



Sense HAT data logger

What you will make

In this activity, you will learn how to use the Sense HAT hardware to build a data-logging device which can capture a range of information about its immediate environment.

Once you've created your data logger, you will be able to use it to conduct your own experiments and record data. Things we've tried with our data logger include:

- Dropping it from a four-storey building
- Putting it in a fridge and observing temperature changes
- Sending it to the edge of space with a helium balloon

What you will learn

By creating a Sense HAT datalogger with your Raspberry Pi you will learn:

- How to collect data from multiple sensors and add it to a list structure
- To write and append data to a text file from within a Python program
- To capture and respond to input from the Sense HAT joystick
- About using threads to allow multiple parts of a program to run at once

This resource covers elements from the following strands of the [Raspberry Pi Digital Making Curriculum \(https://www.raspberrypi.org/curriculum/\)](https://www.raspberrypi.org/curriculum/):

- [Apply abstraction and decomposition to solve more complex problems \(https://www.raspberrypi.org/curriculum/programming/developer\)](https://www.raspberrypi.org/curriculum/programming/developer)
- [Process input data to monitor or react to the environment \(https://www.raspberrypi.org/curriculum/physical-computing/developer\)](https://www.raspberrypi.org/curriculum/physical-computing/developer)

What you will need

Hardware

- A Raspberry Pi computer
- A Sense HAT

Software

You will need the [latest version of Raspbian](https://www.raspberrypi.org/downloads/) (<https://www.raspberrypi.org/downloads/>), which already includes the following software packages:

- Python 3
- Sense HAT for Python 3
- Minecraft Pi

If for any reason you need to install a package manually, follow these instructions:

Install a software package on the Raspberry Pi

Your Raspberry Pi will need to be online to install packages. Before installing a package, update and upgrade Raspbian, your Raspberry Pi's operating system.

- Open a terminal window and enter the following commands to do this:



```
sudo apt-get update
sudo apt-get upgrade
```

- Now you can install the packages you'll need by typing `install` commands into the terminal window. For example, here's how to install the Sense HAT software:

```
sudo apt-get install sense-hat
```

Type this command into the terminal to install the Sense HAT package:

```
sudo apt-get install sense-hat
```

Getting data from the Sense HAT

Using the Sense HAT, you can capture the data from the following sensors:

- Temperature sensor
- Humidity sensor
- Pressure sensor
- Orientation sensor
- Acceleration sensor
- Gyroscope
- Magnetic field sensor

