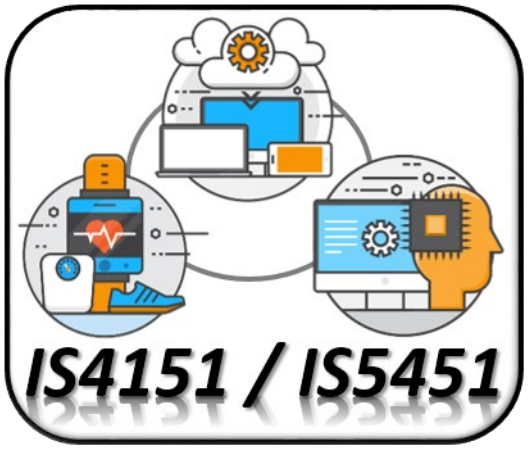


Lecture 6

Single-board Computer (I)

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

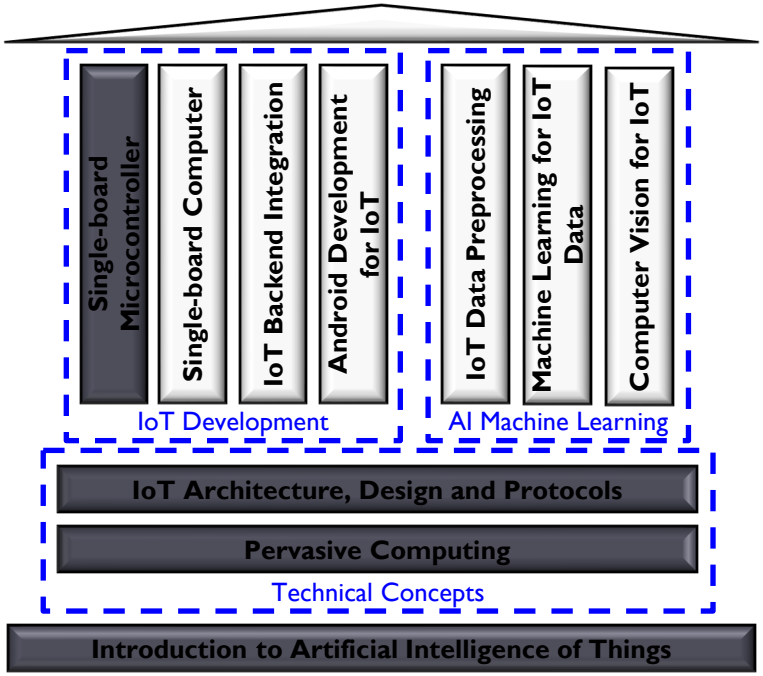
Lecturer: A/P TAN Wee Kek

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.

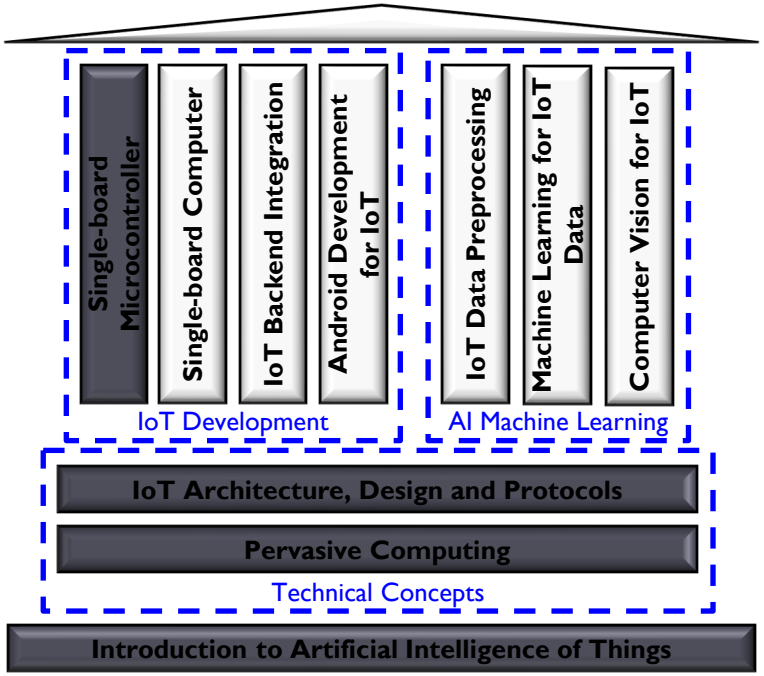
Quick Recap…

- In the preceding two lectures, we learnt:
  - How to build a node device using micro:bit, which offers us good computational and wireless data communication capabilities.
  - Micro:bit provides us with a suite of on-board sensors.
  - We can also control external peripheral devices using both digital and analogue signals.
- However, micro:bit possesses some fundamental limitations that restrict its role to node devices.



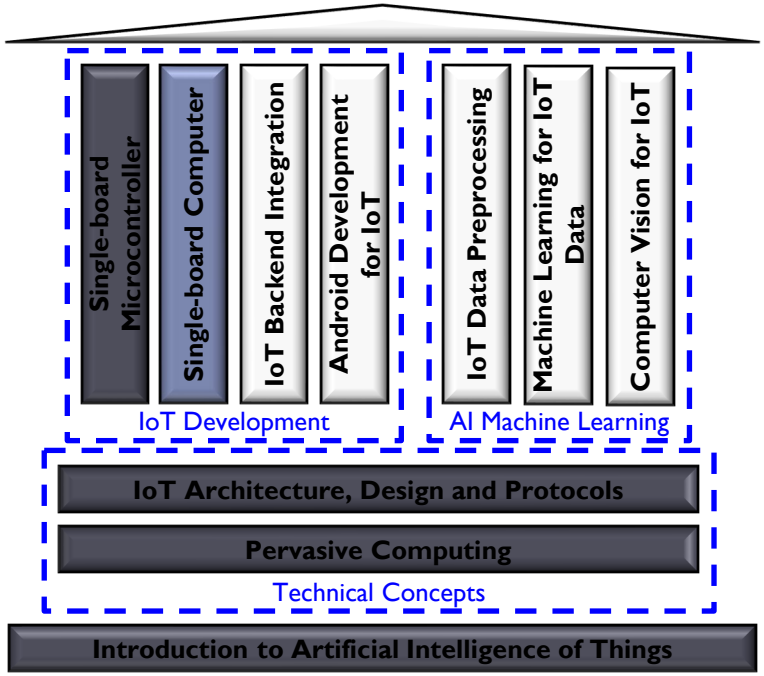
Quick Recap… (cont.)

- This lecture continues our learning journey to find out how to build a hub with Raspberry Rpi to control one or more node devices:
  - A hub can also perform edge processing over multiple node devices.
  - A hub can also extend into a fog processor.



Learning Objectives

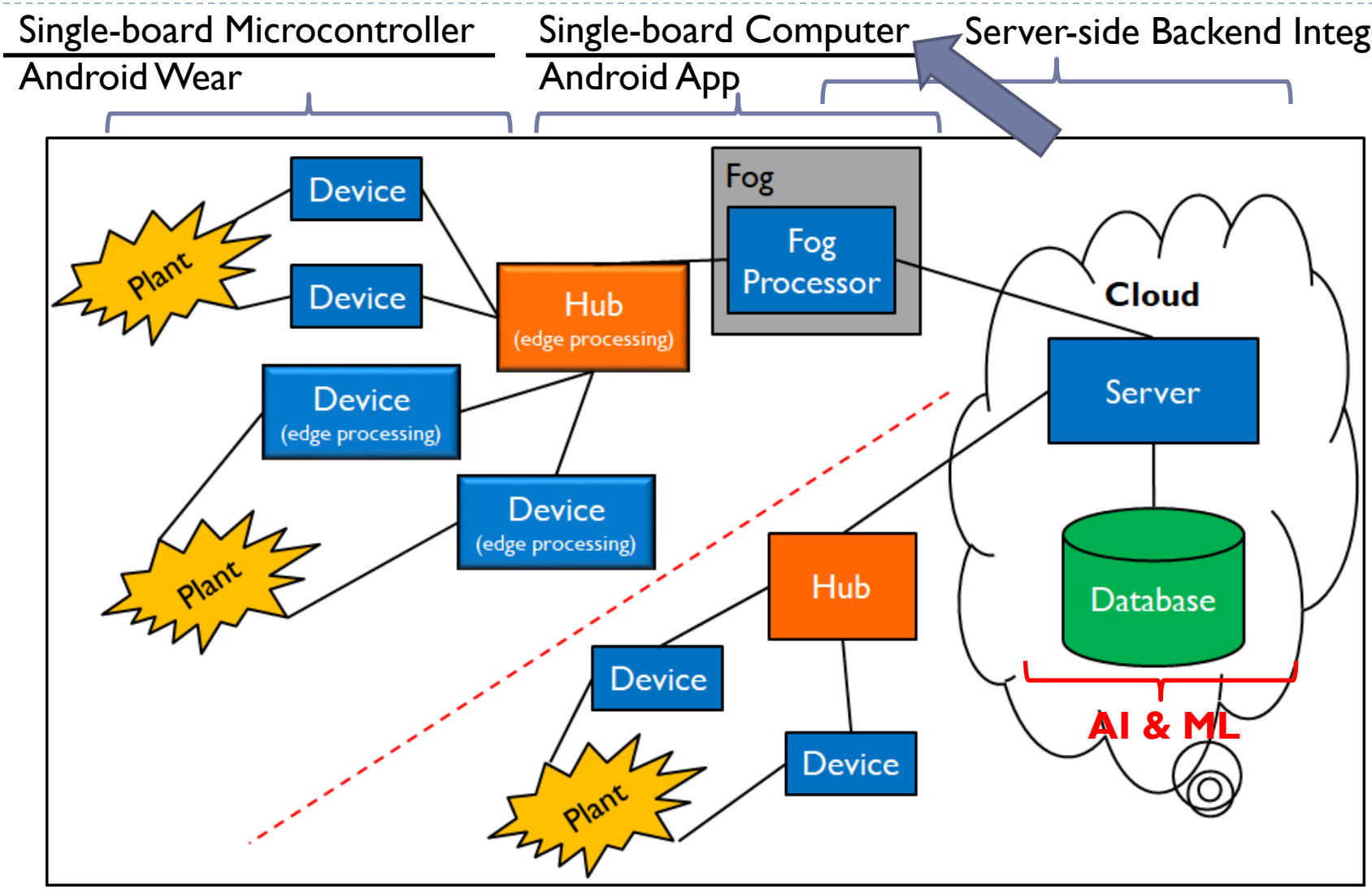
- At the end of this lecture, you should understand:
  - Overview of the Raspberry Pi single-board computer.
  - General purpose programming with Python on Raspberry Pi.
  - Basic electronic component concepts.
  - Basic Raspberry Pi GPIO programming using digital and analogue signal.



Readings

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

Technical Roadmap for IS4151/IS5451

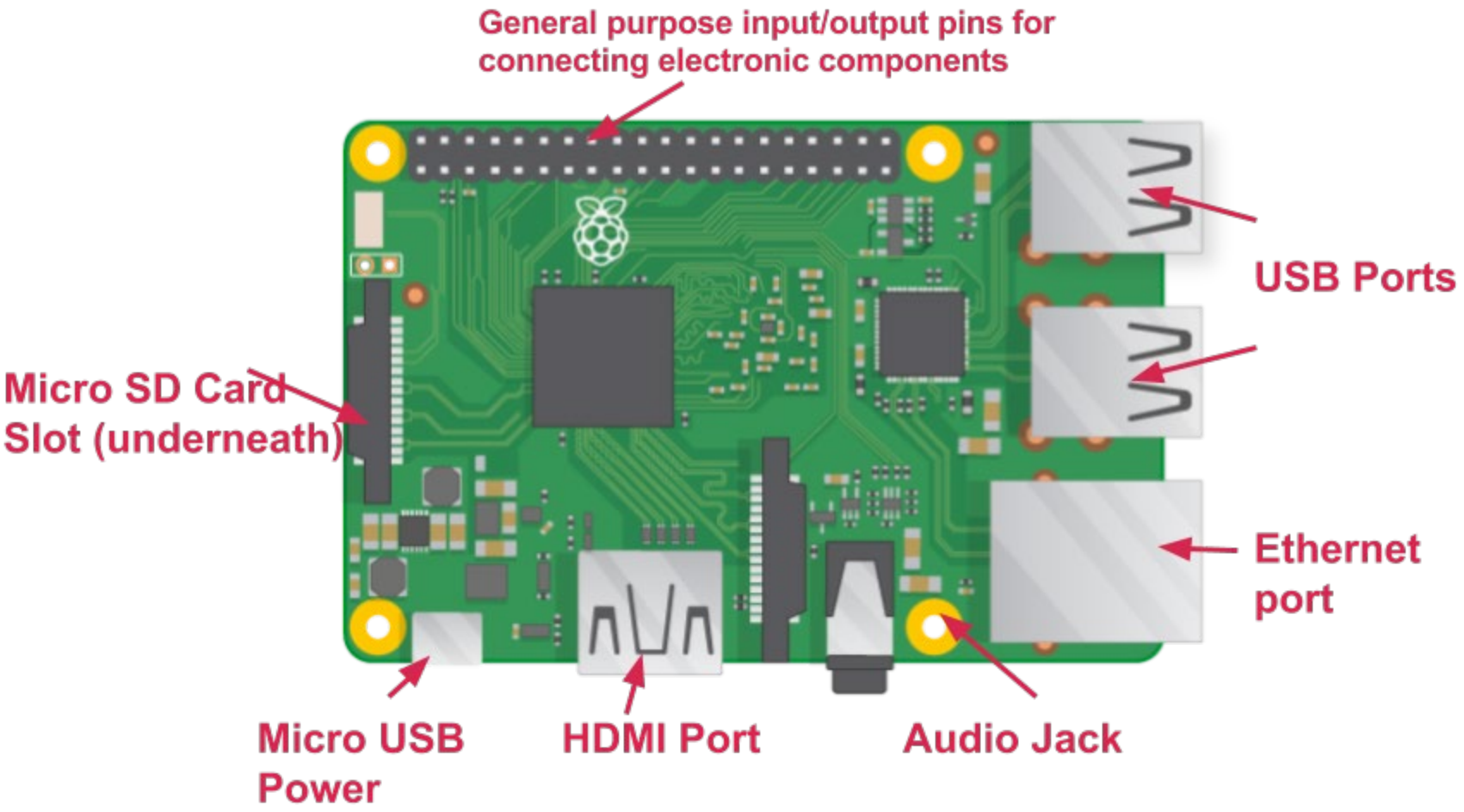


Introduction to the Raspberry Pi

Overview of the Raspberry Pi

- The **Raspberry Pi** is a single-board computer:
  - Raspberry Pi 3 Model B is the third-generation model with wireless LAN and Bluetooth connectivity.
  - Technical specification:
    - Quad Core 1.2GHz Broadcom BCM2837 64bit CPU with 1GB RAM.
    - BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on-board.
    - 40-pin extended GPIO.
    - 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 port and DSI display port for connecting a Raspberry Pi camera and touchscreen display, respectively.
    - Micro SD card slot for loading the operating system and storing data.

Overview of the Raspberry Pi (cont.)

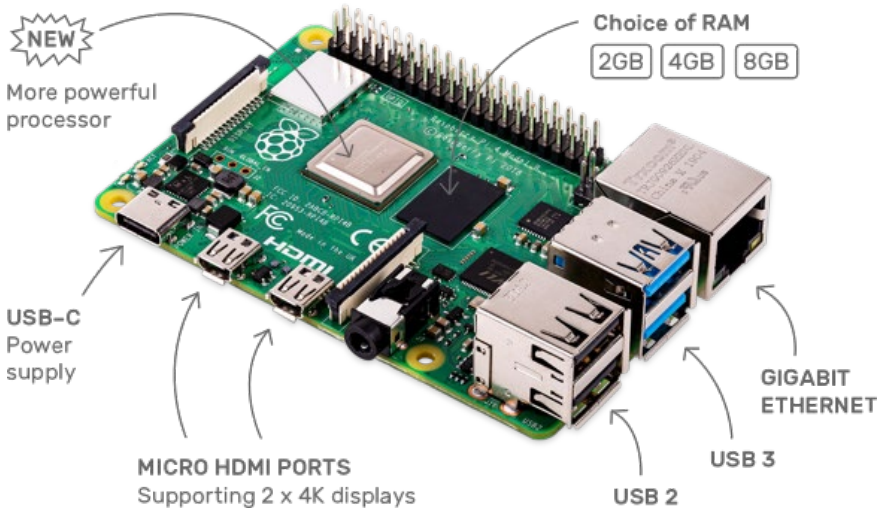


Overview of the Raspberry Pi (cont.)

- HDMI port:
  - For connecting the monitor (or projector) to display the output from the Pi.
  - If the monitor has speakers, you can also hear the sound.
- GPIO ports:
  - Allow you to connect electronic components such as LEDs and buttons to the Pi.
  - Used in conjunction with a full-sized breadboard.
- The HDMI, USB and Ethernet ports collectively is one of the two main differentiation factors between a single-board computer and microcontroller.

Overview of the Raspberry Pi (cont.)

