SHREYA KUMARI

Krshreya2402@gmail.com

BATCH JUNE-JULY

MAJOR PROJECT

IoT controlled smart home

Setup a Google assistant applet from IFTTT to trigger a web request which will update a field in ThingSpeak cloud. Then read the data from ThingSpeak cloud using Arduino to control the appliances in your home from anywhere in the world.

Ex: - From your mobile phone, if you say- “ok google” Turn on the light” from anywhere, it should turn on the light in your home.

Required Components:

1) Arduino Uno with cable
2) Esp8266(Wi-Fi module)
3) 5v Relay two channels
4) 230v bulb along with holder
5) Wires
6) Jumper cables

OVERVIEW OF GOOGLE ASSISTANT CONTROLLED HOME AUTOMATION PROJECT:

The Internet of Things (IoT) is the network of connected physical objects, such as vehicles, buildings and people. These objects, or things are often embedded with electronics to include sensors, actuators and microcontrollers that enable the devices to sense the environment around them, log data in real time, communicate with services or other devices, and be remotely controlled.

We are using an Arduino UNO for the microcontroller side and an ESP8266 module for connecting to the internet. We are going with Arduino and ESP8266 module to keep the cost down.

For this project, we will also build an Arduino Wi-Fi shield using the ESP8266 which sits snug on top of the Arduino UNO board. This shield can be used to program the ESP8266 using AT command or directly using the Arduino IDE. So, the shield can be used for many other creative Arduino projects which require internet connection.

To communicate with the Google assistant on our mobile phone we have used the IFTTT services, which configure the assistant to listen for a particular command and trigger a link if the command is received. Now, as we already know the ESP8266 can read information from the internet only through API calls, so we need a platform which can provide us this API option this is where ThingSpeak comes in action. Basically, the voice command given to Google assistant changes the value of a field in our ThingSpeak channel accordingly. While the ESP8266 periodically checks the value of this field using API calls and sends this value to Arduino using serial communication. The Arduino then performs required action like toggling a relay based on the value received.

