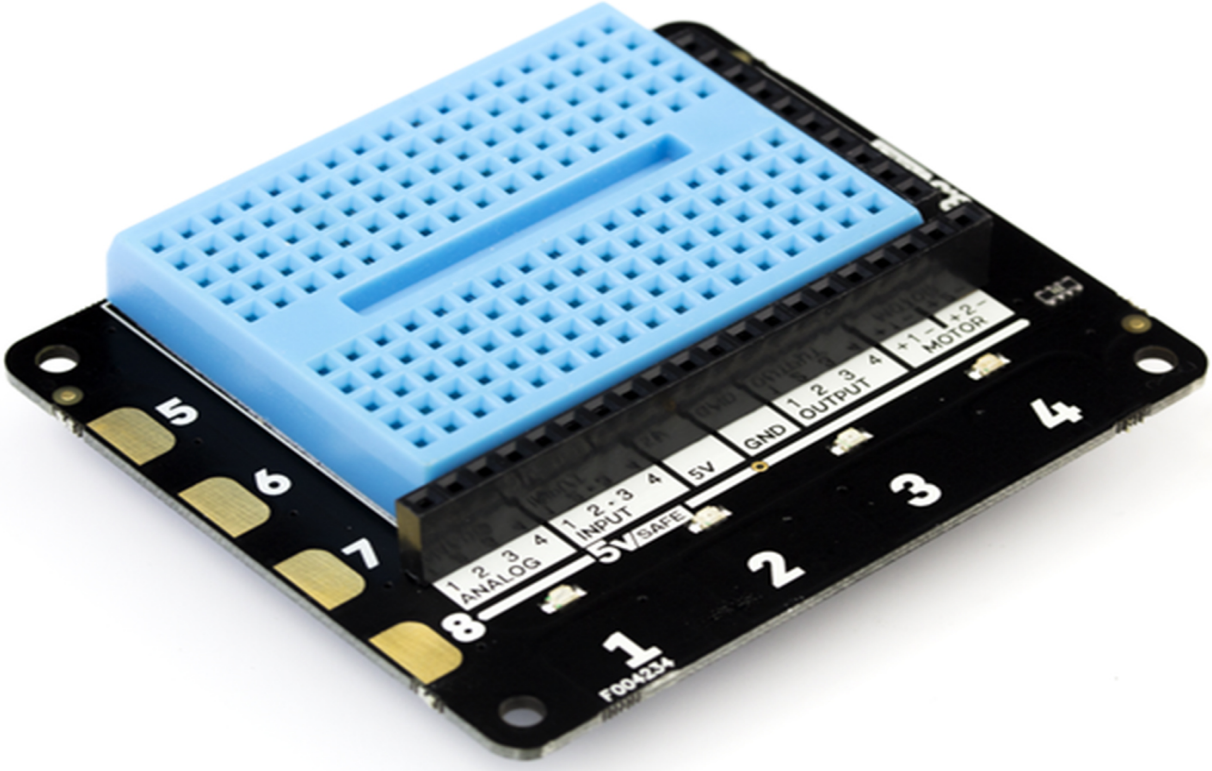
Explorer HAT tricks

fun things to try with your Pimoroni Explorer HAT



ROMILLY COCKING

Explorer HAT tricks

fun things to try with your Pimoroni Explorer HAT

Romilly Cocking

This book is for sale at http://leanpub.com/explorerhattricks

This version was published on 2020-10-08

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Romilly Cocking by spreading the word about this book on Twitter!

The suggested hashtag for this book is #explorerhattricks.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#explorerhattricks

Also By Romilly Cocking

Making the Shrimp

Learn APL on the $5 Raspberry Pi

A Simulation of Cerebellar Cortex

micro:bit MicroPython in 60 minutes

Contents

Preface

Explorer Hat Tricks - *Learning by Discovery*

Build fun Raspberry Pi projects that work

Have you ever wanted to build a realistic set of traffic lights, with beeps and a button for pedestrians to push?

How about making your own weather station?

Always wanted to build a Pi-powered robot but wondered how to control the motors?

This is the book for you.

With *Explorer Hat Tricks* you'll start working on practical examples from the very first chapter.

You'll learn about, and write programs using

- *Digital inputs* that allow you to detect what's happening in the outside world
- *Outputs* that can drive LEDs and relays
- *Touch sensors* that help you to control your projects
- *Analogue inputs* that you can use to track temperature, light, sound levels and more
- *Motor controls* for projects that move.

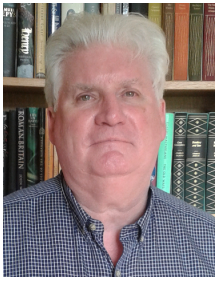
The book is written with beginners in mind. You won't need to solder anything, you don't need to know any electronics, and you'll learn about Python programming as you go.

You'll be guided through each project step-by-step, so you can learn by discovery.

The book will be published on Leanpub. That gives you

- a chance to see a sample before you buy the book
- free lifetime updates
- a 45-day money-back guarantee
- copy-protection-free content in several different formats.

About the author



Romilly Cocking

Romilly Cocking has been programming computers since 1958!

He has been programming in Python for almost 20 years.

He still loves learning and sharing what he's learned with others. He's written several books about single-board computers and physical computing.

He's a contributor to several Open Source projects, and in 2012 he founded Quick2Wire, a start-up that made add-on boards for the Raspberry Pi.

He's a regular presenter at the London-based Raspberry Pint meetups hosted at Microsoft Reactor London.

You can read his blog about Raspberry Pi, AI, robotics and electronics.

How to have fun with this book

What's this about?

This chapter covers what you may want to know or do before you start.

Get help if you need it

If you need help or have spotted an error, there's a Facebook group that's dedicated to the book.

