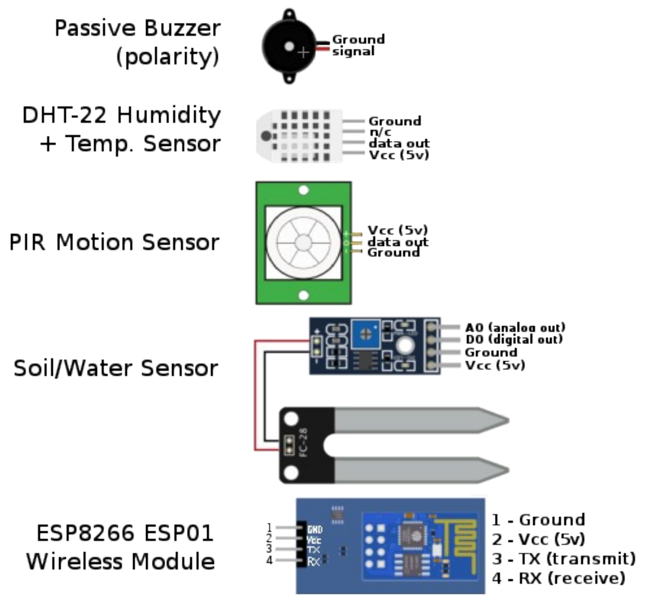
<u>Let's Code Blacksburg! “IoT Home/Datacenter Environment Sensor Monitor”</u>

This 3hr, intermediate level, hands on workshop steps you through building and programming your very own online IoT Environment Monitoring node. It requires the Arduino IDE software being installed, along with the following hardware parts:

- 1 – Arduino board (with extra sensor pin headers or sensor shield & USB cable)
- 1 – Dedicated 9v DC wall PSU
- 1 – Passive buzzer
- 1 – Temp+Humidity DHT-22 sensor
- 1 – PIR Motion Sensor HC-SR501
- 1 – Soil/Floor Water HL-69 sensor
- 1 – WiFi module ESP8266 (ESP01) + 3.3v→5v adapter module
- 11 – F/F Dupont connection wires
- 6 – M/F Dupont connection wires
- (opt) – project case, hot glue, etc

Sensor Pinouts



<u>1) Use First Cookbook Recipe - Blink</u>

Before you build anything or do any real programming, open the arduino programming interface and open **Files / Examples / 01. Basics / Blink**, click the compile/upload icon , and verify that blink is actually working (blinking your pin 13 LED indicator). If that works, then try changing the blink speed of the LED. This will verify that your computer and the installed arduino IDE software (from www.arduino.cc) is properly configured to talk to your arduino. Get TA sign-off before proceeding:

TIP: Once you get something working, ALWAYS:

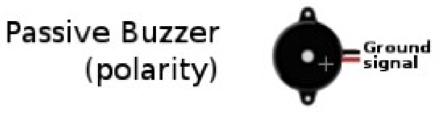
- *SAVE YOUR CODE after each step*
- *<u>give it a meaningful name</u> (like “IoT-Sensor_blink-v1”, in this case)*
- *Before starting new code, open a new program with **Files / New***

Before continuing to the next section, please get signed off by a roaming TA, to help us with troubelshooting if you have problems later.

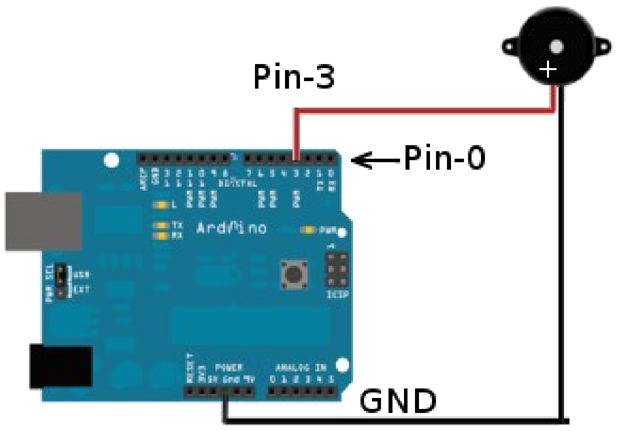
TA SIGN-OFF: ____________

2) Use Cookbook Recipe – *OUTPUT Buzzer Alarms* (pg 30)

Open the Arudino cookbook and examine the recipe for the passive buzzer. Each recipe in the cookbook will guide you through the recipe with a What (theory), How (to hook it up and code it), and Failure sections (what to do if doesn't work).