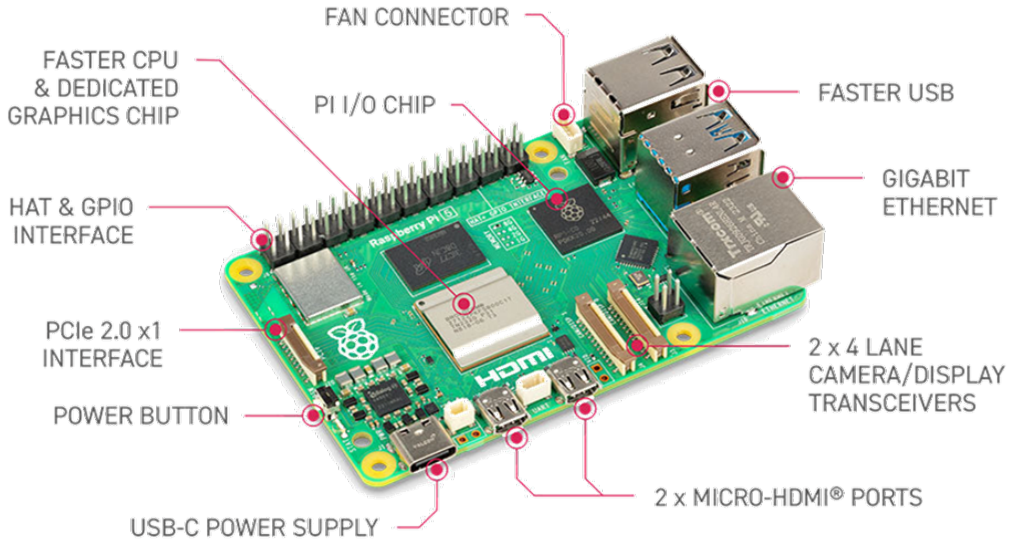
- Raspberry Pi 4 Model B:
  - Offers better specification with a starting price of about SGD $50.
  - Main processor features the Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz with up to 8GB RAM



  - https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

Overview of the Raspberry Pi (cont.)

- Raspberry Pi 5:
  - The latest generation offers 2–3x the speed of Raspberry Pi 4 and built with RP1 I/O controller priced at about SGD $100.
  - Main processor features the Broadcom BCM2712 quad-core Arm Cortex A76 processor @ 2.4GHz with up to 16GB RAM.

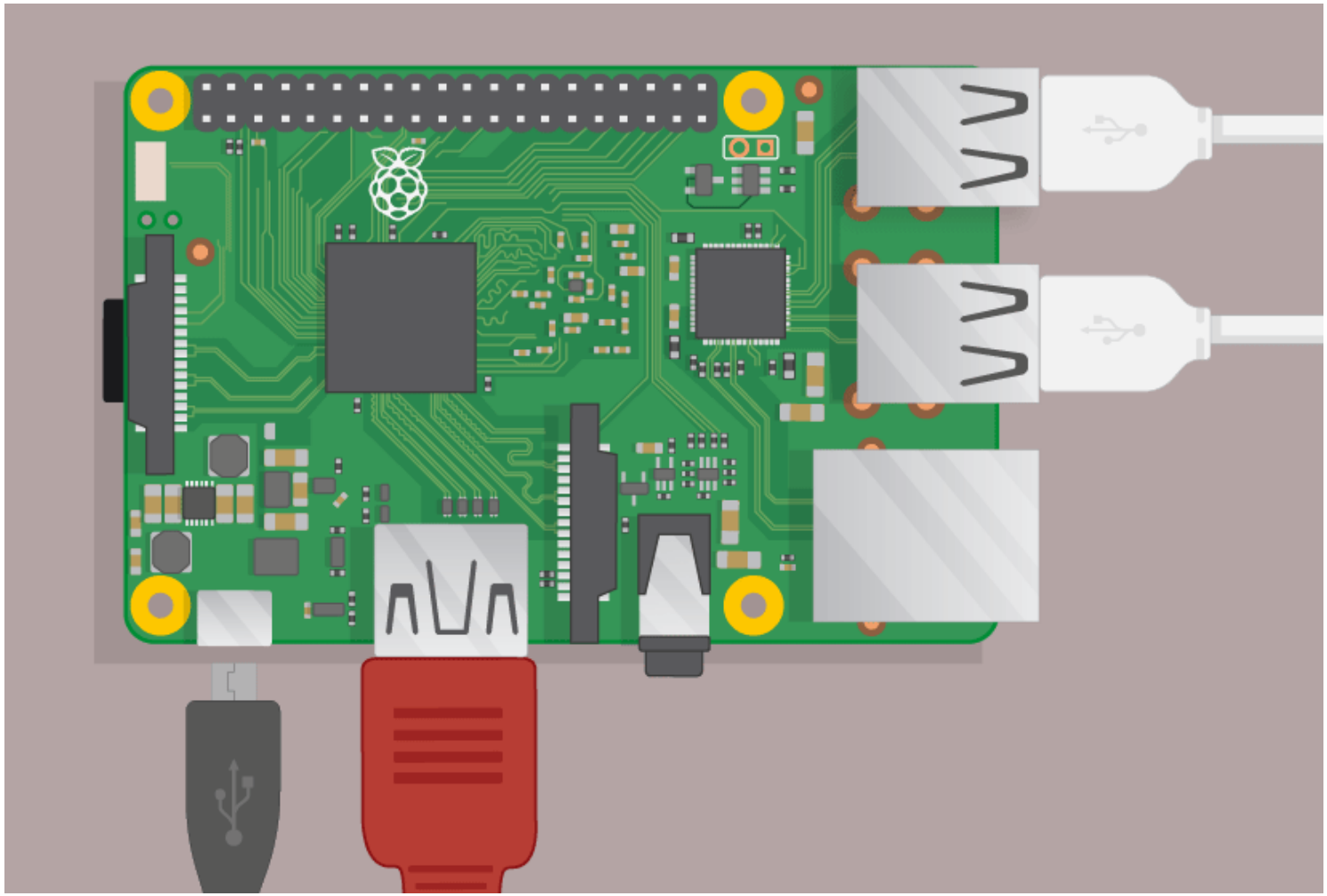


- https://www.raspberrypi.com/products/raspberry-pi-5/

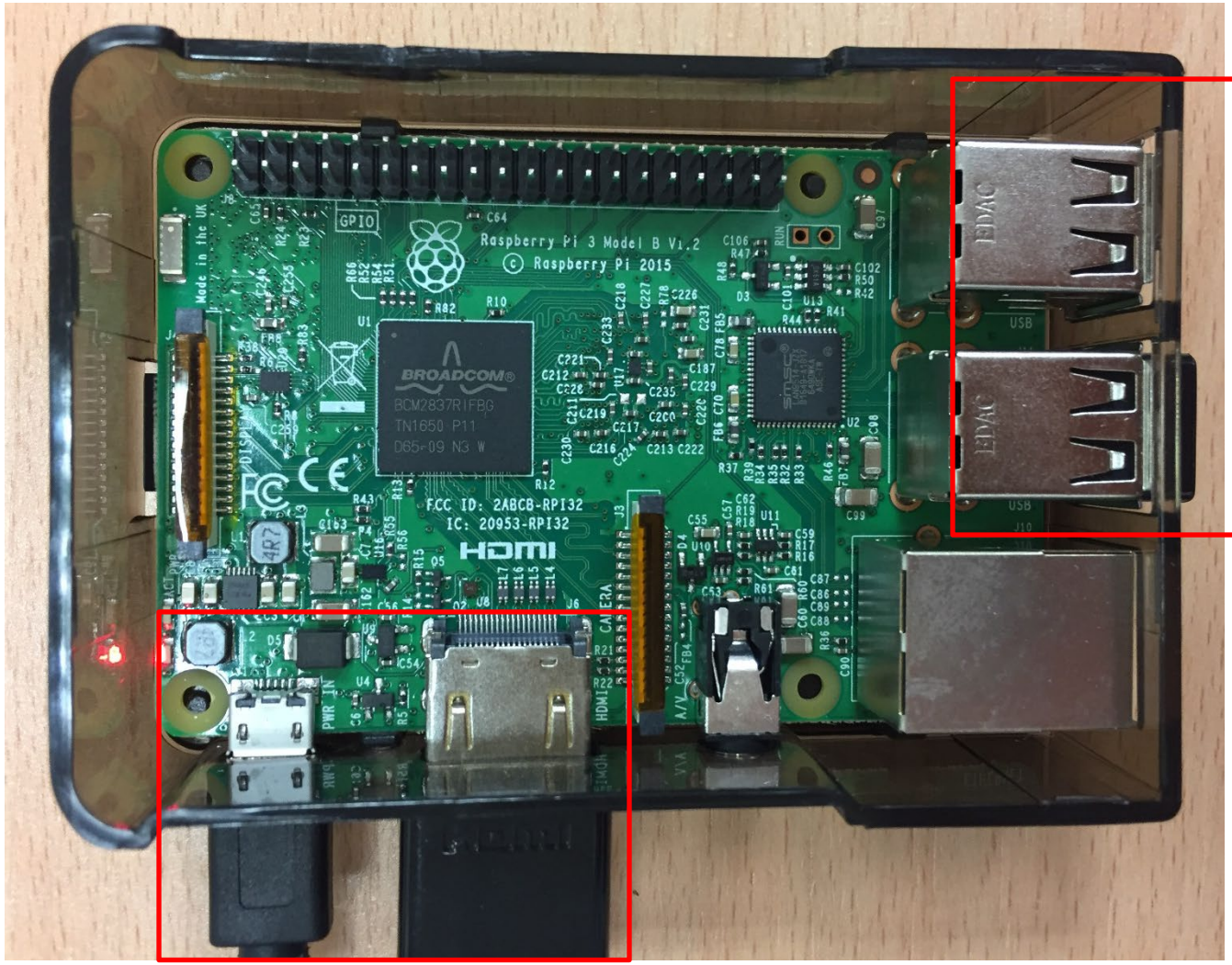
Connecting the Raspberry Pi

- Insert the bundled micro SD card containing Raspberry Pi OS into the card slot:
  - Raspberry Pi OS is a Debian-based operating system.
  - Debian itself is a Unix-like operating system.
- The operating system is the second main differentiation factor between a single-board computer and microcontroller.
- Connect peripheral devices:
  - Input – USB keyboard and mouse to any available USB ports.
  - Output – HDMI monitor or television to the HDMI port (can use a HDMI to VGA/DVI adapter if necessary).

Connecting the Raspberry Pi (cont.)



Connecting the Raspberry Pi (cont.)



Connecting the Raspberry Pi (cont.)

The SD card reader is located on the underside of the Pi

Connecting the Raspberry Pi (cont.)

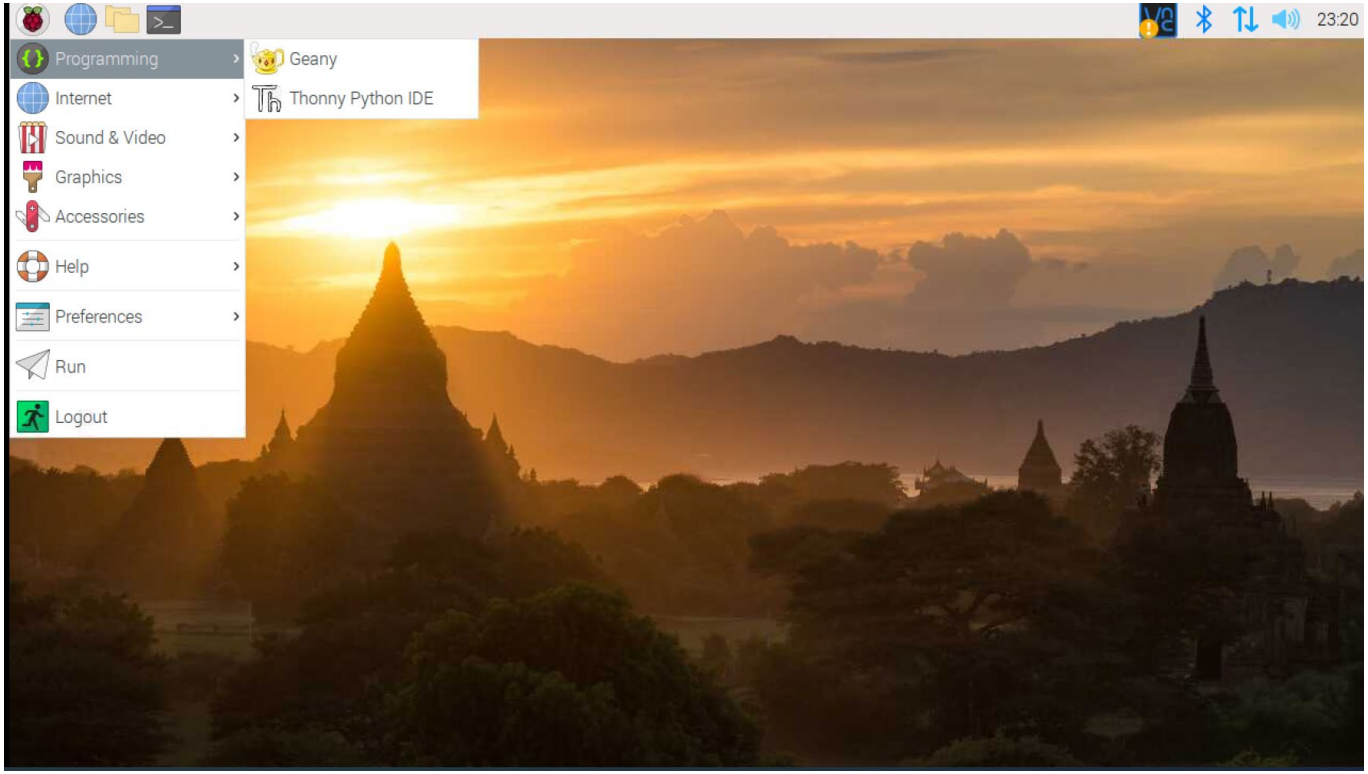
- Plug the power supply into a socket and connect it to the micro USB power port:
  - Ensure that the micro SD card is inserted before you connect the power supply.
  - Raspberry Pi boots up immediately after connecting the power.
  - You should see a red light on the Raspberry Pi and the device booting into Raspberry Pi OS.
  - After boot-up has completed, you will see a graphical desktop.

Connecting the Raspberry Pi (cont.)

The graphical desktop environment for Raspberry Pi OS

Programming on the Raspberry Pi

- Raspberry Pi works like a Mac or Linux machine.
- Can access Thonny Python IDE from the main menu "Programming" folder:



Programming on the Raspberry Pi (cont.)

- Can run a Python script from the Terminal using the command `python3`, e.g., `python3 --version`:
  - Note that both Python 2.x and Python 3.x are installed with Raspberry Pi OS by default.
  - Thus, the command `python` defaults to Python 2.x and you need to use `python3`.
  - For newer version of Raspberry Pi OS, the command `python` defaults to Python 3.x:
    - Use the `--version` switch to confirm whenever in doubt.

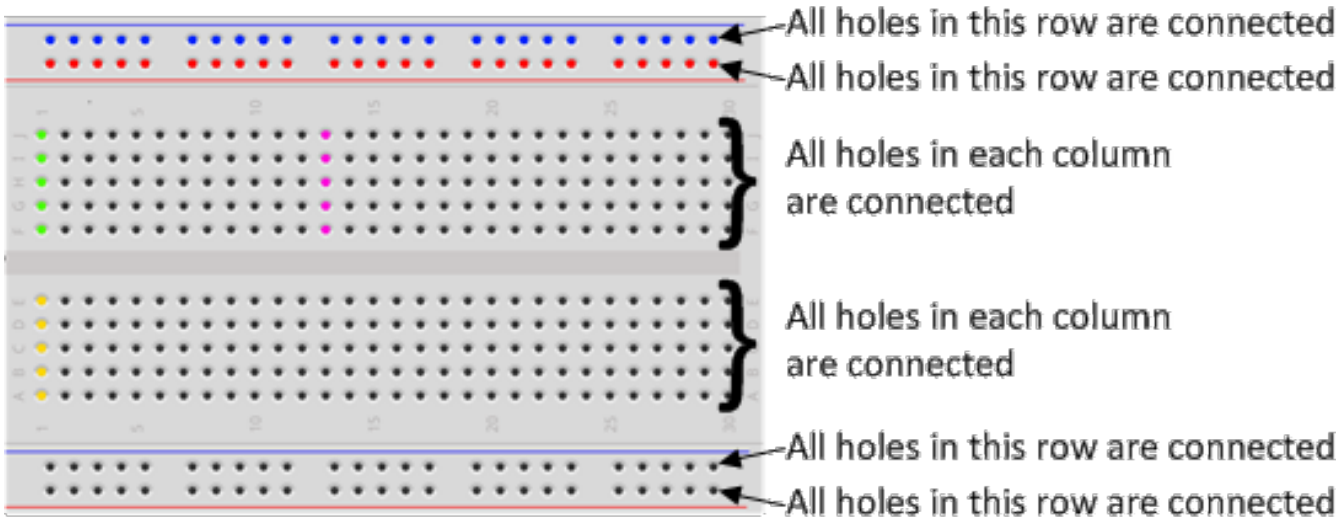
Raspberry Pi GPIO

- The Raspberry Pi is a good hardware prototyping tool.
- It has bi-directional GPIO pins that can be used to drive LEDs, spin motors or read button presses.
- Driving the Raspberry Pi's GPIO pins requires programming:
  - Python and C are two of the best choices.
  - We will be using Python with the RPi.GPIO library (a Python module to control the GPIO on a Raspberry Pi – http://sourceforge.net/projects/raspberry-gpio-python)

Basic Electronics

The Breadboard

- The breadboard is a way of connecting electronic components to each other without having to solder them together.
- It is often used to test a circuit design before creating a Printed Circuit Board (PCB).
- The holes on the breadboard are connected in patterns.



