

# **SYSC 3010 Systems Development Lab**

## **Lab 2: IoT and Hardware**

Please, do the BACKGROUND READING before your lab period. The work cannot be completed within the single lab period. Use the lab period for technical support.

Part 1 and Part 2 can be done in any order.

### **LAB OBJECTIVES**

1. Each individual can demonstrate the use of an IoT cloud service, with a program that they wrote to read and write to their own ThingSpeak channel from their RPi.
  - Optionally (and depending on availability) students are invited to complete the tutorial to explore the potential use of Thingful.
2. Each individual can manipulate JSON data within a Python program.
3. Each individual either has the ability to program hardware connected to their RPi and/or knows the expectations and procedures for simulating a hardware device for this course.

### **SYNCHRONOUS VERSUS ASYNCHRONOUS**

The TAs will be online for the entire lab period. Instructions will be posted at the time. If you have trouble connecting, please use SLACK to initiate contact.

There will be no signup for this lab. Please contact any of TAs as/when you need them for technical support.

### **SUBMISSION**

There will be no marked submission at this time. During the individual lab exam (Lab 4), a student must be able to perform any task involved in this lab. Additionally, the student's Github repository will be checked to ensure that (1) the content for this lab is present and (2) the timestamp for that content falls within this lab period.

All programs for this lab must be in a GitHub folder called **Lab-2**.

## BACKGROUND READING – TO BE COMPLETED BEFORE THE LAB

To begin, please read: [CODE magazine – Introduction to IoT Using the Raspberry Pi](#).

This introduction to IoT is fantastic because it includes a detailed pictorial introduction to **connecting hardware devices to the GPIO pins of a RPi**, as well as the use of a cloud service.

Secondly, the following is an excellent advanced example that uses a Raspberry PI with a Sense HAT board, to save files, to send emails, and to send SMS notifications via InstaPush. It is one hour long, but I guarantee you that you will learn lots about what you can do by watching this video. It is my hope that each individual can do all these tasks by the lab exam in Lab 4.

Video – IoT Tutorial for Beginners - <https://www.youtube.com/watch?v=UrwbeOllc68>

- Skip the beginning and start watching at 24:58 (IoT Demo)
- Minimally, watch the assembly and use of the Sense HAT board (until 34:24)
- Ideally, watch the rest of it before the lab exam.
- [Appendix A](#) contains bug fixes that I used when I followed along with video. These kinds of errors always crop up as software versions change or based on your network settings in your home environment.

## GENERAL CAUTIONS AND LESSONS ABOUT PYTHON

As we explore the many sample programs available online, there are two common stumbling blocks that will make you want to cry and perhaps give up.

1. New Python libraries will need to be installed. When you install Python on a machine, common default libraries are automatically installed. However, you have to manually install lesser used libraries. Installation is done by typing a command, either in a shell window (CMD window, on Windows) or in the shell window within an IDE.

For example, you might see this error when you try to run your Python program  
`ModuleNotFoundError: No module named httpplib.`

Type: `sudo apt-get install httpplib`

2. It is imperative that you pay attention to the version of Python that the sample programs use: **Python 2** or **Python 3**. In many cases, the imported libraries for these two versions are vastly different – with different module names and with different function names.

For example, you might see these error when you try to run your Python program  
`ModuleNotFoundError: No module named urllib.`

But you say to yourself: “I did install `urllib`. What’s the problem?”

There is actually `urllib`, `urllib2`, `urllib3` and `requests`. Four different libraries that you have to scour to come up with the right solution for your version of Python.

**Question:** How do I find out which version of Python I'm using?

- Here's a link with the common answers: <https://phoenixnap.com/kb/check-python-version>
- However, please note: The methods shown here reveal the version of Python used in the command-line (i.e. in a shell window or using the CMD-prompt window). If you instead run your programs from within an IDE (e.g. Thonny on the RPi, or PyCharm on your laptop), you have to check WITHIN THAT IDE what version it uses.