The wiring diagrams in the cookbook often use a breadboard to interconnect components. If you have an arduino with additional data pin, Vcc/5v, and Ground headers, then you can bypass using the breadboard and wire up your sensors directly to the arduino board as pictured in the following wiring diagrams. Regardless if you use a breadboard or the included diagrams, the actual pin numbers you use should be the same here as in the cookbook.



Once connecting the buzzer as shown (noting the polarity), then look through the code in the buzzer recipe and get a feel for how it's working. Feel free to customize or write your own sound functions. Just be aware that the four buzzer functions `buzzer("chirp")`, `buzzer("success")`, `buzzer("fail")`, and `buzzer("alarm")` will be useful later as auditory troubleshooting feedback when working with the ESP01 wireless module while connecting to WiFi.

> *TIP: We're going to use this code later and combine it with our other sensor code, so call this program soemthing like "IoT-sensor_buzzer" and create a new program with File / New in the arduino interface.*

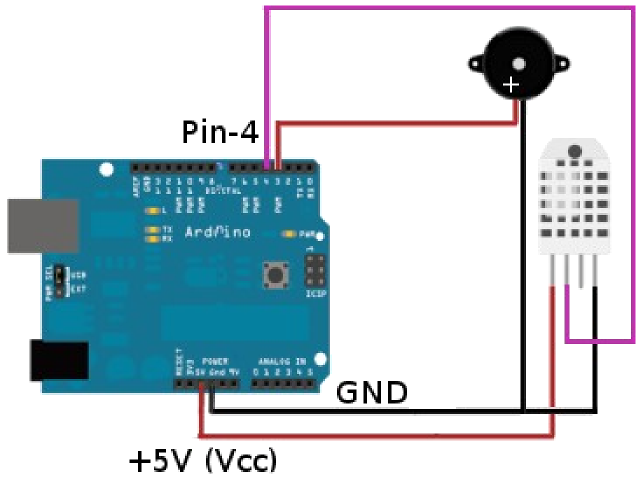
Before continuing to the next section, please get signed off by a roaming TA, to help us with troubelshooting if you have problems later.

TA SIGN-OFF: ____________

3) Use Cookbook Recipe – *INPUT: DHT-22 Humidity/Temperature Sensor* (pg 39)

After saving your buzzer code, in the Arduino interface be suer you have done a File / New to create a new program for creating the DHT-22 Humidity/Temperature code.

The DHT-22 is a versitle, low cost humidity and temperature sensor. However there are several versions of the unit's package on the market. The one shown here has four pins (three plus one unused), while the DHT22 in the cookbook has only three pins.. though they are functionally the same. Use the diagram for the module that most closely matches yours. However always check to ensure that Ground and Vcc(5v) are in the same location as the module in your hand.

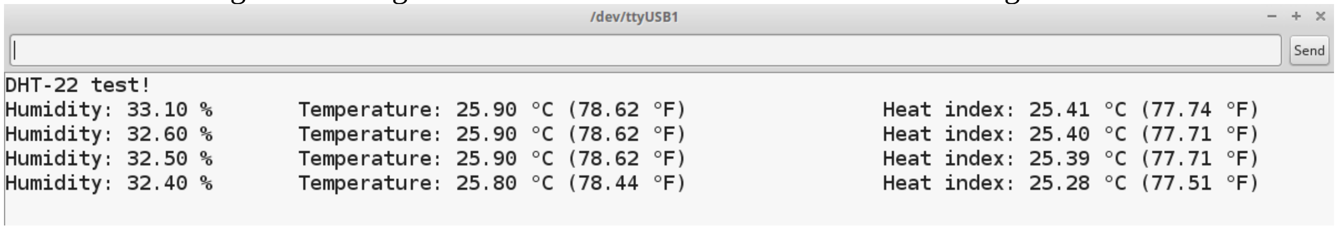


As you connect the DHT-22 humidity sensor (as per the diagram here or in the cookbook), be sure to route that units ground and Vcc(5v) pins to separate connection points on the Arduino or breadboard. When diagrams show wires combining, we do not expect you to splice or piggy-back wires together. Find an appropriate, functionally duplicate pin (male or femal) on the Arduino and simply connect it to that point.

