Jam beginner activities

Here are some short activities which are ideal for beginners at your Raspberry Jam event. The worksheets are included here, and you'll also find them at **rpf.io/jamws** where you can browse them online or print the PDFs. The online versions are available in multiple (spoken) languages.

Make sure you have an up-to-date Raspbian image with Mu installed.

```
sudo apt install mu-editor
```

Worksheets

Traffic lights

You will need:

Learn to control LEDs and code a traffic lights sequence using Scratch, EduBlocks and Python.

Hardware	Software setup
pi-stop or 3 LEDs, resistors and jumper cables	For EduBlocks see **edublocks.org**

Traffic lights controller GUI

You will need:

Create a traffic lights controller GUI (graphical user interface) using Python.

Hardware	Software setup
pi-stop or 3 LEDs, resistors and jumper cables	`sudo pip3 install guizero`

Interactive traffic lights

You will need:

Use LEDs, a button and a buzzer to program an interactive traffic lights sequence with Python.

Hardware	Software setup
pi-stop or 3 LEDs, resistors and jumper cables	None

Minecraft TNT

You will need:

Navigate the world of Minecraft Pi, learn to create with code and blow things up with TNT.

Hardware	Software setup
None	None

Sense HAT random sparkles

You will need:

Make your Sense HAT shine with pride.

Hardware	Software setup
Sense HAT (or use the emulator)	None

Sense HAT smile

You will need:

Learn to read sensor data from the Sense HAT using Python, and how to use pixel art on the Sense HAT's pixel display.

Hardware	Software setup
Sense HAT	`wget rpf.io/shfaces -O mu_code/faces.py`

Sense HAT cheerlights

You will need:

Tweet a colour to light up your Sense HAT!

Hardware	Software setup

Sense HAT (or use the emulator)	`cd mu_code`
	`wget rpf.io/shcheer -O cheerlights.py`
	`wget rpf.io/shcheerauth -O auth.py`
	
	Add Twitter API keys to `auth.py` - **see rpf.io/shch**

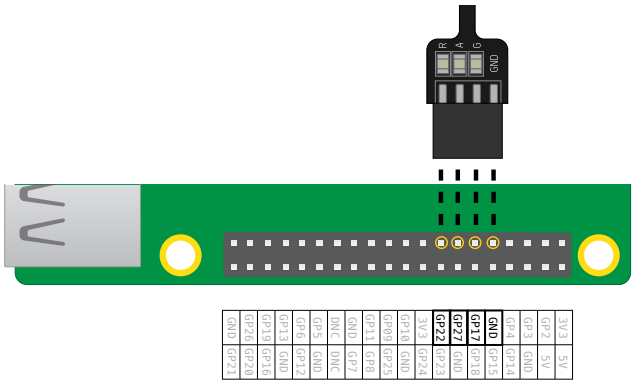
RASPBERRY JAM

Traffic lights with Scratch 1.4

Connect the LEDs

1. Connect your LEDs to the following pins:

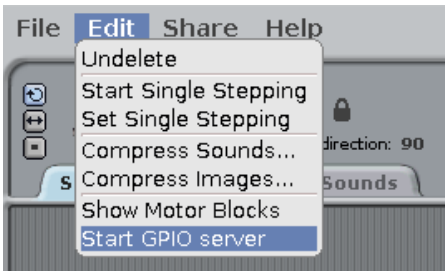
LED	GPIO
Red	22
Amber	27
Green	17



Control the LEDs

1. Open **Scratch** from the Programming menu (**Scratch**, not **Scratch 2**).

2. Click **Edit** in the menu bar and select **Start GPIO server**:



3. Click the **Control** panel. Drag in a **when flag clicked** block and two broadcast blocks. Dock them together in sequence and edit the broadcasts to say **config22out** and **gpio22on** like so:

4. Now click the green flag to run your code. You should see the red LED light up.

5. Now add a **wait 1 secs** block before and after turning the LED off with **broadcast gpio22off**, and wrap it in a **forever** block to blink continuously:

6. Click the green flag again and you should see the LED blink.

7. Now add some more **broadcast** blocks to introduce the other two lights, and make them all flash on and off:

8. Click the green flag again and you should see the three lights flash together.

9. Can you change the number in **wait 1 secs** to speed up or slow down the sequence?

7

Traffic lights sequence

1. Try turning the lights on and off in sequence:

2. Now you know how to control the lights individually, and time the pauses between commands, can you create a traffic light sequence? The sequence goes:

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

It's important to think about timing. How long should the lights stay on for at each stage?

