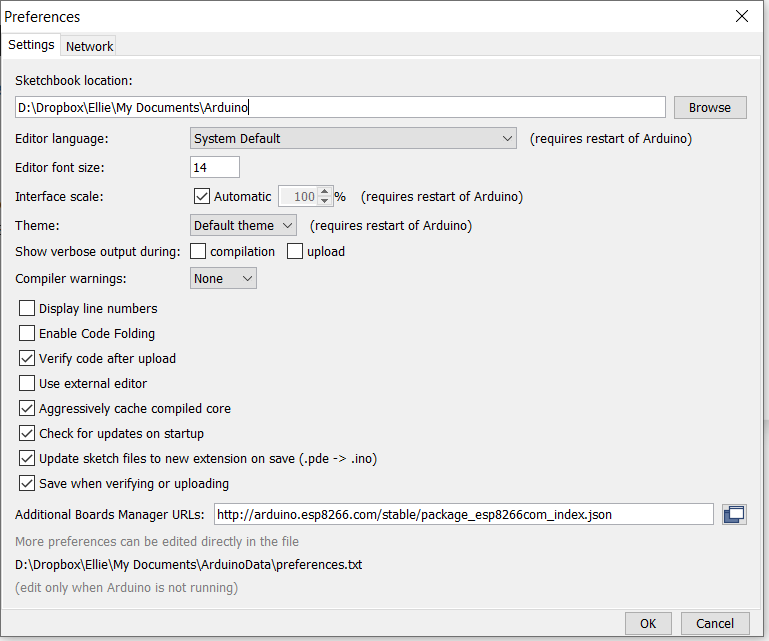
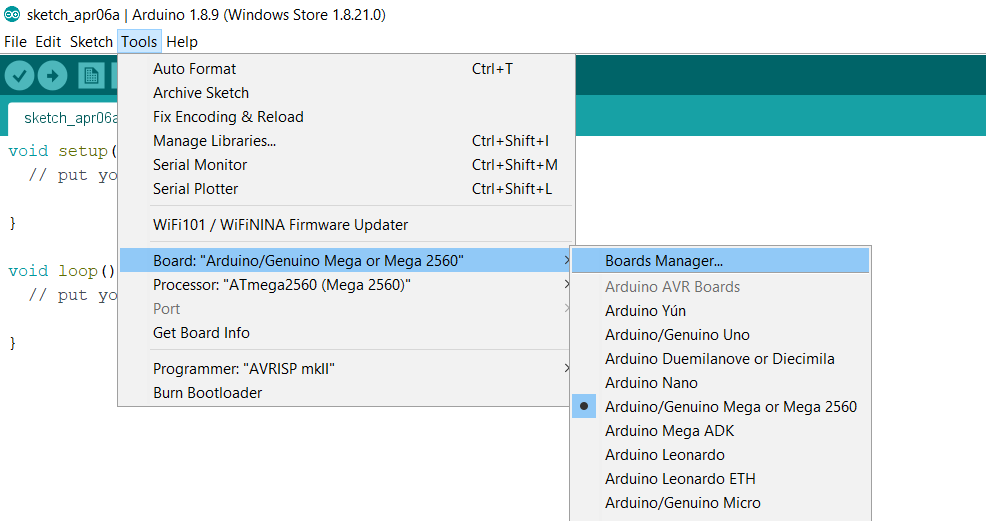
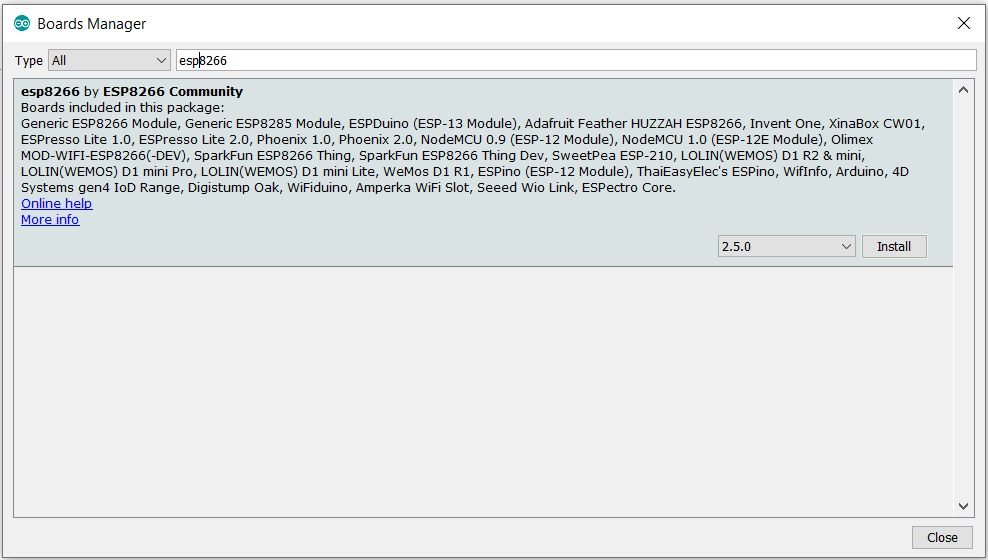