WARNING: Double and tripple check your Ground and Vcc(5v) wiring. Getting these wrong or shorting them out can "let the smoke out" of your circuit, arduino, or even latop.. if not careful.

TIP: Always a) disconnect the arduino before hooking up more sensors or circuits, and b) double check your wiring before plugging in or powering back up. If you want, call the instructor or TA-helper over to get a second set of eyes on it. The instructor may not have spare parts to help you if you blow a component.

Once you get it all hooked up and coded, copiled and uploaded, click on the Arduino serial monitor button in the upper right corner of the Arduino IDE to see the Humidity and temperature data your sensor is recording and sending. The serial monitor data should look something like this:

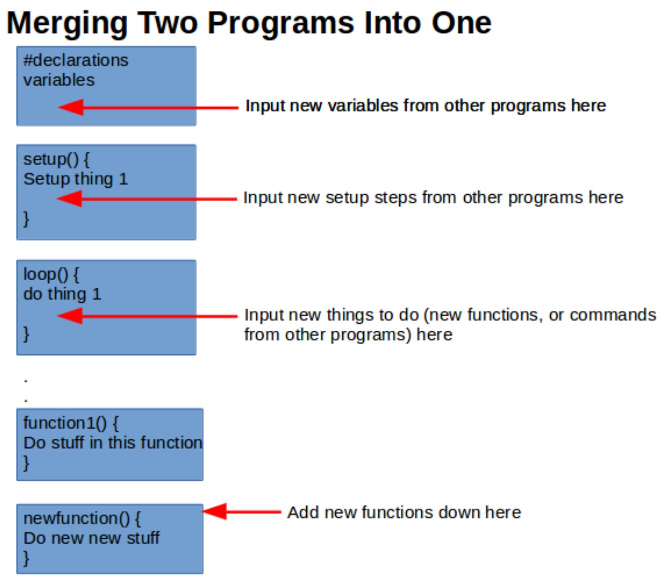


```
DHT-22 test!
Humidity: 33.10 %       Temperature: 25.90 °C (78.62 °F)        Heat index: 25.41 °C (77.74 °F)
Humidity: 32.60 %       Temperature: 25.90 °C (78.62 °F)        Heat index: 25.40 °C (77.71 °F)
Humidity: 32.50 %       Temperature: 25.90 °C (78.62 °F)        Heat index: 25.39 °C (77.71 °F)
Humidity: 32.40 %       Temperature: 25.80 °C (78.44 °F)        Heat index: 25.28 °C (77.51 °F)
```

NOTE: These low cost sensors are not exaclty known for their accuracy, but if you take a few readings, you can easily "calibrate" your sensor in software mathematically, if the difference is linear.

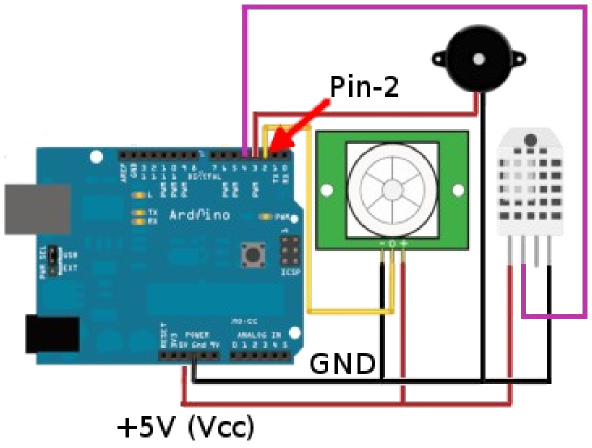
TA SIGN-OFF: ____________

GOAL: Merge buzzer and DHT22 code
If you have this DHT22code working, then save it as "IoT-Sensors_DHT22". Once saved, open File / Save As and give it the new file name of "IoT-Sensors_buzzer_DHT22" as we will attempt to combine your buzzer code in with this working code to get us closer to the end goal. You'll want to add chunks of the four sections of your buzzer code (variables, setup(), loop() and functions), to each of those sections in your new buzzer_DHT22 program. See visually how to merge two program's different variables, setup(), and loop() code blocks as seen here.


