Raspberry Pi and Sensor Data: For Beginners
A Complete Tutorial for Collecting Temperature and Humidity Data
Using a DHT11 Sensor and a Raspberry Pi

Central Michigan University
Summer 2017
**Section 1: Hooking up the Raspberry Pi**
For this information, watch the Getting Started with PiCar video. This will show you exactly how to get your Raspberry Pi powered up and running.

*The Basics of Raspberry Pi*
Your Pi should be running now, but to get started we want to give you a bit of an overview of what a Raspberry Pi is and how exactly we can use it.

1. **A Microcontroller**
Raspberry Pis are a type of microcontroller. You can think of them like tiny computers. Rather than having a desktop computer hooked up to a monitor like the old school picture shown below, we can hook a monitor into the tiny Raspberry Pi to use it like a computer.

2. **Why Raspberry Pi?**
Raspberry Pis have something called “pins” (boxed in red below) that allow us to use computer codes (which you’ll be using soon enough) to control sensors and collect sensor data. In this module, we will be collecting temperature and humidity data using the Raspberry Pi, for example.
3. Um…Pins?
The Raspberry Pi pins are called GPIO, which stands for General Purpose Input Output. A picture is shown below of a Raspberry Pi “pinout”. A pinout is an explanation of the pins and what they do. Each pin has its own job. Some of them are GPIO pins that we can control. Other pins already have assigned roles (like ground, power, etc.) We can’t change those.

You can think of this like an office building. The power and ground pins are the directors; we can’t do anything to change them. The GPIO pins are the interns and grad students…we can make them work for us (as long as we give them specific instructions about what to do and help them out along the way). 😊

This is a picture of a Raspberry Pi pinout.

<table>
<thead>
<tr>
<th>Alternate Function</th>
<th>Alternate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3V PWR</td>
<td>5V PWR</td>
</tr>
<tr>
<td>GPIO 2</td>
<td>5V PWR</td>
</tr>
<tr>
<td>GPIO 3</td>
<td>GND</td>
</tr>
<tr>
<td>GPIO 4</td>
<td>UART0 TX</td>
</tr>
<tr>
<td>GND</td>
<td>UART0 RX</td>
</tr>
<tr>
<td>GPIO 17</td>
<td>GPIO 18</td>
</tr>
<tr>
<td>GPIO 27</td>
<td>GND</td>
</tr>
<tr>
<td>GPIO 22</td>
<td>GPIO 23</td>
</tr>
<tr>
<td>3.3V PWR</td>
<td>GPIO 24</td>
</tr>
<tr>
<td>GPIO 10</td>
<td>GND</td>
</tr>
<tr>
<td>GPIO 9</td>
<td>GPIO 25</td>
</tr>
<tr>
<td>GPIO 11</td>
<td>GPIO 8</td>
</tr>
<tr>
<td>GND</td>
<td>SPI0 CS0</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>GPIO 5</td>
<td>SPI0 CS1</td>
</tr>
<tr>
<td>GPIO 6</td>
<td>GND</td>
</tr>
<tr>
<td>GPIO 13</td>
<td>GND</td>
</tr>
<tr>
<td>GPIO 19</td>
<td>GPIO 16</td>
</tr>
<tr>
<td>GPIO 26</td>
<td>SPI1 CS0</td>
</tr>
<tr>
<td>GND</td>
<td>SPI1 CS1</td>
</tr>
<tr>
<td>34</td>
<td>SPI1 MOSI</td>
</tr>
<tr>
<td>30</td>
<td>SPI1 SCLK</td>
</tr>
</tbody>
</table>

**RASPBERRY PI PINOUT LEGEND**

- **Orange** = GPIO pins (i.e. pins we can program)
- **Black** = Ground
- **Red** = Power (5V)
- **Yellow** = Power (3.3V)
4. BCM Vs. Board

While we’re talking about the Raspberry Pi pinout, it’s worth mentioning the difference between BCM and Board coding.

