinks continuously in a periodic manner. When it gets connected to any other Bluetooth device, its blinking slows down to two seconds.
- This module works on 3.3 V. We can connect 5V supply voltage as well since the module has on board 5 to 3.3 V regulators.
- As HC-05 Bluetooth module has 3.3 V level for RX/TX and microcontroller can detect 3.3 V level, so, no need to shift transmit level of HC-05 module. But we need to shift the transmit voltage level from microcontroller to RX of HC-05 module.

➢ **Circuit Diagram:-**

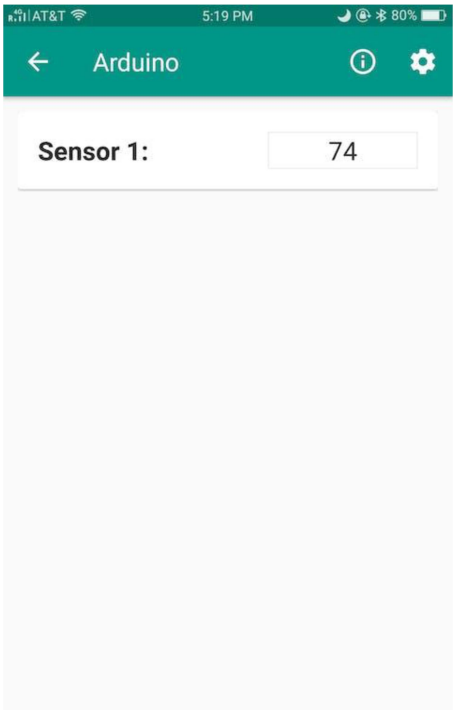


❖ **A program to send sensor data to smart phone using Bluetooth.**

```
#include <SoftwareSerial.h>
SoftwareSerial  bluetooth(10, 11); //RX, TX
int input;
int device = A1;
void setup( )
{
Serial.begin(9600);
bluetooth.begin(9600);
pinMode(device, INPUT);
}
void loop( )
{
input = analogRead(device);
input = map(input, 0, 1023, 0, 180);
bluetooth.print(input);
bluetooth.print(";");
Serial.println(input);
delay(20);
}
```

Result analysis:

1. Power the Arduino.
2. Now, the hc-05 module should blink rapidly.
3. Next, Open the app.
4. Allow it to access Bluetooth settings.
5. In the list, select hc-05.
6. Select receiver mode.
7. Now the module should blink once every 2 seconds.
8. Here, click the link
9. It will load for some time now.
10. Afterwards, this is what we will see



VII. To interface flame/smoke sensor with Arduino /Raspberry Pi and give an alert message when flame/smoke is detected.

MQ2 is one of the commonly used gas sensors in MQ sensor series. It is a Metal Oxide Semiconductor (MOS) type Gas Sensor also known as Chemiresistors as the detection is based upon change of resistance of the sensing material when the Gas comes in contact with the material. Using a simple voltage divider network, concentrations of gas can be detected.

MQ2 Gas sensor works on 5V DC and draws around 800mW.
It can detect LPG, Smoke, Alcohol, Propane, Hydrogen, Methane and Carbon Monoxide concentrations anywhere from 200 to 10000ppm.

What is 1 ppm equal to?

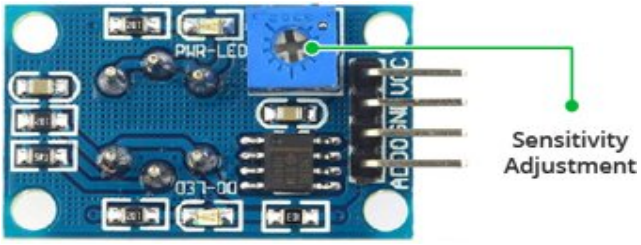
When measuring gases like carbon dioxide, oxygen, or methane, the term concentration is used to describe the amount of gas by volume in the air. The 2 most common units of measurement are parts-per-million, and percent concentration.

Note:-

The sensor is sensitive to multiple gasses – but cannot tell which it is! That's normal; most gas sensors are like that. So, it is best for measuring changes in a known gas density, not detecting which is changing.

Calibrate MQ2 Gas Sensor Module

To calibrate the gas sensor you can hold the gas sensor near smoke/gas you want to detect and keep turning the potentiometer until the Red LED on the module starts glowing. Turn the screw clockwise to increase sensitivity or anticlockwise to decrease sensitivity.

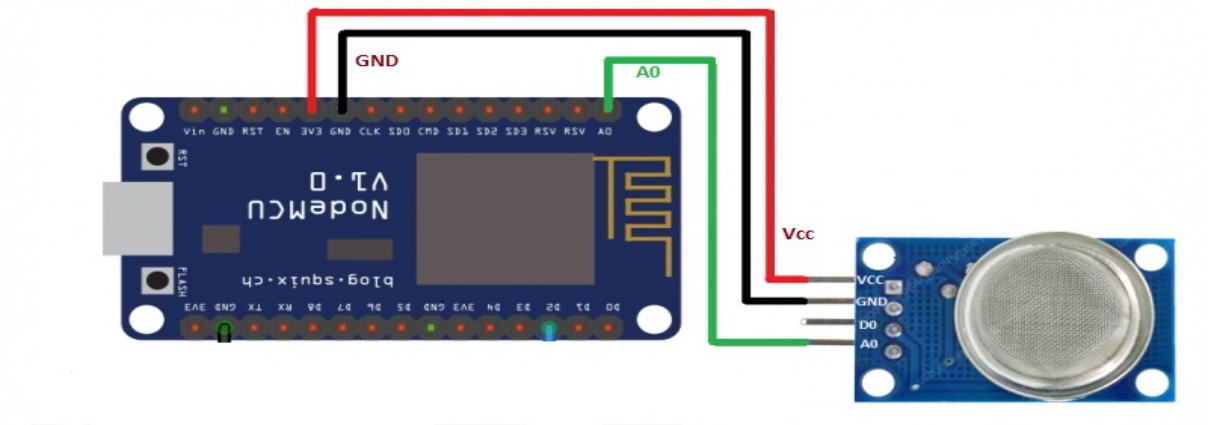


The comparator on the module continuously checks if the analog pin (A0) has hit the threshold value set by potentiometer. When it crosses the threshold, the digital pin (D0) will go HIGH and signal LED turns on.

MQ2 Gas Sensor Module Pinout

➢ **Circuit Diagram:-**

Connecting MQ2 Gas Sensor Module to Node MCU-ESP8266



❖ **A program to interface flame/smoke sensor with Arduino and give an alert message when flame/smoke is detected.**

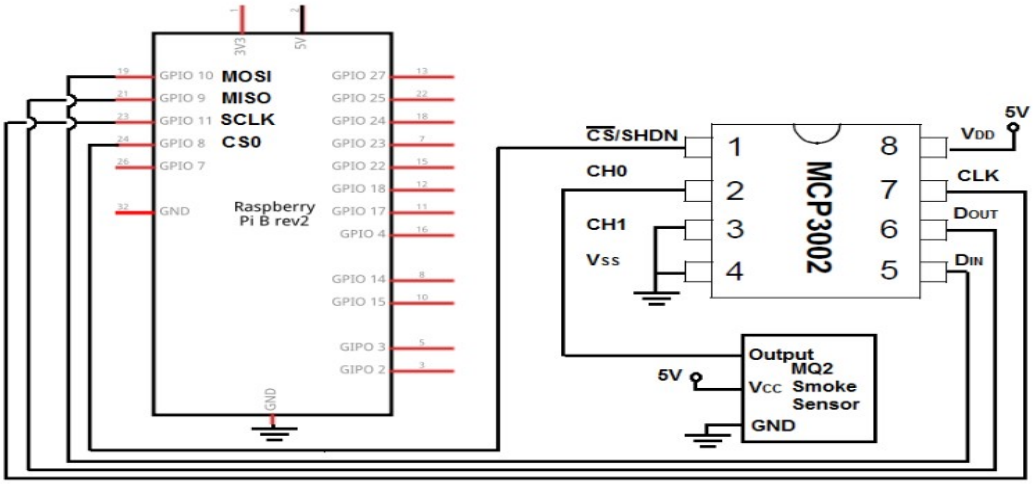
```
int smokeA0 = A0;
int sensorThres = 600; // your threshold value. You might need to change it.

void setup() {
  pinMode(smokeA0, INPUT);
 Serial.begin(115200);
}
void loop() {

int analogSensor = analogRead(smokeA0);

 Serial.println(analogSensor);

 if (analogSensor > sensorThres)  // Checks if it has reached the threshold value

 {
   Serial.println("Smoke is detected ");
 }
 else
 {
   Serial.println(" No Smoke is detected ");
 }
 delay(1000);
}
```

