EXPLORER HAT WORKSHOP

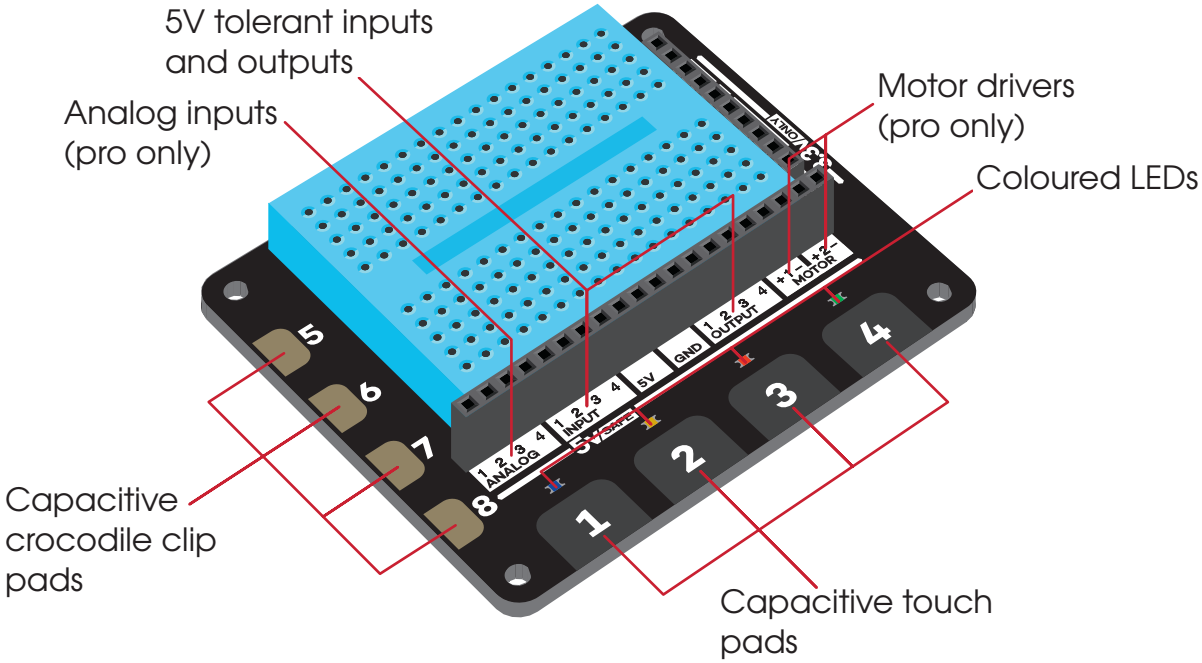
This worksheet will guide you through making a game with a motorised wheel which will rotate for a random amount of time, before stopping to select a question / person from a wheel.

INTRODUCTION

The Explorer HAT Pro is an add-on board from the lovely people at Pimoroni. The board enables anyone to easily learn physical computing using Python. Physical computing is an exciting and innovative area of technology which can provide children with a massive incentive to learn computing.

The board costs £18 (for the pro version; the regular is £10) and has a number of input and output options including capacitive touch pads, coloured LEDs, and motor drivers which we'll be using in this worksheet.

One of the lovely people from Pimoroni



WHAT IS A HAT?

HAT simply stands for "Hardware Attached on Top". It is a specification developed by the Pi Foundation to make development and use of add-on boards for the Raspberry Pi easier. HATs are designed to identify themselves to the Pi automatically, configuring the GPIO pins appropriately for their features.

Other available HATs:

- GPS HAT - for position/time data
- DC & Stepper motor HAT - control all your motors
- PiTFT HAT - add a mini-TFT display to your Pi
- Unicorn HAT - a multi-coloured LED grid
- Astro Pi HAT - sensors for space exploration!

