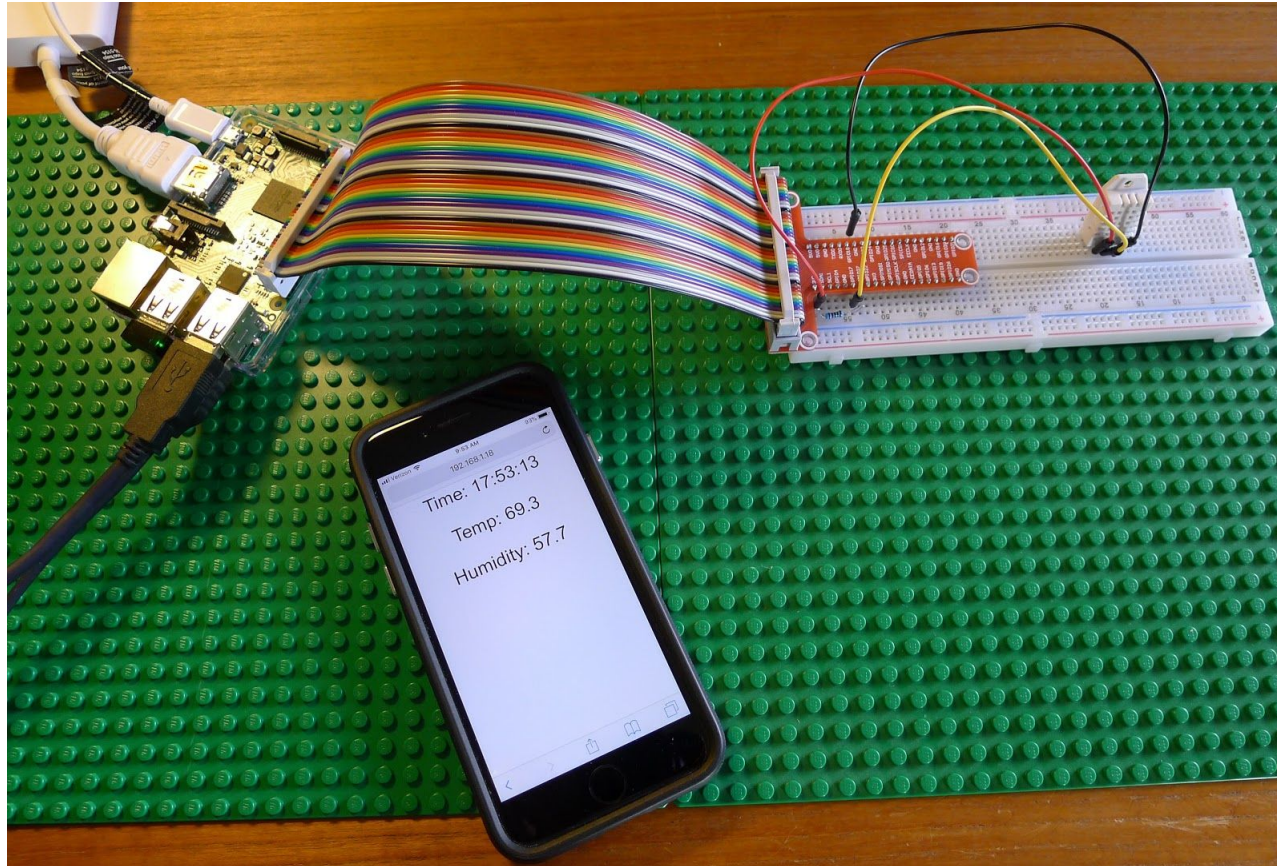


## GSOC Raspberry Pi “Build a Weather Station” Project - Step by Step Instructions:

Create a Raspberry Pi “weather station” that will broadcast the temperature and humidity over a local network, allowing phones, tablets and laptops to access the Raspberry Pi’s weather station from a web page hosted on the Raspberry Pi.

**Allow for 2 hours to complete the project with your troop**, here assuming that you have first read over the instructions and have a basic understanding of the steps and process.



---

### What we'll learn by working on this project:

- 1) Networking - connecting to and identifying the Raspberry Pi on a local network.
- 2) What is a “web server” and how to make the Raspberry Pi a web server.

- 3) Breadboarding - how to experiment with electronics projects and connect the Raspberry Pi to a sensor.
- 4) Coding - writing a python script that will read values from a temperature and humidity sensor and display the results on a web page.

---

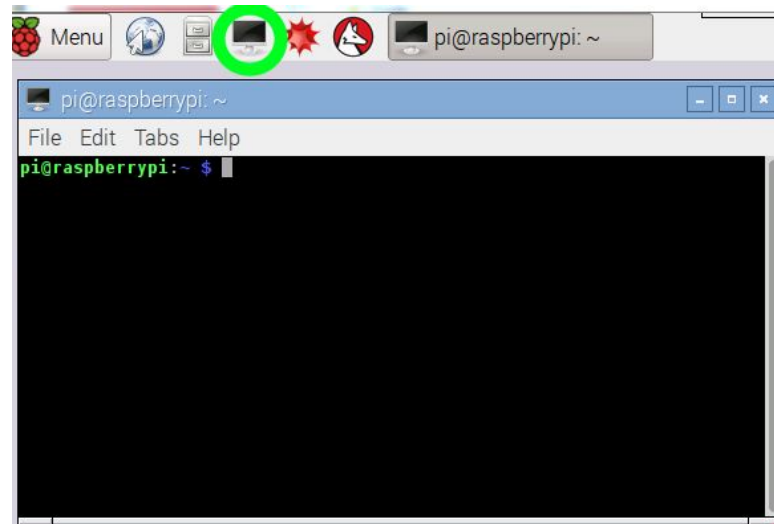
## Step 1: Connecting to the local WiFi network

For this step you will need to know a local WiFi network's SSID (i.e. the WiFi network's name) and password, just like you would if you needed to connect your phone to the WiFi network. This network should also be connected to the internet; for example, with your phone or laptop connected to this WiFi network you should be able to access a website like <http://www.google.com>.

With the Raspberry Pi off (be sure the micro USB power cord is disconnected) verify that the WiFi dongle is plugged into any one of the four USBs port on the Raspberry Pi as shown below.



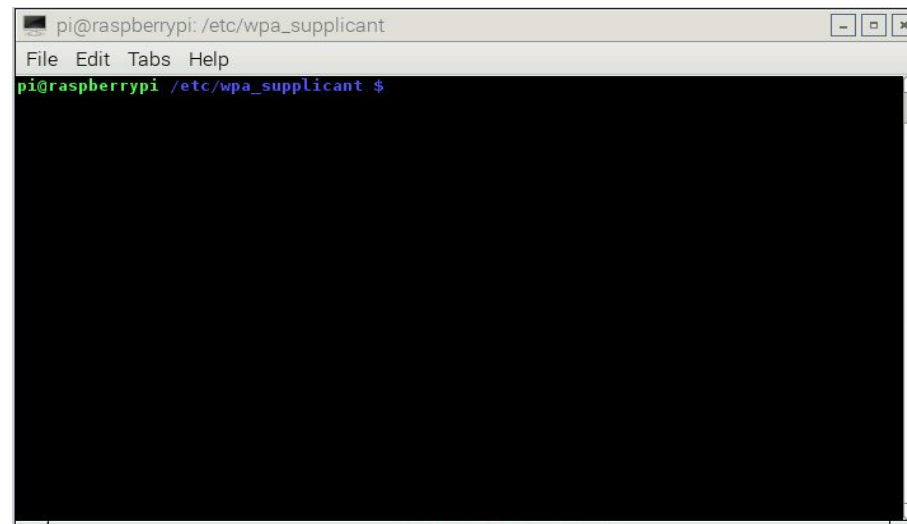
Connect the monitor, keyboard and mouse to the Raspberry pi as per the GSOC kit's instructions. Power up the Raspberry Pi and access the Terminal application from the icon shown at the top of the screen as shown in the next photo. Note: the terminal application is also accessible via the Menu - Accessories - Terminal menu selections.



The terminal application displays a `pi@raspberrypi: ~$` prompt as shown above. At the terminal application prompt (meaning right after `pi@raspberrypi : ~$` in the picture above), type in:

```
cd /etc/wpa_supplicant
```

after the prompt so that your screen will change and look like:



The step above used the “cd” or “change directory” command to change into the directory (e.g. the file folder) containing a configuration file for the Raspberry Pi’s Wifi connectivity. We will now edit this file and enter the SSID (Google “Wifi SSID” if you are not familiar with this acronym) and password of our local WiFi network.