ARDUINO UNO WI-FI SHIELD USING ESP8266

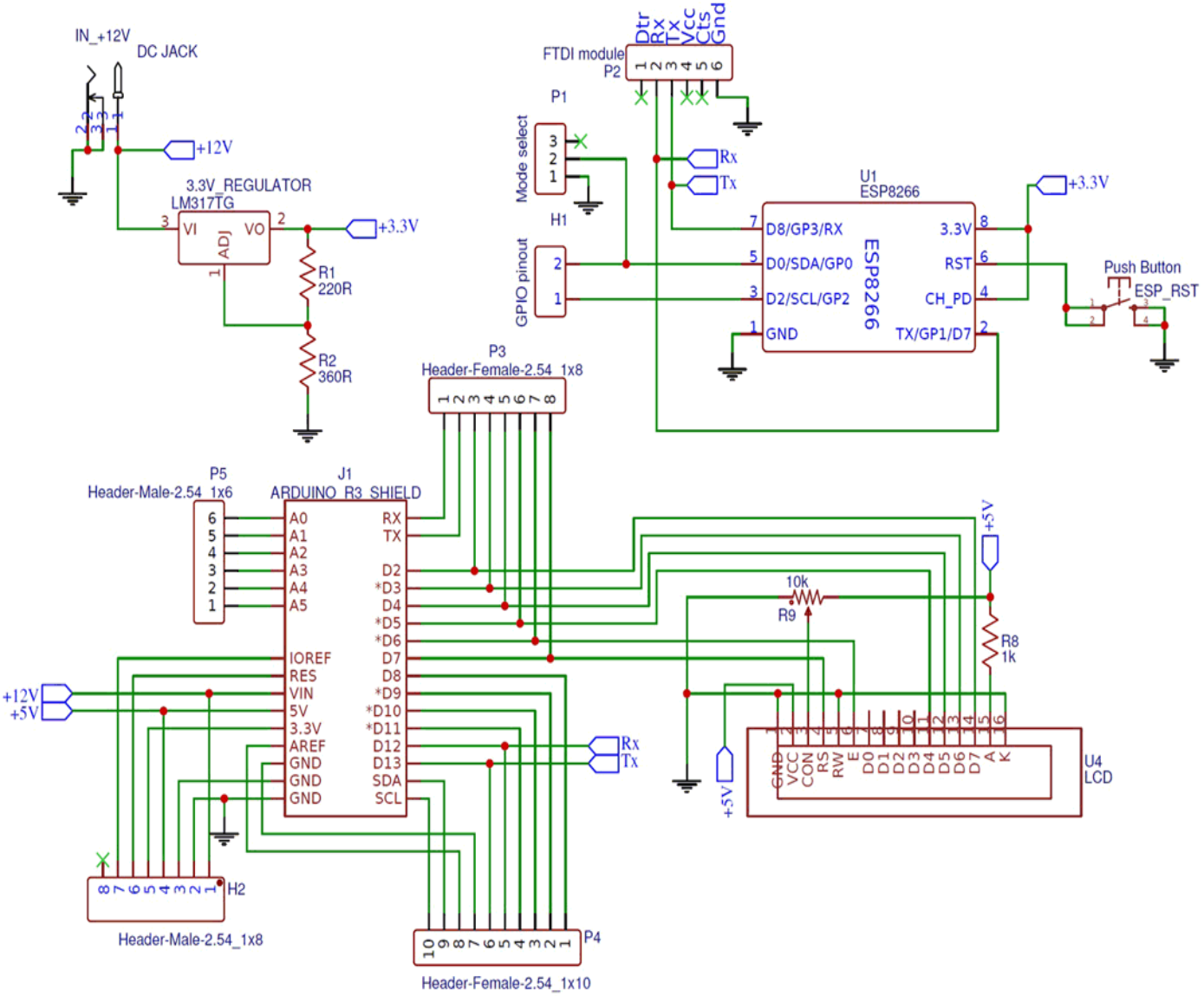
We start by interfacing the Arduino UNO board with the ESP8266 Wi-Fi module. The ESP8266 is a tricky module to use; it needs its own power source and a specific connection set-up to communicate with Arduino.

We must remember that the ESP8266 when purchased comes with a default firmware which is capable of communicating with AT commands. But if the module has been directly programmed with Arduino, then the default firmware will be erased and it has to be flashed again if AT commands are to be used.

The GPIO0 pin of the ESP8266 module is connected to a jumper pin which can be toggled to connect the pin to ground. This allows the user to set the ESP8266 module to work either in AT command mode or Programming mode (Arduino IDE). Both the GPIO0 and GPIO2 is connected to an external connector so that these GPIO pins can also be utilized.

Finally, on the Arduino side, we have connected the Rx and Tx pin of the ESP8266 module to the 12 and 13 pins of Arduino. We did not use the hardware serial (pin 0 and 1) to make debugging easy.

