

## Internet of Things (IoT)

### Topic 3: GPIO Programming

#### Equipment List

Before you start, make sure that you have all the following equipment. Please keep them carefully throughout the workshop, and you are required to return them at the end of the IoT training module.

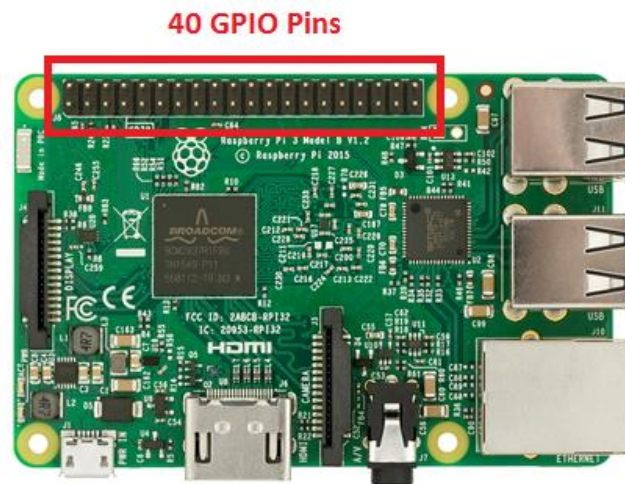
- 1 Breadboard
- 1 SG-90 Micro Servo
- 1 Green LED
- 2 Red LED
- 2 Push button
- 3 330-Ohm Resistor
- 4 Male-to-Male jumper wire
- 5 Male-to-Female jumper wire

#### Deliverables

You should complete all the tasks, then submit the following Python source codes to the course Moodle page.

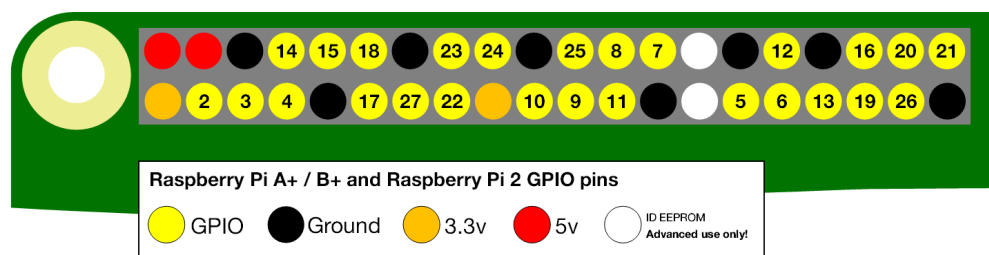
- Task 1 – ***led.py***
- Task 2 – ***reaction.py***
- Task 3 – Answer Moodle Questions
- Task 4 – ***traffic\_light.py***

## Introduction



The Raspberry Pi is more than just a credit card-sized computer! One powerful feature of the Raspberry Pi is the row of GPIO (General Purpose Input/Output) pins along the top edge of the board, making it a perfect candidate for hardware prototyping and IoT (Internet of Things) applications.

The GPIO pins are a physical interface between the Pi and the outside world. In other words, you can use a Pi to control and monitor the inputs and outputs of an electronic circuit. For example, these GPIO pins can be used to drive LEDs, spin motors (with limited voltage), or read button presses, etc.



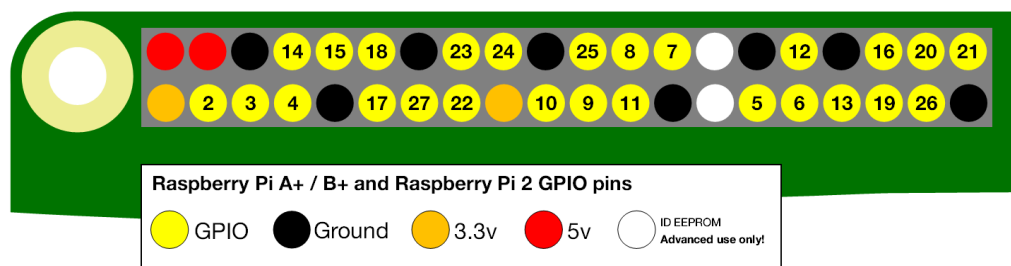
Of the 40 pins, 26 are GPIO pins and the others are power (3.3V or 5V), ground pins, and two ID EEPROM pins (which you should NOT use unless you know what you are doing!).

**WARNING:** Please follow the instructions carefully when you are playing around with the GPIO pins. Randomly plugging wires and power sources into your Pi may lead to short circuits, burning the GPIO pins and may even kill the whole Raspberry Pi board.