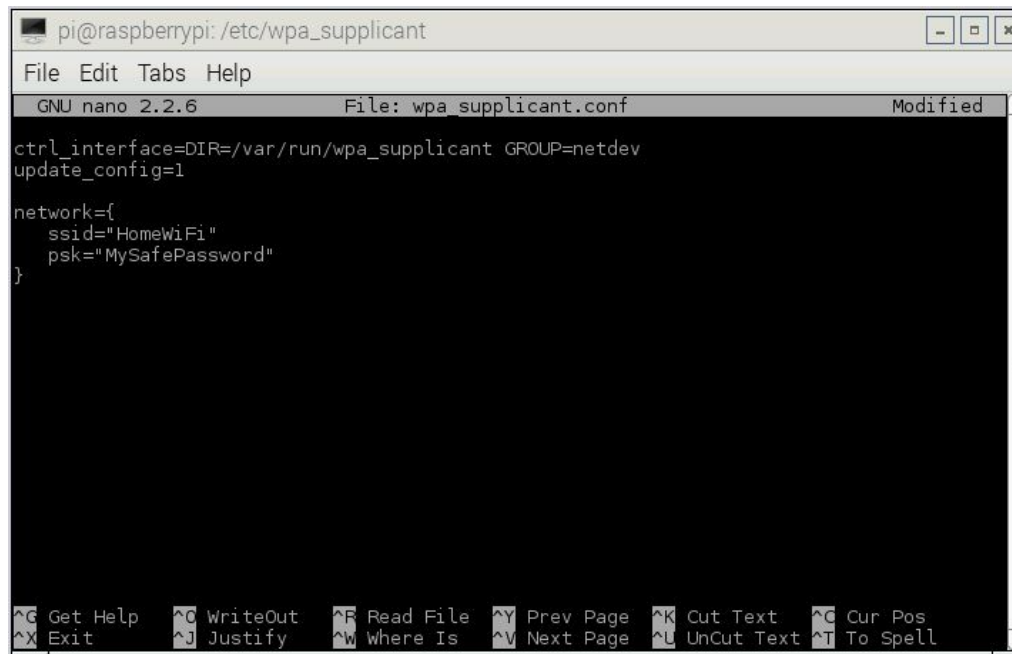
The file that we will be editing is `wpa_supplicant.conf`. This is a protected “system” file, so we will need to use the command “sudo” or **super user do** in order to provide the necessary permissions to make changes to the file. We will also use the “nano” application as our text editor.

At the terminal application prompt, type in:

```
sudo nano wpa_supplicant.conf
```

You are now in the nano text editor application editing the `wpa_supplicant.conf` file.

We need to add a reference to our local WiFi network’s SSID and password to this file so that our Raspberry Pi can connect to this WiFi network. The next screenshot displays an example of how to enter a connection to a fictitious WiFi network with SSID “HomeWiFi” and password “MySafePassword”.



```
pi@raspberrypi: /etc/wpa_supplicant
File Edit Tabs Help
GNU nano 2.2.6 File: wpa_supplicant.conf Modified
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="HomeWiFi"
    psk="MySafePassword"
}

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^L UnCut Text ^T To Spell
```

Do not touch the first two lines, add below these two lines as we did above. Be sure that you replicate the look of the added lines exactly; the opening and closing curly braces, use of ssid and psk to the left of the equal signs and double quotations around the SSID and password are all extremely important! Note that the password is “psk”; it’s easy to accidentally type “psw” so do be careful!

The nano text editor uses a combination of the key “Ctrl” (i.e. control) and another key to execute commands. Hold down the Ctrl (control) key and then select the O key to save the file. We will denote these two keystrokes as Ctrl + O.

After selecting Ctrl + O, note at the bottom of the screen the prompt to confirm you wish to retain the filename wpa\_supplicant.conf. Confirm this by selecting the Enter key, and then subsequently use keys Ctrl + X to exit the nano editor.

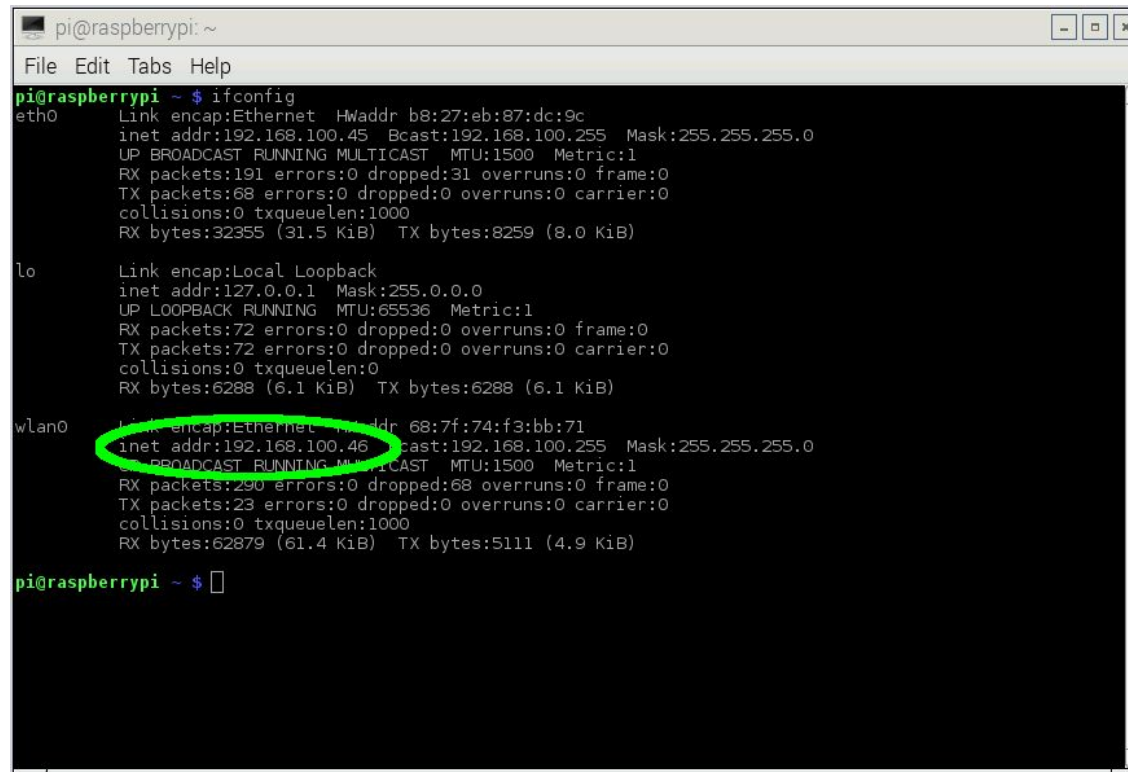
Next, back at the terminal application prompt, type in:

```
sudo reboot
```

... to reboot the Raspberry Pi. Rebooting will take a minute or two, after which you will need to open the terminal application again as we had earlier. At the terminal application prompt, type in:

```
ifconfig
```

This will display information regarding the connectivity of the Raspberry Pi, and example of which is shown in the following screenshot.



```
pi@raspberrypi ~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:87:dc:9c
          inet addr:192.168.100.45  Bcast:192.168.100.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:191 errors:0 dropped:31 overruns:0 frame:0
          TX packets:68 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:32355 (31.5 KiB)  TX bytes:8259 (8.0 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:72 errors:0 dropped:0 overruns:0 frame:0
          TX packets:72 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:6288 (6.1 KiB)  TX bytes:6288 (6.1 KiB)

wlan0     Link encap:Ethernet  HWaddr 68:7f:74:f3:bb:71
          inet addr:192.168.100.46  Bcast:192.168.100.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:290 errors:0 dropped:68 overruns:0 frame:0
          TX packets:23 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:62879 (61.4 KiB)  TX bytes:5111 (4.9 KiB)

pi@raspberrypi ~ $
```

On the left side of the display we see wlan0 which is our WiFi or wireless connection port of the Raspberry Pi. To the right and beneath wlan0 we can determine the IP address (please Google “IP address” if you are unfamiliar with this term) of the Raspberry Pi on the local network. Within this example screen the IP address of our Raspberry Pi is 192.168.100.46, but yours will almost certainly be different than that. **Write down YOUR IP address as we will be referring to it again in instructions ahead!**

If you DO NOT see wlan0 on the left side of the screen as shown above, then you are not connected to the WiFi network. The two most likely problems are: a) forgetting to install the WiFi dongle, b) improper editing of the wpa\_supplicant.conf file, or c) invalid WiFi SSID and/or password. If you can connect your phone to the WiFi network using the SSID / password you have recorded, and you are sure the WiFi dongle is installed (reminder: only install the dongle when the Raspberry Pi is powered down), then the problem is most likely to be editing of the wpa\_supplicant.conf. Repeat the steps above until the “ifconfig” command displays the Raspberry Pi’s IP address on the local WiFi network as shown in the screenshot above.