**Question:** What if my computer uses Python 3, and the sample program uses Python 2?

- You have to find the name of the new library, install that library, and possibly make changes to the code if the library now have new names for the same old functions.
- Example: [Here is a tutorial on encoding URLs in both Python 2 and 3](#). See the difference between the two versions.
- Or, you can look for another example that uses the same version of Python as you are using!

## PART 1- HARDWARE (EMULATOR)

Ideally each student has a least one hardware device attached to their embedded computer (i.e. their RPi). This term, I anticipate a lot of variation between students:

- Hopefully, many of you have the Sense HAT.
- Some of you may have bought a single device, like a temperature sensor.
- Others may not have been able to buy anything (yet).

Because of this wide variation, it is difficult to write a lab with any firm instructions. Instead, I will simply say that **you should use this lab to demonstrate that you can program whatever hardware that you have available and/or use a hardware emulator.**

Reading: [A fun practical introduction to the Sense HAT in the MagPi Essentials](#).

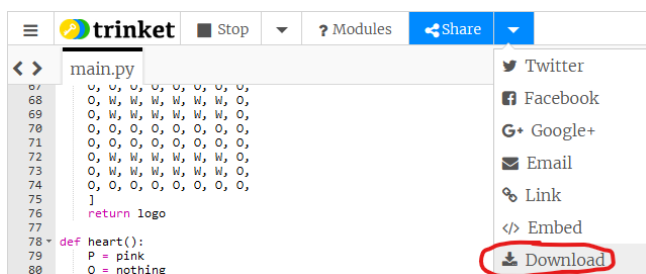
**Step 1 – Everyone** (even those who do not have a Sense HAT or even a RPi), play with the code at this website. The website contains an *emulator*.

**Code the Sense HAT in Your Browser :** <https://trinket.io/sense-hat>

- Why? The website shows you how to program a SenseHAT in Python.
- Why? The website shows you what an *emulator* is: A software program that simulates a piece of hardware.
  - The rule in this course is: If you are unable to buy or have not yet received your hardware, it is your responsibility to write a program to emulate your intended device. The program does not have to be as fancy as this Sense HAT emulator, but it must demonstrate the key functionality of the hardware it is simulating.
  - Simulators (or emulators) are widely used in standard industry practices, both to allow software development to commence before the hardware is available, and for testing purposes.

- Every student will have “hardware” – if you can’t buy it, then you will program it.

1. Study the code so that you can figure out how to change the program. For instance, make the lights turn blue instead of red.
2. Scroll to the bottom of the webpage because there are more sample programs there. For instance, **sensors.py** will show you how to read the temperature.
3. When you are ready, **write your own program** that uses the arrow keys to toggle the display between the initial of your first name and the initial of your surname. For instance, for me, my initials are “CS”. In my program, anytime I hit ANY arrow key, the display shows a C, then a S, then a C, then a S and so on.
4. Download your program (shown in the screenshot below) and name your file **lab2-hardware-step1.py**. Add this file to your GitHub in the lab2 folder (by the end of this lab)



**Step 2** – For everyone who has a RPi but who does not have a Sense HAT: There is a Sense HAT Emulator that is included in the Raspbian operation system.

Please follow this tutorial for working with the Sense HAT Emulator:

<https://www.littlebird.com.au/a/how-to/28/use-the-sense-hat-emulator-in-raspbian>. You may re-use the program you wrote in Step 1 or use any of the sample programs at this site.

If you are using a physical Sense HAT: `import sense_hat`

If you are using an emulated Sense HAT: `import sense_emu`

**Step 3** – For everyone: Write a program that demonstrates the use of your particular hardware.

- If you don’t have any hardware, then you must write an emulator, and a program that uses that emulator. Suggestions are provided in [Appendix B](#). If these suggestions don’t seem to apply to you, it’s time to contact the instructor.
- Call your program **lab2-hardware-step3.py** and make sure it is included in your Github
- At the top of your program, include a header that describes the hardware that you are using
  - If you are copying code from an online tutorial, be sure to give attribution to avoid plagiarism.
- For those with a Sense HAT, try out this tutorial
  - <https://projects.raspberrypi.org/en/projects/sense-hat-random-sparkles/1>

