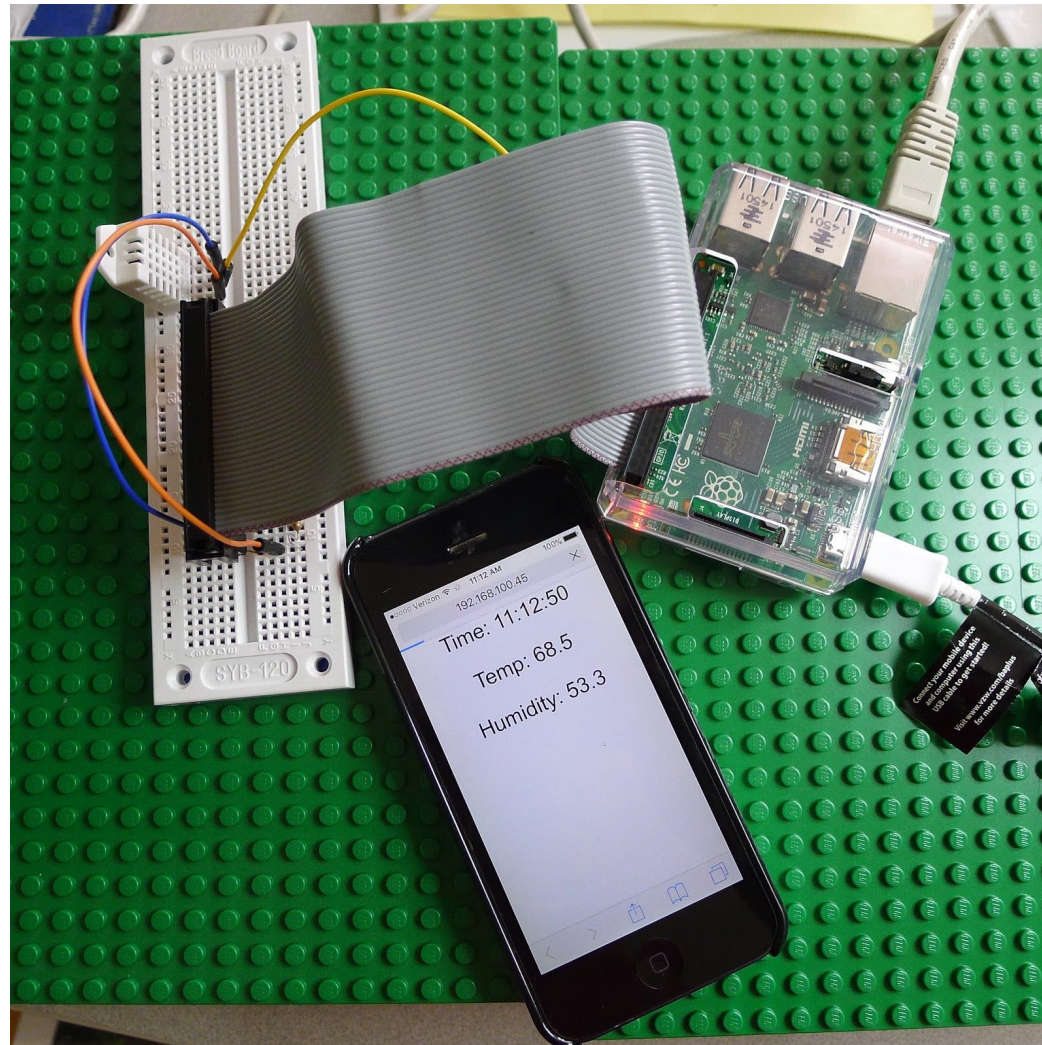


GSOC Raspberry Pi “Build a Weather Station” Project - Step by Step Instructions:

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 the temperature and humidity sensor's readings and display the results on a web page.

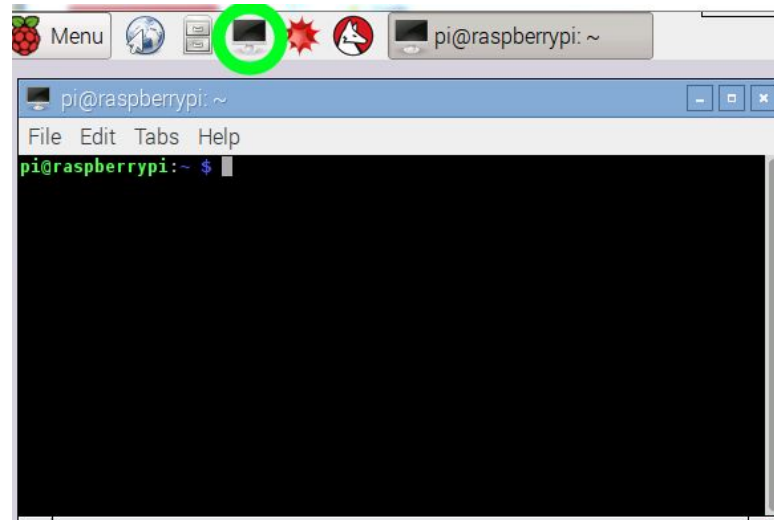
Step 1: Connecting to a WiFi network

For this step you will need to know a local WiFi network's SSID (i.e. the network's name) and password, just like you would need to connect your phone to a 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) plug the WiFi dongle into any one of the four USBs port on the Raspberry Pi as shown below.