Output:-

Smoke is detected
Smoke is detected
No Smoke is detected

➢ **Circuit Diagram:-**

Connecting MQ2 Gas Sensor Module to Raspberry pi



❖ **A program to interface flame/smoke sensor with Raspberry Pi and give an alert message when flame/smoke is detected.**

- *In the first block of code, the first 2 lines, import libraries into the code. The botbook.com library for **MCP3002** is one that saves a lot of coding and makes it easy to read analog values. This above file that you have just written must be in the same directory as the botbook_mcp3002.py library. You must also install the spidev library, which is imported by botbook_mcp3002.*
- *In our next line, we create a variable named Smoke Level and initialize it to 0. Just as the name implies, it will hold the value of the smoke level, which will be the analog voltage output by the smoke sensor.*
- *The next block is our main code. This repeats the program forever, which his common for embedded devices. When you use While True, it's recommended that you add some delay at the end of the loop. This gives the loop some time to pause before executing again, kind of like a breather. All that's needed is just a few milliseconds of delay. In our code, we give it 0.5s, which is 500 milliseconds. In our main code, we call the **readSmokeLevel ()** function, which allows us to read the smoke sensor value, and then we output this value. If the smoke Level is above 120, then we output the line, "**Smoke detected**."*

```
import time
import botbook_mcp3002 as mcp
smokeLevel= 0
def readSmokeLevel():
global smokeLevel
smokeLevel= mcp.readAnalog()

def main():
while True:
readSmokeLevel()
print ("Current smoke level is %i " % smokeLevel)
if smokeLevel > 120:
print("Smoke detected")
time.sleep(0.5)
if_name_=="_main_":
main()
```

Output:-

Current smoke level is 135

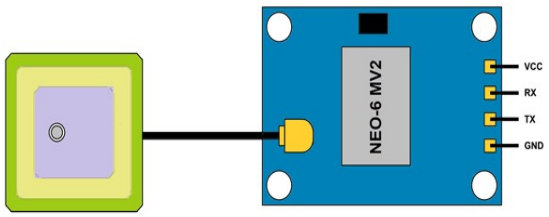
Smoke detected

VIII. GPS module by interfacing with Arduino.

NEO-6M GPS Module: GPS stands for global positioning system and can be used to determine position, time and speed if you are travelling.

1. The NEO-6MV2 is a standalone GPS receiver that's used for navigation. The GPS receiver links with GPS satellites to get its own location. It, then, outputs the latitude and longitude of its position as serial data.
2. The NEO-6 module is based on a 50-channel, ublox-6 positioning engine that boasts a time-to-first-fix (TTFF) of one second. This GPS engine has two million correlators and is capable of massive parallel time/frequency space searches. This enables it to find satellites instantly. The module also has a small form-factor that makes it ideal for battery-operated mobile devices.
3. The NEO-6M GPS modem operates on a supply voltage of 2.7 to 3.6V. It communicates GPS data according to the NMEA or UBX protocol. While NMEA is a standard ASCII protocol, UBX is a u-blox proprietary binary protocol.
4. The receiver chipset has UART, USB, SPI, and DDC (I2C-compliant) interfaces to communicate this data. The chipset has three configuration pins. One of these configuration pins — CFG-GPS0 — is used to enable boot configuration of the power mode.

The module only exposes four channels as shown in this pin diagram:



The NEO-6M GPS module has four pins: VCC, RX, TX, and GND. The module communicates with the Arduino via serial communication using the TX and RX pins, so the wiring couldn't be simpler:

NEO-6M GPS Module Wiring to Node MCU

VCC VIN

RX TX pin defined in the software serial

TX RX pin defined in the software serial

GND GND

Download and install required libraries for GPS to work in Arduino IDE

(i) SoftwareSerial library

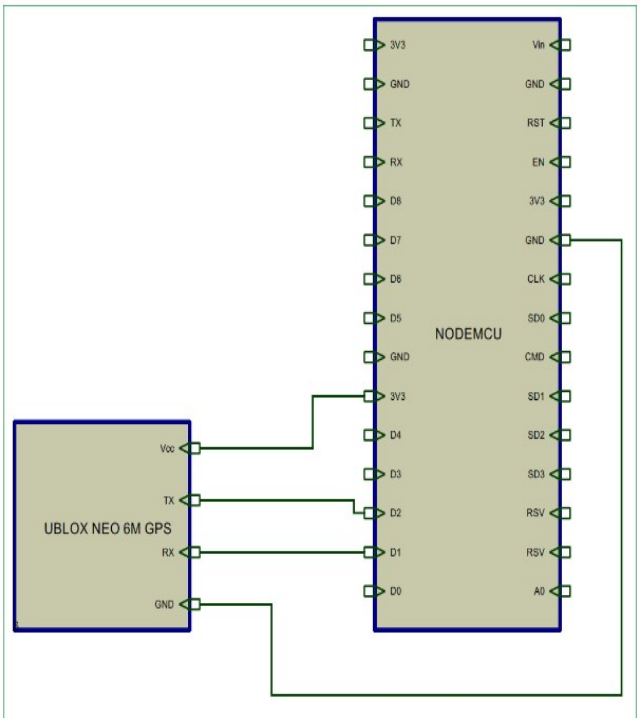
(ii) TinyGPS library

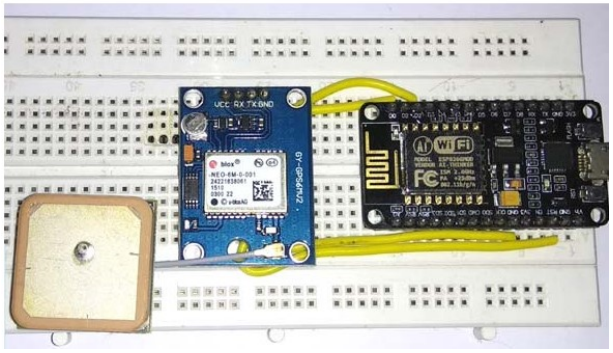
❖ **Perform experiment with GPS module by interfacing with Arduino.**

```
#include <TinyGPS++.h>

#include <SoftwareSerial.h>

#include <ESP8266WiFi.h>

TinyGPSPlus gps;          // The TinyGPS++ object

SoftwareSerial  ss(4, 5); // The serial connection to the GPS device

const char* ssid = "myssid";
```

```
const char* password = "12345678";
float latitude , longitude;
String  lat_str , lng_str;
WiFiServer server(80);
void setup()
{
 Serial.begin(115200);
 ss.begin(9600);
 Serial.println();
 Serial.print("Connecting to ");
 Serial.println(ssid);
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED)
 {
  delay(500);
  Serial.print(".");
 }
 Serial.println("");
 Serial.println("WiFi connected");
 server.begin();
 Serial.println("Server started");
 Serial.println(WiFi.localIP());  // Print the IP address
}
void loop()
{
  while (ss.available() > 0)
    if (gps.encode(ss.read()))
```