Make sure you are registered!

One great benefit of buying books on Leanpub is that you can get free updates whenever the book changes.

To make sure you hear about changes when they become available, you'll need to tell Leanpub that you want to be notified.

Once you've purchased the book you can do that by visiting your Leanpub library.

Click on the cover of this book and tick the box marked *New version available*.

I'm planning follow-ons to this book.

If you also tick the box marked 'Share my email address with the author' I'll let you know when more information is available.

I'll respect your email - no spam, I promise!

Next, a bit of background on the Raspberry Pi.

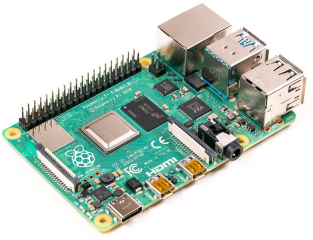
The Raspberry Pi

The Raspberry Pi is *everywhere*. The Pi was developed to encourage young people to learn to program, and it's attracted enthusiasts from every age group, gender, background and skill level.

It's been incredibly successful.

"We have sold 30 million high-quality, low-cost computers worldwide. Raspberry Pi has become the third best-selling general-purpose computer ever, behind only the Mac and the PC."

Philip Colligan, CEO, Raspberry Pi



Raspberry Pi model 4

As a result, there are fantastic resources available to Raspberry Pi users: extra hardware, software, books, videos and courses. There's also a large and supportive community to help you with any problems you encounter.

Physical computing

People use the Pi for all sorts of applications, and the latest version (model 4) is powerful enough for you to use it as a desktop computer. The Pi can also do things that most computers aren't designed for.

The Raspberry Pi has a row of GPIO pins that you can connect to *sensors* and *actuators* in the world around it. This is known as physical computing, and it's in physical computing that the Pi really comes into its own. Sensors observe what's happening in the real world, and actuators make something happen. A sensor might detect that a robot is about to hit a wall, and an actuator might drive or stop the robot motor.

The Pimoroni Explorer HAT

You'll be using special hardware in this book to make it easy for you to try out physical computing experiments and projects.

The hardware is called the *Explorer HAT Pro*, and it is made by a UK company called Pimoroni. Pimoroni was one of the first companies to make and sell add-ons for the Raspberry Pi.

This book will show you how to connect your Pi to external hardware and program your Pi to interact with the outside world.

The book is not an official Pimoroni product.

For the projects in the book, there is a discussion group run by the author on Facebook, and there's also a Discord channel which you can join here.

You can contact Pimoroni using their support forums if you have a problem with the Explorer HAT itself.

How much will the hardware cost?

Here's a rough guide in pounds sterling:

Item	Cost
Pimoroni Explorer HAT Pro	£19.50
Pimoroni parts kit	£11.40
Optional extras	£10

If you don't already have a Raspberry Pi, you will also need

Item	Cost
Raspberry Pi 4 starter kit	£90+

You may have some of all of the hardware already, and you can use an earlier model of the Pi if you have one.

The only requirement is that your Pi has a 40-pin GPIO header and WiFi or an Ethernet connection.

You will also need a TV or computer monitor with an HDMI connection. Details are on the Raspberry Pi website.

Optional extras include an LDR (light-dependent resistor) and a small electric motor. They are available from Pimoroni. You'll find details of how to order below.

The code has been tested on a Pi 4B and a Pi 3B/3B+.

Some will run with the Explorer pHAT and a Zero WH.

What skills do you need?

The ability to read, willingness to learn, and fingers nimble enough to push wires into the holes in the HAT.

What skills will you learn?

You'll learn how to

1. Create and run Python programs
2. Use jumper wires, electronic components and a breadboard to wire up simple electronic projects
3. Interact with the real world by running programs on the Raspberry Pi.

What is Python?

Python is a programming language. You'll learn how to write Python programs (sometimes called Python scripts) to tell your computer what to do.

Why use Python?

Python is widely used and easy to learn. Most of the software written for the Raspberry Pi is written in Python.

Lots of well-known organisations use Python, including

- Instagram
- Google
- NASA
- Netflix
- Uber

so Python is a great language to know.

Your choices

You have a choice about which hardware to use for the exercises and projects, and you can take several different paths through the book depending on your background and goals.

Choosing your hardware

You'll need to get a Pimoroni Explorer HAT Pro.

If you have one of the original Explorer HATs or an Explorer pHAT you can use them but you won't be able to do all of the experiments and projects.

You'll need a Raspberry Pi to drive your Explorer HAT.

If you have a Pi already

If you already have a suitable Raspberry Pi and it's connected to the Internet with a monitor, keyboard and mouse, you'll just need the HAT, HAT kit and maybe the optional extras.

Suitable Pi models include the Raspberry Pi model 1B+, 2B or 2B+, Pi 3B or 3B+, Pi 4B or Pi Zero W.

If you're starting from scratch

If you're starting from scratch you should get a Raspberry Pi model 4, a mouse, keyboard, SD card, power supply and (maybe) a monitor. The Pimoroni starter kit has them all apart from the monitor.

What to order

You can order all the hardware (apart from the monitor or TV) from Pimoroni's UK shop.

You'll find links that you can use to order each item. These are *not* affiliate links. In other words, I don't receive any commission when you buy things using the links. That way you can be sure that the advice I give you is unbiased.

Here's what you will need for the core experiments and projects:

Pimoroni Explorer HAT Pro

Explorer HAT Pro parts kit

Optional:

LDR (light-dependent resistor)

Miniature motor

Extra jumper leads and potentiometers

If you don't have a Pi already:

Raspberry Pi 4 starter kit - 2 GB

The 2 GB version is fine for this book. If you want to do really demanding projects later you may want to get the 4 GB or 8 GB version.