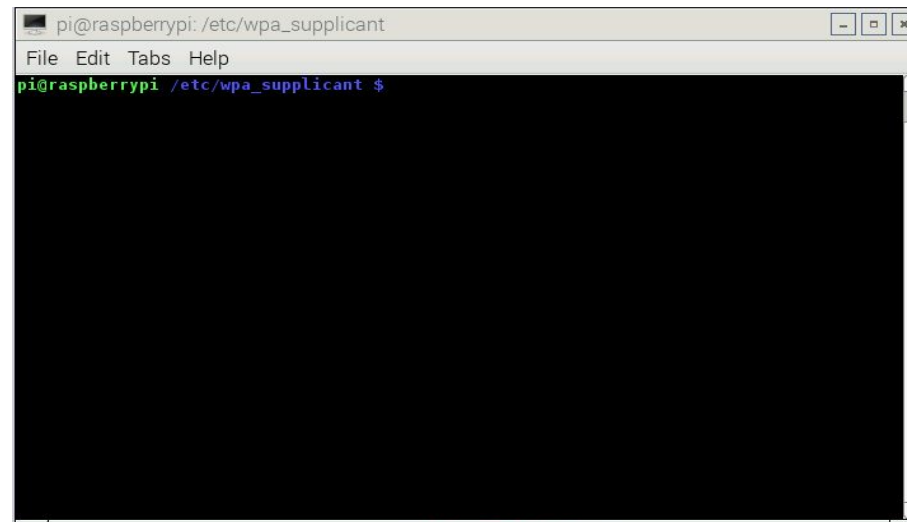
Connect the monitor, keyboard and mouse to the Raspberry pi as per the GSOC kit's instructions. Power up the Raspberry Pi and access the Terminal application from the icon show at the top of the screen as shown below. Note: the terminal application is also accessible via the Menu - Accessories - Terminal 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 (i.e. file folder) containing a configuration file for the Raspberry Pi’s Wifi connectivity. We will edit this file and enter the SSID (Google this if you are not familiar with this acronym) and password of our local WiFi network.

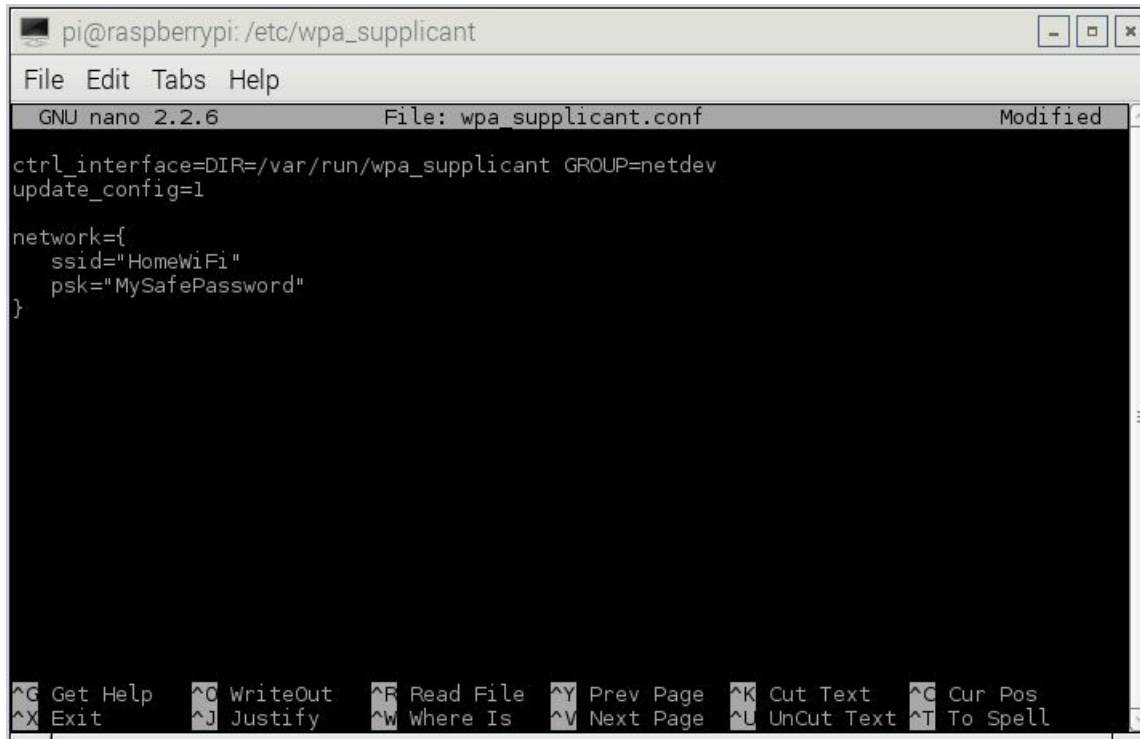
The file that we will be editing is `wpa_supplicant.conf`. This is a protected “system” file, so we will need to use the command “sudo” or **super user do** in order to provide the necessary permissions to make changes to the file. We will also use the “nano” application as our text editor.

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.

