Raspberry Pi GPIO worksheet

There is a 26-way connector on the side of the Raspberry Pi which provides access to the GPIO (General Purpose Input Output) ports on the processor.

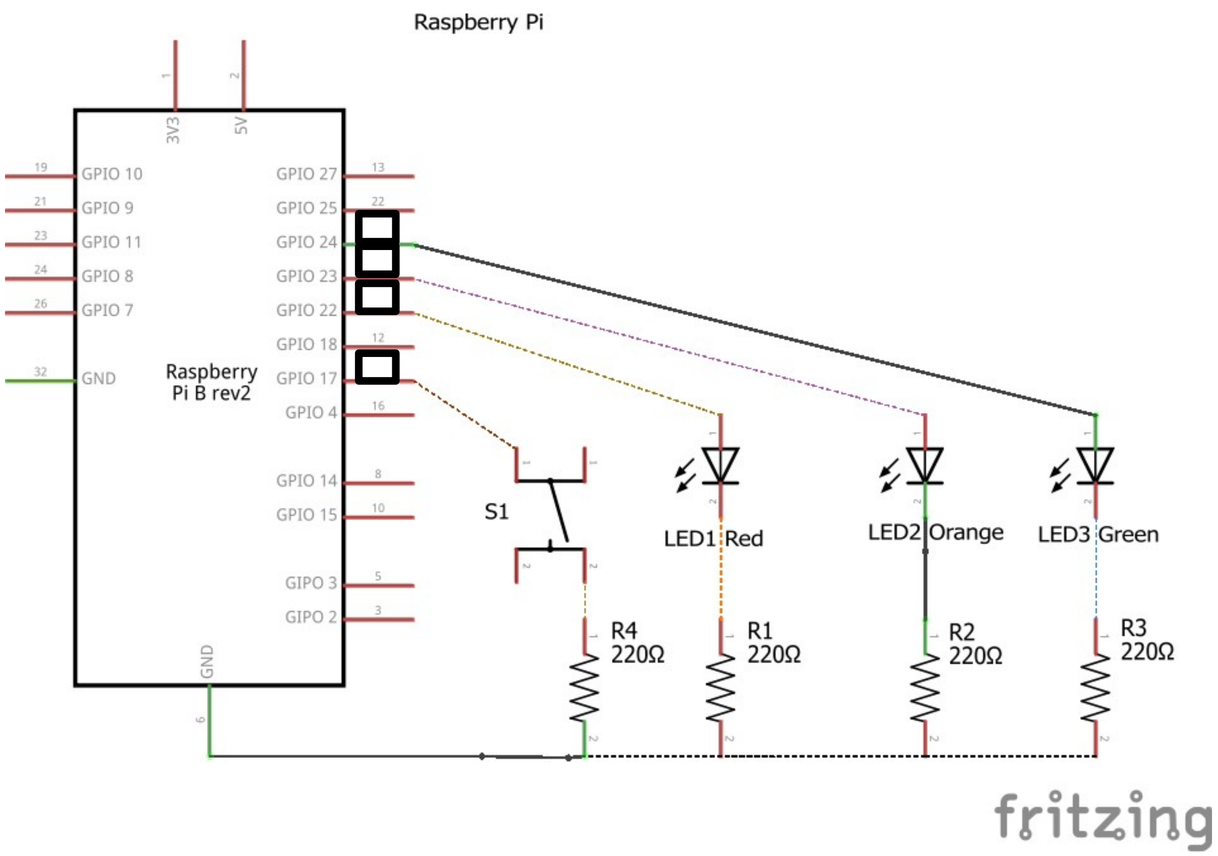
We will use the GPIO ports on the Raspberry Pi to create a traffic light sequence using appropriate coloured LEDs and an optional switch.