All of the codes that we will be using are written in Python Code. (more on that in Part 3). For now, we want you to notice that we have to write instructions to the GPIO pins otherwise they don’t know who we’re talking to. It would be like being in a classroom full of people and saying “Please open the window.” Everyone in the room would look around, wondering if they should go open the window or if they should wait for someone else to open it, and in the meantime, time is ticking away. It would be much more efficient to look at a specific person and say, “Jerry, would you please open the window?” Jerry would respond quickly and open the window. It’s a similar idea here. GPIO pins need to know that you’re talking to them otherwise they won’t work.

In our example of opening the window, we could have also addressed Jerry by his last name: Smith, and he would have responded just as well.

Well, GPIO Pins have two names, sort of like we have first names and last names. But instead of a “first name” and a “last name” they have a “BCM name” and a “Board name.” It doesn’t really matter which one you use, but you must be consistent in the code. You can’t call one student by their first name and everyone else by their last name. In the same way, you have to call everyone by their Board name or by their BCM name. You can’t call one GPIO pin by its Board name and a different pin by its BCM name in the same code.

Anyway, that’s a long explanation to point out what we want to point out below, which is that the BCM name is pointed to in green and the Board name (number) is pointed to in cyan.

For example, we will be using the GPIO pin circled in blue below. Our code is written in BCM, so we will call this pin: Pin 2.

That’s all you need to know about the Pi for now. Let’s move on to Section 2: The Wiring.
Section 2: The Wiring
Engineering requires two main components: Wiring and Coding. Most people find they like one part better than another. In this section, you’ll get some practice with wiring.

Materials Needed:
1) Raspberry Pi
2) Breadboard
3) DHT11 temperature sensor
4) 3 pinout wires (one end male, one end female)

Step 1: This is what engineers call a “breadboard.” It is designed to make wiring the Raspberry Pi easy and organized. In this tutorial, we will walk you through how a breadboard works and how to use one. We will assume that you have had no prior experience using a Raspberry Pi, so be patient with us if you’ve had some prior experience.
Breadboard Basics:
1. Columns and Rows
   - Notice that there are columns and rows in a breadboard. The columns are labeled in lowercase letters from A to J across the top of the board (see picture below).

   ![Column and Row Illustration](image1)

   - Notice that there are small numbers along the rows. These are the row numbers, marked in this example every 5 rows. (See picture below.)

   ![Row Number Illustration](image2)

   - As an example, if we want to plug something into row 4 letter D, we would be speaking about the compartment circled below:

   ![Example Illustration](image3)
2. Power and Ground
All circuits deal with power (+) and ground (-). Power is usually represented with red while ground is usually represented with black or blue. Notice that there are two at the left and right of the breadboard that are separated by a red line and a blue line. The red row is intended for power. The blue row is intended for ground.

3. How Power Flows
Breadboards are powerful tools because they allow engineers to keep their circuits organized. You’ll get the hang of how these work, but we want to get you familiar with how the electricity flows in these breadboards. There are two main ways that power flows through breadboards: Horizontally through rows and Vertically through the Power/Ground inputs. We’ll explain more about this as we get our sensor hooked up in the following steps.

Step 2: A picture of a DHT11 sensor is shown below. This sensor is designed to read temperature and humidity of a given room. Just like the Raspberry Pi has pins, your DHT11 sensor has 3 pins. One of them is for power (+), one for collecting data, and one for ground (-). Notice that these three pins are labeled on your sensor.
We want to insert the DHT11 sensor into the breadboard so that we can wire it to the Pi. The first step is to insert the pins into three separate rows on your breadboard, as shown in the schematic below.

(Note: all three of the pins should be in separate rows.)
**Step 3:** We will connect a male/female cord from the Raspberry Pi to the DHT11. We will start with the power. Notice that we are using a red cord in this picture because power is conventionally represented with red. We will then insert the male end of the cord into the SAME ROW that the DHT11 sensor is plugged into.

Compare this image to the Raspberry Pi pinout we showed in Section 1. Notice that we are using the female end of the red cord to plug onto the Raspberry Pi 5V Power on the Raspberry Pi.
**Step 4:** This is an explanation step. Remember in Section 1 when we said we would explain more of the Horizontal/Vertical way that electricity flows through a breadboard? Now we will explain this a little more thoroughly.