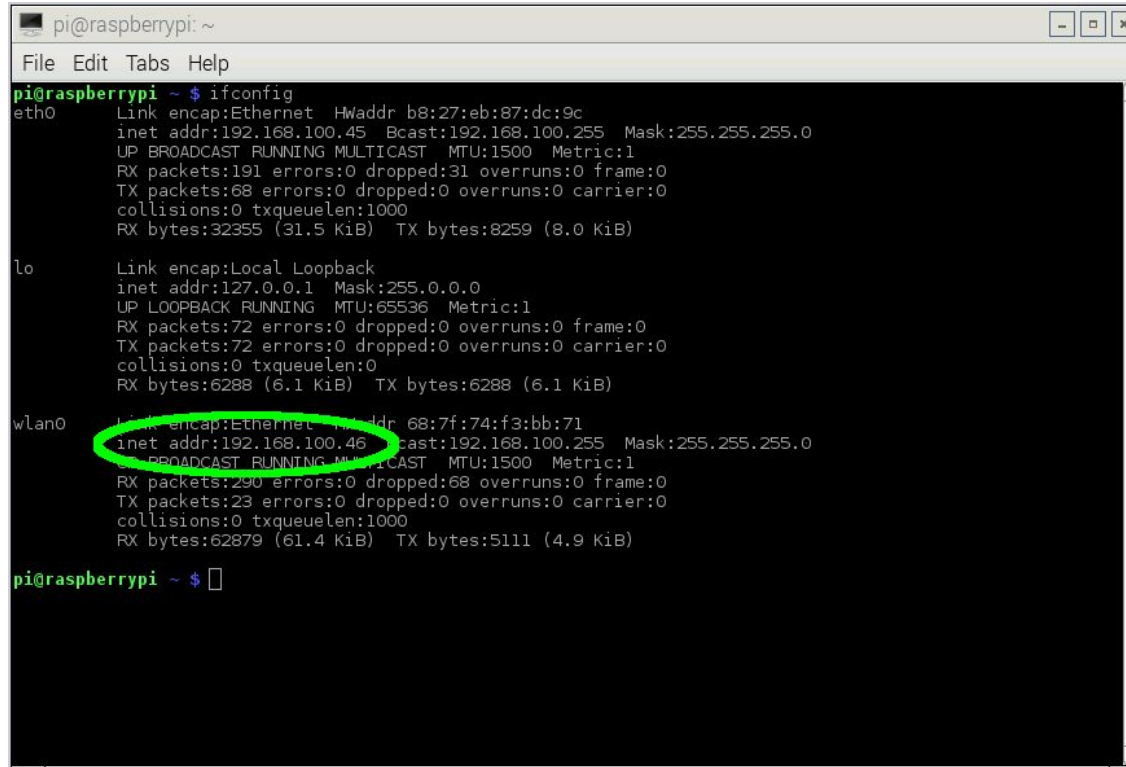
After selecting Ctrl + O, note at the bottom of the screen the prompt to confirm you wish to retain the filename wpa_supplicant.conf. Confirm this by selecting the Enter key, and then subsequently use keys Ctrl + X to exit the nano editor. Next, back at the terminal application prompt, type in:

```
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 below.



```
pi@raspberrypi: ~  
File Edit Tabs Help  
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 of wlan0 we can determine the IP address (please Google this term if you are unfamiliar with IP addressing) 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 using it again over and over!**

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, and you are sure

the WiFi dongle installed (and be sure you install it 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 network.

Each computer or device (i.e. phone, tablet, network connected thermostats, etc.) on a network has a unique IP address, meaning that we can “talk to” that device with other computers once we know the device’s IP address. For example, we can use a laptop or our phone to access a web site that we is hosted on our Raspberry Pi. Now that we know the Raspberry Pi’s IP address, we can “talk to” it from a laptop or phone.

Step 2: What is a “web server” and how to install the web server software

A “web server” is a software application running on a computer that allows that computer to share specially formatted files with other computers using web browser applications. The “web server” can be any computer but it is most typically a computer dedicated specifically to “serving” the specially formatted files. Other computers connecting to the web server are called “client computers”. These client computers must have a web browser application in order to access the web server and they must know the IP address of the web server. URLs, for example, 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 URLs like www.google.com, which is what we will do within this project.

We need to make our Raspberry Pi a “web server” by installing web server software. We will be installing the **LIGHTTPD** web server (called “lighty”) which is a very small and simple web server perfect for this project’s basic needs. See <http://www.lighttpd.net> if you would like to know more about this web server software.