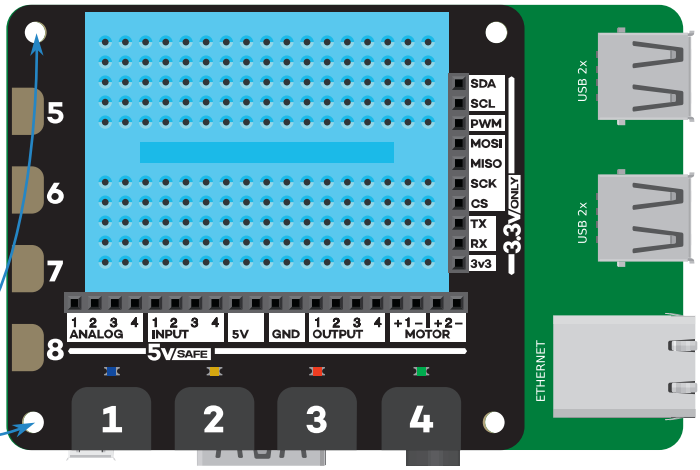
EXPLORER HAT WORKSHOP

INSTALLATION

First, shut down your Raspberry Pi; installing add-on boards like the Explorer HAT Pro can damage the Pi if performed while the Pi has power. Firmly attach your Explorer HAT Pro to the GPIO pins on the edge of the board, ensuring that the HAT covers the Pi.

HINT

Try and ensure that the mounting holes on the Pi and the Explorer HAT line up when viewed from directly above. If they don't, the chances are that the pins are mis-aligned.



EXPLORE!

Start Python by selecting Menu > Programming > Python 3. When Python appears, type `import explorerhat` in the console. You should see "Explorer HAT Pro detected!" appear. If you do not, the most likely cause is that the Explorer HAT is not seated correctly on the Pi.

With the library loaded, you can start to play with the HAT. Try the following commands to control the red LED:

```
>>> explorerhat.light.red.on()
>>> explorerhat.light.red.off()
>>> explorerhat.light.red.toggle()
>>> explorerhat.light.red.blink(0.5, 0.2)
```

The two parameters to the `blink` function give the "on" time and "off" time respectively. There's another function called `pulse` which takes four parameters: fade-in time, fade-out time, on time, and off time. Try the following:

```
>>> explorerhat.light.blue.pulse(0.2, 0.2, 0.5, 0.2)
```

You can also control all the lights at once by leaving out the colour:

```
>>> explorerhat.light.on()
>>> explorerhat.light.pulse(0.2, 0.2, 0.4, 0.3)
```

EXTRAS

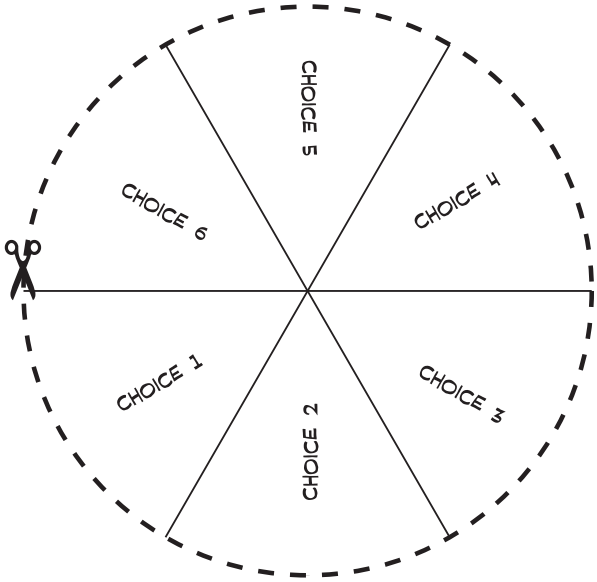
Can you get the four lights pulsing in sequence? Hint: you can write a script to do this using `time.sleep` to pause between commands.

