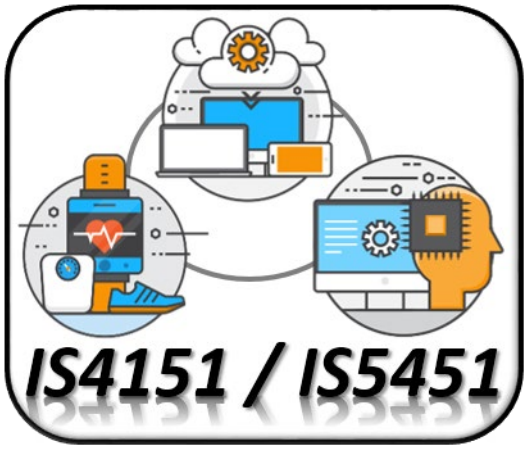


Lecture 7

Single-board Computer (II)

IS4151/IS5451 – AIoT Solutions and Development
AY 2024/25 Semester 2

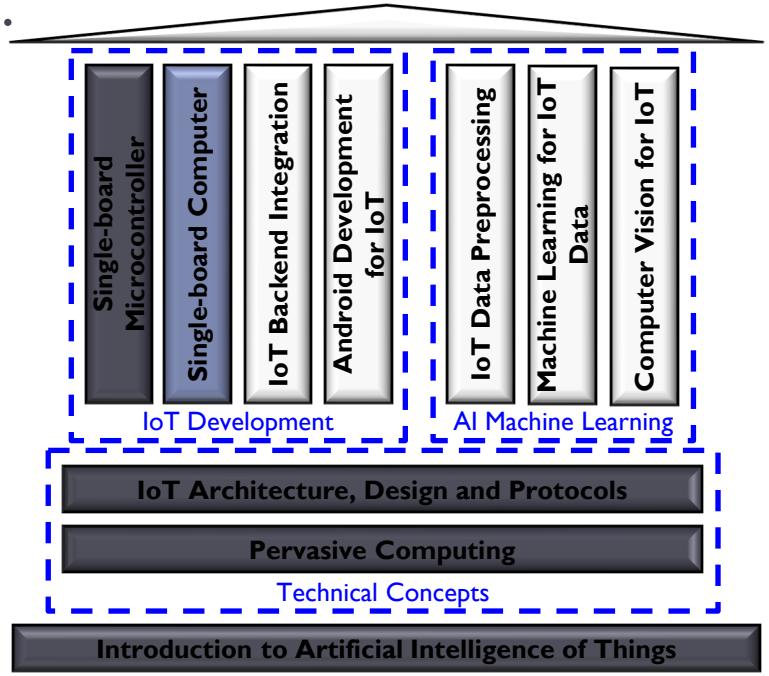
Lecturer: A/P TAN Wee Kek

Email: tanwk@comp.nus.edu.sg :: **Tel**: 6516 6731 :: **Office**: COM3-02-35

Consultation: Tuesday, 2 pm to 4 pm. Additional consultations by appointment are welcome.

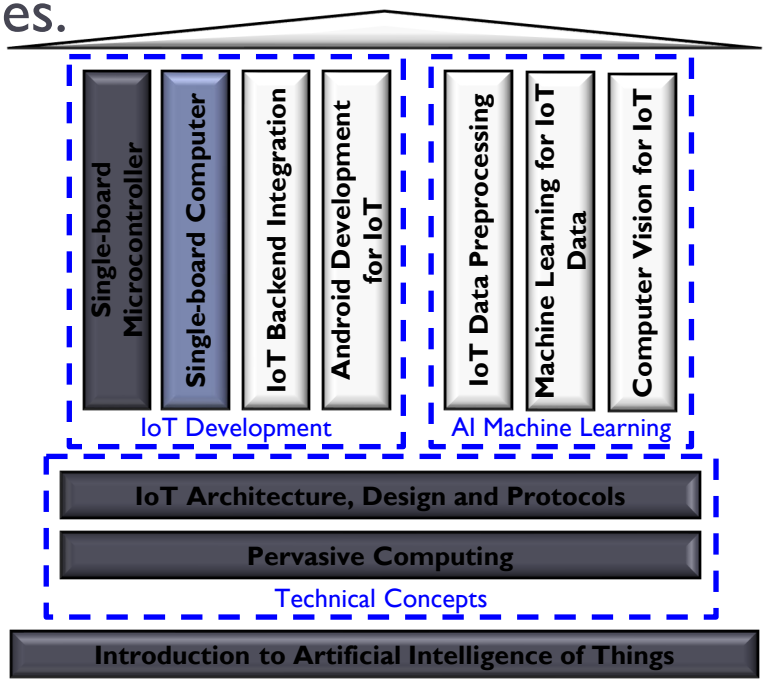
Quick Recap…

- In the previous lecture, we learnt:
  - The technical characteristics of the Raspberry Pi.
  - Some additional concepts on basic electronics.
  - How to perform GPIO programming with the Raspberry PI using both digital and analogue signal.
- RPi is a very capable single-board computer that can play a bigger role beyond a node device.
- This lecture continues our learning journey to extend the role of RPi to a hub.



Learning Objectives

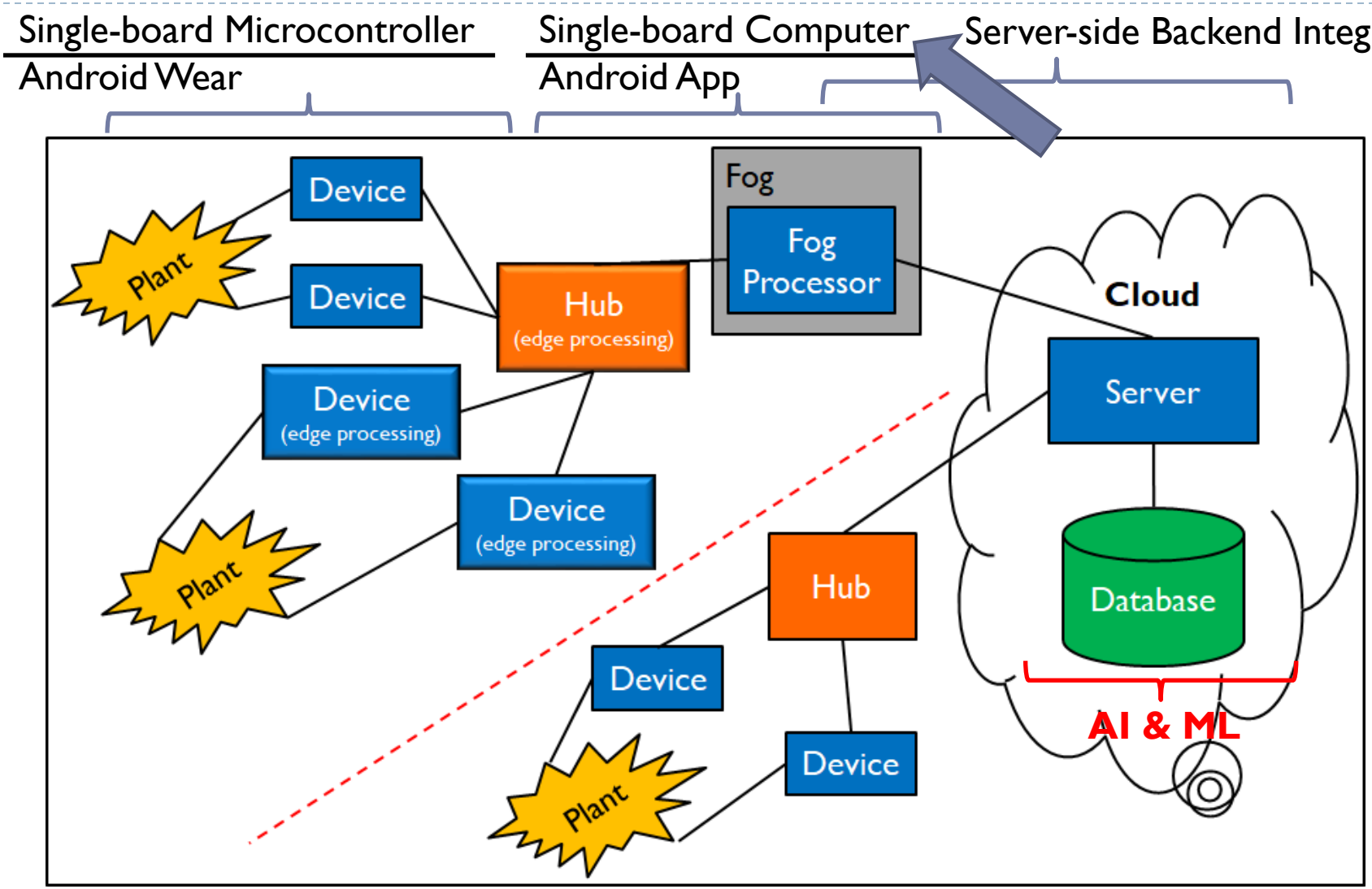
- At the end of this lecture, you should understand:
  - BLE communication with Raspberry Pi.
  - Using Raspberry Pi to control micro:bit devices with BLE and radio.
  - Working with Raspberry Pi's interfaces.



Readings

- Required readings:
  - None.
- Suggested readings:
  - None.

Technical Roadmap for IS4151/IS5451



Computational Capability of Raspberry Pi

- The **Raspberry Pi** is single-board computer:
  - 3 Model B is the third-generation model with wireless LAN and Bluetooth connectivity:
    - Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
    - 1GB RAM
  - The latest model is the 5 Model:
    - Quad Core 2.4GHz Broadcom BCM2712 64bit CPU
    - Up to 16GB RAM
- Just how powerful is the Raspberry Pi 3 Model B relative to a conventional computer?
  - A comparison with my laptop, i.e., Lenovo ThinkPad X1 Extreme:
    - Hexa Core 2.70GHz Intel Core i7-10850H with vPro (12 Threads)
    - 16 GB RAM

Computational Capability of Raspberry Pi (cont.)

- We can use Python's `timeit` module to measure the execution time of a small code snippet on both my laptop and a 3 Model B:

```
pi@raspberrypi: ~/Documents/is4151-is5451/lecture07
File  Edit  Tabs  Help
pi@raspberrypi:~/Documents/is4151-is5451/lecture07 $ python3 src01.py
7.407902810000024
```

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  SQL CONSOLE  DEBUG CONSOLE

D:\Dropbox\Teaching - NUS SoC\IS4151-IS5451 - AY202324S2\Lecture Notes\Week 07\src>python src01.py
1.0021977
```