The LED

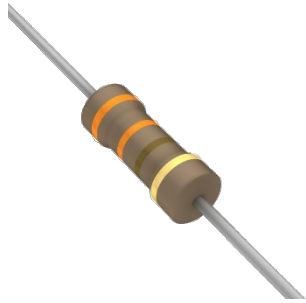
- Anatomy of a LED:
  - One leg is longer than the other.
  - The longer leg (known as the "anode") is always connected to the positive supply of the circuit.
  - The shorter leg (known as the "cathode") is connected to the negative side of the power supply, known as 'ground' or 0V.
- LED will only work if power is supplied in the correct way round, i.e., if the "polarity" is correct:
  - It will not be damaged if you connect the legs in the wrong way.
  - It will just not light up.

The Resistor

- Resistor is used to limit the amount of electricity going through a circuit.
- More specifically, it limits the amount of "current" that is allowed to flow through a circuit.
- The measure of resistance is called the Ohm (Ω), and the larger the resistance, the greater it limits the current.
- The value of a resistor is marked with coloured bands along the length of the resistor body:
  - See the next slide for an example of a 330Ω resistor.
  - The colour coding depends on how many bands are on a resistor.

The Resistor (cont.)

- If there are <u>four</u> colour bands, they will be Orange, Orange, Brown and Gold.
- If there are <u>five</u> bands, then the colours will be Orange, Orange, Black, Black and Brown.



Number of Bands

(•) 4 Band ( ) 5 Band ( ) 6 Band

Resistor Parameters

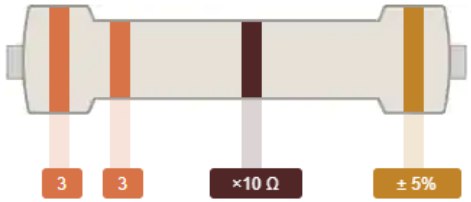
1st Band of Color: Orange 3

2nd Band of Color: Orange 3

Multiplier: Brown ×10 Ω

Tolerance: Gold ± 5%

Output



Resistor value:
330 Ohms 5%

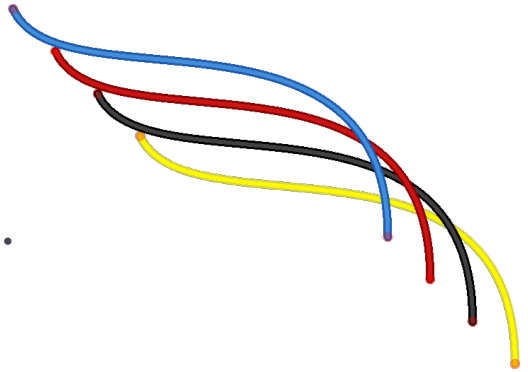
Source: https://www.digikey.sg/en/resources/conversion-calculators/conversion-calculator-resistor-color-code-4-band

The Resistor (cont.)

- A resistor has two legs too but it does not matter which way it is connected:
  - Current flows in both ways through them.
- You must **ALWAYS** use a resistor to connect a LED up to the GPIO pins of a Raspberry Pi:
  - The Raspberry Pi can only supply a small current (maximum is 50 mA as the pins use 3.3 V logic levels).
  - A LED will want to draw more, and if allowed to, it will burn out the Raspberry Pi.
  - Putting a resistor in the circuit will therefore ensure that only this small current will flow, and the Raspberry Pi will not be damaged.

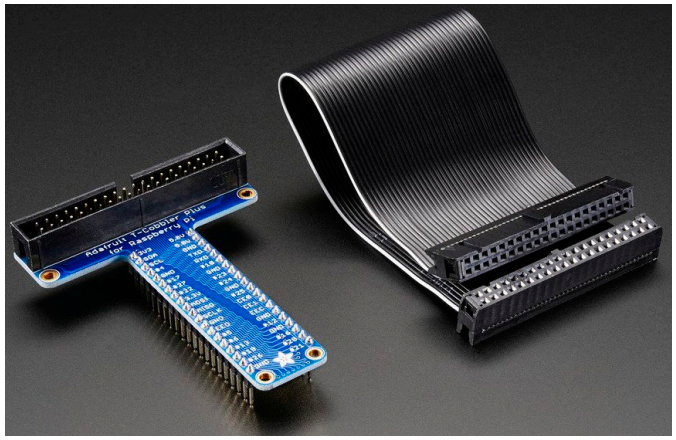
Jumper Wires

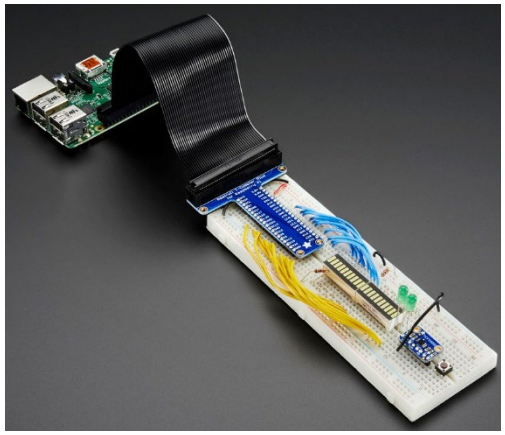
- Jumper wires are used on breadboards to "jump" from one connection to another.
- There are three main types of jumper wires.
- Male-female jumpers:
  - Have different connectors on each end.
  - The end with the "pin" will go into the Breadboard.
  - The end with the plastic socket will go onto the Raspberry Pi's GPIO pins.
- Male-male jumpers:
  - Have the same "pin" connectors on both ends.
  - Jump from one connection on a breadboard to another location on the breadboard.



Jumper Wires (cont.)

- Female-female jumpers:
  - Have the same plastic sockets on both ends.
  - Jump from one pin on the Raspberry Pi to another pin.
  - Less often used.
- If you use an edge connector with a T-cobbler:
  - The T-cobbler is connected to the breadboard.
  - Thus, you only need to use male-male jumpers.



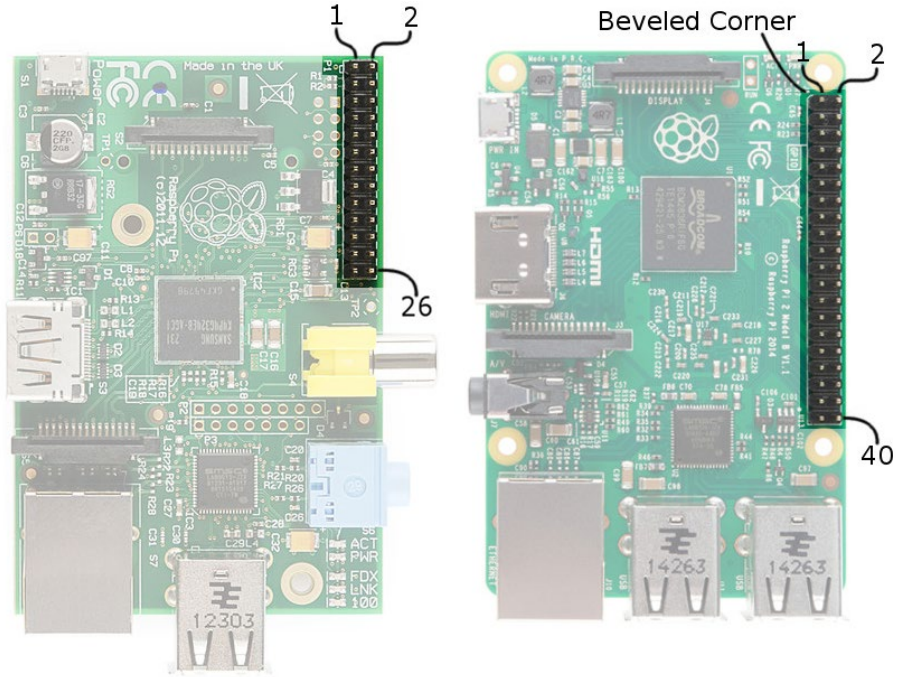


GPIO Pins

- GPIO stands for General Purpose Input Output.
- GPIO allows a single-board computer or microcontroller to control and monitor the outside world by connecting to electronic circuits.
- GPIO pins can work in two modes:
  - Output pin is able to control LEDs (turning on or off), motors or other devices.
  - Input pin is able to detect whether a switch has been pressed, temperature or light.
  - Recall that micro:bit provides dedicated read/write pin blocks that do not require us to specify the pin mode.
  - In Raspberry Pi, we need to specify the pin mode explicitly.

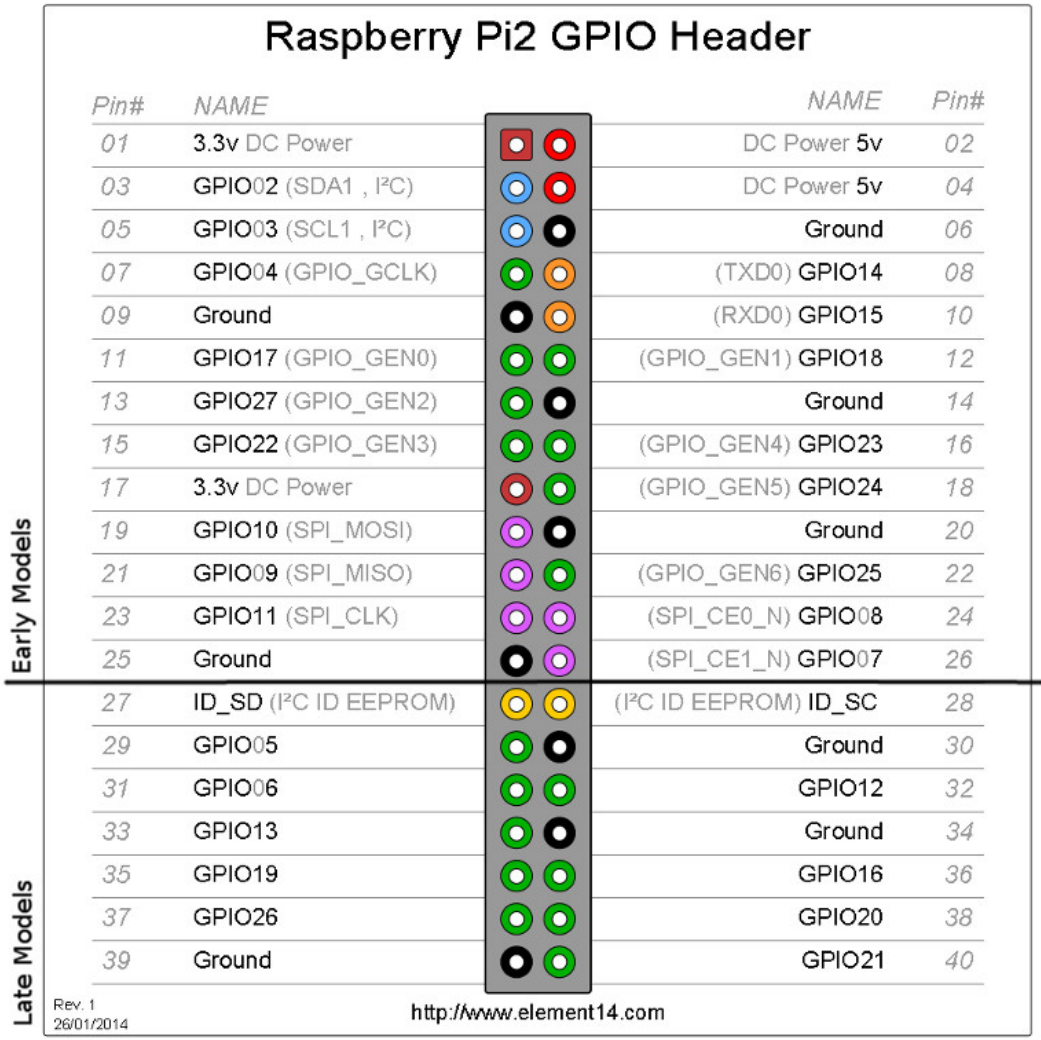
GPIO Pinout

- The Raspberry Pi opens up its GPIO over a standard male header on the board.
- Over the years, the header has expanded from 26 pins to 40 pins while maintaining the original pinout.



GPIO Pinout (cont.)

Odd numbered pins are on the left side



Even numbered pins are on the right side

GPIO Pinout (cont.)

Wedge Silk	Python (BCM)	WiringPi GPIO	Name	P1 Pin Number		Name	WiringPi GPIO	Python (BCM)	Wedge Silk
			3.3v DC Power	1	2	5v DC Power			
SDA		8	GPIO02 (SDA1, I2C)	3	4	5v DC Power			
SCL		9	GPIO03 (SCL1, I2C)	5	6	Ground			
G4	4	7	GPIO04 (GPIO_GCLK)	7	8	GPIO14 (TXD0)	15		TXO
			Ground	9	10	GPIO15 (RXD0)	16		RXI
G17	17	0	GPIO17 (GPIO_GEN0)	11	12	GPIO18 (GPIO_GEN1)	1	18	G18
G27	27	2	GPIO27 (GPIO_GEN2)	13	14	Ground			
G22	22	3	GPIO22 (GPIO_GEN3)	15	16	GPIO23 (GPIO_GEN4)	4	23	G23
			3.3v DC Power	17	18	GPIO24 (GPIO_GEN5)	5	24	G24
MOSI		12	GPIO10 (SPI_MOSI)	19	20	Ground			
MISO		13	GPIO09 (SPI_MISO)	21	22	GPIO25 (GPIO_GEN6)	6	25	G25
		(no worky 14)	GPIO11 (SPI_CLK)	23	24	GPIO08 (SPI_CE0_N)	10		CD0
			Ground	25	26	GPIO07 (SPI_CE1_N)	11		CE1
IDSD		30	ID_SD (I2C ID EEPROM)	27	28	ID_SC (I2C ID EEPROM)	31		IDSC
G05	5	21	GPIO05	29	30	Ground			
G6	6	22	GPIO06	31	32	GPIO12	26	12	G12
G13	13	23	GPIO13	33	34	Ground			
G19	19	24	GPIO19	35	36	GPIO16	27	16	G16
G26	26	25	GPIO26	37	38	GPIO20	28	20	G20
			Ground	39	40	GPIO21	29	21	G21

GPIO Pinout (cont.)

- **GPIO**:
  - Standard pins that are simply used to turn devices on and off.
  - E.g., a LED.
- Other than the standard pins, there are some pins that reference I2C, SPI and UART.
- **I2C (Inter-Integrated Circuit)**:
  - There are two pins that allow you to connect and talk to hardware modules that support the I2C protocol.
  - Use to implement an I2C bus with the GPIO.
  - Many peripheral devices, e.g., RTC (Real Time Clock), can communicate via I2C.

GPIO Pinout (cont.)

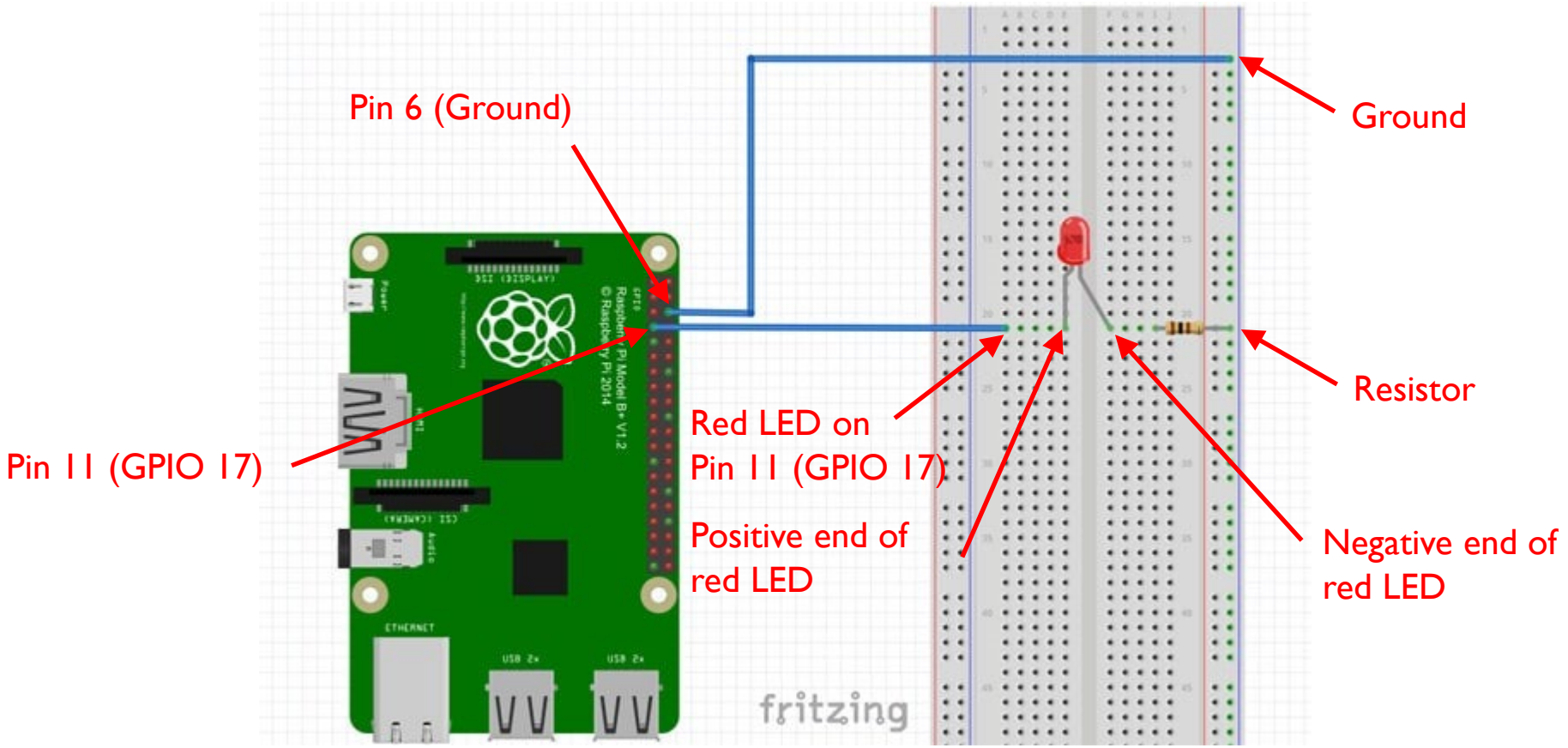
- **SPI (Serial Peripheral Interface Bus)**:
  - These pins can be used to connect and talk to SPI devices.
  - Similar to I2C but uses a different protocol.
  - Use to implement a SPI bus.
  - Many peripheral devices, e.g., ADC (Analog Digital Converter), can communicate via SPI.
- **UART (Universal asynchronous receiver / transmitter)**:
  - These are the serial pins used to communicate with other devices.
- **CLK**:
  - For generating a clock signal.