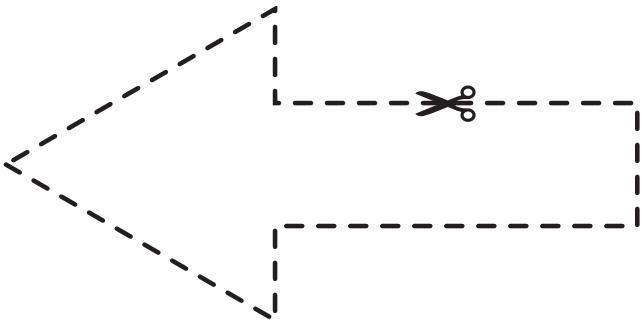
EXPLORER HAT WORKSHOP

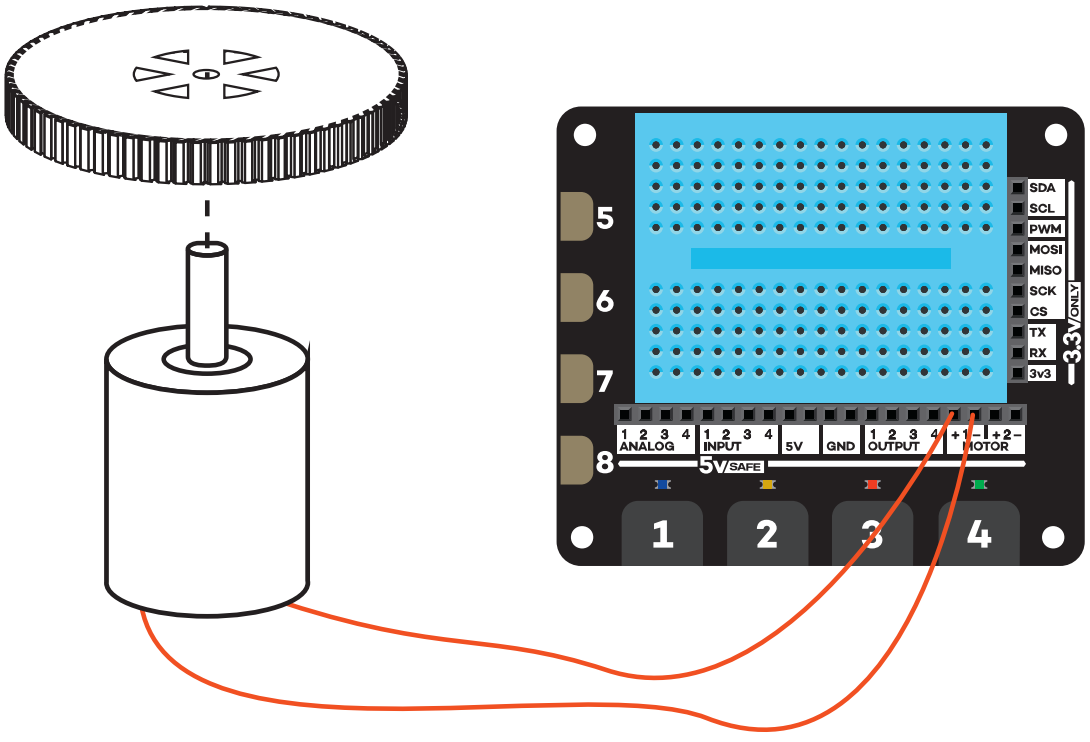
BUILD!

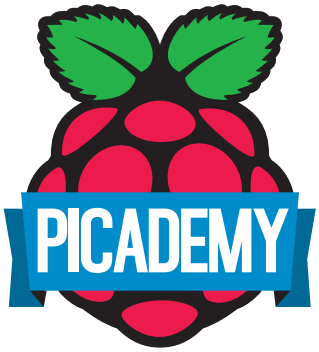
Now that you've got your Explorer HAT working it's time to build something with it. You're going to make a game that randomly chooses a selection from a wheel.

1. Cut out your selection wheel and selector arrow from some coloured card
2. Affix the wheel to the motor, and secure the motor to the desk using blu-tack.
3. Connect your motor to the Motor 1 pins on the Explorer HAT Pro









EXPLORER HAT WORKSHOP

CODE!