```python
import timeit

def my_function():
    y = 3.1415

    for x in range(100):
        y = y ** 0.7

    return y

print(timeit.timeit(my_function, number=100000))
```

src01.py

- The execution time on the Raspberry Pi is about 7x slower.
- That is quite impressive given that the Raspberry Pi is about 60x cheaper.

Computational Capability of Raspberry Pi (cont.)

- The Raspberry Pi is capable of acting as a hub and fog processor:
  - Sufficient processing power, memory and secondary storage.
  - Onboard support for WiFi and Bluetooth Low Energy (BLE).
- Recall that edge computing and fog computing:
  - Transfer computation to a more computationally capable and Internet-enabled device that is geographically closer to the edge than to the cloud.
  - Makes it possible to build local views of data flows.
  - Aggregates data to be sent to the cloud for further offline analysis.

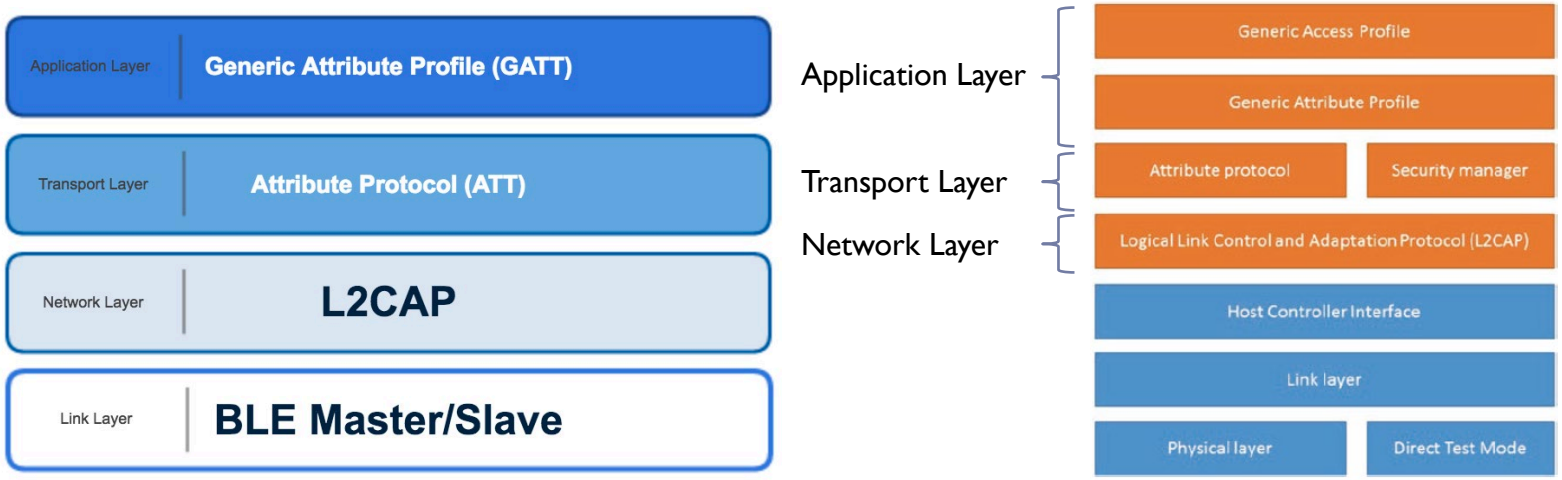
Computational Capability of Raspberry Pi (cont.)

- In this lecture, we will discuss how to program the Raspberry Pi to act as a hub.
- In the next lecture, we will learn how to program the Raspberry Pi to act as a fog processor.

Working with the Bluetooth Low Energy Stack on Raspberry Pi

Quick Recap on BLE…

- Recall that **Bluetooth Low Energy (BLE)**:
  - Is a part of the Bluetooth standard designed for low-power operation such as devices powered from coin cell batteries.
  - A BLE device can work as a transmitter, receiver, or both.
- The transport layer and network layer of BLE is independent of TCP/IP unlike MQTT:



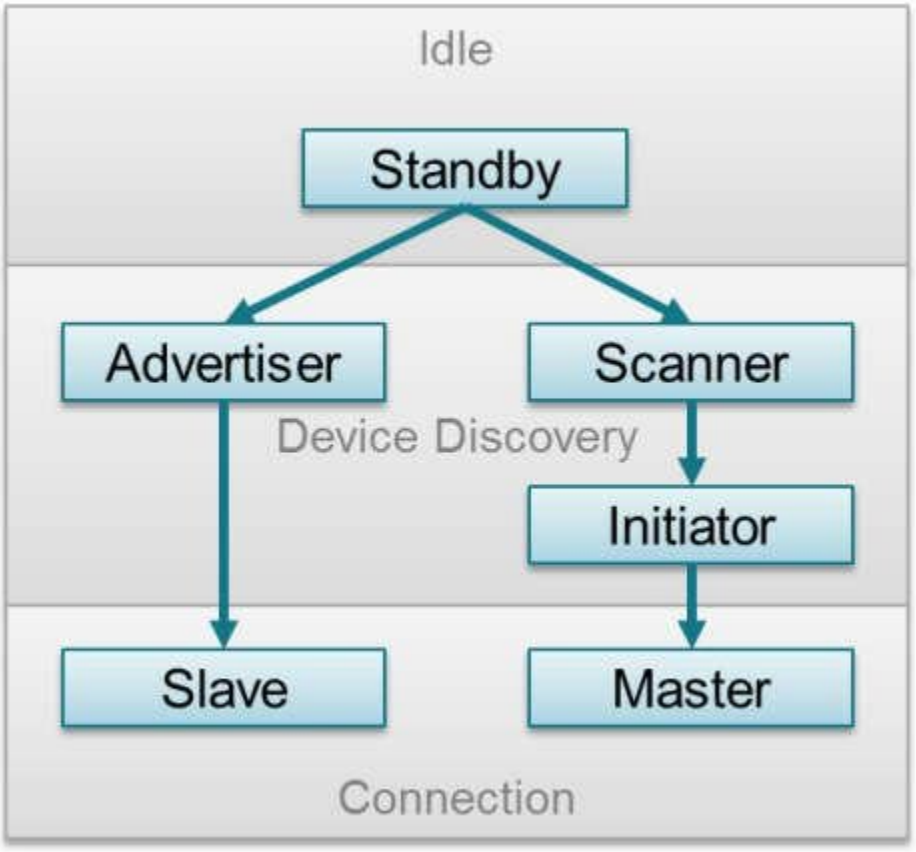
Quick Recap on BLE… (cont.)

- **Generic Access Profile (GAP)** controls connections and advertising among BLE devices.
- **Generic Attributes Profile (GATT)** defines a hierarchical data structure that is exposed to connected BLE devices.
- GATT uses **Attribute Protocol (ATT)** to send and receive messages.

Generic Access Profile (GAP)

- The **GAP** layer of the BLE protocol stack is responsible for connection functionality.
- This layer handles the access modes and procedures of the device including:
  - Device discovery.
  - Link establishment.
  - Link termination.
  - Initiation of security features.
  - Device configuration.
- Based on the role for which the device is configured, the GAP state diagram on the next slide shows the states of the device.

Generic Access Profile (GAP) (cont.)



Generic Access Profile (GAP) (cont.)

- Standby:
  - The device is in the initial idle state upon reset.
- Advertiser:
  - The device is advertising with specific data letting any scanning devices know that it is a connectible device.
  - This advertisement contains the device address and can contain some additional data such as the device name.
- Scanner:
  - When receiving the advertisement, the scanning device sends a scan request to the advertiser.
  - The advertiser responds with a scan response.
  - This process is called device discovery.

Generic Access Profile (GAP) (cont.)

- The scanning device is aware of the advertising device and can initiate a connection with it.

- Initiator:
  - When initiating, the initiator must specify a peer device address to connect.
  - The initiating device sends out a request to establish a connection (link) with the advertising device together with the connection parameters.
- Slave/Master:
  - When a connection (link) is formed, the advertiser device functions as a slave and the initiator device functions as a master.

BlueZ

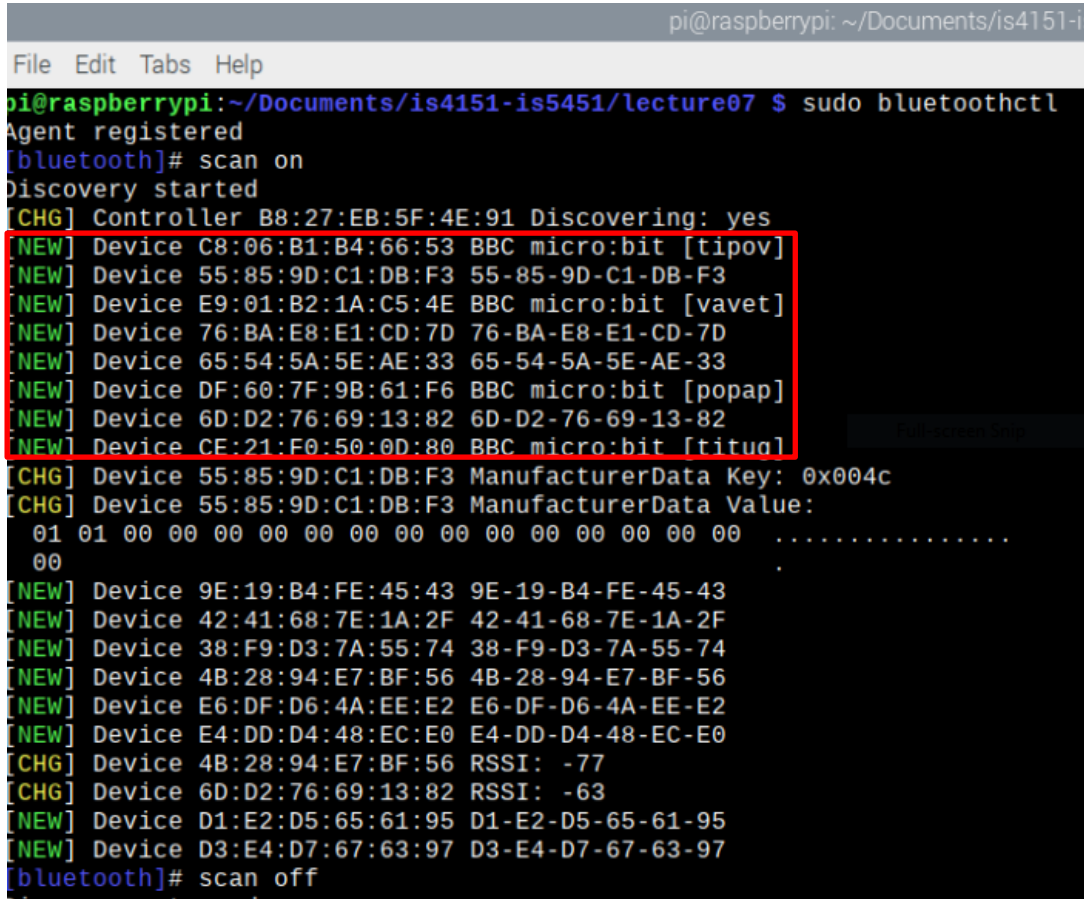