Once you have completed the traffic light sequence, you might want to try adding in a button and a buzzer to make an interactive traffic light for a pedestrian crossing.

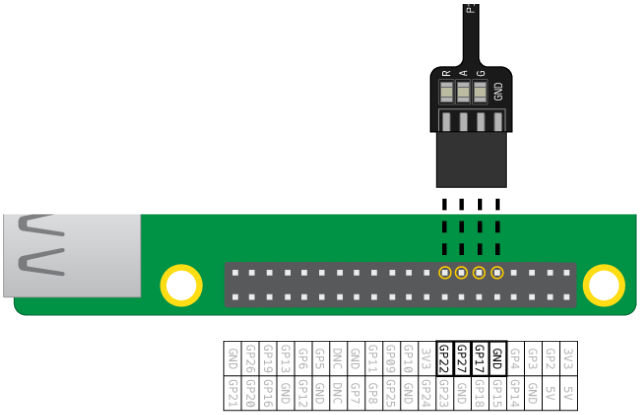
Traffic lights with Scratch 2

RASPBERRY JAM

Connect the LEDs

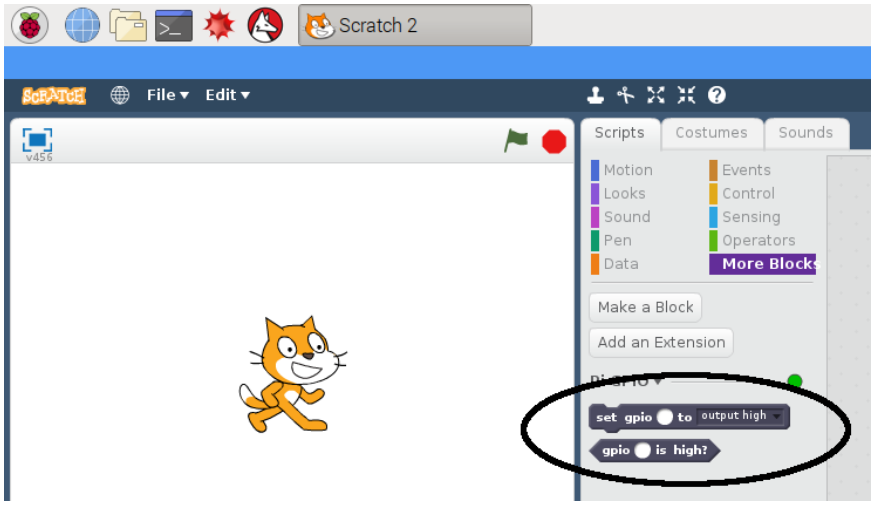
1. Connect your LEDs to the following pins:

LED	GPIO
Red	22
Amber	27
Green	17



Control the LEDs

1. Open **Scratch 2** from the Programming menu (**Scratch 2**, not **Scratch**).
2. Open the **More Blocks** panel, click **Add an Extension**, and select **Pi GPIO**.
3. You should then see two new blocks appear in **More Blocks**:



4. Open the **Events** panel and drag in a `when flag clicked` block.

5. Open the **More Blocks** panel, drag in a `set gpio to output high` block and dock it under the previous block.

   Set the gpio to number 22.