GPIO Pinout (cont.)

- **DC Power**:
  - The power pins draw power directly from the Raspberry Pi.
  - There are two types of power pins – 3.3V and 5.0V
    - Logic pins are 3.3V
- **GND**:
  - These pins are used to ground electronic devices.
  - It does not matter which pin you use as they are all connected to the same line.

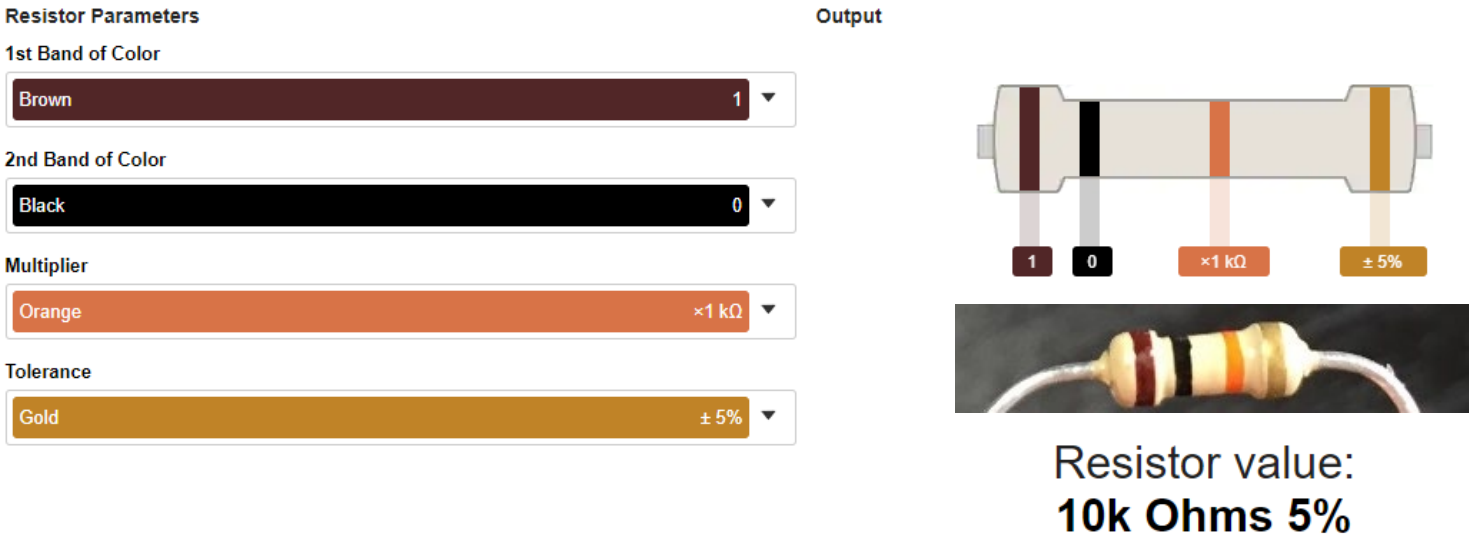
GPIO Programming Using Digital Signal

Basic LED Blinker

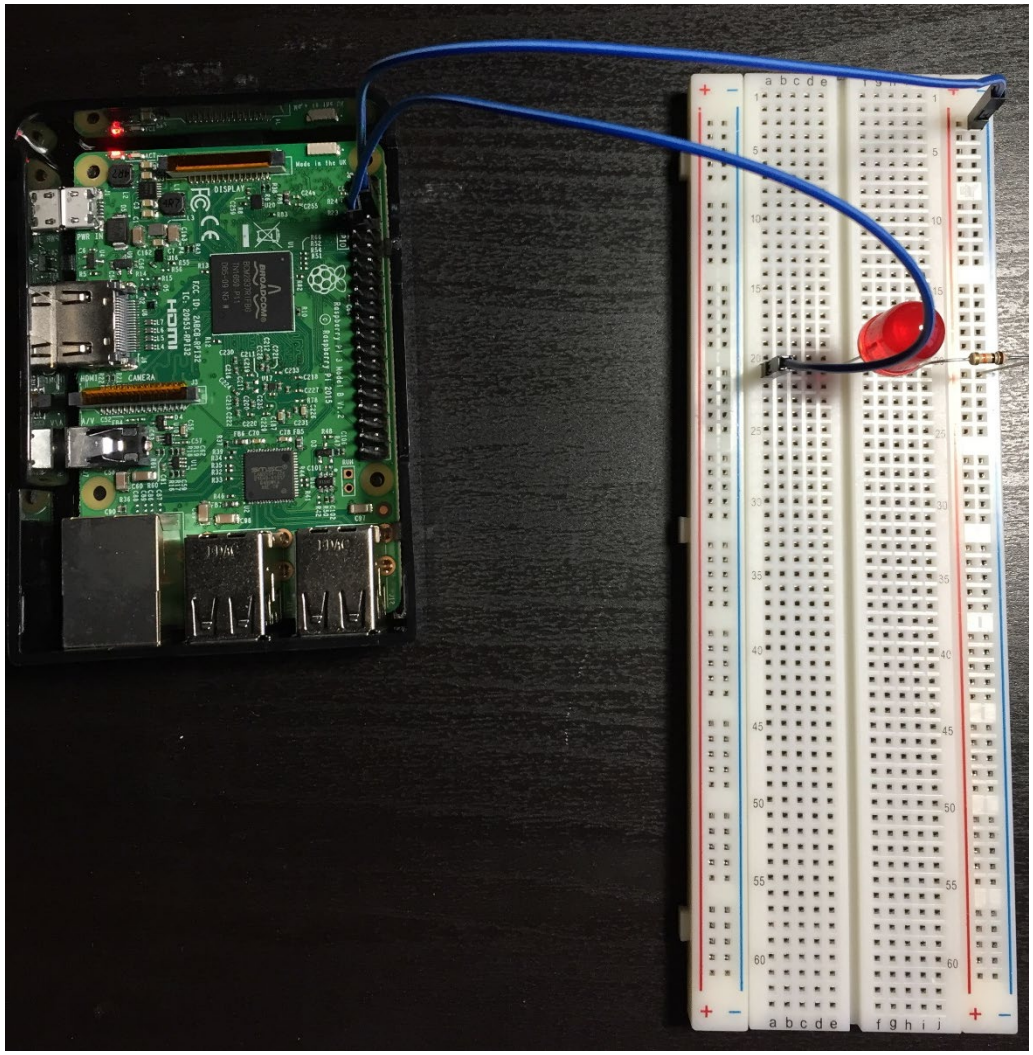


Basic LED Blinker (cont.)

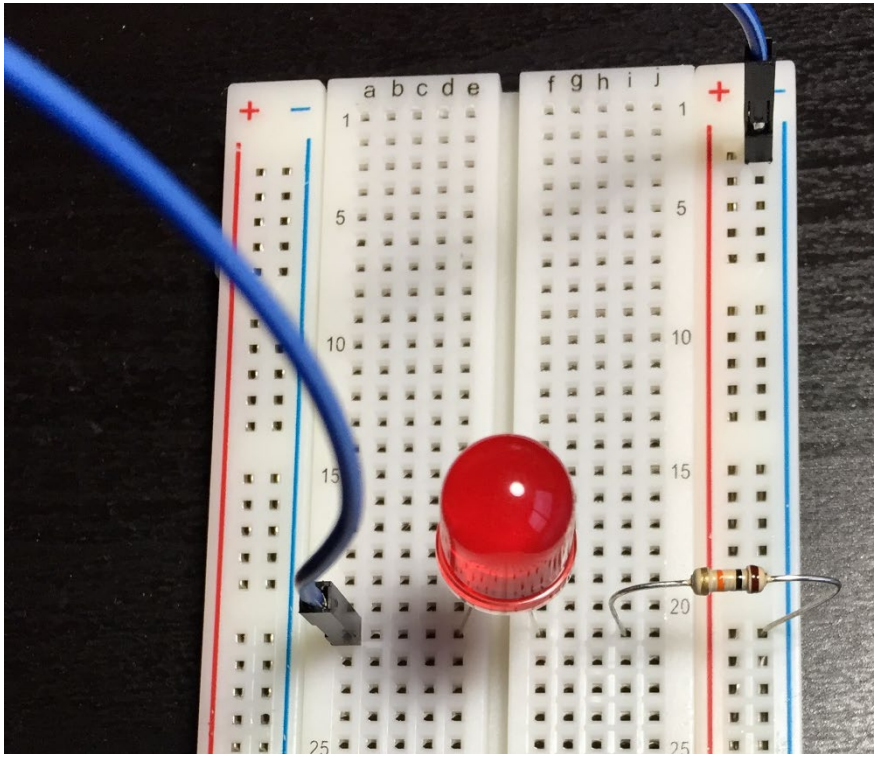
- Connecting red LED:
  - Positive end to Pin 11 (GPIO 17):
    - Pin 11 is the PI connector pin number
    - GPIO 17 is the Broadcom chip-specific number
  - Negative end to a resistor that connects to a ground pin.
- We are using a 10k Ohms resistor with 5% tolerance:



Basic LED Blinker (cont.)



Basic LED Blinker (cont.)



- Replacing the 10k Ohms resistor with a 560 Ohms resistor will provide a much brighter LED.
- This is because the amount of current flowing through the circuit will be higher.

Basic LED Blinker (cont.)

```python
# import the GPIO and time package
import RPi.GPIO as GPIO
import time



# Pin  Definitions
ledPin = 11



# Pin Setup
GPIO.setmode(GPIO.BOARD)
GPIO.setup(ledPin, GPIO.OUT)



print("Program running... Press CTRL+C to exit")

try:
    for i in range(50):
        GPIO.output(ledPin, True)
        time.sleep(1)
        GPIO.output(ledPin, False)
        time.sleep(1)

except KeyboardInterrupt:
    print("Program terminated!")

finally:
    GPIO.cleanup()
```

Blinks LED 50 times only

src01.py

Basic LED Blinker (cont.)

```python
# import the GPIO and time package
import RPi.GPIO as GPIO
import time



# Pin  Definitions
ledPin = 11



# Pin Setup
GPIO.setmode(GPIO.BOARD)
GPIO.setup(ledPin, GPIO.OUT)



print("Program running... Press CTRL+C to exit")

ledStatus = True
GPIO.output(ledPin, ledStatus)

try:
    while True:
        time.sleep(1)
        ledStatus = not ledStatus
        GPIO.output(ledPin, ledStatus)

except KeyboardInterrupt:
    print("Program terminated!")

finally:
    GPIO.cleanup()
```

Blinks LED infinitely until user press Ctrl+C

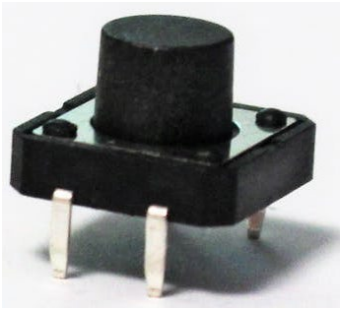
src02.py

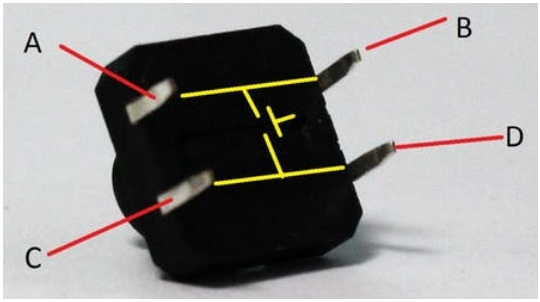
Basic LED Blinker (cont.)

- Observe that in the sample source code `src01.py` and `src02.py`:
  - We have set Pin 11 as an output pin – `GPIO.setup(ledPin, GPIO.OUT)`
  - `GPIO.output(ledPin, True)` – Writes a 1 to turn the LED on.
  - `GPIO.output(ledPin, False)` – Writes a 0 to turn the LED off.
  - This is equivalent to micro:bit's `digitalWritePin` block.

Push Button

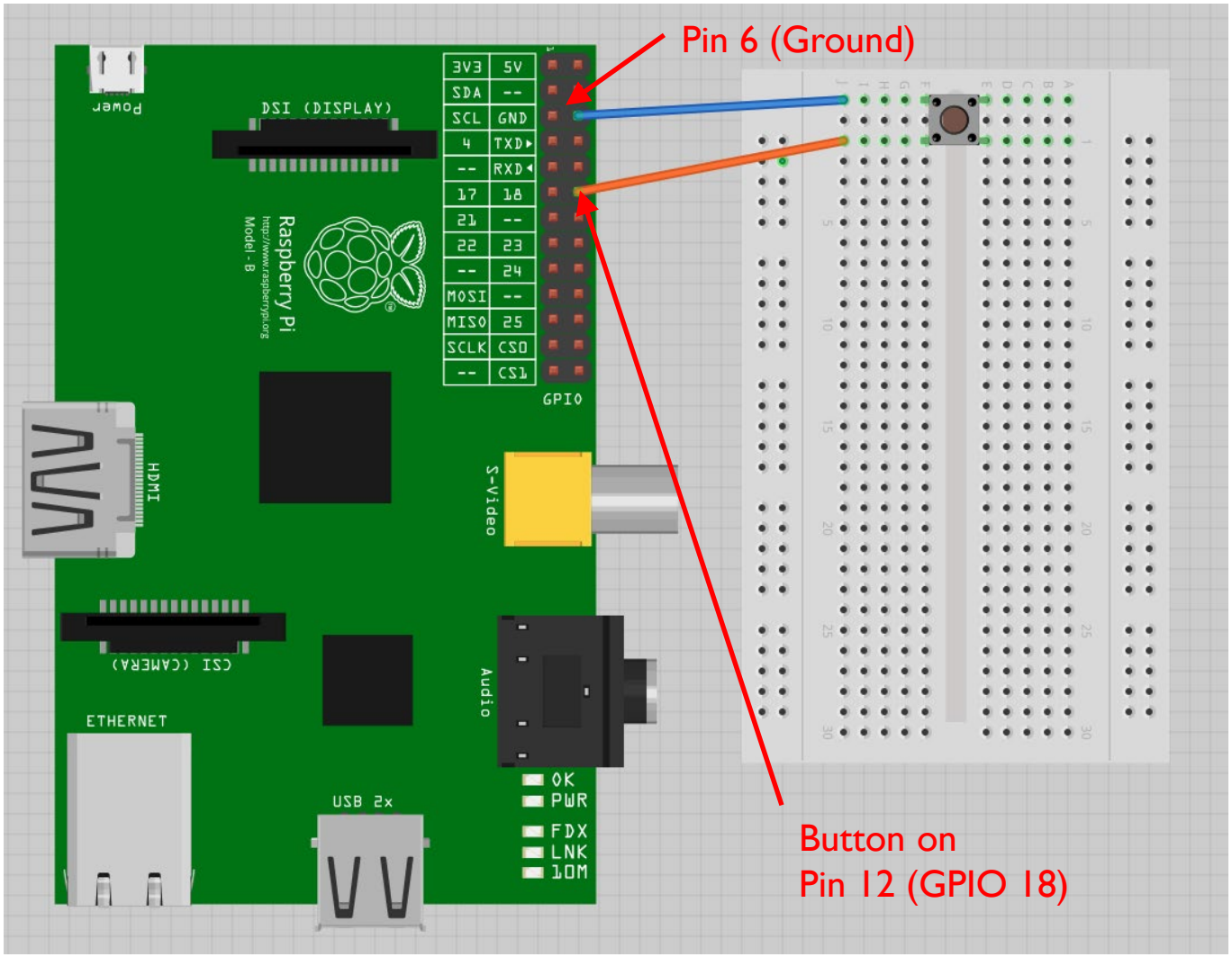
- There is direct connection between the following pairs of pins regardless of whether the button is pressed or not:
  - Between A and B.
  - Between C and D.
- When button is pressed:
  - A and C are getting shorted
  - The same goes for B and D.
  - Actually, all pins are getting shorted.