Each computer or device (i.e. phone, tablet, network connected thermostats, etc.) on a network has a unique IP address, and if we know that address we can “talk to” that device using another computer or phone. Now that we know the Raspberry Pi’s IP address (you did write it down, right?), we can “talk to” our Raspberry Pi from a laptop or phone.



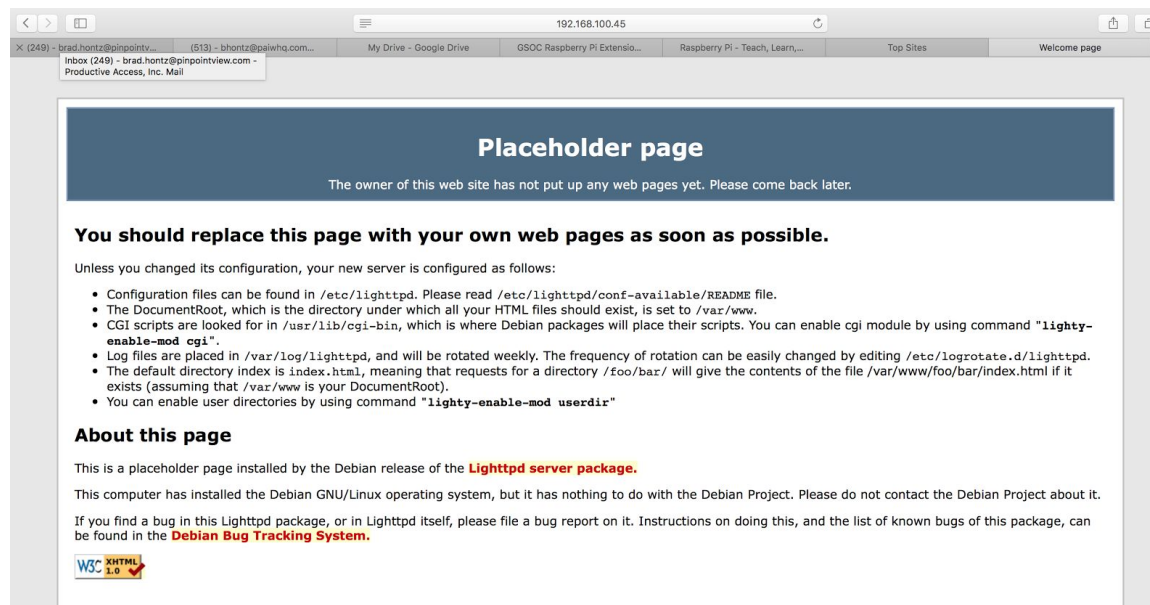
---

## Step 2: So what exactly is a “web server” ?

A “web server” is a software application that runs on a computer. When the web server application is running, other computers and devices (e.g. phones, tablets, etc.) connected to the same network can use a web browser application (e.g. Firefox, Safari, Google Chrome, Internet Explorer, etc.) to access specially formatted files located on the “web server” computer. We’ll refer to the web server computer as the “server” and the other devices connecting to the server as the “clients”.

As mentioned above, a web browser application is used on the client computer to access the server. We can enter the IP address of our server into the web browser application’s URL line (e.g. where you’d type <http://www.google.com> for example) to connect to the server. URLs, for example, [www.google.com](http://www.google.com), are just shortcut methods of providing the IP address to the web browser. You can always type a numerical IP address (like our example 192.168.100.46) directly into the web browser in place of a URL.

The GSOC kit’s Raspberry Pi should already have the LIGHTTPD web server software installed and running. To verify, connect another laptop, tablet or phone to the same WiFi network as the Raspberry Pi. Open the web browser on this “client computer” and type in the IP address of the Raspberry Pi that you recorded earlier. You should see the screen below, which is a default web page installed along with the LIGHTTPD web server application. If you see this screen then the Raspberry Pi is now officially a web server!



---

### Step 3: Breadboarding - connecting the temperature and humidity sensor to the Pi

Use the Raspberry Pi's main menu Shutdown menu option to turn off the Raspberry Pi. Once the screen is blank, pull the small micro usb power cable out of the Raspberry Pi to completely shut it off. **It is important that we completely power down the Raspberry Pi before connecting the Pi to the temperature and humidity sensor. Damage can result to the sensor or to the Raspberry Pi if we connect the two while the Raspberry Pi is “powered”.**

A “breadboard” (also called a “breakout board”) is a tool used by engineers to prototype, or “experiment with” electronic circuits. The breadboard’s holes allow for convenient connections of components without the need to permanently solder the connections together.

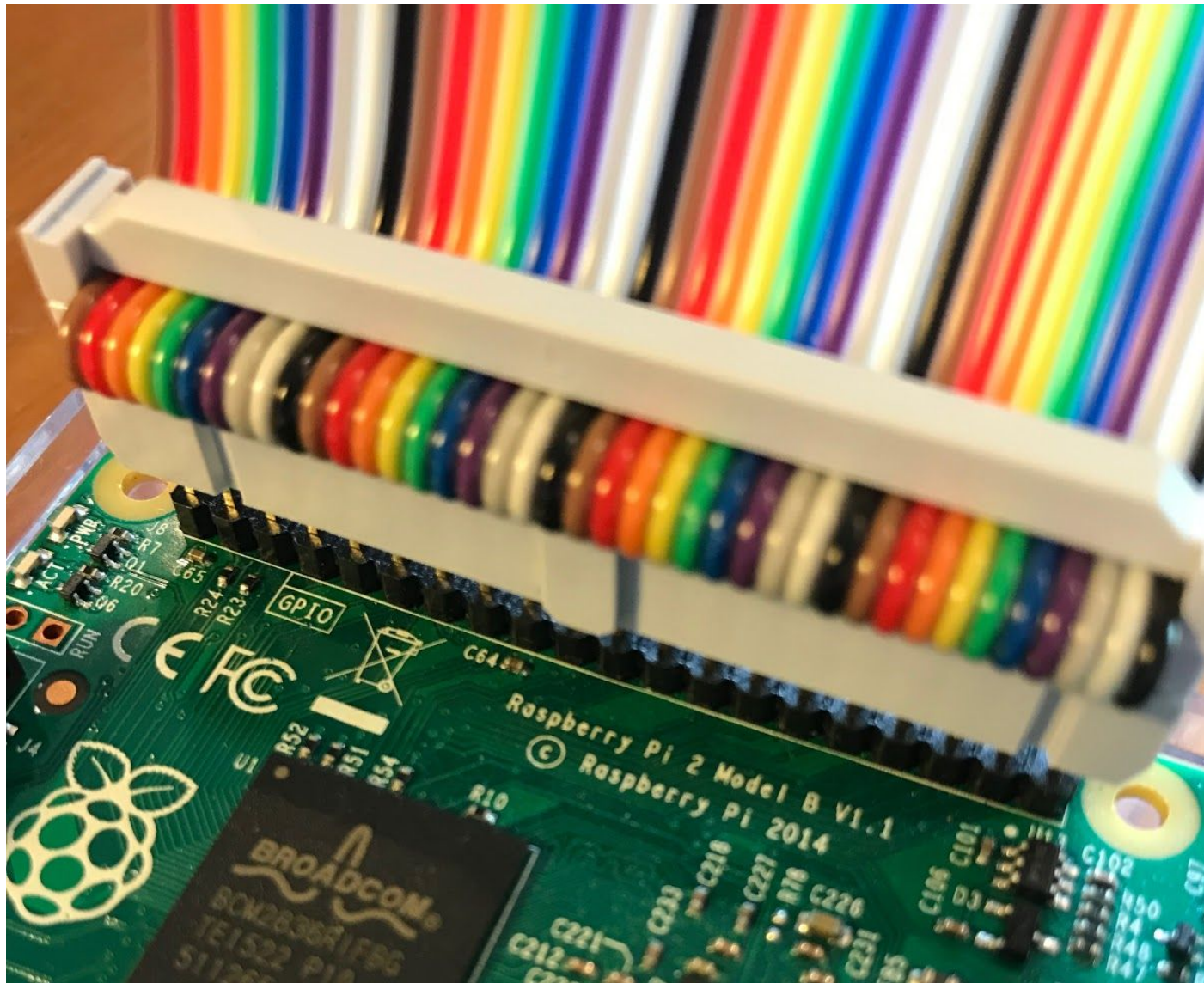
The temperature and humidity kit provided within the GSOC Raspberry Pi box comes with a special cable called a “cobbler” that allows the Raspberry Pi to be conveniently connected to the breadboard.

