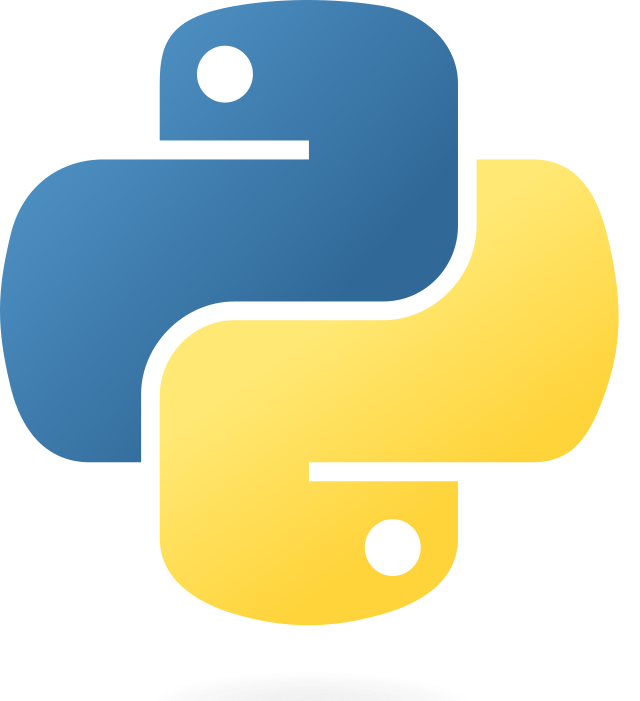
**Constructing and Developing Heat Sensors Manual**

A how-to guide for setting up and installing your own Heat Sensors





Written as Supplementary Material to an Interactive Qualifying Project in partial fulfillment of a Bachelors of Science Degree from Worcester Polytechnic Institute.

6 March, 2023

Written by:

Emmanuel Vargas and Stephen Fanning

["Google Developers text logo"](https://developers.google.com/) by Google is in the [Public Domain, CC0](http://creativecommons.org/publicdomain/zero/1.0/)

["Raspberry Pi OS Logo"](https://www.raspberrypi.org/documentation/) by Raspberry Pi Foundation is licensed under [CC BY-SA 4.0](http://creativecommons.org/licenses/by-sa/4.0)

["Python-logo-notext"](http://www.python.org/) by [Python](http://www.python.org/) is in the [Public Domain, CC0](http://creativecommons.org/publicdomain/zero/1.0/)

# Section 1: Description of Parts

This section contains a list of every part necessary to assemble a heat sensor and a description of what they do. This section also provides enough information to find alternatives for a specific part if necessary, for example if a part turns out to be insufficient for the purposes of the heat audit.

Raspberry Pi Pico W

[Purchase link](https://core-electronics.com.au/raspberry-pi-pico-w-with-soldered-male-headers.html)

[Documentation](https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html#raspberry-pi-pico-w)

[Datasheet](https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf)

**Description:**

The Raspberry Pi Pico W is a low cost programmable single-board computer (SBC) designed for low power applications, such as this heat audit. It is very similar to the Raspberry Pi Pico, with the one difference being a built-in WiFi chip that allows it to access the internet, and thus upload temperature readings to Google Sheets. In our implementation, the Pico W is programmed using MicroPython, a variant of Python designed specifically for embedded systems (See Section 2: Software Description for more details). The Pico W can be programmed by connecting it to a computer using its Micro-USB port, and with the assistance of an IDE such as Visual Studio Code (with PyMakr) or Thonny. The board can be powered in two ways, either through the Micro-USB port or through the VSYS and GND pins, and it must be powered with a voltage between 1.8V and 5.5V. When powered, the Pico W will run the software stored on the board automatically.

The Raspberry Pi Pico W can be purchased with or without pre-soldered pin headers. Our team ordered them without pre-soldered headers in order to save cost, but for anyone without soldering experience it is highly recommended to buy the Pi Pico W’s with pre-soldered headers.

**Alternatives:**

It is difficult to envision an alternative to the Raspberry Pi Pico W in this implementation since it is an extremely low cost SBC, and it is difficult to find an alternative for any less than double its price. However, this exact project could be replicated using an ESP8266, an ESP32, or really any SBC with built-in WiFi capabilities.

Adafruit DHT20

[Purchase Link](https://littlebirdelectronics.com.au/products/dht20-aht20-pin-module-i2c-temperature-and-humidity-sensor)

[Datasheet](https://aqicn.org/air/sensor/spec/asair-dht20.pdf)

**Description:**

The DHT20 (also called the AHT20) is a low-cost temperature sensor module. It is capable of measuring temperature upon request from the microcontroller, and communicates its reading back to the board using I2C communication. Any information regarding specific ratings can be found on the data sheet.

**Alternatives:**

The DHT20 can be replaced with virtually any temperature sensor module, although it will almost certainly involve altering the software to allow it to work. It is possible that a different sensor module that also uses I2C communication could also work, but it is unlikely since the library used within the software is built specifically to interface with the DHT20. Assuming the software can be altered, some examples of alternative parts include the DHT22, DS18B20, PCT2075, or any other temperature sensor module.

Power Supply

[Battery Purchase Link](https://www.ebay.com.au/itm/304318442409?hash=item46dacaeba9:g:NzoAAOSwQeNh54~y)

[Holder Purchase Link](https://www.jaycar.com.au/2-x-d-cell-side-by-side-battery-holder/p/PH9220)

[Battery Life Calculator](https://docs.google.com/spreadsheets/d/1eO7q2Gn5HJnr02GhVSOUE7WyeFM625_JQ-Ds7tQcH9I/edit#gid=1726859087)

**Description:**

In order to power the sensor, this implementation utilizes two unbranded D-size 10,000mAh rechargeable batteries. We decided to use D batteries because they hold a large amount of charge, and we would like as much room as possible for battery degradation to occur while still being able to last a minimum of five weeks between recharges. However, with these batteries, the sensors are theoretically able to last significantly longer than they need to.

Assuming the batteries perform as advertised and provide 10,000mAh of charge, each sensor should be able to provide well over five months of readings before needing a recharge. This is based on the fact that the sensors draw ~1.8mAh during sleep, and ~45mAh on average while processing a reading, which takes about 80s (these measurements were taken using a multimeter).