## Pin Numbering

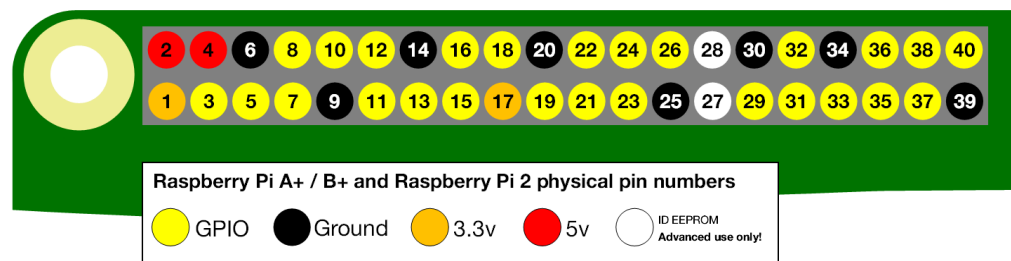
You may notice that there are numbers on the GPIO pins and the order is rather strange. In order to control the GPIO pins, it requires a bit of programming (we will be using Python!). These numbers are used in our programs to refer to the location of GPIO pins, and there are actually two different numbering ways: GPIO numbering and physical numbering.

### 1. GPIO Numbering




This is also called Broadcom SoC numbering. The numbers may change with different versions of Raspberry Pi. We will need to use **GPIO.setmode(GPIO.BCM)** in our programs in order to use this scheme.

### 2. Physical Numbering




Another way to refer to the pins is by simply counting across and down from pin 1 at the bottom of top left corner. This is also called board numbering. We will need to use **GPIO.setmode(GPIO.BOARD)** in our programs in order to use this scheme. **You may refer to this page for reference during the programming process.**






## Raspberry Pi 4 B J8 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1, I <sup>2</sup> C)		DC Power 5v	04
05	GPIO03 (SCL1, I <sup>2</sup> C)		Ground	06
07	GPIO04 (GPCLK0)		(TXD0, UART) GPIO14	08
09	Ground		(RXD0, UART) GPIO15	10
11	GPIO17		(PWM0) GPIO18	12
13	GPIO27		Ground	14
15	GPIO22		GPIO23	16
17	3.3v DC Power		GPIO24	18
19	GPIO10 (SPI0_MOSI)		Ground	20
21	GPIO09 (SPI0_MISO)		GPIO25	22
23	GPIO11 (SPI0_CLK)		(SPI0_CE0_N) GPIO08	24
25	Ground		(SPI0_CE1_N) GPIO07	26
27	GPIO00 (SDA0, I <sup>2</sup> C)		(SCL0, I <sup>2</sup> C) GPIO01	28
29	GPIO05		Ground	30
31	GPIO06		(PWM0) GPIO12	32
33	GPIO13 (PWM1)		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

## Raspberry Pi 4 B J14 PoE Header

01	TR01		TR00	02
03	TR03		TR02	04

## Pinout Grouping Legend

Inter-Integrated Circuit Serial Bus			Serial Peripheral Interface Bus
Ungrouped/Un-Allocated GPIO			Universal Asynchronous Receiver-Transmitter
Reserved for EEPROM			

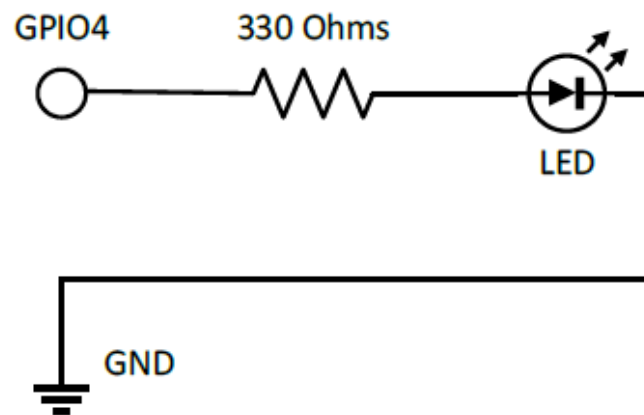
Rev. 2  
19/06/2019 CGS

[www.element14.com/RaspberryPi](http://www.element14.com/RaspberryPi)

