Workshop 15: GPIO Zero

Written by Jack Kelly

Learn the basics of physical computing on Raspberry Pi using the *GPIO Zero* Python library.

Difficulty: Introductory workshop

Contents

0 Introduction

If you've controlled electronics using the Raspberry Pi's GPIO before, you probably used a Python library called *RPi.GPIO* to write your programs. GPIO Zero is an alternative GPIO library for Python, designed to be easier to use than RPi.GPIO, whilst having the same capabilities.

Today, we'll learn how to create simple breadboard circuits using the Raspberry Pi's GPIO, then we'll write Python programs that control those circuits using the GPIO Zero library.

This is an introductory workshop, for people who have tried basic programming in Python before.

How to use these booklets

The aim of these booklets is to help you attempt these workshops at home, and to explain concepts in more detail than at the workshop. You don't need to refer to use these booklets during the workshop, but you can if you'd like to.

When you need to make changes to your code, they'll be presented in *listings* like the example below. Some lines may be repeated across multiple listings, so check the line numbers to make sure you're not copying something twice.

```
1 from gpiozero import LED
2 led = LED(17)
```

Occasionally, a concept will be explained in greater detail in *asides*, like the one below. You can read these as you wish, but they're not required to complete the workshop.

Resistors

Resistors are most commonly used to limit the amount of current flowing through part of an electrical circuit.

For example we use resistors in series with LEDs, as otherwise they could draw so much power that they would destroy themselves.

What you'll need

To create the circuits in this workshop, you will need:

- A solderless breadboard and jumper wires
- Red, yellow and green LEDs
- A push-button tactile switch
- Suitable resistors for the LEDs and button
- A 3-5V **DC** piezo buzzer

All of the software you for this workshop is pre-installed on recent versions of Raspbian.

Everything else

These booklets were created using LaTeX, an advanced typesetting system used for several sorts of books, academic reports and letters.

If you'd like to have a look at using LaTeX, We recommend looking at TEXstudio, which is available on most platforms, and also in the Raspbian repository.

To allow modification and redistribution of these booklets, they are distributed under the CC BY-SA 4.0 License. Latex source documents are available at http://github.com/McrRaspJam/booklet-workshops

If you get stuck, find errors or have feedback about these booklets, please email me at: jam@jackjkelly.com

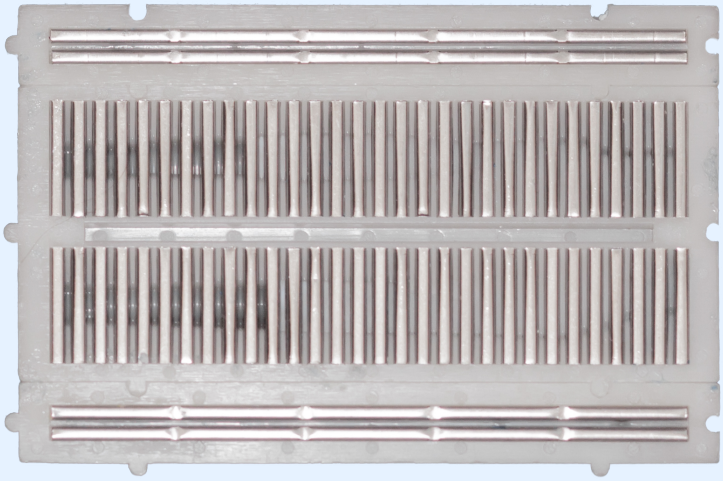
1 Creating a Simple Circuit

We'll start with a very simple circuit. Let's build the circuit first, then we'll take a look at how it works afterwards.

Solderless Breadboards

A common tool for building small electrical circuits is the *solderless breadboard*.