6. Now click the green flag to run your code. You should see the red LED light up.

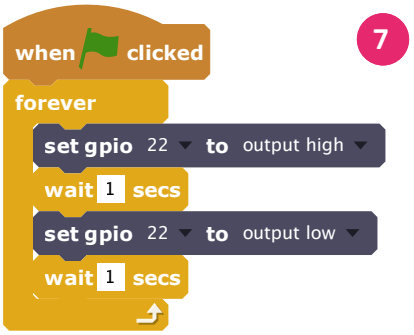
7. Now add a `wait 1 secs` block before and after turning the LED off with `set gpio 22 to output low`, and wrap it in a `forever` block to blink continuously:

8. Click the green flag again and you should see the LED blink.

9. Now add some more `set gpio` blocks to introduce the other two lights on gpio 27 & 17, and make them all flash on and off:

10. Click the green flag again and you should see the three lights flash together.

11. Can you change the number in `wait 1 secs` to speed up or slow down the sequence?



Traffic lights sequence

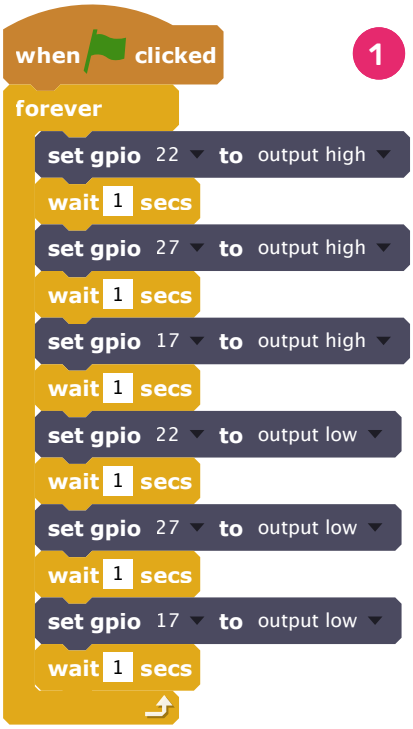
1. Try turning the lights on and off in sequence:

2. Now you know how to control the lights individually, and time the pauses between commands, can you create a traffic lights sequence? The sequence goes:

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

It's important to think about timing. How long should the lights stay on for at each stage?

Once you have completed the traffic light sequence, you might want to try adding in a button and a buzzer to make an interactive traffic light for a pedestrian crossing.

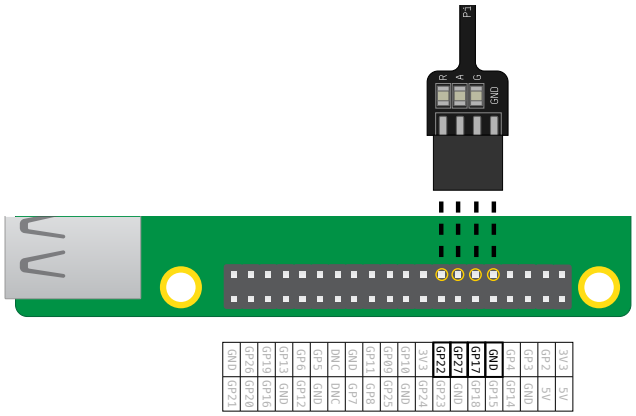


Traffic lights with EduBlocks

Connect the LEDs

1. Connect your LEDs to the following pins:

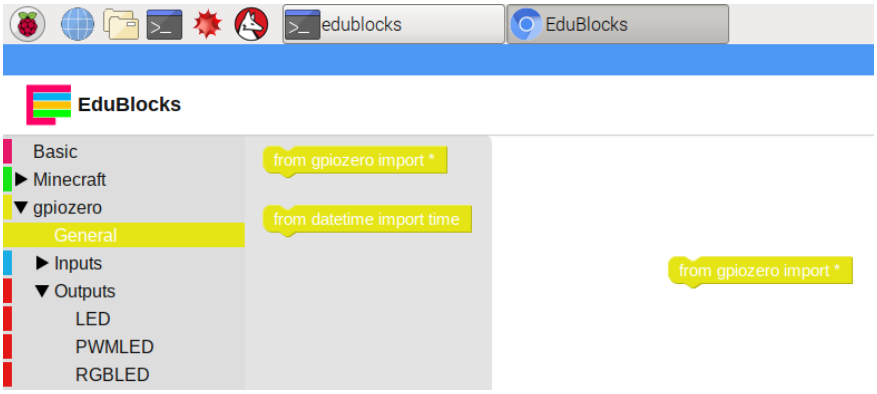
LED	GPIO
Red	22
Amber	27
Green	17



Control the LEDs

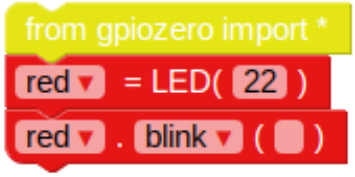
1. Open **EduBlocks** from the Desktop.