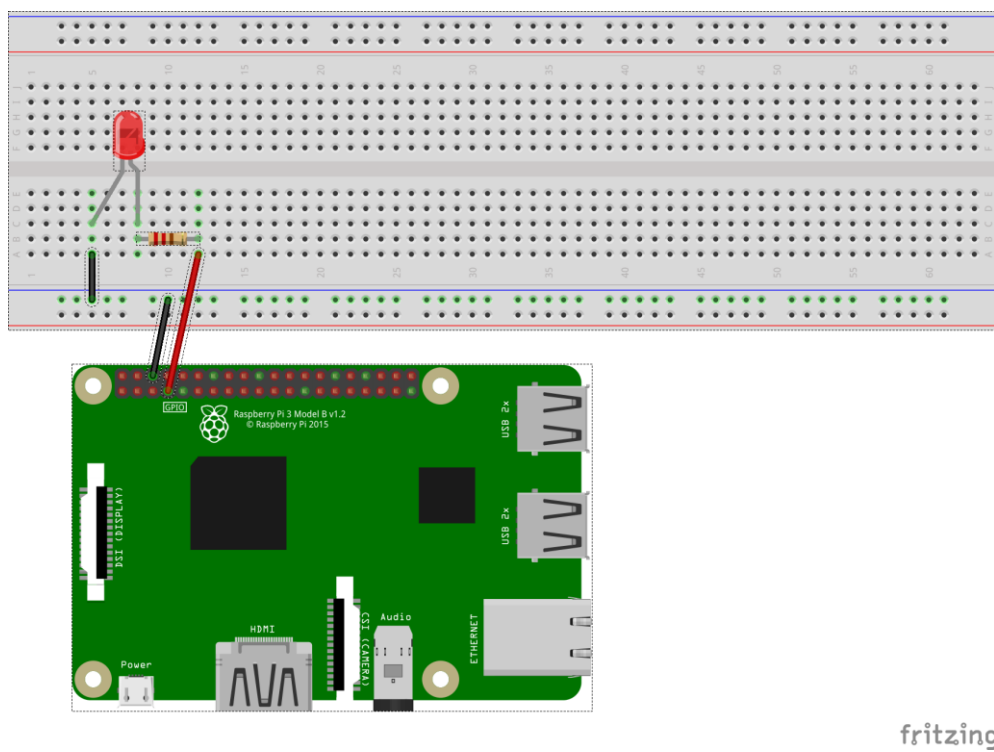
**Note:** As you can see from the reference image, the GPIO pins can also be used as Serial (UART), I2C, SPI, and even some PWM (“analog output”). For more details on the advanced capabilities of the GPIO pins, you may refer to <http://pinout.xyz/>

## GPIO Output

Let's try to set up the following circuit on your breadboard:



**\*You must ALWAYS use resistors to connect LEDs to the GPIO pins! (Why?)**



**\*The longer leg of the LED is the positive side. If the LED doesn't light up in the following exercises, try to invert the polarity of the LED.**

1. Open a Terminal, type  
***sudo idle &***

This will open the default Python 3 programming environment called IDLE as the super user, which allows us to change the GPIO pins on the Pi.

2. Create a new file and save it as ***led.py***
3. Import the modules and libraries needed to control the GPIO pins  
***import RPi.GPIO as GPIO***  
***import time***
4. Set the GPIO pin numbering mode to GPIO Numbering  
***GPIO.setmode(GPIO.BCM)***
5. To avoid annoying GPIO warning messages, type  
***GPIO.setwarnings(False)***
6. Create a variable for the LED and assign its pin number, then specify the LED pin as an output pin (***GPIO.OUT***)  
***led = 4***  
***GPIO.setup(led, GPIO.OUT)***
7. To turn the LED on, simply set the GPIO output pin to **1 (True / GPIO.HIGH)**, which will drive the pin to 3.3V  
***GPIO.output(led, 1)***
8. Add a line to let the program wait for 5 seconds  
***time.sleep(5)***
9. To turn the LED off, set the GPIO output pin to **0 (False / GPIO.LOW)**, which will drive the pin to 0V  
***GPIO.output(led, 0)***
10. At the end of the program, remember to always clean up the GPIO pins so that they will be ready to be used again next time. This will reset all the GPIO pins to input mode to protect against accidental connections to GND (low).  
***GPIO.cleanup()***
11. Now try to **Run** the program and see if the LED can light up for 5 seconds!

---

**Task 1:** *Change the program so that the LED can keep on blinking with a two-second interval, using Physical Numbering pin mode. i.e. Loop the following: On for 1 second, Off for 1 second.*

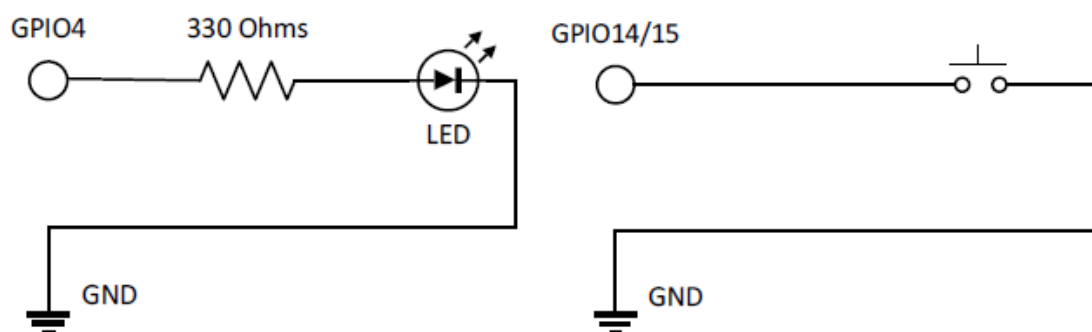
*Submit demonstration video and your completed **led.py** to Moodle*