However, we measured the batteries’ capacity to be 7,190mAh, rather than the 10,000mAh advertised. This lowers a sensor’s theoretical lifespan to 3.6 months, or slightly over 15 weeks. Some sensors, especially those that struggle to connect to WiFi, often take more than just 80s to process a reading, since they take much more time to connect to WiFi. Assuming an (exaggerated) estimated processing time of 4 minutes, each sensor would still last 9 weeks between recharges, still leaving ample room for battery degradation.

Note that these batteries only provide ~1.4V fully charged, whereas the Raspberry Pi Pico W requires a minimum of 1.8V to function properly. For this reason, the sensors use two batteries each, connected in series to double their voltage. These batteries are connected using a standard 2 x D battery holder.

**Alternatives**

Theoretically, any battery can be used to power the sensor as long as it provides between 1.8V and 5.5V and can be recharged. For example, a rechargeable AA battery typically provides 1.2V and 2,000-3,000mAh of charge. Connected in series (like the D batteries are in the current implementation), they could provide 2.4V and 2,000-3,000mAh and theoretically power a sensor for over 5 weeks, but this leaves almost no room for the battery to degrade over time, or for the battery to perform worse than advertised (as is often the case), so it is not recommended.

As alternative batteries, we considered [these branded D batteries](https://www.amazon.com.au/EBL-Capacity-Rechargeable-Batteries-Included/dp/B078SS6J77/ref=sr_1_1_sspa?crid=239JTW0NRLSAO&keywords=rechargeable+D+batteries&qid=1676000916&s=electronics&sprefix=rechargeable+d+batteries%2Celectronics%2C275&sr=1-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUExS1dUTlY5MzBZMDNJJmVuY3J5cHRlZElkPUEwNzc3MDE2UDU0VEdUV0kzODJXJmVuY3J5cHRlZEFkSWQ9QVIyODNQRUw2Q1Y3OSZ3aWRnZXROYW1lPXNwX2F0ZiZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWNrPXRydWU=), which are likely more reliable than the unbranded versions initially purchased, but are also significantly more expensive.

We also considered using [these C batteries](https://www.amazon.com.au/EBL-Capacity-Rechargeable-Batteries-Included/dp/B078SQWHYR/ref=asc_df_B078SQWHYR/?tag=googleshopdsk-22&linkCode=df0&hvadid=341743289727&hvpos=&hvnetw=g&hvrand=4303785308425172678&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9071395&hvtargid=pla-675034167105&psc=1), which are a reliable cheaper option, but only store 5,000mAh (and one person online measured that they actually only store 4,000mAh). While this theoretically does provide enough charge to last over five weeks, it is unclear how long they will maintain this charge for. Using C batteries as alternatives will also require purchasing C battery holders instead of D battery holders.

It is also possible to achieve enough voltage and capacity using four AA rechargeable batteries, [such as these](https://www.amazon.com.au/EBL-2800mAh-Rechargeable-Batteries-Battery/dp/B00DNPT1AO/ref=sr_1_4?keywords=aa%2Brechargeable%2Bbatteries%2Bbulk&qid=1675991689&s=electronics&sr=1-4&th=1). This would involve using two 2 x AA battery holders (which connect the batteries in series), and connecting these holders in parallel to power the sensor. This would accomplish doubling the voltage of an individual battery from 1.2V to 2.4V, and doubling the capacity from 2,800mAh to 5,600mAh. This would serve as a viable alternative to the two D batteries, assuming these batteries work as advertised, but it would be more difficult to wire.

Early prototypes of the heat sensors involved using a rechargeable power bank designed for charging iphones, but these require manually pressing a power button frequently, so these are not a suitable alternative for our purposes.

Note that, before purchasing any different batteries, it is highly recommended to use the Battery Life Calculator spreadsheet linked above to estimate how long a new battery should theoretically last.

# Section 2: Software Description

Software is key to the functionality of the sensor, and any fundamental changes to the sensor will have to be reflected in the software as well. This section describes the software to help in case such a change becomes necessary.

**Programming Language**

The sensors were programmed using MicroPython, a variant of Python designed specifically for embedded systems. Micropython includes much of the same basic syntax and functionality as Python, but allows different libraries that enable more control over lower level functionality of the microcontroller. We chose MicroPython because of its support for deep sleep, which is necessary to lower the sensors’ current draw to 1.8mA between readings. This functionality also exists in C++, but, at the time this project was completed, MicroPython had better libraries for the DHT20 and some reasonable methods for connecting to Google Sheets, as described in the next section.

**Base Code**

The software in the sensors is heavily based on [this ESP32 Weather Station](https://blog.gypsyengineer.com/en/diy-electronics/weather-station-based-on-esp32-and-micropython.html), and it is highly recommended to read through the article before making any major changes to the code. The code from this weather station provided a base for connecting to Google Sheets, and we built all of the other functionality on top of it. This was necessary since there are no existing libraries for MicroPython to interact with Google Sheets, and the creator of the ESP32 Weather Station reworked Python libraries to allow them to work with MicroPython.

**Key Functionality**

At its core, the sensors need to be able to perform a few key functions. These are, in order:

* Connect to WiFi
* Sync Time
* Take temperature readings
* Upload readings
* Deep Sleep

The sensor first connects to WiFi using MicroPython’s built-in ‘network’ library. It then needs to sync the on-board clock to the current time using NTP (Network Time Protocol) in order to connect to Google Sheets, provide a time stamp for its readings, and know when to wake up. The base code utilized a custom implementation for this, but we replaced it with MicroPython’s built-in ‘ntptime’ library. Then, to take temperature readings, we utilized this [Pico-DHT](https://github.com/flrrth/pico-dht20) library made for this exact purpose. These readings are then uploaded to the google sheet using the same process as outlined in the ESP32 Weather Station article. After this process is complete, we used the ‘machine.deepsleep’ function to put the board to sleep until the next time it should take a reading. Note that, when the Pico enters deep sleep, it can no longer maintain its USB serial connection with a computer, and thus any terminal output will be lost.

**Error Handling**

Throughout this process, the sensor undergoes a number of actions that have a high likelihood of failure. For this reason, error handling and reattempts are necessary to ensure that the sensors work consistently. For each action that has an ability to fail, the sensor will retry the action upon failure until the number of retries exceeds the constant ‘RETRY\_LIMIT’, by default set to 50. If this happens, the sensor will log an error in its ‘log.txt’ file, and sleep until it needs to try another reading. The sensor also does this if it encounters a software error at any point. If it does encounter an error, the traceback of the error will be recorded in ‘log.txt’ before sleeping.

