Tutorial 1

Due Monday 11th March

Aim

This tutorial will introduce you to our virtual environment and show you how to use the emulator to gather sensor data and write python code to affect the screen on the device.

Complete this tutorial by working running the code contained below. There are a series of challenges that you should attempt to complete. At the end of the tutorial, complete the checklist and upload it to the relevant weeks drobox on Mylo.

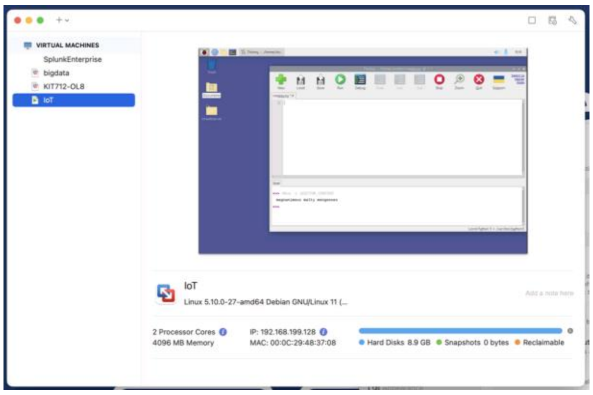
Getting Started

Accessing the virtual machine

This semester, we are going to be working in a virtual machine (VM) environment to complete our tutorials and assignments. The VM is a raspberry pi image that will simulate both an embedded computer device (or IoT hardware), as well as our web server. The virtual machine that can be accessed in two ways.

On campus in a mac lab

For tutorials, you will be working on campus in a mac lab. Rather than login with your standard credentials, you should login in with the username and password 'bigdata'. This account will have the VM available for all of your tutorial work. When you login, VMware fusion will start. From there, you can select the IoT VM (it should be at the bottom of the list. Click to start the VM.



*On occasion, the VM might not initial in a timely fashion. If this happens, then please close the VM and reopen it.

Off campus (or in a different computer lab)

When you are working on your assignments, or finishing tutorial work outside of tutorial time, you will be able to access the bigdata accounts remotely via labshare. Labshare is a system that we use to provide access to a mac on campus remotely via remote desktop. While labshare is in theory available off campus, it requires vpn access which can sometimes be problematic. You also may not be able to access a computer via labshare if all the computers are in use (ie: there is a tutorial currently underway).

If you need to access a computer via labshare, please read Tony Gray's fantastic guide to using labshare – it is in the first weeks tutorial section on Mylo.

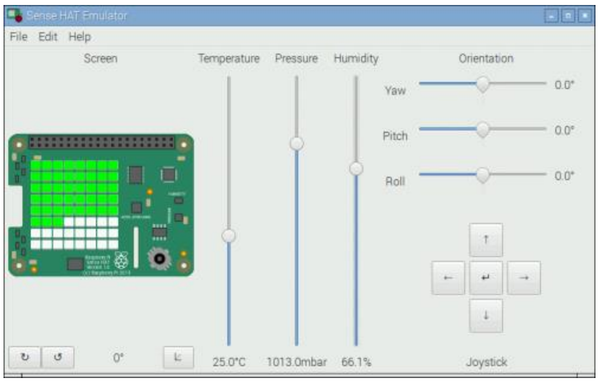
PLEASE NOTE!!!

The virtual machine environment is not persistent – every time you close it down or log out, it will reset to the same state as at the start of your session. This means that saving your work is very important. You will need to transfer your data off the virtual machine to save it. This is covered at the end of this tutorial.

Task 1: Hello World on embedded hardware

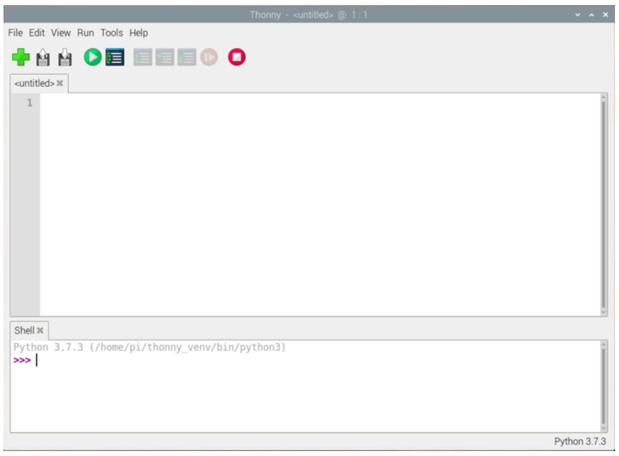
Let’s get started with something a bit traditional. When working with a new environment, it’s really common to write a program to say ‘Hello World’. This will show that we can execute some code and get an expected output. Let’s do Hello World, but let’s do it the embedded systems way.