// ***Circuit Diagram:-***





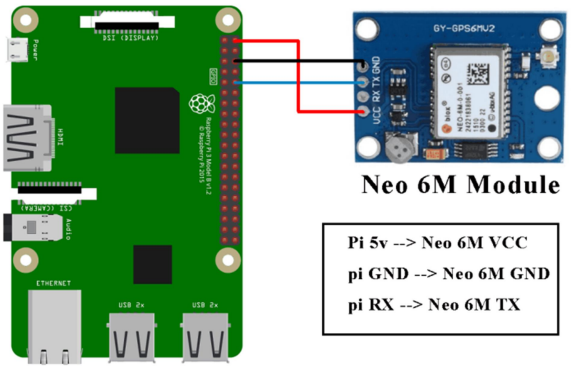
```
  {
   if (gps.location.isValid())
   {
    latitude = gps.location.lat();

    lat_str = String(latitude , 6);

    longitude = gps.location.lng();

    lng_str = String(longitude , 6);

   Serial.println("Gps Latitude : ", lat_str);
   Serial.println("Gps longitude: ",lng_str);
    }
   }
}
```

Output:-

Gps Latitude : 12.8908

Gps longitude: 77.6581

❖ **Perform experiment with GPS module by interfacing with Raspberry pi.**



Neo 6M VCC -----> Raspberry pi 5v
Neo 6M GND -----> Raspberry pi GND
Neo 6M RX -----> Raspberry pi TX (gpio 14) //Not required in this case.
Neo 6M TX -----> Raspberry pi RX (gpio 15)

Getting data from the GPS module:

Now here we need to modify few things. At first we need to edit the /boot/config.txt file. Now you need to open this file in any text editor. Here using nano tool editor:

sudo nano /boot/config.txt

At the end of the file add the following lines:

dtparam=spi=on
dtoverlay=pi3-disable-bt
core_freq=250
enable_uart=1
force_turbo=1 // Now save this by typing ctrl +x, then type y and press enter.

- Raspbian uses the UART as a serial console and so we need to turn off that functionality. To do so we need to change the /boot/cmdline.txt file. For safety before editing the file make a backup of that using the following command:

sudo cp /boot/cmdline.txt /boot/cmdline_backup.txt

Now to edit that file open that in text editor:

sudo nano /boot/cmdline.txt

Replace the content with the following line (delete everything in it and write down the following content):

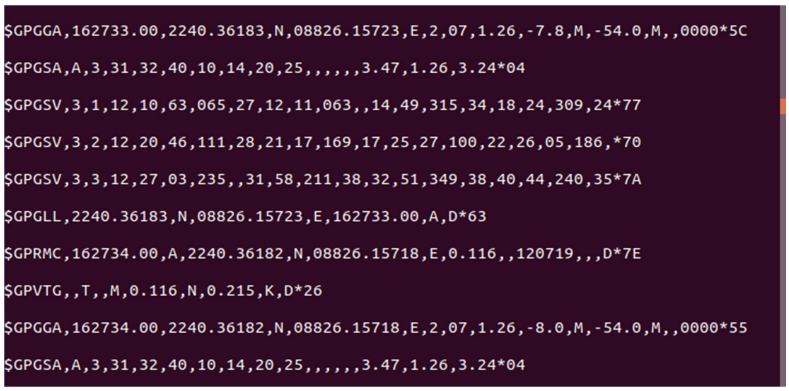
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles

Now save this by typing ctrl +x, then type y and press enter.

sudo reboot

After the reboot now it's time to check how GPS module is working.

When the blue led is blinking, run the following command:
sudo cat /dev/ttyAMA0
Now you will see a lots of data like shown in the below image. That basically means that it's working. To stop this type **Ctrl + c**



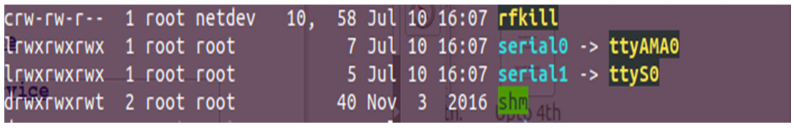
1. By default the Raspberry Pi uses serial port for this "console" login so if we want to use the serial port to get data from the GPS module we need to disable the console login. Now there are two serial ports in Raspberry pi 3: serial0 and serial1. But in-between them serial0 will point to GPIO pins 14 and 15, so we have to use serial 0.

2. Now to see which port is connected with serial0 use the following command:

   ls -l /dev

There are two possible outputs:

- **If your output looks like this:**

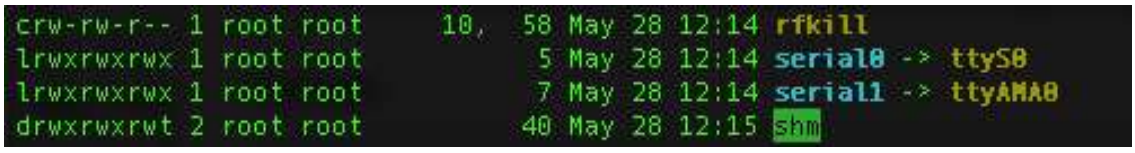


As you can see serial0 is linked with ttyAMA0. So to disable the console you need to use the following commands:

sudo systemctl stop serial-getty@ttyAMA0.service

sudo systemctl disable serial-getty@ttyAMA0.service

- **But if your output looks like this:**



That means serial0 is linked with ttyS0. So to disable the console you need to use the following commands:

sudo systemctl stop serial-getty@ttyS0.service

sudo systemctl disable serial-getty@ttyS0.service

```
import serial                          #import serial pacakge
from time import sleep
import webbrowser                      #import package for opening link in browser
import sys                             #import system package

def GPS_Info():
    global NMEA_buff
    global lat_in_degrees
    global long_in_degrees
    nmea_time = []
    nmea_latitude = []
    nmea_longitude = []
    nmea_time = NMEA_buff[0]               #extract time from GPGGA string
    nmea_latitude = NMEA_buff[1]           #extract latitude from GPGGA string
    nmea_longitude = NMEA_buff[3]          #extract longitude from GPGGA string

    print("NMEA Time: ", nmea_time,'\n')
    print ("NMEA Latitude:", nmea_latitude,"NMEA Longitude:", nmea_longitude,'\n')

    lat = float(nmea_latitude)             #convert string into float for calculation
    longi = float(nmea_longitude)          #convertr string into float for calculation

    lat_in_degrees = convert_to_degrees(lat)    #get latitude in degree decimal format
    long_in_degrees = convert_to_degrees(longi) #get longitude in degree decimal format

#convert raw NMEA string into degree decimal format
def convert_to_degrees(raw_value):
    decimal_value = raw_value/100.00
    degrees = int(decimal_value)
    mm_mmmm = (decimal_value - int(decimal_value))/0.6
    position = degrees + mm_mmmm
    position = "%.4f" %(position)
    return position

gpgga_info = "$GPGGA,"
ser = serial.Serial ("/dev/ttyS0")          #Open port with baud rate
GPGGA_buffer = 0
NMEA_buff = 0
lat_in_degrees = 0
long_in_degrees = 0

try:
    while True:
        received_data = (str)(ser.readline())              #read NMEA string received
        GPGGA_data_available = received_data.find(gpgga_info)   #check for NMEA GPGGA string
        if (GPGGA_data_available>0):
            GPGGA_buffer = received_data.split("$GPGGA,",1)[1]  #store data coming after "$GPGGA," string
```

```
        NMEA_buff = (GPGGA_buffer.split(','))           #store comma separated data in buffer
        GPS_Info()                                      #get time, latitude, longitude

        print("lat in degrees:", lat_in_degrees," long in degree: ", long_in_degrees, '\n')
        map_link = 'http://maps.google.com/?q=' + lat_in_degrees + ',' + long_in_degrees     #create link to plot location on Google map
        print("<<<<<<<<press ctrl+c to plot location on google maps>>>>>>\n")               #press ctrl+c to plot on map and exit
        print("----------------------------------------------------------\n")

except KeyboardInterrupt:
   webbrowser.open(map_link)       #open current position information in google map
   sys.exit(0)
```