CIRCUIT DIAGRAM OF THE WI-FI SHIELD



ARDUINO WI-FI SHIELD PROGRAMMING MODE AND AT MODE

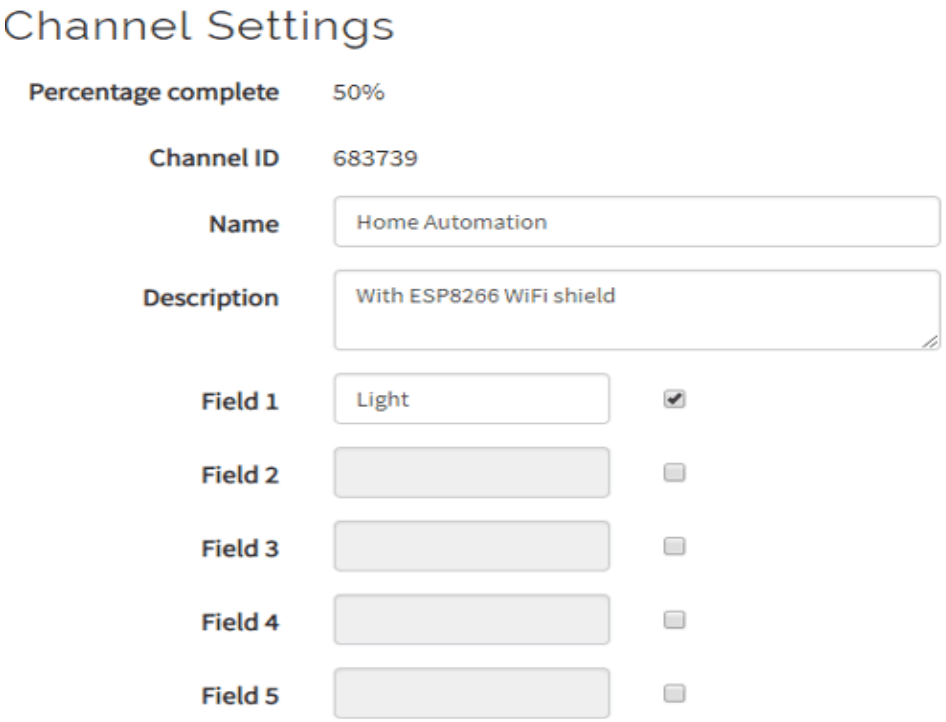
When the ESP8266 is programmed directly using the Arduino IDE or when it is to be flashed, the GPIO 0 pin should be connected to ground and Reset pin has to be momentarily connected to ground every time before the uploading process. Similarly, when the ESP8266 is working with AT commands the GPIO pin should be left free and again the Reset pin should be momentarily connected to ground to reset it.

To make things easier the Wi-Fi shield that we designed has a toggle pin which can shift between Programming mode and AT command mode as shown in the below images. Also, the Reset pin can be connected to ground by simply pressing the reset button (red colour) every time before uploading the code.

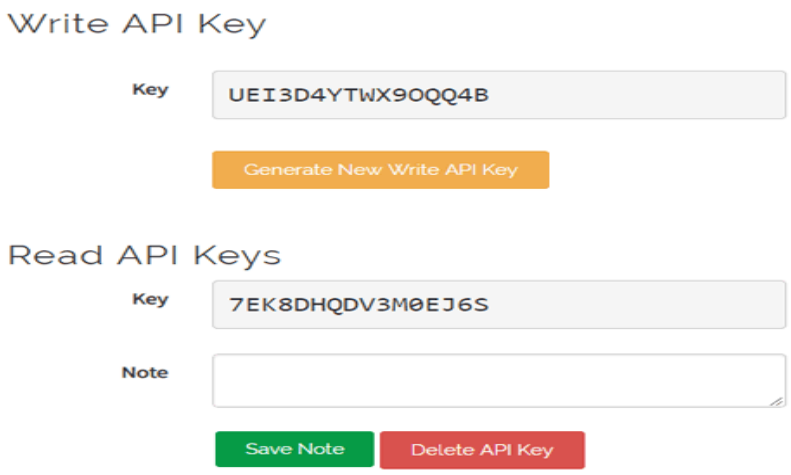
SETTING UP THE ThingSpeak CHANNEL FOR GOOGLE ASSISSTANT

We need a channel in ThingSpeak which will store data from the Google assistant and later allows the same to be retrieved by the ESP8266 using API calls. If we are new to ThingSpeak, set up an account by signing up on thingspeak.com and click on new channel. You can pick any name for your channel and give it a description. Since we are toggling only one light we have used only one field and named it as Light but again you can use as many as you want.

A screen of something like this will appear:



Next click on the **API keys tab**, here we will be provided with two API keys one for Write function and the other for read function. You can read or write values to the filed only using these keys respectively. Every key will be unique, mine is shown below yours will be different for sure. Never share your keys as it can give permission to write or read to your channel. They keys shown below were destroyed after usage.



Now let us look at the API GET calls using which we can write and read data to the field that we just created.

Read from Thing Field:

api.thingspeak.com/channels/683739/fields/1/last.json?api_key=7EK8DHQDV3M0EJ6S&results=2

Write to Thing Field:

api.thingspeak.com/update?api_key=UEI3D4YTWX9OQQ4B&field1=7

These are our API you have to replace the key value with your keys and also change the channel ID according to your ThingSpeak channel. If you have selected the first field as shown in the image then the field value need not be changed.

