

# Introduction to Electronics

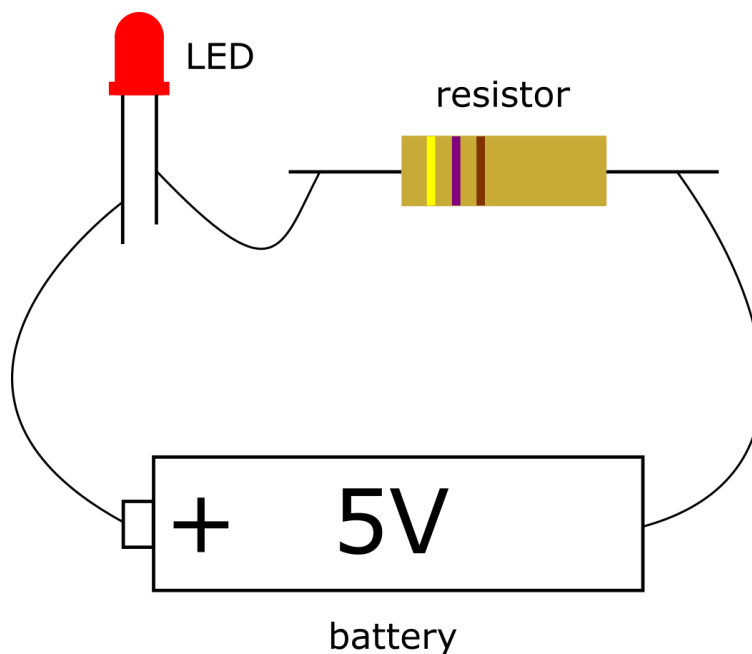
Using a Raspberry Pi3, Explorer HAT Pro and Edublocks/Python

(Worksheet designed and written by Paul Fretwell)

In this worksheet we are going to discover how to build a simple electronic circuit to light an LED. We will use code on the Raspberry Pi computer to control when the LED lights up.

This worksheet will introduce the following components:

- LED: Light emitting diode
- Resistor
- Jumper Lead



This is the circuit we are going to build. But instead of a battery, we are going to use the Explorer HAT Pro connected to a Raspberry Pi computer to supply the power. Then we are going to write some code to control when the power flows around the circuit, turning the LED on and off.

