nustem

Introduction to Digital Making

There's a tradition in programming that the first code you write in any new language outputs 'Hello, world!'

In digital making, the equivalent is flashing an LED. So let's do that.

Ingredients

Platform:	Raspberry Pi / Raspbian Python3
Components:	Red, yellow, green LEDs Resistors Breadboard Jumper wires (male/female)

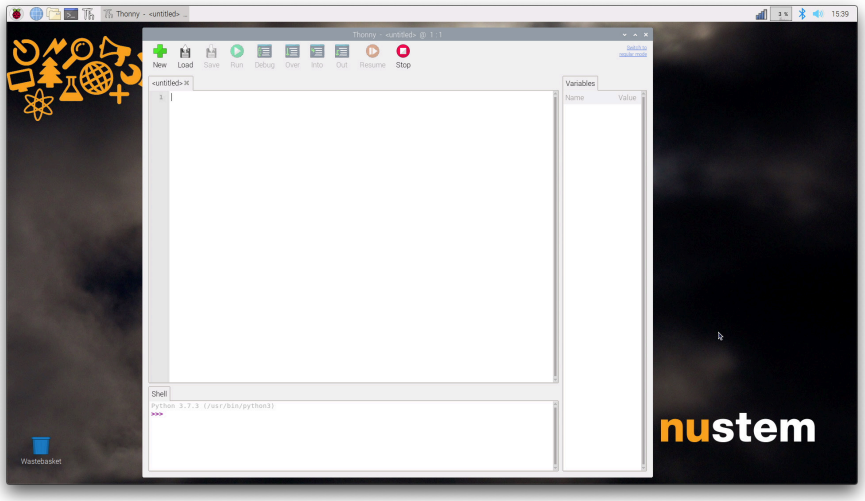
Glossary

GPIO	"General Purpose Input/Output" – the little metal pins on your Pi, used to connect electronic components.
Pin	The little metal doohickie on the Pi, into which you can plug a wire.
LED	Light Emitting Diode A component emits light when fed 5 V... or emits smoke when fed -5 V.
Resistor	A component which discourages your LED from emitting smoke.
Breadboard	A slab of white plastic which has little to do with bread. But it's good for connecting wires.

Blink

Start at the Raspberry Pi desktop. Most things work as you'd expect from Windows or a Mac, and you can mostly ignore the differences.

Click the button for **Thonny** in the top bar, and you'll see this:



This is a text editor, for the **Python** language. The cursor should be blinking in the big white area of the window, so type:

```
from gpiozero import LED
from time import sleep
```

Thonny will change the colours as you type, which gives you a hint that you're spelling things correctly.

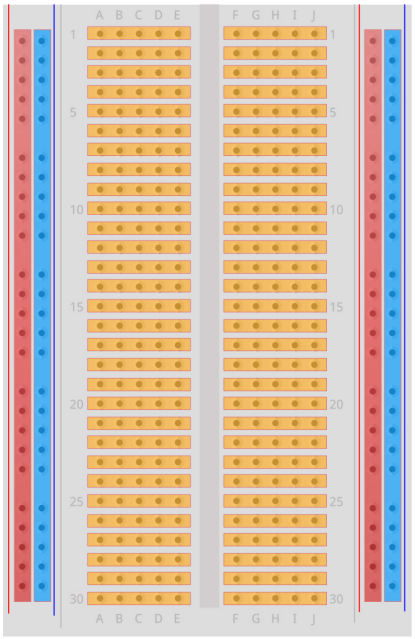
Python understands a few simple commands all on its own, but for this project we're going to need to extend its vocabulary. So we send it to the library to pull a chapter called '`LED`' from a book called '`gpiozero`.' Python then knows about LEDs, the little light-up components we've given you. It also now knows about pausing (`sleep`ing) for a period of seconds.

Northumbria University NEWCASTLE

However, it isn't yet doing anything with this new knowledge, so add the following:

```
red = LED(17)
red.on()
```

These lines instruct Python to make a new `LED` object called `red`, which will be attached to pin number 17 (we'll get to that in a moment). That new LED is then commanded to turn on. Two things need to be sorted before you see anything light up: you need to save and run your program, and you need to wire up the LED.