The ‘log.txt’ also stores a record of every successful reading. Long term, this could lead to issues with the log file becoming too long, so the code automatically shortens the file, deleting earlier entries first, when it exceeds 100 lines (this number can be changed through the ‘MAX\_LOG\_LENGTH’ constant). Combined, all of this error handling allows for consistent operation even in inconsistent circumstances, and it makes debugging issues encountered along the way much more manageable.

# Section 3: Heat Sensor Assembly

Whether it is because of a hardware failure or a demand for more sensors, it may be necessary to construct an additional sensor from scratch after our team has left Melbourne. This section provides step-by-step instructions to build a brand new sensor and incorporate it into the existing system. In addition, for a guided visual please visit our [Heat Sensor Tutorial](https://www.youtube.com/watch?v=NXjj1oKcEbQ) Youtube video.

Part 1: Wiring the Heat Sensor

1. **Components required to ensure compatibility and functionality.**

* [Raspberry Pi Pico W](https://core-electronics.com.au/raspberry-pi-pico-w-with-soldered-male-headers.html?gclid=CjwKCAiAuaKfBhBtEiwAht6H7xBGR9Je1NX_EwGeTsdjGkBobmIEHfcXZOY9vmVTjvysJMGrkhDJJxoCNQcQAvD_BwE)[[1]](#footnote-0)
* [DHT20/AHT20 Temperature Sensor](https://littlebirdelectronics.com.au/products/dht20-aht20-pin-module-i2c-temperature-and-humidity-sensor)
* [2x D Battery Holder](https://www.jaycar.com.au/2-x-d-cell-side-by-side-battery-holder/p/PH9220)
* [Female-Female Jumper Wires](https://littlebirdelectronics.com.au/products/female-female-2-54-to-2-0mm-jumper-wires-x-40)
* [Snappable Double Row Right Angle PCB Pin Headers](https://www.amazon.com/uxcell-2x40-Pins-Connector-Arduino-Prototype/dp/B07DK532DP)[[2]](#footnote-1)
* [D Battery Charger](https://www.amazon.com.au/EBL-Universal-Rechargeable-Intelligent-Technology/dp/B07Y5ZB8Z5/ref=sr_1_6?adgrpid=86809873883&gclid=CjwKCAiA0JKfBhBIEiwAPhZXD5c_OtglSzQHI31VSp6DFpGHJbWV7ge6DLoIb_yYlYaHpO3UDJgvgRoCM1sQAvD_BwE&hvadid=414047470205&hvdev=c&hvlocphy=9071395&hvnetw=g&hvqmt=e&hvrand=13834594474129361508&hvtargid=kwd-117837437&hydadcr=19792_13447&keywords=d+battery+charger&qid=1676002244&sr=8-6)
* [D Cell Batteries](https://www.amazon.com.au/EBL-Capacity-Rechargeable-Batteries-Included/dp/B078SS6J77/ref=sr_1_1_sspa?crid=239JTW0NRLSAO&keywords=rechargeable+D+batteries&qid=1676000916&s=electronics&sprefix=rechargeable+d+batteries%2Celectronics%2C275&sr=1-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUExS1dUTlY5MzBZMDNJJmVuY3J5cHRlZElkPUEwNzc3MDE2UDU0VEdUV0kzODJXJmVuY3J5cHRlZEFkSWQ9QVIyODNQRUw2Q1Y3OSZ3aWRnZXROYW1lPXNwX2F0ZiZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWNrPXRydWU=)
* USB to MicroUSB Data Cable[[3]](#footnote-2)
* Electrical Tape

1. **Connecting the Raspberry Pi Pico W with the DHT20/AHT20 Temperature Sensor**
   1. Orient the DHT20/AHT20 sensor so that “ASAIR” is facing upwards. (As shown in the image below).