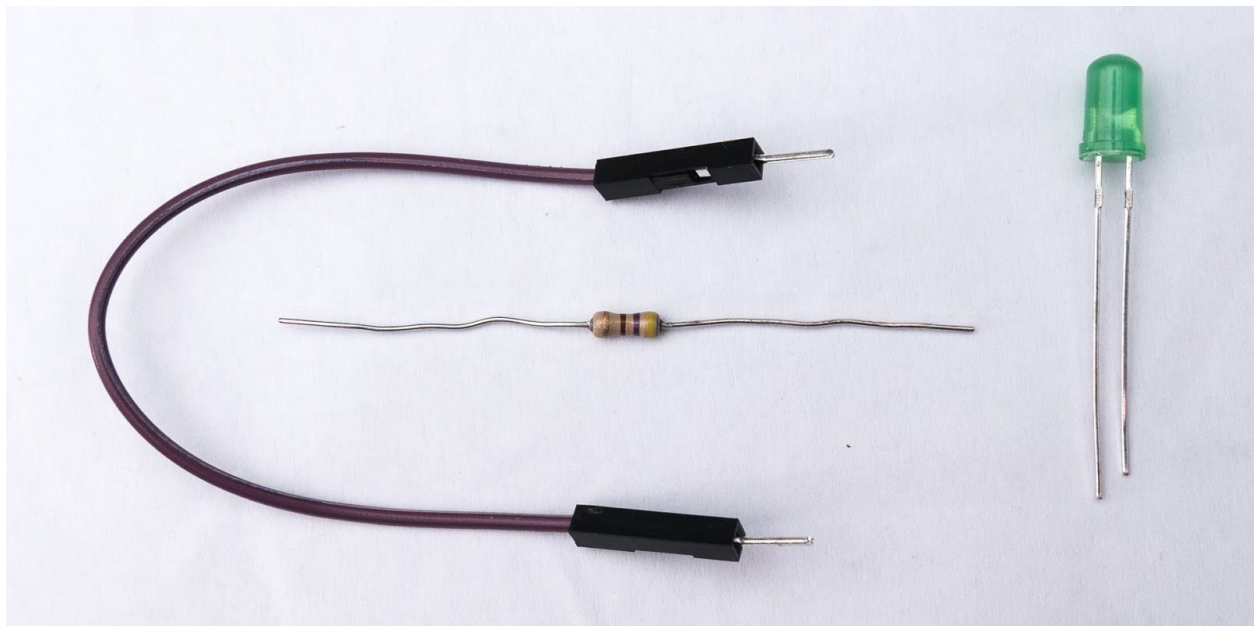
## Lighting an LED

An LED is a type of electronic component called a diode. A diode only allows current to pass through it in one direction. Current is the flow of electricity around the circuit. When current is passing through the LED it will light up. The LED must be connected the correct way round in the circuit, or it will block the flow of electricity and will not light up.

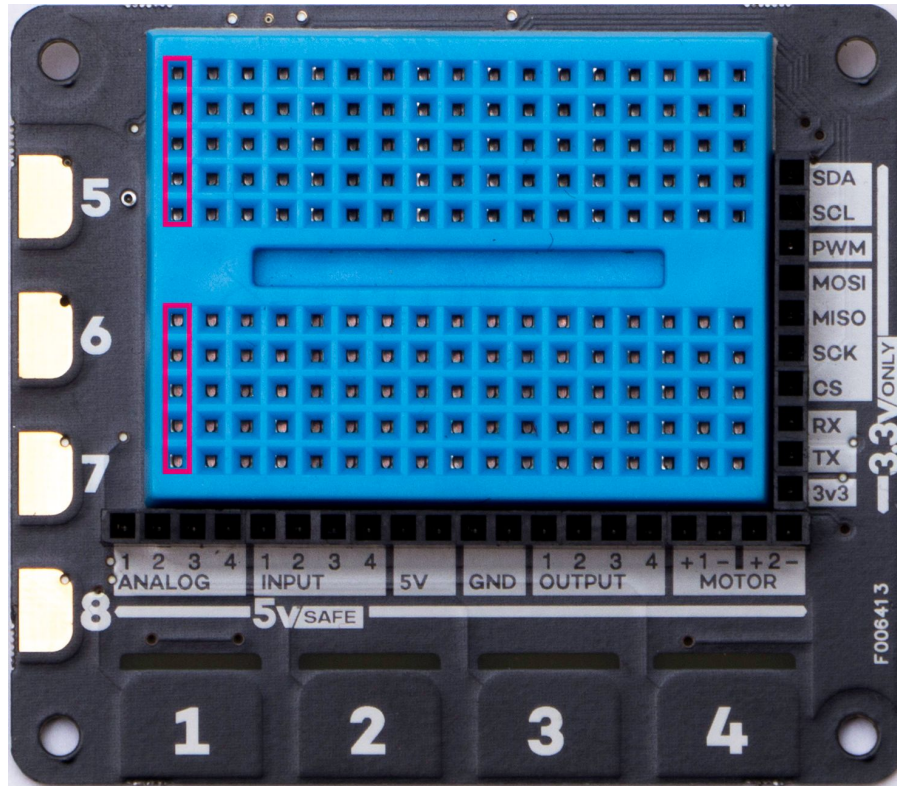
We are going to power our circuit using a 5 volt power supply from the Raspberry Pi. Voltage is the measure of how much pressure is being used to push the electricity around the circuit, and current is a measure of how much electricity is flowing. The higher the voltage, the more energy will flow through the LED. More energy will make the LED glow brighter, but too much will damage the LED and it will no longer work. We can reduce the current in our circuit to protect the LED using a component called a resistor.

A resistor can be connected either way round in the circuit and it works the same way. The higher the resistance value of a resistor, the more it will limit the current in the circuit. To make our LED glow brightly without damaging it, we will use a resistor with a value of 470 Ohms ( $\Omega$ ). You can safely use any resistor over about 470 $\Omega$  for this project.