A quick note about spelling

I'm based in the UK, as are Pimoroni and the Raspberry Pi foundation, so this book uses UK spelling.

You'll normally see *colour*, *analogue* and so forth instead of *color* or *analog*.

But there are some exceptions!

The Pimoroni library, and the board it supports, use US spelling for `analog` so the code examples have to do the same.

A quick preview

Here's a quick guide to the contents of this book.

Each box on the right is a chapter. The thumbnail images in each chapter show the projects you'll do.

Explorer HAT tricks 

Contents

Next

Once your hardware has arrived, you can start having fun!

Ready, steady, go!

You've got the required hardware. Time to get going!

What's this about?

This chapter covers

1. preparing your hardware
2. installing the required software
3. checking the installation.

Prepare your Raspberry Pi

If you haven't got your Raspberry Pi connected and ready for you to work on it, now is the time to prepare it.

Setting up a Pi for the first time is outside the scope of this book, but you'll find very clear instructions on the Raspberry Pi website.

Once your Pi is running you will need to plug the Pimoroni Explorer HAT Pro onto your Pi and install the extra software that you will need.

Installing the HAT on the Pi

You need to carry out the hardware installation with the Pi shut down and powered off!

Take a close look at the Explorer HAT and the Pi.

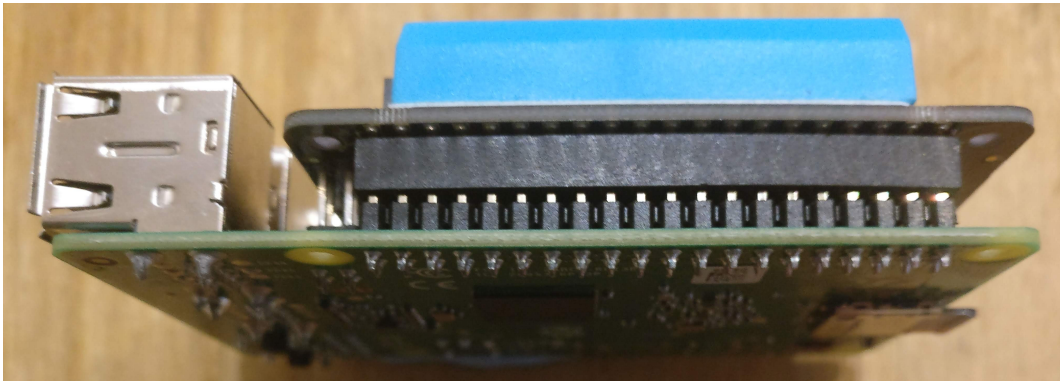
The bottom of the Explorer HAT has a connector plug with 40 holes in two rows of 20.

The Pi has a header consisting of 40 pins that match the holes in the HAT's connector.

Line up the HAT's connector with the Pi's header so that the pins are ready to slip into the connector's holes.

It's easy to have the HAT a little too far to the left or the right. Make sure that all the pins have a hole above them!

If you look at the HAT and Pi from the side they should look like this.



Raspberry Pi with HAT plugged in correctly

Enabling interfaces

The Explorer HAT needs to use some special hardware on the Pi which you need to enable. You'll find detailed instructions below, or you can watch the video on YouTube.

1. Click on the Raspberry Pi icon in the top left corner of the desktop.
2. Select `Preferences/Raspberry Pi configuration`.
3. Select the `Interfaces` tab.
4. Enable `I2C` and `SPI`. Leave the other items unchanged.

   a) Click on `I2C`.
   b) Select `yes` to enable it and press Enter.
   c) Click on `SPI`.
   d) Select `yes` to enable it and press Enter.

5. Click on `OK` to save the new configuration.

Now your Pi is correctly configured for use with the Explorer HAT.

Use Python 3

There are two versions of Python in use at the moment: Python 2 and Python 3. Both are installed on your Raspberry Pi, but all the examples in this book use Python 3.

Python 3 is the more up-to-date version. It fixes a few problems with the older version, and it has a lot of useful new features. While you can use the Explorer HAT with the old version of Python, it's not recommended, and this book shows you how to drive your HAT using Python 3.

Install software

Once you've got the HAT safely plugged in, turn on your Pi.

You'll need to install two lots of software before you can run your first experiment. Before you do, it's a good idea to make sure that the software on your Pi is up to date.

There's a great guide to updating the software on your Pi on the Pimoroni website.

Once you've done that you'll be ready to install the additional software.