You can also try loading these API calls on your Brower and check how it working. In the above *write to thing field* we are writing 7 (appended at the last) to the channel. You can load this on your browser and check if the value is being reflected on your ThingSpeak account. Similarly, the read from Thing Field API call when loaded in browser should give you the value that you have sent to field previously, in this case 7.

SETTING UP IFTTT APPLETS

Now we know how to send and read values form the field, next we have to **set two applets** in our IFTTT account. If you are new to IFTTT simply sign up for it and link you Gmail account, this Gmail account should be the same one form which you are using the Google voice assistant. Of the two applets both will be used *write value to the field* using the above discussed link. But one would **listen for "Turn on Reading Light" and write "1" to the field while the other applet will listen for "Turn off Reading Light" and write "0" using the above API calls.**

To create an Applet, get into my applets and click on "New Applet". Then in this "This" section selects the Google voice assistant and in the "That" section selects the Webhooks service.

- **Set Up Software**

1) Create a ThingSpeak account and at least one channel. Record the write API key from the **API Keys** tab in your channel view.

2) Add a lamp indicator to your channel. This example uses a lamp indicator widget set so the lamp is on if the field 1 value is greater than 0.

3) Create an IFTTT account if you do not already have one. Log in to your account.

4) Install the Google Assistant app on your mobile device.

- **Create IFTTT Applet to Turn Lamp On**

IFTTT applets require a trigger and an action. For this example, the trigger is a command from Google Assistant and the action is Webhooks. The webhook sends a command to the ThingSpeak REST API to change the channel value.

Sign in to your IFTTT account. Choose **My Applets** from the top menu and then select **New Applet**.

- **Set trigger**

1) Click **+this** to set the trigger.
2) Enter google assistant in the search bar and select google assistant as our trigger service.
3) Choose say a simple phrase.
4) Enter phrase to trigger our lamp. Since the assistant is also capable of web searches, avoid simple patterns such as “Turn on light”.

- **Set Action**

Now choose **Webhooks** as your action.

1) Select **+that** to continue.
2) Enter Webhooks in the search box and select the **Webhooks** card.
3) Complete the Webhooks action fields. Enter the URL to change the field value to 1. The URL has the following form.

https://api.thingspeak.com/update?api_key=XXXXXXXXXXXXXXXX&field1=1

- **Create IFTTT Applet to Turn Lamp Off**

Repeat the previous steps to set up another applet to turn the lamp off. Use a different phrase to indicate that you want to change the lamp state to off. This example uses the command Kill the lamp, which is easy to distinguish from the on command. Use Webhooks and the REST call to set the field value to 0 for the off state.

```
https://api.thingspeak.com/update?api_key=XXXXXXXXXXXXXXXX&field1=0
```

PROGRAMMING THE ARDUINO FOR GOOGLE ASSISTANT HOME AUTOMATION

Both the IFTTT and the ThingSpeak account should be set by now. So, based on the command given to our Google assistant the IFTTT will send a value (0 or 1) to our ThingSpeak account. Now, on our Arduino side we have to write a program to check if the value of field is 0 or 1. If 0 we have to turn off the light and if 1, we have to turn it on.

The **complete program to do the same is given at the end**, I am breaking the program into meaningful snippets and explaining them below. We begin by defining the pins to which the ESP and the LCD is connected to Arduino. You can refer the circuit diagram to verify the same.

```
SoftwareSerial ESP(12,13); //ESP is connected to 12 and 13 pin of Arduino
const int rs = 7, en = 6, d4 = 5, d5 = 4, d6 = 3, d7 = 2; //Mention the pin number for LCD connection
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

Then we have to feed in few critical parameters, like the name of the Wi-Fi to which the ESP should connect to its password and then finally the API call request that we obtained from our ThingSpeak. Be sure that you have

changed these credentials according to your applications. Verify the API key and load it in browser to make sure.

