Temperature Log

What you will make

The system on a chip (SoC) of the Raspberry Pi has a temperature sensor that can be used to measure its temperature from the command line. It can provide information on how much heat the chip has generated during operation and also report on the temperature of the environment. This project's aim is to create a simple Python script that can run automatically as you boot up your Raspberry Pi, take measurements from the temperature sensor at given intervals, and write them into log files that can be viewed later. You'll also be able to view the data as an interactively plotted graph.

What you will learn

By completing the Temperature project you will learn:

- How to run system commands in Python
- How to write data to a file
- How to interactively plot data with matplotlib
- How to set scripts to run automatically using crontab

This resource covers elements from the following strands of the Raspberry Pi Digital Making Curriculum:

What you will need

Software

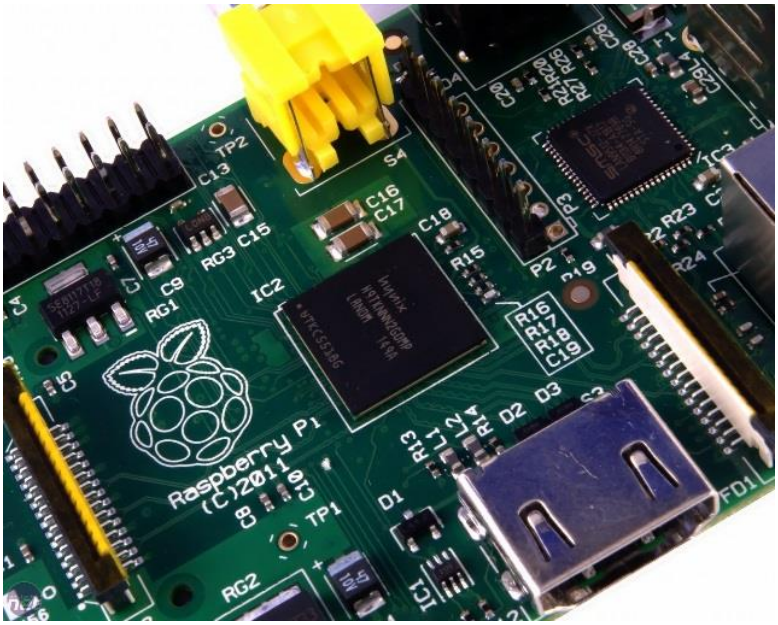
Software installation

To install the software you need, run the following command in the terminal:

```
sudo apt-get install python3-matplotlib
```

Temperature log

The system on a chip (SoC) of the Raspberry Pi has a temperature sensor that can be used to measure its temperature from the command line. It can provide information on how much heat the chip has generated during operation, and can also report on the temperature of the environment. This project's aim is to create a simple script that can run automatically as you boot up your Raspberry Pi, take measurements from the temperature sensor at given intervals, and write them into log files that can be viewed later.



Creating a Python script to monitor temperature

- Open a new Python 3 shell by going to **Menu** > **Programming** > **Python 3 (IDLE)**.
- Now create a new Python script by clicking on **File** > **New File**.
- You can use the GPIO Zero module to find the CPU temperature. First you'll need to import the `CPUTemperature` class:
- `from gpiozero import CPUTemperature`
- Then you can create a `cpu` object:
- `cpu = CPUTemperature()`
- Save and run this program (press **Ctrl + S** and then **F5**) and then swap over into the shell. Here, you can easily query the CPU temperature.

```
>>> cpu.temperature
32.552
```

Writing the data to a CSV file

It would be useful if that data could be stored somewhere. A CSV (comma-separated values) file is ideal for this, as it can be used by applications like Excel and LibreOffice.

- You'll want to log the date and time while getting the CPU temperatures, so you'll need some extra libraries for this. Add this to your imports:

```
from time import sleep, strftime, time
```

> These extra methods let you pause your program (`sleep`), get today's date as a string (`strftime`), and get the exact time in what's known as UNIX time (`time`).

To write to a file, you first need to create it. At the end of your file, add the following line:

```
with open("cpu_temp.csv", "a") as log:
```

> This creates a new file called `cpu_temp.csv` and opens it with the name `log`. It also opens it in **append** mode, so that lines are only written to the end of the file.