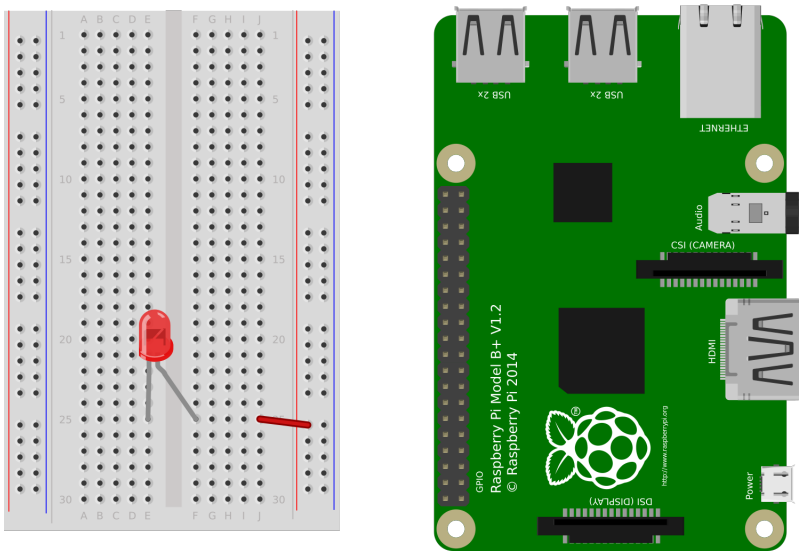
The breadboard consists of rows of square sockets, in which electrical components can be easily placed, with metal wire 'lanes' running underneath the breadboard to make electrical connections.



The layout of 'lanes' under a standard breadboard is shown above. Components need to be carefully placed to ensure a full circuit is made.

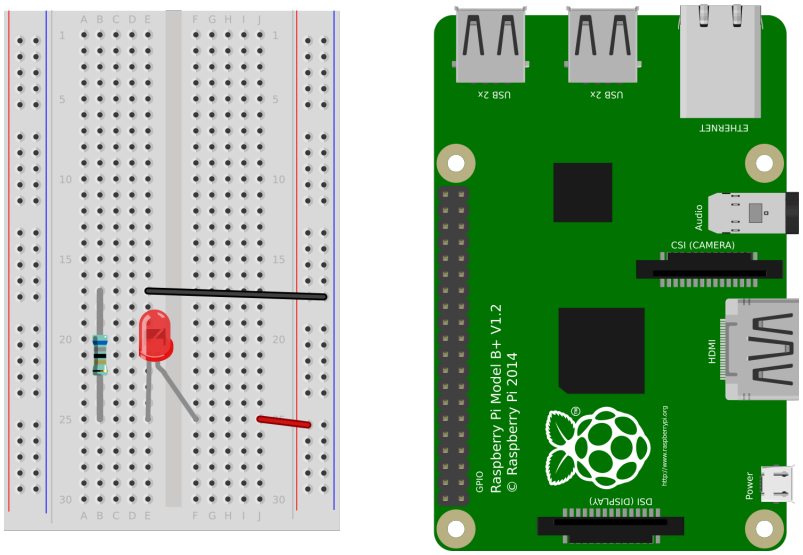
Take your breadboard, and place it next to your Raspberry Pi. Then, take a red LED and place it across the central gap in your breadboard, as shown. (The longer lead of the LED should be on the side nearest your Raspberry Pi)

Then, connect the positive side of the LED to the positive power rail on the breadboard using a male-to-male jumper cable, as shown.



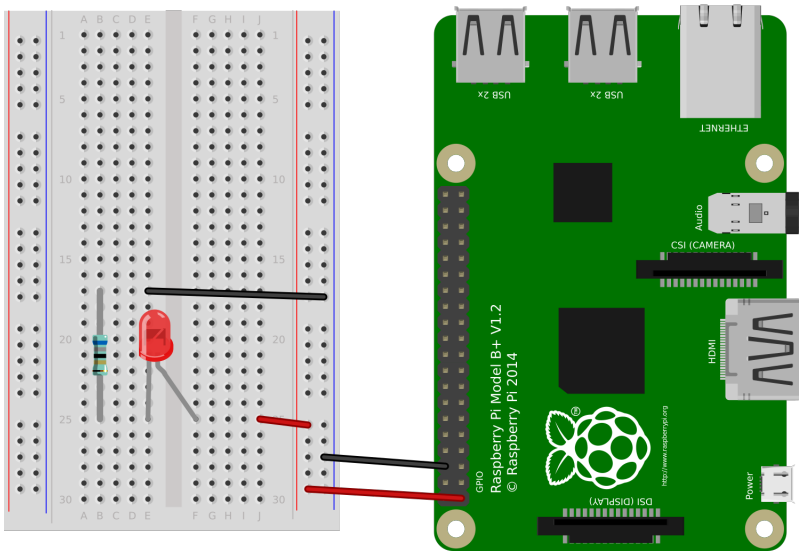
We then place a 68Ω resistor, connecting from the negative side of the LED to any other row, as shown.