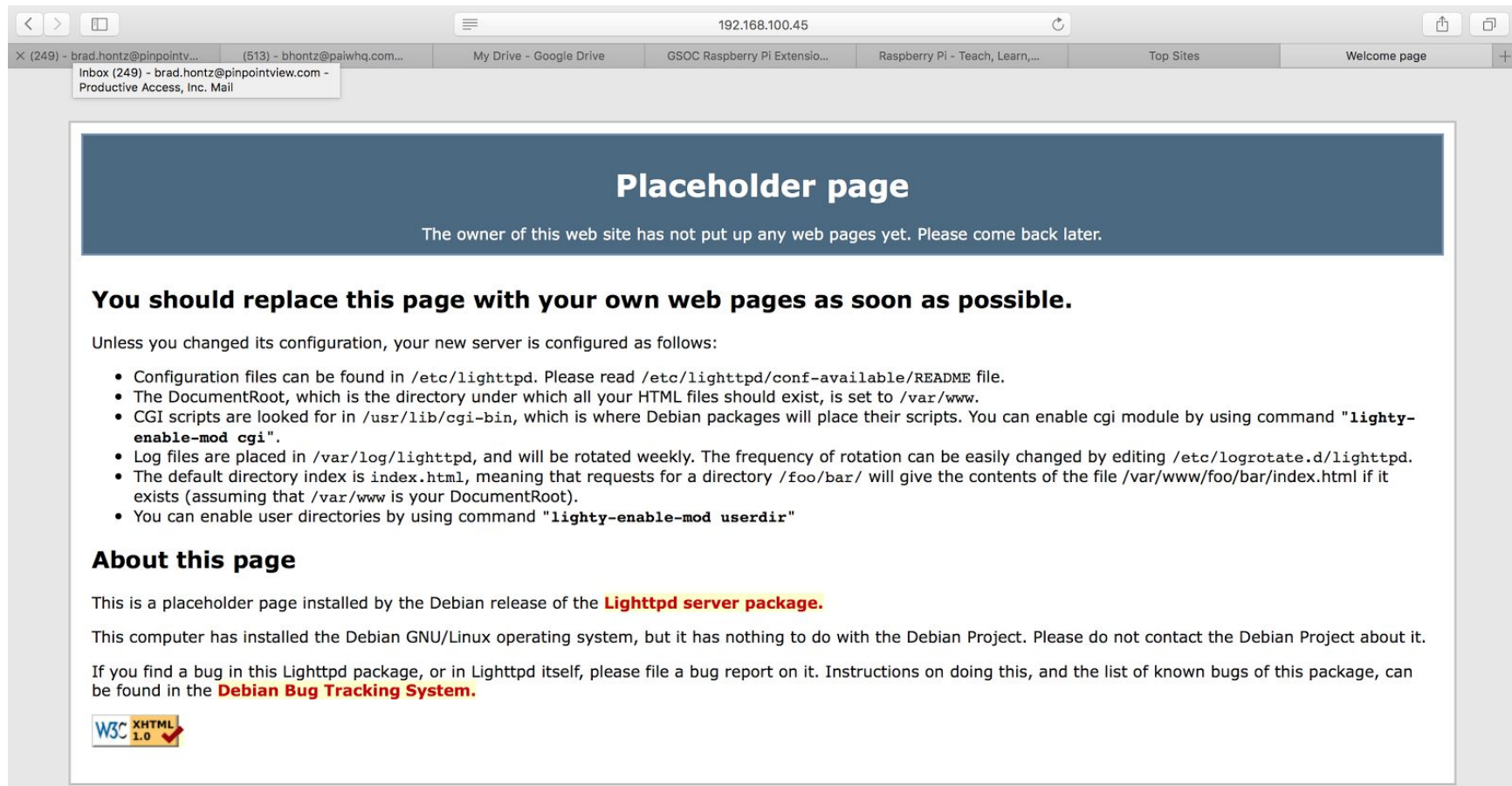
Now that our Raspberry Pi is connected to a network, and presumably the internet, we will be able to install new software onto the Raspberry Pi. From the Raspberry Pi’s terminal application prompt, type in:

```
sudo apt-get install lighttpd
```

If the web server is already installed, you’ll note a message indicating “already the latest version”, otherwise, a number of commands will scroll by on the screen as the lighttpd and any dependent applications are installed on the Raspberry Pi. You may need to confirm installation at a Yes/No prompt to successfully complete the installation. Reboot the Raspberry Pi to start the web server by typing in:

```
sudo reboot
```

Reboot takes a minute or two, and thereafter we can test to see if our Raspberry Pi webserver is “broadcasting” web pages. Connect another laptop, tablet or phone to the same WiFi network as our Raspberry Pi. Open the web browser on this “client computer” and type in the IP address of the Raspberry Pi that we recorded above. 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 a officially a web server!



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

First use the Raspberry Pi's Menu - Shutdown menu options to turn off the Raspberry Pi. After the screen shuts down (i.e. is blank) pull the small micro usb power cable out of the Raspberry Pi. **It is important that we 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 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.

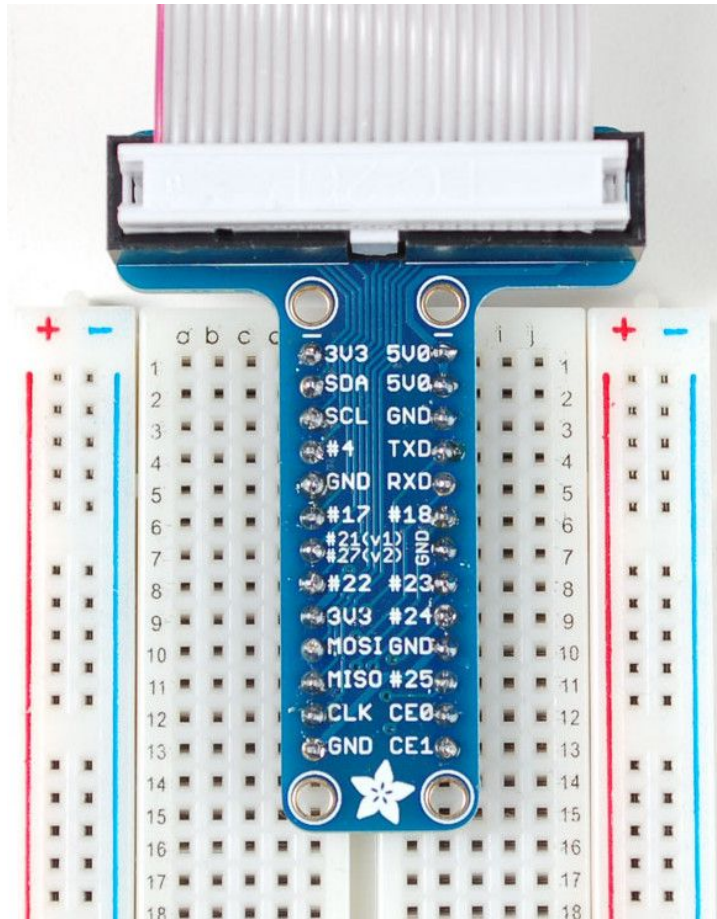
Shown below are a series of pins on the Raspberry Pi, which are called the General Purpose Input and Output bus, or **GPIO** for short. The “cobbler” ribbon cable connection to breadboard (second image shown below) has a schematic depicting the meaning of each of the Raspberry Pi's GPIO pins. Note that 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”). Just like a battery, 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 schematic below. Each of these represents a unique “data port” from which we can use to communicate readings between a sensor and our Raspberry Pi. We will need to use only one of these GPIO data ports, 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 number that we choose 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 below. **Note that the red line on the ribbon cable denotes “pin #1” of the ribbon cable, so be sure that the red line on the ribbon is connected on the “pin #1” side of the Raspberry Pi's GPIO bus as shown in the picture below.**