---

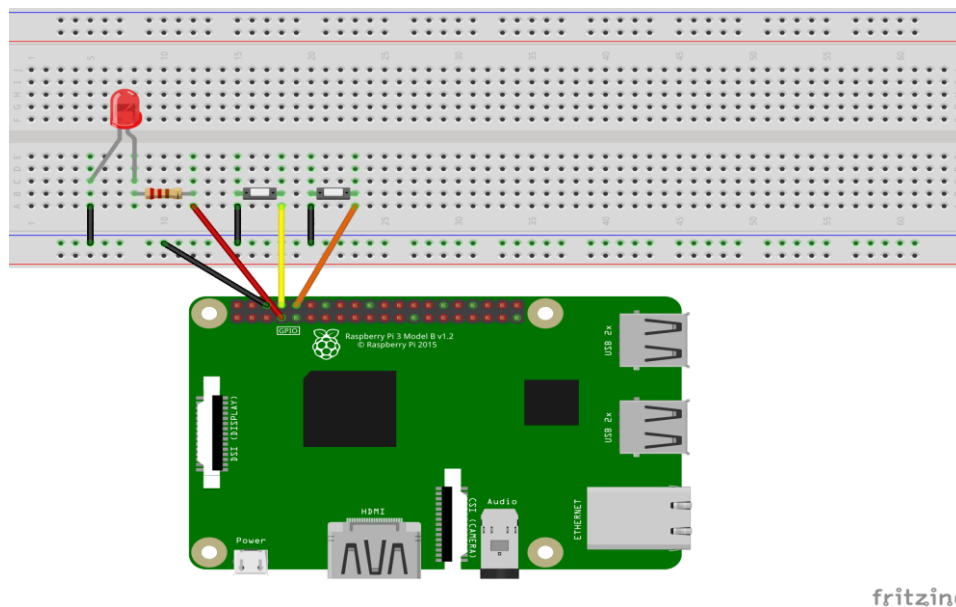
## GPIO Input

Similar to GPIO outputs, GPIO inputs are digital - meaning that it can only be 1 or 0 (on/off, True/False). By default the GPIO Input will float between 0 and 1 if it is not connected to a voltage. Therefore, it would be hard to tell if you were going up or down, or even what up or down meant without a reference! We will need to assign a default value for GPIO inputs: **PUD\_UP** (strong input force will pull the value down to 0) or **PUD\_DOWN** (strong input force will pull the value up to 1).

Let's try to make a simple 2-player reaction game with LEDs and push buttons. First construct the following 2 circuits on your breadboard using **GPIO4**, **GPIO14** and **GPIO15**.



*(Unlike LEDs, push buttons carry large-enough resistance so we do not need to add resistors for them)*



1. Create a new file and save it as ***reaction.py***
2. Import the modules and libraries needed, we will also use the “random” library for generating random numbers  
***import RPi.GPIO as GPIO***  
***import time***  
***import random***
3. Set the corresponding GPIO pin mode  
***GPIO.setmode(GPIO.BCM)***  
***GPIO.setwarnings(False)***
4. Create a variable for the LED and assign its pin number, then specify the LED pin as an output pin  
***led = 4***  
***GPIO.setup(led, GPIO.OUT)***
5. To turn the LED on after some random seconds, you may make use of the “***random.uniform(t1,t2)***” function, which will return a random number between ***t1*** and ***t2***  
***GPIO.output(led, 0)***  
***time.sleep( ??? )***  
***GPIO.output(led, 1)***
6. Create new variables for the left and right push buttons, then assign their corresponding pin numbers  
***left\_button = ???***  
***right\_button = ???***
7. Set the push buttons as input pins (***GPIO.IN***), and assign ***GPIO.PUD\_UP*** as their default value  
***GPIO.setup(left\_button, GPIO.IN, GPIO.PUD\_UP)***  
***GPIO.setup(right\_button, GPIO.IN, GPIO.PUD\_UP)***
8. Since we are using ***PUD\_UP*** as the default value, when the push button is being pressed, the GPIO input value detected will be ***0 (False)***. Set up a loop so that the program will keep waiting until a button has been pressed. Use ***GPIO.input( pin\_number )*** to get the input value of a specific pin.  
***while True:***  
     ***if ( ??? ) == False:***  
         ***print('Left button pressed.')***  
         ***break***  
     ***if ( ??? ) == False:***  
         ***print('Right button pressed.')***  
         ***break***



9. At the end of the program, remember to clean up the GPIO pins

```
try:
    while True: ##this is the while loop of the previous step
        ???
except KeyboardInterrupt:
    print("CTRL-C: Terminating program.")
finally:
    print("Cleaning up GPIO...")
    GPIO.cleanup()
```

10. Now try to **Run** the program and play the game!

---

**Task 2:** Complete the above program by filling in the “???” parts so that the LED will sleep for random seconds between 1 and 5.