Then, connect the other end of the resistor to the negative power rail on the breadboard using another jumper cable.



We've now created a full circuit from positive to negative. To power the circuit, we need to connect it to the power pins on our Raspberry Pi.

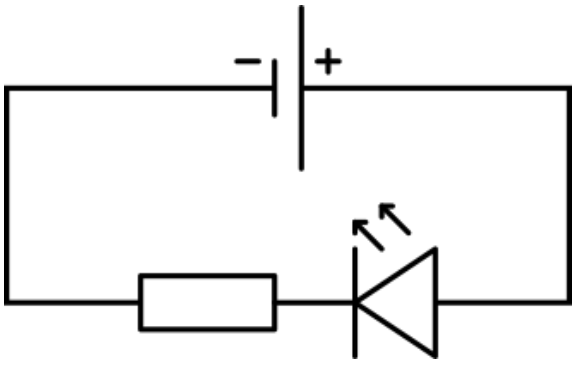
Use a female-to-male jumper cable to connect the **positive** power rail on the breadboard to **3V3**, and the **negative** rail to **GND**.



If all the connections are correct, your LED should light up.

Our Circuit

We've just created a simple circuit that powers a single LED. If you've been taught circuit schematics before, the circuit we just built might look like this:



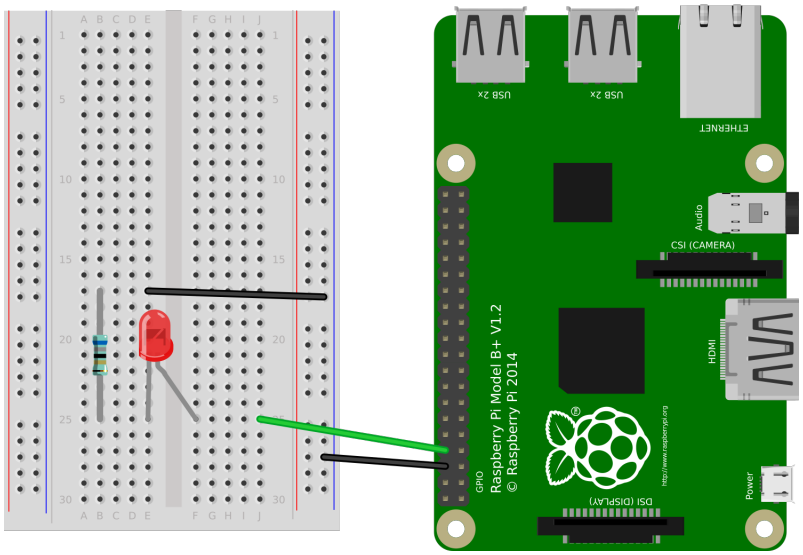
Right now our Raspberry Pi computer isn't controlling the LED, we're simply using the power supply to light the LED; this means our Raspberry Pi is currently just a $35 battery.

In order to control the LED, we'll need to alter the circuit to use GPIO pins.

1.1 Creating a GPIO circuit

Altering the circuit is straightforward. We will use one of the GPIO pins as an *output*, meaning it will supply the power to the LED.

Remove the jumper cables connecting to the positive power rail of the breadboard, and place a single female-to-male connecting the LED to a GPIO pin, as shown.



Programming the LED

Now we have a simple GPIO circuit, we can have a go at programming our LED.

Boot to the Raspbian desktop if you haven't yet done so, and from the application menu open **IDLE 3**. Click `File` → `Open...` and open `1_led.py`.

1_led.py

```
1 from gpiozero import LED
2 from time import sleep
```

This program doesn't yet do anything, but it has loaded *GPIO Zero* for us to use.

First, we start by defining our LED. Make a note of the GPIO number you connected your LED to, (we used GPIO14) and enter the following line, substituting the pin number you connected:

```
3
4 led = LED(14)
```

Then use the *function* `on()` to turn on your LED:

```
5
6 led.on()
```

Now try running your program using `Run` → `Run Module`. When your program starts running your LED should turn on.