To simulate an IoT device, we are going to be using a raspberry pi with a simulated sensor device called a Sense Hat. The Sense hat is an add on board that contains a range of sensors, including environmental (temperature, pressure, humidity), positional (accelerometer, compass), a 5 position joystick (direction and press) as well as an 8x8 RGB matrix of LEDs that can be used as a display to scroll text and display simple images.



The Sense Hat is a physical piece of hardware, but in this unit we will be using an emulated version of the device named Sense Hat Emulator. You can find the Sense Hat Emulator in the Programming Menu in the main raspberry pi drop down menu in the top left. We can write code and send it to our hardware the same way we would if it was a physical device.

We can write our code in python, using the Thonny IDE. You can find Thonny in the same menu as you found the Sense Hat Emulator. Thonny is a fairly simple IDE, but it will do everything we need for this unit. Python is a simple language to learn if you haven’t encountered it before – just be aware that white space (indentation) is important. You will pick up enough of the syntax as we go to write your programs.



```
Python 3.7.3 (/home/pi/thonny_venv/bin/python3)
>>> |
```

Let’s get started by writing a simple Hello World program. In Thonny, write the following code in the main window:

```
print(“Hello World”)
```

That’s it. Hit Run in the top menu. You’ll get an output in the console (shell) below that reads Hello World. This is an unremarkable program – it’s just printing to the console window and it’s not currently interacting with the Sense Hat. Embedded devices don’t always have a screen to display console output, so programmers generally blink a LED light in order to prove that they can get code to execute. Here is a basic blink program that will make the first pixel in the LED array blink.

```
        from sense_emu import SenseHat
        import time

        s=SenseHat()
        while True:
                s.set_pixel(0,0,255,0,0)
                time.sleep(1)
                s.set_pixel(0,0,0,0,0)
                time.sleep(1)
```

You should see the first pixel in the LED array start blinking red on the Sense Hat emulator. This program is quite straight forward. It imports the libraries for the Sense Hat and the time module. It creates an instance of the Sense hat, and then it loops through some code, turning the first LED (0,0) on to a red colour (255,0,0), waiting a second, then turning it off, then waiting another second. Make sure that you can get the code executing, and then try to complete the following challenges.

Challenges:

- 1.) Change the LED from Red to a different colour. Hint: The colours are specified in an RGB format.
- 2.) Change the rate at which the LEDs blink.
- 3.) Make a different LED blink.

Task 2: Getting data from the environmental sensors

The Sense Hat contains three sensors for measuring environmental factors, namely temperature, air pressure and humidity. When we’re using the emulator, we lack the hardware to give us real data, but we can simulate feedback from the sensors by adjusting the three sliders on the emulator. These sliders can be adjusted and our program can respond to the changes in values in real time.

A simple method to check the values of these sensors is by **polling** the sensors output. Polling involves periodically checking the current value of a sensor. We can store the value, print it, or compare it against something. Here is some code to poll the temperature sensor every 5 seconds and print the value to the console.

```
from sense_emu import SenseHat
import time

s = SenseHat()
while True:
        temperature = s.get_temperature()
        print("Temperature: ", temperature)
        time.sleep(5)
```

Run your code. You will see the temperature values printer to the console every 5 seconds. While this code is running, drag the Temperature slider in the emulator. Whenever the temperature is polled, it will display the current value of the temperate slider as a floating point number. You can also poll the air pressure with get_pressure(), or the humidity with get_humidity().

Polling sensors is a simple way to gather data from our sensors, but using the sleep method means that our program can't be doing anything else at the same time. We will look at another method to check values without blocking code from executing next week. For now, try to complete the challenges below.

Challenges:

- 4.) Extend the code above to gather data from all three sensors and print them to the console.
- 5.) Introduce an if-else condition to check display whether it is hot or cold (pick a suitable value for the temperature threshold and compare it).