Also, instead of outputting ‘**Left/Right button pressed**’, the program should ask for players’ name before the game starts, and display the winner’s name at the end (e.g. ‘**Player\_Name wins!**’)

[Submit your demonstration video and completed \*\*reaction.py\*\* to Moodle](#)

---

Hints: To get input from the keyboard and assign it to a variable, use

```
left_player = input('Please enter the left player name : ')
```

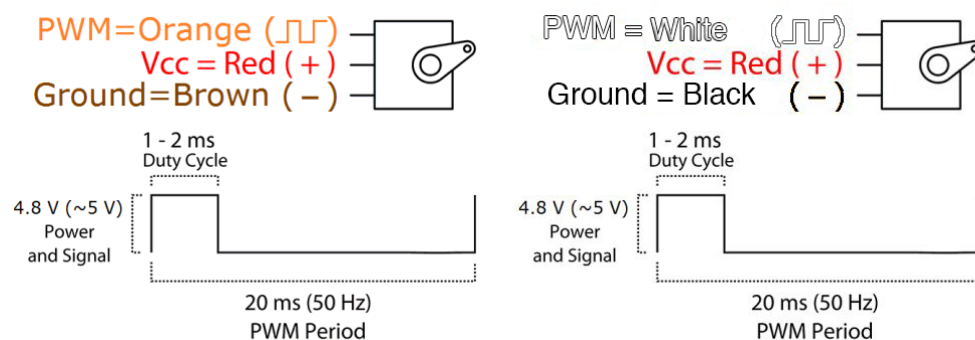
## Analog Output

For digital signals, they can only be high (usually 3.3V or 5V) or low (ground) at any time. But what if we want to produce a variable voltage level?

**Pulse Width Modulation (PWM)** is a method for generating an analog signal using a digital source. It can change the proportion of time the signal is high compared to when it is low over a consistent time interval. PWM signal consists of two main components that define its behavior: a Duty Cycle and a Frequency.

By Pulse Width Modulation, we can get a variable voltage digitally, that is we get voltages between 0 and VCC by switching the VCC on and off periodically. PWM signals are required for a wide variety of control applications! For example, to control the speed of DC motors, the dimming of LEDs, or the direction of a servo motor, etc.

Let's try to control a **SG90 Micro Servo** motor with the Raspberry Pi.



The SG90 expects a frequency of 50 Hz on the control line, and the position it moves depends on the pulse width of the signal. For the Raspberry Pi, we do not have a change pulse width method for PWM, but we can change the Duty Cycle. Note that:

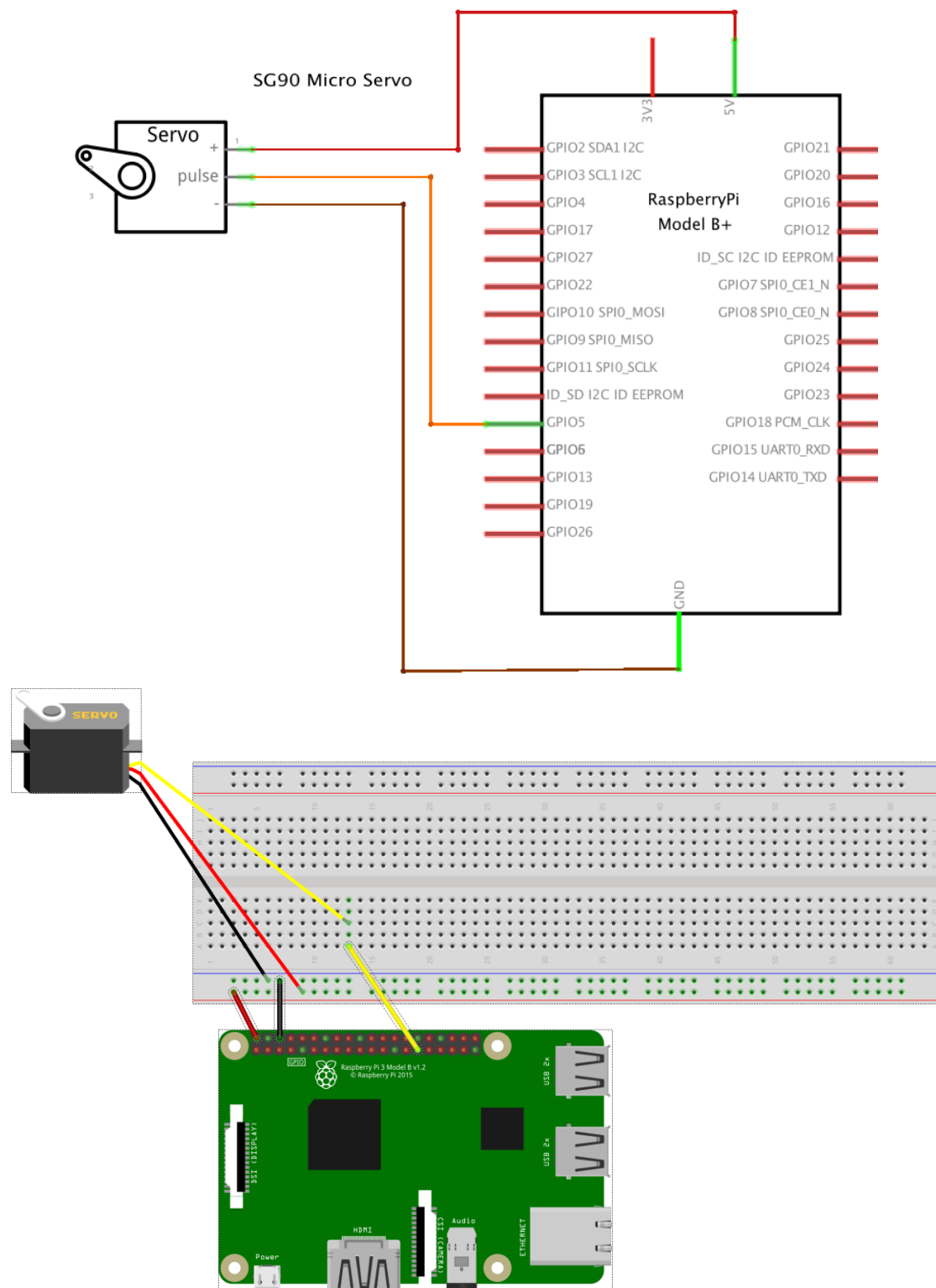
$$\text{Duty Cycle} = \text{Pulse Width} * \text{Frequency}$$