The GPIO ports

Each GPIO port can be configured as either an input or output. The GPIO ports can only provide a small current. When connecting an LED then it is necessary to limit the current to prevent damage to the Raspberry Pi. We will therefore use 220Ω resistors along with the LEDs to limit the current through the ports. We will also use a resistor for the input switch. The switch would not need a resistor as it is an input, but one needs to be included in case the output is configured as an output with the switch pressed.

Circuit diagram

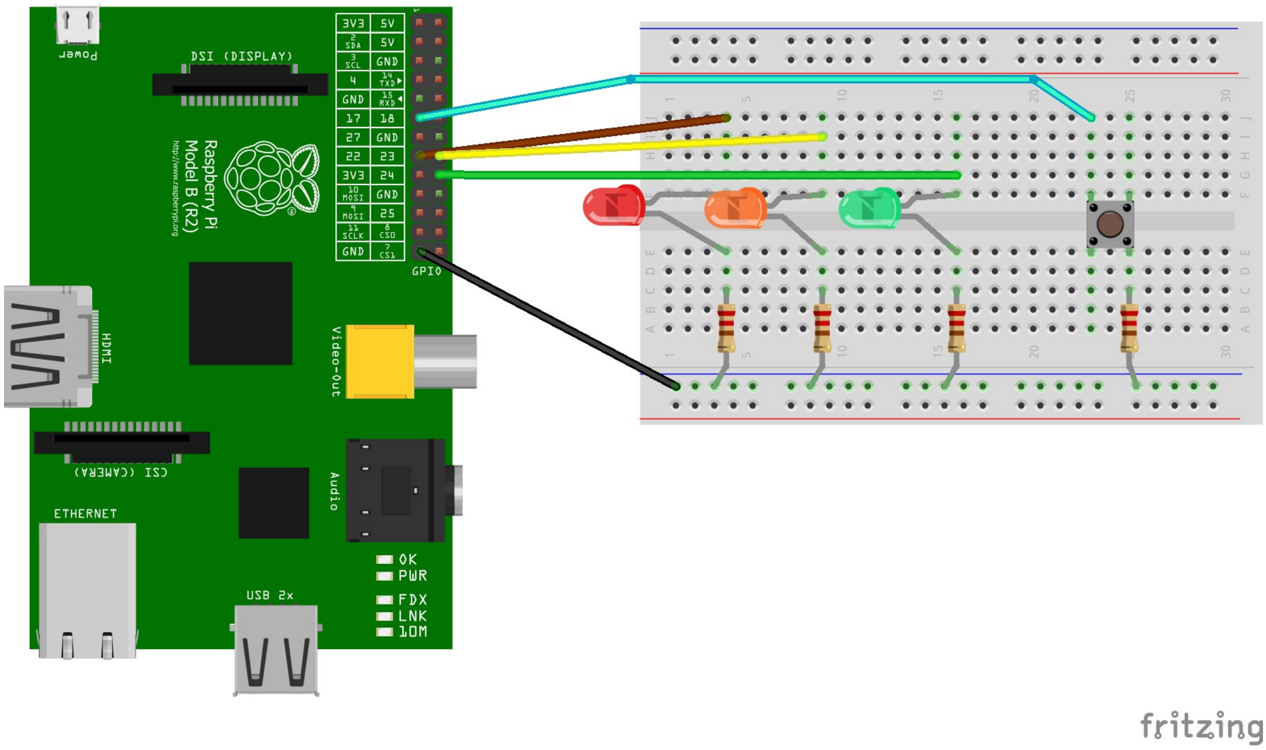
This diagram shows the circuit diagram. It shows which port from the Raspberry Pi GPIO is connected to each LED and then through the appropriate resistor to Gnd (0v). We will use GPIO ports 22, 23 and 24 for the LEDs and GPIO 17 for the switch. From a diagram of the GPIO port layout find out what the physical pin number is for these ports and write them in the circuit diagram.