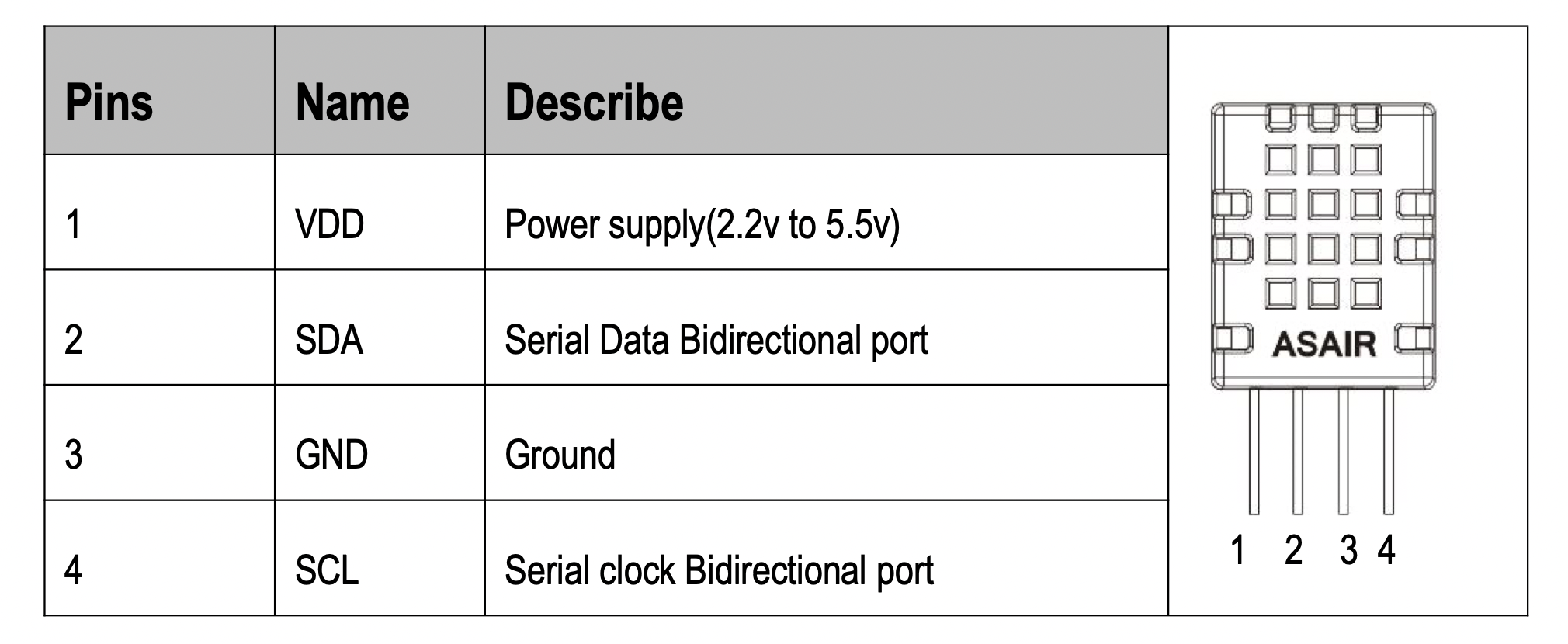
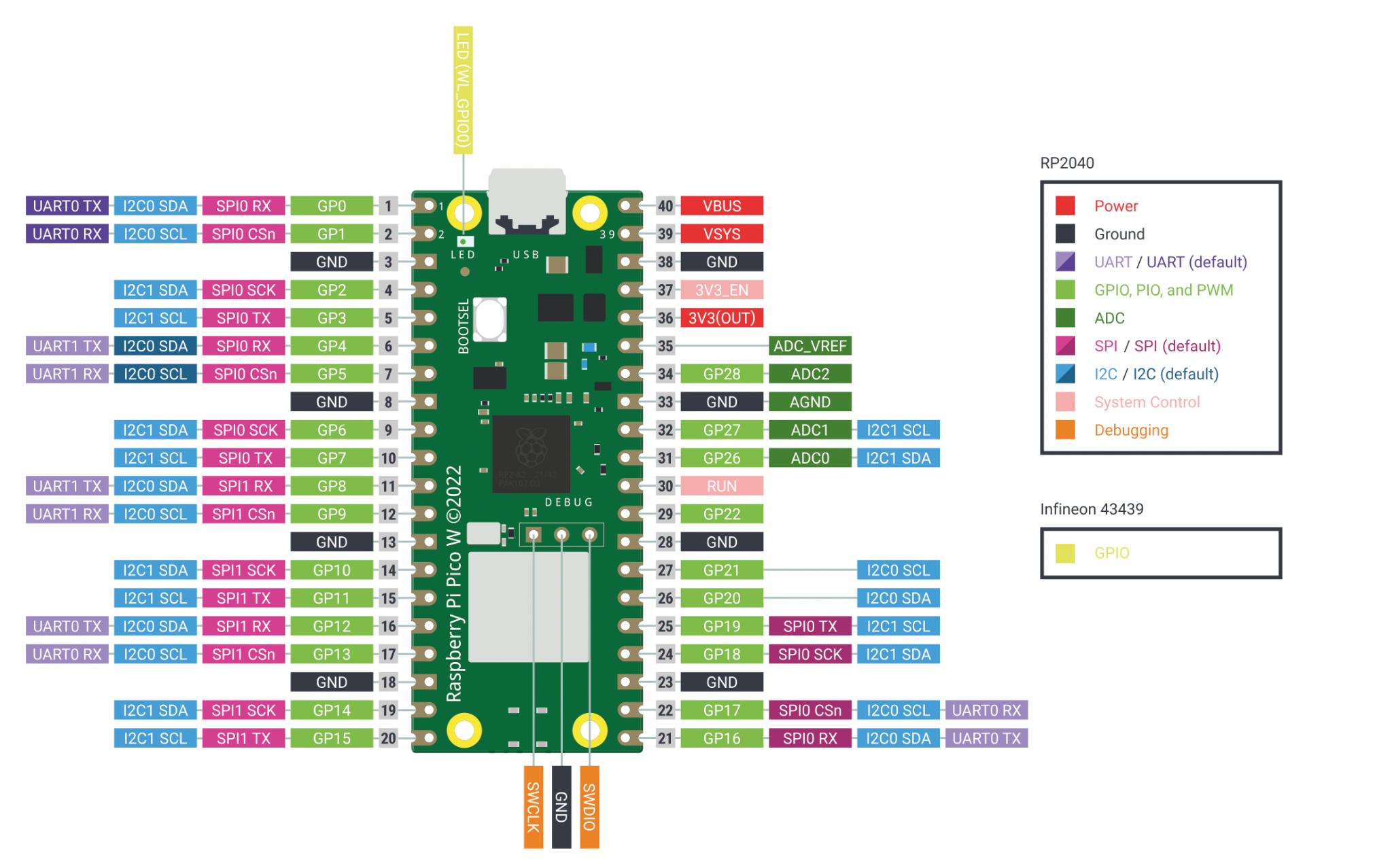
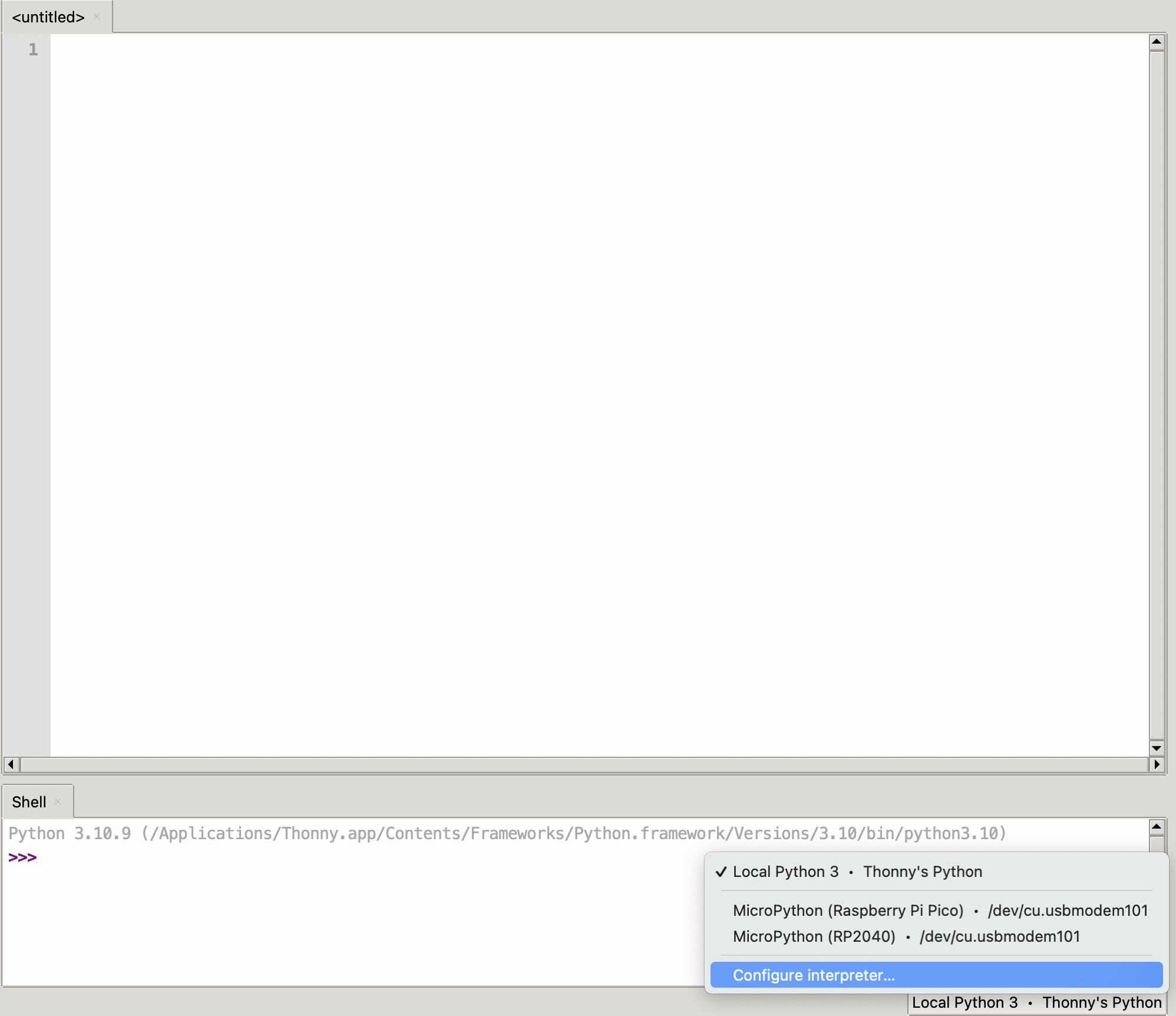