2. Click the **gpiozero** drop-down, click **General** and drag the **from gpiozero import *** block into the workspace.



3. Click the **Outputs** drop-down under **gpiozero** and click **LED**. Drag an **led = LED(pin)** block into the workspace beneath the import block. Rename the variable from **led** to **red**, and change **pin** to **22**.

4. Drag in an **led.on** block, and dock it beneath the previous block. Change the **on** drop-down to **blink**.
Your code blocks should now look like this:



5. Now click the **Run** button to run your code. You should see the red LED blink.

6. Now add some more LED blocks to introduce the other two lights, and make them blink at different speeds.

7. Run your code again, and you should see the three lights flashing at different rates.

8. If a larger number makes a light blink slower, what number would make it run faster? Try to make your lights blink faster.

Traffic lights sequence

1 The **on** function allows you to turn a light on. You can use **sleep** to pause between commands. Bring in the **import time** block from the **Basic** section. Try this example to turn the lights on in sequence:

```
from gpiozero import *
import time

red = LED( 22 )
amber = LED( 27 )
green = LED( 17 )

red . on ( )
time.sleep( 1 )
amber . on ( )
time.sleep( 1 )
green . on ( )
time.sleep( 1 )
```

The main controls for LEDs are **on**, **off**, **toggle** and **blink**.

2 Try turning the lights on and off in sequence:

```
from gpiozero import *
import time

red = LED( 22 )
amber = LED( 27 )
green = LED( 17 )

red . on ( )
time.sleep( 1 )
amber . on ( )
time.sleep( 1 )
green . on ( )
time.sleep( 1 )
red . off ( )
time.sleep( 1 )
amber . off ( )
time.sleep( 1 )
green . off ( )
```

3 Try repeating this by putting the code inside a **while** loop:

```
from gpiozero import *
import time

red = LED( 22 )
amber = LED( 27 )
green = LED( 17 )

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

4 Now you know how to control the lights individually, and time the pauses between commands, can you create a traffic light sequence? The sequence goes:

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

It's important to think about timing. How long should the lights stay on for at each stage?

Once you have completed the traffic light sequence, you might want to try adding in a button and a buzzer to make an interactive traffic light for a pedestrian crossing.

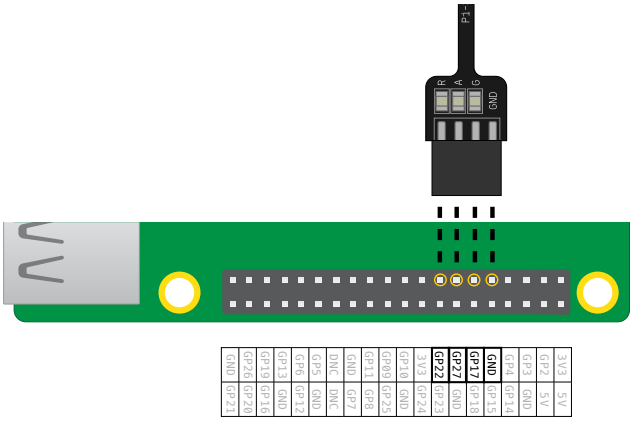
Traffic lights with Python

RASPBERRY JAM

Connect the LEDs

1. Connect your LEDs to the following pins:

LED	GPIO
Red	22
Amber	27
Green	17



Control the LEDs

1. Open **Mu** from the main menu.
2. Enter the following code:

```
from gpiozero import LED

red = LED(22)

red.blink()
```

3. Now save your program and press **F5** to run your code. You should see the red light flash on and off continuously.
4. Now modify your code to introduce the other two lights, and make them blink at different speeds:

```
from gpiozero import LED

red = LED(22)
amber = LED(27)
green = LED(17)

red.blink(1, 1)
amber.blink(2, 2)
green.blink(3, 3)
```

5. Run your code again and you should see the three lights flashing at different rates.
6. If a larger number makes a light blink slower, what number would make it run faster? Try to make your lights blink faster.

Traffic lights sequence

1 The `on` function allows you to turn a light on. You can use `sleep` to pause between commands. Try this example to turn the lights on in sequence:

```
from gpiozero import LED
from time import sleep

red = LED(22)
amber = LED(27)
green = LED(17)

red.on()
sleep(1)
amber.on()
sleep(1)
green.on()
sleep(1)
```

The main controls for LEDs are `on`, `off`, `toggle` and `blink`.

2 Try turning the lights on and off in sequence:

```
red.on()
sleep(1)
amber.on()
sleep(1)
green.on()
sleep(1)
red.off()
sleep(1)
amber.off()
sleep(1)
green.off()
```

3 Try repeating this by putting the code inside a `while` loop:

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

4 Now you know how to control the lights individually, and time the pauses between commands, can you create a traffic lights sequence? The sequence goes:

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

It's important to think about timing. How long should the lights stay on for at each stage?

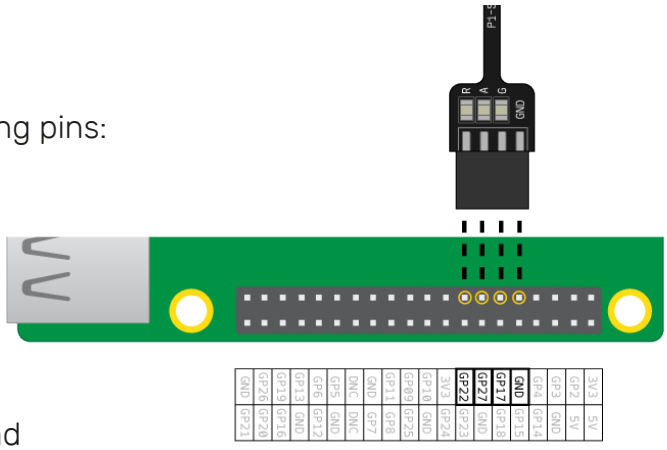
Once you have completed the traffic lights sequence, you might want to try adding in a button and a buzzer to make an interactive version.

Traffic Lights Controller GUI

Flash the LEDs

1. Connect your LEDs to the following pins:

LED	GPIO
Red	22
Amber	27
Green	17



2. Open **Mu** from the main menu and click the **REPL** icon.

3. Enter the following commands, one by one, into the REPL, and observe the LEDs:

```
from gpiozero import TrafficLights
lights = TrafficLights(22, 27, 17)
lights.on()
lights.off()
lights.blink()
```

4. Now blink the LEDs at different speeds — the two numbers in the `()` are **on time** and **off time**, in seconds:

```
lights.blink(2, 2)
lights.blink(5, 5)
lights.blink(0.1, 0.1)
```

5. Try blinking each LED at a different rate:

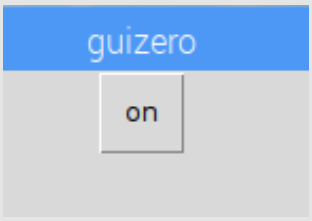
```
lights.red.blink(1, 1)
lights.amber.blink(2, 2)
lights.green.blink(3, 3)
```

Create a GUI

Close the REPL and use the main **Mu** window to type your program into a file.

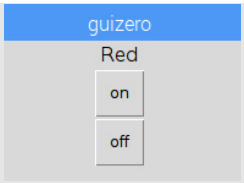
1 Create a GUI button to turn the red LED on:

```
from guizero import App, Text, PushButton
from gpiozero import TrafficLights

lights = TrafficLights(22, 27, 17)

app = App()

PushButton(app, command=lights.red.on, text="on")

app.display()
```



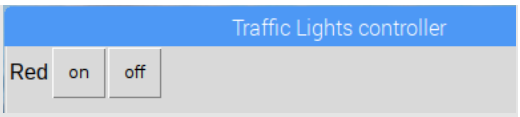
2 Add a text label and a second button to turn the red LED off:

```
Text(app, "Red")
PushButton(app, command=lights.red.on, text="on")
PushButton(app, command=lights.red.off, text="off")
```



3 Now give your GUI app a name, and use the grid layout:

```
app = App("Traffic Lights controller", layout="grid")

Text(app, "Red", grid=[0, 0])
PushButton(app, command=lights.red.on, text="on", grid=[1, 0])
PushButton(app, command=lights.red.off, text="off", grid=[2, 0])
```



Challenges

1 Add on/off buttons for all three LEDs, and make sure the buttons are aligned in the grid.

2 Add buttons to make each LED blink.

3 Add buttons to turn all LEDs on/off at the same time.

4 Write your own function to make the LEDs do the traffic lights sequence.

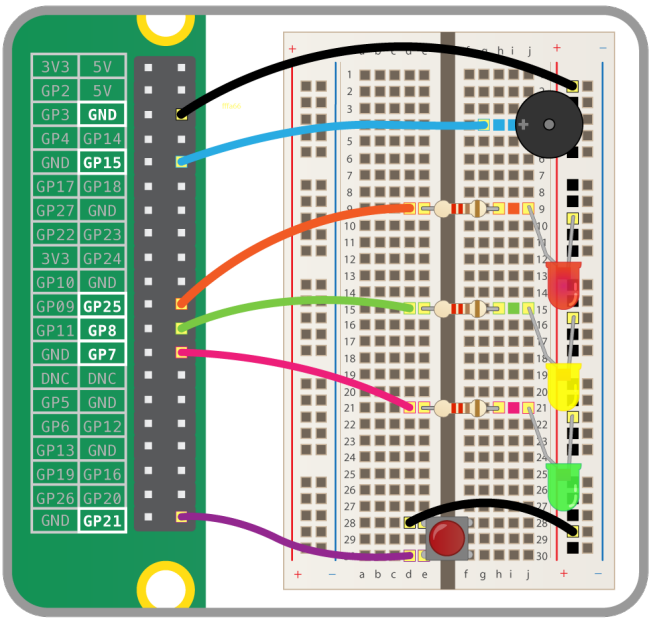
Hint Include `from time import sleep` in your program, use `def sequence()`, and set the command to `sequence`.

Interactive traffic lights with Python

Connect the LEDs

1 Connect your LEDs to the following pins:

LED	GPIO
Button	21
Red LED	25
Amber LED	28
Green LED	27
Buzzer	15



Control the LEDs and button

1 Open **Mu** from the main menu.

2 Enter the following code:

```
from gpiozero import LED, Button

red = LED(25)
button = Button(21)

while True:
    if button.is_pressed:
        red.on()
    else:
        red.off()
```

3 Save your code and run it with **F5**

4 Remove the while loop and add two more LEDs

```
from gpiozero import LED, Button

red = LED(25)
amber = LED(28)
green = LED(27)

button = Button(21)
```

5 Get them to come on when the button is pressed:

6 Run the code and press the button.

```
while True:
    if button.is_pressed:
        green.on()
        amber.on()
        red.on()
    else:
        green.off()
        amber.off()
        red.off()
```

Traffic lights

You can use the built-in `TrafficLights` class instead of three individual LEDs.

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

```
from gpiozero import TrafficLights, Button
from time import sleep

button = Button(21)
lights = TrafficLights(25, 28, 27)

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

2 Try changing the lights to `blink`:

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

2 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.

3 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.

4 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 the lights back to green until the button is pressed again.

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

```
buzzer = Buzzer(15)

buzzer.on()
buzzer.off()
buzzer.beep(0.1, 0.1)
```

6 Your final interactive traffic lights code should start on a green light and then:

- Wait for the button to be pressed
- When pressed, change to red/amber, then green
- Beep for a while to say it's time to cross
- Go to amber and then green
- Repeat

RASPBERRY JAM

Minecraft TNT

RASPBERRY JAM

Enter the Minecraft world

1. Open Minecraft Pi from the main menu. Start a game and create a new world.
2. Walk around using the WSAD keys on the keyboard. Use Space to jump, and double tap Space to fly.