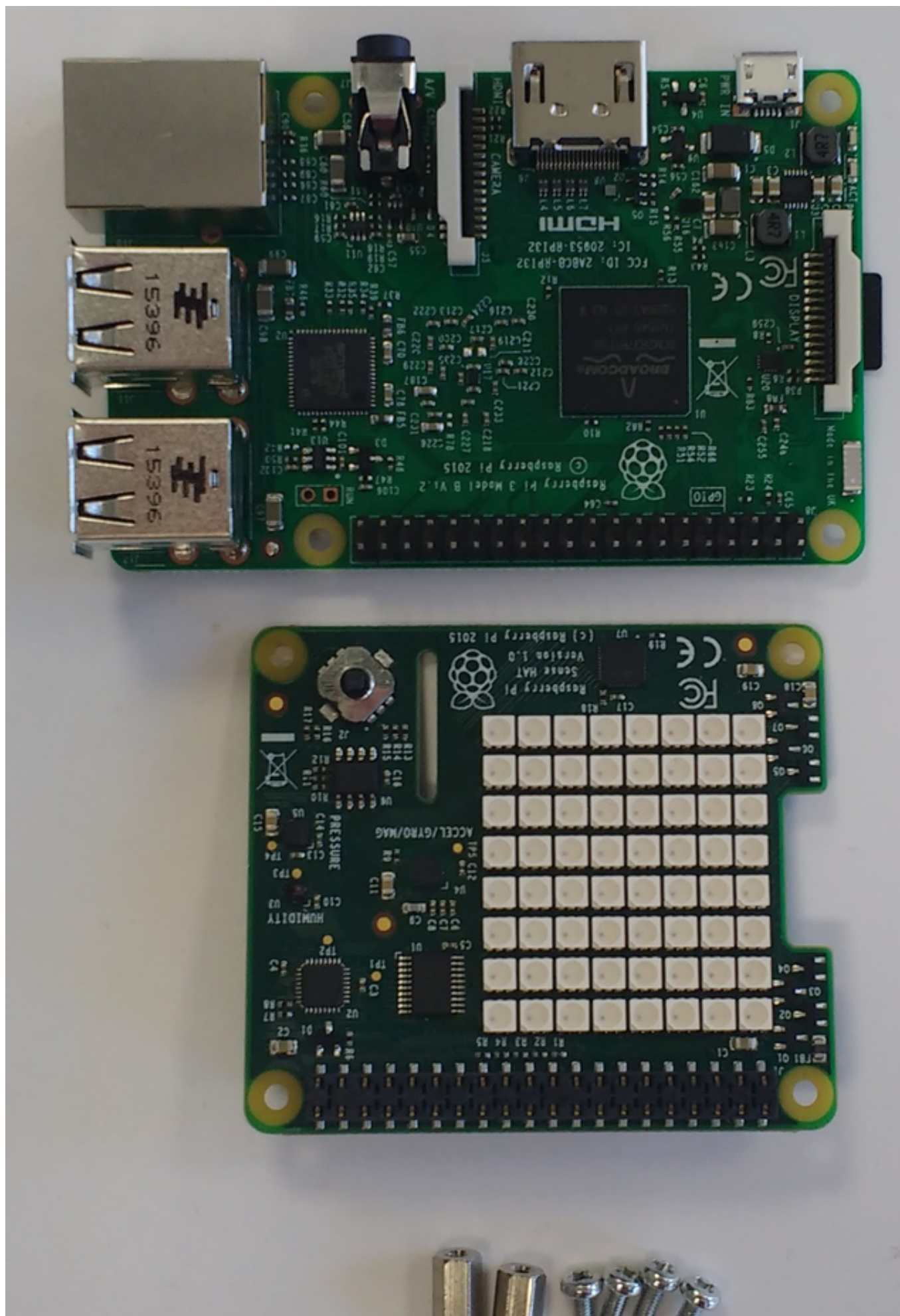
First, write a short script to get readings from the HAT's sensors and output them to the screen.

- Attach your Sense HAT to your Raspberry Pi.

Attaching a Sense HAT

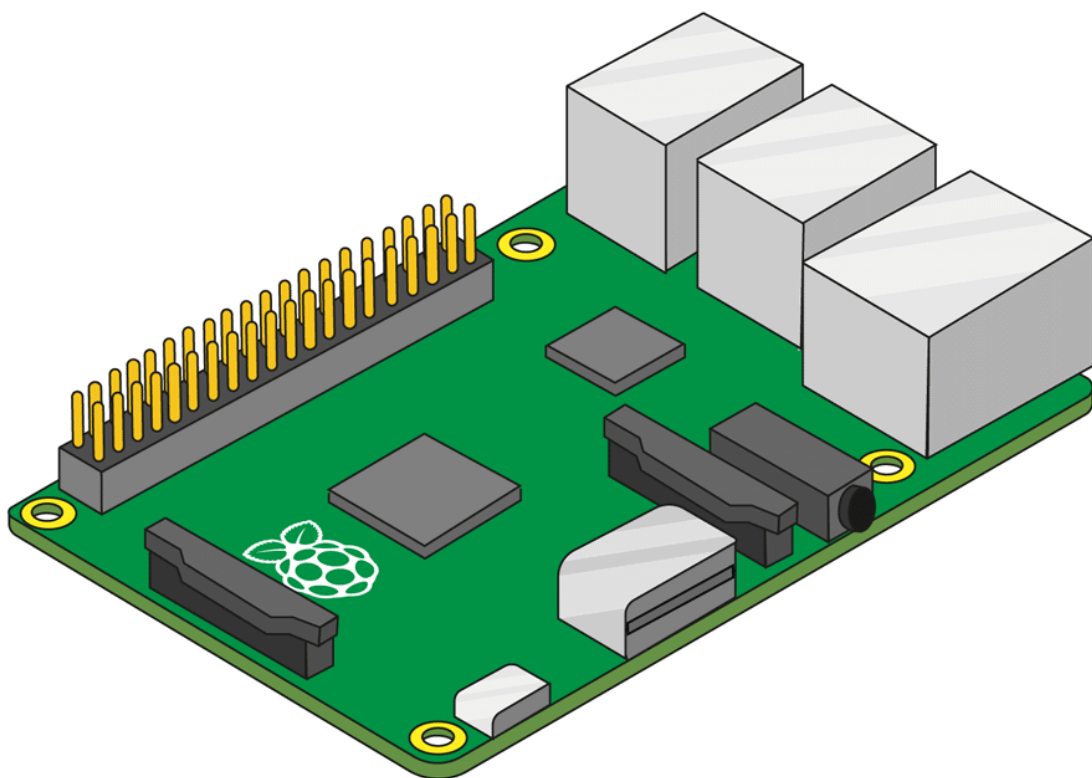
Before attaching any HAT to your Raspberry Pi, ensure that the Pi is shut down.