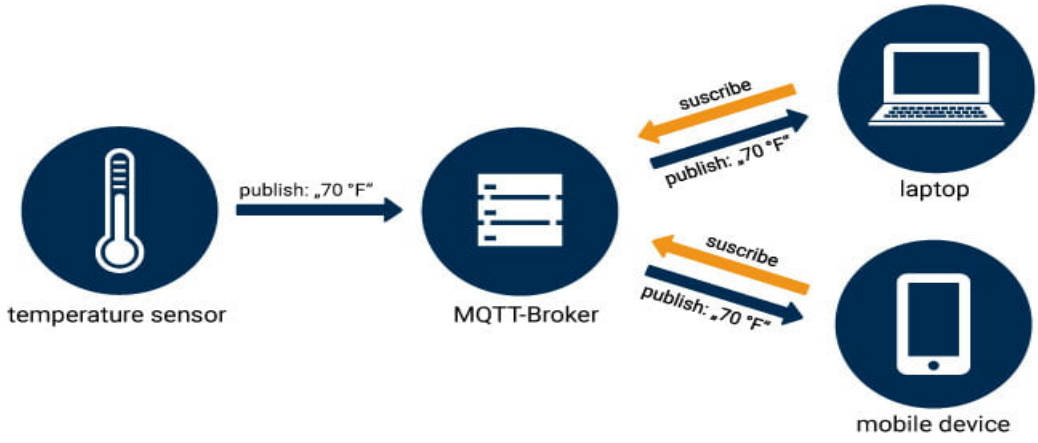
Output:-

IX. To send and receive messages using MQTT protocol.

MQTT (Message Queuing Telemetry Transport) is a messaging protocol for restricted low-bandwidth networks and extremely high-latency IoT devices. Since Message Queuing Telemetry Transport is specialized for low-bandwidth, high-latency environments, it is an ideal **protocol** for machine-to-machine (M2M) communication.

MQTT works on the publisher / subscriber principle and is operated via a central broker. This means that the sender and receiver have no direct connection. The data sources report their data via a publish and all recipients with interest in certain messages ("marked by the topic") get the data delivered because they have registered as subscribers.

What is a MQTT broker?



The **MQTT broker** is the center of every Publish / Subscribe protocol. Depending on the implementation, a broker can manage up to thousands of simultaneously connected **MQTT clients.**

- The broker is responsible for receiving all messages, filtering the messages, determining who subscribed to each message and sending the message to those subscribed clients.
- The Broker also holds the sessions of all persistent clients, including subscriptions and missed messages.
- Another task of the Broker is the authentication and authorization of clients.

In short, the Broker is the central hub through which every message must be routed. It is therefore important that your broker is highly scalable, can be integrated into back-end systems, is easy to monitor and, of course, is fail-safe.

What is a MQTT payload?

Messages are shared with other devices or software via a broker using MQTT. Every message has a topic, based on which the message can be processed further by the Broker. Additionally, each message contains message content, the so-called payload.

However, it is helpful to specify a particular structure for the message content so that it can be read by other devices or software. Potential message structures are **JSON, XML or OPC UA.**

MQTT client

All devices and software, such as the OPC Router, that are connected to the broker in some way are referred to as MQTT clients. A client can send messages to the broker (publish) and receive messages from the broker (subscribe). When sending a message to the broker, an MQTT topic must be specified, which can be used to further process the message. Messages can be sent with different Quality of Service (QoS):

- Quality of Service 0: The client's message is sent exactly once, regardless of whether it has arrived at the broker.
- Quality of Service 1: The client's message is sent over and over again until the broker responds with an confirmation of receipt. This can result in a message arriving at the broker multiple times.
- Quality of Service 2: The client sends a message once and simultaneously ensures that it has arrived at the broker. Quality of Service 2 communication requires more bandwidth than Quality of Service 0 or 1.

Finally, an MQTT client has the "Last Will" function. If the connection to the broker is lost, a last message is sent so that the connection error is noticed by the broker and can be passed on to the user.

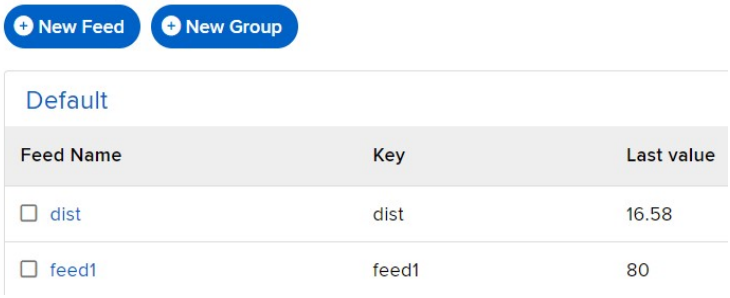
❖ **A program to send and receive messages using MQTT protocol.**

```
#include <ESP8266WiFi.h>
#include "Adafruit_MQTT.h"      //using Adafruit_MQTT_protocal.
#include "Adafruit_MQTT_Client.h"

/************************* WiFi Access Point *********************************/
#define WLAN_SSID       "myssid"
#define WLAN_PASS       "1234567890"
/************************* Adafruit.io Setup *********************************/
#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    "adafruit_name"
#define AIO_KEY         "aio_key"
/************ Global State (you don't need to change this!) ******************/
// Create an ESP8266 WiFiClient class to connect to the MQTT server.
WiFiClient client;
// or... use WiFiFlientSecure for SSL
//WiFiClientSecure client;
// Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
//****************************** Feeds ***************************************//
// Setup a feed called 'distance' & 'feed1' for publishing & Subscribe.

Adafruit_MQTT_Publish distance = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/dist");
Adafruit_MQTT_Subscribe feed1 = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/feed1");
/*************************** Sketch Code ************************************/
// sample example of Mqtt by using Ultrasonic sensor (Distance measurement).
const int trigPin = 5;  //D1
const int echoPin = 4; //D2
define sound velocity in cm/uS
#define soundbuzzer 14
#define SOUND_VELOCITY 0.034
#define CM_TO_INCH 0.393701
int sound =500;
long duration;
```

```
float distanceCm;
float distanceInch;
void MQTT_connect();
//********************************** Setup ******************************************//
void setup() {
  Serial.begin(115200);
  pinMode(trigPin, OUTPUT);  // Sets the trigPin as an Output
  pinMode(echoPin, INPUT);  // Sets the echoPin as an Input
  pinMode(soundbuzzer, OUTPUT);
  delay(10);
  Serial.println(F("Adafruit MQTT demo"));
  // Connect to WiFi access point.
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(WLAN_SSID);
  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("try to connecting please wait.....");
  }
  Serial.println();
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  mqtt.subscribe(&feed1);
}
uint32_t x=0;
uint8_t slider1=0;
void loop() {
  // Ensure the connection to the MQTT server is alive (this will make the first
  // connection and automatically reconnect when disconnected).  See the MQTT_connect
  // function definition further below.
  MQTT_connect();
   //*********************Subscription**********************//
  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(5000))) {
    if (subscription == &feed1) {
       Serial.println("feed1:");
       Serial.println((char *)feed1.lastread);
           }
    }
    //*****************Publishing*****************************//
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
    // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
    // Calculate the distance
  distanceCm = duration * SOUND_VELOCITY/2;
    // Convert to inches
```

```
  distanceInch = distanceCm * CM_TO_INCH;
   // Prints the distance on the Serial Monitor
  Serial.print("Distance (cm): ");
  //For Cloud Monitoring data
  distance.publish(distanceCm);
  // For Serial monitor data
  Serial.println(distanceCm);
  Serial.print("Distance (inch): ");
  Serial.println(distanceInch);
 if(distanceCm >15)
  {                    //if distance is 15cm or above value, then buzzer tone, else no buzzer tone.
    tone(soundbuzzer, sound);
    delay(1000);
    noTone(soundbuzzer);
    delay(1000);
    }
   else
   {    noTone(soundbuzzer);
   }
  delay(2000);
}
// Function to connect and reconnect as necessary to the MQTT server.
// Should be called in the loop function and it will take care if connecting.
void MQTT_connect() {
  int8_t ret;
  // Stop if already connected.
  if (mqtt.connected()) {
    return;
  }
  Serial.print("Connecting to MQTT... ");
  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) {            // connect will return 0 for connected
       Serial.println(mqtt.connectErrorString(ret));
       Serial.println("Retrying MQTT connection in 5 seconds...");
       mqtt.disconnect();
       delay(5000);  // wait 5 seconds
       retries--;
       if (retries == 0) {
         // basically die and wait for WDT to reset me
         while (1);
       }
  }
  Serial.println("MQTT Connected!");
}
```

Output: - *//For Cloud monitoring data*



Adafruit MQTT demo
Connecting to myssid
WiFi connected
IP address: 192.168.10.221
Connecting to MQTT
MQTT Connected!
// Prints the distance on the Serial Monitor
Distance (cm): 16.58
Distance (inch): 6.52
feed1: 80

Note: - //dist feed is Publishing
// feed1 feed is Subscription

//========== * ==========//

X. To upload sensor data to local/cloud server using Wi-Fi.

The **Arduino IoT Cloud** is a platform that allows **anyone** to create IoT projects, with a user friendly interface, and an all in one solution for **configuration, writing code, uploading** and **visualization.**

Basically, this is the getting started about Arduino IoT Cloud where we will learn about using the Arduino IoT Cloud with ESP8266.