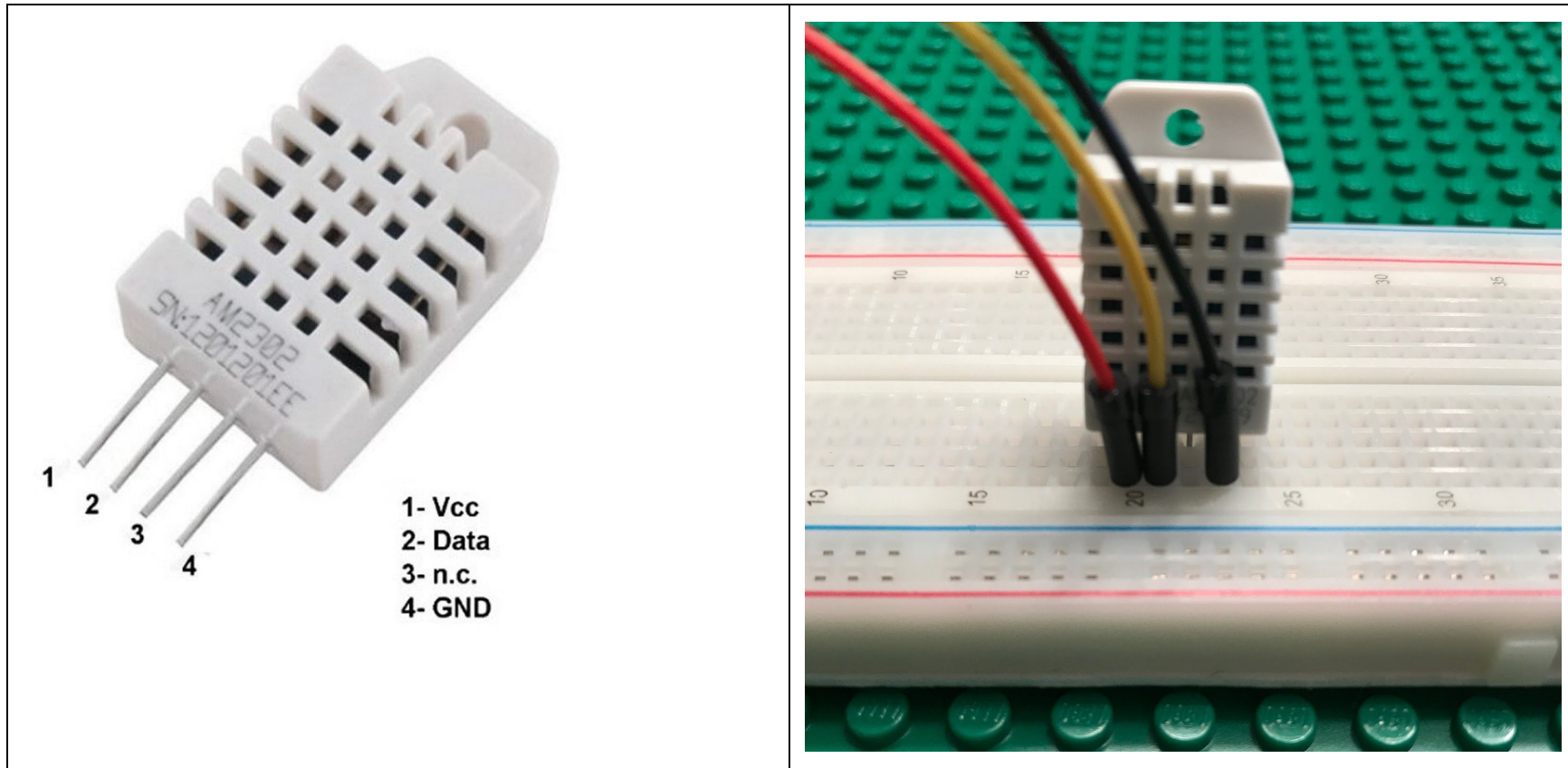
Now connect the ribbon and “cobbler” (if it isn't already) to the breadboard as shown in the picture below:



Looking at the breadboard picture above, holes in the breadboard that are within the same column 'a', 'b', 'c', etc. are INDEPENDENT of one another, or said another way, 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 breadboard between columns 'e' and 'f' means that columns 'a' through 'e' and 'f' through 'j' are independent of one another.