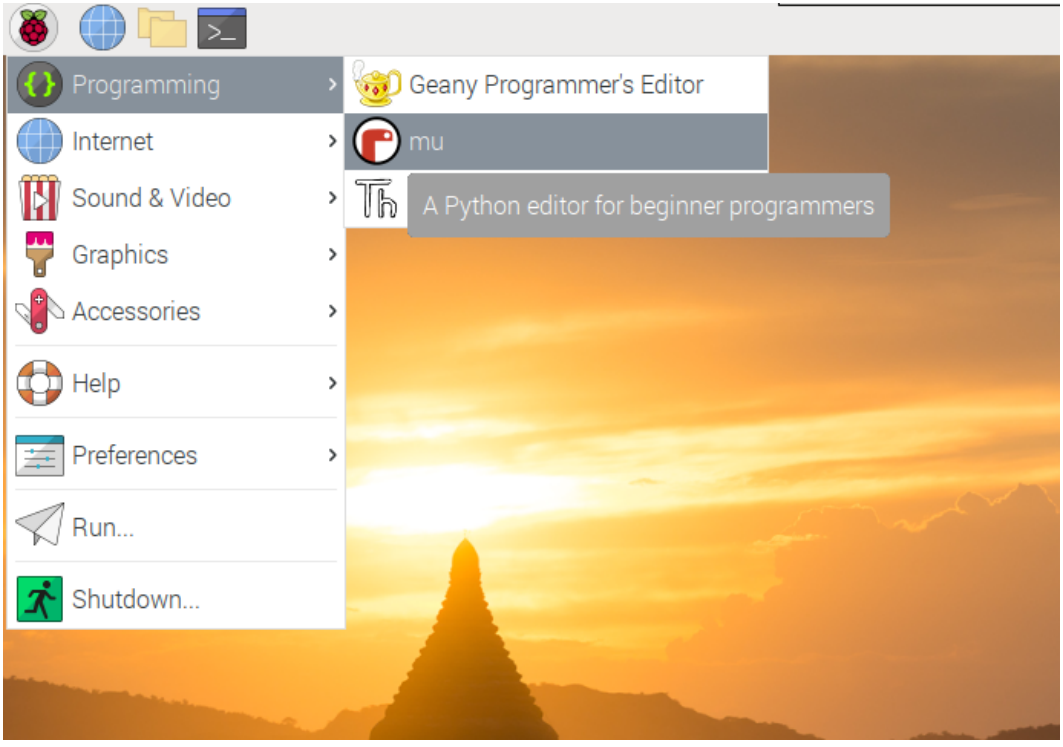
Install *mu*

The first piece of software is a Python editor called *mu*. There are lots of editors available, but *mu* is designed for beginners and it's recommended by the Raspberry Pi foundation.

Note: If you're already an experienced Python programmer, you can use your usual editor, but you'll need to run the programs from the command line.

Mu is easy to install from the main Raspberry Pi menu. Click on the raspberry on the left of your screen and select `Programming`. From *that* menu select `mu`.

Your screen should look like this.



Installing the mu editor

Click on `mu` and the Pi will install it.

This can take a while, so now is a good time to take a short break!

Install the Pimoroni Explorer HAT library

Next you'll need to install Pimoroni's library for the Explorer HAT. The library is a collection of pre-written software that makes it easier for you to write programs that control the HAT.

From the Raspberry Pi desktop, open a terminal window by typing `Ctrl-Shift-T`.

In the window, type `sudo apt-get install python3-explorerhat`.

It's likely that the library is already installed, in which case the command will tell you so, but if it isn't the command will install it.

Install code for the projects

Although you will learn most if you type in the code for each project, you may not have time, and you may want to compare your code with what's in the book.

You can download the code for all of the projects here. Once you've downloaded it, unzip it into a directory of your choice.

Check the installation

It's time to check that everything is working.

In the process that follows you'll get a sneak preview of the first experiments you'll work on. Don't worry too much about what you're doing; that's explained in more detail in the next chapter.

1. If you closed the terminal window, re-open it by typing `Ctrl-Shift-T`.
2. In the terminal window, type `python3`. You should see some output followed by a prompt `>>>`.
3. Type `import explorerhat as eh`. You should see a line *Explorer HAT Pro detected...* followed by another prompt `>>>`.
4. Type `eh.light.red.on()`. The red LED (light-emitting diode) on your HAT should light up, and the message *True* will be displayed.
5. Type `eh.light.red.off()`. The LED should go off and *True* will be displayed again.
6. Type `exit()` to terminate your Python session.
7. Type `Ctrl-D` to close the terminal window.

Summary

Congratulations! You've prepared and checked the the HAT hardware and software.

Next

You're ready to start your first project and learn more about blinking the HAT's on-board LEDs.

Hello, explorer!

Now that you have your Raspberry Pi prepared and your Explorer HAT ready for action, it's time to start experimenting.

What's this about?

In this chapter you'll learn how to control the LEDs (light-emitting diodes) that are built in to the Explorer HAT Pro.

An LED is an electronic component that emits light when you apply a voltage to it. LEDs are available in many different sizes and colours. The Explorer HAT Pro has four LEDs built in. They are coloured red, yellow, blue and green.

LEDs are really useful because they are easy to see and easy to turn on and off. Turning an LED on or off is a great way for your program to tell you that something has happened, and it's really easy to do from Python.

Turning an LED on and off from Python

You'll be writing a lot of programs as you work through this book, but in your first experiment you will run some Python statements directly.

You'll be using a feature of the *mu* editor called the REPL.

REPL is short for **R**ead **E**valuate **P**rint **L**oop. When you use *mu*'s REPL, *mu* will

1. **R**ead what you type
2. **E**valuate what you type using Python
3. **P**rint the result, and
4. **L**oop back to step 1.

In the previous chapter you ran some Python test code from the terminal window using Python's built-in REPL. In this chapter you'll run the same code from inside *mu*, and then you'll use *mu* to create and run some Python programs.

1. Start the *mu* editor by clicking on the `Raspberry Pi/Programming/mu` menu.
2. Click on the keyboard icon marked `REPL`. A new pane will open at the bottom of the *mu* window.
3. In the REPL pane, type `import explorerhat as eh`. This is a Python statement. It tells Python that you are going to use Pimoroni's `explorerhat` library, and that you will refer to it as `eh` to save typing.
4. Type `eh.light.red.on()`. The red LED on your Explorer HAT should turn on and you should see the word `True` appear in the REPL pane. The statement runs the method (function) `on()` on the red LED, and returns a result `True`, which is displayed in the REPL pane.
5. Type `eh.light.red.off()`. The red LED on your Explorer HAT should turn off, and the result `True` is displayed.