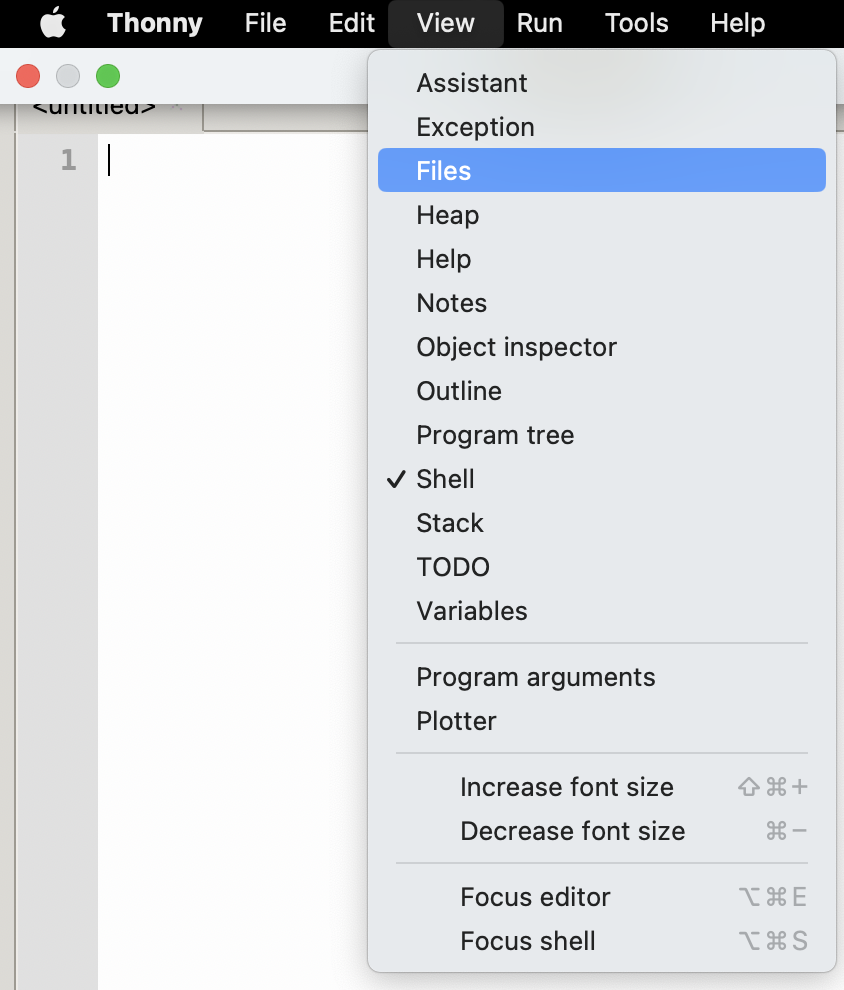
Figure 1: Schematic of DHT20/AHT20 Temperature Sensor

1. Use a Female-Female jumper wire to connect Pin 1 from the DHT20/AHT20 to 3V3 on Pico W microcontroller.
2. Use a Female-Female jumper wire to connect Pin 2 from the DHT20/AHT20 to GP8 on Pico W microcontroller.
3. Use a Female-Female jumper wire to connect Pin 3 from the DHT20/AHT20 to GND (any GND will suffice) on Pico W microcontroller.
4. Use a Female-Female jumper wire to connect Pin 4 from the DHT20/AHT20 to GP9 on Pico W microcontroller.