Wiring up the circuit

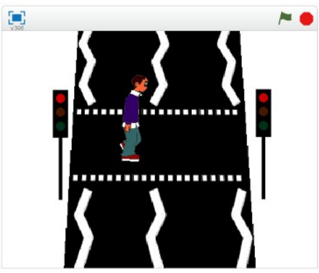
This diagram shows the components plugged into a breadboard with jumper leads to connect to the Raspberry Pi. If you don't have a breadboard then it is also possible to create a circuit using crocodile clips to hold the wires together, or using female jumper cables.

You should have already identified how the numbering of the GPIO connector works. As a reminder looking at the diagram below it is numbered left to right and them top to bottom. So pin 1 is top left, pin 2 is top right, pin 25 is bottom left and pin 26 is bottom right.



It is important that the LEDs are connected the correct way around, otherwise they will not work. The longer lead is normally the positive side of the LED (anode), or if there is a flat part (on a standard round LED) then that is normally the negative side (cathode). The resistors can be wired either way around – they should be labelled red (2), red (2), brown (x10), the 4th band is the tolerance (there is sometimes a 5th stripe).

There are different switches that can be used. The one shown is a single push switch with 4 terminals. The two left hand pins are connected together as are the two right hand pins. Any push-to-make switch can be used as long as one side connects to the appropriate GPIO pin and the other to the resistor.



Testing the outputs with the Python command shell

To access the GPIO ports then we need to run with root (superuser) permissions. This is easiest by running IDLE as root from a terminal shell.

```
sudo idle
```

IDLE starts in the shell mode where you can enter the following commands directly to test the circuit.

First import the GPIO module and setup to use the GPIO port numbering scheme.

```
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)
```

Each GPIO port needs to be setup as an output (GPIO.OUT) or an input as appropriate (GPIO.IN).

The following will test the first LED by setting it as an output, turning the output on (set to true) and then turn it off again (set to false).

```
>>> GPIO.setup(22, GPIO.OUT)
>>> GPIO.output(22, True)
>>> GPIO.output(22, False)
```

Repeat this for all 3 LEDs.