Control an LED from inside a Python program

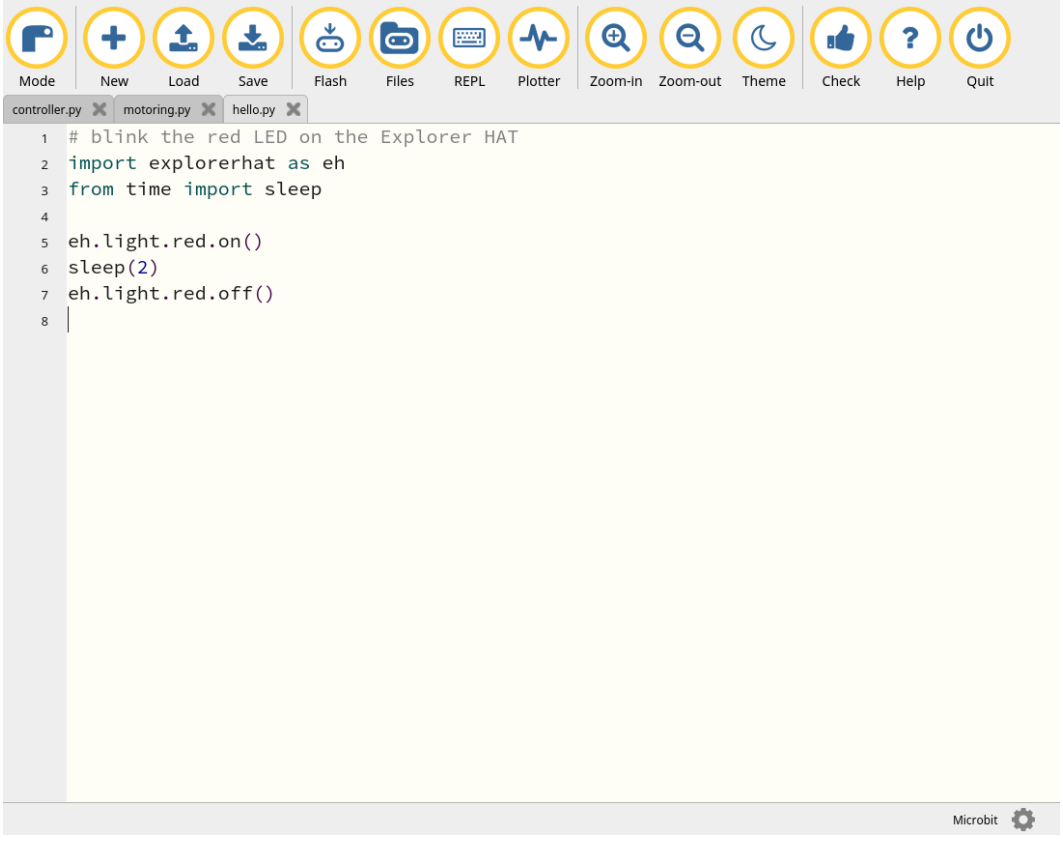
In this example, and the ones that follow, you'll see line numbers beside the lines of code. *Do not enter the line numbers!* They are there so that you can see which line of code the book is talking about.

1. Click on the REPL icon again to close the REPL window.
2. In the main *mu* window, type the following program:

```
# blink the red LED on the Explorer HAT
import explorerhat as eh
from time import sleep

eh.light.red.on()
sleep(2)
eh.light.red.off()
```

The *mu* editor should look like this:



A simple program in mu

1. Click on the `Save` button. Name the file `hello.py`

2. Click on the `Run` button. Watch your Explorer HAT. The red LED should turn on for two seconds and then turn off again.

What's going on?

Line 1 of the program starts with a `#` symbol, which tells Python that the line is a *comment*. A comment is a program line that isn't meant to be executed. The comment is there to tell a human reader something about the program, and Python will ignore the comment line when it executes the program.

Line 2 tells Python that you want to use the `explorerhat` package and that you will refer to it as `eh`. A *package* is a collection of Python code that someone has made available to be used in other programs. In this case, the `explorerhat` package contains code written by Phil Howard (@gadgetoid), who works for Pimoroni. The package makes it easy for you to use the features of the Pimoroni Explorer HAT from Python.

The part of the line that reads `as eh` just saves a bit of typing. If you just said `import explorerhat` you'd need to use the name in full every time you wanted to refer to the package.

Line 3 tells Python that you want to import the `sleep` function from Python's `time` package. You'll use the `sleep` function on line 6 below.

Line 4 is blank. It's there to separate the `import` statements from the code that follows, so that your program is easier for you and others to read. Python will ignore the blank line when it executes your program.

Line 5 tells Python to turn the red LED on. `eh.light.red` refers to the red LED; `on()` tells Python to turn the LED on by running some code in the `explorerhat` package.

Line 6 tells Python to sleep (wait) for *two seconds* before running the next line.

Line 7 tells Python to turn the LED off.

Blink an LED repeatedly

That program works, but you might want to blink the LED more than once. In this section you'll learn how to blink the LED repeatedly.

You'll also learn about a key feature of Python, which is different from many other programming languages.

In Python, **spaces matter**. When you look at the code for your next experiment, you'll see that lines 5-8 are indented by 4 spaces. *If you don't put those spaces in your program, it won't work correctly!* You'll find out why you need those spaces when you read the line-by-line explanation of the program below.

Typing in the program

1. Open a new file in *mu* by clicking the `New` button.
2. Type in the following program:

```
1 import explorerhat as eh
2 from time import sleep
3
4 while True:
5     eh.light.red.on()
6     sleep(1)
7     eh.light.red.off()
8     sleep(1)
```

Running the program

Mu lets you execute the program you've written using its `Run` button, but the way it stops a running program does not play well with the Explorer HAT library.

For that reason, it's best to save your program in *mu* and to run your program from a terminal window.

Here's what you should do:

1. Save the program from within *mu*:
   a) Click the `Save` button.
   b) If asked, type the name of the program: `blink.py`.
2. Open a terminal window by typing `Ctrl-Shift-T`.