Let's get coding! Use File > New Window to create a new program and save it as `wheel_of_fortune.py`. Type the following code into your program:

```
import explorerhat
from time import sleep
from random import randint
```

As before we've imported the `explorerhat` module. We've also imported some functions from other modules that we're going to need: `sleep` (which waits for the specified number of seconds) and `randint` (which returns a random integer number within the specified range).

Now we need to define our own function! This function will be called whenever the first capacitive-touch button on the Explorer HAT Pro is pressed. The function takes two arguments: `channel` and `event`. These are passed to any function which responds to these buttons, and indicate which button is involved and what happened (whether the button was pressed or released). We're not going to use them here though.

```
def wheel(channel, event):
    duration = randint(1, 10)
    print(duration)
    explorerhat.motor.one.forward(100)
    sleep(duration)
    explorerhat.motor.one.stop()
```

Within the function we've created a variable called duration which contains a random integer. We print the duration for the purpose of debugging (this is a very common and useful trick!).Then we turn on the power to motor 1 on the Explorer HAT Pro. Next, we wait for `duration` seconds to pass (while the motor is running), and finally stop the motor.

Finally, we need to tell the Explorer HAT Pro to call our function whenever capacitive-touch button one is pressed. This requires one final line of code (this is not part of our function so make sure it's not intended):

```
explorerhat.touch.one.pressed(wheel)
```

Now run your code, and touch button one on the Explorer HAT Pro. If everything is working correctly, you should see a number printed on the Python console, and the wheel should turn for that number of seconds!

EXPLORER HAT WORKSHOP

TROUBLESHOOTING

Firstly, check that your code matches the complete listing below, including indentation, and capitalisation.

```
import explorerhat
from time import sleep
from random import randint

def wheel(channel, event):
    duration = randint(1, 10)
    print(duration)
    explorerhat.motor.one.forward(100)
    sleep(duration)
    explorerhat.motor.one.stop()

explorerhat.touch.one.pressed(wheel)
```

The next thing to check is the wiring. Ensure that the motor is connected to the +/- pins of motor one. Note that you cannot mix and match motor pins (i.e. if the motor is connected to the +pin of motor one, and the -pin of motor two, it won't work).

EXTENSIONS

You can use `backward` instead of `forward` to make the motor run in the opposite direction. You can also use numbers between 0 and 100 to indicate the speed the motor should turn at (these simply represent a percentage of the motor's full speed).

- Can you flash the LEDs while the motor is turning?
- Can you time the rotation of the motor to land the wheel on specific selections?
- Can you set one button to start the motor running and another to stop it?

WHY DO I NEED A MOTOR DRIVER?

The Pi's GPIO ports can only supply a few mA of current (16mA max). Attempting to draw more than this **will damage the Pi!** Motors typically require at least 400mA to start spinning (although they draw far less after startup). Thus a motor driver chip permits control of a high-current supply (like the Pi's 5V rail, or an external supply) from a low-current control signal (like a Pi's GPIO ports).

Motor drivers are often H-bridge circuits (shown in the diagram to the right), capable of driving a motor forwards or backwards (as well braking or free-running motors in certain cases).

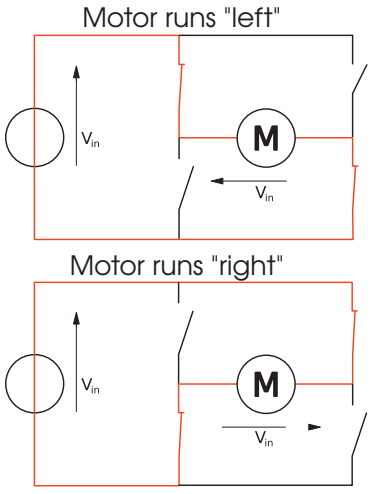


Image © 2006 Cyril Buttay via Wikipedia (CC-BY SA 3.0)