## PART 2 – IOT CLOUD SERVICES - THINGSPEAK

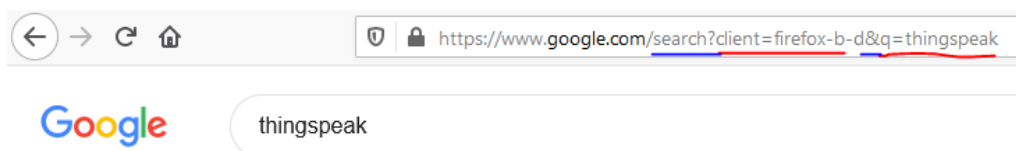
*Aside: If it is more convenient, you may start learning by building Python programs on your laptop or desktop. **In the end though, you must be able to run these programs on your RPi.** If you take this route, beware of version differences in your Python environments.*

IoT stands for Internet-of-Things. Any “thing” that is internet-enabled can send data and receive commands. The intention in this course is to use our embedded computer RPis as the “thing”.

In the fall 2020, we will rely on the IoT cloud service called [ThingSpeak](#) to serve as our communication backbone for an IoT application of your choice. **Your task in this lab is to ensure that you individually know how to both read and write data from a Thingspeak channel.** For this lab (and the lab exam), you must demonstrate your ability on a free private channel that you create within your own ThingSpeak account. Later in your project, we will share channels within and between teams.

### Background Information and Some Cautionary Warnings

1. ThingSpeak includes MATLAB analytics to make fancy graphs. You are welcome to explore MATLAB but you are not required to do so.
  - For experienced students only: Read this tutorial to learn about the potential of Analytics: <https://www.mathworks.com/matlabcentral/fileexchange/52456-analyzing-traffic-using-a-webcam-a-raspberry-pi-and-thingspeak>
2. You will need a [MathWorks](#) account, using your Carleton credentials (Do NOT use the cmail version). MathWorks will get you access to MatLab and SimuLink, but you do not have to download any of that software. **You only need the account itself to access ThingSpeak.** We will try to post more detailed subscription instructions at the time of the lab. We need to work out these details together.
3. There are two API protocols for writing programs with ThingSpeak channels: **REST** and MQTT. In brief, REST is simple and is based on the HTTP request-reply protocol used by our web browsers; **we will use REST in this lab.** MQTT is a lighter weight protocol that is widely used in IoT; if you are interested in learning more about IoT, please feel free to look at MCTT during the term (A good topic perhaps for your technical memo).
  - For experienced students only: Read about REST versus MQTT: <https://www.mathworks.com/help/thingspeak/choose-between-rest-and-mqtt.html>
4. You need a basic understanding of the HTTP protocol used by REST. Have you heard about HTTP requests and replies? To begin, consider the Google search that I started in my Firefox browser.



In the above screenshot, I have typed “thingspeak” into the Google search bar. When I hit return, the URL shown is the **HTTP request**: <https://www.google.com/search?client=firefox-b-d&q=thingspeak>

- *www.google.com* is the server to which the request will be sent.
- *search?* tells the server that I want to do a search
- *client=firefox-b-d&q=thingspeak* tells what I want to search ( Notice the & between the criteria)
- Try doing a search yourself and study the format of the HTTP request

The google server will perform the search and return a **HTTP reply**. We don’t see the HTTP reply directly; instead my Firefox browsers *renders* (translates and displays) into a HTML list of options.

When you **write** to a ThingSpeak channel, you will similarly format and send a **HTTP Request** (to write data)

When you **read** from a ThingSpeak channel, you will format and send a **HTTP Request**, and then wait to receive the **HTTP reply** which you will have to then translate into data.