Let's make our program a little more interesting by making our LED blink. We'll start by doing this the manual way, using a loop. Alter your program so it looks as follows.

```
1 from gpiozero import LED
2 from time import sleep
3
4 led = LED(14)
```

```
5
6  while True:
7          led.on()
8          sleep(1)
9
10         led.off()
11         sleep(1)
```

Run your program again, and the LED should start blinking.

How can you alter the program to make the blinking twice as fast? how about making it blink so that the light is on for 2 seconds and off for 1?

That's not all though. GPIO Zero has lots of useful functions for each type of device. For LEDs, a function called `blink()` is provided to make it easy to blink an LED. We can replace our loop with a single line:

```
4  led = LED(14)
5  led.blink()
```

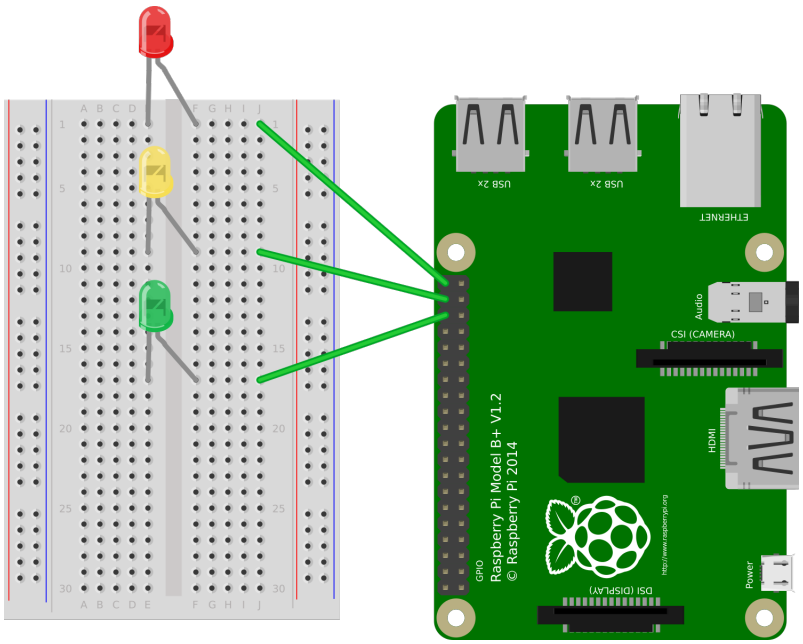
GPIO Zero functions

GPIO Zero supports a lot of different devices, and each device has a number of useful functions for you to use.

The easiest way to see them all is in the online documentation, which can be found at gpiozero.readthedocs.io

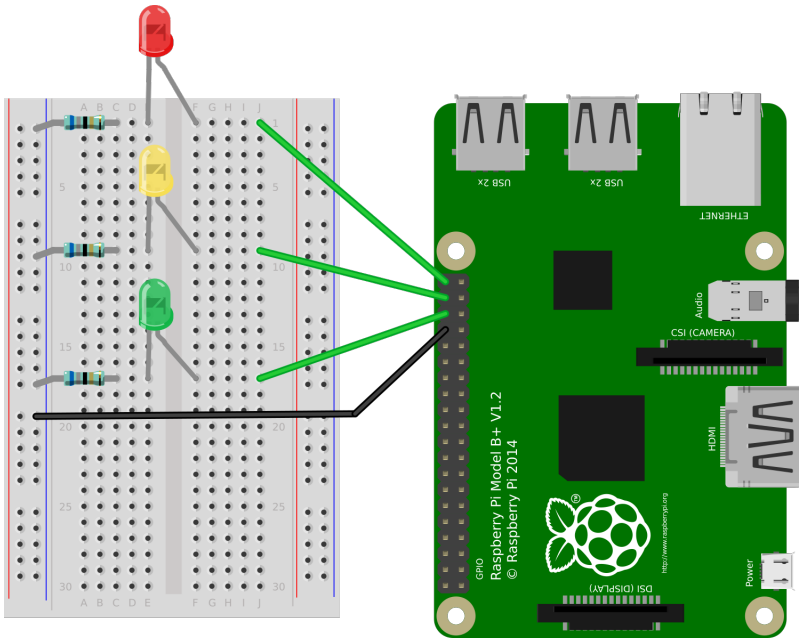
2 Traffic Lights

Start by placing the 3 LEDs across the centre of the breadboard, as we did before, and connect each one to a GPIO pin using a jumper cable.