```
String WiFi_SSID = "Oneplus";
String WiFi_Pass = "nightfury";
String sendData = "GET /channels/683739/fields/1/last.json?api_key=7EK8DHQDV3M0EJ6S&results=2";
String output = ""; //Initialize a null string variable
```

Inside the **setup function**, we declare pin 10 as output this is where we connect our load thorough a relay. Then we display a small intro text on the LCD and initialize serial monitor using the below lines of code.

```
pinMode(10,OUTPUT);
lcd.begin(16, 2); //Initialise 16*2 LCD
lcd.print("  Arduino WiFi"); //Intro Message line 1
lcd.setCursor(0, 1);
lcd.print("    Shield    "); //Intro Message line 2
delay(2000);
Serial.begin (9600);
```

By default, the ESP works with a baud rate of 115200, but the Arduino is not fast enough to read data from the ESP at such high speed. It does read data but I personally found a lot of garbage values many a time. Hence, I decided to **change the ESP to work with 9600 baud rates using the AT+CIOBAUD=9600** as shown below. After changing the baud rate, we can re-initialize the software serial to work with 9600 baud rates.

```
 ESP.begin(115200);
ESP.println("AT+CIOBAUD=9600");
delay(100);
ESP.begin(9600);
```

Next, we have a **series of AT commands that has to be sent to the module only once**. They include turning off the Echo option (ATE0) then setting the ESP to work in station mode (AT+CWMODE=1) and then connecting it to the router using (AT+CWJAP) etc. Once it is executed the ESP will remember these details and will connect to our Router as a station every time, we power it on. So, you can comment these lines after first time usage (optional though).

```
ESP_talk("ATE0", 1000); //Turn off Echo
ESP_talk("AT+CWMODE=1", 1000); //Set ESP as station
ESP_talk("AT+CWJAP=\""+ WiFi_SSID +"\",\""+ WiFi_Pass +"\"", 5000);
//Connect     to WiFi
delay(1000);
ESP_talk("AT+CIPMUX=1",1000);
delay(1000);
```

Inside the **main loop function**, we have to **connect to our ThingSpeak API**. This can be done by first starting a TCP connection to the ThingSpeak network by the following command.

```
ESP_talk("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",80",1000);
```

Then **we have to tell how many characters are being sent to the connection using the AT+CIPSEND**. In my case it is 76. Because the below command that we will send, we have 74 characters and with that we have to add 2 for “/n” which gives 76.

```
GET/channels/683739/fields/1/last.json?api_key=7EK8DHQDV3M0EJ6S&
results=2
```

Then we actually send the above data which is stored in the variable send data. These commands are sent with a delay of 100ms for stability but it is not mandatory. The program for the same is shown below.

```
ESP_talk("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",80",1000);
//AT+CIPSTART="TCP","api.thingspeak.com",80
delay(100);
ESP_talk("AT+CIPSEND=76",1000);
delay(100);
ESP_talk(sendData,1000);
delay(100);
```

The above code will fetch the field data as a string value from the ThingSpeak website and store it in the variable “output” which will look something like this.

```
+IPD,64:{"created_at":"201922T12:13:32Z","entry_id":15,"field1":"0"}CL
OSED
```

As you can see, off the entire string value, we only need to check if the field1 value is 0 or 1. So we use the **charAT** function in Arduino to **fetch that particular char value form the entire string.** The location of the value is 11 steps behind from the last value. So, the code looks like

```
int light_value = int (output.charAt(output.length()-11))-48;
```

The last step is to **compare this value with 0 and 1**. Then toggle the light on if it is a 1 and turn it off if it is a 0. The LCD is also made to display the result, based on the field value.

```
 lcd.clear();
lcd.print("Listning...."); //Intro Message line 1
lcd.setCursor(0, 1);
if (light_value == 0) //light should be off
{
  lcd.print("Light is OFF :-(" );
  digitalWrite(10,LOW);
}
if (light_value == 1) //light should be off
{
 lcd.print(":-) Light is ON");
  digitalWrite(10,HIGH);
}
```

You would have noticed the ***ESP_talk* function** being used extensively throughout the program. This function basically has two parameters one is the actual command which is sent to the ESP module and other is the time out value within which the ESP should respond back for the sent command. The response from the ESP is then stored in the variable output. This comes in very handy when debugging the ESP module. The function definition is shown below.