Shown below are a series of pin connectors on the Raspberry Pi, which are called the General Purpose Input and Output bus, or **GPIO** for short. The “cobbler” ribbon cable connection to breadboard (second image shown below) contains a visual schematic depicting the meaning of each of the Raspberry Pi’s GPIO pins. Note that pin on the top left is labeled 3v3 (+3.3v) whereas the 3rd pin from the top on the right is labeled GND (or “ground”). These two pins are the equivalent of the positive (+) and negative (-) poles of a 3.3v battery, and we will utilize these two pins to provide power to our temperature and humidity sensor.

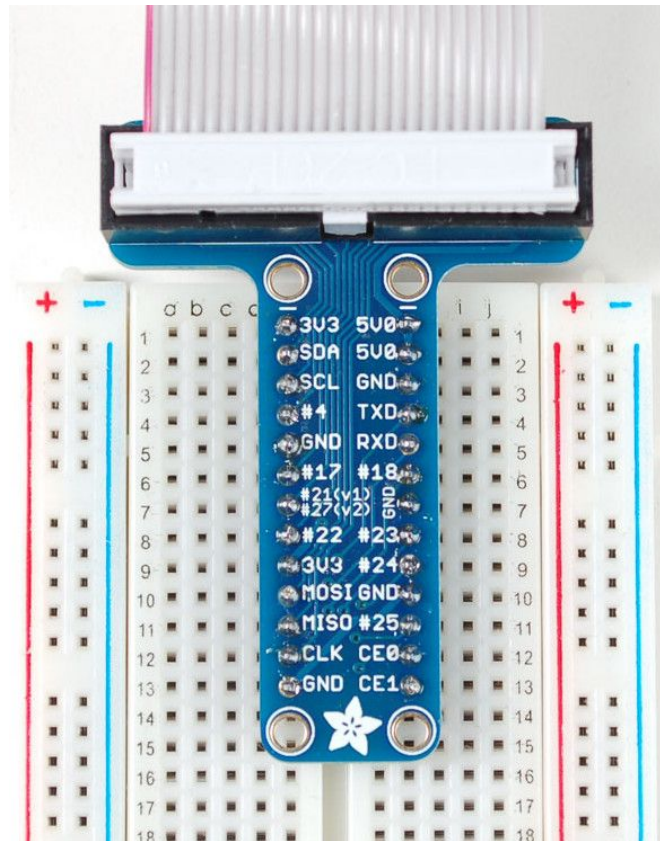
There are a number of pins that have a # and number (e.g. #4, #17, #22, #23 ...) displayed on the cobbler cable as shown below. Each of these pins represents a unique “data port” which can be used to communicate readings between the temperature sensor and our Raspberry Pi. We will need to use only one of these GPIO data ports for this project, but a more complicated sensor project (e.g. using a camera) may require the use of several GPIO data ports at once. For our purposes, it doesn’t really matter which GPIO data port we select to use, we just need to remember the GPIO data port number that we elected to use!

Let’s start by connecting the “cobbler” ribbon cable to the breadboard and connecting the other end of the ribbon cable to the Raspberry Pi as shown in the next image. Take note of the direction of the ribbon cable, you can see the “bump” at the center of the ribbon cable plug below, this plug should face towards the center of the Raspberry Pi as shown.





Now connect the ribbon and “cobbler” (if it isn’t already) to the breadboard as shown in the following image:



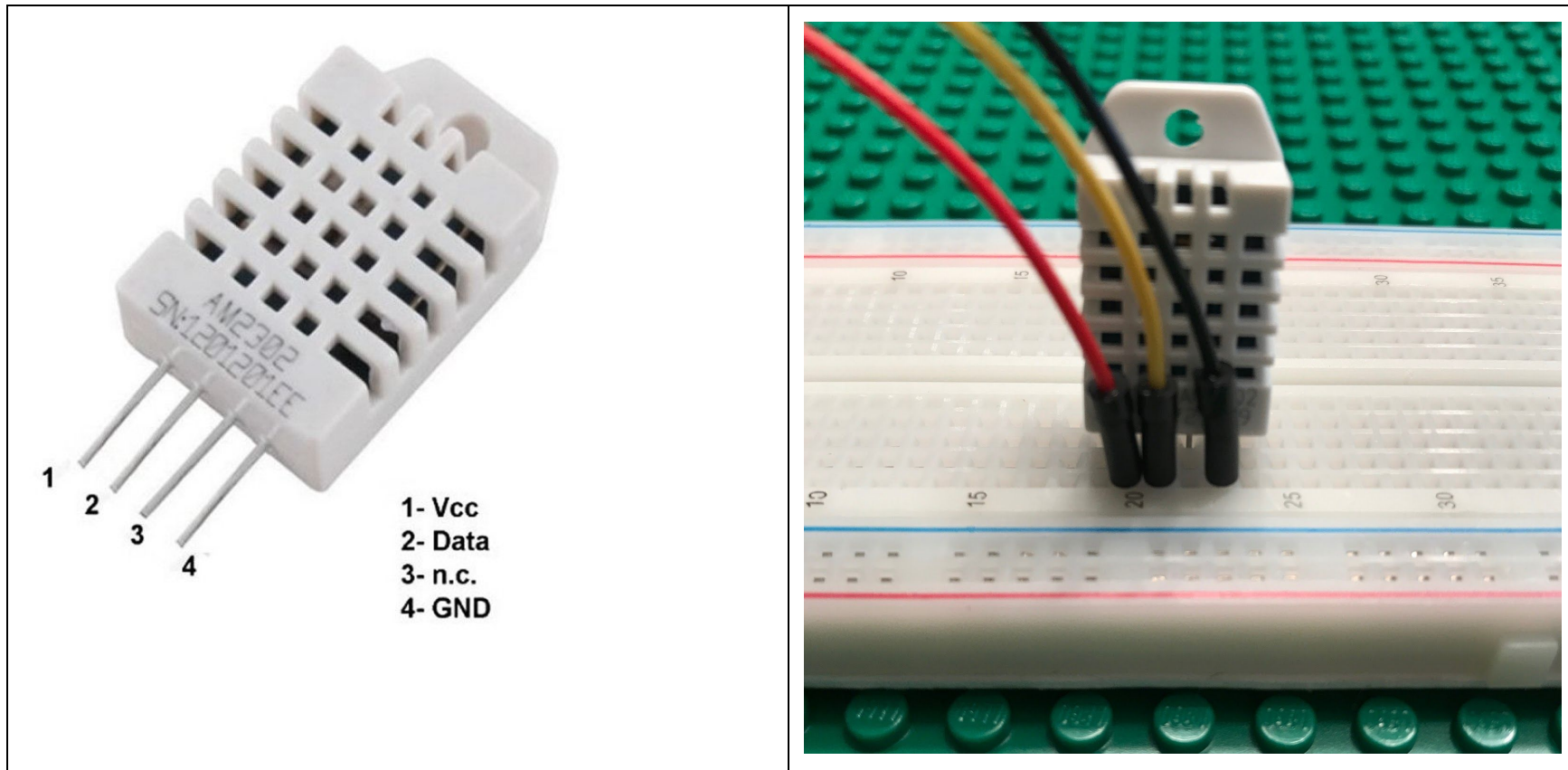
*Above: the breadboard and “cobbler” with one end of the ribbon cable attached to the cobbler.*

Looking at the breadboard in the previous picture, holes in the breadboard that are within the same column ‘a’, ‘b’, ‘c’, etc. are INDEPENDENT of one another, or said another way, they are not “electronically” connected together. Conversely, breadboard holes that are in rows 1, 2, 3, etc. are each connected to holes within that same row across columns ‘a’ through ‘e’, and then again across columns ‘f’ through ‘j’. The break (i.e. deep groove) in the center of the breadboard between columns ‘e’ and ‘f’ means that columns ‘a’ through ‘e’ and ‘f’ through ‘j’ are also independent of one another. We won’t be using the + and - columns as shown above, but just be aware that these breadboard columns act differently; holes within the + column and - columns (respectively) ARE connected together!

With the understanding of the paragraph above, if we were to plug a wire into the column ‘a’ - row 1 hole of the picture above, then our wire would be connected to the 3v3 pin of the Raspberry Pi’s GPIO. Similarly if we were to plug a wire into column ‘j’- row 3 hole

of the picture above, that wire would be connected to the GND (i.e. “ground”) of the Raspberry Pi’s GPIO. We can now connect our temperature and humidity sensor (i.e. the DHT22 sensor) to our Raspberry Pi, using wires and the breadboard.

Remembering that the columns (i.e. a, b, c, etc.) of our breadboard are NOT connected together, so we can CAREFULLY plug our DHT22 sensor’s four pins into a single column below our cobbler cable connection. Columns ‘d’ or ‘e’ on the breadboard are ideal for this purpose. Carefully record the row numbers of the four DHT22 sensor pins as you plug in the sensor. You should be counting the pins from left to right with the front of the sensor facing you as shown in the picture below.