Merging Two Programs Into One
#declarations
variables
Input new variables from other programs here
setup() {
Setup thing 1
}
Input new setup steps from other programs here
loop() {
do thing 1
}
Input new things to do (new functions, or commands from other programs) here
function1() {
Do stuff in this function
}
newfunction() {
Do new new stuff
}
Add new functions down here

4) Optionally Use Cookbook Recipe – “*INPUT: People Motion Sensor*” (pg41)

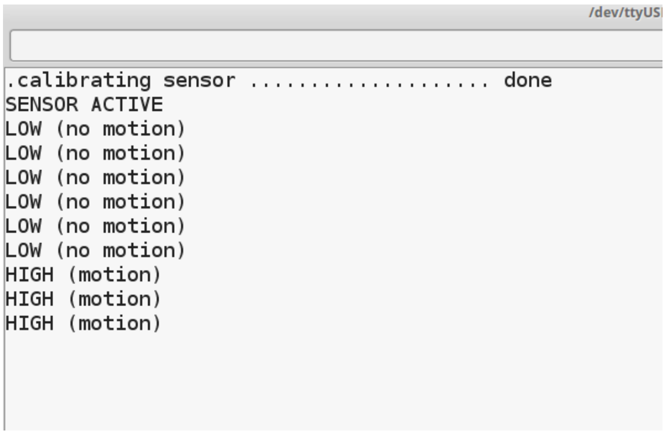
While motion sensing may not be needed for your environemnt monitoring node, it is cheap, simple and can easily be added to this project if detecting or logging this type of data in important to you. If you are behind in this lab, getting the last module in steps 5 & 6 which are probably more critical from an IoT project functionality perspective.



Refer to the Cookbook Recipe for this device and verify that if your sensor has a jumper on theback of its board, that it is configured correctly. Also be sure to get the Vcc(5v) and GND polarity on this module correct as you can blow these if you get the power hooked up wrong. The center data-pin (output) will be going to the Arduino digital pin-2, configured as an input.

> *NOTE: While this PIR motion sensor is super sensitive and really cool to use, it also requires a 20-30 second “settling time” for the PIR sensor (which senses changes in the room's visible heat profile) to calibrate to a static, no-motion, room scene. As such, it is very difficult to get this sensor working in a classroom environment where you do not control the motion and heat signatures around you. If you hope to get it working, the sensor has a 110° field of view and can see motion up to 20ft away, so to impliment this sensor in a busy classroom you will need to cover the sensor with a notebook or sheet of paper each time you make a software adjutsment to allow it to “calibrate”. If you have problems getting it to not constantly tirgger in class, it might be best to do later in a more controlled environment.*

Once you get your PIR motion sensor working, this is something like what you should see in your serial console window:



```
.calibrating sensor .................... done
SENSOR ACTIVE
LOW (no motion)
LOW (no motion)
LOW (no motion)
LOW (no motion)
LOW (no motion)
LOW (no motion)
HIGH (motion)
HIGH (motion)
HIGH (motion)
```