```
void ESP_talk(String AT_cmd, const int timeout)
{
  Serial.print("Sent: ");
  Serial.print(AT_cmd);
  ESP.println(AT_cmd); //print to ESP through software serial
  Serial.println("");//Move to next line

  long int time = millis();

  output=""; //clear the string


  while ( (time + timeout) > millis())
  {
    while (ESP.available())
    {
      char i = ESP.read(); // read one char
      output += i; //Combine char to string
    }
  }
  Serial.print("Received: ");
  Serial.print(output);
}
```

USING GOOGLE ASSISTANT TO TOGGLE LIGHTS

So, we are all set to use our **Arduino ESP8266 Wi-Fi module to control the Home Appliances** from anywhere using Google Assistant.

As explained the above program is used to toggle the pin 10.

CODE

```
//Use the new flah tool ad bin file to get the latest firmware and then use AT+CIOBAUD=9600 to change the baud, // AT+UART_DEF=9600,8,1,0,0 (for old firmware)

#include <LiquidCrystal.h>       //Librarey for LCD display
#include <SoftwareSerial.h>    //Librarey for serial connection with ESP

SoftwareSerial ESP(12,13);    //ESP is connected to 12 and 13 pin of Arduino

const int rs = 7, en = 6, d4 = 5, d5 = 4, d6 = 3, d7 = 2;   //Mention the pin number for LCD connection
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

String WiFi_SSID = "Oneplus";
String WiFi_Pass = "starrynight";
String sendData = "GET /channels/683739/fields/1/last.json?api_key=7EK8DHQDV3M0EJ6S&results=2";
String output = "";          //Initialize a null string variable

void setup()

 {
  pinMode(10,OUTPUT);

  lcd.begin(16, 2);                //Initialise 16*2 LCD
  lcd.print("  Arduino WiFi");       //Intro Message line 1
  lcd.setCursor(0, 1);
  lcd.print("    Shield    ");          //Intro Message line 2
```

```
  delay(2000);

  Serial.begin (9600);


  ESP.begin(115200);
  ESP.println("AT+CIOBAUD=9600");
  delay(100);
  ESP.begin(9600);


  ESP_talk("ATE0", 1000);          //Turn off Echo

  /*ESP_talk("AT+CWMODE=1", 1000);        //Set ESP as station

   ESP_talk("AT+CWJAP=\""+ WiFi_SSID +"\",\""+ WiFi_Pass +"\"", 5000);
//Connect to WiFi
   delay(1000);

   ESP_talk("AT+CIPMUX=1",1000);
   delay(1000); */
}

void loop()

 {

    ESP_talk("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",80",1000);
//AT+CIPSTART="TCP","api.thingspeak.com",80
    delay(100);
    ESP_talk("AT+CIPSEND=76",1000);
    delay(100);
    ESP_talk(sendData,1000);
```

```
    delay(100);

    int light_value = int (output.charAt(output.length()-11))-48;       //read
the    required value form string and convert it to int
    Serial.println(light_value);

    lcd.clear();
    lcd.print("Listning....");        //Intro Message line 1
    lcd.setCursor(0, 1);

    if (light_value == 0)           //light should be off
    {
      lcd.print("Light is OFF :-(" );
      digitalWrite(10,LOW);
    }
    if (light_value == 1)            //light should be off
    {
      lcd.print(":-) Light is ON");
      digitalWrite(10,HIGH);
    }

    delay(500);
}

void ESP_talk(String AT_cmd, const int timeout)
{
  Serial.print("Sent: ");
  Serial.print(AT_cmd);
  ESP.println(AT_cmd);         //print to ESP through software serial
  Serial.println("");                //Move to next line
```

```
  long int time = millis();

  output="";                    //clear the string

  while ( (time + timeout) > millis())
  {
    while (ESP.available())
    {
      char i = ESP.read();          // read one char
      output += i;                  //Combine char to string
    }
  }

  Serial.print("Received: ");
  Serial.print(output);
}
```