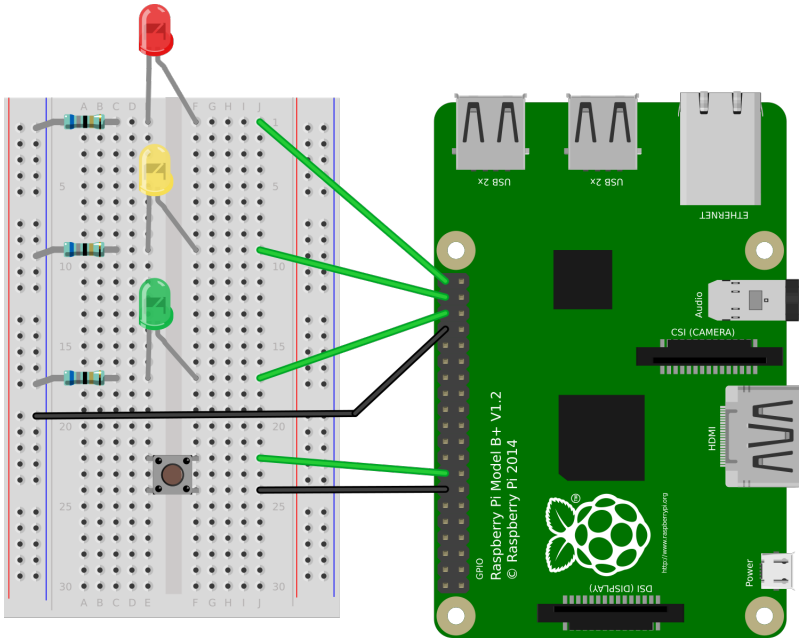


Each LED still needs a resistor. To save on jumper wires, we'll use the resistors to bridge each LED cathode directly to the negative rail, as shown below.

Then, use a jumper cable to connect the negative rail to a GND pin on the Raspberry Pi.



We'll also place a button on the breadboard in its own circuit. Connect one side of the switch to another GPIO pin, and the other side to a GND pin.



We now have working circuits for our traffic lights.

Designing the traffic lights

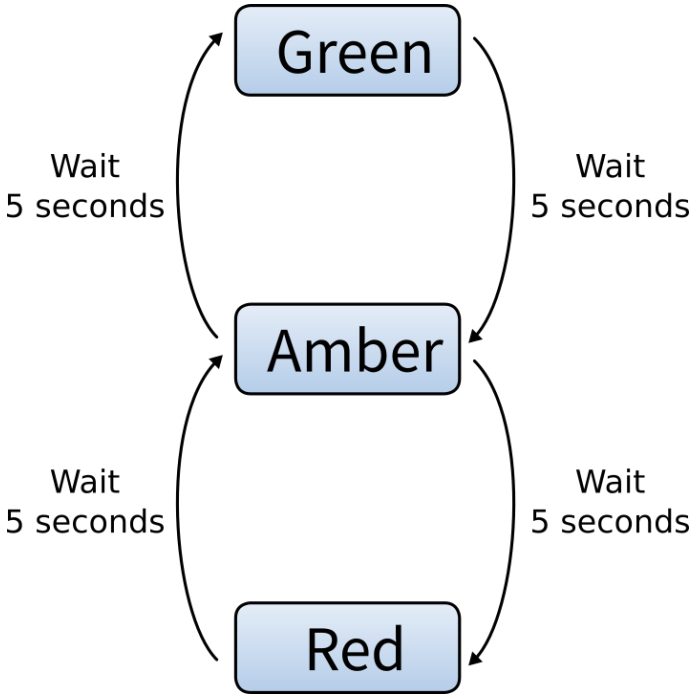
Now we have a circuit, we need to design how our traffic lights will behave. We'll be designing them as a **pelican crossing**.

Figure 1: A pelican crossing

Ignoring the button to begin with, we want our traffic lights to stay on green for most of the time, but switch to red occasionally to allow pedestrians to cross.

One tool that is used for these kinds of design problem is a *state transition diagram*. We can use one of these to break down the traffic light sequence into simple steps.

The following is a state transition diagram for a set of traffic lights. The squares are the 'states'—in our case the colour of the lights—and the lines are the transitions—the thing that determines when states switch, and to where.