These are the components we need to build our circuit:

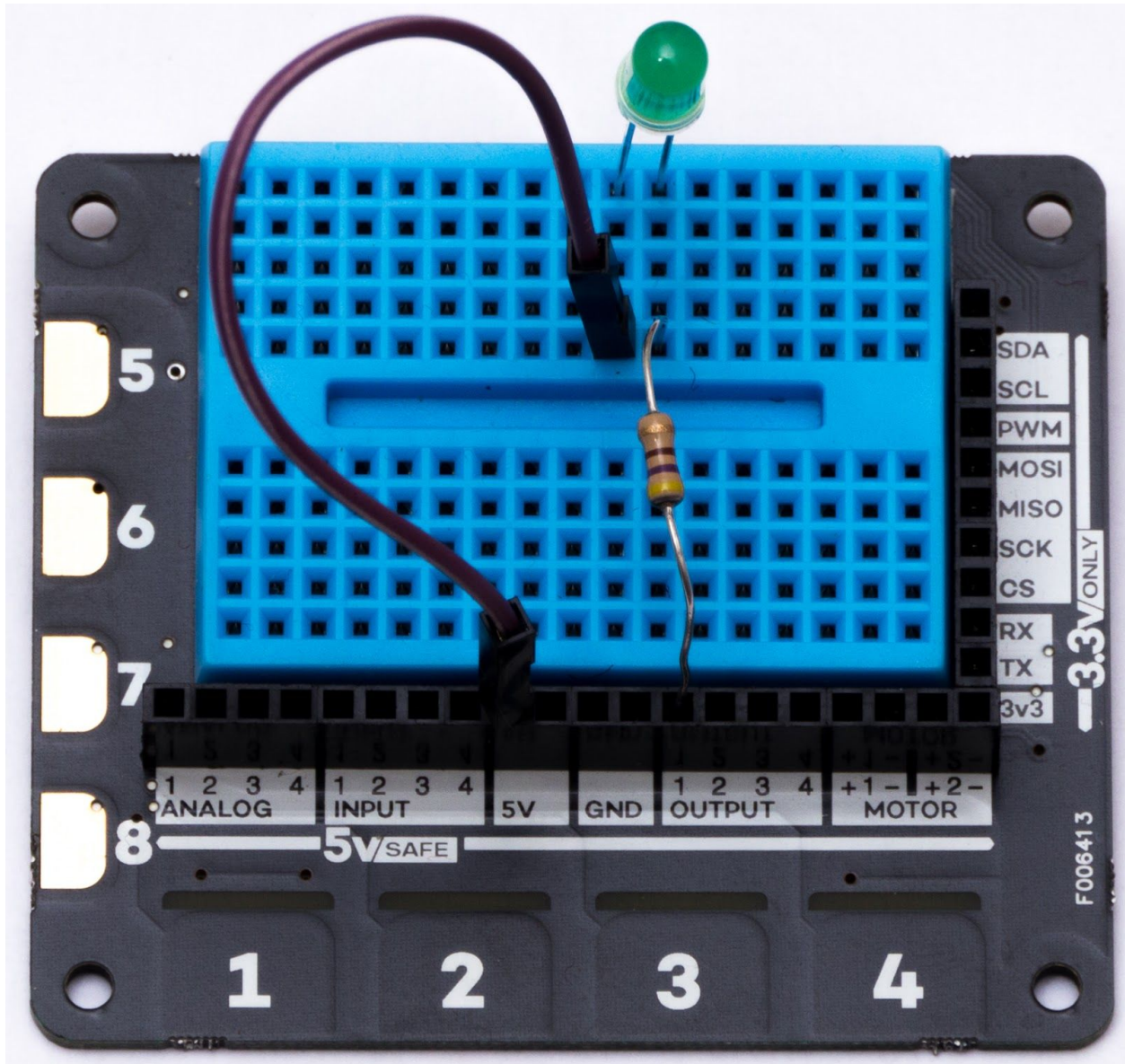


The wire with pins on each end is called a jumper wire, and like the components it can be pushed into the breadboard area on the Explorer HAT Pro to build circuits without soldering.