Features

Below is a list of Arduino IoT Cloud features.

- **Data Monitoring** - learn how to easily monitor your Arduino's sensor values through a dashboard.
- **Variable Synchronization** - variable synchronization allows you to sync variables across devices, enabling communication between devices with minimal coding.
- **Scheduler** - schedule jobs to go on/off for a specific amount of time (seconds, minutes, hours).
- **Over-The-Air (OTA) Uploads** - upload code to devices not connected to your computer.
- **Webhooks** - integrate your project with another service, such as IFTTT.
- **Amazon Alexa Support** - make your project voice controlled with the Amazon Alexa integration.
- **Dashboard Sharing** - share your data with other people around the world.

Compatible Hardware

To use the Arduino IoT Cloud, a cloud compatible board is required. You can choose between using an official Arduino board, or a board based on the ESP32 / ESP8266 microcontroller. The Arduino IoT Cloud currently supports connection via Wi-Fi, LoRaWAN® (via The Things Network) and mobile networks.

ESP32 / ESP8266

The Arduino IoT Cloud supports a wide range of third party boards based on the ESP32 and ESP8266 microcontrollers with support for Wi-Fi. To set them up, simply choose the **third party option** in the device setup.

Setting up the Arduino IoT Cloud and accessing the different features available involves a few simple steps.

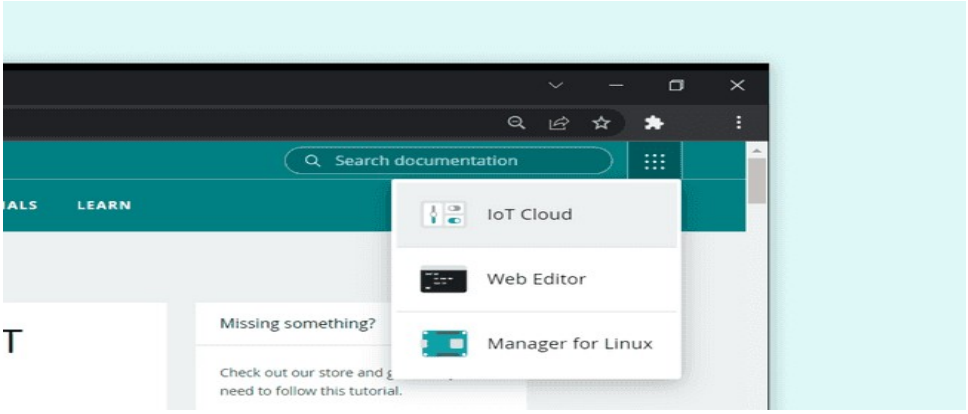
1. Creating an Arduino Account

To starting using the Arduino IoT cloud, we first need to log in or sign up to Arduino.

2. Go to the Arduino IoT Cloud

After we have signed up, you can access the Arduino IoT Cloud from any page on arduino.cc by clicking on the four dots menu in the top right corner. You can also go directly to the Arduino IoT Cloud.

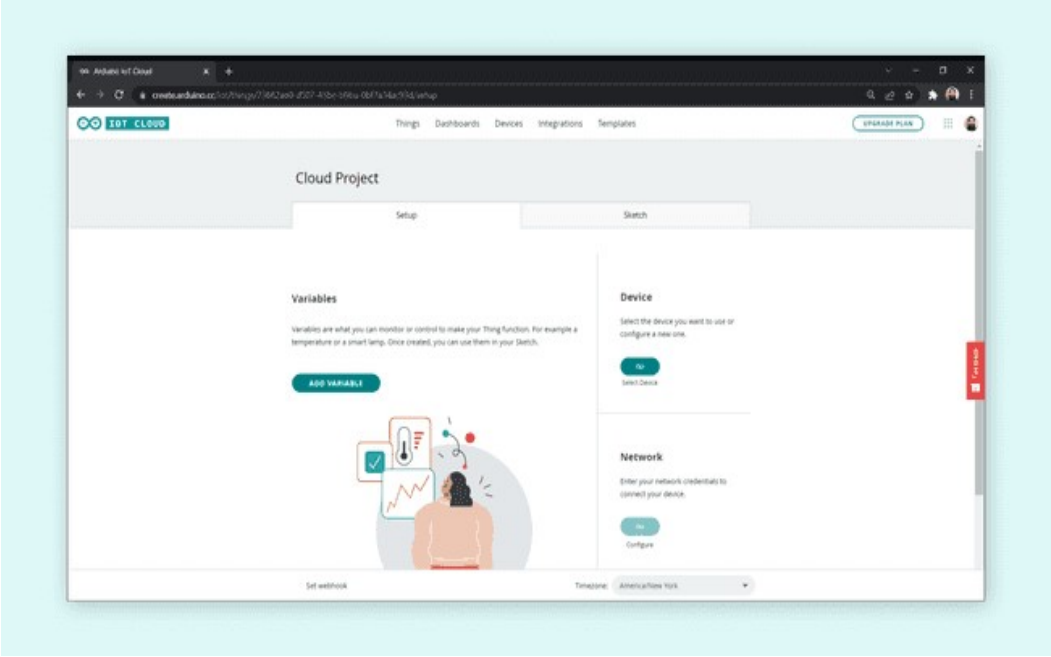
Navigating to the cloud



3. Creating a Thing

The journey always begins by creating a new **Thing**. In the Thing overview, we can choose what device to use, what Wi-Fi network we want to connect to, and create variables that we can monitor and control. This is the main configuration space, where all changes we make are automatically generated into a **special sketch file**.