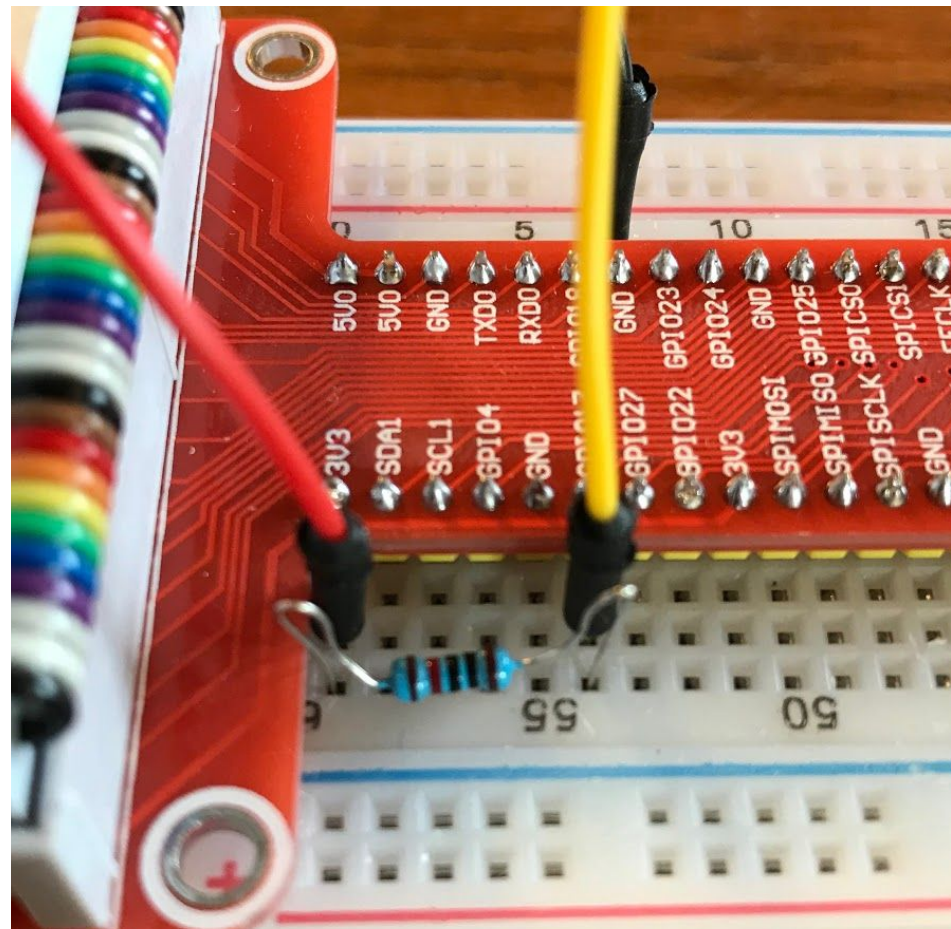
In the breadboard picture on the right above, pin #1 of the DHT22 sensor was placed within row 20 of the breadboard, pin #2 in row 21, and pin #4 in row 23. Note that the picture on the left indicates that pin #1 is “Vcc” which for our purposes means 3v3, pin #2 is the data (communication) pin which we will connect to one of the Raspberry Pi’s GPIO numbered ports, and pin #4 is GND or ground. Pin #3 of the DHT22 is labeled “n.c.” which means “no connection” or “unused”. You can choose other breadboard rows



(e.g. 25,26,27,28 for example) for the DHT22's pins if you'd like; just be sure to carefully record which pins of the DHT22 are in which rows of the breadboard, and be sure that all four of the DHT22's pins are within the same breadboard column.

Our DHT22 sensor requires a resistor between the DHT22's pin #1 (Vcc) and pin #2 (data) to work properly for our application. We need to use a 10k (i.e. 10,000) ohm resistor for this purpose, and this resistor is included within the GSOC Raspberry Pi kit. **Don't proceed without the resistor! The resistor is important to include as it protects the Raspberry Pi from potential damage.**

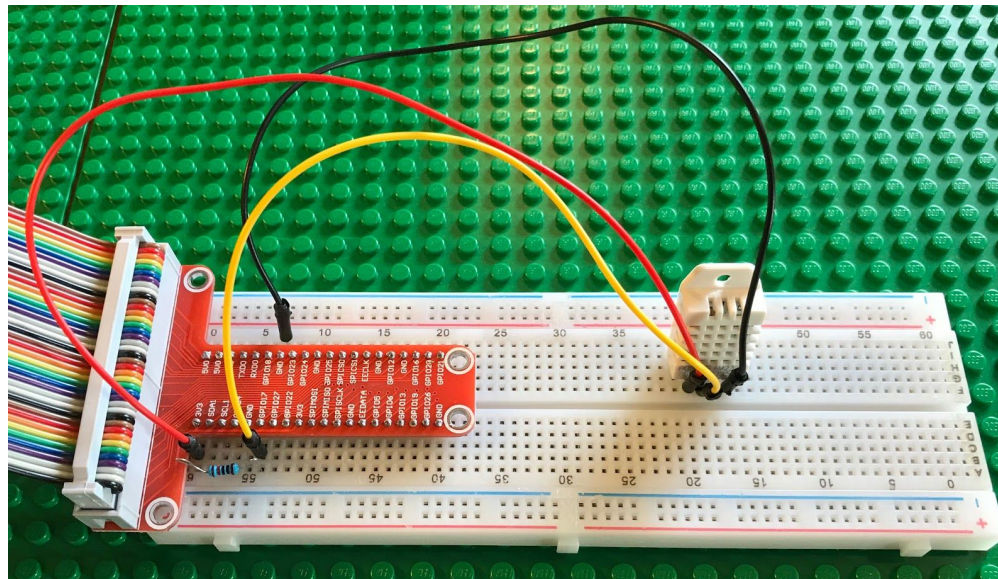
Find the 10k ohm resistor (color stripes are brown, black, black, red, brown) as per the picture below and **very carefully bend the leads so that it can fit between 6 holes** on the breadboard as shown in the following image. You don't have to place the resistor in the breadboard just yet, just prepare the resistor by bending it in this manner shown below (you can use your fingers to bend the leads). The resistor can be inserted in either direction.



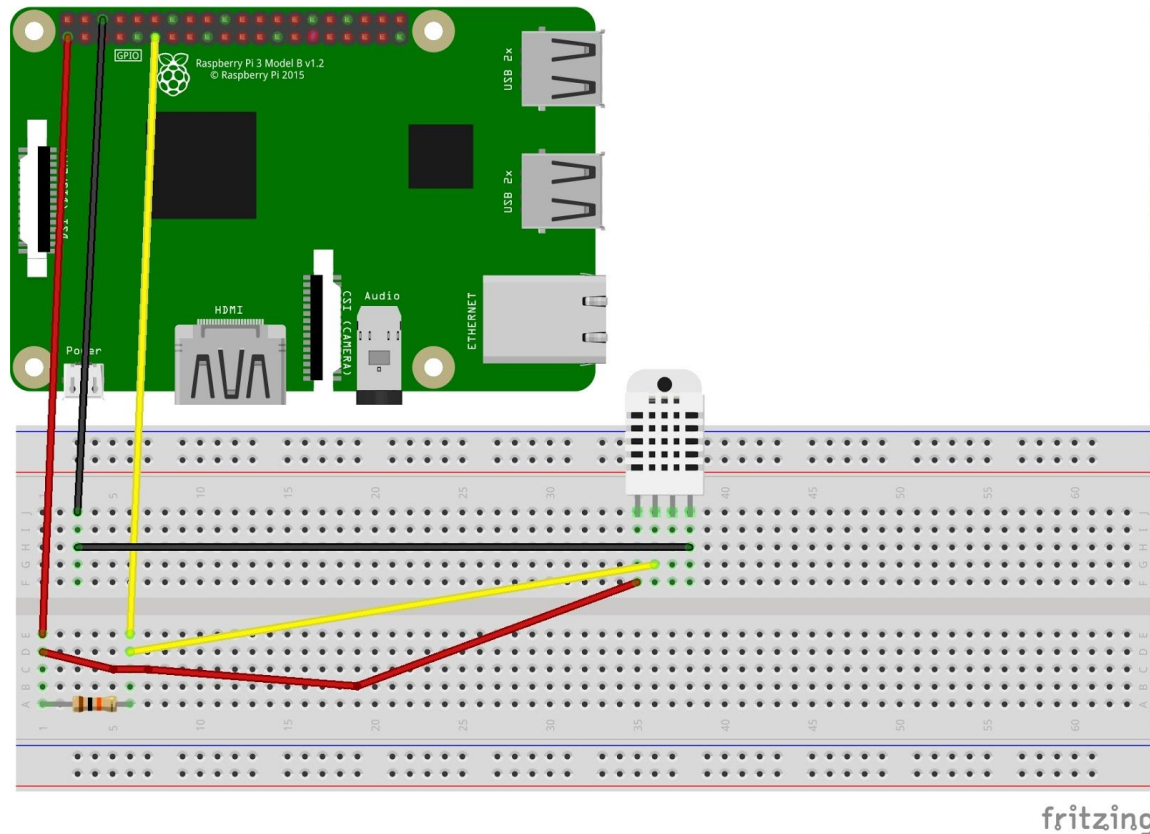
Now we can connect the DHT22 sensor, the resistor and the cobbler ribbon cable together using any three of the small jumper wires that are included with the kit. Previously we noted that the DHT22's sensor receives power from the Raspberry Pi by connecting the sensor's pin #1 to the 3v3 pin of the Raspberry Pi, and the sensor's pin #4 to a GND (ground). We can connect the Raspberry Pi's 3v3 pin to DHT22's pin #1 with a jumper wire between column 'a' - row #1 and column 'a' - row #20 of the breadboard (using our example above). Similarly, connect the Raspberry Pi's GND (ground) pin to the the DHT22's pin #4 with a jumper wire between column 'j' - row #3 and column 'a' - row #23 of the breadboard. Traditionally we'd use a red wire to represent 3v3 and a black wire to represent ground, but any wire color will work.