Given a 50 Hz frequency, we can calculate the required duty cycle for any pulse width. For example:

- We need a 1.5ms pulse to center the servo,  
**Duty Cycle =  $0.0015 * 50 = 0.075$  (7.5%)**
- Similarly, a 1ms pulse to move the servo to -90 degrees,  
**Duty Cycle =  $0.001 * 50 = 0.05$  (5%)**
- Similarly, a 2ms pulse to move the servo to +90 degrees,  
**Duty Cycle =  $0.002 * 50 = 0.1$  (10%)**

Thus the duty cycle range should be from 5 - 10% with the center at 7.5%. Of course, these are just the theoretical values calculated based on the data sheet of SG90. We may need to calibrate the exact numbers for the servo motor later.

Now try to construct the following circuit **(please check the wire colors carefully!)**:



fritzing

1. Create a new file and save it as ***servo.py***
2. Import the modules and libraries needed  
***import RPi.GPIO as GPIO***  
***import time***
3. Set the corresponding GPIO pin mode  
***GPIO.setmode(GPIO.BCM)***  
***GPIO.setwarnings(False)***
4. Create a variable for the servo and assign its pin number, then specify the pin as an output pin  
***servo\_pin = 5***  
***GPIO.setup(servo\_pin, GPIO.OUT)***
5. Create a variable called *duty\_cycle*, and assign a default value of 7.5 to it (this should be the theoretical center position of the servo!)  
***duty\_cycle = 7.5***
6. Create a PWM channel on the servo pin with a frequency of 50Hz  
***pwm\_servo = GPIO.PWM(servo\_pin, 50)***
7. Set the initial value for PWM  
***pwm\_servo.start(duty\_cycle)***
8. Copy and paste the following codes  
***try:***  
    ***while True:***  
        ***duty\_cycle = float(input("Enter a Duty Cycle value:"))***  
        ***pwm\_servo.ChangeDutyCycle(duty\_cycle)***  
***except KeyboardInterrupt:***  
    ***print("CTRL-C: Terminating program.")***  
***finally:***  
    ***print("Cleaning up GPIO...")***  
    ***GPIO.cleanup()***

---

*Task 3: Test the servo motor with different duty cycle values*

*Answer Q1 in Moodle: what are the minimum and maximum values effective duty cycle values of your motor?*

*Answer Q2 in Moodle: what is the use of the “try...except...finally” statements in the program?*