Now, you'll need to start an infinite loop that will run until you kill the program with **Ctrl + C**:

```
with open("cpu_temp.csv", "a") as log:
    while True:
```

Inside the loop, you can get the temperature and store it as a variable.

```
    with open("cpu_temp.csv", "a") as log:
        while True:
            temp = cpu.temperature
```

Now you want to write both the current date and time, plus the temperature, to the CSV file:

```
    with open("cpu_temp.csv", "a") as log:
        while True:
            temp = cpu.temperature
            log.write("{0},{1}\n".format(strftime("%Y-%m-%d
%H:%M:%S"),str(temp)))
```

That line's a little complicated, so let's break it down a bit:

> `log.write()` will write whatever string is in the brackets to the CSV file.
>
> `"{0},{1}\n"` is a string containing two placeholders separated by a comma, and ending in a new line.
>
> `strftime("%Y-%m-%d %H:%M:%S")` is inserted into the first placeholder. It's the current date and time as a string.
>
> `str(temp)` is the CPU temperature converted to a string, which is written into the second placeholder after the comma.

Lastly, you can add a single line to the end of your file to pause the script between writes. Here it's pausing for one second, but you can use any interval that you want:

```
    sleep(1)
```

The entire script should now look like this:

```
from gpiozero import CPUTemperature
from time import sleep, strftime, time

with open("cpu temp.csv", "a") as log:
    while True:
        temp = cpu.temperature
        log.write("{0},{1}\n".format(strftime("%Y-%m-%d
%H:%M:%S"),str(temp)))
        sleep(1)
```

Live-graphing the data

You can produce a graph of CPU temperatures which will update as the data is recorded. For this, you'll need the **matplotlib** library. The instructions for installing this are here.

First of all, import the matplotlib library where your other imports are:

```
import matplotlib.pyplot as plt
```

The next three lines can go after your imports. They tell matplotlib that you'll be doing interactive plotting, and also create the two lists that will hold the data to be plotted:

```
plt.ion()
x = []
y = []
```

The next lines all go into your `while True` loop, before the CSV is written, but after the `temp = get_temp()` line. Firstly, you add the current temperature to the end of the `y` list, and the time to the end of the `x` list:

```
y.append(temp)
x.append(time())
```

Next, the plot needs to be cleared, and then the points and lines calculated:

```
plt.clf()
plt.scatter(x,y)
plt.plot(x,y)
```

Lastly, the plot can be drawn:

```
plt.draw()
```

Run your program and you should see the graph being interactively drawn. Open up some programs, such as Minecraft or Mathematica, and watch the CPU temperature increase.

Automating the script

It might be useful to have this script running when the Raspberry Pi starts up. To do this, it's best to clean up the script a little, so that you can easily comment out the lines that draw the graph. Below is the same script tidied into functions, and with the graph-drawing line commented out:

```
from gpiozero import CPUTemperature
from time import sleep, strftime, time
import matplotlib.pyplot as plt

cpu = CPUTemperature()

plt.ion()
x = []
y = []

def write_temp(temp):
    with open("cpu_temp.csv", "a") as log:
        log.write("{0},{1}\n".format(strftime("%Y-%m-%d
%H:%M:%S"),str(temp)))

def graph(temp):
    y.append(temp)
    x.append(time())
    plt.clf()
    plt.scatter(x,y)
    plt.plot(x,y)
    plt.draw()

while True:
    temp = cpu.temperature
    write_temp(temp)
#    graph(temp)
    sleep(1)
```

- Automating scripts is simple with **crontab**. This is basically a file where commands can be placed that will run at certain times or after certain events. To begin, open up a terminal window (press **Ctrl + Alt + T**).
- To edit the crontab, you just type:

```
crontab -e
```

- Scroll to the bottom of the file and add this single line:
- `@reboot python3.4 /home/pi/temp_monitor.py`

  This assumes that your script is called `temp_monitor.py` and that it's saved in your home directory.

- Now reboot your Raspberry Pi. Give it a little time to run, then type the following in a terminal window:

```
cat cpu_temp.csv
```

  This will enable you to see the contents of the CSV file.

- If you want to see a graph, then just uncomment the `graph(temp)` line using IDLE and run the file.

What next?

- Why not have a look at the Getting Started with the Twitter API resource, and have your Raspberry Pi tweet you when the CPU temperature gets too high?
- There are other ways of sensing temperature, and a host of other environmental variables. Have a look at Getting started with the Sense HAT for some more ideas.