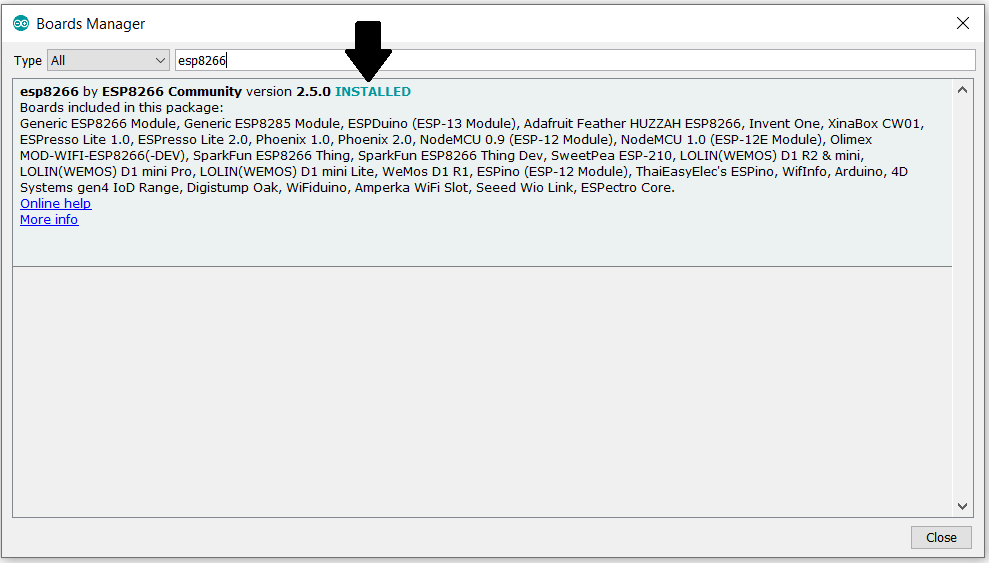
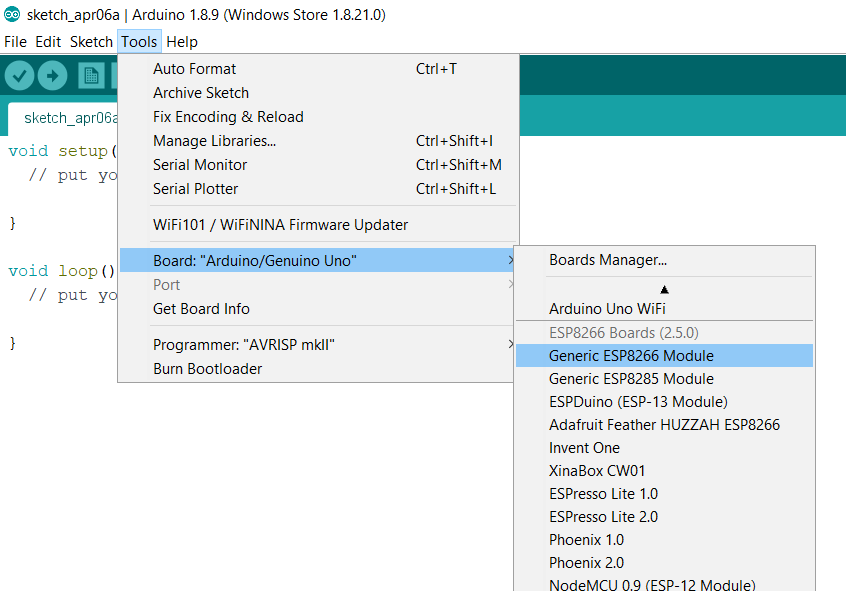
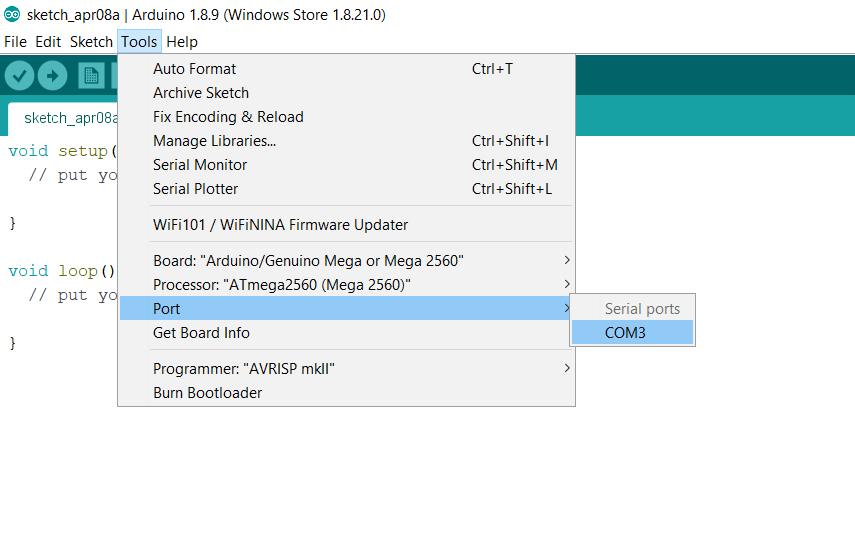