Now let's connect our DHT22's data wire, pin #2 on our DHT22. We'll elect to connect our data wire to the Raspberry Pi's **GPIO port #17** (remember this number when we talk about the python script below!), which is row #6 on the left side of our cobbler ribbon cable. Connect the data wire from column 'a' - row #6 to column 'a' - row #21 of our breadboard. Elect a different color wire for this purpose; yellow is often used to represent "data".

Lastly we need to connect the 10k resistor between pins #1 and #2 of the DHT22 sensor. It would be very inconvenient to bend the resistor to fit into two side by side holes of the breadboard, so instead we will place the resistor between the 3v3 and GPIO port #17 of the Raspberry Pi, which are six holes apart (rows 1 and 6). Place one end (doesn't matter which end) of the resistor in column 'c' - row #1 and the other in column 'c' - row #6. Note that the wires in these same two rows also connect to pins #1 and #2 of our DHT22 sensor, so we are meeting the sensor's requirements by mounting the resistor in this way.



We now have all of our required connections in place! The *schematic* below illustrates the connections we've made between our DHT22 sensor and our Raspberry Pi's GPIO pins using our data wires and cobbler ribbon. Please compare your connections and the schematic below to be sure your connections correspond before proceeding. *Note: the cobbler ribbon cable is not shown, just the relevant Raspberry Pi GPIO pin to DHT22 sensor connections.*



Key to above: GPIO pin 1 (3.3v) connects to left most DHT22 pin (Vcc), GPIO pin 11 (data port #17) connects to the 2nd from the left DHT22 pin (Data), GPIO pin 6 (GND) connects to the right most DHT22 pin (GND). The 10k ohm resistor connects between the GPIO pin 1 and GPIO pin 11 (i.e. the 3.3v and data port #17).

After carefully confirming the breadboard connections, reconnect the Raspberry Pi's micro USB power cable to restart the Raspberry Pi.

---

**Step 4: Coding - writing the Python script that will read the sensor and display the results on a web page**



“Python” is a computer programming language that is included with the Raspberry Pi kits. This section is not meant to be a python coding lesson, nor do you really need to know anything about the python language in order to complete this step. We’ll simply walk through the components of a python script provided here and teach you how to author the script directly on the Raspberry Pi. This section assumes that the Raspberry Pi is powered on and is connected to the local WiFi network. The remainder of tasks within this step require use of the Raspberry Pi’s terminal application as we have in previous steps.

First off, the script file that we will create needs to be created with the Raspberry Pi’s web server default content directory. If you look at the “Placeholder page” picture within the web server section of this document and read that page’s content, you’ll note that the web server expects web page content (i.e. the “document root”) to be placed in the folder /var/www. We need to first change into this folder before creating our new script file, which we can do with “cd” command we’ve used previously. From the Pi’s terminal application, change into the /var/www folder by typing the following command at the terminal application prompt:

```
cd /var/www
```

The terminal application’s prompt will change reflecting the /var/www folder. Next type in the following command:

```
nano my_weather_station.py
```

This will place us back in the nano text editor application, but this time we’re creating a new file so the nano’s application screen appears blank! *Note: this file may already exist if previous users of the GSOC Raspberry Pi kit already completed this step!*

Type in the text of the python script as shown below, being very careful to utilize the same indentations as shown below, and as always when copying code it is critical to copy it exactly! If you would prefer to download this file it is available at this link: [https://github.com/bhontz/raspberrypi\\_weather\\_station/blob/master/my\\_weather\\_station.py](https://github.com/bhontz/raspberrypi_weather_station/blob/master/my_weather_station.py).

```
import sys, Adafruit_DHT, time
from wsgiref.simple_server import make_server

html = """
<html>
<style>
    body {font-family: Arial, sans-serif; font-size: 96px;}
</style>
<body>
    <p align="center">Time: %s</p>
```

```

    <p align="center">Temp: %s</p>
    <p align="center">Humidity: %s</p>
</body>
</html>
"""

def application(environ, start_response):

    sensor = Adafruit_DHT.DHT22
    pin = 17 # The Raspberry Pi GPIO PIN we elected to use!

    humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
    temperature = (temperature * 9.0 / 5.0) + 32.0 # conversion from Celsius to Fahrenheit

    response_body = html % (time.strftime("%H:%M:%S", time.localtime()), "%0.1f" % temperature, "%0.1f" %
humidity)

    status = '200 OK'

    response_headers = [('Refresh', '5'), ('Content-Type', 'text/html'),
                        ('Content-Length', str(len(response_body)))]
    start_response(status, response_headers)

    return [response_body]

if len(sys.argv) < 2 or sys.argv[1] == None or sys.argv[1] == "" :
    print "Usage: %s IP_ADDRESS_OF_Raspberry_Pi" % sys.argv[0]
    sys.exit(0)

httpd = make_server(sys.argv[1], 8051, application)
httpd.serve_forever()

```

Once you have entered the code into the nano editor and carefully checked it, use the commands Ctrl + W (i.e. hold down the Ctrl key while selecting the W key) to save the script file. You will need to enter your file name (e.g. my\_weather\_station.py) and confirm with the Enter key. Use the key commands Ctrl + X to exit the nano editor.

Before we run our python script, let's first review our script in detail to understand what it does. The first two lines:

```
import sys, Adafruit_DHT, time
```

```
from wsgiref.simple_server import make_server
```

... reference additional pre-existing python code modules that our script “builds upon”. One of these additional modules, Adafruit\_DHT, comes with the DHT22 sensor and contains the code to read our sensor.

The next set of lines within the script, as shown below, define a new variable we name *html*:

```
html = """
<html>
<style>
    body {font-family: Arial, sans-serif; font-size: 96px;}
</style>
<body>
    <p align="center">Time: %s</p>
    <p align="center">Temp: %s</p>
    <p align="center">Humidity: %s</p>
</body>
</html>
"""
```

Our new html variable contains the HTML syntax that we will be sending to client computers connecting to our Raspberry Pi web server. We use three HTML syntax “paragraphs” (<p></p>) containing Time, Temp and Humidity respectively. Within the HTML syntax “style” (<style></style>), we define a font-size of 96 points, so these lines will be appear very large! Feel free to experiment and add an additional row, like <p align="center">Weather by Raspberry Pi!</p> immediately above the “Time:” line, make the font size smaller or larger, etc.

Now we get into the main part of our script which is represented by a function that is called “application”. This function must be named identically to the reference within the script: `httpd = make_server(sys.argv[1], 8051, application)` which is found at the end of our script. This function must also have two arguments, `environ` and `start_response`, as shown below:

```
def application(environ, start_response):
```

Within our application function, we create a variable *sensor* that contains the Adafruit\_DHT module’s code for the DHT22 sensor.:

```
sensor = Adafruit_DHT.DHT22
```

Next, we create a variable *pin* which we set to the GPIO # that we'd connected our "data" wire to. Remember that we needed to connected our data wire to the Raspberry Pi's GPIO port #17? :

```
pin = 17 # The Raspberry's GPIO PIN we elected to use!
```

Next we read the Adafruit\_DHT's module's function called `read_retry` which takes our two new variables, *sensor* and *pin*, as arguments. The `read_retry` function then returns two new variables, *humidity* and *temperature* (in that order) which represent the sensor's readings.