The Explorer HAT Pro includes a mini breadboard (the blue block full of holes) to build circuits directly on the HAT. On this breadboard, each vertical column of 5 holes are connected together electrically. (Two of these groups are highlighted with red borders in the picture above.) Electrical current will flow between holes in the same group as the 5 holes are all connected together inside the breadboard. The components need to be placed so that each leg is in a different column.

Below the breadboard are connectors to the 5V power (like the +ve end of a battery), GND (meaning ground, this is like the -ve end of a battery), inputs and outputs from the Raspberry Pi GPIO header.



For our circuit, the LED is connected to the 5V power connector on the Explorer HAT Pro (using the jumper wire). Be sure to connect your LED the correct way round; the longer leg should be in a group of holes connected via the jumper wire to the 5V connector.

The shorter leg of the LED is connected to output 1 on the connector using the resistor. This way power flowing from the 5V connector to the output has to flow through the resistor, which protects our LED from being damaged.

Now we are ready to write some code!

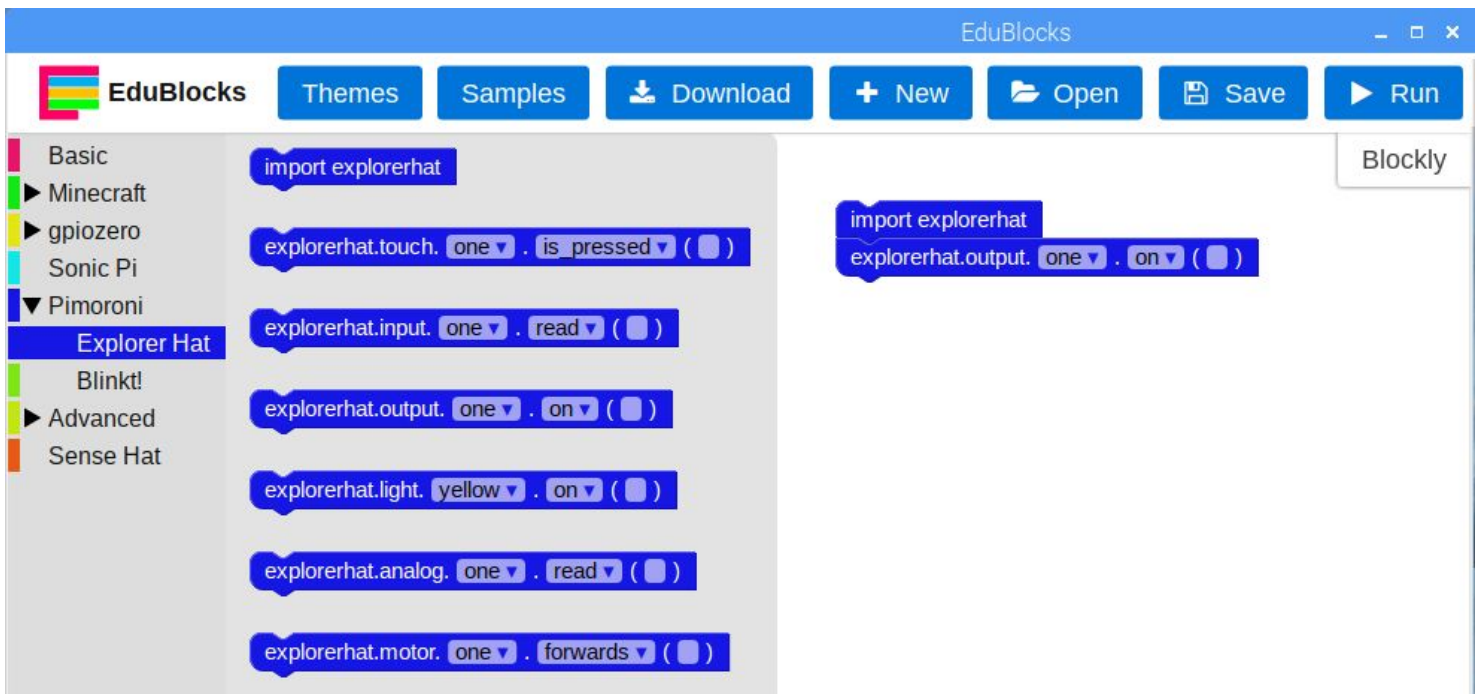


## Controlling an LED Using EduBlocks

EduBlocks on the Raspberry Pi includes code blocks for the Explorer HAT. You can find them in the **Pimoroni** blocks group.

