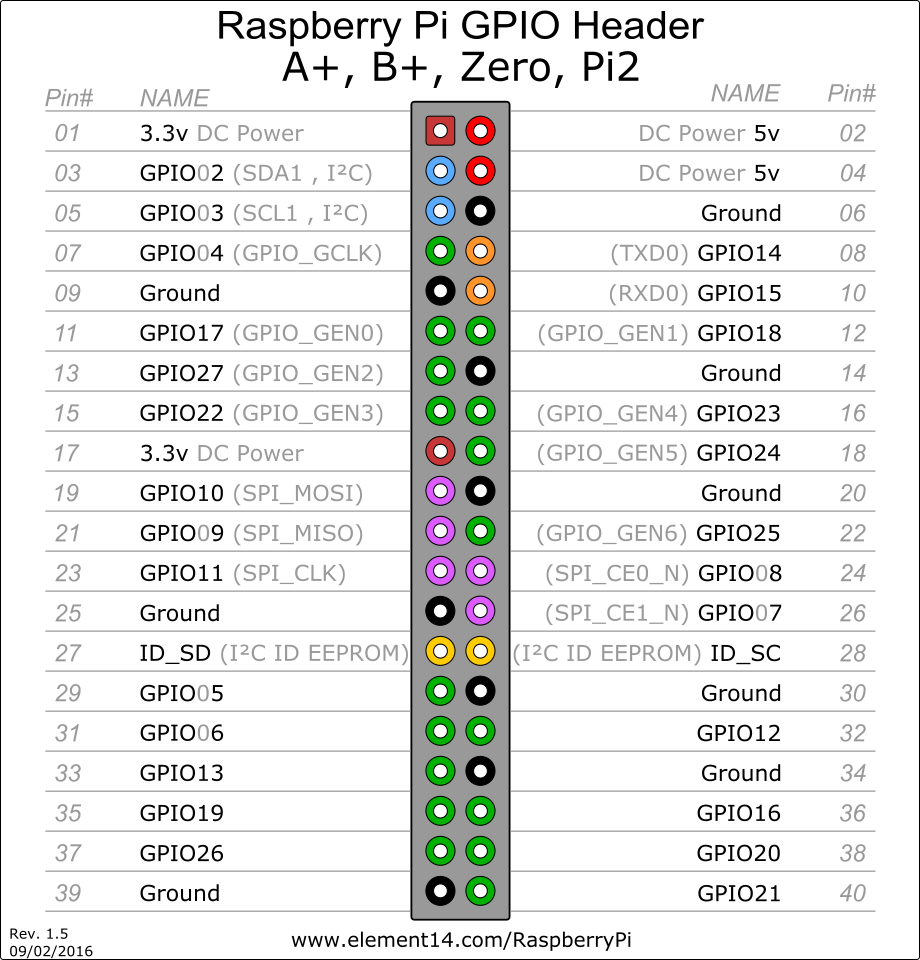
1. Course Goal
   1. Exposure to a Linux based OS
   2. Sensor interaction
   3. Data storage options
   4. API integration/interaction
   5. Python scripting
   6. Create IoT device with consumable data points
2. Course Material/Items
   1. PI Zero W
   2. Jessie Lite (Linux OS)
   3. DHT11 Sensor
   4. 8GB MicroSD card
   5. Female to Male wire harness
   6. Solderless GPIO headers
   7. Wall charger
   8. Micro USB cord
   9. PI Zero case
3. Connect to PI Console
   1. Install Bonjour and connect the pi to your pc via the USB port
      1. Link to install [Bonjour](https://support.apple.com/kb/dl999?locale=en_US)
   2. SSH steps to connect over WiFi(<IP Address>:22) pi:raspberry
   3. SSH steps to connect over Ethernet Emulator (raspberrypi.local:22) pi:raspberry
   4. Once connected to device, rename device using ‘sudo raspi-config’ > Hostname
   5. Update the device password for the pi user using ‘sudo raspi-config’
   6. Update the Ubuntu packages ‘apt-get update’
4. Required Software
   1. Install Sqlite3 apt-get install sqlite3
   2. Install Python Legacy apt-get install python (Will already be installed)

The following steps will be used to install the required packages and establish the project folder:

1. Install python-pip  
   apt-get install python-pip
2. Install python gpio package  
   easy\_install gpio

easy\_install requests

1. Enable GPIO  
   apt-get install rpi.gpio  
   raspi-config -> Interfacing Options -> Enable I2C  
   raspi-config -> Interfacing Options -> Enable SPI
2. Install git  
   apt-get install git
3. Install python dev tools and ssh  
   sudo apt-get install build-essential python-dev python-openssl
4. Make new project directory in root  
   cd /root  
   mkdir TempApp
5. Change to TempApp directory  
   cd TempApp
6. Get git Adafruit Repo  
   git clone <https://github.com/adafruit/Adafruit_Python_DHT.git>
7. Change to Adafruit folder  
   cd Adafruit\_Python\_DHT
8. Run the setup to install the module  
   python setup.py install
9. Read from the sensor
   1. Reference the GPIO pin layout:  
      
   2. Highlight the difference between the 3.3v and 5v (Arduino/PI)
   3. GPIO pins that will be used for the task 3.3v, GND, and pin 4 on right
   4. Create a new .py file to write the read script in /root/TempApp/
   5. In the Adafruit folder open the example Adafruit\_Python\_DHT/examples/AdafruitDHT.py. This file provides an outline on how to interact with the DHT11 sensor.
   6. **sensorRead.py** is the completed script for this step
10. Create the local storage database
    1. Within the project folder, create a new sqlite3 database
    2. sqlite3 temperature.db
    3. Create the temperature table to store the data from the sensor:In  
       CREATE TABLE temperature(

device\_name TEXT,

temp\_in\_f TEXT,

temp\_in\_c TEXT,

percent\_humidity TEXT,

processed BOOLEAN,

processed\_at DATE,

created\_at DATE);

6.4 Insert a test record into the newly created table.

6.5 Select the test record, then remove the test record.

6.6 To see table structure use .schema to view the table.

6.7 To exit type .exit

1. Store Reading in Sqlite3 temperature table - **writeRecord.py**
   1. Return to the .py script that we created in step 5
   2. Alter the script to insert the value into the Sqlite3 temperature table
      1. The requirement for this step is to populate all table values  
         Device Name is the Host Name of the device  
         None is the equivalent of NULL  
         Add the insert statement to the ‘if’ portion of the script  
         Reference the writeRecord.py file for the desired outcome
   3. Upon a successful run, verify the data in the sqlite3 temperature table
      1. sqlite3 temperature.db
2. Restful API Interaction – **apiSubmit.py**
   1. In the project folder, create a new .py file used to submit the API post request
   2. Read the most recently created record in the temperature database
      1. Reference apiSubmit.py line 20
      2. Select the first row returned from the query
   3. Apply the values as variables to be used in the API POST
      1. The URL that will be used is:  
          <http://ec2-13-58-115-247.us-east-2.compute.amazonaws.com:3000/temperature>
      2. Reference the **apiSubmit.py** file for the payload format and how to execute the post request using the requests package.
   4. After the API call has been made and
   5. Check the MongoDB for the inserted record using the ‘device’=’device\_name’ and ‘sourcecreatets’=’created\_at’
   6. MongoDB connection information
      1. http://ec2-13-58-115-247.us-east-2.compute.amazonaws.com:27017 dscapiuser@bdtemp | Datasource2017!
3. Catch up script – **catchupScript.py**
   1. Create a script that will process records that were not successfully submitted to the API
   2. Copy the apiSubmit.py script and rename it (catcupScrip.py)
   3. We will add a while statement to create a loop that will identify records that need to be resent
      1. We will continue to reassign the one\_row variable until it None(NULL)
   4. To test the script execute the writeRecord.py script multiple times then execute catchupScript.py
4. Scheduling Prep – **tempApp.py**
   1. To allow the scripts to run in order without having multiple cronjobs, we will trigger them from the writeRecord.py script
   2. Import the package subprocess
   3. Add the subprocess line to the if statement to be kicked off after the record is inserted into the Sqlite temperature table.
   4. Reference the tempApp.py file for formatting and placement
5. Schedule the tempApp
   1. Open crontab using crontab -e
      1. Option 2 for Nano editor
   2. Add the following line: \* \* \* \* \* python /root/TempApp/tempApp.py &
   3. Job will execute the scripts every minute while the system is on
6. Conclusion
   1. We have successfully created a device that will submit temperature/humidity data every minute to a cloud data platform. This data can be accessed at any time by using the url from 8.3.1. This URL can be imported into BI tools like Microstrategy Desktop and Power BI as a data source.