*Horizontal Flow:*
Power is coming out of the Raspberry Pi in 5V and into the breadboard. Notice the row is lit up in green below. That means that the entire ROW 1 is receiving power. If we plug in another sensor or another wire into that cord, it will receive power as well. So, since our DHT11 power pin (+) is plugged into that row, it is receiving power from the Raspberry Pi. We call this “horizontal flow” because the power is flowing horizontally through the row of the breadboard. We’ll come back to Vertical Flow a little later.

Notice that the power is flowing only through the five slots: A, B, C, D, and E. Power does NOT flow horizontally from A-K. It *only* flows from A-E or through F-K. It does not flow through the center divot.

**Step 5:** In this step, we will connect a male/female cord from the Raspberry Pi to the data pin of the DHT11. We used a blue cord in this example, but the color does not matter in this case.

Note that the female end connects from the Rasperry Pi. We chose to use GPIO 2. You could have chosen any of the pins labeled “GPIO ___” in the pinout picture below, but stick with GPIO 2 for now otherwise your code won’t work. Don’t worry, we will change it and experiment with it later.

The male end of the cord plugs into the breadboard. *Be sure to insert it into ROW 2, which is where the data pin is plugged in.*

Your pi should look like the schematic shown below:
Step 6: We will now insert the final cord, the ground. We will use the conventional black cord for this one. Notice again that we will use the ground pin from the pinout below. The female end will connect to the Raspberry Pi. The male pin will plug into the breadboard. Be sure to plug the cord into the same row (ROW 3) of the sensor’s ground pin (denoted with a -).

Note: Raspberry Pis have multiple ground pins on the board (shown below). We could have used any of these pins, but we chose to use the one we did for the sake of simplicity.

Your completed setup should look like this shown below:

INSERT SCHEMATIC

Step 7: From here on out, all wiring schematics will not have explanation of the male/female cords or of how electricity flows through a Raspberry Pi. In future tutorials, you will be simply shown a finished schematic (like the one shown below) and you will be expected to read the schematic and mimic the step on your own.

Congratulations! Your DHT11 is now powered up and ready to go. Up next: the code.
Section 3: The Code

Engineering requires cohesion between two parts: the wiring and the coding. We completed coding in Section 2, and we will move to the coding in this section.

Step 1: Before we get into it, we want to use this as an explanation step to help you get some bearings on coding. Computer codes are written in different languages, just like humans speak different languages. We will be using code written in Python 3.

Coding Conventions for Python 3

Every language has certain parts of speech that are present: subjects, verbs, prepositions, etc. If speaking in English, you may say “I speak” where “I” is the subject and “speak” is the verb. In Spanish, you would say “Yo hablo” where “yo” is the subject and “hablo” is the verb. They’re different languages that use different words, but they BOTH have subjects and verbs.

The same is true for computers. Most codes have some “parts of speech” like: Libraries, Comments, Functions/Definitions, and Main Programs. The nitty gritty of how you say those things differ depending on the language, but these 4 parts are pretty consistent across all languages.

We’ll introduce you to the basics below:

1) Library – Libraries are powerful things. They have a TON of code written in them. When you import a library into your code, you are able to access all of the knowledge and codes that are built into your library already.

2) Comments – Engineers often use comments in their codes to explain to themselves or others what steps in the code mean. They don’t affect the code itself but just allow others to understand what is happening. In Python, we use a # to write a comment. For bigger sections of comments, we can also use three apostrophes (‘ ‘ ‘) to make comments.

3) Functions/Definitions – These are sort of like things that we set up to define what exactly we want the code to DO.

4) Main Program – This is where we actually write what we want the code to do. You could think of this sort of like a verb.
Step 2: Now we’re going to actually do something. On your Raspberry Pi, you will want to access the file called dht11.py.

*Note: Files that end in `.py` mean that they are written in Python code, just like a document that ends with `.doc` or `.docx` on a Mac or Windows denotes a file written using Microsoft Word.*

To access the file, click on the folder icon at the top.