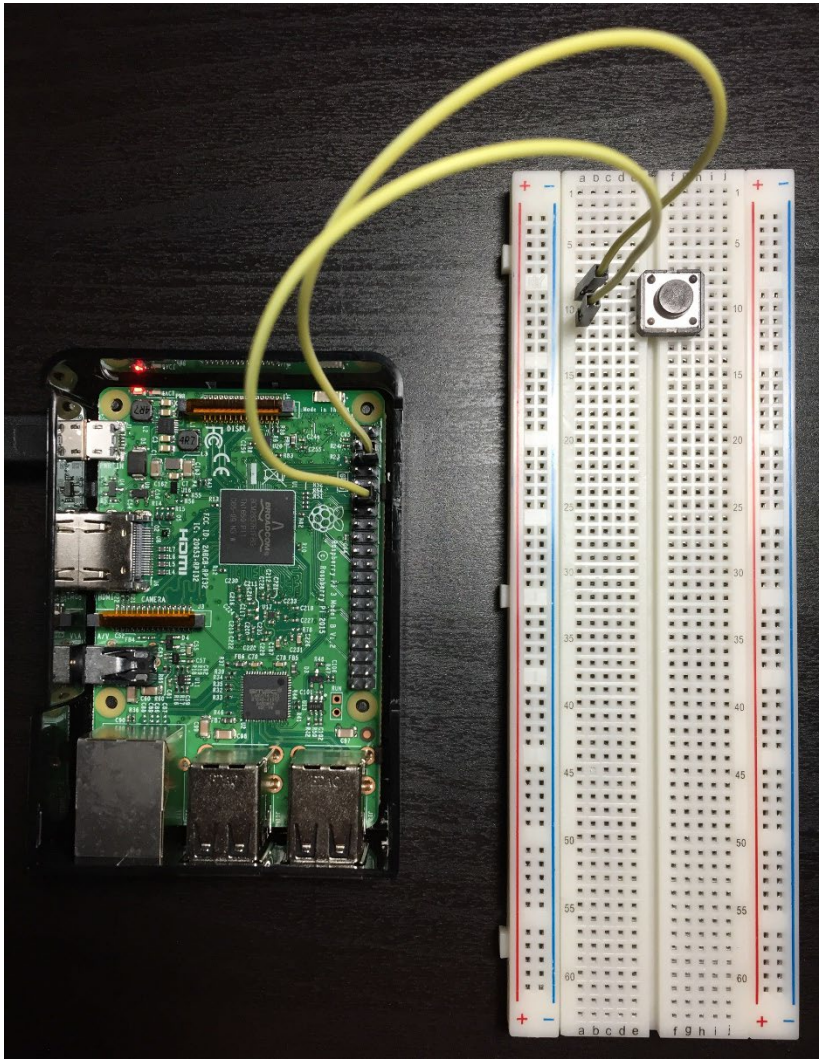


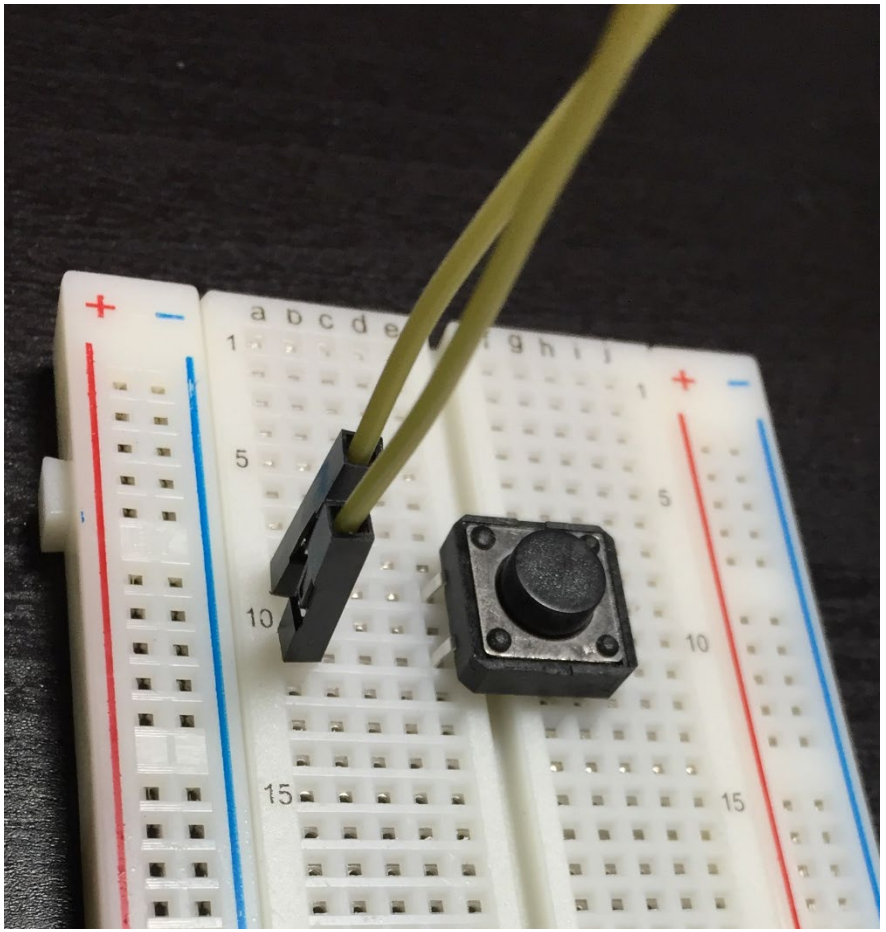
- Thus, to use the push button as a switch, use either:
  - A-C pair or B-D pair ➔ More commonly used.
  - A-D pair or B-C pair.

Detecting Push Button State



Detecting Push Button State (cont.)





Detecting Push Button State (cont.)

```python
# import the GPIO and time package
import RPi.GPIO as GPIO
import time



# Pin Definitions
butPin = 12



# Pin Setup
GPIO.setmode(GPIO.BOARD)
GPIO.setup(butPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)



print("Program running... Press CTRL+C to exit")

try:
    while True:
        if GPIO.input(butPin):
            print("Button released!")
        else:
            print("Button pressed!")
        time.sleep(0.25)

except KeyboardInterrupt:
    print("Program terminated!")

finally:
    GPIO.cleanup()
```

src03.py

Detecting Push Button State (cont.)

- Observe that in the sample source code `src03.py`:
  - We have set Pin 12 as an input pin – `GPIO.setup(butPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)`
  - In the button setup, we need to enable the Pi's internal pull-up or pull-down resistor:
  - In our circuit, Pin 12 is connected to GND and will read low when it is closed.
  - Thus, we enable the pull-up register so that when it is open, it defaults to the high state.
  - If we do not enable the pull-up register, the status of the pin would not be clearly defined, i.e., floating:
    - The pin is susceptible to random electromagnetic radiation or static from the surrounding.

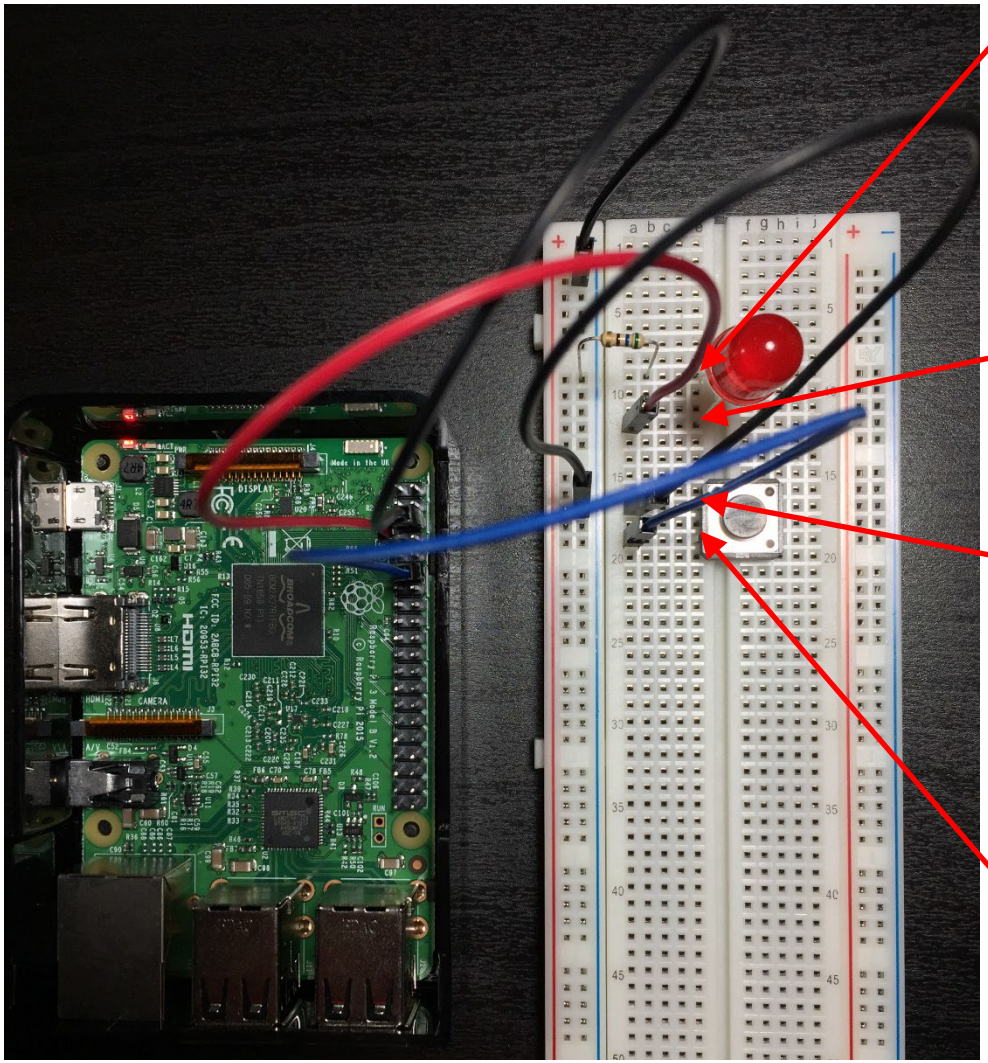
Detecting Push Button State (cont.)

- If we remove the `pull_up_down` option, the program will report the button status randomly when the button is not pressed.
- If our GPIO Pin is connected to a power pin, it will read high when the circuit is closed.
- In this case, we will enable the pull-down register so that when the circuit is open, it defaults to the low state.
- When the GPIO Pin is connected to GND, and we somehow set the `pull_up_down` option to pull-down:
  - The program will always report the button status as pressed regardless of whether the button is actually pressed.
  - This is because the pin defaults to the low state when the circuit is open.

Detecting Push Button State (cont.)

- `GPIO.input(ledPin)` – Reads either a 0 or 1.
- This is equivalent to micro:bit's `digitalReadPin` block.

Controlling LED with Push Button Switch



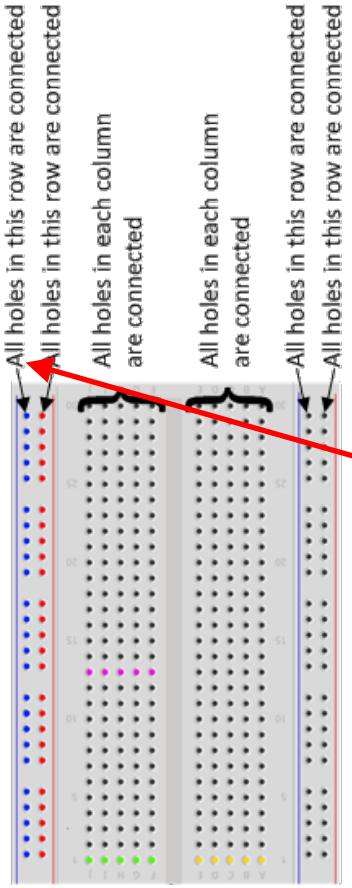
Negative end of red LED to resistor and then to Pin 6 (Ground) via Black M-F jumper

Positive end of red LED to Pin 11 (GPIO 17) via Red jumper

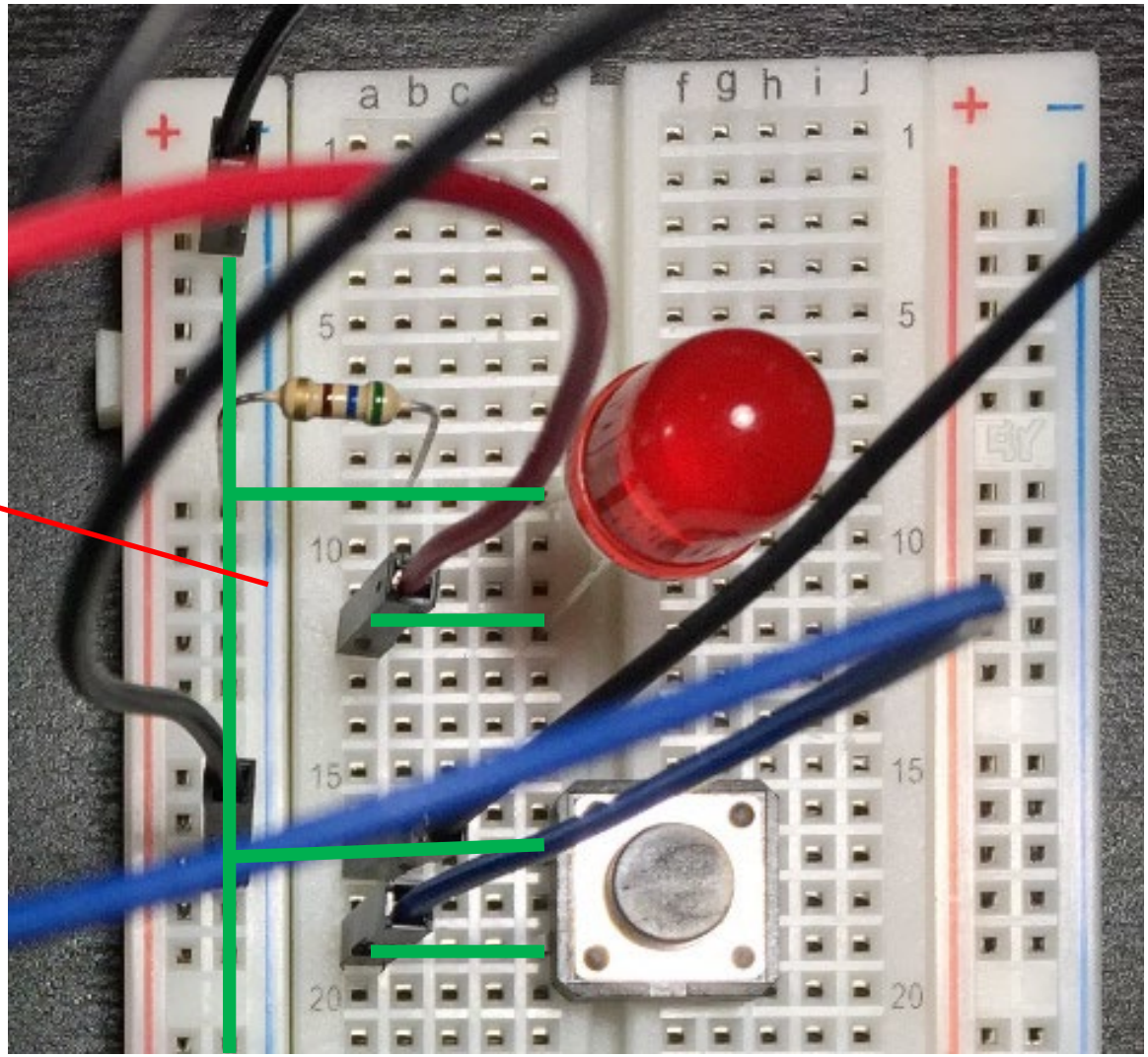
Pin A of Button to Pin 6 (Ground) via Black M-M jumper

Pin C of Button to Pin 12 (GPIO 18) via Blue jumper

Controlling LED with Push Button Switch (cont.)



Note the use of the ground row on the breadboard to share a common ground pin (i.e., Pin 6) among multiple devices.