3. Go to the directory where *mu* stored your code by typing `cd /home/pi/mu_code`.
4. Run the program by typing `python3 blink.py`.

While your program runs, the red LED on your Explorer HAT should turn on and off every second.

When you get bored, type `Ctrl-C`. The Explorer HAT library will print messages about cleaning up and the program will stop.

What's *this* program doing?

Line 1 should be familiar by now. It imports Pimoroni's `explorerhat` package and allows the program to use `eh` as a shorthand name.

As before, **Line 2** imports the `sleep` function from Python's `time` package.

Line 3 is new. Python's `while` statement tells Python to execute the code that follows over and over again. Python programmers call that a `while` *loop*. There are a couple of things you need to know about `while` loops.

1. `while` is followed by a *condition*. That's a Python expression which can take the value `True` or `False`. The code that follows `while <condition>:` is repeated over and over for as long as the condition is true. In this case, the condition is always true so the code will be repeated for ever - or at least until someone stops the program.
2. You often want to execute a few statements over and over, but not the whole of the program. A little later you'll code a program that does just that. For now, what you need to know is that you tell Python which code to repeat by *indenting* it - putting extra spaces in front of each line you want to repeat. In this example, you need to indent lines 5-8 as they are the ones to be repeated.

Lines 5,6,7 and 8 will turn the red LED on, wait one second, turn it off, and wait for another second.

Since lines 5-8 are repeated, the LED will blink on and off repeatedly, until you stop the program.

Red is not the only LED

The Pro version of the Explorer HAT has four LEDs: red, yellow, blue and green.

In the next experiment you'll write a program that will turn them on, one at a time, and then turn them off again. The program will do that ten times and then stop.

Using *mu*'s `New` button, open a new file and enter the following Python code. As before, make sure you enter four spaces before each of the indented lines.

```
1  import explorerhat as eh
2  from time import sleep
3
4  for i in range(10):
5      eh.light.blue.on()
6      sleep(0.1)
7      eh.light.yellow.on()
8      sleep(0.1)
9      eh.light.red.on()
10     sleep(0.1)
11     eh.light.green.on()
12     sleep(0.1)
13     eh.light.off()
14     sleep(0.1)
15 print('Time to stop!')
```

What will this do?

Lines 1 to 3 should be very familiar.

Line 4 is new; it is another type of Python loop. The `for` loop, combined with the `range` function, executes the code block that follows a given number of times.

Lines 5-14 should look fairly familiar. They will turn on the four built-in LEDs, with pauses in between, and then turn all four off at once. The statement `eh.light.off()` doesn't specify a particular light so it affects them all. The pauses are for 0.1 seconds (one tenth of a second).

Line 15 uses a new Python function: `print`, which will print the string 'Time to stop!'. A string is a series of characters (letters, numbers and punctuation) - for example, a message. In Python you can define a string by putting the string in quotation (speech) marks. As you'll see shortly, the `print` statement will tell Python to display the string as output. *mu* will display the output when you run the program.

To test your program, save it as `all-blink.py` and the press the `Run` button. You should see all four LEDs light up in sequence, then turn off; that should be repeated ten times, and then the program will print *Time to stop!*

Summary

You've covered a lot of ground in these starter projects.

You now know how to

1. execute Python statements using a REPL
2. write, save and run a Python program using *mu*
3. import and use Python packages
4. turn the Explorer HAT LEDs on and off
5. delay a program using Python's `sleep` function
6. use Python's `print` function to display output to the user
7. use `for` loops and `while` loops.

Next

The next two chapters use the on-board LEDs for more complex projects.

Traffic lights project - first version

The next project simulates a set of traffic lights using the LEDs on the Explorer HAT.

What's this about?

In this chapter you will write a more complex program to control a simulated set of traffic lights, and you will use several powerful features of the Python language.

UK traffic light sequence

Adapted from Wikipedia:

In Britain, and much of Europe, normal traffic lights follow this sequence:

Red - Stop. Do not go
Red and amber - Get ready to go, but do not go yet
Green - Go if the intersection or crossing is clear; you are not allowed to block the intersection or crossing
Amber - Stop if it is safe to do so.

You will write a Python program which will run through the traffic light sequence over and over again.

Like the earlier program, this one starts by importing the library code that it needs.

required imports

```
import explorerhat as eh
from time import sleep
```

Some new Python features

In order to keep the program simple and readable, the code uses some Python features that weren't needed in the previous examples.

The program uses *constants*. Constants refer to values that don't change during the execution of the program.

Python programmers usually write the names of constants in upper case.

The code fragment below defines constants for the three colours of traffic light - red, yellow and green.

Setting Python constants

```
RED = eh.light.red
AMBER = eh.light.yellow
GREEN = eh.light.green
```

The next code fragment creates a Python *list*.

In Python, a list is just a sequence of values.

Since the program needs to turn unwanted lights off when it turns wanted lights on, it will be helpful to create a list containing all three lights.

Listing all the lights

```
ALL_LIGHTS = [RED, AMBER, GREEN]
```

The part of the program that does the work is a `while` loop like the one in the LED blinking example from the previous chapter.

Here it is:

Simulating the traffic lights

```
while True:
        show(5, RED)
        show(1, RED, AMBER)
        show(5, GREEN)
        show(1, AMBER)
```

Defining Python functions

As you can see, the code refers to something called `show`.

`show` is a function which you need to define in the program. You'll see its definition soon.

Functions allow you to define code that you need to use over and over again without having to type the code repeatedly.

It's a bad idea to have repeated code in your programs. If you find a bug in repeated code you will have to fix it in several different places.

It also makes the code harder to read because it's longer.

The `show` function turns the required lights on, turns the others off and then waits for a period of time.