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

Now we do something a little tricky in the next line of script. Do you recall seeing the "%s" symbols within the html variable we discussed above? The "%s" symbols are "placeholders" that can be filled in with values, and that's what we do within this line:

```
response_body = html % (time.strftime("%H:%M:%S", time.localtime()), "%0.1f" %  
temperature, "%0.1f" % humidity)
```

We replace the "%s" on variable *html*'s time, temp, and humidity paragraphs (in that order) with the local time formatted to H:M:S (hours:minutes:seconds). We use a "formatting picture" of %0.1f for temperature and humidity, which provides for one decimal place of precision when displaying these variables. We then create a new variable called *response\_body* which is set equal to our previous *html* variable with these three new values replacing variable *html*'s "%s" placeholders.

The final lines within the application function send HTML commands that our webserver must communicate to our client computers before it sends our web page. The function `start_response` passes an HTML *status* variable that we initially define within our script as "200 OK", and separately a *response\_headers* variable representing a required HTML header defining the type and size of the HTML page (i.e. our variable *response\_body*) that we subsequently send to client computers. Within our *response\_headers*, we define a "Refresh" header parameter which makes our web page automatically refresh, and by doing so, concurrently refreshes the temperature and humidity settings.

Lastly, on the final line of our function, we "return" the variable *response\_body*, which is our completed web page as represented by variable *response\_body*.

```
status = '200 OK'
```

```

response_headers = [('Refresh', '5'), ('Content-Type', 'text/html'),
                    ('Content-Length', str(len(response_body)))]
start_response(status, response_headers)

return [response_body]

```

The next block of code checks to see if we remembered to run our script with an parameter (i.e. values we specify on the command line in addition to our script's name). The parameter we'll use is the IP address of our Raspberry Pi web server. By passing the IP address to the script as a parameter, we can conveniently connect the Raspberry Pi to different networks without editing our python script. Remember that on each WiFi network our Raspberry Pi will have a different IP address.

```

if len(sys.argv) < 2 or sys.argv[1] == None or sys.argv[1] == "" :
    print "Usage: %s IP_ADDRESS_OF_Raspberry_Pi" % sys.argv[0]
    sys.exit(0)

```

Based upon the code fragment above, if we were to try to run this script and forget to specify the IP address of the Raspberry Pi after our script's name, we'd receive the message:

```
Usage: my_weather_station.py IP_ADDRESS_OF_Raspberry_Pi
```

This message serves as a reminder to rerun the script, but this time add the IP address of the Raspberry Pi after the script's name.

The final two lines of our script:

```

httpd = make_server(sys.argv[1], 8051, application)
httpd.serve_forever()

```

... **starts the hosting of our web page at “port 8051” of our IP address**, which our client computers will need to reference (more on that below). “serve\_forever” means that this script will run forever unless it is interrupted in some way. We can use the Raspberry Pi terminal application's Ctrl + C commands to stop or break out of the running script. Now let's start running our script!

## **Running our script on the Raspberry Pi**

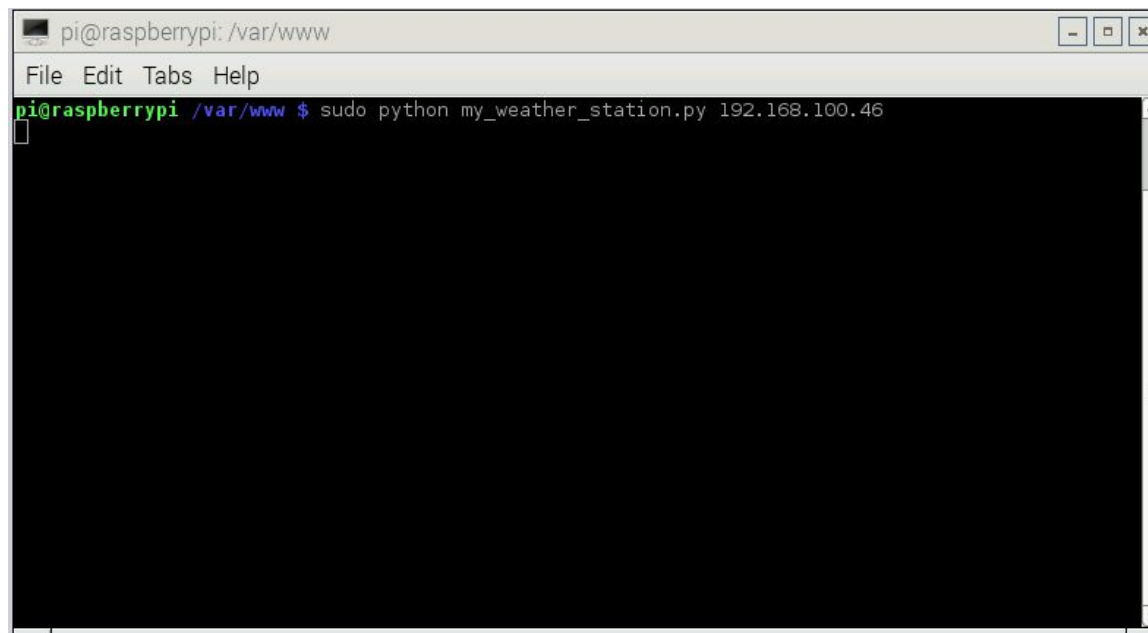
After exiting the nano text editor application from within the Raspberry Pi's terminal application, we can run our script which will start reading our breadboard's temperature and humidity sensor and display it via our webserver. We should still be within the /var/www folder of our Raspberry Pi; which as you may recall is the folder containing our webserver content and our python script.

Within the terminal application, type:

```
sudo python my_weather_station.py 192.168.100.46
```

Note that we need to use “sudo” before running python because our AdaFruit module's code requires super user access. Also note I've used the Raspberry Pi IP address 192.168.100.46 consistent with the examples within this document, **but you will almost certainly have a different IP address for your Raspberry Pi. Be sure to use the IP address of your Raspberry Pi!**

The next screenshot shows the terminal application window after entering the command above and hitting the Enter key:



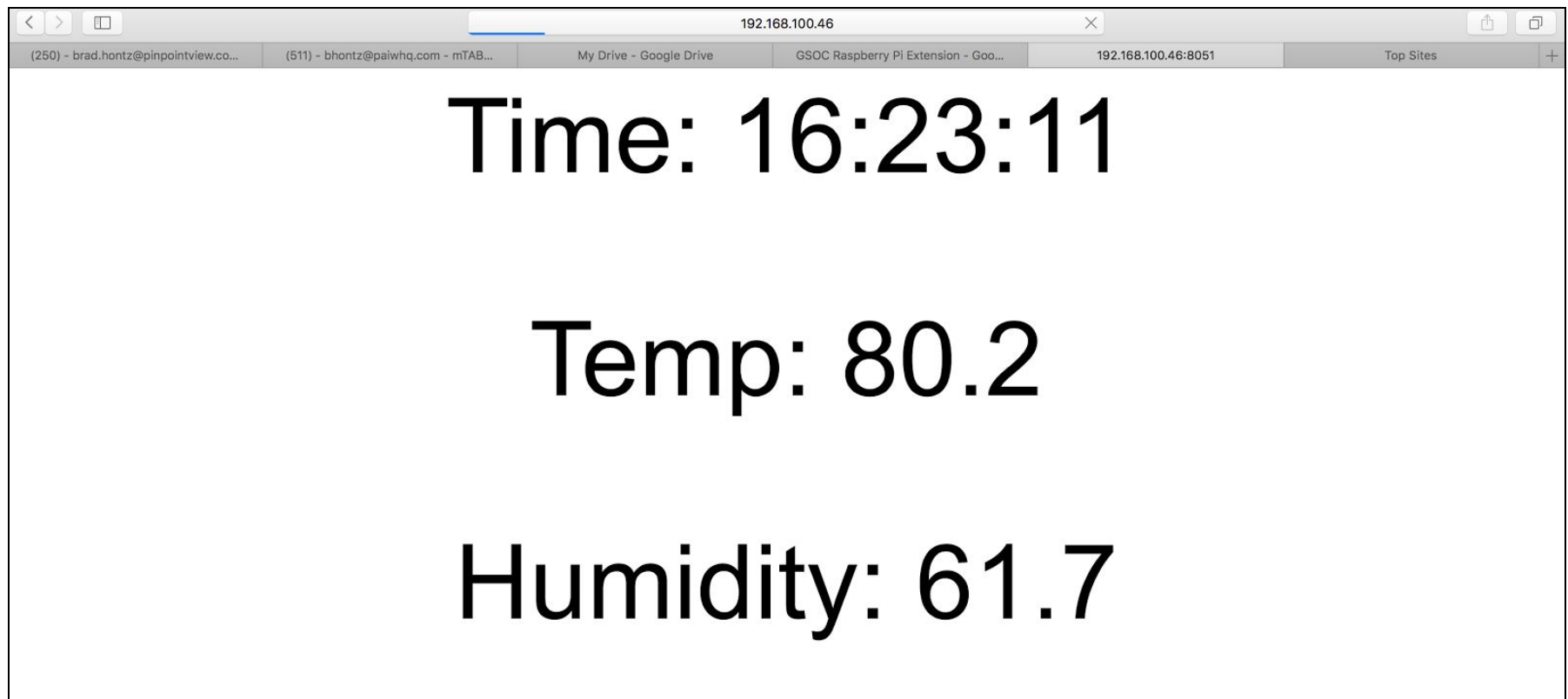
**It appears as if nothing is happening!** Let's connect a client computer to our webserver and see if our script is really working!