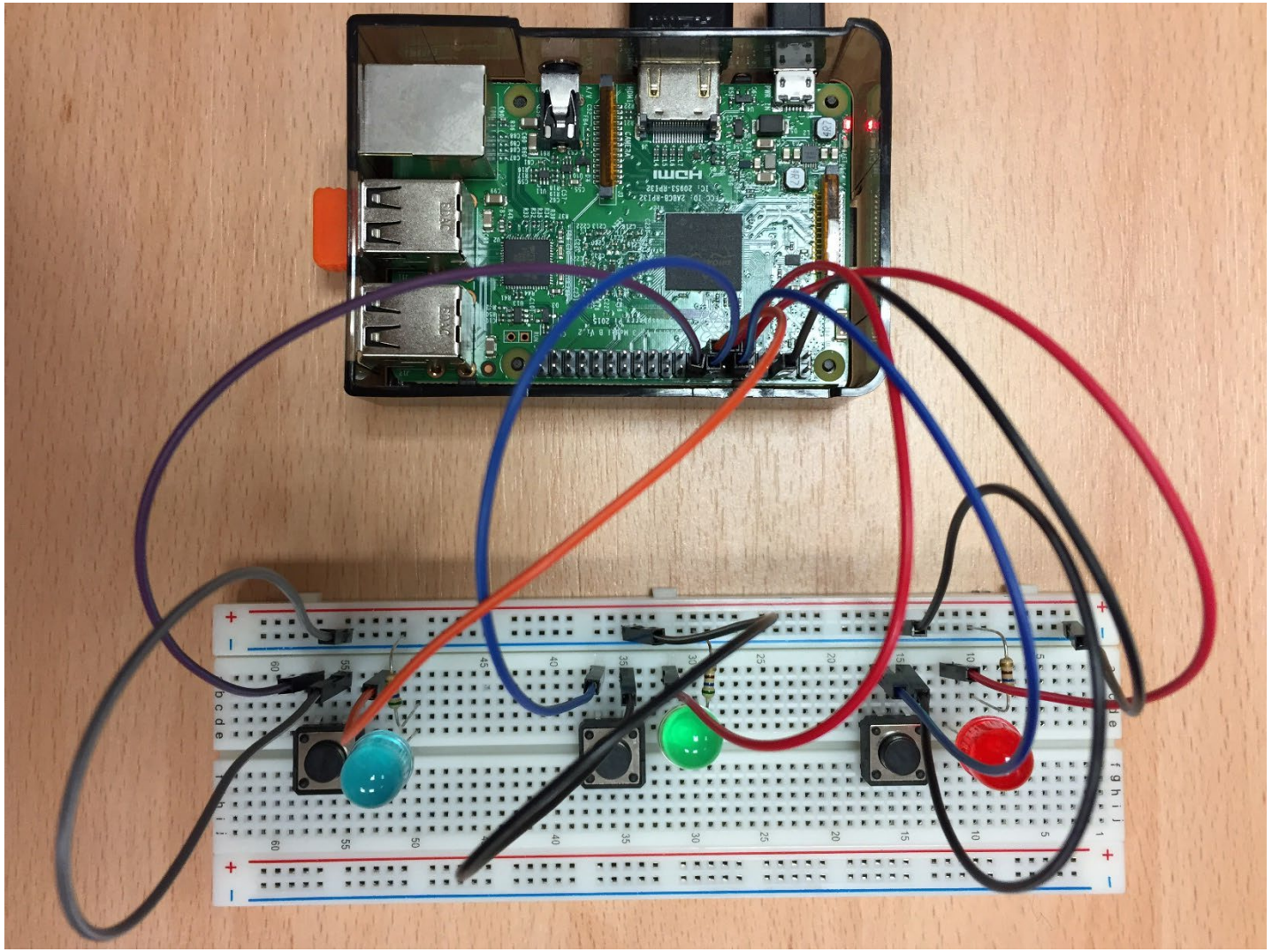


Controlling LED with Push Button Switch (cont.)

```
# import the GPIO and time package
import RPi.GPIO as GPIO
import time



# Pin  Definitions
ledPin = 11
butPin = 12



# Pin Setup
GPIO.setmode(GPIO.BOARD)
GPIO.setup(ledPin, GPIO.OUT)
GPIO.setup(butPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)



print("Program running... Press CTRL+C to exit")

try:
    while True:
        if GPIO.input(butPin):
            # button is released
            GPIO.output(ledPin, False)
        else:
            # button is pressed
            GPIO.output(ledPin, True)
        time.sleep(0.1)

except KeyboardInterrupt:
    print("Program terminated!")

finally:
    GPIO.cleanup()
```

src04.py

Working with Multiple LEDs and Push Button Switches

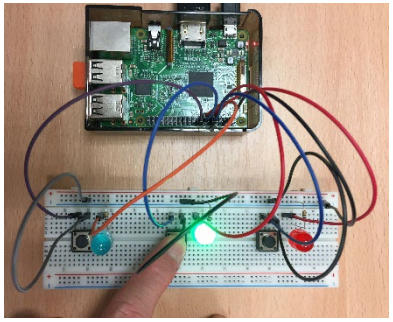


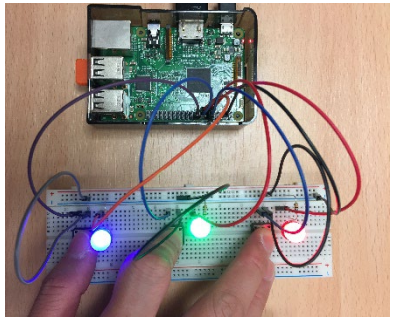
Working with Multiple LEDs and Push Button Switches (cont.)

- The circuit board is set up as follow:

	LED	Push Button Switch
Red	Pin 11	Pin 12
Green	Pin 13	Pin 16
Blue	Pin 15	Pin 18

- All the LEDs and buttons are grounded on Pin 6.
- Pressing individual button lights up the respective LED.
- Pressing all three buttons simultaneously lights up all three LEDs.





Working with Multiple LEDs and Push Button Switches (cont.)

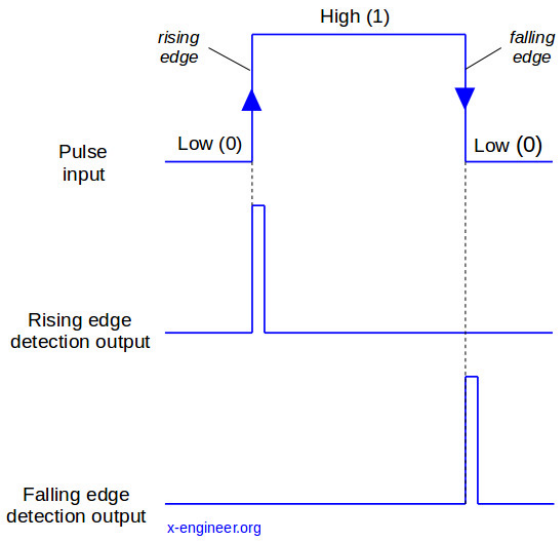
```python
# import the GPIO and time package
import RPi.GPIO as GPIO
import time



# Pin  Definitions
ledRedPin = 11
ledGreenPin = 13
ledBluePin = 15
butRedPin = 12
butGreenPin = 16
butBluePin = 18



# Pin Setup
GPIO.setmode(GPIO.BOARD)
GPIO.setup(ledRedPin, GPIO.OUT)
GPIO.setup(ledGreenPin, GPIO.OUT)
GPIO.setup(ledBluePin, GPIO.OUT)
GPIO.setup(butRedPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(butGreenPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(butBluePin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

```python
print("Program running... Press CTRL+C to exit")

try:
    while True:
        if GPIO.input(butRedPin):
            GPIO.output(ledRedPin, False)
        else:
            GPIO.output(ledRedPin, True)

        if GPIO.input(butGreenPin):
            GPIO.output(ledGreenPin, False)
        else:
            GPIO.output(ledGreenPin, True)

        if GPIO.input(butBluePin):
            GPIO.output(ledBluePin, False)
        else:
            GPIO.output(ledBluePin, True)

        time.sleep(0.1)

except KeyboardInterrupt:
    print("Program terminated!")

finally:
    GPIO.cleanup()
```

src05.py

Signal Edge

- In electronics, a signal edge is a transition of a digital signal from low to high (0 to 1) or from high to low (1 to 0):
  - A rising edge (or positive edge) is the low-to-high transition.
  - A falling edge (or negative edge) is the high-to-low transition.
  - Either edge – When the input signal is changing state, from high to low or from low to high.
- Signal edge detection can be used to program certain sensors that involve analogue value in a digital manner.



Example of a Heart Rate Sensor

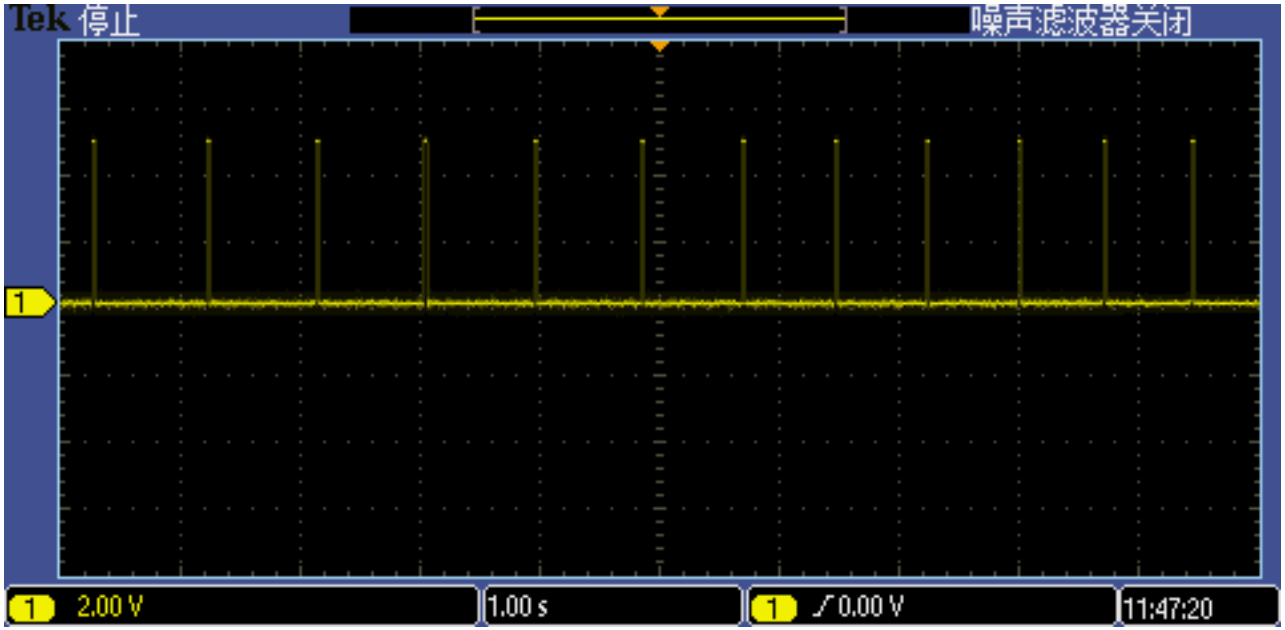
- Heart rate monitors typically use one of two different methods to record heart signals:
  - Either electrical or optical.
  - Both types of signals can provide the same basic heart rate data.
- ECG (Electrocardiography) sensors:
  - Measure the bio-potential generated by electrical signals that control the expansion and contraction of heart chambers.
  - Typically implemented in medical devices.
- PPG (Photoplethysmography) sensors:
  - Use a light-based technology to measure the blood volume controlled by the heart's pumping action.

Grove Ear-clip Heart Rate Sensor

- The Grove ear-clip heart rate module is a PPG sensor:
  - Can be used to monitor heart rate of patient and athlete.
  - The result can be displayed on a screen via a serial port and can be saved for analysis.
  - The module is characterized by high sensitivity, low power consumption and high portability.

Grove Ear-clip Heart Rate Sensor (cont.)

- Heart rate data is essentially an analogue value, but the Grove module handles it digitally:
  - Uses a rising edge event to transmit a single heartbeat.
  - The digital signal when measuring the heart rate is shown below:



Grove Ear-clip Heart Rate Sensor (cont.)

Sensor power to Pin 4 (5V)

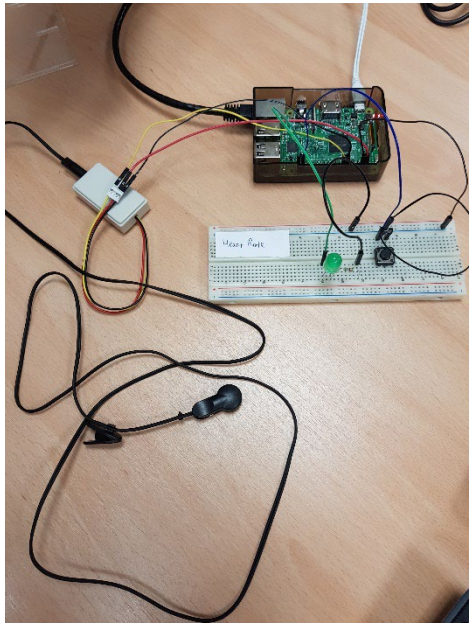
Sensor GND to Pin 14

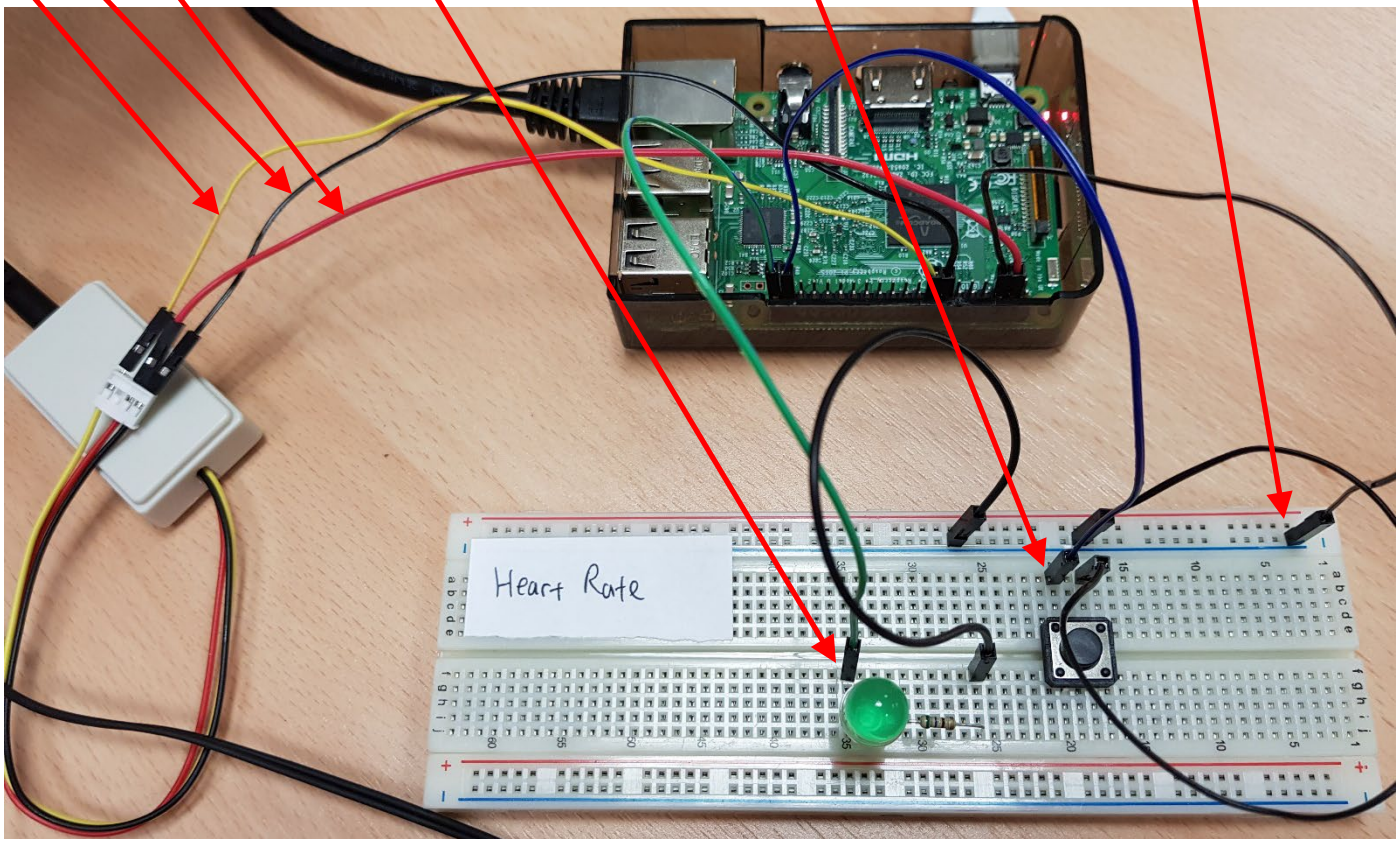
Sensor data line to Pin 16

Green LED to Pin 40

Push button to Pin 38

Breadboard GND line to Pin 6 (Ground)





Grove Ear-clip Heart Rate Sensor (cont.)

- Sample source code `src06.py`:
  - Uses the Rpi.GPIO library `add_event_detect()` to attach a Python function as the event handler when a rising edge event is detected on Pin 16.
  - 1 rising edge corresponds to 1 heartbeat.
  - Wait for 10 heart beats, i.e., a "good" continuous rising edge detection, before starting actual measurement.
  - Actual measurement takes place within a 15 second timeframe and then extrapolate to number of heart beats in one minute.