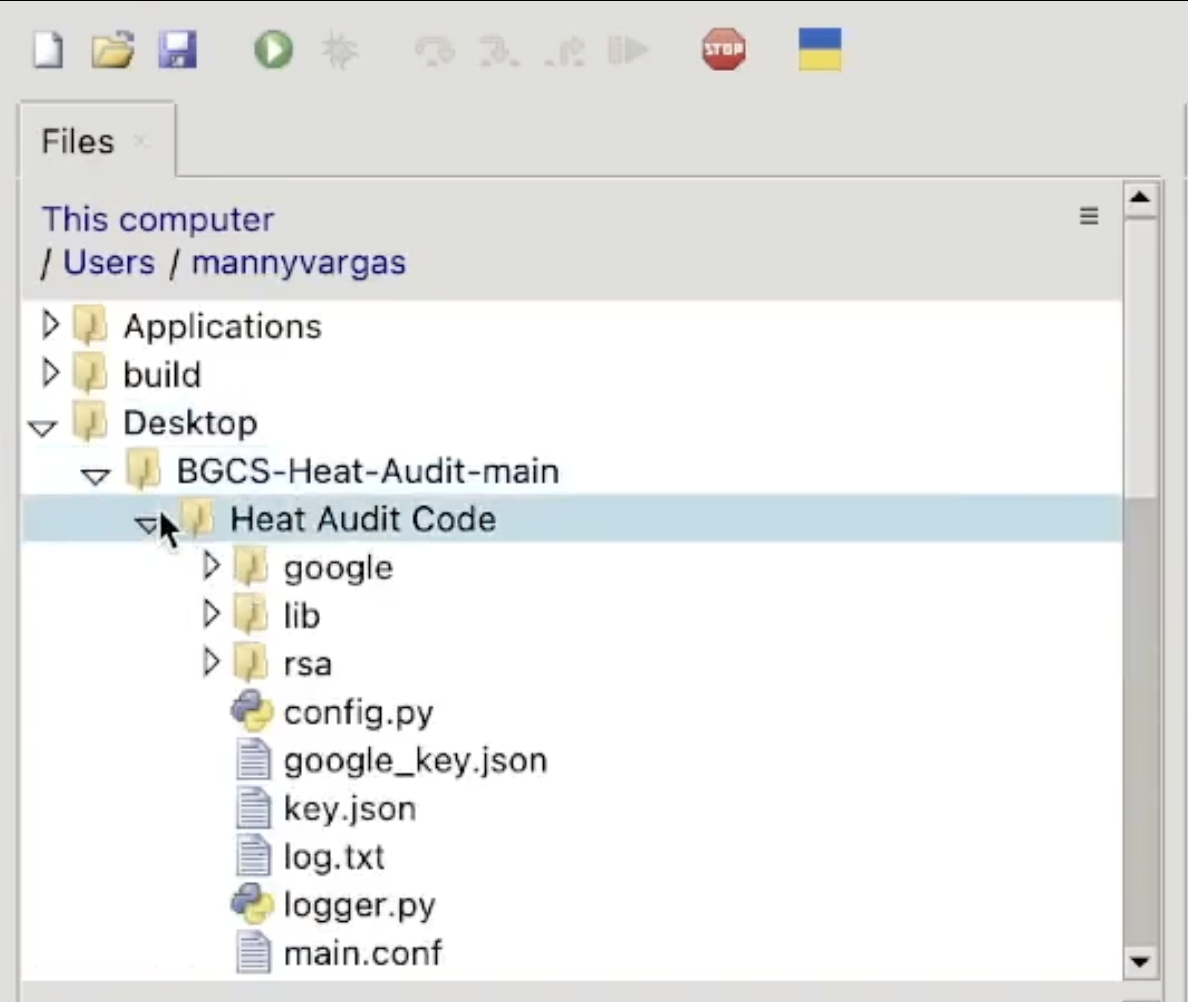
Figure 2: Pin Diagram or “Pinout” of Raspberry Pi Pico W

Part 2: Uploading Code to Heat Sensor

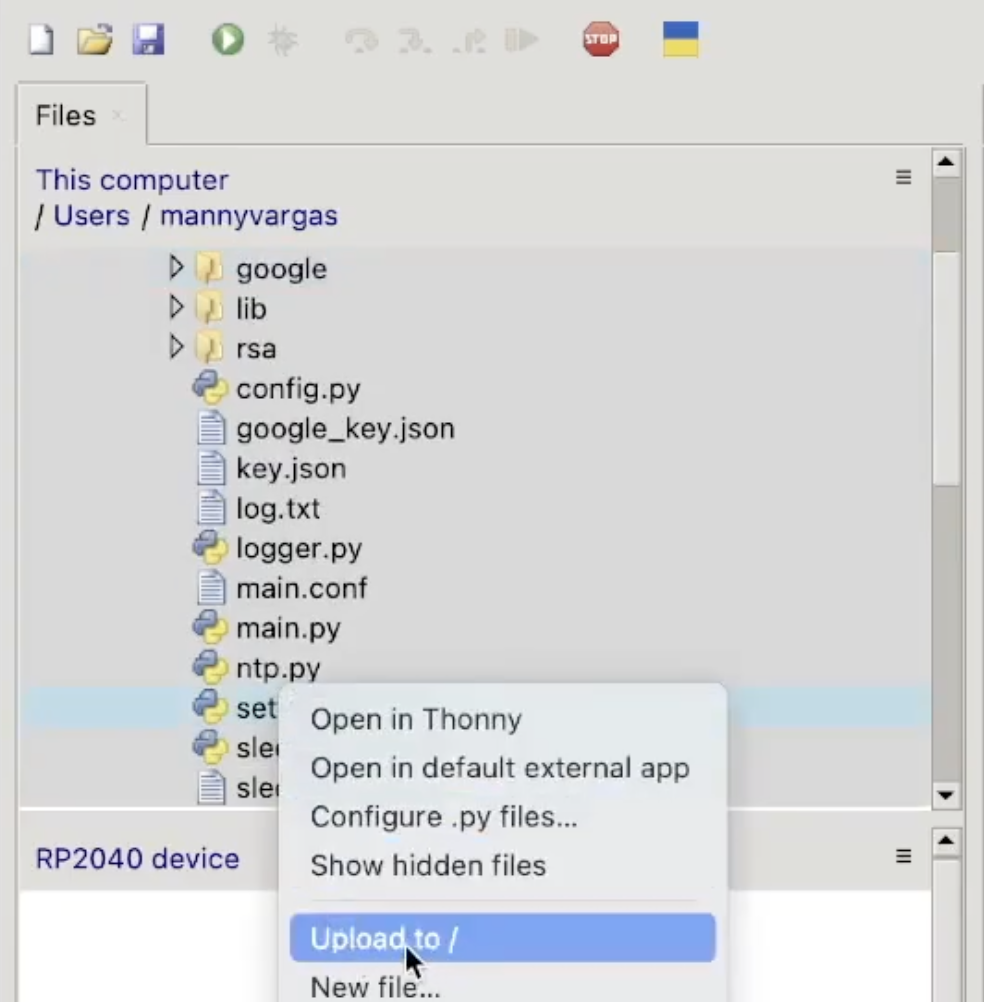
1. **Download necessary software and firmware**
   1. Download and install [Thonny IDE](https://thonny.org).[[4]](#footnote-3)
   2. Download the BGCS-Heat-Audit found on [Github](https://github.com/mannysvb/BGCS-Heat-Audit). This includes the firmware for the Pico W and the code required in order to run.[[5]](#footnote-4)
      1. Click the green “<> Code” button. 
      2. Select “Download ZIP”.[[6]](#footnote-5)
2. **Install firmware on Raspberry Pi Pico W**
   1. Connect the USB-MicroUSB cable to the Pico while holding the “BOOTSEL” button (white button).
   2. This will introduce a volume drive named “RPI-RP2”. Next open BGCS-Heat-Audit-main, locate rp2-pico-w-20230203…..(ending in .uf2), and drop the file into “RPI-RP2". If done correctly, the drive should now disappear from your file explorer.
3. **Install code on Raspberry Pi Pico W**
   1. Open the Thonny IDE application, if you see MicroPython v1.19.1-852………with RP2040 skip to step D.
   2. Click the bottom right tab of the application and select “Configure Interpreter…”
   3. Ensure the following fields are the same as the following:
   4. Open the file explorer pane within Thonny by navigating to View>Files.



