Traffic lights

For this worksheet you'll need a breadboard, three LEDs, a button, a buzzer, and the necessary jumper cables and resistors. You can purchase these individually, or get everything you need in the CamJam EduKit.

Wiring

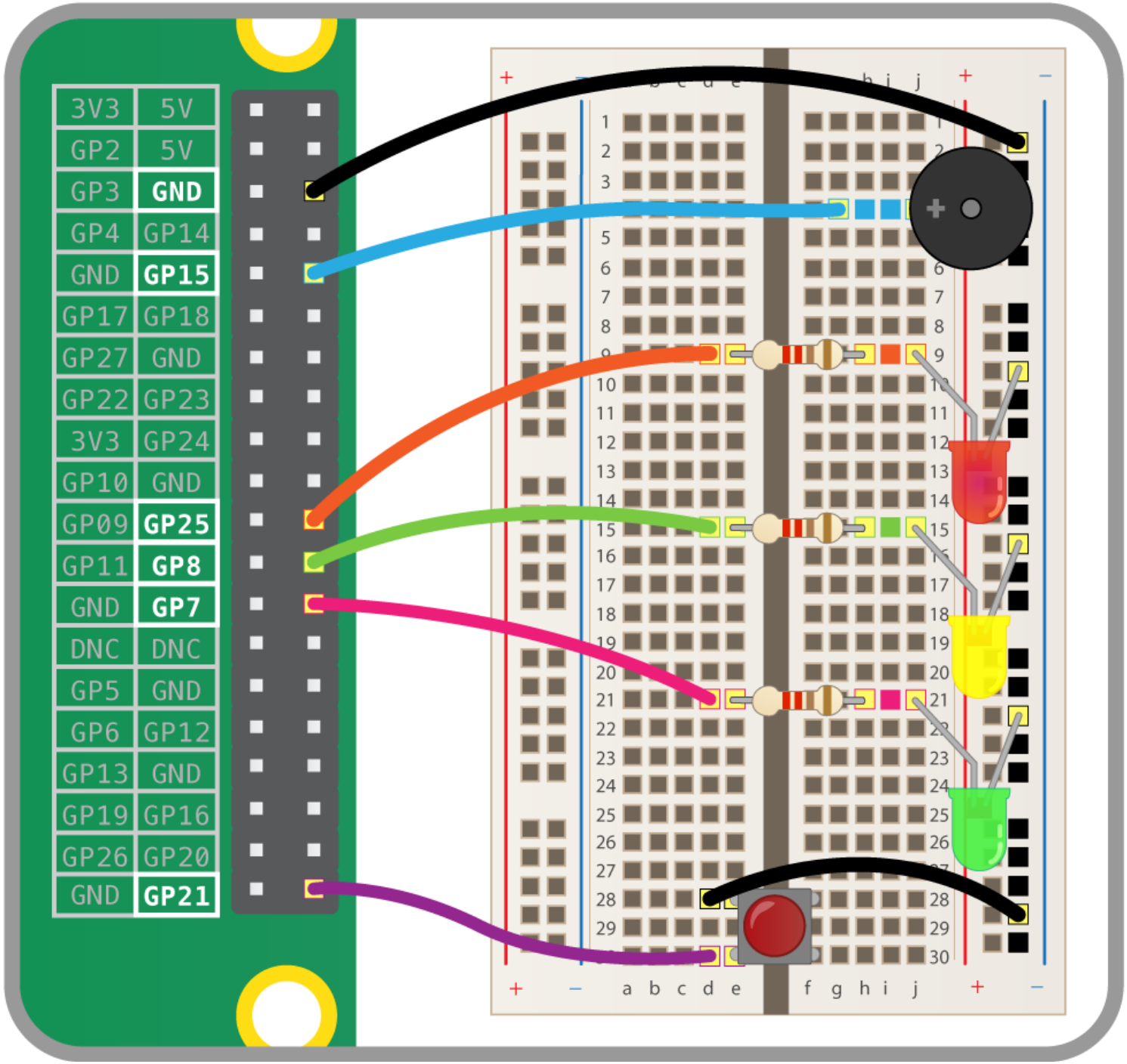
To get started, you'll need to place all the components on the breadboard and connect them to the appropriate GPIO pins on the Raspberry Pi.