```
Program running... Press Ctrl+C to exit
Measuring heart rate...
Heart rate is 92
Heart rate is 88
Heart rate is 88
Heart rate is 88
Heart rate is 88
Stopped...
^CProgram terminated!
```

GPIO Programming Using Analogue Signal

Analogue Inputs

- The Raspberry Pi's GPIO pins are digital pins and thus you can only:
  - Set outputs to high or low.
  - Read inputs as high or low.
- Need to use an ADC to read the value of analogue input devices such as potentiometers:
  - Recall that the micro:bit has an onboard ADC.
  - Raspberry Pi does **NOT** have an onboard ADC.
  - Need to place a MCP3008 ADC on the breadboard and connect it to the Raspberry Pi.
  - Alternatively, can use the Analog Zero board, which provides the MCP3008 chip on a handy add-on board

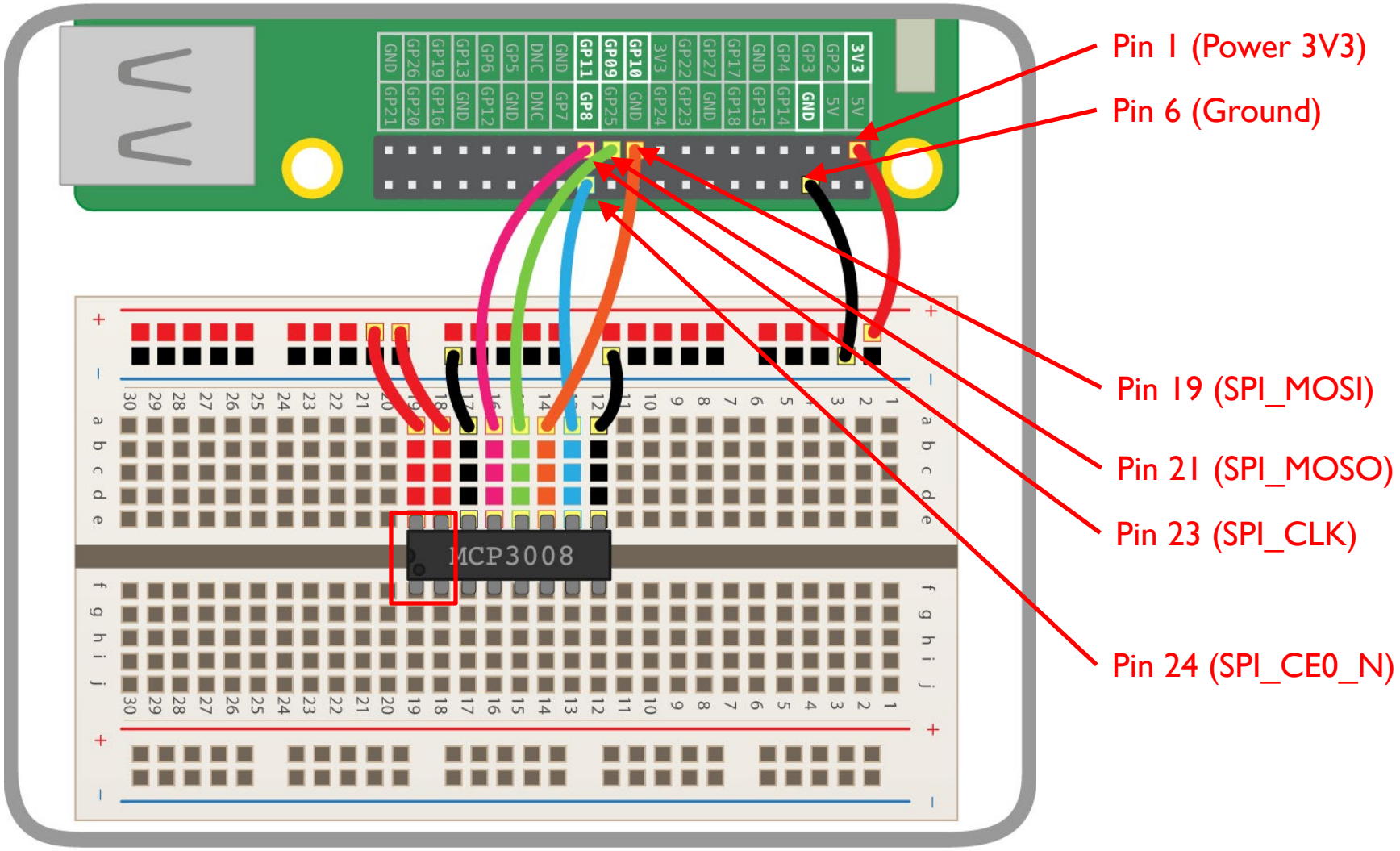
Analogue Inputs (cont.)

- The analogue values are communicated to the Pi using the SPI protocol:
  - While this will work in the GPIO Zero library out of the box, better results can be obtained if full SPI support is enabled.
  - In the Raspberry Pi Configuration dialogue from the main menu, enable SPI in the Interfaces tab:
    - Need to reboot the Pi after enabling the SPI interface.

Analogue Inputs (cont.)

- The MCP3008 is an ADC providing eight input channels:
  - The eight connectors on one side are connected to the Pi's GPIO pins.
  - The other eight are available to connect analogue input devices to read their values.
- Place the MCP3008 chip on the breadboard and carefully connect it up as shown in the diagram on the next slide:
  - Observe a small notch, or dot, in one end of the chip.
  - In the diagram, this end of the chip is aligned with column 19 on the breadboard.

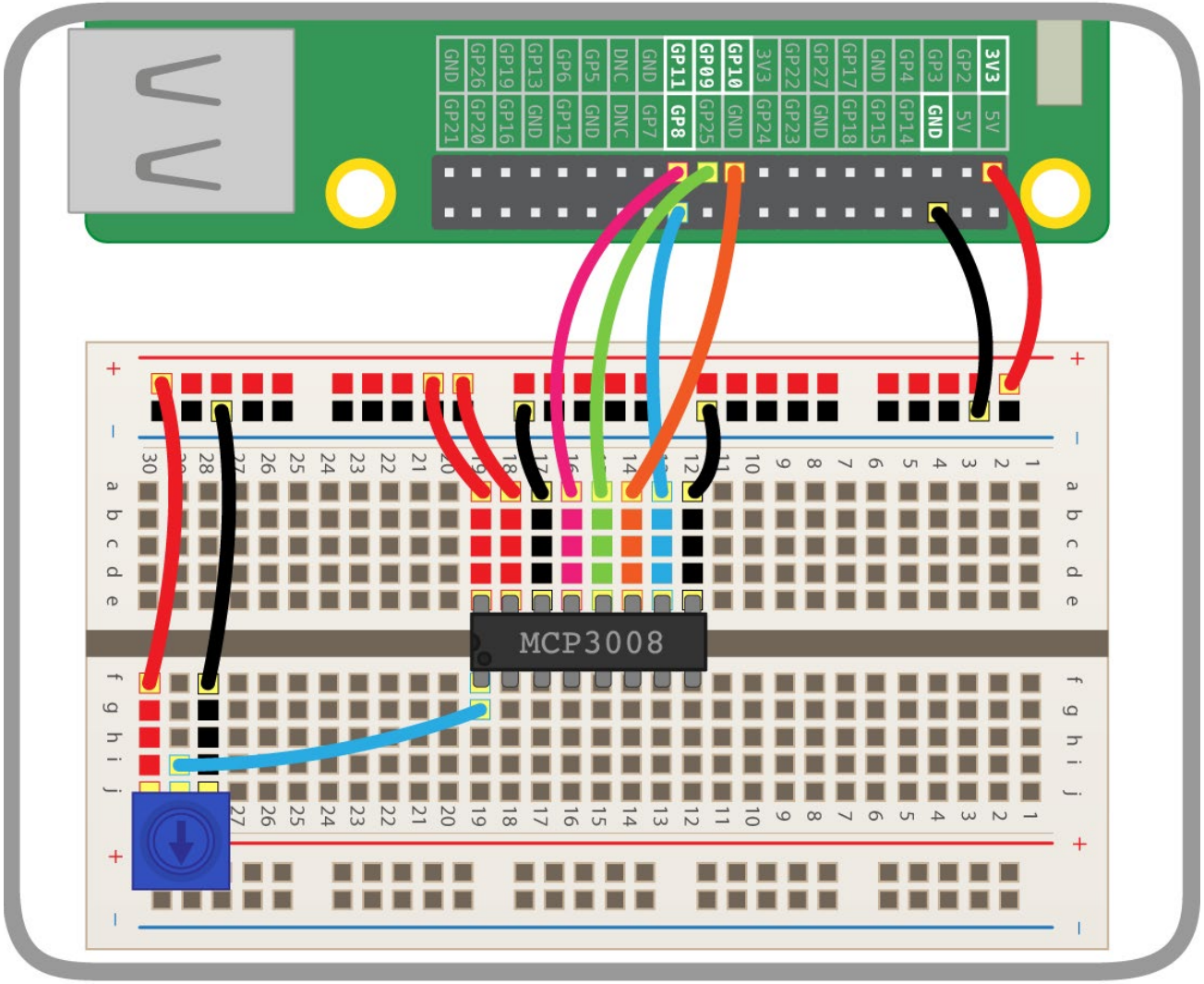
Analogue Inputs (cont.)



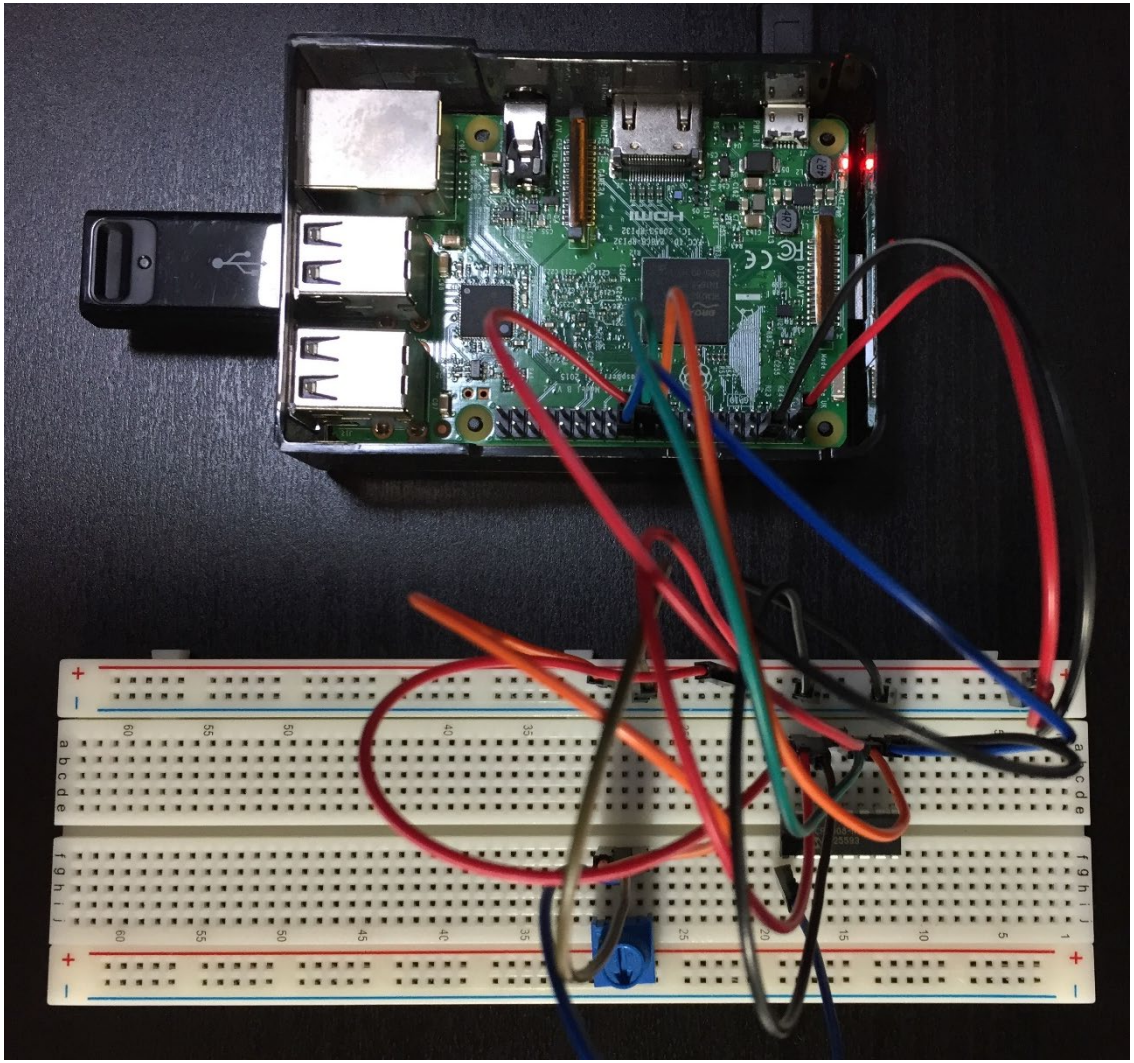
Analogue Inputs (cont.)

- Adding a potentiometer:
  - With the ADC connected to the Pi, you can connect devices up to the input channels.
  - A potentiometer is a good example of an analogue input device:
    - It is a variable resistor, and the Pi reads the voltage (from 0V to 3.3V).
    - A potentiometer's pins are ground, data and 3V3.
    - Thus, it is connected to ground and 3V3 supply with the actual voltage being read from the middle pin.
  - Place a potentiometer on the breadboard and wire:
    - One side to the ground rail.
    - The other side to the 3V3 rail.
    - The middle pin to the first input channel.

Analogue Inputs (cont.)



Analogue Inputs (cont.)



Analogue Inputs (cont.)

- We need to import the MCP3008 class from the GPIO Zero library in order to program the Pi to read the analogue input:
  - 0 represents the ADC's channel 0.
  - There are 8 channels (0 to 7), and we are using the first one.