Then double click on the folder called “Sensor Tutorial Files.”

Then double click on the folder called DHT11_Python-master.
Then double click on the file called “dht11_example.py.”

Note: Remember how we said that Raspberry Pis are sort of like computers? Can you see how the Raspbian Operating System is similar to Windows?

Step 3: A new window should’ve opened that looks like the one below. This is the actual file with the code inside. Your code should look like this:
Step 4: It’s time for some explanation again. Python Codes have some basic structure to them. Just like you might have a morning routine that you perform in the same order every day (i.e. wake up, get dressed, eat your breakfast, brush your teeth), Python Codes have an outline is that is the same for each file that you will use.

There are four basic parts of a Python code that each have a specific function. We’ve listed the name of the part and the function below.

1) **Import Libraries** – This is where you tell the program what libraries you want access to.
2) **GPIO Setup** – This is where you tell the program which GPIO pins you want to use. You will also decide in this section if you will be speaking to the pins using BCM or Board.
3) **Functions/Definitions** – This is where you can create your own variables that you’ll use later in your program. For example, if you’re using a temperature sensor (like we are here), we can define a variable to be temp.
4) **Main Program** – This is where you actually tell the program what to do. In our example, we use a “while loop” to do this. More on that later.

```
# import Libraries
import RPi.GPIO as GPIO
import dht11
import time
import datetime

# GPIO Setup
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.cleanup()

instance = dht11.DHT11(pin=2) # read data using GPIO pin 2

while True:
    result = instance.read()
    if result.is_valid():
        print("Last valid input: " + str(datetime.datetime.now()))
        print("Temperature: %d %s" % (result.temperature, result.humidity))
        print("Humidity: %d %s" % (result.humidity, result.humidity))
    time.sleep(1)
```
Step 5: A few other things to point out. Notice that this program is written using GPIO.BCM. This means that we are calling the pins by their “BCM name.” The sensor is plugged into GPIO 2.

Also notice the “print” instructions in the code. Print in this context doesn’t mean print on a printer. It means “display on the computer.”
**Step 6:** We will now RUN the program. You can do this by pressing F5.

**Step 7:** A new window should appear like the one shown below. Some blue letters will appear with a valid reading date, a temperature reading, and a humidity reading.

![Python 3.4.2 Shell](image)

**Step 8:** If you wait another second, more blue words will appear with another temperature and humidity reading.

![Python 3.4.2 Shell](image)

**Step 9:** To end the code, press Ctrl+C. This is the standard way engineers end programs.

**Congratulations! Your temperature sensor is now collecting data and displaying it for you to see! It is officially working.**
Section 4: Experimenting for Learning

1. Vertical Flow of Electricity in a Breadboard
Since we’re still learning, we want to take the time now to explain how power is transmitted *Vertically* through a breadboard. We discussed how it is transmitted *Horizontally* through rows in Section 2 Step 4, but now we want to move onto something a little more complex in this part because we know you can handle it.

Remember those two special columns that have capital letters: Column A (next to the blue line) and Column B (next to the red line)? These are special and separated by those colored lines for a reason.

In future tutorials, you will perhaps need to setup multiple sensors and LED lights that all need power. The Raspberry Pi only has a few power outputs, so it is more efficient to use ONE power pin on the Raspberry Pi to plug power into MULTIPLE sensors. That’s what this vertical Column B is for.

Notice that Column B is next to the red line. This denotes power. It is intended for users to connect power from the Raspberry Pi into that row. Practice doing this now.

Take the male end of your red cord and plug it into Column B.

*Before Picture:*

*After Picture:*

Now, the ENTIRE Column B is receiving power. (Look at the green dots of the schematic below.) This is powerful because you now have 30 compartments of power that are all receiving power from ONE pin on the Raspberry Pi.

You can think of this Column B kind of like a power strip (picture shown below). It is plugged into ONE outlet in the wall (i.e. the Raspberry Pi) and it has 6 new outlets you can plug other devices into (like the breadboard has 30 new outlets you can plug other sensors into).
If the red column denotes power, then the blue column denotes ground. Unplug your black ground cord from the breadboard and move it to the red column. Now there are 30 “activated” ground compartments on your Pi that you can plug any device into to receive power.