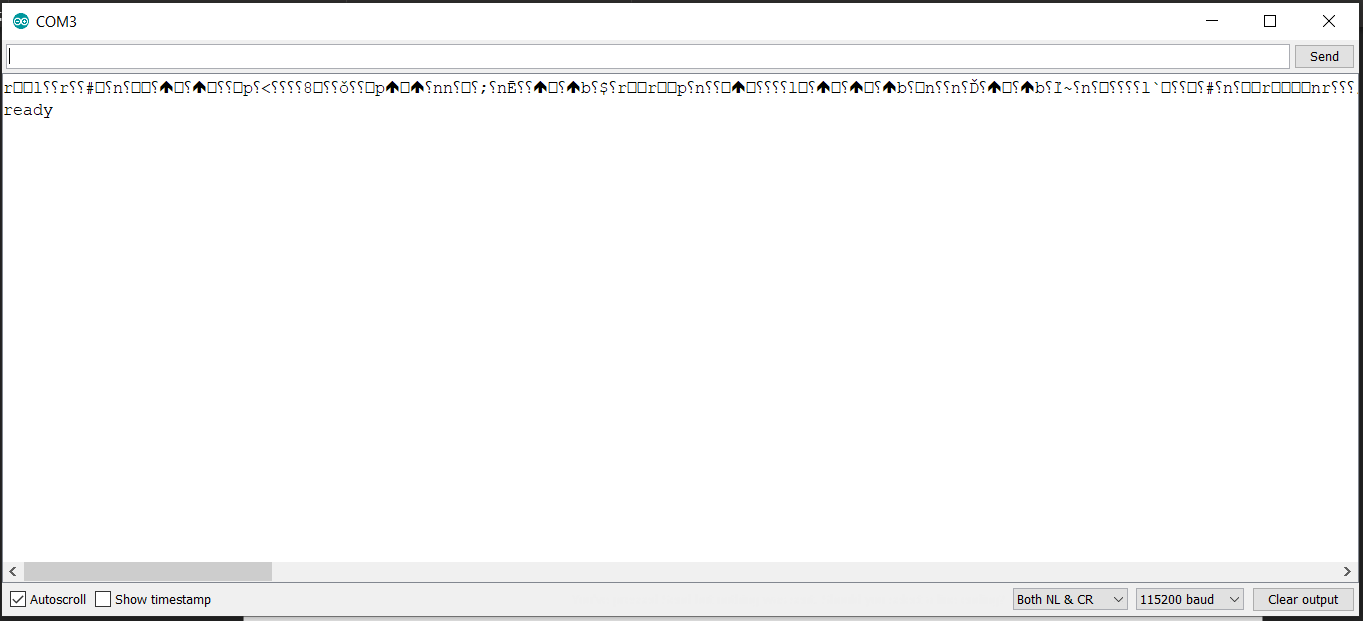
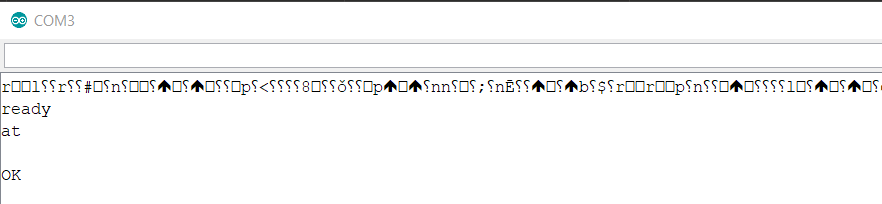
Step zero is to get the Arduino IDE setup to work with the ESP8266.