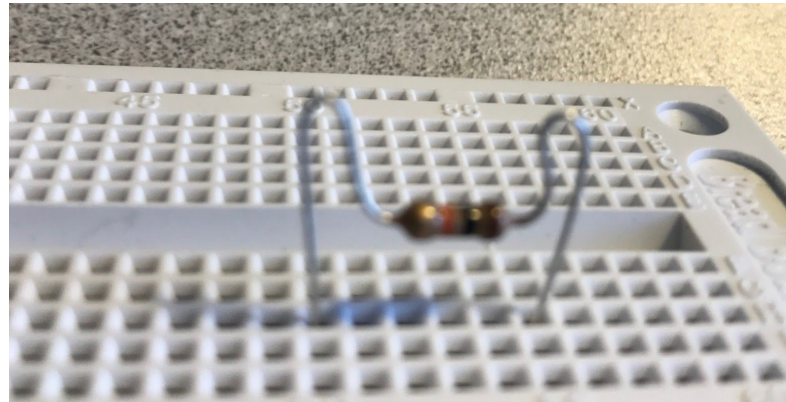
With the understanding of the paragraph above, if we were to place a wire in the column 'a' - row 1 hole of the picture above, then our wire would be connected to the 3v3 pin of the Raspberry Pi's GPIO. Similarly if we were to place a wire in column 'j' - row 3 hole of the picture above, that wire would be connected to the GND (i.e. “ground”) of the Raspberry Pi's GPIO. We can now connect our temperature and humidity sensor (called the DHT22) 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, we 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 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 "not used". 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 kit. Find the 10k ohm resistor (color stripes are brown, black, orange and gold) 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 picture below. 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.



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 connect to pins #1 and #2 of our DHT22 sensor, so we are meeting the resistor requirements by mounting the resistor in this way.

We now have all of our required connections in place! Reconnect the micro USB power cable and restart the Raspberry Pi.

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

This section is not meant to be a python coding lesson, nor do you really need to know anything about python in order to complete this step. Here will simply walk through the components of a python script provided here and teach you how to author the script on the Raspberry Pi. This section assumes that the Raspberry Pi is powered on and is connected to the local network, which in turn has internet access. You can test this by using the Raspberry Pi's own browser to see if you can access a website like "<http://www.google.com>". The internet connectivity is required to install new software on our Raspberry Pi in order to complete this step. The remainder of tasks within this step require using 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 in the web server section of this document and read the content, you'll note that the web server expects pages (i.e. the "document root") to be placed in the folder /var/www. We need to change into this folder before creating our new script file, which we can do with "cd" command we used previously. From the Pi's terminal application, change into the /var/www folder by typing this command at the terminal application prompt:

```
cd /var/www
```

The terminal application's prompt should change, and thereafter at the new prompt type in:

```
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!

Carefully 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 << NEED A HARD LINK HERE IF THIS DOC IS PRINTED >>

```

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 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 code modules that our script uses or “builds upon”. One of these additional modules, Adafruit_DHT, comes with the DHT22 sensor and contains the code to read our sensor. We will need to install this module on the Raspberry Pi, and it is convenient to use the PIP (**P**ython **I**nstall **P**ackages) application for this purpose. At the terminal application prompt, type in:

```
pip help
```

If a message is returned similar to `:pip: command not found` then the PIP application itself first needs to be installed on the Raspberry Pi to support the installation of the Adafruit_DHT module.

Use the two commands below to install the PIP application on the Raspberry Pi if the PIP application was not already installed. Note that this installation will take a little time as the first of the two commands below updates everything that is out of date on the Raspberry Pi prior to installing PIP.

```
sudo apt-get update
sudo apt-get install build-essential python-dev
```

Test your installation by once again entering `pip help` at the terminal application command prompt. If a list of PIP commands options is presented, then you are ready to install the Adafruit_DHT library by typing in the command below at the terminal application prompt:

```
pip install adafruit_python_dht
```

The next set of lines within the script, as shown below, are set equal to the variable “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>
"""
```

The lines above represent the HTML syntax that we will be sending to client computers connecting to our Raspberry Pi web server. Here we have three lines or “paragraphs” (<p></p>) with each line containing Time, Temp and Humidity respectively. Within the HTML <style></style> block of this code block, we define a font-size of 96 points, so these lines will 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 remember that we connected our data wire to GPIO #17! (i.e. “pin” 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` rows (in that order) with the local time formatted to H:M:S (hours:minutes:seconds), and 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 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, and separately a `response_headers` variable represents a required HTML header defining the type and size of the HTML page (i.e. our variable `response_body`) that we've prepared to send to client computers. We also pass a `Refresh` header parameter within our `response_header` variable 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” our variable `response_body`, which is our completed web page as represented by variable `response_body`, which we defined above.

```
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 argument (i.e. a value we specify after our script name) which in this case is the IP address of our Raspberry Pi web server. We wish to have the IP address passed as an parameter (sometimes called an “argument”) to our script by the user running the script so that we can connect the Raspberry Pi to different networks. Remember that on each network our Raspberry Pi will have a different IP address, so by passing the Raspberry Pi’s IP address as an parameter, we avoid having to edit our script each time we run our Raspberry Pi on a different network.

```
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)
```

If we try to run this script and we forget to specify the IP address of the Raspberry Pi after our script’s name, we’ll 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. I noted my_weather_station.py consistently as an example script file name, but you may have selected a different name for your script.