- Remove the Sense HAT and parts from their packaging.





- Use two of the provided screws to attach the spacers to your Raspberry Pi, as shown below.
- Then push the Sense HAT carefully onto the pins of your Raspberry Pi, and secure it with the remaining screws.



- Once your Sense HAT is attached, boot up your Pi.

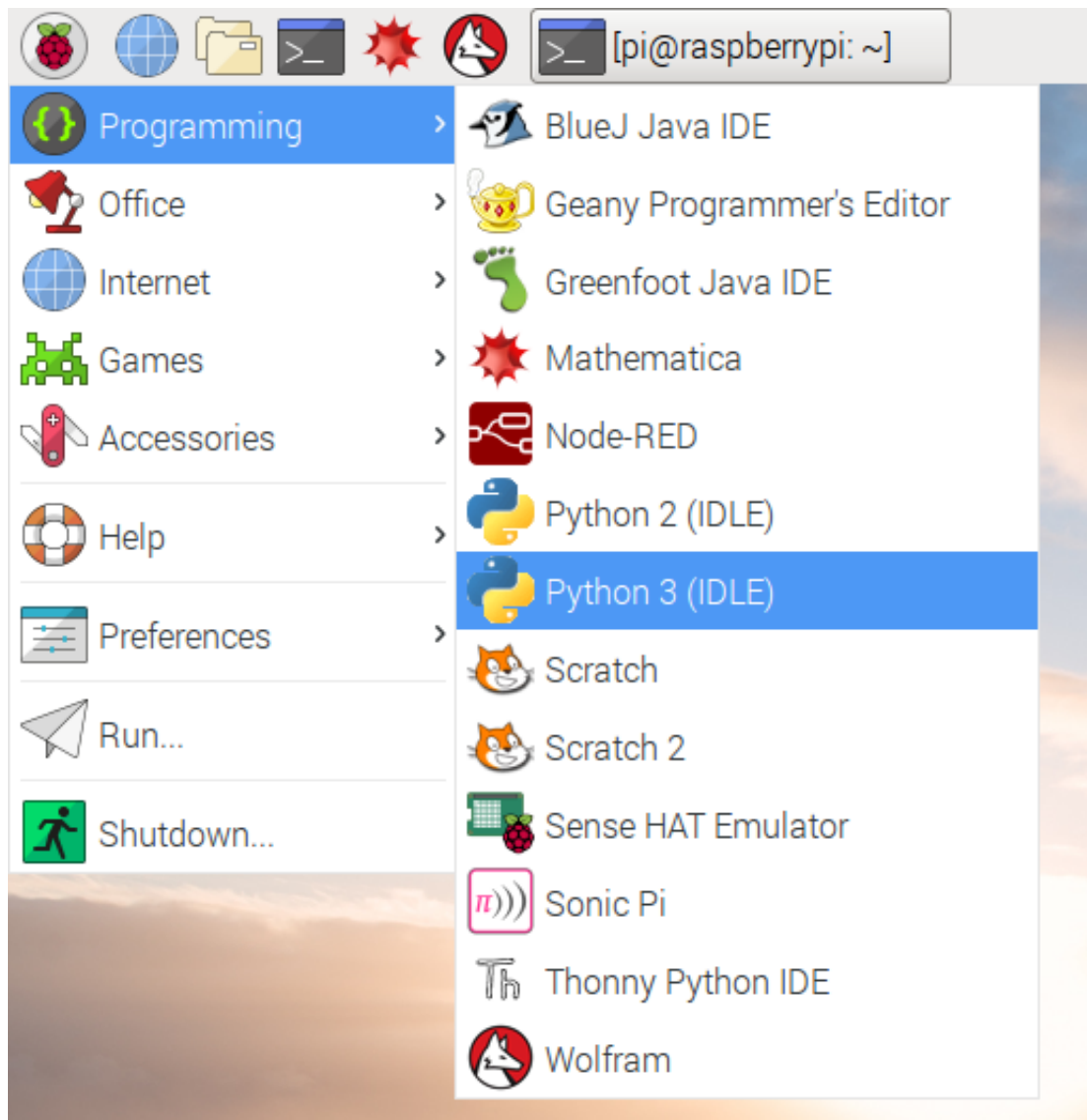
- Open IDLE, and create a new file to work in.