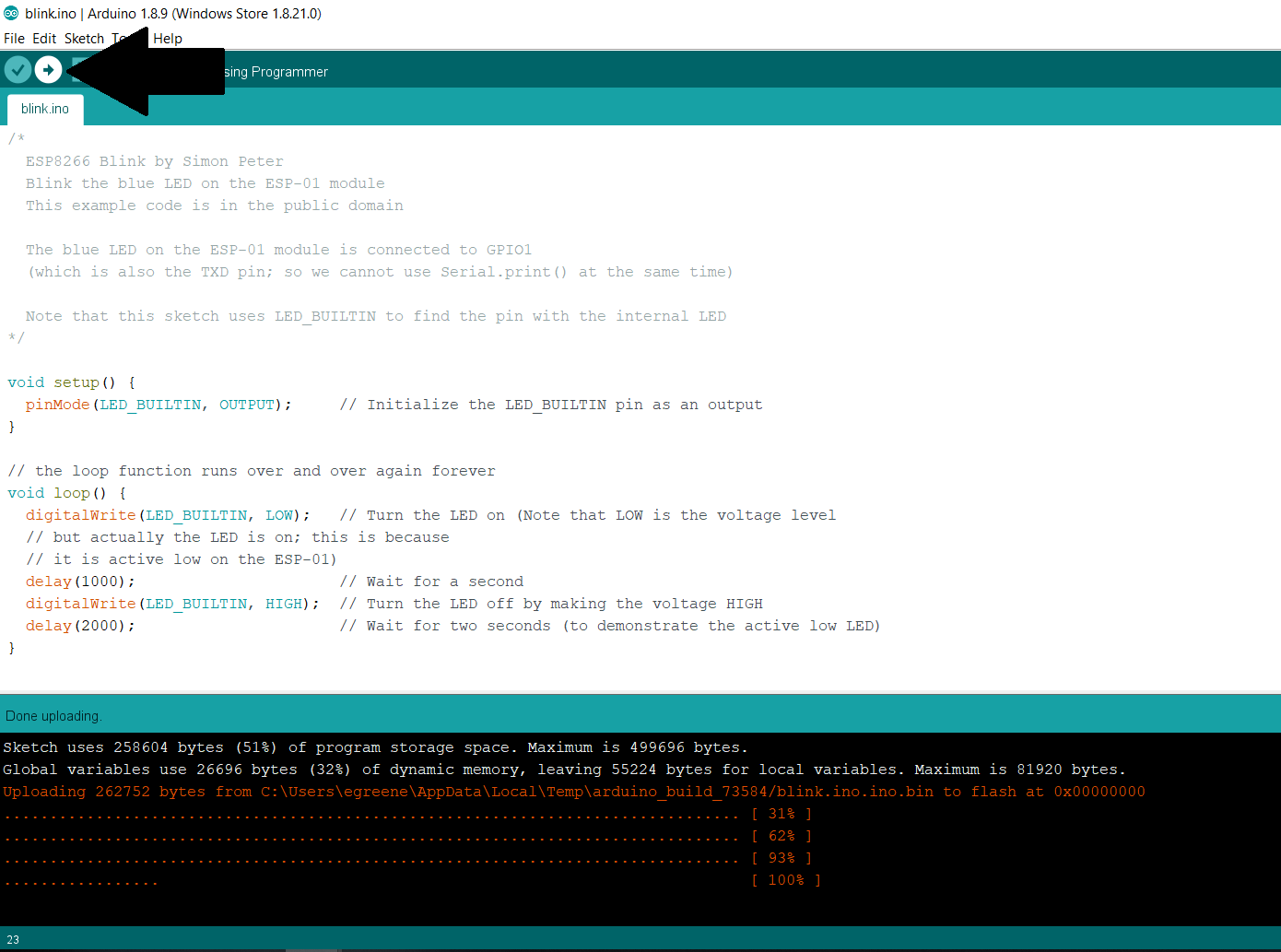
* 1. For Windows, get the IDE from the Windows Store.
  2. Under the preferences menu, add in the following additional Boards Manager url. 
  3. Launch the Arduino Board Manager  
     