```
1  from gpiozero import MCP3008
2
3  pot = MCP3008(0)
4
5  print("Program running... Press CTRL+C to exit")
6
7  try:
8
9      while True:
10         print(pot.value)
11
12 except KeyboardInterrupt:
13     print("Program terminated!")
```

src07.py

Analogue Inputs (cont.)

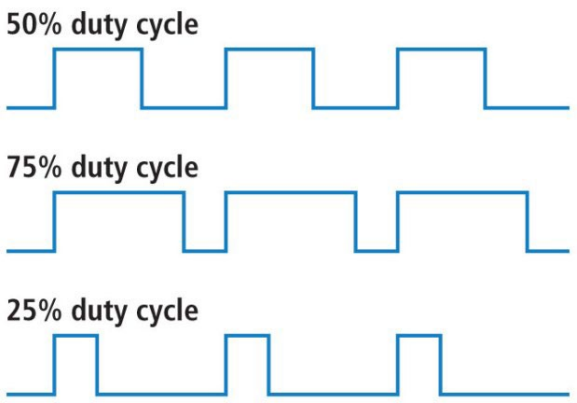
- Notice that when the potentiometer dial is in the halfway mark, the program will print out a value that is close to 0.50:
  - As the dial is turned anticlockwise, the value will decrease to 0.0.
  - And vice-versa, turning the dial clockwise, will increase the value to 1.0.

PWM (Pulse-Width Modulation) LED

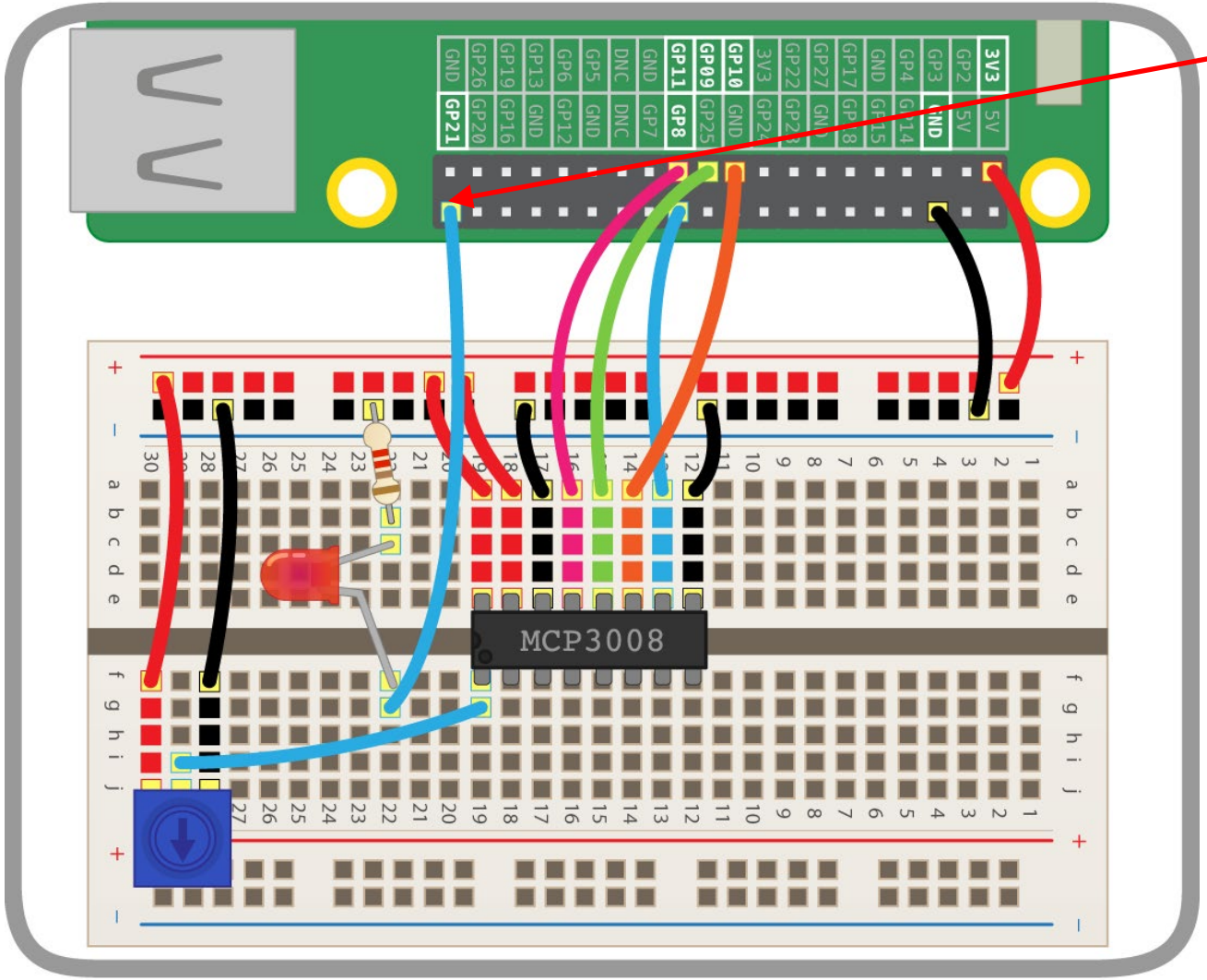
- Now able to read values from the potentiometer.
- Can connect the potentiometer to another GPIO device such as a LED:
  - This allows us to control the brightness of the LED using PWM or pulse-width modulation.
  - Add an LED to the breadboard and connect it to the Pi on Pin 40 (GPIO Pin 21).
  - The circuit is shown in the diagram on the next slide.
  - Remember to connect a resistor to the LED.
- What is PWM?
  - PWM is a fancy term for describing a type of digital signal.

PWM (Pulse-Width Modulation) LED (cont.)

- Allows us to vary how much time (i.e., the width) the digital signal is high in an analog fashion.
- E.g., while the signal can only be high (usually 3.3V) or low (ground) at any time, we can change the proportion of time the signal is high compared to when it is low over a consistent time interval:
  - 100% duty cycle means setting the voltage to 3.3V or high.
  - 0% duty cycle means setting the voltage to 0V or grounding it.
  - This allows us to control the brightness of LED by adjusting the duty cycle.



PWM (Pulse-Width Modulation) LED (cont.)



PWM (Pulse-Width Modulation) LED (cont.)

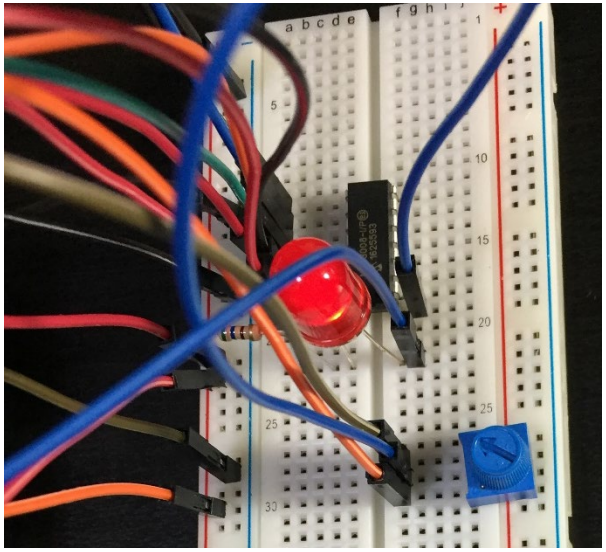
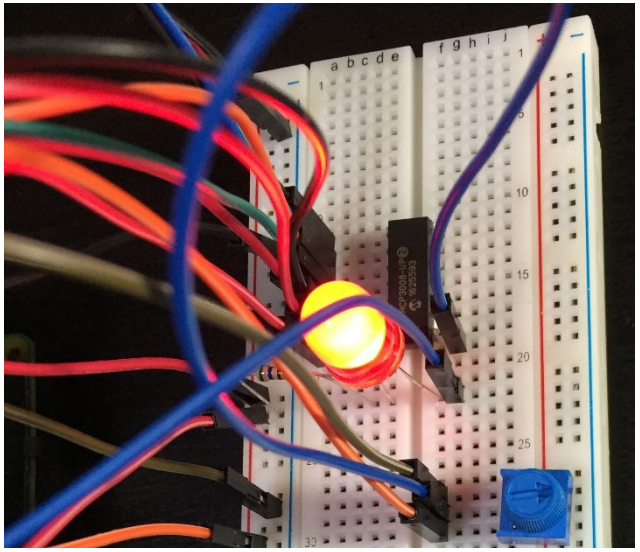
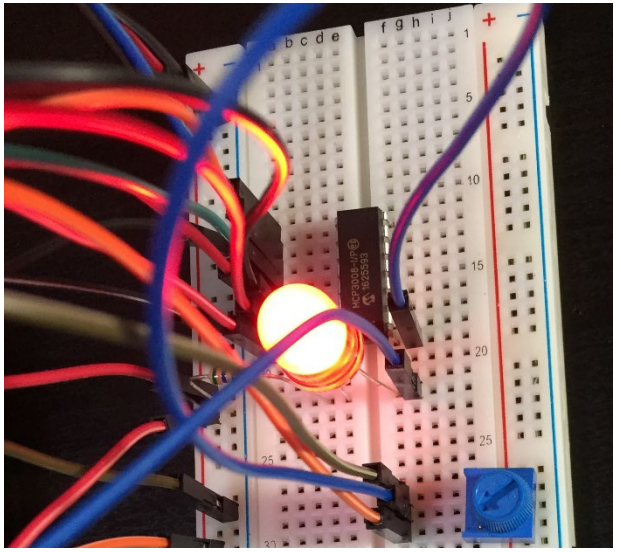
- GPIO Zero has a powerful feature of source and values:
  - Every device has a `value` property (the current value) and a `values` property (a stream of the device's values at all times).
  - Every output device has a source property which can be used to set what the device's value should be.
- In our current setup:
  - `pot.value` gives the potentiometer's current value (it is read only, as it's an input device)
  - `led.value` is the LED's current value (it is read/write – see the value and change the value)
  - `pot.values` is a generator constantly yielding the potentiometer's current value

PWM (Pulse-Width Modulation) LED (cont.)

- `led.source` is a way of setting where the LED gets its values from
- Rather than continuously setting the value of the LED to the value of the potentiometer in a loop, we can just pair the two devices with the line `led.source = pot.values`.
  - The alternative loop will look like:

```
while True:
    led.value = pot.value
```

PWM (Pulse-Width Modulation) LED (cont.)

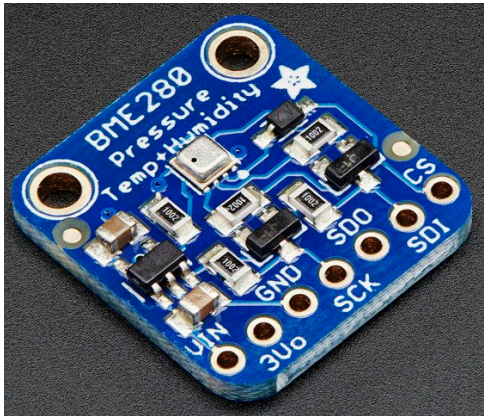
The brightness of the red LED gradually increases as we turn the potentiometer dial in clockwise direction.

```
1  import time
2  from gpiozero import MCP3008
3  from gpiozero import PWMLED
4
5  pot = MCP3008(0)
6  led = PWMLED(21)
7  led.on() # test - the led should be lit
8  led.off() # test - the led should go off
9  led.value = 0.5 # test - the led should be lit at half brightness
10 led.source = pot.values # actual connecting of the LED to the potentiometer
11
12 print("Program running... Press CTRL+C to exit")
13
14 try:
15
16     while True:
17         time.sleep(0.1)
18
19 except KeyboardInterrupt:
20     print("Program terminated!")
```

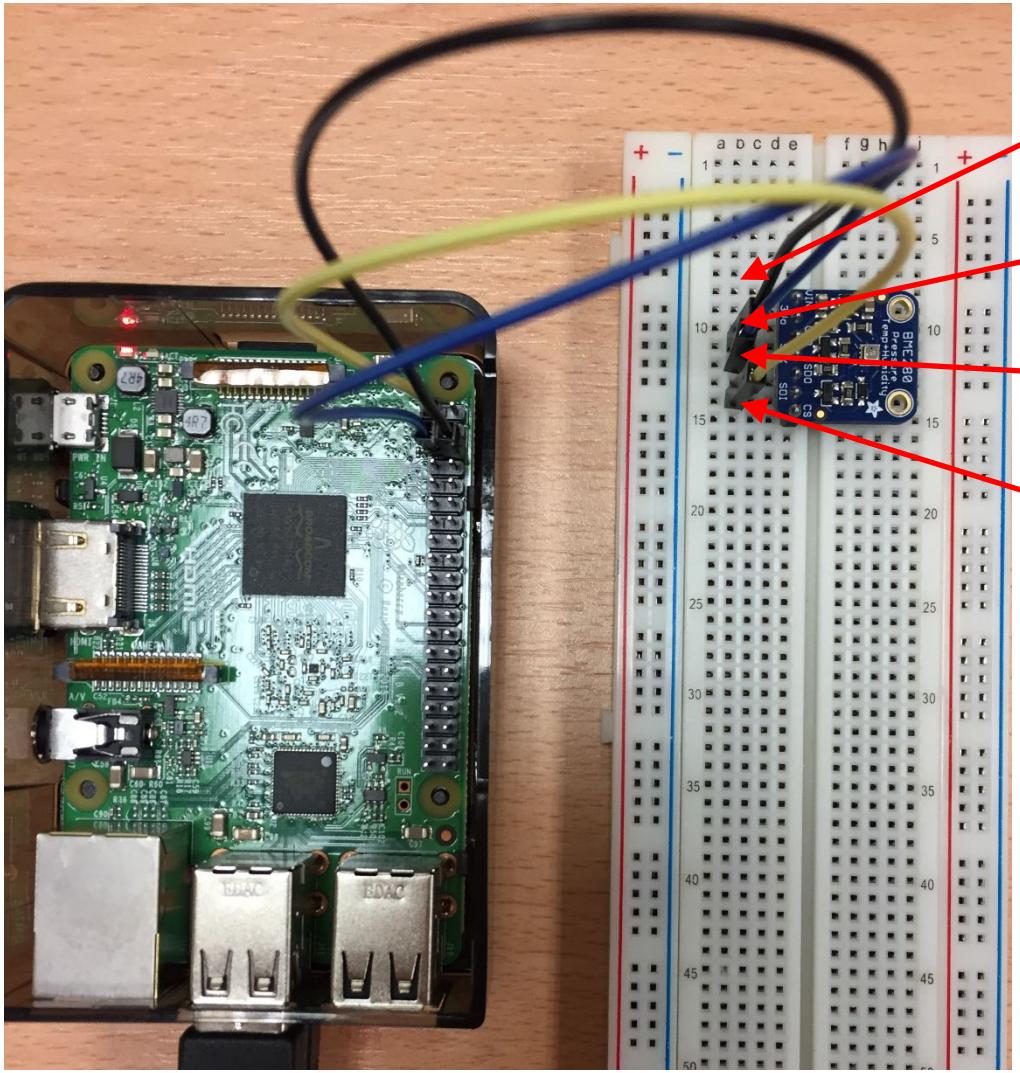
src08.py

Bosch BME280 Environmental Sensor

- **Bosch BME280 sensor**:
  - Three-in-one environmental sensor with temperature, barometric pressure and humidity:
    - Temperature with ±1.0°C accuracy.
    - Barometric pressure with ±1 hPa absolute accuracy.
    - Humidity with ±3% accuracy.
  - Good for all sorts of weather/environmental sensing.
  - Can be used in both I2C and SPI.
- Adafruit's version – The surface-mount sensor is soldered onto a PCB.
- Comes with a 3.3V regulator and level shifting so you can use it with a 3V or 5V logic microcontroller.



Bosch BME280 Environmental Sensor (cont.)



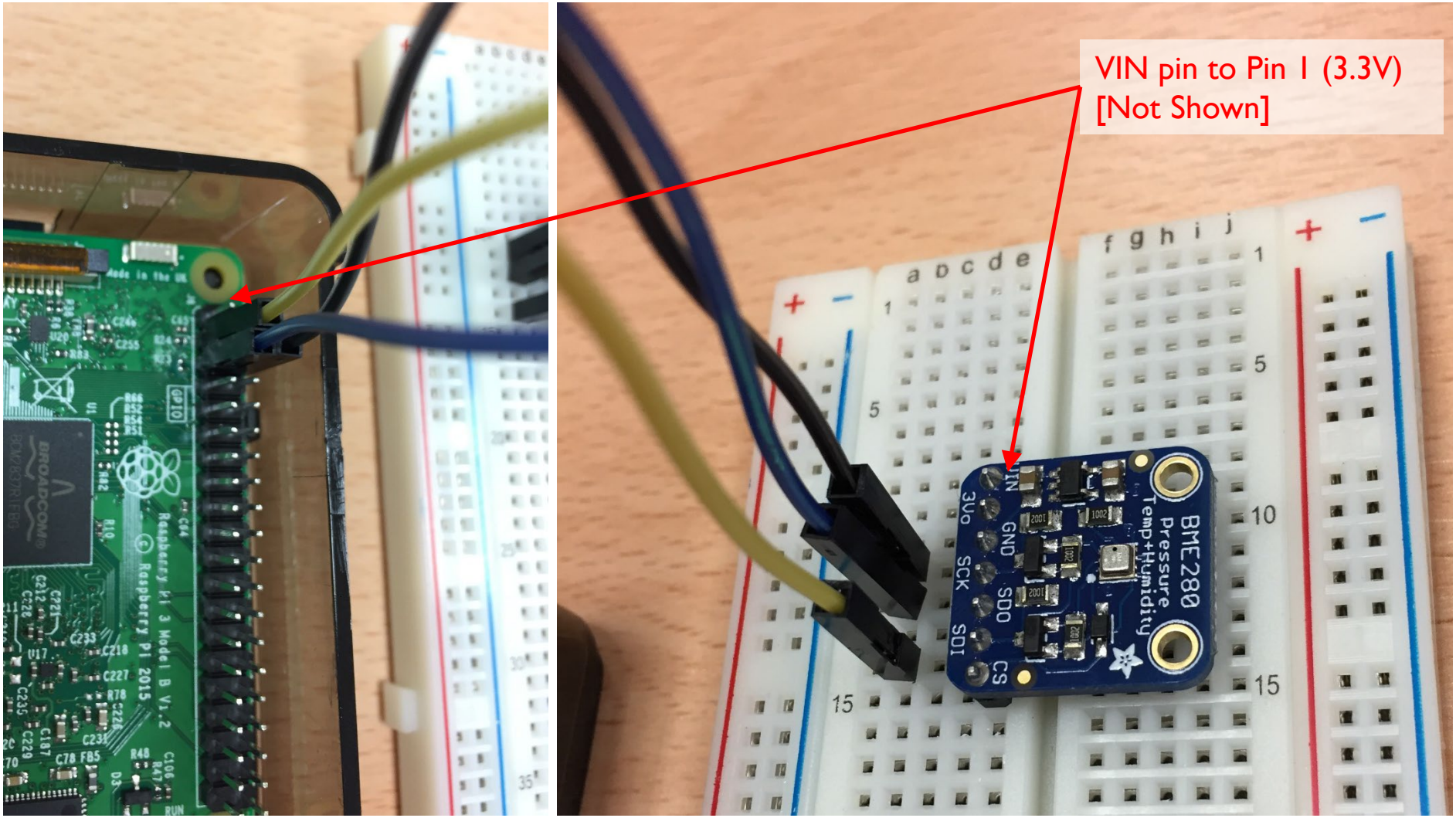
VIN pin to Pin 1 (3.3V) [Not Shown]

GND pin to Pin 6 (Ground)

SCK pin to Pin 5 (SCL1, I2C)

SDI pin to Pin 3 (SDA1, I2C)

Bosch BME280 Environmental Sensor (cont.)



Adafruit CircuitPython for BME280

- Allows you to easily read in a single set of data points from the BME280.
- Refer to this web page for more information.
- Enable I2C interface in Raspberry Pi Configuration and restart your Raspberry Pi
- Run the following commands to install the library and get the data:

```
sudo python3 -m pip install adafruit-
   circuitpython-bme280
python3 src09.py
```

Adafruit CircuitPython for BME280 (cont.)

src09.py 2

src09.py > ...

```python
import time

import board
from adafruit_bme280 import basic as adafruit_bme280

i2c = board.I2C()  # uses board.SCL and board.SDA
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)

try:

    while True:

        print('Temperature = {0:0.3f} deg C'.format(bme280.temperature))
        print('Pressure    = {0:0.2f} hpa'.format(bme280.pressure))
        print('Humidity    = {0:0.2f} %'.format(bme280.humidity))

        time.sleep(1)

except KeyboardInterrupt:
    print("Program terminated!")

finally:
    pass
```

src09.py

```
File  Edit  Tabs  Help
pi@raspberrypi:~/Documents/is4
Temperature = 24.998 deg C
Pressure    = 1010.06 hpa
Humidity    = 54.31 %
Temperature = 24.993 deg C
Pressure    = 1010.05 hpa
Humidity    = 54.06 %
Temperature = 25.003 deg C
Pressure    = 1009.99 hpa
Humidity    = 53.92 %
Temperature = 25.003 deg C
Pressure    = 1010.07 hpa
Humidity    = 54.45 %
Temperature = 25.013 deg C
Pressure    = 1010.07 hpa
Humidity    = 54.11 %
Temperature = 25.013 deg C
Pressure    = 1010.06 hpa
Humidity    = 54.28 %
Temperature = 25.013 deg C
Pressure    = 1010.12 hpa
Humidity    = 54.49 %
Temperature = 25.013 deg C
Pressure    = 1010.08 hpa
Humidity    = 54.80 %
Temperature = 25.023 deg C
Pressure    = 1010.07 hpa
Humidity    = 54.95 %
```

Summary

- Raspberry Pi is a single-board computer that enables hardware prototype via its general-purpose input/output (GPIO) connector.
- Using the RPi.GPIO library, we can easily work with external sensors and devices using digital signal.
- Signal edge detection can be used to program certain analogue device digitally.
- By adding an ADC to the Raspberry Pi, we can handle analogue signal too.

Q&A

Next Lecture…

- Learn about:
  - Working with Raspberry Pi's interfaces.
  - BLE communication with Raspberry Pi.
  - Using Raspberry Pi to control micro:bit devices with BLE and radio.