Opening IDLE3

IDLE is Python's **I**ntegrated **D**evelopment **E**nvironment, which you can use to write and run code.

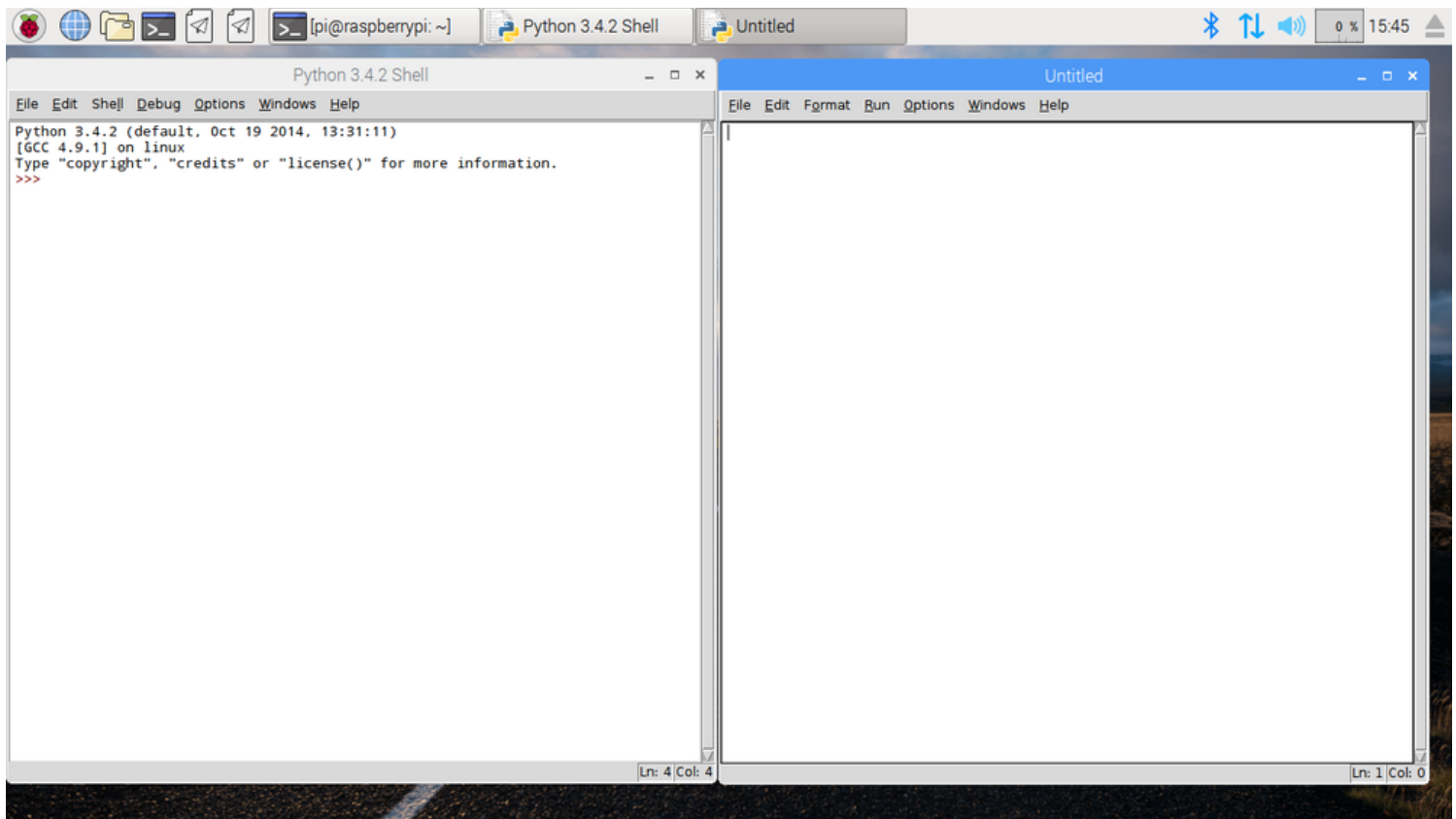
To open IDLE, go to the menu and choose **Programming**.