We only need 2 blocks to test our LED circuit.

1. We need to import the explorerhat code library
2. We need an **explorerhat.output** block to turn on the output one

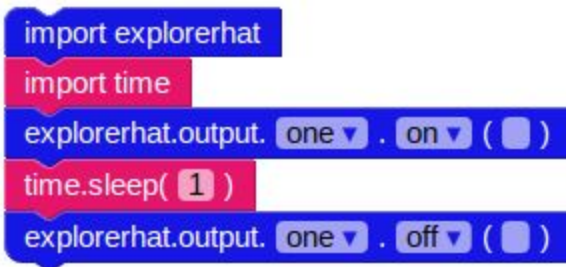


Click the 'Run' button to test the code. If your circuit is correctly built then the LED should light up. If your code ran without any errors but your LED did not light up then check your circuit. Did you connect the LED the right way round? Are the component legs in the right holes in the breadboard?

Press the Escape (Esc) key on your keyboard to close the output window which opened when the EduBlocks code ran. You will see that the LED stays on. This is because we have not added any code to turn it off.

If we want to see the LED turn on and then off again, then we need to make our code wait after turning on the LED. Otherwise it will turn off so quickly after turning on that we will never see it light up. We can use the 'time' code library to make our code wait before running any more instructions. To do this we need to add an **'import time'** block from the **Basic** blocks group to the start of our code.

Now we can use the 'time.sleep' block after we turn on our LED to make the code wait. Enter the number of seconds you want your code to wait in this block. Then add another **explorerhat.output** block on the end, and change the values in this block to turn output one off.

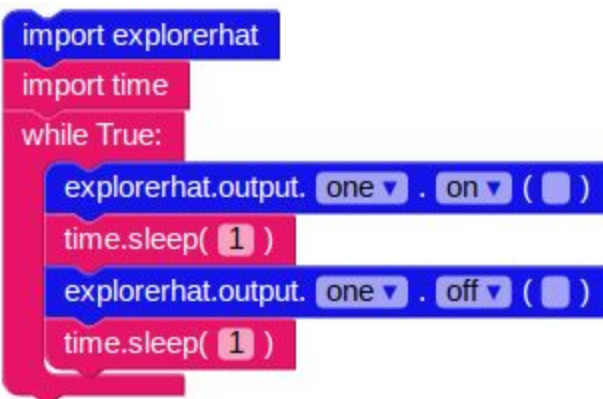


```
import explorerhat
import time
explorerhat.output.one.on( )
time.sleep( 1 )
explorerhat.output.one.off( )
```

Try running your code again. You should see your LED turn on, and then turn off again.

## Running Code in a Loop

We can add a loop to our program so that the LED keeps turning on and off. The '**while True**' loop will keep running forever (or until we stop our program). We need to add another wait at the end of the code in the loop, otherwise the code back at the top of the loop will be run as soon as the last line of code in the loop finishes. We want to wait long enough to see the LED turned off before we turn it on again. So we need to add a second **time.sleep** block at the end of the loop.



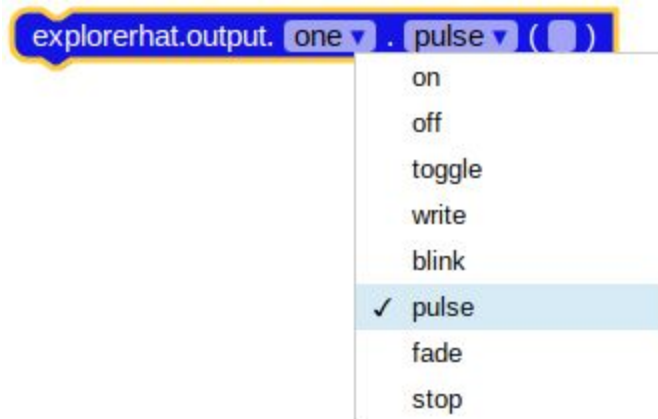
```
import explorerhat
import time
while True:
    explorerhat.output.one.on( )
    time.sleep( 1 )
    explorerhat.output.one.off( )
    time.sleep( 1 )
```

Run this code and you should see the LED turn on for 1 second then turn off for 1 second, and repeat until you stop the program. Hold down the CTRL key and press 'c' to stop your code.