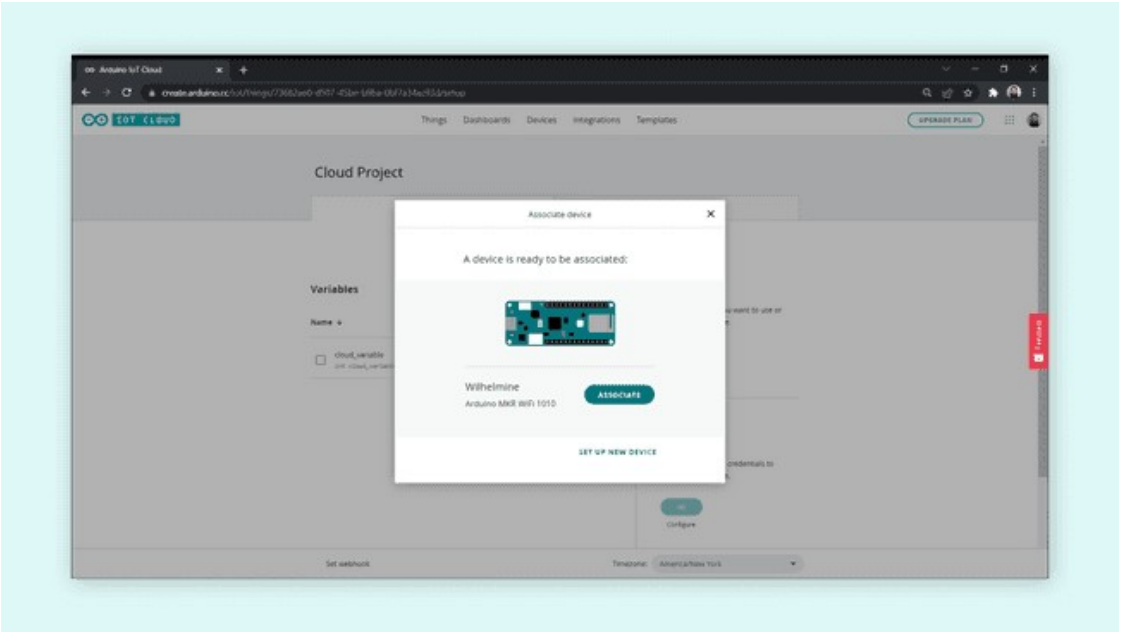
The Thing overview



4. Configuring a Device

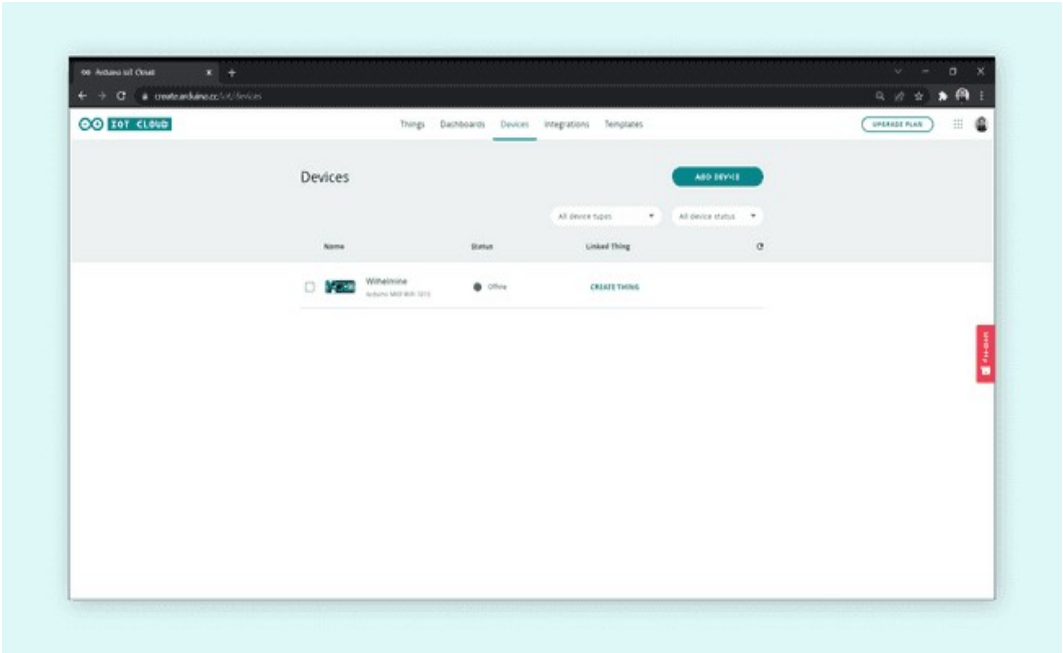
Devices can easily be added and linked to a Thing. The Arduino IoT Cloud requires your computer to have the Arduino Agent installed. The configuration process is quick and easy, and can be done by clicking on the **"Select device"** button in the Thing overview. Here, we can choose from any board that has been configured, or select the **"Configure new device"** option.

Configuring a device.



We can also get a complete overview of our devices by clicking the **"Devices"** tab at the top of the Arduino IoT Cloud interface. Here we can manage and add new devices.

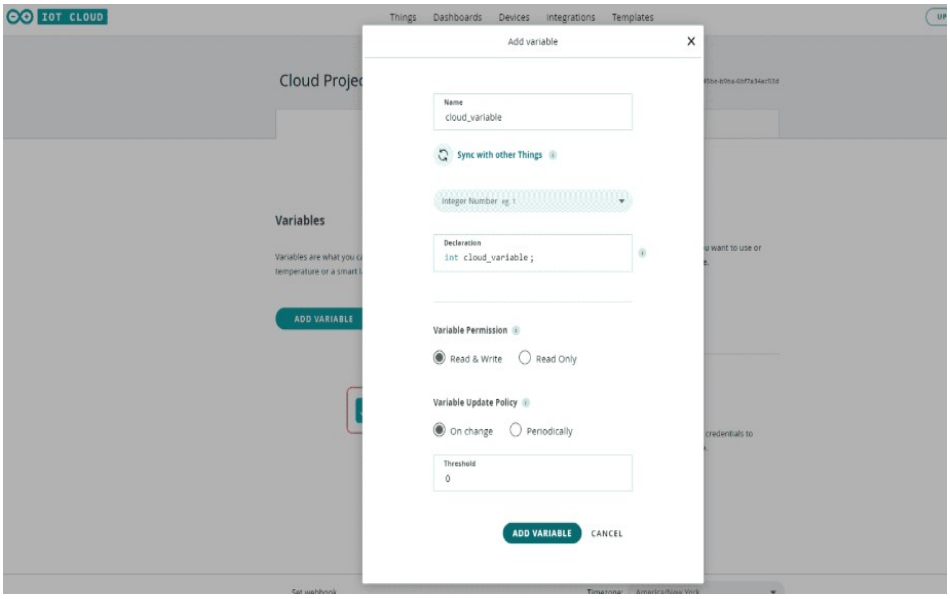
The device tab.



5. Creating Variables

The variables we create are automatically generated into a sketch file. There are several data types we can choose from, such as **int, float, boolean, long, char**. There are also special variables, such as **Temperature, Velocity, and Luminance** that can be used. When clicking on the **"Add variable"** button, we can choose name, data type, update setting and interaction mode.

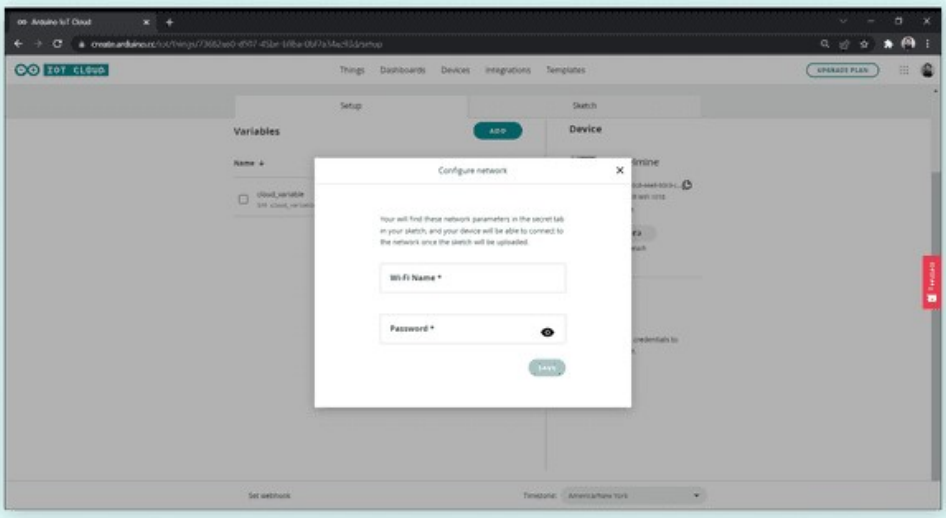
Creating variables



6. Connecting to a Network

To connect to a Wi-Fi network, simply click the **"Configure"** button in the network section. Enter the credentials and click **"Save"**. This information is also generated into your sketch file!

Entering network credentials.