You should see two versions of IDLE - make sure you click on the one that says **Python 3 (IDLE)**.



To create a new file in IDLE, you can click on **File** and then **New File** in IDLE's menu bar.

This will open a second window in which you can write your code.



- To begin this script, you will need to import the Python modules to control your Sense HAT and to fetch the data and time from the Raspberry Pi. Start by adding these three lines of code:

```
from sense_hat import SenseHat
from datetime import datetime

sense = SenseHat()
```

- Now you're going to create a function which will fetch **all** the sensor data and return it all as a list. Start by defining your function and creating an empty list.

```
def get_sense_data():
    sense_data = []
```

- The section below shows you how to collect the data from the different sensors. In each case you want to append the data to the `sense_data` list. Finish the function by returning the `sense_data` list.

Reading all the sensors

- To read the environmental sensors, you can use the following three commands:

```
sense.get_temperature()  
sense.get_pressure()  
sense.get_humidity()
```

- To read the orientation of the Sense HAT, use the following three lines:

```
orientation = sense.get_orientation()  
orientation["yaw"]  
orientation["pitch"]  
orientation["roll"]
```

- You can get the raw compass readings using the following code:

```
mag = sense.get_compass_raw()  
mag["x"]  
mag["y"]  
mag["z"]
```

- You can get the raw accelerometer reading using the following code:

```
acc = sense.get_accelerometer_raw()  
acc["x"]  
acc["y"]  
acc["z"]
```

- Finally you can get the raw gyroscope readings using the following code:

```
gyro = sense.get_gyroscope_raw()  
gyro["x"]  
gyro["y"]  
gyro["z"]
```

- The only other data that you need is the date and time. To find this out, you can use the following code:

```
datetime.now()
```

Adding data to a list in Python

Python lists are mutable. This means data can be added or removed from them. Let's try this out!

- Start by creating an empty list.

```
my_list = []
```

- By using the keyword **append**, you can add any data you want to the list.

```
my_list.append('A string')
```

- This will produce a list that looks like this:

```
['A string']
```

- Lists can hold data of any type:

```
my_list.append(1)
my_list.append(['another', 'list'])
my_list.append(('a', 'tuple'))
```

- The final result of all these operations would be:

```
['A string', 1, ['another', 'list'], ('a', 'tuple')]
```

I need a hint

To begin with, fetch the environmental sensor readings within your function and add them to the list:

```
def get_sense_data():
    sense_data = []
    sense_data.append(sense.get_temperature())
    sense_data.append(sense.get_pressure())
    sense_data.append(sense.get_humidity())

    return sense_data
```

- You can get the three orientation readings and add them to the list.

```
def get_sense_data():
    sense_data = []
    sense_data.append(sense.get_temperature())
    sense_data.append(sense.get_pressure())
    sense_data.append(sense.get_humidity())

    orientation = sense.get_orientation()
    sense_data.append(orientation["yaw"])
    sense_data.append(orientation["pitch"])
    sense_data.append(orientation["roll"])

    return sense_data
```

- Get the remaining sensor readings along with the data and time by adding the following lines to your function:

```
```python
mag = sense.get_compass_raw()
sense_data.append(mag["x"])
sense_data.append(mag["y"])
sense_data.append(mag["z"])
```

```
acc = sense.get_accelerometer_raw()
sense_data.append(acc["x"])
sense_data.append(acc["y"])
sense_data.append(acc["z"])
```

```
gyro = sense.get_gyroscope_raw()
sense_data.append(gyro["x"])
sense_data.append(gyro["y"])
sense_data.append(gyro["z"])
```

```
sense_data.append(datetime.now())
```
```

- To finish off, you can look at the data by printing out the list within an infinite loop. Add the following to the end of your script, and then save and run the code.

```
while True:
    print(get_sense_data())
```

- You should see a continuous stream of data in the the shell, with each line looking something like this:

```
[26.7224178314209, 25.068750381469727, 53.77205276489258, 1014.18017578125,
3.8002126669234286, 306.1720338870328, 0.3019065275890227, 71.13333892822266,
59.19926834106445, 39.75812911987305, 0.9896639585494995, 0.12468399852514267,
-0.004147999919950962, -0.0013064055237919092, -0.0006561130285263062,
-0.0011542239226400852, datetime.datetime(2015, 9, 23, 11, 53, 9, 267584)]
```

Writing the data to a file

The program you have produced so far is able to continually check the Sense HAT sensors and write this data to the screen. However, unless you're a very fast reader, this is not very helpful.