* 1. Using the file explorer pane within Thonny, locate the folder Heat Audit Code.



* 1. Select all files and folders within “Heat Audit Code”, right click, and select “Upload to /”.



* 1. Once the files and folders have finished uploading, open “main.py” in RP2040 device, and ensure that line 40 in the code equals [“BGCS\_OFFICE", "BGCS\_SCHOOL"]. Additionally that line 22 accurately reflects the location in which the sensor will be placed in.
     1. These are the following locations to choose from: Grasslands, HeatHaven1, HeatHaven2, Food Forest, Community Gardens, Play Bushtucker, Project Real.[[7]](#footnote-6)
  2. Save the file, replug the USB, then select Stop/Restart backend by clicking  .
  3. Run the script with this button , the code should start running and after ~80s you should receive one temperature reading in Google Sheets. (It is likely that after running the script, red text may appear and ultimately force Thonny to crash and exit the application. Or the shell will show that data has successfully been uploaded, both scenarios are normal).

Part 3: Placing Heat Sensor Outside

1. **Assembling external power source**
   1. Using the provided tupperware, tape the inside part of the lid to prevent sunlight from coming through.[[8]](#footnote-7)
   2. Secure and tape the Pico W on the inside part of the lid so that it hangs upside-down.
   3. Poke 4 holes through the side of the tupperware and place a 4 pin header[[9]](#footnote-8) through the container.
   4. Connect the four wires from the Pico W to the header.
   5. Connect the four wires from the DHT20 Sensor module to the header (ensure that Part 1: Point 2 still applies).
   6. Place 2 x D Cell Batteries in the battery holder.
   7. Connect a Female-Female jumper wire from “VSYS” on the Pico W to the red wire from the battery holder (secure this connection with electrical tape).
   8. Connect a Female-Female jumper wire from “GND” (any GND will suffice) on the Pico W to the black wire from the battery holder (secure this connection with electrical tape).
   9. The board should blink several times after connecting the wire, if this does not happen then repeat step G again.

# Section 4: Troubleshooting

Due to the intended lifespan of this project, it is inevitable that errors will arise within the sensors at some point. Whether due to hardware failure, poor WiFi connection, or software issues, it is important to be able to find the source of the problem and address it properly. This section covers how to fix a sensor that is not working properly, including both how to find the source of a problem and how to fix it.

Part 1: Visible Errant Behavior

This part covers the types of errant behavior you might see in these sensors, and describes what the underlying problem causing that behavior likely is. In order to see the solutions to these underlying problems (underlined in the text), see the relevant header in Part 2. Note that for the majority of these behaviors, it is helpful to first consult the error log to see if the board was able to record any issues.

**Checking the Error Log**

For most issues, it is recommended to check the error log to see what issues specifically the sensor is encountering. To do so, connect to the Pico W as described in part 2, open “log.txt” to see what the error is. You could find each of the following errors:

* ERROR: WiFi failed to connect.
* ERROR: NTP failed to sync time.
* ERROR: Error encountered while sending data.
* ERROR: No response from spreadsheet.

Each of these errors means that the Pico W was unable to connect to WiFi at different stages of sending the reading. It is also possible that you will encounter the following:

* Software error encountered. Traceback below:

Followed by a traceback of where the software error occurred. This could mean a number of things. If the traceback contains the file ‘urequests.py’, it is once again likely that the board was unable to connect to WiFi. If the traceback contains ‘dht20.py’, then it is likely that the DHT disconnected. If neither are contained, then it is possible that the problem could be a software error.

**Missing Readings**

This is perhaps the most likely and hardest to solve issue within the entire heat audit. By far the most likely reason behind missing readings is that the sensor is unable to connect to WiFi. This is likely the case if you are seeing missing readings scattered throughout the data. This happens because the WiFi within the main building does not consistently reach the sensor, leaving the sensor unable to connect to the database and transmit its readings. This has been a consistent issue with the “Grasslands” sensor.

The second most likely reason behind missing readings is that the battery is dead, or the battery is disconnected. This is almost definitely the case if you see consistent readings for some time, and then a long, unending string of missed readings afterwards.

Another possibility if a long, unending string of missed readings is exhibited is a complete crash of the sensor due to a software error. This is unlikely due to the built-in error handling, but it is possible that the sensor could encounter a software error that might cause this behavior. This same behavior could also be due to a hardware failure, likely within the Raspberry Pi Pico itself.

**“DHT Disconnected” or readings of “-999”**

See DHT Disconnected.

**Unusual Temperature Readings**

If you notice temperature readings that seem abnormal, the first step is to verify that these readings are actually incorrect. For example, if it is a hot day and the sensor says 19, it is possible that the sensor itself is in the shade or buried. If these readings are clearly incorrect, then this is unfortunately indicative of a hardware error.