7. Editing the Sketch

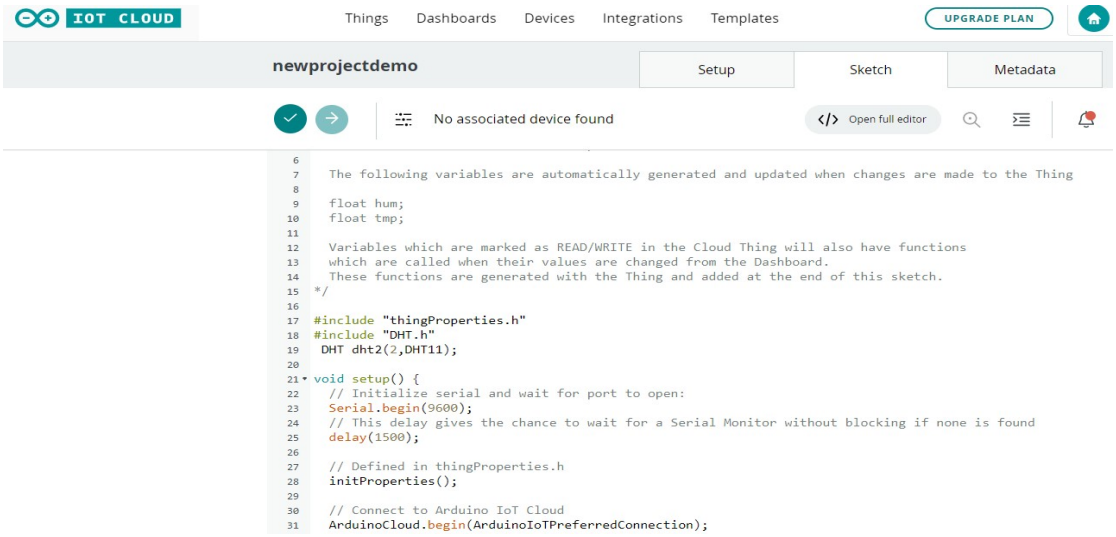
Now that we have configured variables, devices and network settings, we can get to programming our devices! An automatically generated sketch file can be found in the "Sketch" tab. It has the same structure as a typical .ino file, but with some additional code to make the connection to your network and to the cloud.

A sketch that, for example, reads an analog sensor, and use the **cloud variable** to store it. When the sketch has been uploaded, it will work as a regular sketch, but it will also update the cloud variables that we use!

Additionally, each time we create a variable that has the **Read & Write** permission enabled, a function is also generated, at the bottom of your sketch file. Each time this variable changes, it will execute the code within this function! This means that we can leave most of the code out of the **loop ()** and only run code when needed.

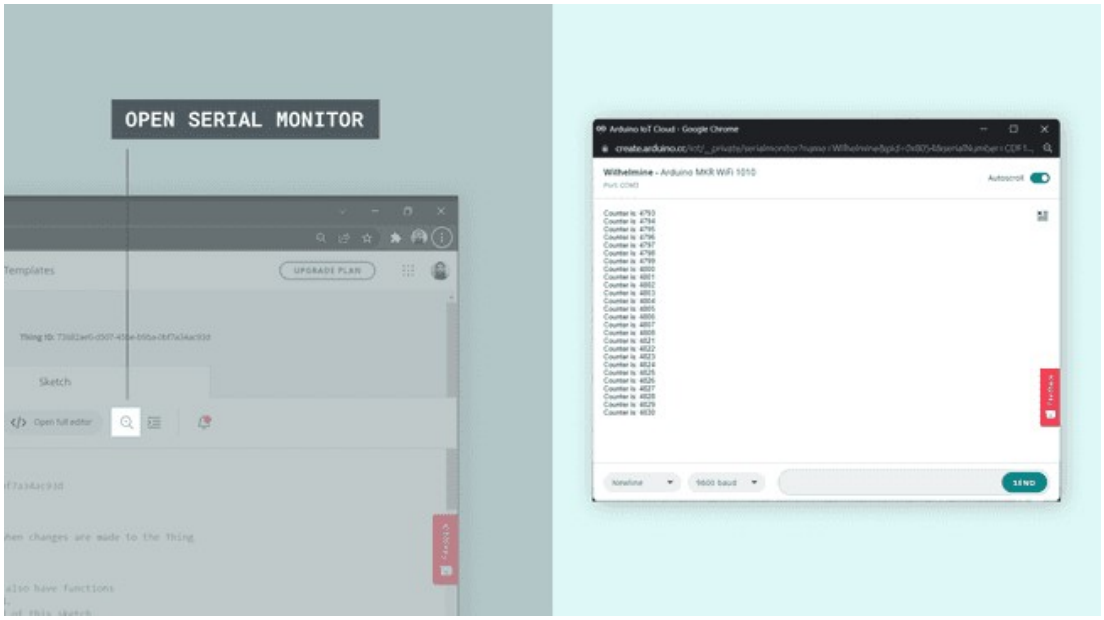
To upload the program to our board, simply click the **"Upload"** button.

Editing a sketch in the cloud editor.



The editor also has a **Serial Monitor Tool**, which can be opened by clicking the magnifying glass in the toolbar. Here you can view information regarding your connection, or commands printed via Serial.print()

.



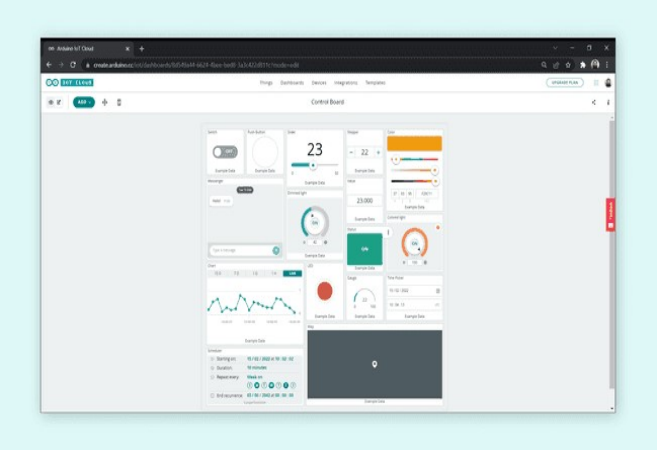
After we have successfully uploaded the code, we can open the **"Serial Monitor"** tab to view information regarding our connection. If it is successful, it will print **"connected to network_name"** and **"connected to cloud"**. If it fails to connect, it will print the errors here as well.

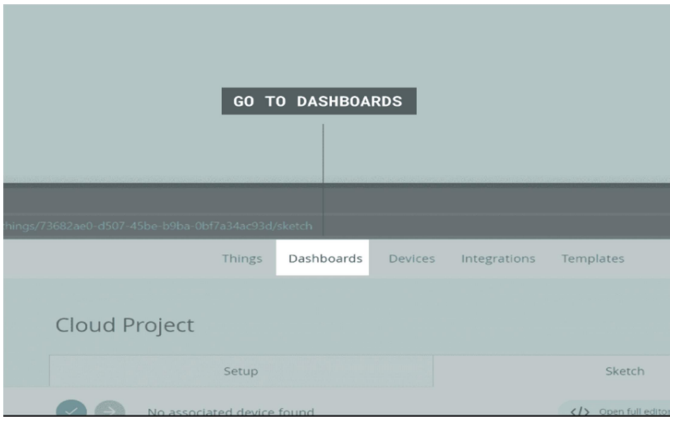
8. Creating a Dashboard

Dashboards are visual user interface for interacting with your boards over the cloud, and we can setup many different setups depending on what your IoT project needs. We can access our dashboards by clicking on the **"Dashboards"** tab at the top of the Arduino IoT Cloud interface, where we can create new dashboards, and see a list of dashboards created for other Things.

If we click on **"Create new dashboard"**, we enter a dashboard editor. Here, we can create something called **widgets**. Widgets are the visual representation of our variables we create, and there are many different to choose from. Below is an example using several types of widgets.

The different widgets available.