2. Changing GPIO Pin numbers in Codes
Our code started out using GPIO 2. Let’s move the female end of the blue cord to a different GPIO pin. You may choose any pin shown in the pinout below that is in orange and says GPIO #.

Once you have moved your pin, change the number in the Python code from 2 to the GPIO pin shown in the Pinout.

For example, we moved our pin to GPIO 21. Our code now looks like this:

```
# import Libraries
import RPi.GPIO as GPIO
import dht11
import time
import datetime

# GPIO Setup
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.cleanup()

instance = dht11.DHT11(pin=21) # read data using GPIO pin 21
```

Our setup matches the schematic like this:

INSERT SCHEMATIC

Run your code. Did it work?
3. Changing from BCM to Board.
Our code is written in BCM. Let’s change it to Board. In the line of instructions that reads GPIO.BCM, change this to GPIO.BOARD, as shown below:

Now change the pin number from 21 to 40. We are still using the same pin, but we are calling it by its “Board name” rather than its BCM name.

4. Using Comments - Hashtag
Hashtags can be used to add comments to the code. They can also be used by engineers to just cancel out a code without deleting it. This is very useful when you just want to ignore some information for a short period of time but may want to use it again later.

Say that we want to only view the temperature and not the humidity. Rather than delete the humidity, we can add a hashtag in front of this line. This makes it a comment, which is not part of the code.

Add a hashtag to line 20. It should look like this:
Now press F5 to run your code. The blue printout should look like this:

```
Last valid input: 2017-08-16 21:30:08.385985
Temperature: 24 C
Last valid input: 2017-08-16 21:30:09.505921
Temperature: 24 C
Last valid input: 2017-08-16 21:30:10.660397
Temperature: 24 C
Last valid input: 2017-08-16 21:30:11.823593
Temperature: 24 C
```

Did you notice how the humidity is no longer displaying? Only the temperature data is displaying on the screen. Cool huh?
5. Using Comments – Apostrophes
When we want to block out larger sections of code, we can use three apostrophes ‘’’ to do this. These three sets of apostrophes act sort of like parentheses. Parentheses have an open parenthesis like this: ( The information in the parentheses must eventually be closed with a close parenthesis like this: ). Apostrophes work like that too. We use three apostrophes to “open” the parentheses ‘’’. Then we put three apostrophes at the end of the part that we want to close in parentheses. The words inbetween our apostrophe parentheses will appear green, like the picture below.

Try it out! Delete the hashtag in line 20. Let’s “comment out” lines 18-19, which print the “last valid input” and the “temperature” readings. Add three parenthesis at the start and end of that line of code, like this:

```
while True:
    result = instance.read()
    if result.is_valid():
        print("Last valid input: " + str(datetime.datetime.now()))
        print("Temperature: %d C" % result.temperature)
        print("Humidity: %d %" % result.humidity)
```

Press F5 to run the code. The printed data now looks like this:

```
Humidity: 49 %
Humidity: 50 %
Humidity: 49 %
Humidity: 49 %
```
6. Change the Time
The command `time.sleep(1)` tells the sensor to report the data every 1 second. To make it report the data once every 5 seconds, change the line to read `time.sleep(5)` as shown below.

```python
# import Libraries
import RPi.GPIO as GPIO
import dht11
import time
import datetime

# GPIO Setup
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.cleanup()

instance = dht11.DHT11(pin=40)  # read data using GPIO pin 40

while True:
    result = instance.read()
    if result.is_valid():
        print("Last valid input: " + str(datetime.datetime.now()))
        print("Temperature: %d C" % result.temperature)
        print("Humidity: %d %%%" % result.humidity)

    time.sleep(5)  # This reports the reading every 1 second. To make it report d
```

CONGRATULATIONS! You have learned about Breadboards, DHT11 Sensors, and Python Codes. You are ready to move along to the next tutorial.