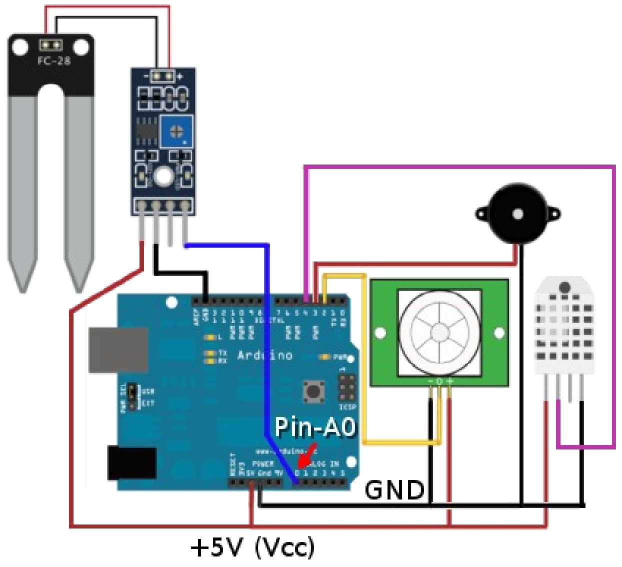
As you move a hand into and out of view, it should register motion (HIGH) and no motion (LOW).

TA SIGN-OFF: ____________

Now try merge your other sensor's code in with your PIR motion sensor code, get that working, and call it `IoT_Sensors_buzzer_DHT22_PIR` . If it's getting too big to handle, we've provided a merged version here: https://github.com/LetsCodeBlacksburg/arduino_IoT_sensors/ called `IoT_Sensors_4b_buzzer_DHT22_PIR` that should work nicely.

5) **<u>Hook up the Floor/Soil Water Sensor as Shown Below (no recipe)</u>**

While there is no recipe for the floor/soil water sensor.. it is so simple none is really needed. Just refer to the hookup diagram the the right. First use two female/female connector wires to connect your U-shaped water sensor to its sensor board, then connect the Vcc(5v) and GND pins to usable points on the Arduino, and then the sensor's AO or "analog out" pin to the A0 (analog 0) input on the arduino as seen here.



Coding this is as simple as using the analogRead(A0) commad, like this:

```
Serial.println(analogRead(A0));
```

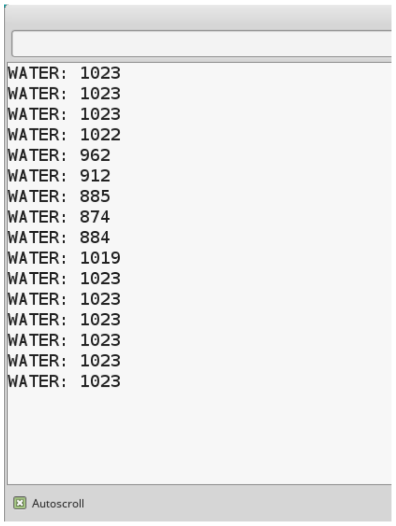
So your entire water sensor code for this build might look like this:

```
//Water Sensor variables:
const int waterPin = A0;
const int waterVal = 0;

void setup() {
  Serial.begin(9600);
}

void loop(){
  delay(1000);
  waterVal = analogRead(waterPin);
  Serial.print("WATER: ");Serial.println(waterVal);
}
```

Since the `waterVal` measurement is based on the Arduino's 10bit A/D converter's measurement, a 10bit full scale reading is 2^{10}, or 1023 (1023=dry or no water). However, just licking your fingers and placing them across the sensor leads (perfectly safe) will yield a lower number in the 800-900s as seen in the serial console's output to the right.



```
WATER: 1023
WATER: 1023
WATER: 1023
WATER: 1022
WATER: 962
WATER: 912
WATER: 885
WATER: 874
WATER: 884
WATER: 1019
WATER: 1023
WATER: 1023
WATER: 1023
WATER: 1023
WATER: 1023
WATER: 1023
```

You will want to test this sensor in your environemnt, on your floor, or inserted into your carpet.. taking multiple wet and dry readings until you have a reliable threshold value. In the example here, I might pick a threshold value of < 1000 = wet. Below are some average test values that were useful in *my* configuration, but you will need to validate your own:

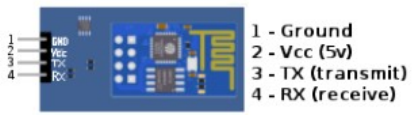
Dry	**1023**
Dry carpet	**1020**
Wet fingers	**986**
Distilled h20	**1002**
Distilled h20 + salt	**498**
Tap water	**886**
Licked Fingers	**977**

