nustem

Analogue Input

Attach and measure an analogue device, then connect it to other stuff.

Ingredients

Platform:	Raspberry Pi / Raspbian Python3
Components:	MCP3007 ADC Servo Neopixel
Libraries:	gpiozero Neopixel

Glossary

ADC	"Analogue Digital Converter" – a circuit which measures an analogue signal and outputs a digital representation of it.

Digital & Analogue inputs

Buttons are digital input devices; they're either pressed or not, on or off. That's fine, but often you want to *measure* a value; for example: how hard is the button being pressed? With a conventional button and GPIOzero's standard input devices, that information is lost.

To capture it, you need an analogue input – one that returns a precise measurement. Devices like Arduinos and Micro:Bits can measure between 0.0 and 3.3 Volts (or 5.0 Volts, sometimes). Raspberry Pis... can't. However, GPIOzero *does* know about add-on chips which can do the job.

The most common such chip is the MCP3008, which has eight analogue input pins. We've done the wiring for you, on the large breadboard.

pot.py

All the code for this worksheet is in `Digital-Making/examples/pot-input/`. Start by loading `pot.py` in Thonny.

Run the code, and it should spit out the potentiometer reading every quarter of a second. The MCP3008 measures a voltage between 0 and 3.3 Volts... but for arcane reasons it outputs an integer between 0 and 1023. GPIOzero then converts that to a floating point number between 0.0 and 1.0. Just roll with it.

The output is a bit messy; if you want to tidy it up a little, format the printed value by editing the line like this:

```
print("{0:1.3f}".format(pot.value))
```

Hmm. Python string formatting is... weird. The `0:1.3f` takes the first passed value (in this case, `pot.value`) and displays it as a single digit, then three digits after the point. Whatever.

potservo.py

Load this example into Thonny. It asks GPIOzero to set up a servo for us to control, and also defines a `map` function. This scales a value from an input range to the equivalent value in an output range. We need to do this because the potentiometer measurement is returning 0.0 to 1.0, but the new servo can move between -1.0 and +1.0.

`map` function: See the Envirocorn worksheet for more information.

Northumbria University NEWCASTLE

Work through the code so you're reasonably comfortable with it, then run it.

The servo should move when you twiddle the potentiometer. It *might* be a bit jerky: Raspberry Pis can be surprisingly temperamental with servo control. This is mostly because servos are arcane bits of vintage analogue technology. The usual fix is to swap the servos around until you find one that actually works with a Pi, and/or to swap power supplies. Arduinos and Micro:Bits are more forgiving with cheap and nasty servos.

pixel.py

Time to make some pretty colours. Load the `pixel.py` script in Thonny, but note that you won't be able to run it from there.

NeoPixels are a brand name for ws2812b LEDs, which can do full red/green/blue colour mixing. You can chain them together, and they're very pretty.

The `pixel.py` script sets up a single NeoPixel attached to the Pi, and runs through a couple of tests; it lights red, then green, then blue, and finally uses another library (`colorwheel.py`) to rotate the colour smoothly around a hue wheel (see diagram, left).

To run the code, you'll need to open a Terminal window and type the following lines:

```
cd ~/Digital-Making/examples/pot-input
sudo python3 pixel.py
```

With luck the NeoPixel will burst into life. To run the code again, you'll need just the `sudo` line. Just like servos, NeoPixels can be a bit fiddly on Pis; running the code like this is a bit of a work-around.

Adafruit's guide is good:

`https://learn.adafruit.com/adafruit-neopixel-uberguide/the-magic-of-neopixels`

Exercise: Pulling it all together

Back in Thonny, load the file `potservopixel.py`. This should look familiar: it's the potentiometer/servo code, we've just added the setup bits for the NeoPixel. Referring back to the pixel.py example, can you work out what to add so the pixel colour is set as you rotate the potentiometer?

There are a couple of hints in the code itself.

Remember that you'll need to run this program from the terminal:

```
sudo python3 potservopixel.py
```

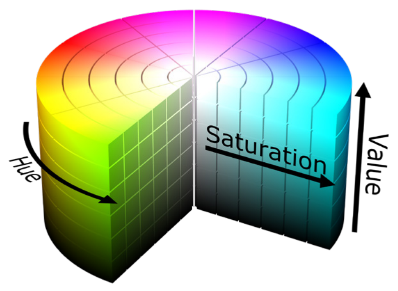
Aside: colour spaces

We're usually taught that colours can be represented as combinations of the primaries red, green and blue. In computing, that tends to be a set of R, G and B values, each between 0 and 255, giving 255^3 = 16million colours.

Of course, we only favour RGB space because that's more-or-less what our eyes' cone cells see. There are many other ways of representing colour, some of which make more sense, and it's possible to convert between them all.

Hue/Saturation/Lightness ('HSL') is another space, where 'hue' is treated as an angle around a colour wheel, 'saturation' is how much of that hue is added, and lightness is how bright the resulting colour is.

In practice, we more often use HSV ('value') space, which is slightly different, but whatever: the point is that 'hue angle' is very useful for generating pretty rainbows.



Seriously, the point of this aside was 'pretty rainbows.'

The `colorwheel.py` module take a hue angle between 0 and 255, and returns RGB values.