The values in brackets specify the length of the delay and which lights should be turned on.

Here's the definition of `show`.

```
def show(duration, *lights):
    for current_light in ALL_LIGHTS:
        if current_light in lights:
            current_light.on()
        else:
            current_light.off()
    sleep(duration)
```

The definition of `show` makes use of some useful Python features.

The names inside the brackets after the name `show` specify the values that will be provided when the function is called in your program.

In this case, `duration` is the length of time the lights should be turned on for. The function will wait until the specified time has elapsed before carrying on to the next part of your program.

In the earlier code which called `show`, the function was sometimes asked to turn on a single light, and sometimes given two lights to turn on (red and amber).

`*lights` tells Python that the function will be called with *one or more* values, which can be referred to within the function using the name `lights`.

Python would even allow you to call `show` with just a duration, in which case `lights` would be an *empty list* - a list with no items in it.

The first statement is a `for` statement. It will run through the block of code that follows several times - once for each light in the list `*lights`.

In Python, blocks of code are **indented** - spaced in from the code above.

The block that follows the `for` statement is four lines long, starting with `if` and ending with `light.off()`.

`current_light` is a variable. Each time the code block is executed, the variable `current_light` will refer to the next light in `ALL_LIGHTS`, which is the list [RED, AMBER, GREEN].

The code block that will be repeated is an `if` statement.

In Python, an `if` is followed by a condition, and then by some code that will be executed if the condition is true.

In this case, the `if` statement includes an `else`, which will be executed it the condition is *not* true.

The condition uses Python's `in` syntax.

The condition will be *True* if the current value of `current_light` occurs in `*lights`, which is the list of lights to be turned on. It will be *False* if the light is not in the list.

For each light (red, amber or green), the light will be turned on or off, depending on whether it's in `*lights`, the list of lights given when `show` is called.

The complete program

Here is the whole program:

```
1  import explorerhat as eh
2  from time import sleep
3
4  # Name the built-in lights so the code below is more readable.
5  RED = eh.light.red
6  AMBER = eh.light.yellow
7  GREEN = eh.light.green
8  ALL_LIGHTS = [RED, AMBER, GREEN]
9
10
11 def show(duration, *lights):
12     for current_light in ALL_LIGHTS:
13         if current_light in lights:
14             current_light.on()
15         else:
16             current_light.off()
17     sleep(duration)
18
19
20 while True:
21     show(5, RED)
```

```
22      show(1, RED, AMBER)
23      show(5, GREEN)
24      show(1, AMBER)
```

Once you've entered and saved the program as `traffic-lights-01.py`, open a terminal window, make sure you're in the correct directory, and type `python3 traffic-lights-01.py`.

The terminal window will display a message: `Explorer HAT Pro detected`.

After that you should see the traffic lights cycling through the sequence.

Terminate the program by typing `Ctrl-C`.

The lights will stop and you'll see a message from the Explorer HAT library telling you that it is turning things off and tidying up.

Summary

You've seen how to

- use lists in Python
- specify a list of arguments when you define a function
- use `if` statements to control the flow of your program
- use Python's `in` syntax to decide if something is in a list.

Next

In the next chapter you'll write another project that will flash messages in Morse code!

SOS! - send Morse code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

What's this about?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Morse code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Sending Morse code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

The program

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Next

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

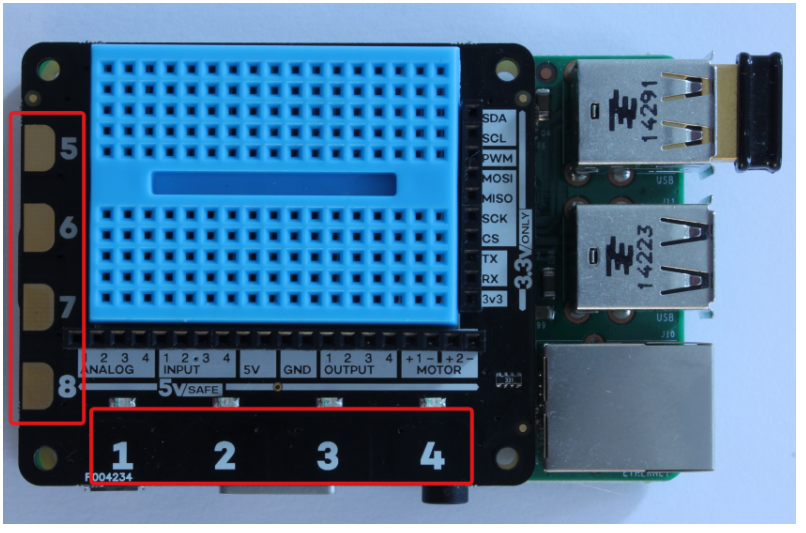
The touchy Explorer HAT

What's this about?

Most Raspberry Pi projects need input from the user. Push buttons work well for some projects, and you'll learn how to use them in a later chapter, but the Explorer HAT has a simpler alternative.

Touchpads

If you look at the HAT you can see eight little pads next to the digits 1 to 8.



HAT Touchpads 1-8

These are **touch-sensitive** inputs; you can write a program that detects when someone touches one (or more than one) of the little pads.

Using *mu*, create a new program with the following code:

```
1 import explorerhat
2 from time import sleep
3
4 while True:
5     explorerhat.light.red.off()
6     if explorerhat.touch.three.is_pressed():
7         explorerhat.light.red.on()
8     sleep(0.1)
```

Save it as `led-touch-01.py` and run it.

Now touch the pad numbered 3. The red LED should come on. When you take your finger off the button the LED should go off.

Here's how the code works:

Lines 1-2 are old friends. They import the packages required by the program.

Line 4 creates a `while` loop.

Line 5 turns the red LED off.