NOTE: If your desire is to measure for both tap/pipe water floods as well as air conditioner condensation overflow floods, be sure to test using actual tap and air conditioner water on your floor. This is important as AC condensation is distilled and alone does not conduct water well. Assuming the same threshold value as tap water is a faulty assumption, as it will yeild completely different threshold numbers in your environemnt (e.g. a water & AC utility closet with a tile floor) than say tap/pipe water, which as dissolved conductive minerals in it. If you have problems reliably measuring AC condensation water because of its naturally high electrical resistance, as seen in our test values above, you might want to spread a sprinkling of table salt around your sensor's measurement sensor as to get better AC condensation flood values. In all things, testing and validation is critical.

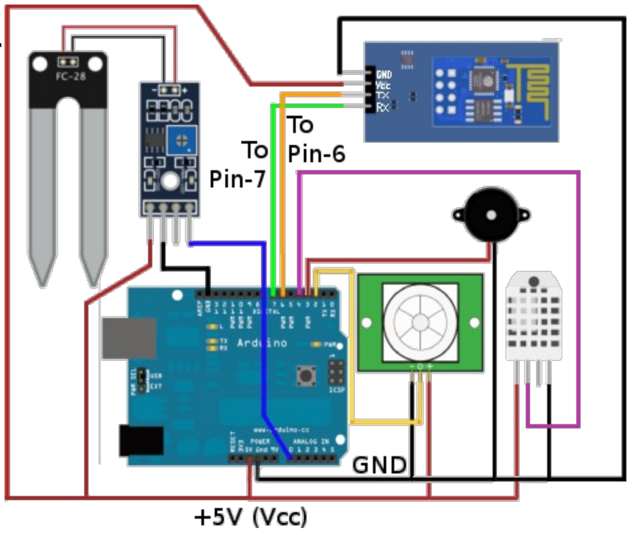
Merge All Sensor Code or Download Example Code:
If you have had a hard time getting all of the code for each sensor merged, you can download the final **`IoT-Sensors_5b_buzzer_DHT22_PIR_water`** code, which includes all prior sensors integrated into oneprogram. Download the Step-5b code from out LCBB git repository here:
https://github.com/LetsCodeBlacksburg/arduino_IoT_sensors/ **TA SIGN-OFF:** ____________

Now that you have most all of the sensors working, let's impliment the ESP01 WiFi module to get our creation onto the network and start logging or sending alerts.

6) **Reference Cookbook Recipe – *"COMM: Wireless Web & Email Communications" (pg59)***
Bringing it all together, we're now going to wire in the ESP8266 ESP01 WiFi module. The module is a powerful microcontroller unto itself, but also functions as a type of dedicated "WiFi modem" that the Arduino can talk to over serial send and receive lines and let it deal with the comms stack.



Here's the complete wiring diagram of our IoT sensor device.



*IMPORTANT: If you merely follow the recipe in the cookbook, and are using a classic Arduino UNO/R3, then your Arduino can not reliably communicate over the Software Serial driver at the ESP01's default 115.2 kbaud data rate. To use the ESP01 module on the UNO reliably, you must re-initialze the ESP01's UART to force it to talk down to the 9600 baud rate of your Arduino UNO. To do this, do not use the sample code listing in the cookbook, but instead download the **`IoT-Sensors_6_ESP01`** sample code from here:*
https://github.com/LetsCodeBlacksburg/arduino_IoT_sensors
This will save you a lot of time and frustration.

NOTE: To join a WiFi network, you will need to plug in the wireless network SSID and password for your space. The WiFiEsp library we're using (by bportaluri) must be installed from the Arduino's built in library search/add tool (see cookbook). However the this library can only handle running on WPA and WPA2 (w/shared passwrod), not WPA Enterprise (username + different passwords). If you have problems getting onto your workshop's WiFi, let your Instructor or TA know.

Now either merge in your previous sensor code (or the **`IoT_Sensors_5b`** sensor code) into the working WiFi code. If you're out of time, then the feature complete sample code is listed in our workshop github repos as **`IoT_Sensors_6b_compete`** here:
https://github.com/LetsCodeBlacksburg/arduino_IoT_sensors