---

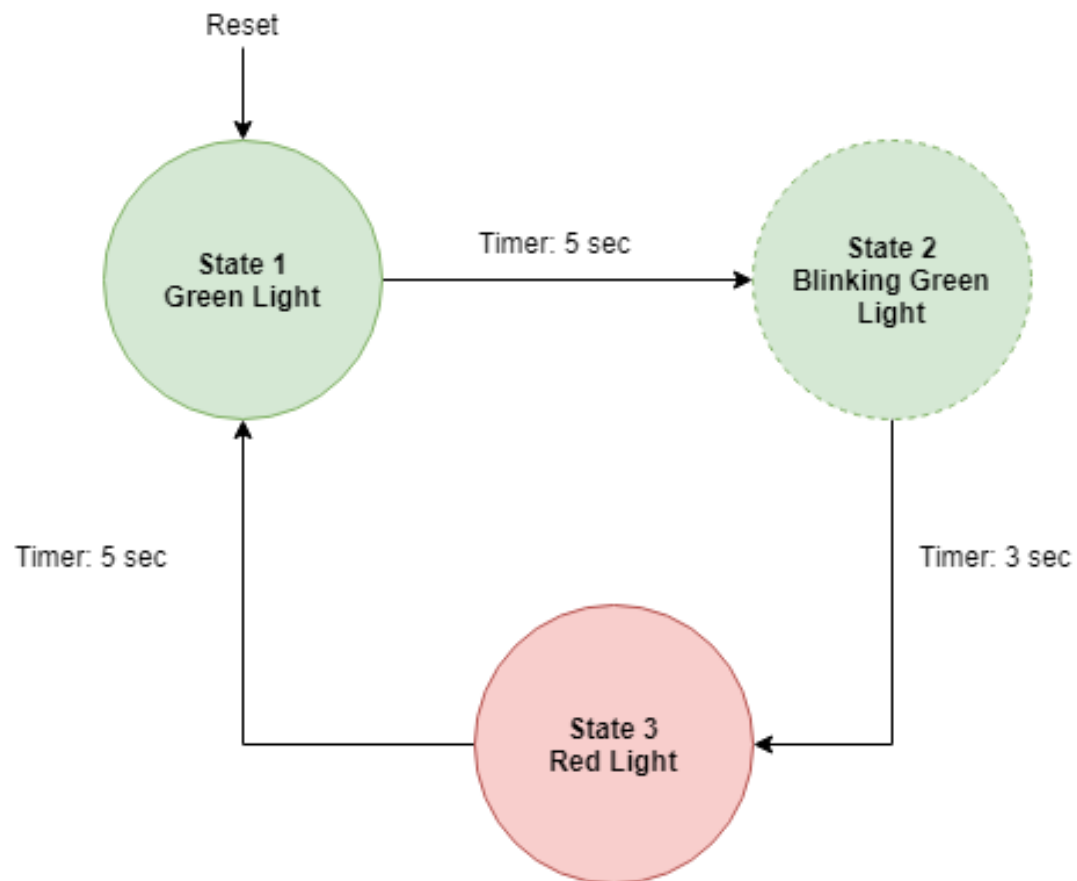
*Hints: Start with the theoretical duty cycle values, and then gradually move them up and down. Note the values just before the limits, and you might have to use 0.5% increments towards the end.*

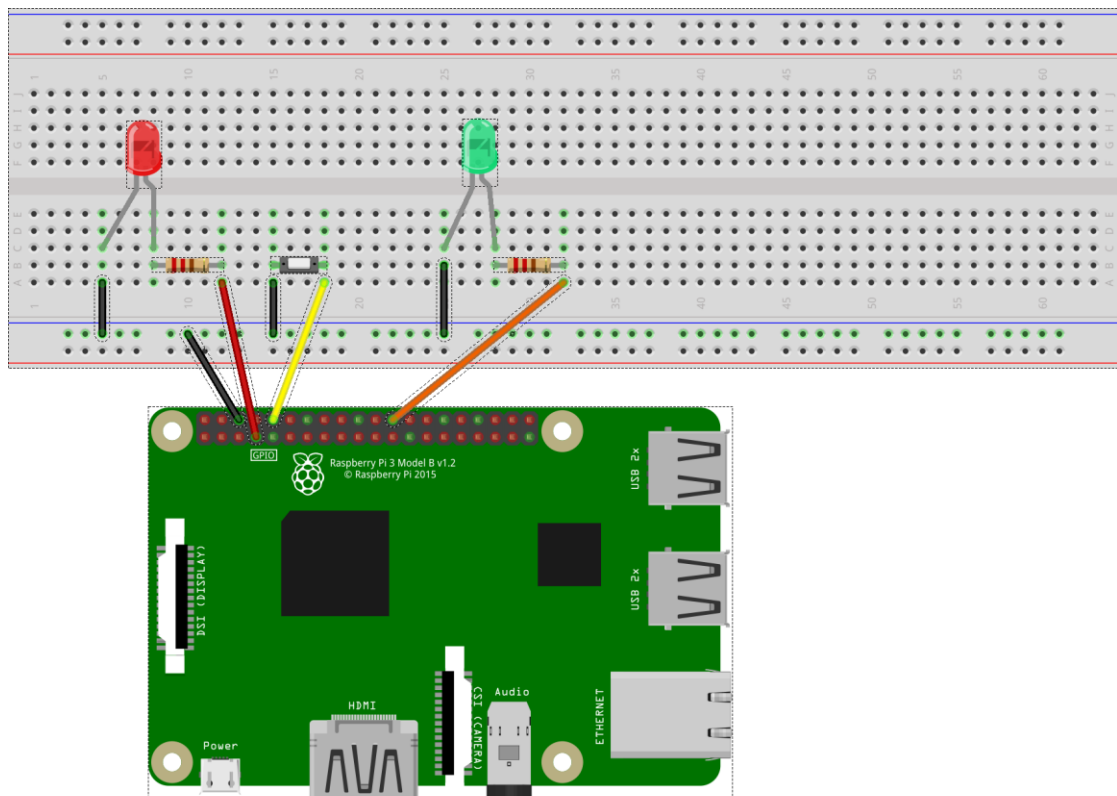
*Note: You may notice that the servo shakes a little bit. It is because the RPi.GPIO library implements a software timed PWM, therefore it suffers from scheduling jitter and makes it unstable! You would need to add an extra hardware, or simply use another Python library module that uses hardware timing, in order to solve the problem.*

### Integrated Exercise - Traffic Light

Let's try to integrate all the things you have just learnt and build your own traffic light using LEDs and a button!