This state transition diagram isn't very accurate, so you should think of some ways to improve it before you start programming.

- Currently, all of the lights are on for the same amount of time. How long should each transition take?
- Are there any times when two lights should be on? (You may need another state)

Programming the traffic lights

From IDLE, click `File` → `Open...` and open `2_trafficlight.py`. Fill in the device definitions with the GPIO pins you chose for your circuit. It should look like the following before continuing.

```
1 from gpiozero import LED, Button
2 from time import sleep
3
4 led_red = LED(21)
5 led_amber = LED(20)
6 led_green = LED(16)
7 button = Button(23)
```

Because our traffic lights run continuously, we'll probably want an infinite loop in our program.

```
8
9  while True:
```

We'll start in the 'green' state, by turning on that LED, and add the first transition by adding a `sleep()` delay following this.

```
10         #Cars on Green
11         led_green.on()
12         sleep(5)
```

The rest is up to you, follow this line with the action for your next state, then keep continuing until you have looped back to the starting state.

Adding the button

Once you're happy with your program so far, you can now implement the pedestrian button.

Start by modifying your state transition diagram. Instead of waiting, one of your transitions will require the button to be pressed.

You can program you button press in a couple of ways. The traditional way of reading GPIO inputs still works:

```
1  if button.is_pressed == true:
2      led_green.off()
```

However, you can also use the function `wait_for_button()`, which is like a `sleep()` that lasts until the button is pressed.

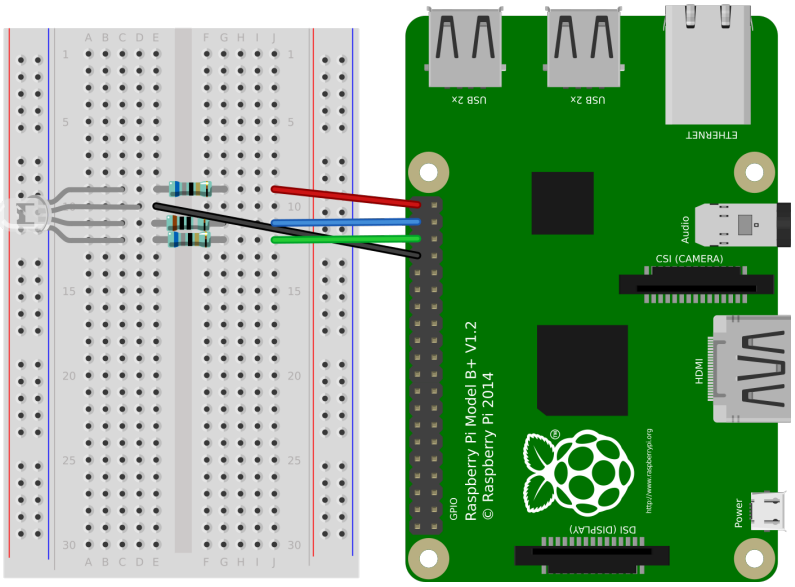
```
1  button.wait_for_press()
2  led_green.off()
```

3 More Devices (RGB LEDs)

An RGB LED is just three separate LEDs placed in the same component package. A standard through-hole RBB LED will have four pins, and can be either *common anode* or *common cathode*.

Our LEDs are cathode, meaning they have three $3.3V$ pins, and a single GND pin.

We can create the circuit for the LED as follows. The individual diodes still require resistors, so we will place them at the positive side of our LED, one for each colour pin.



Note that because the blue LED has a much higher forward voltage ($3.3V$) we use a different 1Ω resistor for that pin.

Programming the RGB LED

You can program the RGB LED as three separate LEDs in your program—you could try running your traffic light program using the RGB LED—however, GPIO Zero offers a separate RGBLED type, which makes colour mixing much easier.

You can use the following program to test your RGB LED.

```
1 from gpiozero import RGBLED
2
3 led = RGBLED(21, 16, 20)
4 led.color = (1, 0, 1)
```

If you're up for a challenge: Reading the online documentation for the function `pulse()`, try writing a program which cycles through every colour in the rainbow.