A breadboard, yesterday

Breadboard

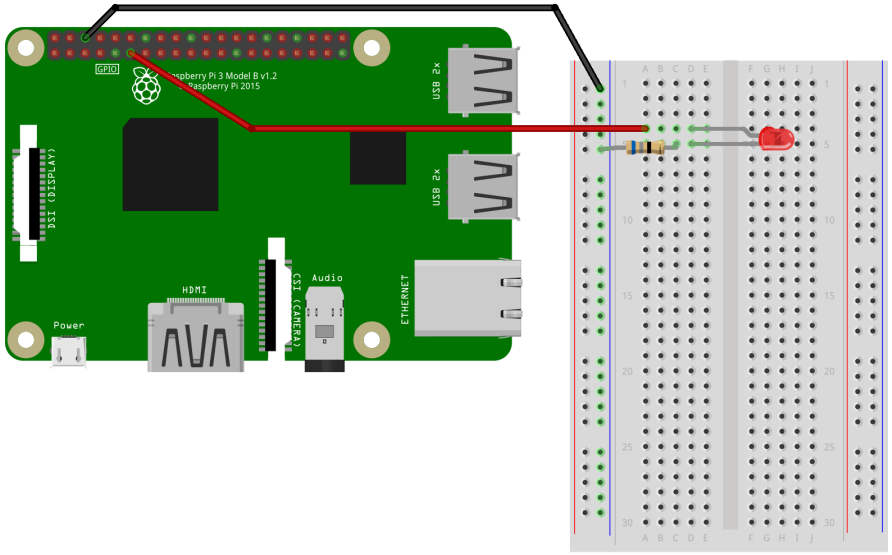
The slab of white plastic with all the holes is called **breadboard**. The holes are connected to each other as indicated in the diagram (left).

In our fancy pi-top desktop computers the Pi's pins are all hidden... but they're duplicated on the pi-topPROTO breakout board. You may have a little cardboard doohickie to labels the pins, which helps... if your eyesight is good. Alternatively, type `pinout` at a Terminal prompt for a handy guide. Which is conveniently turned sideways.

Wiring the first LED

Wire up a ground pin on the Pi (third in from the left, top row, labelled `GND`) to the blue-striped breadboard column. It doesn't matter what colour of wire you use.

Now add the red LED. Note that it has a long leg and a short leg, and it only works one way around. On the diagram, there's a kink in the long leg. You'll need a resistor to complete the circuit back to ground.



Run your code!

Click the green arrow '`Run`' button on Thonny's toolbar. In a slight anticlimax, you'll have to save the script first. It's OK to save it in your `Documents` folder: call it something like `lights.py`.

You should see your LED light up. `Woohoo!`

If instead you see error messages in Thonny's bottom pane, ask one of us for help.

Blink

What's more exciting than lighting up an LED? Turning it off again. Obviously.

Change your code so it now reads:

```
from gpiozero import LED
from time import sleep

red = LED(17)
red.on()
sleep(0.5)
red.off()
```

Run that, and you should see the LED... turn off, because it was still on from the last run. Run it again, and it'll turn on for half a second before going out. Spot where the half second comes from.

Now we need that sequence to loop around. Change your code so it reads:

```
from gpiozero import LED
from time import sleep

red = LED(17)

while True:
    red.on()
    sleep(0.5)
    red.off()
    sleep(0.5)
```

Python is a little unusual, in that white space is significant. That is, you have to indent these lines to tell Python that they're part of the `while` *loop. The convention is to use four spaces, but in Thonny hitting Tab will type those for you too.*

Run that, and you should have a blinking LED.

Questions: Why do you need the second sleep(0.5) command? And can you fathom how the **`while`** loop works?

Traffic Lights!

At last! You now know everything you need to make a full set of red/amber/green traffic lights. You'll need to wire up two more LEDs, and add some lines to your program to make them work.

Hints:

- The next two pins on the Raspberry Pi along from the one you've used for the red LED are numbered 27 and 22, respectively. No, it doesn't make any sense.
- You'll need to tell Python about a new LED before you can use it – as in the line `red = LED(17)`.
- You might get bored typing `sleep(0.5)`. You could instead type something like `sleep(delay)` instead, and tell Python that `delay = 0.5`. Yes, we know this is actually *more* typing, but it's also a really big idea.

All done?

GPIOzero is a fantastic library which makes attaching a range of components to a Raspberry Pi very simple. Have a read of the documentation and see how much makes sense: `gpiozero.readthedocs.io`

Try this – functions

Very often, you'll want to wrap up bits of code so you can refer to them from somewhere else. Try this:

```
def sequence():
    red.on()
    sleep(1.0)
    ...
    red.on()

sequence()
```

the rest of your loop code goes in here

Wrap your light sequence in a *function definition*, then *call* it from lower down in your code. Oooh!

Try this – Button control

The GPIOZero library can handle buttons for you. You'll find instructions here:

`https://gpiozero.readthedocs.io/en/stable/recipes.html#button`

See if you can get a button to trigger the light sequence.

Hints:

- Wrapping your sequence in a function call, as above, is a good start.
- The documentation shows you several different ways of detecting a button press. Is one more appropriate than others in this case?