1. First, you need to understand how each component is connected:

    - A push button requires 1 ground pin and 1 GPIO pin
    - An LED requires 1 ground pin and 1 GPIO pin, with a current limiting resistor
    - A buzzer requires 1 ground pin and 1 GPIO pin

Each component requires its own individual GPIO pin, but components can share a ground pin. We will use the breadboard to enable this.

2. Place the components on the breadboard and connect them to the Raspberry Pi GPIO pins, according to the following diagram:


3V3 5V
GP2 5V
GP3 GND
GP4 GP14
GND GP15
GP17 GP18
GP27 GND
GP22 GP23
3V3 GP24
GP10 GND
GP09 GP25
GP11 GP8
GND GP7
DNC DNC
GP5 GND
GP6 GP12
GP13 GND
GP19 GP16
GP26 GP20
GND GP21
a b c d e
f g h i j

Note that the row along the long side of the breadboard is connected to a ground pin on the Raspberry Pi, so all the components in that row (which is used as a ground rail) are hence connected to ground.

3. Observe the following table, showing which GPIO pin each component is connected to:

Component	GPIO pin
Button	21
Red LED	25
Amber LED	8
Green LED	7
Buzzer	15

Dive into Python

Open the Python application IDLE and get started by testing out the button.

1. Open **Python 3** from the main menu:

2. Create a new file by clicking **File** > **New File**. This will open up a second window.

3. Save the new file straight away by clicking **File** > **Save**; name the file `trafficlights.py` and save it in your home folder.

4. Enter the following code:

```
from gpiozero import Button

button = Button(21)

while True:
    print(button.is_pressed)
```

In GPIO Zero, you create an object for each component used. Each component interface must be imported from the `gpiozero` module, and an instance created on the GPIO pin number to which it is connected.

5. Save and run the code by pressing `Ctrl + S` and `F5`.

6. This will bring the original Python window into focus and will be constantly printing `False`. When you press the button this will switch to `True`, and when you let go it will return to `False`.

   `button.is_pressed` is a property of the `button` object, which provides the state of the button (pressed or not) at any given time.

7. Now return to the code window and modify your `while` loop to show the following:

```
while True:
    if button.is_pressed:
        print("Hello")
    else:
        print("Goodbye")
```

8. Run the code again and you'll see "Hello" printed when the button is pressed, and "Goodbye" when the button is not pressed.

9. Modify the loop again:

```
while True:
    button.wait_for_press()
    print("Pressed")
    button.wait_for_release()
    print("Released")
```

10. When you run the code this time, nothing will happen until you press the button, when you'll see "Pressed", then when you let go you'll see "Released". This will occur each time the button is pressed, but rather than continuously printing one or the other, it only does it once per press.

Add an LED

Now you'll add an LED into the code and use GPIO Zero to allow the button to determine when the LED is lit.

1. In your code, add to the `from gpiozero import...` line at the top to also bring in `LED`:

```
from gpiozero import Button, LED
```

2. Add a line below `button = Button(21)` to create an instance of an `LED` object:

```
led = LED(25)
```

3. Now modify your `while` loop to turn the LED on when the button is pressed:

```
while True:
    button.wait_for_press()
    led.on()
    button.wait_for_release()
    led.off()
```

4. Run your code and the LED will come on when you press the button. Hold the button down to keep the LED lit.

5. Now swap the `on` and `off` lines to reverse the logic:

```
while True:
    led.on()
    button.wait_for_press()
    led.off()
    button.wait_for_release()
```

6. Run the code and you'll see the LED stays on until the button is pressed.

7. Now replace `led.on()` with `led.blink()`:

```
while True:
    led.blink()
    button.wait_for_press()
    led.off()
    button.wait_for_release()
```

8. Run the code and you'll see the LED blink on and off until the button is pressed, at which point it will turn off completely. When the button is released, it will start blinking again.