  4. Wait for the Board manager to pull the list from the internet and search for ESP8266.  
     
  5. Click Install.
  6. When the module shows installed, click close  
     
  7. Finally, select the Generic ESP8266 module from the Tools >> Board Menu.   
     

1. Connecting to the ESP8266
   1. My Microcontroller is an ESP-01 board, and there are some things you need to know.
      1. It requires a 3.3v supply. A 5v supply cooks it.
      2. It draws a fair amount of current for a 3.3v device.
      3. It requires 3.3v levels on the serial interface
   2. Physical wiring to the ESP8266
      1. I’m using male-to-female jumper wires to connect the ESP to my breadboard.

Red-3.3v supply on Breadboard  
Black-Ground on breadboard  
Blue-RST  
Grey-TX  
Green-RX  
Purple-CH\_PD – VCC  
GPIO2 not connected.

Yellow - GPIO0 (leave disconnected)

* 1. For the 3.3v serial connection I’m using an Adafruit FTDI Friend
     1. Look at the back of the FTDI friend. Confirm that it is configured for 3.3v output.
     2. USB – To my PC  
        RX to the TX wire(Grey) of the ESP8266 board  
        TX to the RX wire(Green) of the ESP8266 board  
        GND to the common ground on the breadboard  
        VCC **NOT CONNECTED** – This outputs 5v by default, and will cook the ESP8266
  2. Find the correct serial port
     1. Launch device manger with the FTDI friend plugged in to the USB connection.
     2. Expand the ports section, note the available ports.
     3. If you have only one, that’s your serial port. Note the COM#.  
        Mine is COM3
     4. If you have multiple serial ports listed, Unplug in the FTDI friend and then plug it back in. The port that disappears and reappears is the FTDI friend.
  3. Launch the Serial Monitor in the Arduino IDE
     1. Launch the Arduino IDE
     2. Select the COM port you found earlier under the Tools > Ports men
     3. Launch the serial monitor under Tools > Serial Monitor
     4. Select 115200 as the baud rate in the lower right of the Serial Monitor
     5. Select “Both NL & CR” as the end-of line in the lower right of the Serial Monitor
     6. Double-check your wiring, and turn on the breadboard 3.3v power supply with any luck your Serial monitor will display this  
        
     7. Optional, Test your esp8266 using the stock AT commands.  
        (This assumes that your ESP8266 is running the stock firmware.)
        1. In the serial Monitor, type AT and press enter. This should reply with OK.  
           