It would be more useful to write this data to a CSV (comma separated values) file, which you can examine once your logging program has finished. To create this file, you will need to do the following:

- Specify the file name for this file
- Add a header row to the start of the file
- Periodically write a batch of data out to the file

Start by first learning how to write list data to a CSV file in Python, you'll take care of the header later.

Writing data to a CSV file with Python

A **C**omma **S**eparated **V**alues (CSV) file is a useful way to store tabulated data. This type of file is simple for programming languages to read, and it can easily be imported into other applications such as Spreadsheets.

- First, you need to import the `writer` class from the `csv` module.

```
from csv import writer
```

- Next, create a new file, and specify that each line of data is going on a new line.

```
import csv
with open('some.csv', 'w', newline='') as f:
```

- Then set up a writer object to write lines to the CSV file.

```
import csv
with open('some.csv', 'w', newline='') as f:
    writer = csv.writer(f)
```

- Then you can write a line of data. By default, the data needs to be in form of a list.

```
import csv
with open('some.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['Here', 'is', 'some', 'data'])
```

- Now you can alter your code to continuously write the data from your `get_sense_data()` function to a CSV file. Here's one way to proceed:
 - Before your loop starts, open a `csv` file and create your writer
 - Within your loop, write the returned data from the function to the file.

I need a hint

Import the `writer` class, and then open the file and create a `writer` object.

```
from csv import writer ## This line is at the top of your code

## This comes after your get_sense_data() function
with open('data.csv', 'w', newline='') as f:
    data_writer = writer(f)

    while True:
```

Create a variable to hold the data from the function call.

```
with open('data.csv', 'w', newline='') as f:
    data_writer = writer(f)

    while True:
        data = get_sense_data()
```

Now just write that data to the file.

```
with open('data.csv', 'w', newline='') as f:
    data_writer = writer(f)

    while True:
        data = get_sense_data()
        data_writer.writerow(data)
```

Adding a header to the CSV file

You're collecting many different types of data in the CSV file. So that you know which type of data each column contains, it would be useful to add a header row to the CSV file.

To do this you can simply write an additional row to the CSV file before you start the infinite loop.

- Add this line after you create your `writer` object and before the `while True` loop starts:

```
data_writer.writerow(['temp','pres','hum',  
                      'yaw','pitch','roll',  
                      'mag_x','mag_y','mag_z',  
                      'acc_x','acc_y','acc_z',  
                      'gyro_x','gyro_y','gyro_z',  
                      'datetime'])
```

- Make sure the headers are in the same order as the data produced by your `get_sense_data()` function.
-

Recording at specific time intervals

At the moment your script records data as quickly as it possibly can. This is very useful for some experiments, but you may prefer to record data once per second, or even less frequently.

Normally in such situations you would use a `sleep()` function to pause the script. However, this can result in inaccurate readings from some of the Sense HAT's orientation sensors, which need to be regularly polled.

To get around this, you can use `timedelta` to check the time difference between two readings.

Using timedelta in Python

You can use the `datetime` module to calculate the difference between two time points.

- This is easiest to demonstrate in a Python shell rather than using a script.

```
>>> from datetime import datetime  
>>> t1 = datetime.now()  
>>> t2 = datetime.now()  
>>> t2 - t1
```


- Depending on how quickly you could type the `t2 = datetime.now()` command, you should see something like this:

```
datetime.timedelta(0, 6, 843882)
```

- This is a `timedelta` object. It tells you how many days, seconds, and microseconds it took you to type the line starting with `t2`.
- To extract the days, seconds, or microseconds, you can do the following:

```
>>> dt = t2 - t1
>>> dt.days
0
>>> dt.seconds
6
>>> dt.microseconds
843822
```

To use this approach to collect data you would need to do the following:

- At the start of your script, create a `timestamp` variable set to `datetime.now()`
- Decide how long you want the interval between data records to be (in seconds), and create a variable called `delay` to store that number
- Within your infinite loop, if the difference between the `timestamp` and the time of the reading returned by your `get_sense_data()` function is greater than `delay`, write data and reset `timestamp`

- Have a go at adding a one-second delay to your data writing intervals, and use the hints below if you get stuck.