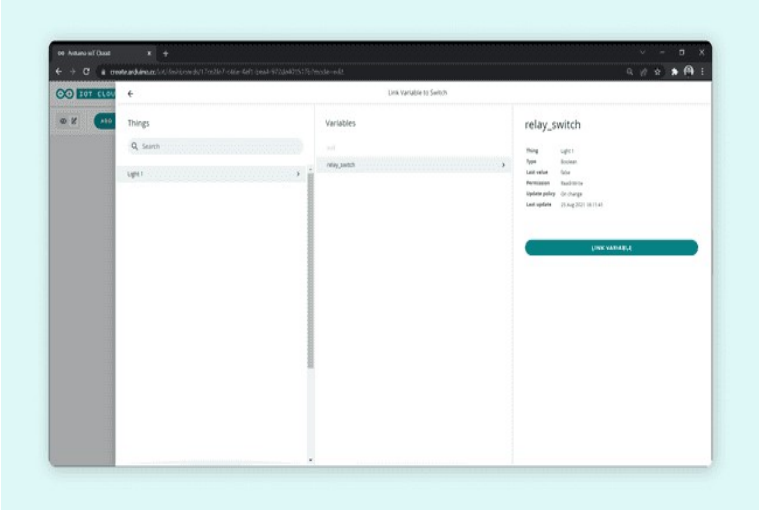




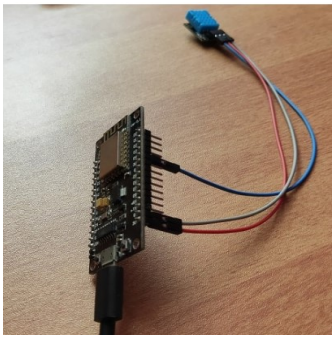
When we create widgets, we also need to **link them to our variables**. This is done by clicking on a widget we create, select a Thing, and select a variable that we want to link. Once it is linked, we can either interact with it, for example a button or we can monitor a value from a sensor. As long as our board is connected to the cloud, the values will update!

Let's say we have a **temperature widget** that we want to link to the **temperature** variable inside the **Cloud project** thing.

Linking a variable to a widget.



❖ **A Program to upload sensor data to local/cloud server using Wi-Fi.**



Requirements:-

- DHT11 Sensor x 1
- ESP8266 NodeMCU x 1
- Computer/laptop
- Data cable
- Jumper cables

```
/*
 Arduino IoT Cloud Variables description
 The following variables are automatically generated and updated when changes are made to the Thing
 float hum;
 float tmp;
 Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
 which are called when their values are changed from the Dashboard.
 These functions are generated with the Thing and added at the end of this sketch.
*/
```

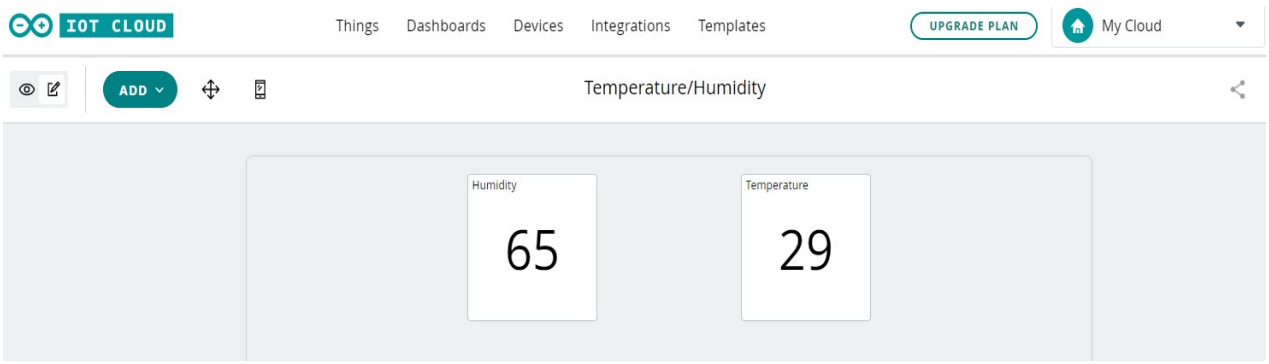
// **demodht.ino (**Arduino sketch code**)**

```
#include "thingProperties.h"
#include "DHT.h"
DHT dht2(2, DHT11);

void setup() {
 // Initialize serial and wait for port to open:
 Serial.begin(9600);
 // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
 delay(1500);

 // Defined in thingProperties.h
 initProperties();

 // Connect to Arduino IoT Cloud
 ArduinoCloud.begin(ArduinoIoTPreferredConnection);
 /*
   The following function allows you to obtain more information
   related to the state of network and IoT Cloud connection and errors
   the higher number the more granular information you'll get.
   The default is 0 (only errors).
   Maximum is 4
 */
 setDebugMessageLevel(2);
 ArduinoCloud.printDebugInfo();
}
void loop() {
 ArduinoCloud.update();
 temperature = dht2.readTemperature();
 humidity= dht2.readHumidity();                    // uploading sensor data to cloud server.
  delay(1000);
}
```

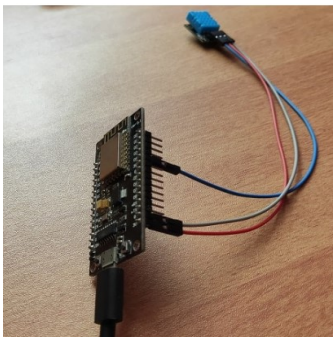
// Code generated by Arduino IoT Cloud, DO NOT EDIT.
thingProperties.h

```
#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char DEVICE_LOGIN_NAME[]  = "05098fe9-6063-41da-b806-4fj1k8c02350";

const char SSID[]               = SECRET_SSID;    // Network SSID (name)
const char PASS[]               = SECRET_OPTIONAL_PASS;    // Network password (use for WPA, or use as key for WEP)
const char DEVICE_KEY[]  = SECRET_DEVICE_KEY;    // Secret device password

float humidity;
float temperature;

void initProperties(){

  ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(humidity, READ, 2 * SECONDS, NULL);
  ArduinoCloud.addProperty(temperature, READ, 2 * SECONDS, NULL);

}
WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

Output:-



XI. A program to retrieve sensor data from local/cloud server.



Requirements:-

- DHT11 Sensor x 1
- ESP8266 NodeMCU x 1
- Computer/laptop
- Data cable
- Jumper cables

```
/*
  Arduino IoT Cloud Variables description
 The following variables are automatically generated and updated when changes are made to the Thing
 float hum;
 float tmp;
 Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
 which are called when their values are changed from the Dashboard.
 These functions are generated with the Thing and added at the end of this sketch.
*/
// demodht.ino (Arduino sketch code)
#include "thingProperties.h"
#include "DHT.h"
DHT dht2(2, DHT11);

void setup() {
 // Initialize serial and wait for port to open:
 Serial.begin(9600);
 // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
 delay(1500);

 // Defined in thingProperties.h
 initProperties();

 // Connect to Arduino IoT Cloud
 ArduinoCloud.begin(ArduinoIoTPreferredConnection);
 /*
   The following function allows you to obtain more information
   related to the state of network and IoT Cloud connection and errors
   the higher number the more granular information you'll get.
   The default is 0 (only errors).
   Maximum is 4
 */
 setDebugMessageLevel(2);
 ArduinoCloud.printDebugInfo();
}
void loop() {
 ArduinoCloud.update();
 temperature = dht2.readTemperature();
 humidity = dht2.readHumidity();
  delay(1000);
 Serial.println ("temperature=", temperature );   // retrieve sensor data to cloud server.
Serial.println ("Current humidity = ",humidity);
}
```

// Code generated by Arduino IoT Cloud, DO NOT EDIT.

thingProperties.h

```
#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char DEVICE_LOGIN_NAME[]  = "05098fe9-6063-41da-b806-4fj1k8c02350";

const char SSID[]               = SECRET_SSID;    // Network SSID (name)
const char PASS[]               = SECRET_OPTIONAL_PASS;    // Network password (use for WPA, or use as key for WEP)
const char DEVICE_KEY[]  = SECRET_DEVICE_KEY;    // Secret device password

float humidity;
float temperature;

void initProperties(){

  ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(humidity, READ, 2 * SECONDS, NULL);
  ArduinoCloud.addProperty(temperature, READ, 2 * SECONDS, NULL);

}
WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

Output:-

Retrieve sensor data to cloud server into the serial monitoring.