- On the Raspberry Pi OS, the implementation of the Bluetooth protocol stack is BlueZ:
  - BlueZ is the official Linux Bluetooth protocol stack.
  - Allows Raspberry Pi to communicate with Bluetooth classic and Bluetooth low energy (LE) device
  - More information about BlueZ – http://www.bluez.org/
- Need to install the bluez and bluez-utils packages:
  - The bluez package provides the Bluetooth protocol stack.
  - The bluez-utils package provides the `bluetoothctl` utility tool.
  - Both packages should be installed by default:
    - May need to upgrade to the latest version.

BlueZ (cont.)

- The `bluetoothctl` utility can be used to perform device discovery.
  - We will test with the micro:bit Bluetooth temperature service.
  - See `microbit-ble-temperature.js`.

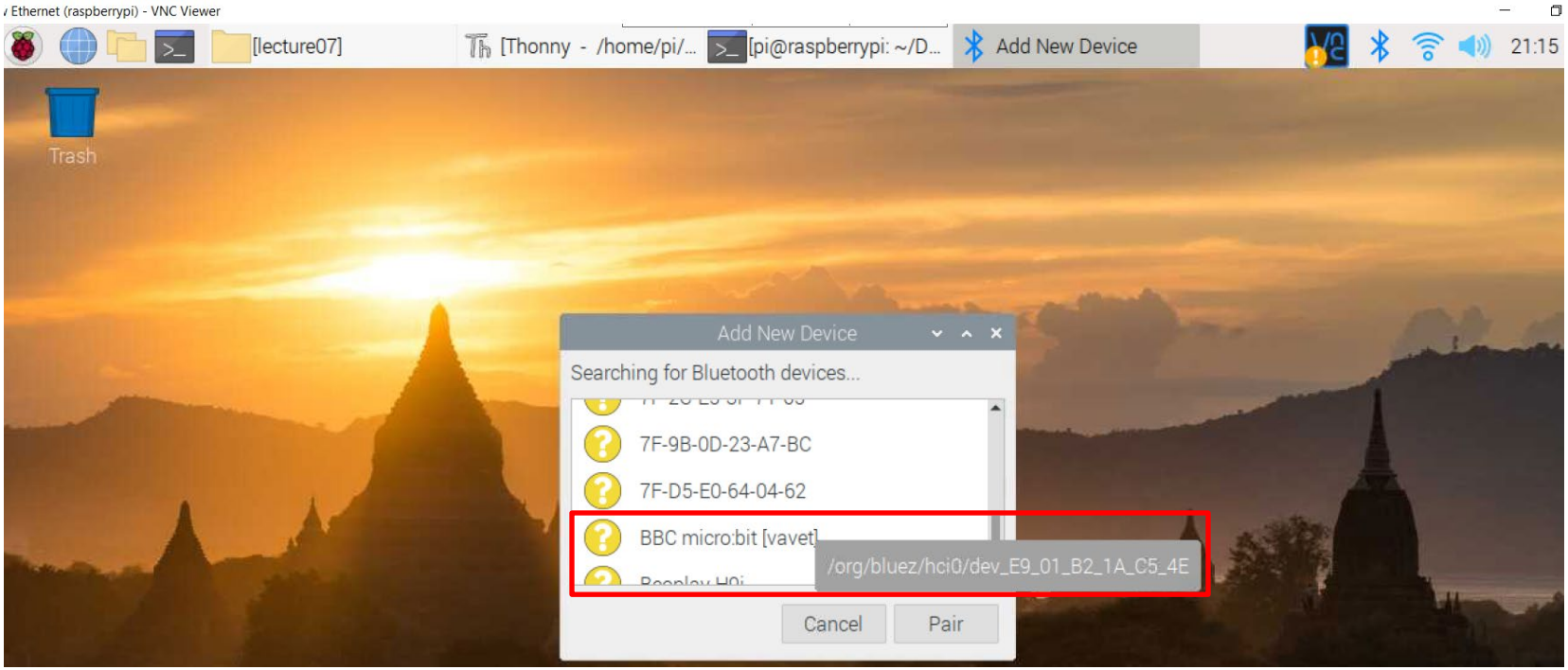
Bluetoothctl

- Using bluetoothctl to perform device scanning and discovery: `sudo bluetoothctl`

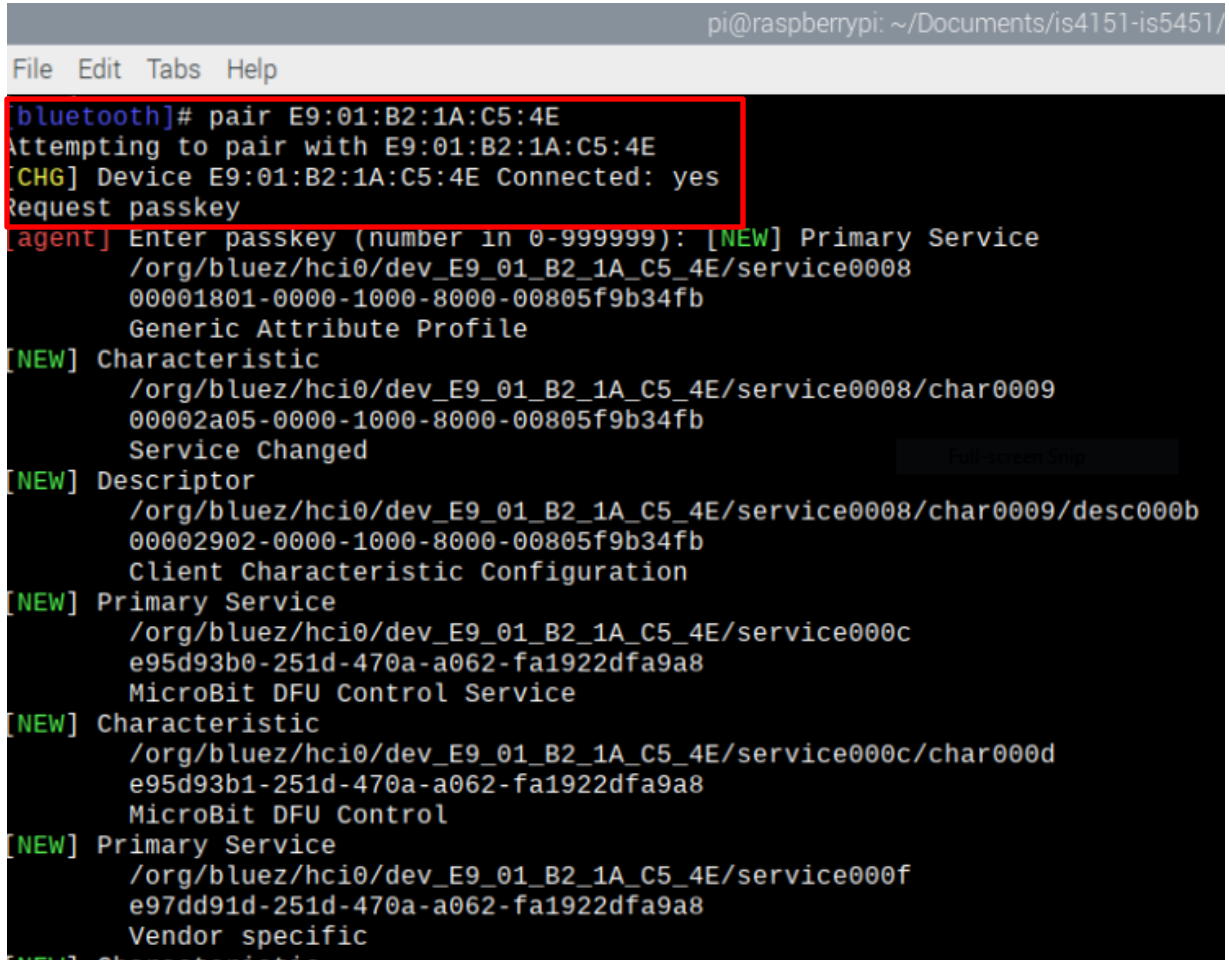


Bluetoothctl (cont.)

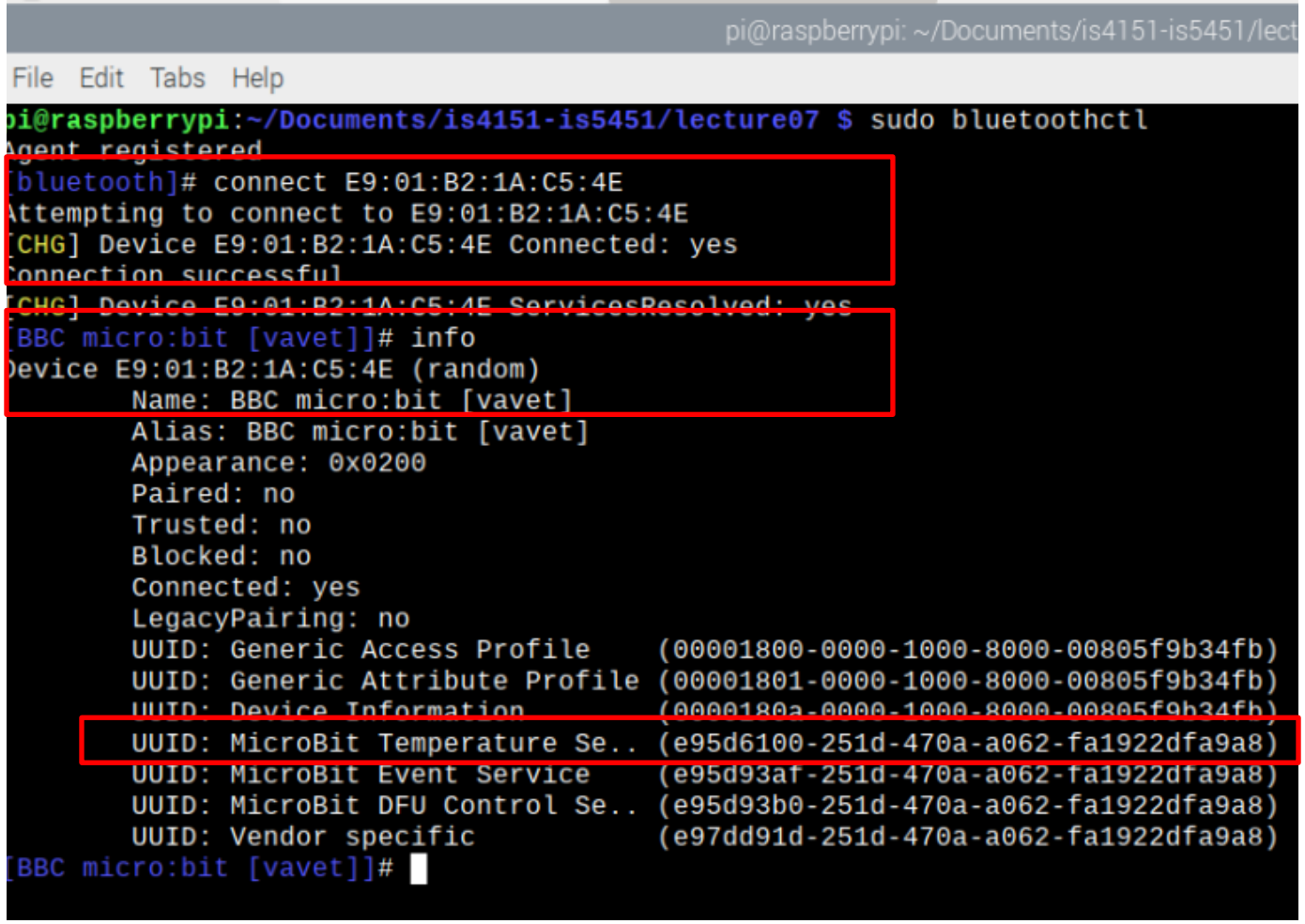
- If you are unable to identify your micro:bit device in the scan results by name:
  - Goto "Manage Bluetooth devices"
  - Select "Add New Device"



Bluetoothctl (cont.)



Bluetoothctl (cont.)



e95d6100-251d-470a-a062-fa1922dfa9a8 refers to the micro:bit Bluetooth temperature service.

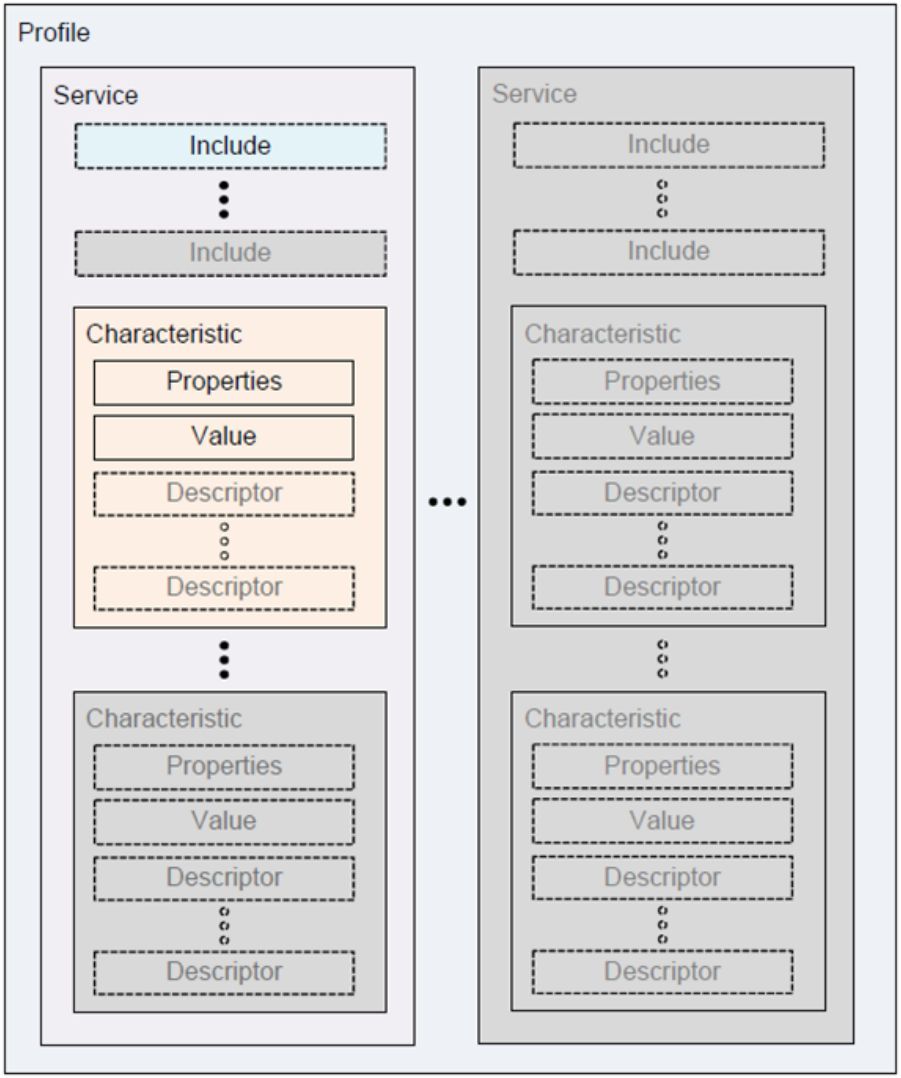
Generic Attributes Profile (GATT)

- **GATT** establishes in detail how to exchange all profile and user data over a BLE connection:
  - GAP defines the low-level interactions with devices.
  - In contrast, GATT only deals with actual data transfer procedures and formats.
- GATT also provides the reference framework for all GATT-based profiles:
  - Profiles cover precise use cases and ensure interoperability between devices from different vendors.
  - All standard BLE profiles are therefore based on GATT and must comply with it to operate correctly.

Generic Attributes Profile (GATT) (cont.)

- GATT uses ATT as its transport protocol to exchange data between devices.
- The data is organized hierarchically into top-level sections called **services**.
- Each service groups conceptually related pieces of user data called characteristics.
- **Characteristic**:
  - A characteristic consists of a type (represented by a UUID), a value, a set of properties or attributes indicating the operations the characteristic supports.
  - It may also include one or more descriptors – metadata or configuration flags – relating to the owning characteristic.

Generic Attributes Profile (GATT) (cont.)



Generic Attributes Profile (GATT) (cont.)

- **GATT server**:
  - It receives requests from a client and sends responses back.
  - It also sends server-initiated updates when configured to do so.
  - It is responsible for storing and making the user data available to the client, organized in attributes.
  - Every BLE device must include at least a basic GATT server that can respond to client requests, even if only to return an error response.
- **GATT client**:
  - It sends requests to a server and receives responses (and server-initiated updates) from it.
  - The GATT client does not know anything in advance about the server's attributes.

Generic Attributes Profile (GATT) (cont.)

- Must first inquire about the presence and nature of those attributes by performing service discovery.
- After completing service discovery, it can then start reading and writing attributes found in the server, as well as receiving server-initiated updates.

- **Properties or attributes**:
  - Properties or attributes are the smallest data entity defined by GATT (and ATT).
  - They are addressable pieces of information that can contain relevant user data (or metadata).
  - The attribute handle is a unique 16-bit identifier for each attribute on a particular GATT server.

Generic Attributes Profile (GATT) (cont.)

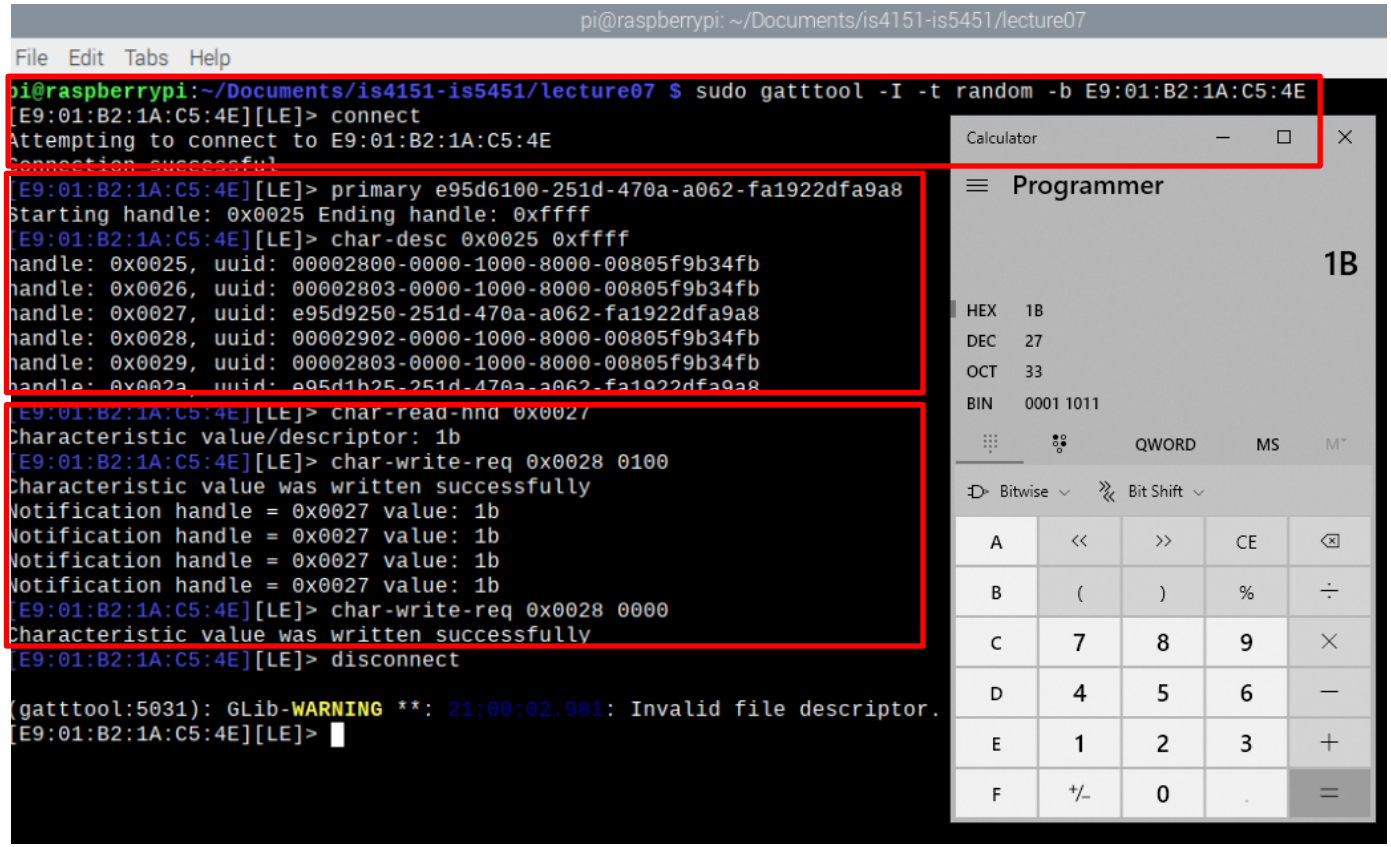
- It is the part of each attribute that makes it addressable, and it is guaranteed not to change.
- The attribute type is nothing other than a UUID, i.e., universally unique identifier.

Gattlib

- Gattlib is the library that is used to access GATT information from BLE (Bluetooth Low Energy) devices.
- Installation of the bluetooth and bluez libraries on the Raspberry Pi will install gattlib.
- Use the `gatttool` utility to query a GATT server.

Gatttool

- Using gatttool to connect to micro:bit:



e95d9250-251d-470a-a062-fa1922dfa9a8 refers to the micro:bit Bluetooth temperature service's temperature value.

Working with Micro:bit Bluetooth Services

Quick Recap on micro:bit BLE…

- Recall that micro:bit supports wireless connectivity via Bluetooth and Radio:
  - Specifically, Bluetooth Low Energy (BLE) is used for communication with other non-micro:bit devices.
- A device such as a Raspberry Pi or smartphone can use any of the Bluetooth "services" provided by a micro:bit:
  - In general, device needs to be paired with micro:bit first.
  - However, we can configure micro:bit to operate in "no pairing required" mode such that any device can connect directly.
  - Once paired and connected, the device can exchange data relating to many of the micro:bit's features.
  - Data exchange is enabled via various Bluetooth services.

Bluetooth Temperature Service

- A micro:bit is able to provide a rough measure of the current environmental temperature in degrees celsius:
  - The approximate temperature value is inferred from the temperature of its main processor.
- The Bluetooth temperature service allows another device to wirelessly:
  - Query the micro:bit's current temperature reading.
  - Receive a constant stream of temperature data values.
- This service allows any device to function as a graphical thermometer using micro:bit as the sensor.
- No additional code is needed on the micro:bit to use the Bluetooth temperature service from another device.

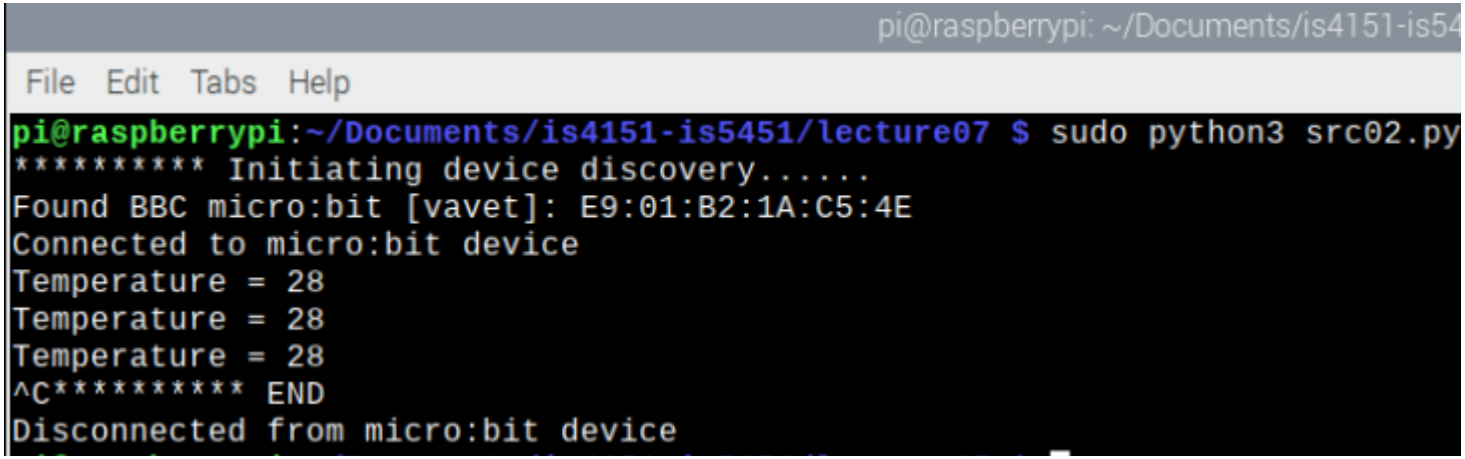
Bluetooth Temperature Service (cont.)

- Service parameters:

Type	UUID	Description
Primary Service	e95d6100-251d-470a-a062-fa1922dfa9a8	Temperature Service
Descriptor	00002902-0000-1000-8000-00805f9b34fb	Client Characteristic Configuration
Characteristic	e95d9250-251d-470a-a062-fa1922dfa9a8	Temperature Value
Characteristic	e95d1b25-251d-470a-a062-fa1922dfa9a8	Temperature Period

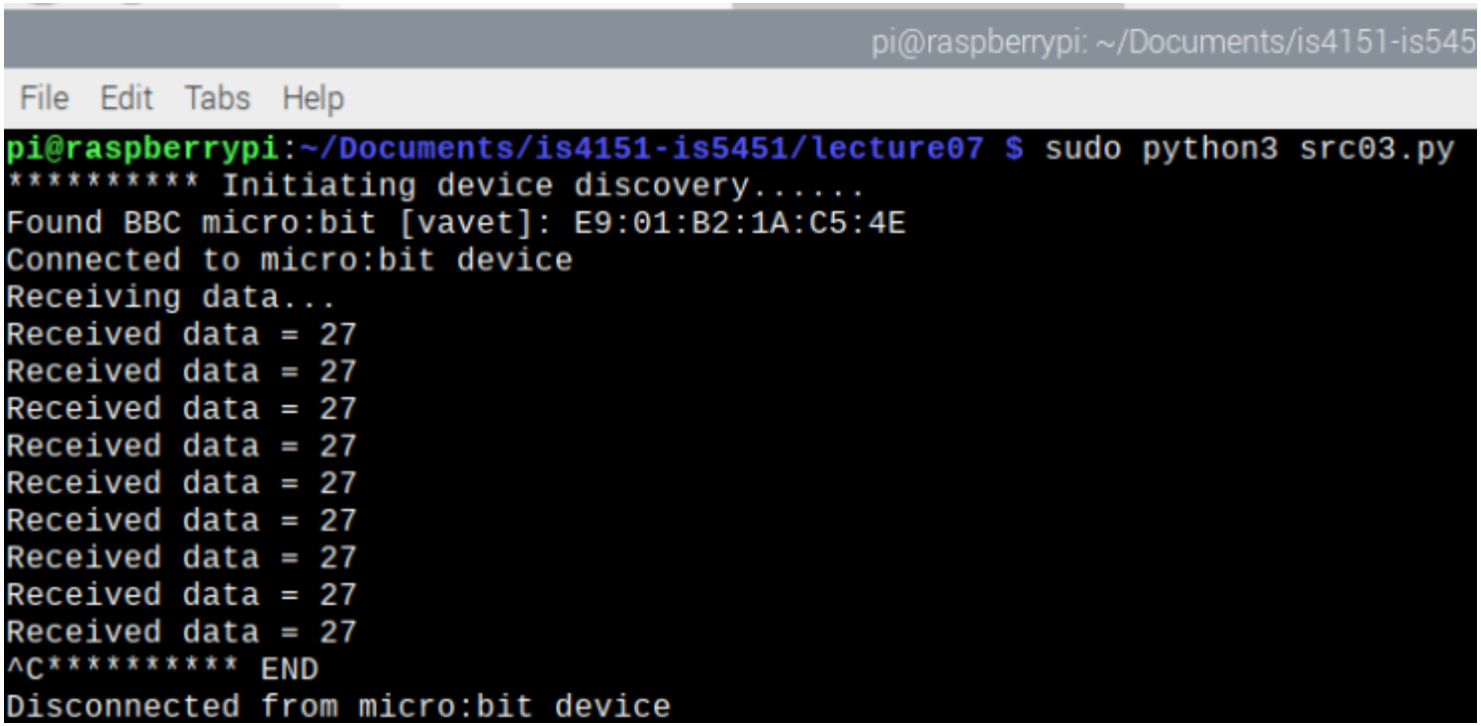
Bluetooth Temperature Service (cont.)

- Reading temperature manually at periodic interval from one micro:bit device:
  - See sample source file `src02.py`



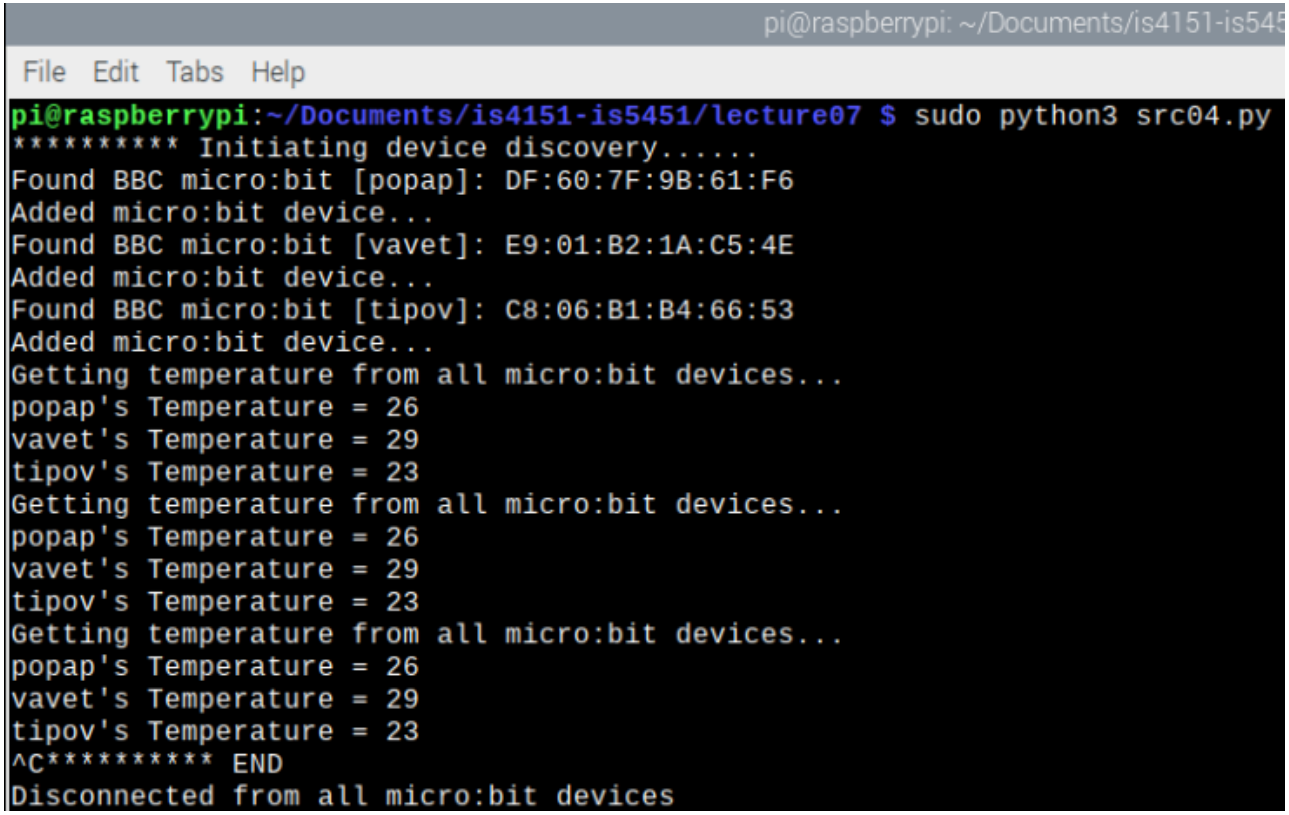
Bluetooth Temperature Service (cont.)

- Subscribing to auto-update of temperature from one micro:bit device:
  - See sample source file `src03.py`



Bluetooth Temperature Service (cont.)

- Reading temperature manually at periodic interval from multiple micro:bit devices:
  - See sample source file `src04.py`



Bluetooth Magnetometer Service

- Micro:bit's magnetometer:
  - Measures the strength and direction of magnetic fields.
  - It can be used as a digital compass and indicate the way the micro:bit is pointing relative to the magnetic north.