I need a hint

Start by setting the `timestamp` and `delay` variables near the top of your script:

```
timestamp = datetime.now()
delay = 1
```

The time the reading was taken is the **last** item in the list created by your `get_sense_data()` function, so you can now calculate the time difference like this:

```
while True:
    data = get_sense_data()
    dt = data[-1] - timestamp
```

If that `dt` variable is greater than the `delay` you specified, then the data can be written and `timestamp` can be reset.

```
while True:
    data = get_sense_data()
    dt = data[-1] - timestamp
    if dt.seconds > delay:
        data_writer.writerow(data)
        timestamp = datetime.now()
```

- Have a go at trying different values for the `delay` each time you run your script.
 - Can you set a `microsecond` delay?
-

Starting your data logger on boot

This step is completely optional, but you might want to have your script run as soon as the Raspberry Pi boots up. To do this, you can use a **Cron job**. Have a look at the section below to learn how to edit your **crontab** to start scripts on boot.

Automate tasks with Cron

Sometime you don't want to manually start a script that you have written. You may need the script to run once every hour, or maybe once every thirty seconds, or every time your computer starts. On *nix (Computers running a UNIX-like operating system (macOS or GNU/Linux)) systems this is a fairly easy task, because you can use a

program called **Cron**. Cron will run any command you tell it to run, whenever you have scheduled for it to do so. It will reference what is known as the **cron table**, which is normally abbreviated to **crontab**.

Editing the crontab

- To open the crontab, you first need to open a terminal window. Then you can type:

```
crontab -e
```

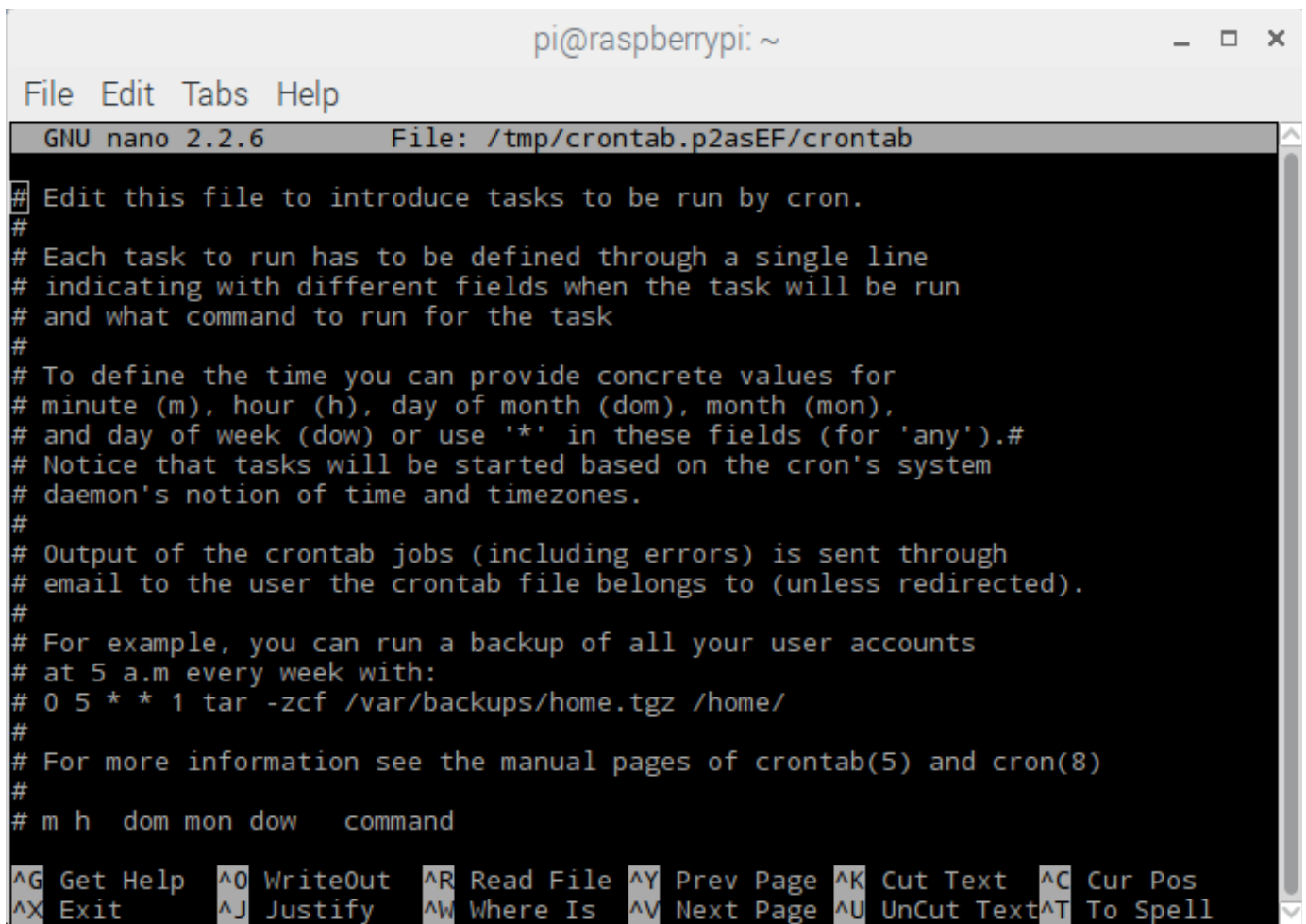
- The `-e` in this command is short for *edit*. If this is your first time opening your crontab, then you'll be asked which text editor (A program for reading and editing text files) you would like to use.

```
pi@raspberrypi:~ $ crontab -e
no crontab for pi - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/ed
 2. /bin/nano          <---- easiest
 3. /usr/bin/vim.basic
 4. /usr/bin/vim.tiny

Choose 1-4 [2]:
```