        2. At this prompt you can enter AT commands from the default ESP8266 command set listed here <https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf>.)
        3. Type AT+GMR and press enter to see the current firmware version
        4. Type AT+CWLAP to list the available access points  
           If this returns an error, use AT+CWMODE=1 to set it into Wifi Client Mode and try again.
        5. If you want to use the ESP8266 as wireless interface for an Arduino, stop here. You’ve confirmed that the board works and accepts serial connections. I’m going to be replacing the default firmware with an Arduino sketch, so I must keep going.

1. Place the ESP8266 board into programming mode and upload a sketch
   1. Power off the breadboard.
   2. Connect the GPIO0 to Ground to place the device in programming mode
   3. Get the Blink.ino example sketch from <https://github.com/esp8266/Arduino/blob/master/libraries/esp8266/examples/Blink/Blink.ino>
   4. Open or Paste the sketch into the Arduino IDE and save it.
   5. Confirm that the generic ESP8266 is selected under the Tools > Board Menu
   6. Power on the breadboard
   7. Use the Arrow button on the tool to compile and upload the sketch  
      
   8. When the upload completes, power off the breadboard.
   9. Connect GPIO2 to a LED with a current limiting resistor.
   10. Connect the other end of the LED to ground.
   11. Unplug the GPIO0 (Yellow) pin from ground to exit programming mode.
   12. Power on the breadboard. The LED you connected to pin 2 should blink.
   13. Revel in your god-like powers.
2. Figure out how to trigger the relay on your relay board.  
   If you’ve bought well documented parts, good on you. Have a look at the data sheet and figure out how to trigger your relay and skip onto the next step. I didn’t do that, so I have to be clever.  
     
   Looking at my board I see that GPIO0 is connected to the input of an optoisolator. There is no visible connection to GPIO2, so I choose to believe that GPIO0 is the input I need. That’s half the battle. I also need to figure out what voltage the board operates at, i.e. what does it need for VCC and does it have an onboard voltage regulator.  
     
   Looking at the board I looked up several components and hit the jackpot with the LM1117-3.3. This is a low dropout 3.3 volt voltage regulator. I connected the relay board to my power supply with the ESP8266 *not* installed. Turning on the power supply I fed it 3.3 volts. I checked the voltage between ground and VCC pins for where the esp8266 would be installed. The output was 2.5 volts. This makes sense. The voltage regulator has a minimum voltage drop, so the 3.3v input wasn’t enough. I turned the input power up to 5V and measured 3.32v on where the ESP8266 should be. Yeah! Last but not least I plugged a 330 ohm resistor into the holes for VCC and GPIO0. That clicked the relay on as expected.  
     
   The final verdict: It appears to work with a 5 volt supply and toggles the relay with GPIO0.
3. Write the code.  
   For me the minimum viable product objective is to connect to the WiFi and display a button that triggers the relay for 5 seconds.  
     
   Read the important note in the next section before pushing code to the microcontroller.
4. Publishing the code to the Esp8266
   1. **NOTE! Read this so you don’t fry your chip!**Entering programming mode requires grounding GPIO0. Using GPIO as an output while it is connected to ground will exceed current limits on the GPIO and can and will fry it. It did not damage my board, but it did prevent me from programming it until I figured this out.   
        
      **To safely program the esp8266 power off the board, ground GPIO0, power on the board, remove GPIO0 from ground, and *then* program the board.**
   2. Follow the procedure above to place the board into programming mode and push the compiled code to the esp8266 using the Arduino IDE.
5. Ideas for future improvement
   1. SSL support
   2. One-wire serial display on gpio2 to show current IP address
   3. A second button that toggles on/off
      1. With a thing that forces it off after ~25 seconds.
   4. Blink out the ip address on a LED