5. On the MathWorks ThingSpeak site, there are many examples. It is wonderful and you can get a lot of ideas from browsing them, including hardware schematics for fun sensors (Example: <https://www.mathworks.com/help/thingspeak/MoistureMonitor.html>). Be careful though. Some are written in Matlab, some in Arduino code. The only ones written in Python (at this date) are:
  - o [Bulk-Update Using a Raspberry Pi Board](#) – Highly recommended later to handle complex data, but do it later, after you’ve handled simple data.
  - o [Publishing multiple sensor values to a channel feed](#) – Highly recommended later, but it uses MQTT, so learn this later in the project.

Simple REST Python examples are more often found in the Raspberry Pi sites.

**Step 1:** Write a Python program to write data to a ThingSpeak channel.

Follow this tutorial: [How to Send Data to ThingSpeak Cloud using Raspberry Pi](#).

If you don’t have a RPi, you can do the same program on your laptop – Simply replace the line of code that reads the RPi’s CPU temperature with some code that randomly generates a number.

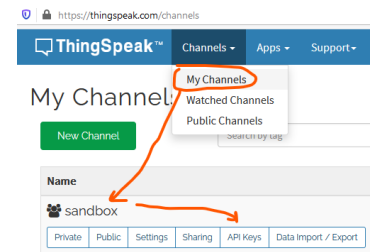
Call your program **lab2-thingspeak-step1.py** and make sure it is included in your Github

**Step 2:** Read the data from your ThingSpeak channel, using your browser as a tool for learning JSON.

It’s now time for your crash course on JSON.

Before we actually write a program to read from a ThingSpeak channel, let’s first do it in a browser, so that you can understand (1) the format of the required HTTP request and the returned HTTP reply and (2) the JSON formatting underlying the HTTP reply.

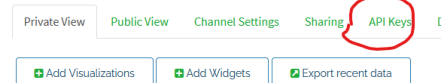
Login into your ThingSpeak account. Find the channel that you just created. In my example below, my channel is called Sandbox.



If you click on the tab called “Private” (because my channel is private, perhaps yours is public?), you can see the data that has been written, and importantly, take note of the **channel ID**. My channel ID is 1073314.

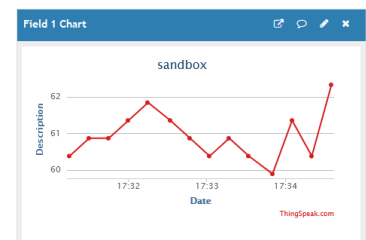
## sandbox

Channel ID: **1073314** | A place for trying out tests  
Author: [redacted]  
Access: **Private**



## Channel Stats

Created: 2 months ago  
Last entry: 2 months ago  
Entries: 14



You’ve already seen the use of the Write API key in the previous step. You will now use the Read API key. Before writing a Python program, we will play with the API Requests at the bottom right.

The API requests show the format of the HTTP request required to perform a read or write. Copy the entire URL for **Read a Channel Feed** (Make sure to slide over to the end). Paste the URL into your browser and hit RETURN (sending the HTTP request to ThingSpeak).

Later, you can also experiment with **Read a Channel Field**. It will read one particular field in a channel feed.

**Write API Key**

Key: [redacted]

Generate New Write API Key

**Read API Keys**

Key: [redacted]

Note: [redacted]

Save Note Delete API Key

Add New Read API Key

**API Keys Settings**

- **Write API Key:** Use this key to write data to a channel. If you feel your key has been compromised, click **Generate New Write API Key**.
- **Read API Keys:** Use this key to allow other people to view your private channel feeds and charts. Click **Generate New Read API Key** to generate an additional read key for the channel.
- **Note:** Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

**API Requests**

**Write a Channel Feed**

GET https://api.thingspeak.com/update?api\_key=[redacted]&field1=[redacted]

**Read a Channel Feed**

GET https://api.thingspeak.com/channels/1073314/feeds.json?api\_key=[redacted]

**Read a Channel Field**

GET https://api.thingspeak.com/channels/1073314/fields/1.json?api\_key=[redacted]

**Read Channel Status Updates**

GET https://api.thingspeak.com/channels/1073314/status.json?api\_key=[redacted]