- Unless you have plenty of experience using **ed** or **vim**, the simplest editor to use is **nano**, so type 2 to choose it and press **Enter**.



```
pi@raspberrypi: ~
File Edit Tabs Help
GNU nano 2.2.6 File: /tmp/crontab.p2asEF/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow command
^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 ^U UnCut Text ^T To Spell
```

- nano is a simple command line (A way of interfacing with your computer using text commands) text editor (A program for reading and editing text files). If you want to learn more about using nano, you can have a look at [this resource \(nix-bash-using-nano\)](#).

Syntax for Cron

The crontab contains all the basic information you need to get started. Each line that starts with a # is a comment, and therefore ignored by the computer. At the bottom of the crontab you should see a line that looks like this:

```
# m h dom mon dow command
```

- m is short for **minute**
- h is short for **hour**
- dom is short for **day of the month**
- mon is short for **month**

- `dow` is short for **day of the week**
- `command` is the bash command that you want to run

Creating a new Cron job

To create a Cron job you need to decide under which circumstances you would like it to run. For instance, if you wanted to run a Python script every 30 minutes, you would write the following:

```
30 * * * * python3 /home/pi/my_cool_script.py
```

The `30` is telling the script to run every 30 minutes. The asterisks indicate that the script needs to run for all **legal values** for the other fields.

Here are a few more examples.

What will happen...

crontab syntax

Run a script at 11:59 every Tuesday

```
59 11 * * 2 python3  
/home/pi/my_script.py
```

Run a script once a week on Monday

```
0 0 * * 1 python3  
/home/pi/my_script.py
```

Run a script at 12:00 on the 1st of Jan
and June

```
0 12 1 1,6 * python3  
/home/pi/my_script.py
```

Run on boot

One incredibly useful feature of Cron is its ability to run a command when the computer boots up. To do this, you use the `@reboot` syntax. For instance:

```
@reboot python3 /home/pi/my_cool_script.py
```

Edit and save the file

You can add in your cron job to the bottom of the crontab. Then save and exit nano by pressing `Ctrl+x` and then typing in 'y' when you are prompted to save.

Challenge: selecting the data to be recorded

You might not always want to record all the sensor data. One solution to this is to simply comment out the lines you don't need in your `get_sense_data()` function.

Another solution would be to use **conditional selection**.

- Can you set up your script so that you pass into your `get_sense_data` function the sensors you want to use, and these are then the only ones that are recorded? Don't forget to add a method to alter the header row of your CSV file as well.
-

Published by the Raspberry Pi Foundation – www.raspberrypi.org

Licensed under Creative Commons "*Attribution-ShareAlike 4.0 International* (CC BY-SA 4.0)"
Full project source code available at <https://github.com/RaspberryPiLearning/sense-hat-data-logger>