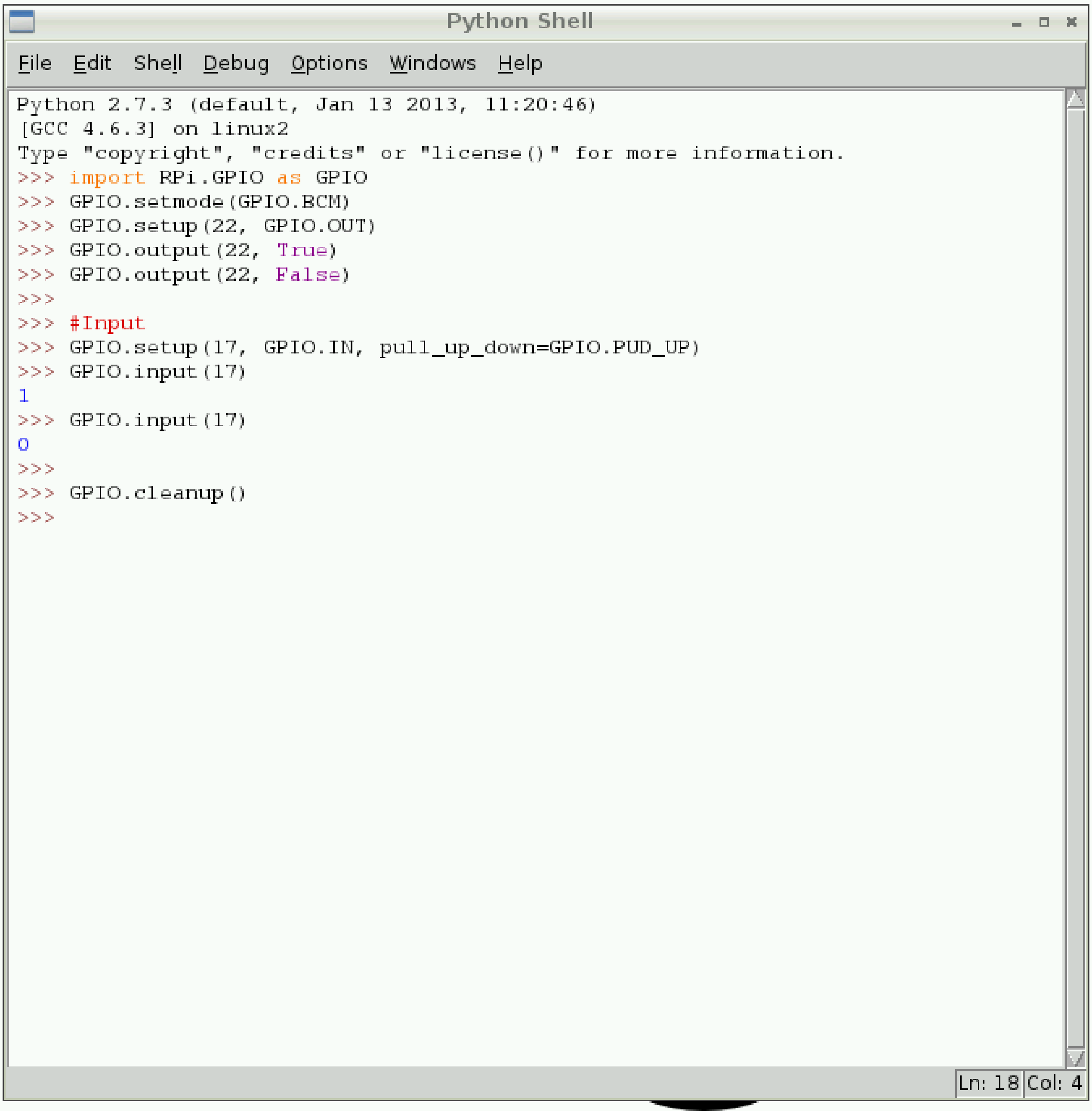
To test the input we need to set the pin up as an output. We also need to enable the internal pull-up resistors otherwise the input will be floating when the buton is not pressed.

```
>>> GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
>>> GPIO.input(17)
1
>>> GPIO.input(17)
0
```

The first time the input command is run is without the button pressed, which results in a 1 thanks to the internal pull-up resistor. The second time is when the switch is pressed which pulls the input down to a 0.

Finally the cleanup should be run to release the GPIO ports, otherwise there will be a warning when the program is next run.

The following screenshot shows these commands as run in the shell. Note that only one LED was tested in this instance, but you should test all 3 LEDs can be turned on and off using the appropriate GPIO port numbers.



You can now choose File > New File to create a program that can be saved.

If you are familiar with Python programming then you can create a program that lights the LEDs in sequence. If you have a switch connected then you could create a pedestrian crossing that responds to the push from a pedestrian, alternatively you could create a traffic light that changes between red and green after a period of time.

If you need help then ask for the next page, which provides some example code.

Creating the traffic light program

If you haven't created your own program you can use the following code. Take care when copying this into the IDLE editor.