Task 3: Using the Sense Hat Display

We've already used the SenseHat's LED display to test our blinking LED code. While not a high resolution device (8x8 – a long way from 4k resolution), we still have enough resolution to create scrolling text, or simple images. Lets try our Hello World again, but this time lets output our message to the LED screen rather than the console.

```
from sense_emu import SenseHat

s = SenseHat()
while True:
        s.show_message("Hello World")
```

Run your code and you should see the message scroll across the display on the screen. This is all thanks to the SenseHat library – it controls when which LED's to turn on and off for each letter in the message, as well as animate them across the LED matrix. This method can take a simple text string, but you can also specify more arguments to control speed and colour with the following format:

```
show_message(text_string,scroll_speed=0.1,text_colour=[255,255,255],
back_colour=[0, 0, 0])
```

Challenge:

- 6.)Try printing the output from the temperature sensor to the display. Hint: You will need to cast the temperature float to a string using str().
- 7.) Experiment with the colour of the text and it's speed. The speed is specified in seconds, so use a fraction or your results will be extremely slow…

In addition to being able to display text, the LED screen can also be used to create basic images. In the code below, we are going to define an image. An image is a list of 64 elements representing each pixel. Each pixel element is itself and list of the 3 RGB colour elements of the pixel. Here's an example:

```
from sense_emu import SenseHat
s = SenseHat()

w = (255,255,255)
b = (0,0,0)

logo = [
  w, b, b, b, b, b, b, w,
  b, w, b, b, b, b, w, b,
  b, b, w, b, b, w, b, b,
  b, b, b, w, w, b, b, b,
  b, b, b, w, w, b, b, b,
  b, b, w, b, b, w, b, b,
  b, w, b, b, b, b, w, b,
  w, b, b, b, b, b, b, w,
  ]

s.set_pixels(logo)
```

In this example, we've created two lists – w (white) and b(black) These two lists correspond to the RGB colour values for those two colours. We then create a list called logo. Logo is represented across 8 lines for clarity (so that it matches the LED matrix), with each element representing a white or a black pixel. When you run the code above, you should see a cross element appear on the screen.

Challenges:

- 8.) Create another image to display a tick instead of a cross. Feel free to create more colours to experiment with the images created.
- 9.) Update your code to read from the temperature sensor. When the temperature is hot, display a tick; when the temperature is cold, display a cross.

Task 4: Saving your work

Now that you've completed the work for this tutorial, you will need to save your work and transfer it off the virtual machine. There are several ways to do this, but first you will need to save your work in Thonny as a file. Just click the save button in Thonny and give your file a name – saving to the desktop is the easiest solution as the file will be cleared up when you shut down the VM anyway.

With the file saved, you now need to move it from the VM to somewhere more permanent. Some options include:

- Login to your university onedrive via the web browser on the VM. You can upload your files to your one drive so they can be accessed again in future. This method is handy, but you will need to enter your university credentials each time (as the VM is not persistent).
- Transfer the files to the desktop of the mac. You can simply drag from the VM and drop your files onto the Mac's desktop. From here, you will still need to **back them up somewhere more permanently**, as you are using a local account on the mac that will be cleaned up at the end of your session. You can plug a USB device into the mac and copy the file to that device, or use your onedrive via the web on the mac rather than in the VM.

Later in semester and for assignments, you can use this process in reverse to add your work in progress back to the virtual machine so that you can keep working on your files.

Once you've saved your work, you should complete this tutorial. Prior to the submission deadline (generally the Monday following your tutorial), you should upload a copy of this document with the checklist below completed. You may wish to upload your code as well, particularly if you were having problems with your code.

It's important to save your tutorial code as it will be invaluable during your assignments.

Please note:

Your tutorial work counts towards your final mark, but your work in each tutorial is not individually assessed. However, when you require assistance for the unit, we will be looking at your submissions to determine how you are progressing and how best we can offer help. If you are attending consultation, then this is the first thing that we will look at together.

Checklist

Challenge		***Completed* (Y/N/Notes)**
Challenge 1		
Challenge 2		
Challenge 3		
Challenge 4		
Challenge 5		
Challenge 6		
Challenge 7		
Challenge 8		
Challenge 9		