**Double Readings**

If the temperature sensors occasionally upload two readings in an hour, this is actually not a problem. If the temperature reading has the same time stamp, then this means that the sensor uploaded its data but did not receive confirmation (likely due to a sub-par WiFi connection). When this happens, the sensor will resend the data until it receives confirmation. This is not an issue since the extra reading is filtered out by the google sheet’s processing anyway, and “fixing” this issue would involve removing an important fail-safe.

If the readings are not at the same time stamp, but slightly apart, then this is also not a problem, but it is due to a known bug in the software. Specifically, this is likely due to the machine attempting to deep sleep for 0 seconds. However, due to the relative insignificance of this double reading (as described earlier) and the seemingly difficult nature of fixing this bug, it was decided that the bug was not worth the time to fix, especially due to the limited time our team had to complete the project.

**Readings at Unusual Times**

If the sensor is uploading temperature readings at times that are not relatively on the hour, then there are two likely explanations. The first of which is that the sensor was unpowered and re-powered. When a sensor is re-powered, it takes a reading and uploads that reading to the database regardless of what the time is. This is to confirm that the sensor is actually working as intended when it is initially placed. So, if there is a reading in the database at a time that is not right on the hour, it is possible that someone unplugged it and then plugged it back in.

Another possibility is that the sensor took a very long time to process the reading. This may or may not be a serious problem, but regardless it is almost certainly due to the sensor being unable to connect to WiFi, at least for a period of time.

Part 2: Common Problems and Their Solutions

This part covers methods of solving common problems that the sensors run into. Typically, these are the underlying problems of errant behaviors seen in Part 1. Note that this part does not cover *every* possible problem that the sensor could encounter, nor does it cover every possible solution for these problems, but it does cover every problem encountered in testing and the recommended solutions.

**Dead Battery**

A dead battery will likely be a problem over time as the batteries degrade, and this problem will likely be more prevalent in sensors that struggle to connect to WiFi, since they spend more time attempting to connect, and thus consume more power. You can confirm that this is the problem by measuring the voltage of the batteries with a digital multimeter. Banksia Gardens owns a digital multimeter capable of doing this, and you can follow [this tutorial](https://learn.sparkfun.com/tutorials/how-to-use-a-multimeter/measuring-voltage) to test the batteries’ voltage. The batteries are advertised to provide 1.2V, and we measured that these batteries, when fully charged, supply 1.37V-1.40V.

There are a few options to solve this problem. If the sensor is consistently struggling to connect to WiFi (this would be apparent in the timestamps on the readings being significantly after the hour, for example 5:12:34 instead of the intended 5:00:10), then fixing this problem will also measurably improve the battery life. Otherwise, lowering the ‘RETRY\_LIMIT’ constant in the code will save battery life for a struggling sensor, but possibly at the cost of consistency.

**Battery Disconnected**

This problem will look very similar to a dead battery, but is much different. If you measure the voltage of the dead battery and it does not indicate that the battery is dead, it is likely that the wires between the battery and the microcontroller were disconnected. To solve this problem, simply reconnect the wires and tape them together. You will know that the battery is connected when the LED on the Pi Pico blinks 10 times. If it successfully blinks 10 times, then the battery is most likely charged enough, and the wires are properly connected.

**Unable to Connect to WiFi**

This is possibly the most persistent problem among some of the sensors, and there is no simple solution. One option is to raise the sensor up a centimeter or two, since elevation tends to make a measurable difference in a sensor’s ability to connect to WiFi. If this does not work, it is possible that the only option is to either relocate the sensor to an area with better WiFi connection, or potentially to boost the WiFi connection in that area through purchasing and installing an extender.

**DHT20 disconnected**

This means that the DHT20 Temperature Sensor Module was disconnected from the board. To solve this simply open the sensor and look for unplugged wires, and then plug them back into the board following the instructions from Section 2. After this, it is ideal if the sensor is tested while plugged into a computer to see the terminal output. If you see that the sensor moves on from initialization and starts ‘jwt signing’, then the sensor is attached properly.

**Software Error**

Although unlikely, it is possible that an error in the software could be encountered as the sensors take their readings. To fix this would require altering the code itself to fix this issue, and then reuploading it to each sensor. This will require programming experience.

**Hardware Failure**

In the event that a piece of hardware fails, the only practical solution is likely to replace the part. If a part continues failing, it may be worth purchasing alternatives described in Section 1.

Sources:

ASAIR. (2021). Interface Definition & Pin Distribution. <https://aqicn.org/air/sensor/spec/asair-dht20.pdf>

Google. (2021). Google Developers text logo [Google Developers logo].

<https://developers.google.com/>

Python. Python-logo-notext. [Python logo]. [www.python.org](http://www.python.org).

Raspberry Pi Foundation. (2020). Raspberry Pi OS Logo [The logo of Raspberry Pi OS].

<https://www.raspberrypi.org/documentation/>

Raspberry Pi Foundation. (2022). Raspberry Pi Pico W Pinout diagram. <https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf>

1. This product comes with pre-soldered headers. For a cheaper alternative, it is possible to purchase the Pico W and male headers separately. However, you will have to solder the headers manually. [↑](#footnote-ref-0)
2. Any header will suffice, as long as the pins are long enough, and are snapped off in a 2x2. You can adjust the direction and position of the pins using pliers. An Australian link was not found, but any electronic/hardware store should be familiar with pins that will suit this project. [↑](#footnote-ref-1)
3. This is available at any electronic store such as JayCar. It is worth noting that a regular charging cable will not work. [↑](#footnote-ref-2)
4. You may use Visual Studio Code and the experimental extension Pymakr as an alternative. [↑](#footnote-ref-3)
5. For users outside of the Banksia organization, please refer to this [GitHub](https://github.com/mannysvb/WPI-Heat-Audit-Public) and follow along the readme. [↑](#footnote-ref-4)
6. For Windows users, you will have to unzip the folder by extracting the files. [↑](#footnote-ref-5)
7. To add new locations to the database it is required to add a new sheet in the “Raw Heat Sensor Temperature Data” and then import it onto the main database. This [clip](https://www.youtube.com/watch?v=-R3MYKRmdAc) shows the steps to add a new location. [↑](#footnote-ref-6)
8. Heat will damage the batteries over-time. [↑](#footnote-ref-7)
9. A plastic container was used to place the Heat Sensor in, and as shown in the video a Snappable Double Row Right Angle PCB Pin Header was poked through the plastic to create an extension of the jumper wires. [↑](#footnote-ref-8)