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 if need be, but first let’s get our script running!

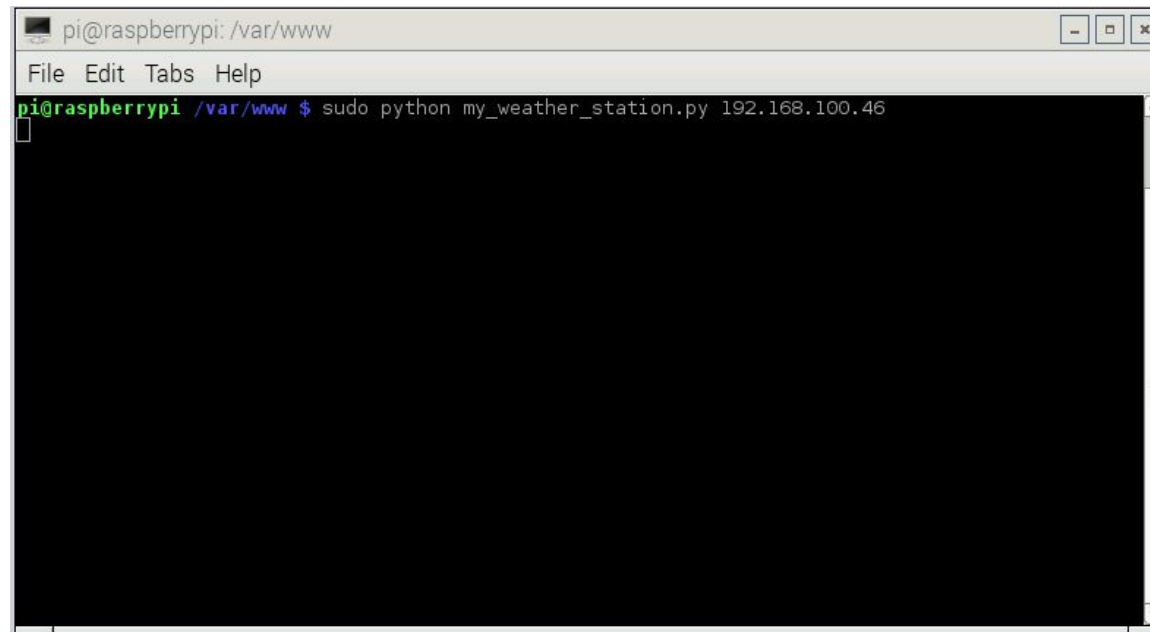
Running our script on the Raspberry Pi

After exiting the nano text editor application from within the Raspberry Pi’s terminal application, we can now run our script and start our web server hosting. We should still be within the /var/www folder of our Raspberry Pi; this is the folder that contains our webserver content. We then type in at the terminal application prompt:

```
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 script name `my_weather_station.py` and the Raspberry Pi IP address `192.168.100.46` consistent with this document’s example, but you may have elected to use a different script name. Just as importantly, you will almost certainly have a different IP address for your Raspberry Pi. Be sure to use the correct script name and your actual Raspberry Pi’s IP address!

Here’s what the terminal application window screen looks like after having entered the command above and hitting the Enter key:

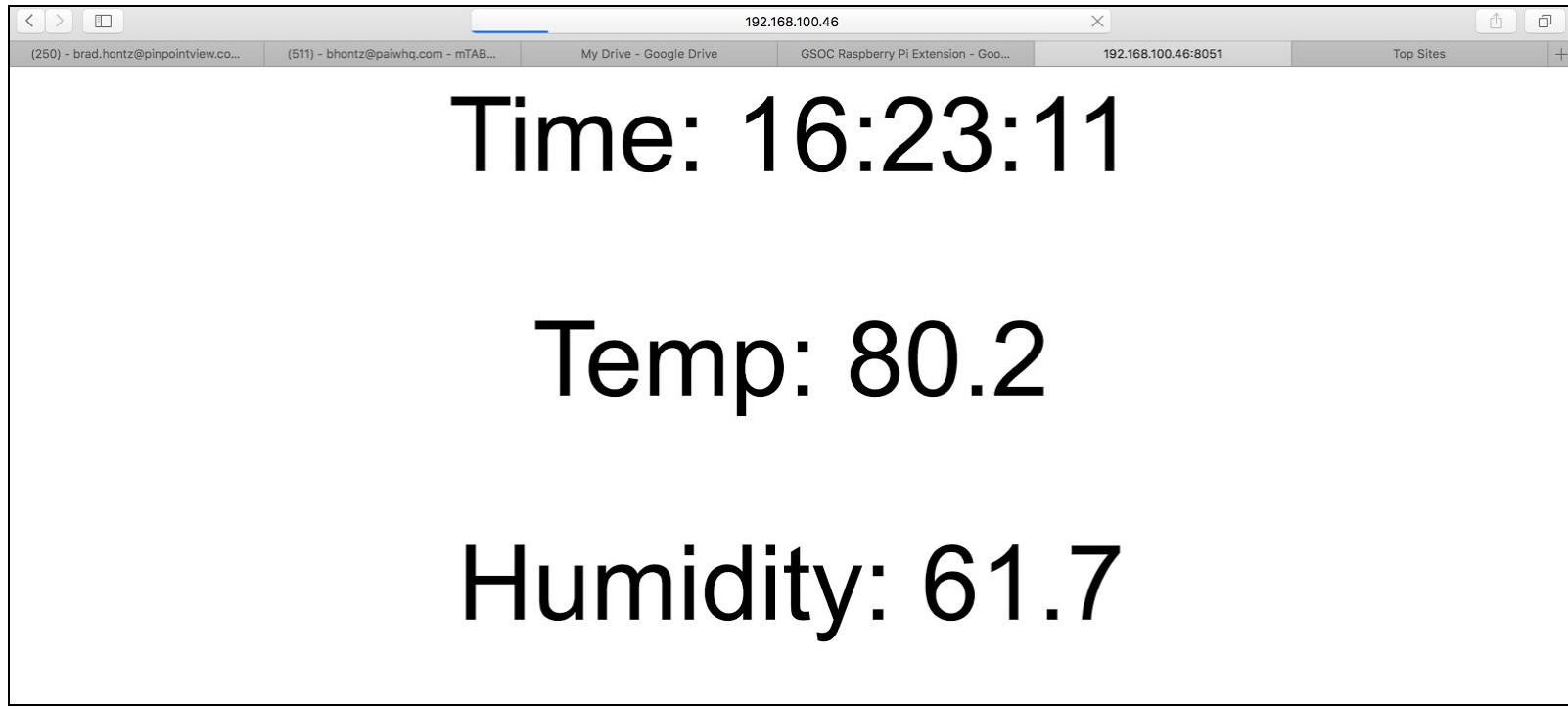