- The Bluetooth magnetometer service allows another device to wirelessly receive data from the micro:bit's magnetometer.
- This service allows a device or application to display the direction of travel with real-time updates.

Bluetooth Magnetometer Service (cont.)

- Service parameters:

Type	UUID	Description
Primary Service	e95df2d8-251d-470a-a062-fa1922dfa9a8	Magnetometer Service
Descriptor	00002902-0000-1000-8000-00805f9b34fb	Client Characteristic Configuration
Characteristic	e95db358-251d-470a-a062-fa1922dfa9a8	Calibrate
Characteristic	e95d9715-251d-470a-a062-fa1922dfa9a8	Bearing
Characteristic	e95dfb11-251d-470a-a062-fa1922dfa9a8	Raw Sensor Values (X, Y, Z)

Bluetooth UART Service

- The Bluetooth UART service allows another device to exchange any data it wants to with the micro:bit:
  - Data exchanges are done in small chunks.
  - The data are intended to be joined together before further processing.
- More about UART:
  - UART stands for Universal Asynchronous Receiver Transmitter.
  - It is used to perform serial data communications usually between two devices connected by a physical, wired connection.
  - The Bluetooth UART service emulates the behaviour of a physical UART system.

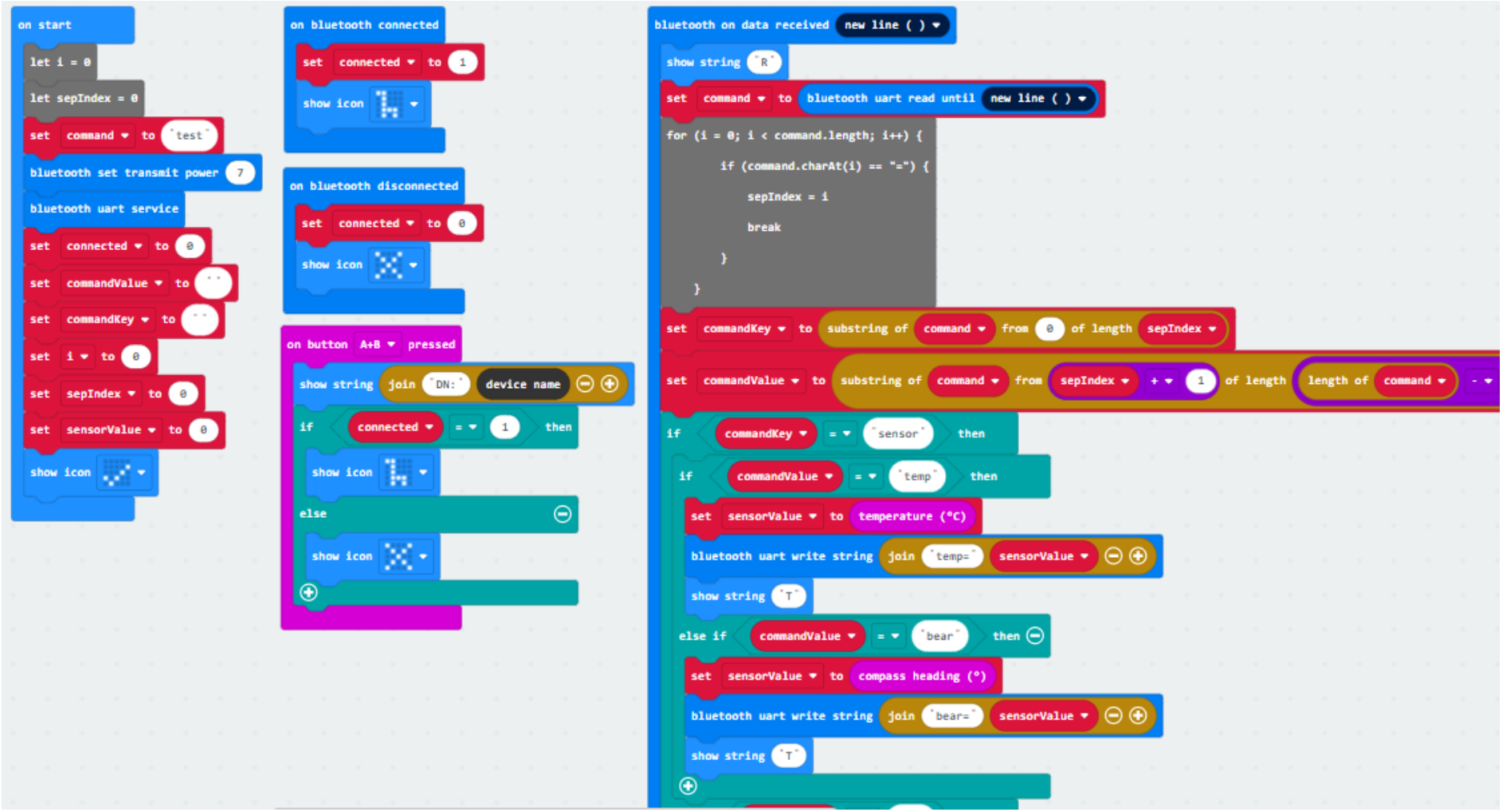
Bluetooth UART Service (cont.)

- Bluetooth UART allows the exchange of a maximum of 20 bytes of data at a time in either direction.
- When the Bluetooth UART service is in-use:
  - Micro:bit sets up a 60-byte buffer and data it receives will be accumulated in the buffer until it is full.
  - We can indicate a special character which will be used to indicate that the entire message in at most three chunks has now been sent by the other connected device.
  - At this point, micro:bit will release the entire contents of its buffer to any code trying to read it.
  - The special character is known as the "delimiter".

Bluetooth UART Service (cont.)

- The Bluetooth UART service can be used for many use cases because the messages can contain "anything".
- Unlike most other Bluetooth services such as temperature and magnetometer, the UART service requires additional micro:bit code:
  - For detecting the connection state of the other device.
  - For reading and using data from the UART buffer.
  - For writing data to the UART buffer for transmission to another device.

Bluetooth UART Service (cont.)



microbit-ble-uart-advanced.js

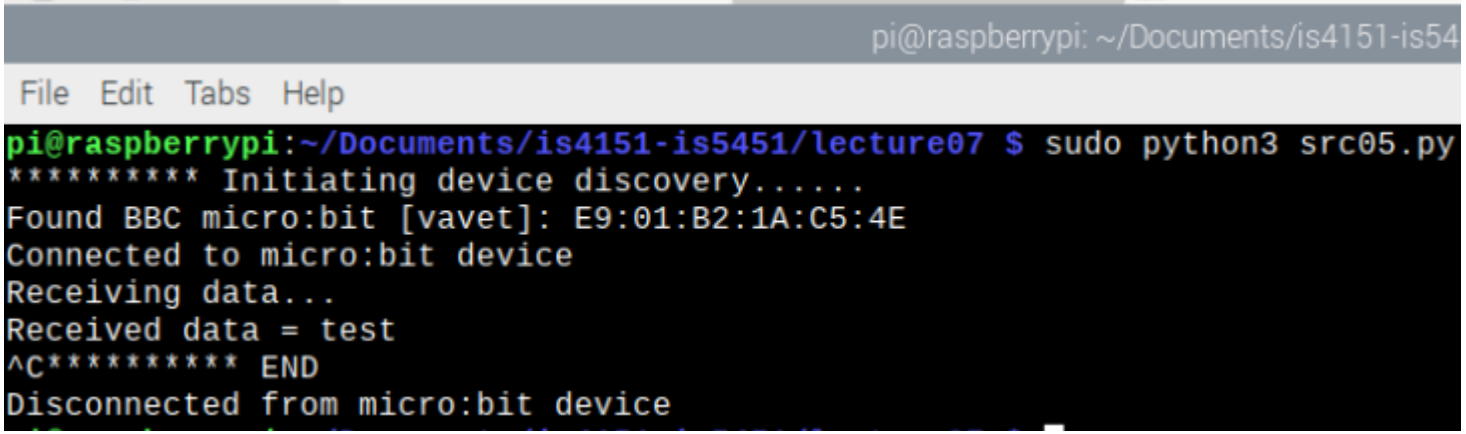
Bluetooth UART Service (cont.)

- Service parameters:

Type	UUID	Description
Primary Service	6e400001-b5a3-f393-e0a9-e50e24dcca9e	UART Service
Descriptor	00002902-0000-1000-8000-00805f9b34fb	Client Characteristic Configuration
Characteristic	6e400002-b5a3-f393-e0a9-e50e24dcca9e	UART TX
Characteristic	6e400003-b5a3-f393-e0a9-e50e24dcca9e	UART RX

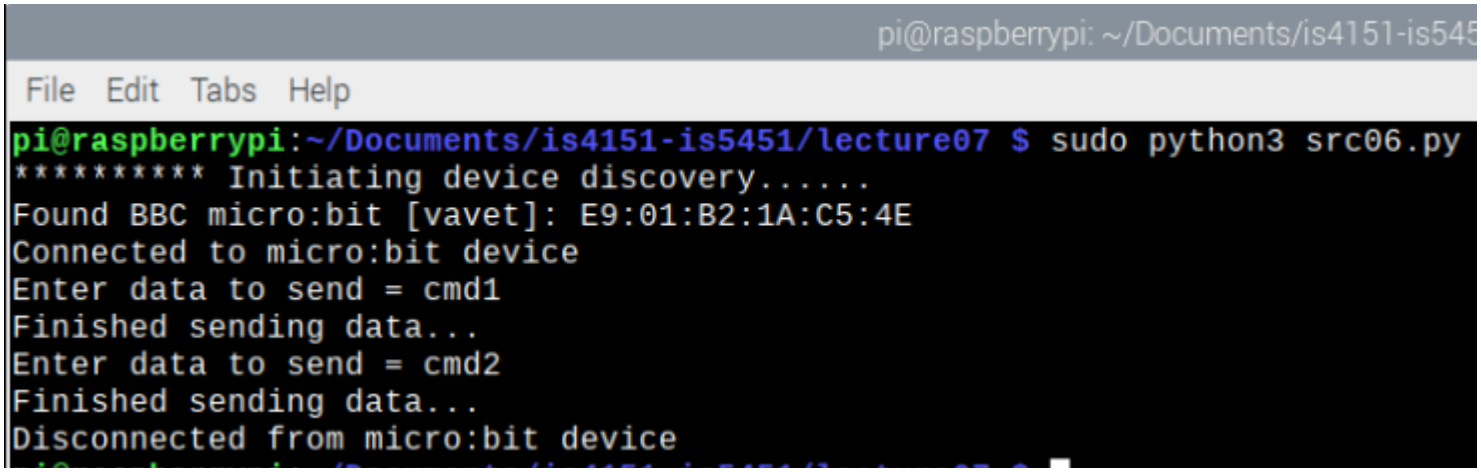
Bluetooth UART Service (cont.)

- Subscribing to auto-update of UART data from one micro:bit device:
  - See sample source files `microbit-ble-uart.js` and `src05.py`



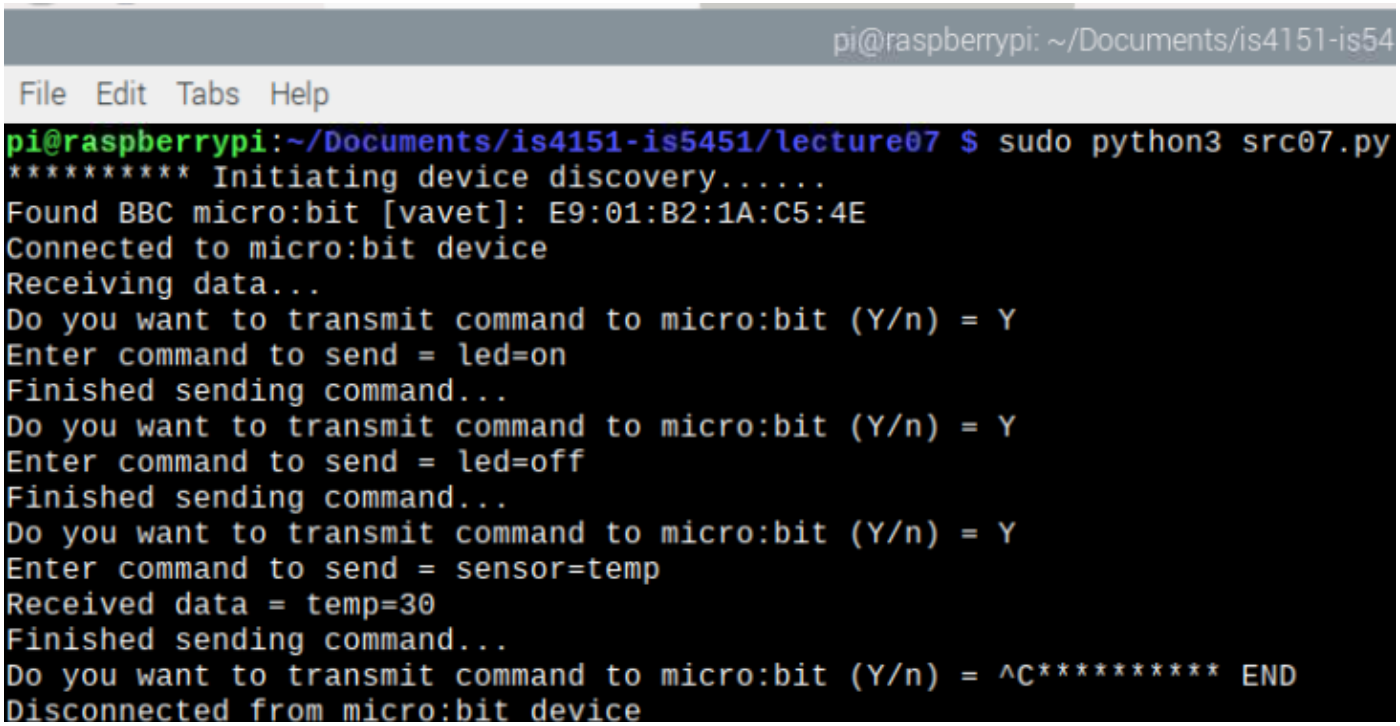
Bluetooth UART Service (cont.)

- Sending UART data manually to one micro:bit device:
  - See sample source file `src06.py`



Bluetooth UART Service (cont.)

- Sending and receiving UART data manually to/from one micro:bit device:
  - See sample source files `microbit-ble-uart-advanced.js` and `src07.py`



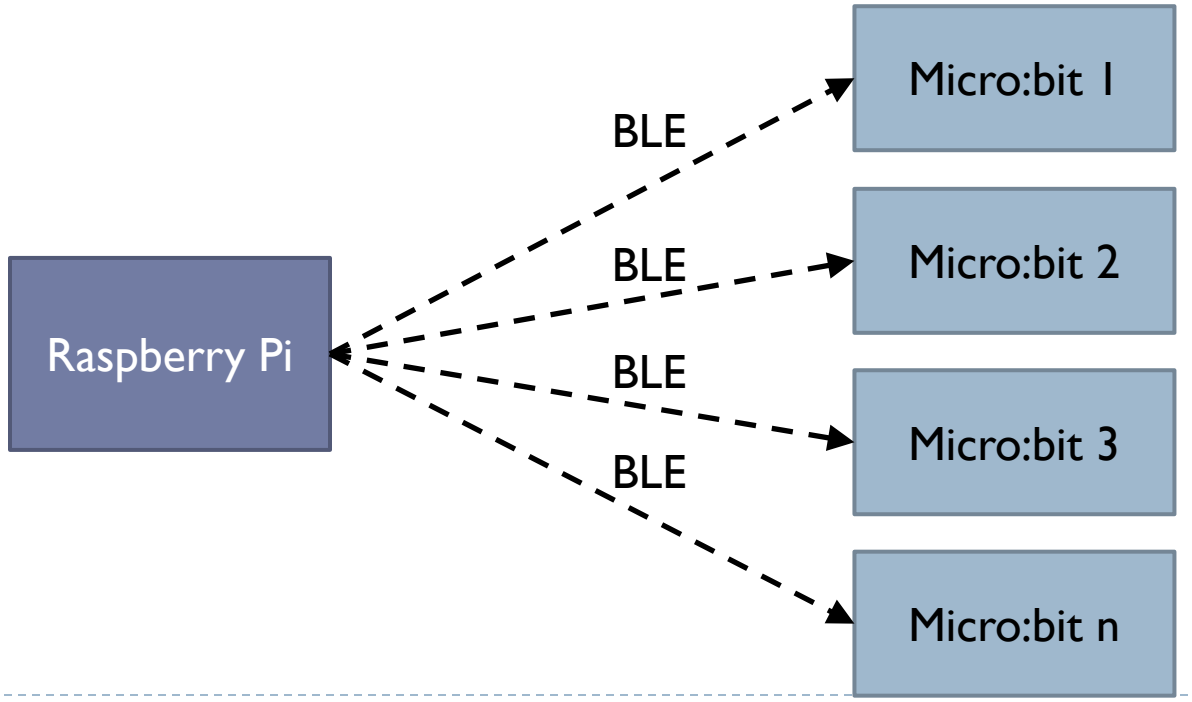
Bluetooth UART Service (cont.)

- Sending and receiving UART data manually to/from multiple micro:bit devices:
  - See sample source file `src08.py`