4 Morse Code Challenge

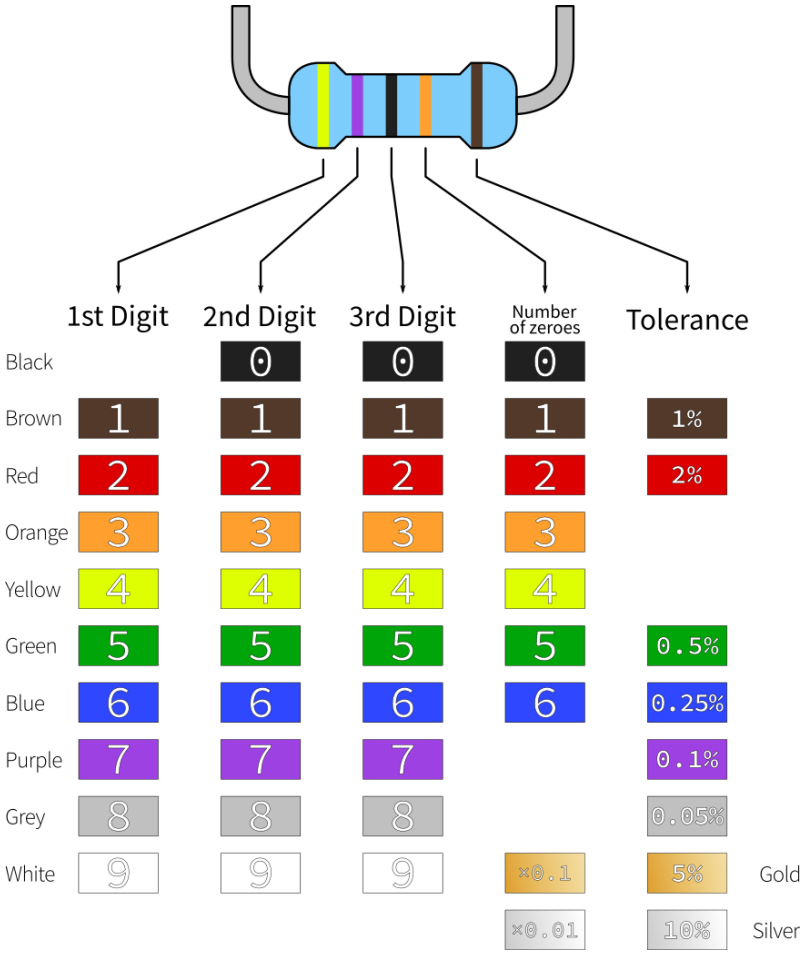
This section will be added after the next Jam

Appendices

A Resistor Values

The resistance of axial through-hole resistors can be determined from the coloured bands printed on their exterior.

A guide for 5-band resistors is shown below, depicting a $470k\Omega$ ($470,000\Omega$) resistor.



B International Morse Code

Standard rules

- The length of a dot is 1 unit.
- The length of a dash is 3 units.
- The space between parts of the same letter is 1 unit.
- The space between letters is 3 units.
- The space between words is 7 units.

Beginner's rules

This rule-set is adjusted to make it easier to listen to and write down incoming Morse code messages. It's recommended that you write your morse code program to use these timings.

- A unit is $\sim$**0.25** seconds.
- The length of a dot is 1 unit.
- The length of a dash is 4 units.
- The space between parts of the same letter is 1 unit.
- The space between letters is 8 units.
- The space between words is $\sim$20 units.

Characters

Alphabet

A	· —
B	— · · ·
C	— · — ·
D	— · ·
E	·
F	· · — ·
G	— — ·
H	· · · ·
I	· ·
J	· — — —
K	— · —
L	· — · ·
M	— —
N	— ·
O	— — —
P	· — — ·
Q	— — · —
R	· — ·
S	· · ·
T	—
U	· · —
V	· · · —
W	· — —
X	— · · —
Y	— · — —
Z	— — · ·

Numeral

0	— — — — —
1	· — — — —
2	· · — — —
3	· · · — —
4	· · · · —
5	· · · · ·
6	— · · · ·
7	— — · · ·
8	— — — · ·
9	— — — — ·

Punctuation

.	·–·–·–
,	––··––
?	··––··
!	–·–·––
(	–·––·
)	–·––·–
:	–––···
;	–·–·–·
=	–···–
+	·–·–·
-	–····–
”	·–··–·
’	·––––·

Additional ITU recognised characters exist, these can be found at http://en.wikipedia.org/wiki/Morse_code