3. Press **Tab** on the keyboard to release your mouse cursor, and open **Mu** from the main menu.
4. Move your windows around so Minecraft and Mu are side by side.

Controlling Minecraft with Python

1. Enter the following code into Mu:

```
from mcpi.minecraft import Minecraft

mc = Minecraft.create()

mc.postToChat("Hello world")
```

2. Run the code with **F5** and you should see the message "Hello world" appear in the Minecraft window.
3. Add the following lines to your code:

```
x, y, z = mc.player.getPos()
mc.setBlock(x+1, y, z, 1)
```

4. Run the code and you should see a block of stone appear near your player. If it's not in front of you, try looking around.

5. Change the value in the `setBlock` line from `1` to `2`:

```
mc.setBlock(x+1, y, z, 2)
```

6. You should now see a block of grass appear. Try changing the number again and see which block gets placed.

7. Try changing `setBlock` to `setBlocks` to build a 10x10x10 cube rather than a single block:

```
mc.setBlock(x+1, y, z, 2)
```

You should see a large solid cube of stone appear!

TNT

The block ID for TNT is `46`. There are two types of TNT: unexplosive TNT and explosive TNT. You want explosive TNT.

1. Build a solid cube of TNT. To get explosive TNT, you need to add a `1` to the end of your `setBlocks` line:

```
mc.setBlocks(x+1, y+1, z+1, x+11, y+11, z+11, 46, 1)
```

2. Go up to the cube of TNT and right-click to hit it with your sword. This will activate the TNT. Stand back and watch the show!

RASPBERRY JAM

Sense HAT random sparkles

RASPBERRY JAM

For this activity, you can either use the physical Sense HAT hardware, the desktop emulator in Raspbian, or the web-based emulator on **trinket.io**

If you're using the Sense HAT, attach it to your Raspberry Pi before booting.
If you're using the Trinket emulator, open a web browser and go to **trinket.io/sense-hat**

Using set_pixel

First, we'll think up some random numbers and use the `set_pixel` function to place a random colour on a random location on the Sense HAT display.

1. If you're using a Raspberry Pi, open **Mu**. If you're using the web emulator, delete the example code before you begin.
2. In the new file, start by importing the Sense HAT module.

If you're using a physical Sense HAT or the Trinket emulator, the import line is:

```
from sense_hat import SenseHat
```

or

If you're using the desktop emulator, the import line is:

```
from sense_emu import SenseHat
```

The rest of the code will be identical for all versions.

3. Next, create a connection to your Sense HAT by adding:

```
sense = SenseHat()
```

4. Now think of a random number between 0 and 7 and assign it to the variable `x`, for example:

```
x = 4
```

5. Think of another random number between 0 and 7, then assign it to `y`:

```
y = 5
```

6. Think of three random numbers between 0 and 255, then assign them to `r`, `g`, and `b`:

```
r = 19
g = 180
b = 230
```

7. Now use the `set_pixel` function to place your random colour at your random location on the display:

```
sense.set_pixel(x, y, r, g, b)
```

8. Now press the **Run** button to run your code. You should see a single pixel light up.

9. Now pick a new set of random numbers, change them all, and run the program again. A second pixel should appear on the display!

Using the random module

So far you have picked your own random numbers, but you can let the computer choose them instead.

1. Add another `import` line at the top of your program, below `import SenseHat`:

```
from random import randint
```

2. Now change your `x =` and `y =` lines to automatically select a random position:

```
x = randint(0, 7)
y = randint(0, 7)
```

3. Run your program again, and you should see another random pixel being placed on the display. It will be the same colour as the last pixel you placed.

4. Now change your colour value lines to:

```
r = randint(0, 255)
g = randint(0, 255)
b = randint(0, 255)
```

Now your program will automatically select a random colour.

5. Run it again, and you should see another pixel appear in a random location with a random colour.

Run it a few more times, and you should see more of the grid fill up with random pixels.