```
********** Initiating device discovery......
Found BBC micro:bit [vavet]: E9:01:B2:1A:C5:4E
Added micro:bit device...
Found BBC micro:bit [tipov]: C8:06:B1:B4:66:53
Added micro:bit device...
Found BBC micro:bit [popap]: DF:60:7F:9B:61:F6
Added micro:bit device...
Do you want to transmit command to micro:bit (Y/n) = Y
Enter command to send = sensor=temp
Received data = temp=34
Received data = temp=29
Received data = temp=32
Finished sending command to all micro:bit devices...
Do you want to transmit command to micro:bit (Y/n) = Y
Enter command to send = led=on
Finished sending command to all micro:bit devices...
Do you want to transmit command to micro:bit (Y/n) = Y
Enter command to send = led=off
Finished sending command to all micro:bit devices...
Do you want to transmit command to micro:bit (Y/n) = ^C********** END
Disconnected from all micro:bit devices
```

Bluetooth UART Service (cont.)

- The preceding example in sample source file `src08.py` demonstrates one approach for a Raspberry Pi to:
  - Act as a hub.
  - Control and interact with multiple micro:bit devices via BLE:



Working with Raspberry Pi's Interfaces

Quick Recap on Raspberry Pi Interfaces…

- Recall that Raspberry Pi is a single-board computer as compared to the micro:bit, which is a microcontroller:
  - Raspberry Pi has a set of proper interfaces in addition to the GPIO pins.
  - These interfaces include:
    - 4 USB 2 ports, with switched Micro USB power source up to 2.5A.
    - 4 Pole stereo output and composite video port.
    - Full size HDMI.
    - CSI camera and DSI display port for connecting a Raspberry Pi camera and touchscreen display, respectively.
- Also recall that Raspberry Pi runs on a full-fledged operating system, another major differentiation factor from a microcontroller.

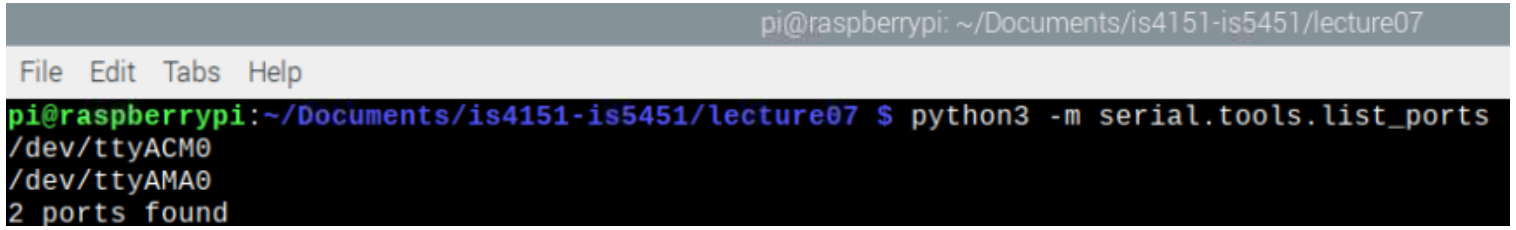
Quick Recap on Raspberry Pi Interfaces… (cont.)

- These interfaces and the operating system enable us to implement additional use cases that strengthen the role of Raspberry Pi as a hub and fog processor.

Serial Communication Over USB

- Recall that micro:bit supports reading and writing data over a serial connection.
- By default, micro:bit is set to use the USB connection for serial data:
  - When micro:bit is connected to a Raspberry Pi, it will appear as a new serial port to the Raspberry PI OS, e.g., `/dev/ttyACM0`
  - The `pySerial` module can be used to check the available serial ports on a Raspberry Pi:

```
python3 -m serial.tools.list_ports
```

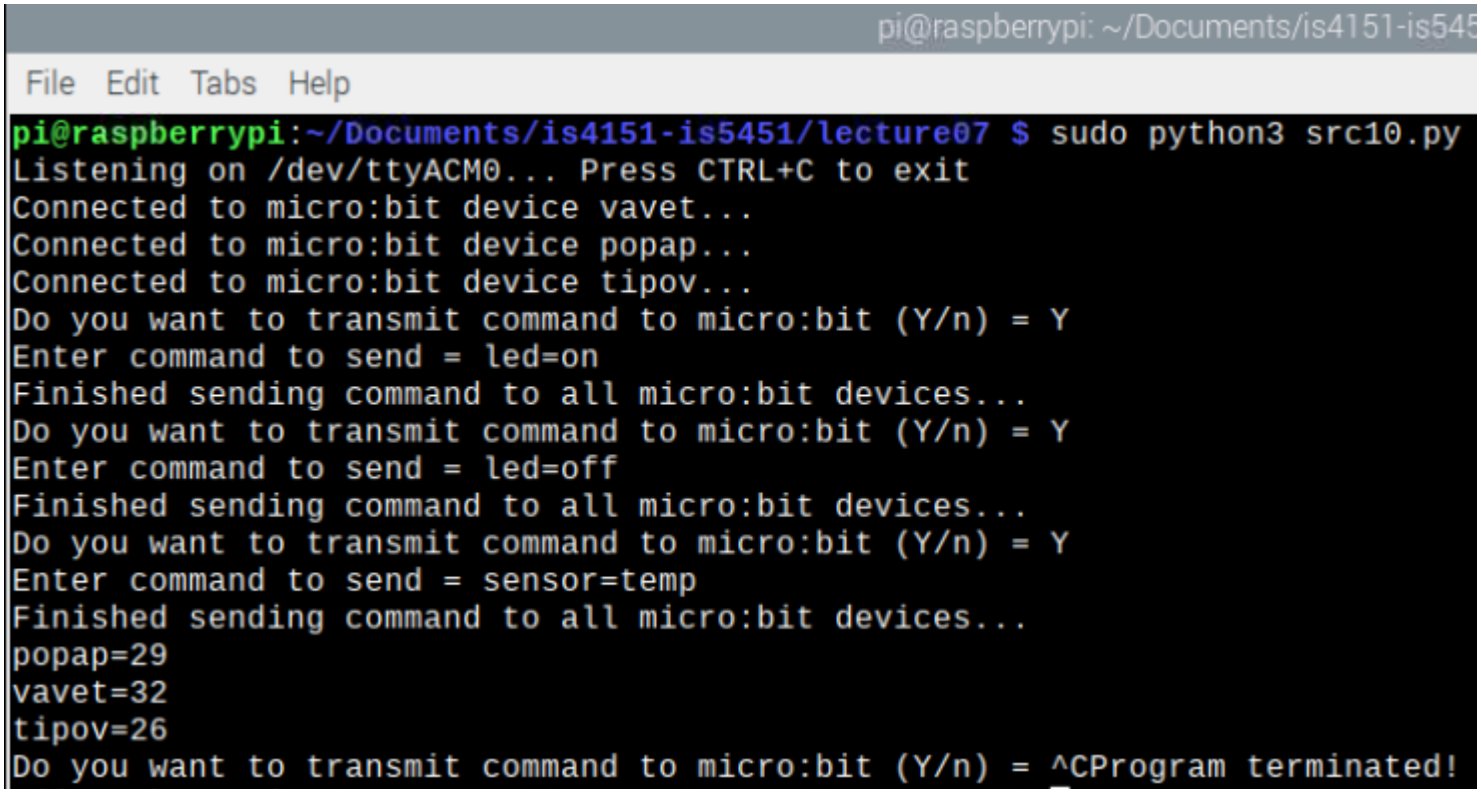


Serial Communication Over USB (cont.)

- We can use `pySerial` to communicate with another micro:bit device easily over USB instead of GPIO:
  - See sample source file `microbit-serial-over-usb-basic.js` and `src09.py`

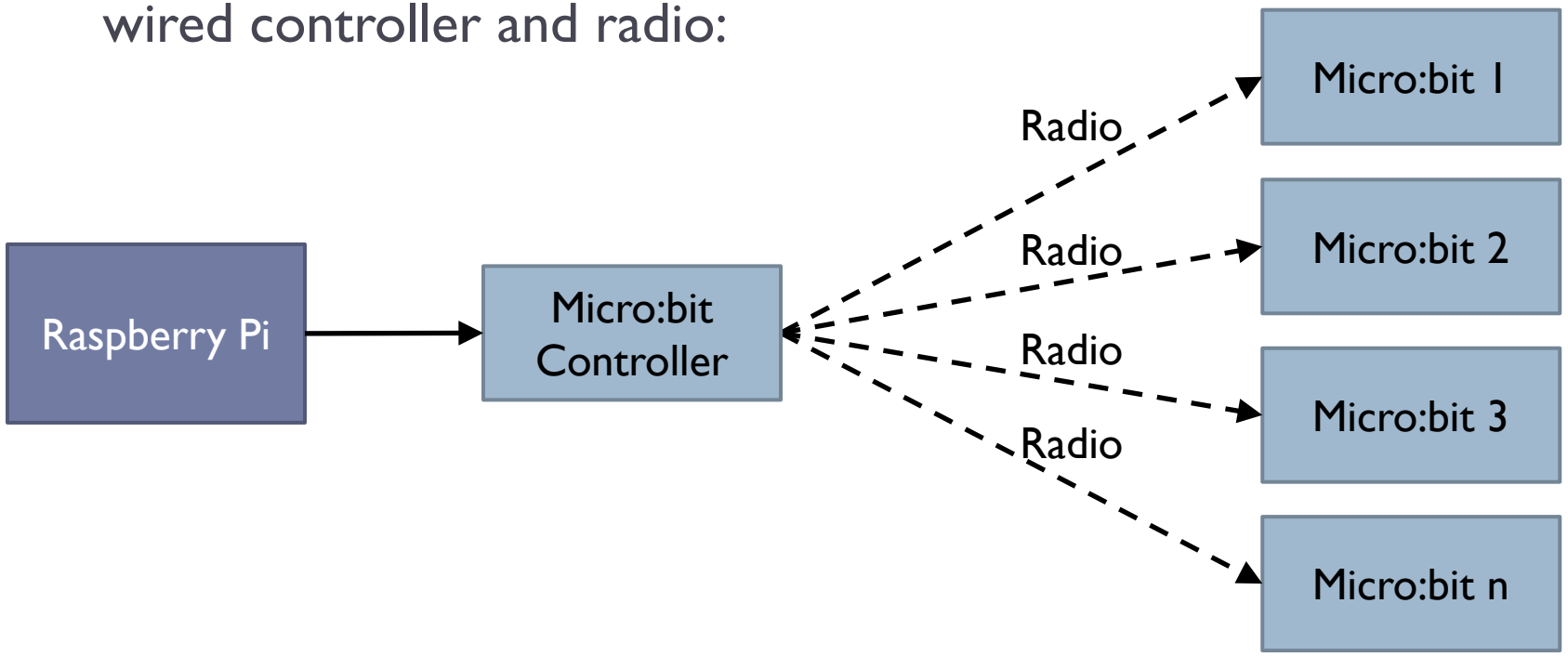
Serial Communication Over USB (cont.)

- The micro:bit device can then communicate with other micro:bit devices using radio wireless communication:
  - See sample source file `src10.py`



Serial Communication Over USB (cont.)

- The preceding example in sample source file `src10.py` demonstrates another approach for a Raspberry Pi to:
  - Act as a hub.
  - Control and interact with multiple micro:bit devices via one wired controller and radio:

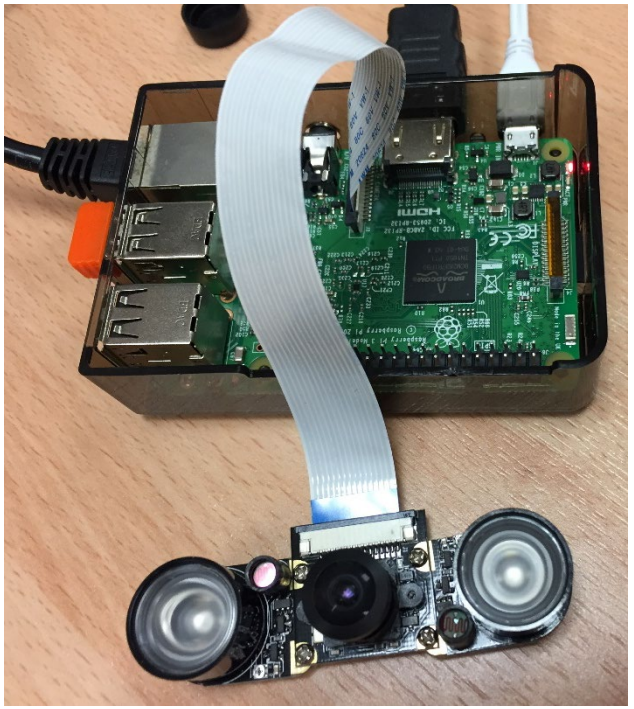


Raspberry Pi Camera

- The Raspberry Pi camera together with Python's `picamera` module allows the device to take still pictures, record video, and apply image effects:
  - Camera preview only works when a monitor is connected to the Raspberry Pi.
  - Remote access (such as SSH and VNC) will not allow you to see the camera preview.