9. Try adding some parameters to `blink` to make it blink faster or slower:

    - `led.blink(2, 2)` - 2 seconds on, 2 seconds off
    - `led.blink(0.5, 0.5)` - half a second on, half a second off

- `led.blink(0.1, 0.2)` - one tenth of a second on, one fifth of a second off

`blink`'s first two (optional) parameters are `on_time` and `off_time`: they both default to 1 second.

Traffic lights

You have three LEDs: red, amber, and green. Perfect for traffic lights! There's even a built-in interface for traffic lights in GPIO Zero.

1. Amend the `from gpiozero import...` line to replace `LED` with `TrafficLights`:

```
from gpiozero import Button, TrafficLights
```

2. Replace your `led = LED(25)` line with the following:

```
lights = TrafficLights(25, 8, 7)
```

   The `TrafficLights` interface takes three GPIO pin numbers, one for each pin: red, amber, and green (in that order).

3. Now amend your `while` loop to control the `TrafficLights` object:

```
while True:
    button.wait_for_press()
```

```
    lights.on()
    button.wait_for_release()
    lights.off()
```

The `TrafficLights` interface is very similar to that of an individual LED: you can use `on`, `off`, and `blink`, all of which control all three lights at once.

4. Try the `blink` example:

```
while True:
    lights.blink()
    button.wait_for_press()
    lights.off()
    button.wait_for_release()
```

Add a buzzer

Now you'll add your buzzer to make some noise.

1. Add `Buzzer` to the `from gpiozero import...` line:

```
from gpiozero import Button, TrafficLights, Buzzer
```

2. Add a line below your creation of `button` and `lights` to add a `Buzzer` object:

```
buzzer = Buzzer(15)
```

3. `Buzzer` works exactly like `LED`, so try adding a `buzzer.on()` and `buzzer.off()` into your loop:

```
while True:
    lights.on()
    buzzer.off()
    button.wait_for_press()
    lights.off()
    buzzer.on()
    button.wait_for_release()
```

4. `Buzzer` has a `beep()` method which works like `LED`'s `blink`. Try it out:

```
while True:
    lights.blink()
    buzzer.beep()
    button.wait_for_press()
    lights.off()
    buzzer.off()
    button.wait_for_release()
```

Traffic lights sequence

As well as controlling the whole set of lights together, you can also control each LED individually. With traffic light LEDs, a button and a buzzer, you can create your own traffic lights sequence, complete with pedestrian crossing!

1. At the top of your file, below `from gpiozero import...`, add a line to import the `sleep` function:

```
from time import sleep
```

2. Modify your loop to perform an automated sequence of LEDs being lit:

```
while True:
    lights.green.on()
    sleep(1)
    lights.amber.on()
    sleep(1)
    lights.red.on()
    sleep(1)
    lights.off()
```

3. Add a `wait_for_press` so that pressing the button initiates the sequence:

```
while True:
    button.wait_for_press()
    lights.green.on()
    sleep(1)
    lights.amber.on()
    sleep(1)
    lights.red.on()
    sleep(1)
    lights.off()
```

Try some more sequences of your own.

4. Now try creating the full traffic lights sequence:

    - Green on
    - Amber on
    - Red on
    - Red and amber on
    - Green on

    Be sure to turn the correct lights on and off at the right time, and make sure you use `sleep` to time the sequence perfectly.

5. Try adding the button for a pedestrian crossing. The button should move the lights to red (not immediately), and give the pedestrians time to cross before moving back to green until the button is pressed again.

6. Now try adding a buzzer to beep quickly to indicate that it is safe to cross, for the benefit of visually impaired pedestrians.

What next?

- Try adding a second button for the other side of the road. You'll probably need to use GPIO Zero `button.when_pressed` rather than `wait_for_press`, which can only be used for one button at a time.

- Refer to the documentation at gpiozero.readthedocs.org for more information on what can be done with GPIO Zero.
- Continue to the next worksheet on using a Light Dependent Resitor