First make sure your client computer (i.e. your laptop, tablet or phone) is on the same Wifi network as your Raspberry Pi. Next, open up the web browser on your client computer and enter the Raspberry Pi's IP address followed by :8051 on the line you'd normally



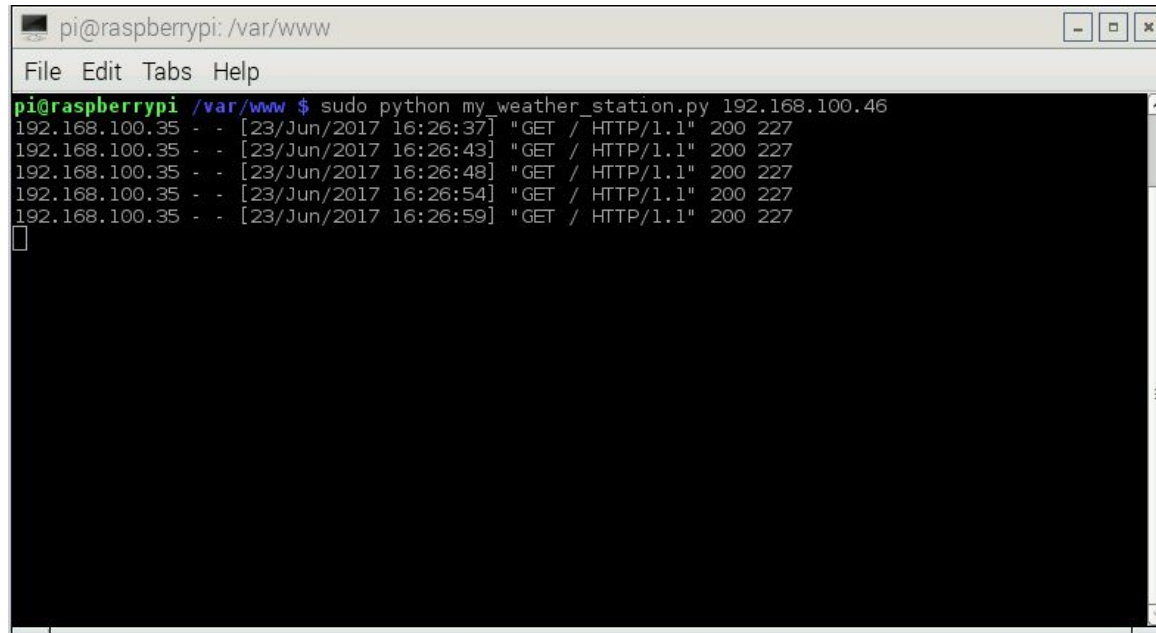
enter URLs like “<http://www.google.com>”. **Don’t forget to add :8051 to the end of the Raspberry Pi’s IP address!** (remember the reference to the 8051 “port” within our python script discussion?) If you do not add this additional port reference, you will simply see the Placeholder Page on your web browser shown on page 7 of this document.

To illustrate, here’s what my Mac laptop’s web browser looks like after connecting to the Raspberry Pi’s IP address 192.168.100.46 at port 8051 (i.e. I entered **192.168.100.46:8051** into my browser’s address bar):



**Success!** We’re reading the time, temperature and humidity from our Raspberry Pi on my laptop! Note that this page “refreshes” every minute or so. You can blow onto the DHT22 sensor (or even more carefully, use a small hairdryer) and note how the temperature changes when the web page next refreshes.

Additionally, after a client computer makes a connection, you’ll note that the Raspberry Pi’s terminal window will start to display “log messages” as per the example in the following picture.



```
pi@raspberrypi: /var/www
File Edit Tabs Help
pi@raspberrypi /var/www $ sudo python my_weather_station.py 192.168.100.46
192.168.100.35 - - [23/Jun/2017 16:26:37] "GET / HTTP/1.1" 200 227
192.168.100.35 - - [23/Jun/2017 16:26:43] "GET / HTTP/1.1" 200 227
192.168.100.35 - - [23/Jun/2017 16:26:48] "GET / HTTP/1.1" 200 227
192.168.100.35 - - [23/Jun/2017 16:26:54] "GET / HTTP/1.1" 200 227
192.168.100.35 - - [23/Jun/2017 16:26:59] "GET / HTTP/1.1" 200 227
```

You can connect multiple client computers (i.e. others can use their phones or laptops) in the same way, for example, you could have each of your girls connect their own phones to the WiFi network and enter the Raspberry Pi's IP address:8051 into the phone's web browser!

After you've completed this experiment, use the Raspberry Pi terminal window's commands Ctrl+C to stop the script running if you want to have access to the terminal window, or you can just shut the Raspberry Pi down when you're done by removing the Raspberry Pi's micro usb power cable.

## Step 5: Post-project discussion and learnings

Thanks for participating in this workshop, we've covered a lot of ground here and it would be helpful to review the key aspects of this project's learnings with your girls:

- 1) The Raspberry Pi is so much more than just a "small computer"! The GSOC kits as configured allow the girls to learn how to set up a computer and play a game or browse the web, but we've shown here that the Raspberry Pi is capable of much more; we can READ A SENSOR and REACT in some manner. While we "reacted" by displaying the sensors readings on a web page, we could have used the Raspberry Pi to physically control a device by turning it on or off, for example.

- 2) Within this project we turned the Raspberry Pi into a webserver and “broadcast” the results of our temperature / humidity sensor readings to phones and laptops connected to the same network as our Raspberry Pi. What other similar projects could we create that uses this same model? What if our Raspberry Pi connected to weather, surf report, or other similar external websites to obtain data? Couldn’t we broadcast the external information we obtained in this same way?
- 3) Talk about the “Internet of Things” (IoT) or ask the girls to research this subject to discuss in a subsequent meeting. In a nutshell, the Internet of Things entails sensing and control of devices within your home or office, using the internet as the local network. When the internet is your “local network” you can control these devices from any location. Talk about how this project could potentially serve as the underpinnings of a home automation thermostat, or similarly, controlling a home humidifier.
- 4) Expanding on point 3, brainstorm other types of sensors / controllers that could be potentially be created using the Raspberry Pi kits. Here are some ideas:
  - Pool pH / chlorine level sensor and chemical dispenser.
  - Potted plant water (moisture) sensor and automated watering.
  - Motion sensors - creating a burglar alarm, turn on lights when entering a room.
  - Light sensors - control home lighting (when it’s dark, turn on the lights!).
  - Using a CAMERA as a sensor - identify movement or specific objects (people vs. pets for example) and react in some way (send a text, sound an alarm, etc.).
  - Robotics - robots use sensors to determine their place in the environment and to know how to react, could a Raspberry Pi serve as the “brains” of a robot?

Other resources for girls interested in pursuing this type of project:

Hackster.IO ([www.hackster.io](http://www.hackster.io))

Requires creation of an account or using a Facebook account to join. Peer to peer sharing of electronic projects that are similar to this project, literally thousands of project ideas.

Adafruit ([www.adafruit.com](http://www.adafruit.com))

A vendor of electronic equipment, you can purchase project materials directly from the adafruit website. The website contains tons of ideas and project “recipes”.

Subscribe to the Raspberry Pi “MagPi” magazine: <https://www.raspberrypi.org/magpi/subscribe/>

Attend a Raspberry Pi Jam: <https://www.raspberrypi.org/jam/> The GSOC hosted a booth at 2017 Raspberry Pi Jam held at the Discovery Cube in Orange, CA and exhibited this very project!

Check out local “maker spaces” like Vocado in Riverside: <https://www.vocademy.com>

Get inspired by the works of others. Coco at <http://www.veryhappyrobot.com> is a great local example!

Join an existing GSOC First Lego League (FLL) or create a new team. See <http://www.firstlegoleague.com> and/or speak with the GSOC STEM programming team.

If you have any questions regarding this project or would like to discuss other similar project ideas, please feel free to drop me a line:

Brad Hontz  
GSOC STEM program volunteer  
Email: [brad.hontz@pinpointview.com](mailto:brad.hontz@pinpointview.com)

Last updated: 12/28/2017