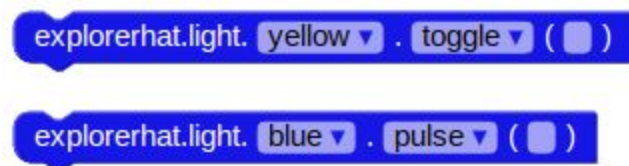
Remember you can see the Python code for your program at any time by clicking the 'Blockly' button in the EduBlocks editor.

## Experiment with Your Code!

The `explorerhat.output` block has more modes than just on/off. Try experimenting with some of the others:



The Explorer HAT Pro also has 4 LEDs built in. You can control these using the **`explorerhat.light`** blocks. Why not try adding some of these to your program?

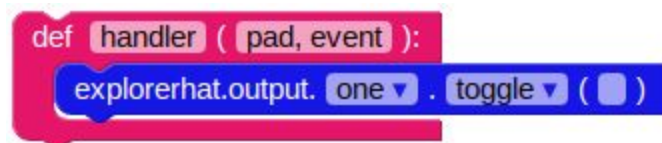


There are 4 outputs in total on the Explorer HAT. You could try adding more LEDs to your circuit. Remember each LED needs to take power from the 5V connector, and must have a resistor in the circuit before it is connected to an output. Why not try using some different colour LEDs?

## Explorer HAT Pro Touch Inputs

The Explorer HAT Pro features 8 capacitive touch pad inputs. These pads are numbered 1 - 8 on the HAT, and can detect when you touch them with your finger. Pads 5 - 8 are bare metal and can also be used to attached leads using croc clips so that you can detect touch in other objects (e.g. Bare conductive paint drawings, fruit or vegetables). To use these touch inputs in EduBlocks we need to write functions to link to the inputs. A function is some code inside a **def** block. The touch input handler function requires two arguments: **pad & event**.

Here is an example of a function to handle a touch input event. You can give your function any name you want in your code. We have called ours '**handler**'.



```
def handler ( pad, event ):  
  explorerhat.output. one . toggle ( )
```

When this function is linked to a touch input, it will be called each time that touch input registers an event. Events can include: **pressed**, **held**, **released**. When the handler function is called, it will be passed the number of the pad which raised the event, and the type of event. So you could use the value of the event argument in your code to run different code when you put your finger on a touch pad (**pressed**), or when you lift your finger off again (**released**). The pad number is useful if we decide to use the same handler function for more than one of the touch pads, as it will tell us which pad raised the event. In this simple example we are just going to link the handler function to touch pad number one, for the '**pressed**' event. So we do not need to use the value of the arguments. The code library still passes them to the function though, so we have to declare them inside the brackets of the **def** block.



Here is the complete program code. First we need to import the code libraries we are using in our program. Then we need to define our handler function. Inside our handler we are just going to toggle output **one** on and off each time the handler is called. So if our handler is called when the LED is off, then it will turn it on. If it is called when the LED is on, then it will turn it off.

After the function definition, we use an **explorerhat.touch** block to link the **pressed** event for touch pad number **one** to our function name '**handler**'. Now while the program is running, each time the touch pad one is pressed (touched with a finger) our handler function will be called and toggle the state of the output **one** (which should turn our LED on and off). We need to keep the program running of course, so we put a '**while True**' forever loop at the end so that our program does not exit. Inside this loop, we give the computer running our code time to do other things by putting in a short sleep of 0.2 seconds.

```
import explorerhat
import time
def handler ( pad,event ):
    explorerhat.output. one . toggle ( )
explorerhat.touch. one . pressed ( handler )
while True:
    time.sleep( 0.2 )
```

Run your code and you should be able to turn the LED in your circuit on and off by touching the pad number 1 on your Explorer HAT.

Why not try adding some more handler functions using what you have learned, and link these to the other touch pads?