```
1  from picamera import PiCamera
2  from time import sleep
3
4  camera = PiCamera()
5
6  camera.start_preview()
7  sleep(10)
8  camera.stop_preview()
```

src11.py



Raspberry Pi Camera (cont.)

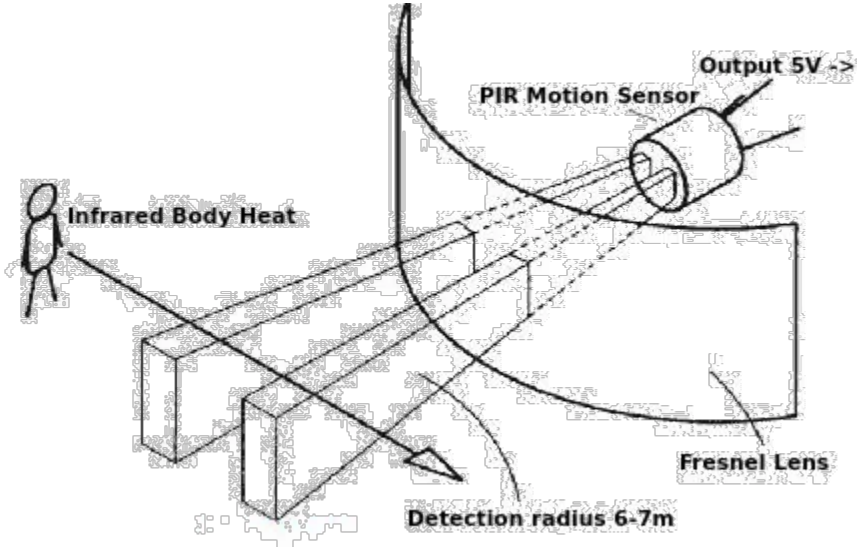
Raspberry Pi Camera (cont.)

- The most common use case for the camera is taking still pictures:
  - The image files can be stored locally or transmitted to a cloud server.
  - Need to sleep for at least 2 seconds before capturing, to give the sensor time to set its light levels.
  - See sample source file `src12.py`
- We can also record video easily:
  - Image and video recording can be combined with other suitable sensors, e.g., PIR motion sensor, as well as data analytics to implement advanced use cases.
  - See sample source file `src13.py`

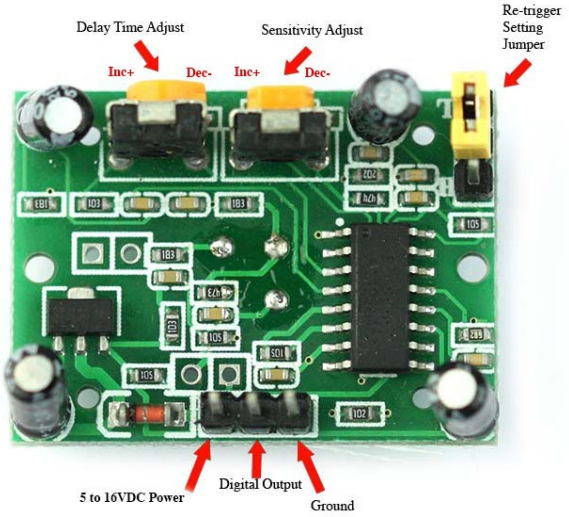
Passive Infrared (PIR) Motion Sensor

- PIR sensor is used to detect motion from human or other objects from about 6 meters away.
- Working principle of PIR sensor:
  - The motion sensor consists of a fresnel lens, an infrared detector, and supporting detection circuitry.
  - The lens on the sensor focuses any infrared radiation present around it toward the infrared detector.
  - Our bodies generate infrared heat, and as a result, this heat is picked up by the motion sensor.
  - The sensor outputs a 5V signal for a period of time as soon as it detects the presence of a person.

Passive Infrared (PIR) Motion Sensor (cont.)

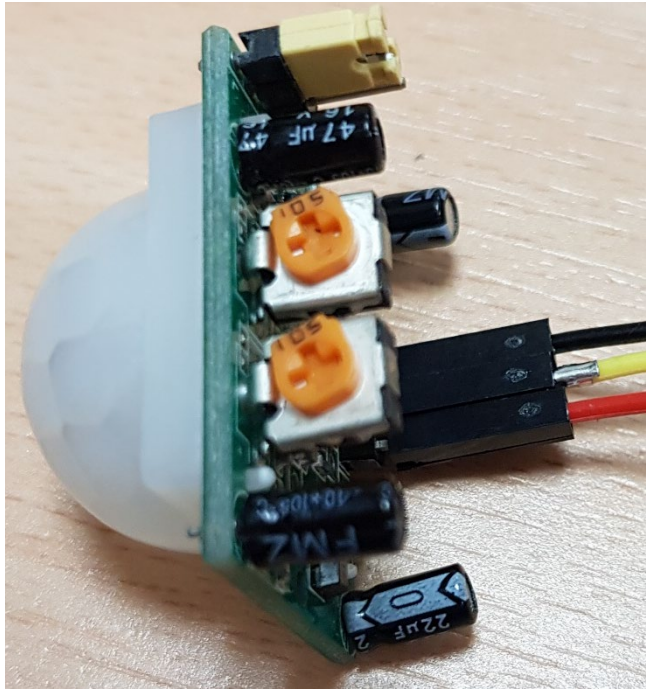


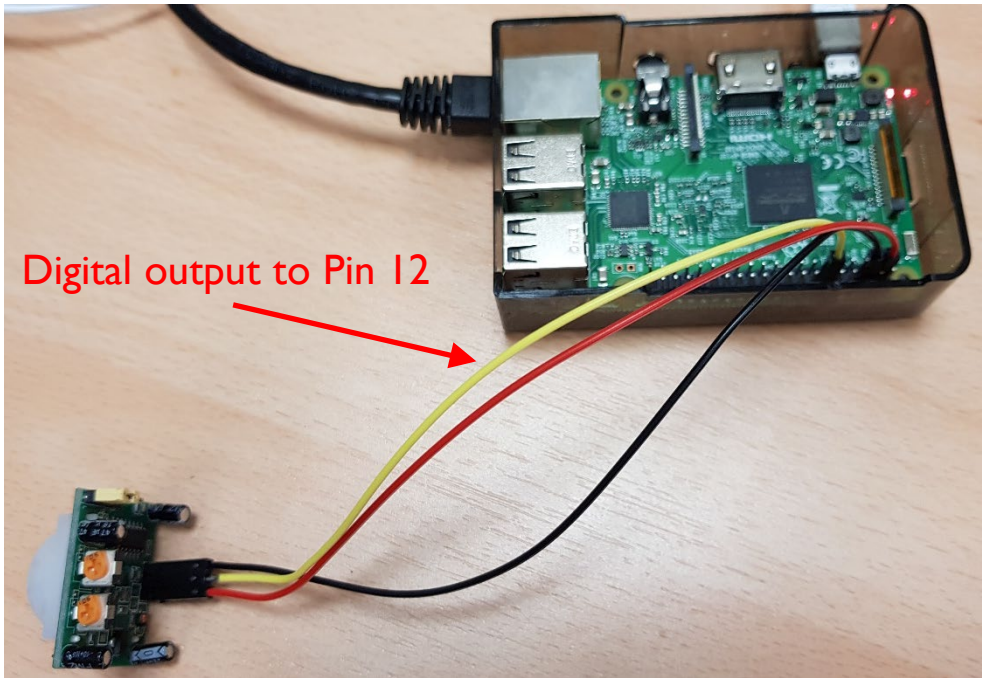
- The PIR sensor that we are using in this demonstration has an adjustable delay before firing (approximately 2-4 seconds) and adjustable sensitivity.



Passive Infrared (PIR) Motion Sensor (cont.)

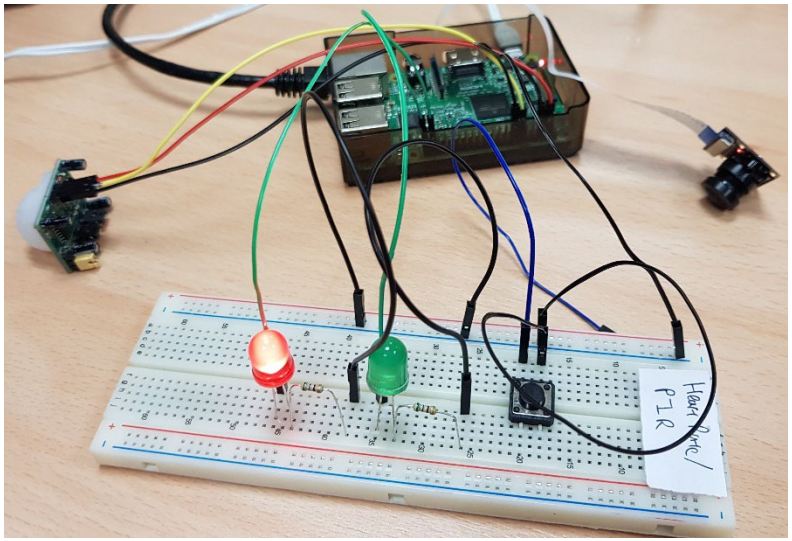
- The digital output pin of the PIR sensor in the circuit below is connected to GPIO pin 12.
- See sample source file `src14.py` for a simple demonstration of detecting motion with the PIR sensor.





Security Camera

- We can combine a PIR motion sensor with a Raspberry Pi camera module to create a simple security camera system:
  - Capture a still picture or record video when a motion is detected.
  - The picture or video can be emailed to user or uploaded to a cloud server.
- See sample source file `src15.py`.



Summary

- Micro:bit provides several Bluetooth services that allow an external device to interact with its sensors, buttons and even GPIO pins wirelessly.
- Raspberry Pi has onboard support for Bluetooth Low Energy (BLE) wireless communication that allows it to interact with a micro:bit's Bluetooth services.
- Micro:bit's Bluetooth UART service provides a flexible and powerful mechanism for a Raspberry Pi device to communicate with, and control, one or more micro:bit devices.

Q&A

Next Lecture…

- Learn about:
  - What is Service-Oriented Architecture.
  - What is RESTful web service.
  - How to create RESTful web service in Python with Flask and Connexion.
  - How to test RESTful web service in Postman.
  - How to consume RESTful web service in Python.
  - Persisting the data to a relational database.