Learn More

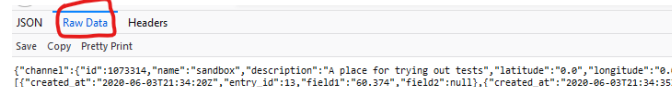
My URL was [GET https://api.thingspeak.com/channels/1073314/feeds.json?api\\_key=MY\\_API\\_KEY&results=2](https://api.thingspeak.com/channels/1073314/feeds.json?api_key=MY_API_KEY&results=2)

(I have hidden MY\_API\_KEY)

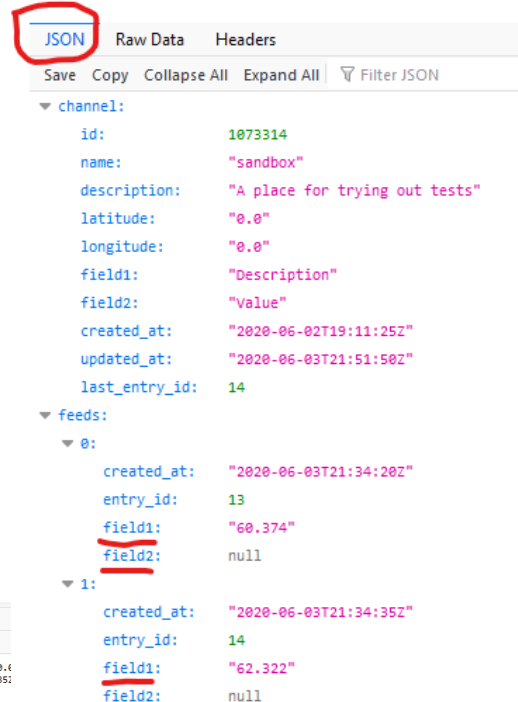
The final parameter in the URL (results=2) means that the two most recent readings are read. You can change this number to collect more data.

The data returned in my HTTP reply is on the right. Notice on the top menu that there are two tabs: JSON and Raw Data. The JSON tab displays the data in a very verbose but readable format.

Below is the same information but displayed as Raw Data. It is the same data but its display is terse.



```
{
  "channel": {
    "id": 1073314,
    "name": "sandbox",
    "description": "A place for trying out tests",
    "latitude": "0.0",
    "longitude": "0.0",
    "field1": "Description",
    "field2": "Value",
    "created_at": "2020-06-02T19:11:25Z",
    "updated_at": "2020-06-03T21:51:50Z",
    "last_entry_id": 14
  },
  "feeds": [
    {
      "created_at": "2020-06-03T21:34:20Z",
      "entry_id": 13,
      "field1": "60.374",
      "field2": null
    },
    {
      "created_at": "2020-06-03T21:34:35Z",
      "entry_id": 14,
      "field1": "62.322",
      "field2": null
    }
  ]
}
```



```
{
  "channel": {
    "id": 1073314,
    "name": "sandbox",
    "description": "A place for trying out tests",
    "latitude": "0.0",
    "longitude": "0.0",
    "field1": "Description",
    "field2": "Value",
    "created_at": "2020-06-02T19:11:25Z",
    "updated_at": "2020-06-03T21:51:50Z",
    "last_entry_id": 14
  },
  "feeds": [
    {
      "created_at": "2020-06-03T21:34:20Z",
      "entry_id": 13,
      "field1": "60.374",
      "field2": null
    },
    {
      "created_at": "2020-06-03T21:34:35Z",
      "entry_id": 14,
      "field1": "62.322",
      "field2": null
    }
  ]
}
```

JSON is a way of organizing data. It is based on **{name-value}** pairs. These name/value pairs can be structured, with nested pairs of **{}**. Notice how **field1** is nested inside of **0**, which is nested inside for **feeds**. Remember that my HTTP request requested 2 results. That's why there are two feeds. A programming way of talking about this is to imagine an array of structures. `feeds[0].field1 = 60.374` and `feeds[1].field2 = 62.322`.