trafficlight.py

```
# GPIO traffic light program

# pin numbers
GPIO_RED = 22
GPIO_AMBER = 23
GPIO_GREEN = 24
GPIO_SWITCH = 17

# time delay in secs
TIME_CHANGE = 1
TIME_RED = 5
TIME_GREEN = 5    #only req if not using switch

# import RPi.GPIO module
import RPi.GPIO as GPIO
# import time module used for sleep
import time

# Use GPIO pin numbering disable in-use warnings
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Setup relevant pins
GPIO.setup(GPIO_RED, GPIO.OUT)
GPIO.setup(GPIO_AMBER, GPIO.OUT)
GPIO.setup(GPIO_GREEN, GPIO.OUT)
GPIO.setup(GPIO_SWITCH, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# start with red light on
GPIO.output(GPIO_RED, True)
GPIO.output(GPIO_AMBER, False)
GPIO.output(GPIO_GREEN, False)


# Loop keeps running
while 1:
      # red light on - sleep for set period of time
      time.sleep (TIME_RED)

      # Red and amber (puffin crossing - so don't flash)
      GPIO.output (GPIO_AMBER, True)

      # sleep for change time
      time.sleep (TIME_CHANGE)

      # Change to green
      GPIO.output(GPIO_RED, False)
      GPIO.output(GPIO_AMBER, False)
      GPIO.output(GPIO_GREEN, True)

      # sleep for button to press (or replace following code with a sleep if not
using switch)
```

```
# loops until GPIO becomes "False" ie switch pressed
while GPIO.input(GPIO_SWITCH):
      # sleep used to reduce processor usage
      #sleep for 1/4 sec between checking switch
      time.sleep (0.25)

# Change to amber
GPIO.output(GPIO_AMBER, True)
GPIO.output(GPIO_GREEN, False)

# sleep for change time
time.sleep (TIME_CHANGE)

# Change to red
GPIO.output(GPIO_RED, True)
GPIO.output(GPIO_AMBER, False)
```

The code includes lots of comments prefixed with #, which help to explain what is happening. Read through the code before running to work out how it works.

Test the code by choosing "Run Module" from the menu.

If there are no errors then there will not be any output shown in the shell. Any errors will indicate the approximate line number in the code where to start looking.

If there are no errors, but the code doesn't behave as you expect then it can be useful to add print commands within the code to "print" a message to the shell screen.

New improved traffic lights

You should now have a basic traffic light program working, but it's still not quite complete.
Here are some suggestions for improvements:

- Add two more LEDs to represent the green and red person lights.
- Add another LED to represent the "WAIT" button for the pedestrian [this would go on between the button being pressed and the green person LED coming on].*
- You may also want to add a delay if the button was pressed recently, otherwise pedestrians pressing the button could make the traffic stop almost continuously. [Hint: if you store the value of time.time() when someone crosses you can then use that to determine if it's been long enough since the last person crossed].
- You could configure the amber LED to flash instead of having red and amber (pelican vs. puffin crossing).
- For a bigger challenge how about adding a beeping noise to help the visually impaired know when it's safe to cross. [Hint: a transistor or similar will be required to connect a buzzer to the GPIO ports, or you could use an external speaker on the Raspberry Pi audio output]

* Note that to prevent damage to the Raspberry Pi then you should not turn on more than about five LEDs at once. This will add a sixth, but as long as the logic is correct there should not be more than four LEDs lit at one time.

Beyond traffic light LEDs

Apparently some people find other things more interesting than LED traffic lights!

Fear not – we used traffic lights as it's an example because it's easy to code and almost everyone knows how traffic lights work. Well at least those from the UK should recognise this sequence – other countries have may have a different light sequence.

Now that you know how to turn the GPIO pins on and off you can apply this to whatever interest you have.

- Control motors on a robot, model car, train, boat or quad-copter
- Fire Nerf missiles from a launcher
- Control disco lights
- Create your own intruder alarm system for your bedroom
- Have programs interact with the real world – flash lights when you score a point, or have your desk vibrate when you hit a block in Minecraft.

The key thing to consider is that the amount of current that can be provided through the GPIO port is very small. For the examples above you will need to look at providing some kind of buffer to increase the power to be enough to switch the loads. There is lots of information on the internet, but the following should give you some starting points of where to look.

For a motor then you should look at a H-Bridge driver IC such as the Raspberry Pi motor controller created by RyanTeck, a simple transistor switch or buffer may be enough for switching more LEDs. Or you could try one of the Raspberry Pi add-on boards.

If you want to vary the amount of power provided through the GPIO then look at using PWM (Pulse Width Modulation) which is available through the RPi.GPIO module.