Add a loop

Rather than having to keep running your program, you can add a loop so that it will keep going.

1. First, add an `import` to the top of your file:

```
from time import sleep
```

You'll use this to pause the program between pixels.

2. Add a `while True:` to your code so that the random lines, `set_pixel` and `sleep` are all within the loop:

```
while True:
    x = randint(0, 7)
    y = randint(0, 7)
    r = randint(0, 255)
    g = randint(0, 255)
    b = randint(0, 255)
    sense.set_pixel(x, y, r, g, b)
    sleep(0.1)
```

3. Run the code and you should see random sparkles in action!

4. Try changing the sleep time to make it even shorter.

Sense HAT smile

RASPBERRY JAM

Test the Sense HAT

1. Open **Mu** and open the **REPL**. Type the following commands directly into the **REPL**:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.show_message("Hello world")
```

Press Enter after each line, and after the third line, the message should appear on the Sense HAT's display.

2. Now try retreiving the sensor values:

```
sense.temperature
sense.humidity
sense.pressure
sense.accelerometer
sense.gyroscope
sense.orientation
```

When you press Enter, you will see the sensor's value.

Faces

1. Now close the REPL and use the main window to type the following program into a file:

```
from sense_hat import SenseHat
from faces import normal, happy, sad
from time import sleep

sense = SenseHat()

sense.set_pixels(sad)
sleep(1)
sense.set_pixels(normal)
sleep(1)
sense.set_pixels(happy)
```

2. Run the code with **F5** and you should see a sad face, a normal face, and a happy face appear.

And breathe...

1. Replace the last 5 lines with:

```
start_humidity = sense.humidity

while True:
    print(sense.humidity)
    if sense.humidity > start_humidity + 10:
        sense.set_pixels(happy)
    elif sense.humidity > start_humidity + 5:
        sense.set_pixels(normal)
    else:
        sense.set_pixels(sad)
    sleep(1)
```

2. Run the code again. Now breathe on the Sense HAT and see if you can make it smile!

Sense HAT cheerlights

RASPBERRY JAM

1. Open **Mu** and click the **REPL** icon to open up a Python shell.

2. Import the `colorzero` library by typing:

```
from colorzero import Color
```

3. Create a colour object with the word 'red':

```
c = Color('red')
```

4. Inspect the different representations of the colour by typing each of these in turn:

```
c.rgb
c.rgb_bytes
c.html
```

5. You should see the colour red represented in different ways. Try the same with a different colour name.

6. The Sense HAT library expects RGB values from 0 to 255. Try setting the LEDs to different colours using:

```
from sense_hat import SenseHat
from colorzero import Color

sense = SenseHat()

color = Color('red')

sense.clear(color.rgb_bytes)
```

Cheerlights

1. Click the **Load** button and open the `cheerlights.py` file.

2. The starter code simply prints out the tweet contents. Press the **Run** button and get someone to tweet `#cheerlights red` — you should see the word 'red' in the output.

3. Modify the `on_success` method in the `CheerlightsStreamer` class to set the Sense HAT LEDs to the tweeted colour:

```
def on_success(self, data):
    if 'text' in data:
        tweet = data['text'].replace(hashtag, '')
        color_text = tweet.strip()
        color = Color(color_text)
        sense.clear(color.rgb_bytes)
```

4. Try tweeting different colour names to test it out!

5. You might notice that some colour names you try don't work — and they can cause the program to crash. Add an exception handler to deal with this, and to let you know when there's an error:

```
def on_success(self, data):
    if 'text' in data:
        tweet = data['text']
        tweet = tweet.replace(hashtag, '')
        color_text = tweet.strip()
        try:
            color = Color(color_text)
            sense.clear(color.rgb_bytes)
        except ValueError:
            print('Failed: {}'.format(color_text))
```

Challenges

1. Can you transition from one colour to the next rather than instantly changing it?

2. Can you light up one pixel at a time (in order or at random) by using `set_pixels` instead of `clear`?

3. Can you add support for more colour representations like RGB or hex?

4. Can you add support for keywords like 'rainbow' to perform a cycle of colours rather than a single colour?