The key thing to learn in this JSON crash course is that in order to parse a single value (e.g. the field1 value), you have to provide the full path of each nested name in order to get to that value. For instance, notice that `channels.field1 = "Description"` and `feeds:0:field1 = "60.374"`.

If you understand this superficial introduction, you are ready for the next tutorial video that will teach you how to repeat this same process in a Python program.

**Step 3:** Write a Python program to read JSON data from a ThingSpeak channel.

Follow this tutorial by Soumil : <https://www.youtube.com/watch?v=whXaVYSXItQ>

I love this guy, despite his spelling mistakes in his variable names! Let's hire him. Perhaps you want to present your technical memos as a video?

Call your program **lab2-thingspeak-step3.py** and make sure it is included in your Github



## PART 3- THINGFUL (OPTIONAL)

Check out this other IoT cloud service: <https://www.thingful.net/>

Thingful is just getting up to speed in hosting educational ventures. I have tentative agreement for you to use their services. Give the tutorial a try, and if you're interested in using this for your project, talk to me. At the very least, watch the video because it might give you some ideas for your project.

Can you imagine? **You can instantly find out where all the shared bicycles are in Miami!**

The following instructions were copied from an email sent to me:

*Briefly, Datapipes enable you to find, access and pipe real-time IoT and sensor data from around the world to wherever you need it - google doc, email, http URL (for integration in your own apps), eventually your own database or S3 bucket if necessary.*

*If you are interested, you can access the datapipes prototype here: <https://datapipes.thingful.net/> where you can also register and setup an account immediately. Watch a video tutorial here: <http://bit.ly/datapipes-walkthrough>*

*You might try out searches like:*

- 'air quality' in 'london'
- 'weather' in 'nigeria'
- 'bikes' in 'miami'
- 'parking' in 'berlin'
- 'webcam' in 'finland'
- 'noise' in 'japan'
- 'bus' in 'new york'
- 'cars' in 'vienna'
- 'wind' in 'texas'
- 'cloud' in 'italy'

## APPENDIX A – BUG FIXES FOR IOT TUTORIAL FOR BEGINNERS

IoT Tutorial for Beginners: <https://www.youtube.com/watch?v=UrwbeOllc68>

Following are the list of changes that I need to make in my home environment to make this tutorial work:

- The `email` package was refactored at some point and the Mime support is now in the `email.mime` package.

See: <https://docs.python.org/2/library/email.html>

See: <https://stackoverflow.com/questions/36967766/python-unresolved-reference-import-mimemultipart>

- My email failed with the message: `Network is unreachable`

Solution for my machine: The port was wrong. He says to use 25. The port is 587 (from <https://www.bc-robotics.com/tutorials/sending-email-using-python-raspberry-pi/>)

- If you are using gmail, you may have to turn (temporarily) on “Access to less secure apps”.
  - See: <https://support.google.com/accounts/answer/6010255?hl=en>

## APPENDIX B – WRITING YOUR OWN EMULATOR

Writing an emulator sounds more daunting than it is. You don’t have to make a fancy GUI driven emulator (although this is possible in your final project if you need some depth to demonstrate your programming proficiency)

Suppose you want to emulate a flip-switch. You simply need a program that prompts-and-waits for user input. Whenever the user hits the S key, for instance, your program will consider that a switch has been flipped.

Suppose you want to emulate a thermometer. You simply need a program that either generates a random number within a valid temperature range every time you request a reading, or perhaps returns a linearly increasing number if you want to simulate a temperature increase.

Hopefully, both of these tasks now seem reasonable and achievable. There is however one key software criterion to be met: Good software engineering design must be used. The emulator must be built as a module (perhaps a class to be imported). The emulator must be separate from the other python code that makes use of the emulator.

- The idea is to represent this hardware as separate from the software that “uses” this hardware.
- In this way, you could instead have 5 switches and 10 thermometers in your system.
- Or perhaps, you can let another group use your emulator.

For your submission, you will have two files: one is the emulator that you write, named appropriately; and the second is the same file that everyone else has, called **lab2-hardware-step3.py**.