```
pi@raspberrypi: /var/www
File Edit Tabs Help
pi@raspberrypi /var/www $ sudo python my_weather_station.py 192.168.100.46

```

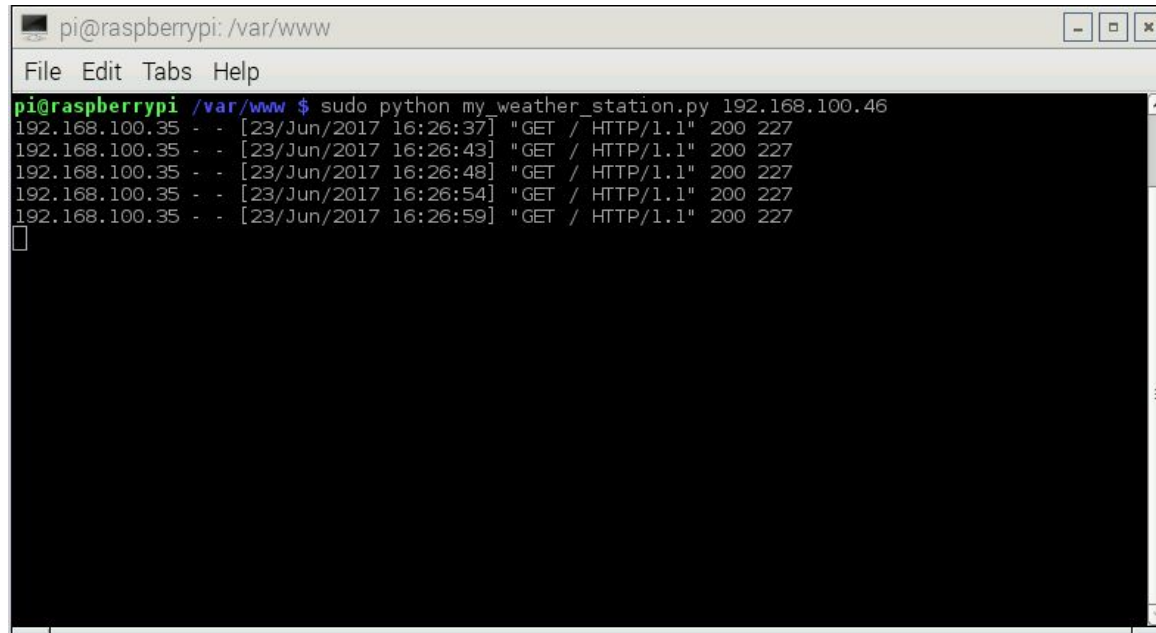
It appears nothing is happening! Let’s connect a client computer to our webserver and see if our script is really working! First make sure your client computer (i.e. your laptop, tablet or phone) is on the same Wifi network as your Raspberry Pi. Next, open up the web browser on your client computer and 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` after the Raspberry Pi’s IP address!** (remember the reference to that “port” within our python script discussion?)

Once again, in the screenshot below I’m connecting to the Raspberry Pi with the example IP address used in this document, yours will be different! Here’s what my Mac laptop’s web browser looks like after connecting to the Raspberry Pi’s 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. Carefully blow onto the DHT22 sensor (or even more carefully, use a small hairdryer) and note how the temperature changes when the web page refreshes.

After a client computer makes a connection, your Raspberry Pi's terminal window will start to display "log messages" as per the example shown below:



```
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;ds have each of your girls connect their phones!

You can use the Raspberry Pi terminal window's commands Ctrl+C to stop the script running if you want to to have access to the terminal window, or you can just shut the Raspberry Pi down when you're done.

Thanks for participating in this workshop, I hope that you enjoyed learning about other useful ways in which to use the Raspberry Pi!