Please refer to the video in Moodle on the application you are going to build. This is the FSM for the traffic light:





fritzing

Create a new file and save it as ***traffic\_light.py***. Construct a circuit with a red LED and a green LED (you should know how to do it correctly by now!), and complete the program according to the above FSM specification. Also, you are required to use the “***try...except...finally***” statements for exception handling (e.g. Keyboard Interrupt).

*Hints: Try to use functions for a better programming design, so that your code would be more readable and reusable!*

For example:

```
# Define a function to control both the Red and Green LEDs
def light_on(red_on, green_on):
    ...
    ...
```

**(Make sure you know what you are doing before moving to the next part! Feel free to ask the TA for any help if you are not sure!)**

Next, we would like to **add a push button (Wait button) and a red LED (signal LED)** for the pedestrians. The Wait button should only be enabled during State 3 (i.e. Red traffic light), and the signal LED should be lit up to indicate that the Wait button has been pressed for **at least** 0.5 seconds or a longer time.

The FSM behavior of State 3 is also changed as well: If the Wait button is not pressed, the traffic light will now stay in State 3 forever.

If the Wait button is pressed for at least 0.5 seconds, and then released, and by the time the button is released, the system has been in State 3 for **less than** 5 seconds, then a signal LED should be turned on **immediately**, and a 3-second countdown should start **after** the system has been in State 3 for 5 seconds.

If the wait button is pressed for at least 0.5 seconds, and then released, and by the time the button is released, the system has been in State 3 for **more than or equal to** 5 seconds, then a signal LED should be turned on **immediately**, and a 3-second countdown should start **immediately**.

After that, go back to State 1 and turn off the signal LED. The traffic light should always stay in State 3 for **at least** 5 seconds before starting the 3-second countdown for the Wait button (otherwise it would be unfair to the drivers!).

*Hints:*

1. *Draw out the FSM of the updated system*
2. *The Red Light would be on for at least 8 seconds consecutively*
3. *Note that when you are pressing on the Wait button, the traffic light LEDs should not be affected at all! Therefore, you may need to spawn a new thread for the button detection in order not to block the main thread. Refer to thread.pptx and the video for more information about **threads**.*
4. *The Wait button should be disabled for a reasonable amount of time after a valid press during State 3.*
- 5.

#### How to detect long button presses?

```
if GPIO.input(left_button)==False:
    # Record the start time
    start_time = time.time()
    while (GPIO.input(left_button)==False):
        # do nth!
        pass
    # Record the end time
    end_time = time.time()
    interval = end_time - start_time
    print('Left button pressed for ', interval)
```



### How to use global variables? (share variables between threads)

```
# global variable
var = False
def f():
    global var
    print(var)
```

### How to use Threads?

```
# We will need to use the threading library
import threading

# Define a new subclass of the default Thread class
class buttonThread (threading.Thread):

    # Override the __init__() method if you want to add more arguments
    def __init__(self, threadID, name):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name

    # Override the run() method to implement the thread behavior
    def run(self):
        print ('Starting ' + self.name + ' for button detection.')
        ...
        try:
            ...
        except:
            print('Thread Exception.')

...
if __name__ == '__main__':
    # Inside main thread
    ...
    # Create an instance of the Thread subclass you have just created
    thread1 = buttonThread(1, "Thread-1")

    # Daemon threads will be stopped when the main program is terminated
    # This must be set before calling the start() method!!
    thread1.daemon = True

    # Start a new thread by invoking start(), then the run() method will
    be called
    thread1.start()
```

---

*Task 4: [Submit your demonstration video and completed \*\*traffic light.py\*\* to Moodle.](#)*

---

Congratulations! That's all for GPIO Programming Session. ☺

Please remember to keep the traffic light circuits and the program you wrote, as we are going to reuse them again!

---

#### *References*

1. <http://reefwingrobotics.blogspot.hk/2017/02/raspberry-pi-and-towerpro-sg90-micro.html>
2. <https://learn.sparkfun.com/tutorials/raspberry-gpio>
3. <https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins>
4. <https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/>
5. [https://www.tutorialspoint.com/python/python\\_multithreading.htm](https://www.tutorialspoint.com/python/python_multithreading.htm)