Lines 6 and 7 introduce the Python `if` statement. Line 6 checks to see if you are touching pad 3; if you are, Python will execute line 7, which will turn the red LED on.

If you didn't have line 5, what would happen?

As soon as you touched pad 3, the red LED would be turned on *and would stay on for ever*!

When you've finished experimenting with the program, stop it by typing `Ctrl-C`.

That version of the program works, but it's not as clear as it might be. At first sight, line 5 is a bit puzzling.

There's another way of writing the program which makes things a bit clearer.

```
1  import explorerhat
2  from time import sleep
3
4
5  while True:
6      if explorerhat.touch.three.is_pressed():
7          explorerhat.light.red.on()
8      else:
9          explorerhat.light.red.off()
10     sleep(0.1)
```

This uses an alternate form of the `if` statement. If pad 3 is touched, line 6 will turn the LED on. If not, line 8 will turn it off.

Here's a version of program that uses all four built-in LEDs and controls them using pads 1-4.

```
1  import explorerhat
2  from time import sleep
3
4
5  while True:
6      explorerhat.light.off()
7      if explorerhat.touch.one.is_pressed():
8          explorerhat.light.blue.on()
9      if explorerhat.touch.two.is_pressed():
10         explorerhat.light.yellow.on()
11     if explorerhat.touch.three.is_pressed():
12         explorerhat.light.red.on()
13     if explorerhat.touch.four.is_pressed():
14         explorerhat.light.green.on()
15     sleep(0.1)
```

Type the program in, save it as `led-touch-03.py` and run it as before.

When done, stop the program by typing `Ctrl-C`.

Like version 1, this turns LEDs off at the start of the `while` loop and then turns each LED on if the corresponding pad is touched. If you touch more than one pad, more than one LED will turn on.

Here's a tip for Python experts!

The final version uses indexing to dramatically simplify the code.

Pimoroni's Explorer HAT library represents lights and touchpads using a class called `ObjectCollection` which allows you to refer to a specific light or touchpad by name or index. The example below uses an index that runs through the values `0-3` to check the first four touchpads in turn, setting the corresponding light on or off.

```
import explorerhat as eh
from time import sleep


while True:
    for i in range(4):
        if eh.touch[i].is_pressed():
            eh.light[i].on()
        else:
            eh.light[i].off()
    sleep(0.1)
```

Type the program in, save it as `led-touch-04.py` and run it as before.

When done, stop the program by typing `Ctrl-C`.

Summary

You've now learned how to use the touchpads on the HAT.

Next

So far you've made good use of the HAT's four on-board LEDs, but some projects need more.

When building a prototype you'll probably use LEDs and resistors with a breadboard.

You'll do that in the next chapter.

LED to the breadboard

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

What's this about?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Breadboards

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

A quick experiment

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Fritzing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Time to get wiring!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Blinking the LED

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Blinking program

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Touchpad control

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Next

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

What's the buzz?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

What's this about?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Two sorts of buzzer

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Wiring up the buzzer

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Driving the buzzer

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Buzzing Morse code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Next

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Traffic lights project - second version

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

What's this about?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Next

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Measuring analogue values

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

What's this about?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Why do you need analogue inputs?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Analogue measurements with the Explorer HAT

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Reading analogue values in your program

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Using a potentiometer

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Start building a weather station

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Measuring temperature with the TMP36

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Measuring light levels with an LDR

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Using a voltage divider

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Next

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

The plot thickens

What's this about?

One of the best ways to understand values that change over time is to plot them on a graph.

Python has good plotting libraries, and the *mu* editor makes them very simple to use.

In this chapter you'll learn how to use *mu*'s plotting to display the outputs from your weather station.

You'll be able to see how the temperature and light level change hour by hour.

Calculating the LDR's resistance

Different models of LDR have different resistance values, and this will affect how your circuit operates. Below you'll see how to adjust your code to suit a particular LDR.

You'll need to modify the program that you wrote in the previous chapter.

The analogue input on the Explorer HAT has a high input resistance, so little current flows through it when you're measuring a voltage.

That means that almost all the current flowing through the fixed resistor (R1) also flows though the LDR.

You can use Ohm's law to calculate the resistance of the LDR from the voltage connected to analogue input 2.

Don't worry if you dislike mathematics: the formula you need is already coded in the Python program below.

If `V2` is the measured voltage, `R1` is 10k ohms and `ldr` is the resistance of the LDR,

5 * ldr / (R1 + ldr) = V2
5 * ldr = V2 * (R1 + ldr)
5 * ldr - V2 * ldr = V2 * R1
(5 - V2) * ldr = V2 * R1
ldr = V2 * R1 / (5 - V2)

Here's the program:

```
import explorerhat as eh
from time import sleep


while True:
    v2 = eh.analog.two.read()
    r1 = 10000
    ldr = r1 * v2 / (5 - v2)
    print('v2 is %4.1f and LDR is %4.1f ohms' % (v2, ldr))
    sleep(1)
```

Save it as `calibrate.py` and run it in the usual way.

While it's running, shade the LDR with your hand so that it's in the dark. You'll see the printed values change.

Here's the output I got:

```
pi$ python3 calibrate.py
Explorer HAT Pro detected...
v2 is  0.5 and LDR is 1040.0 ohms
v2 is  0.5 and LDR is 1054.6 ohms
v2 is  1.1 and LDR is 2745.3 ohms
v2 is  2.2 and LDR is 7774.6 ohms
v2 is  3.1 and LDR is 15773.2 ohms
v2 is  3.3 and LDR is 18752.2 ohms
v2 is  0.5 and LDR is 1084.0 ohms
v2 is  0.5 and LDR is 1032.7 ohms
v2 is  0.5 and LDR is 1047.3 ohms
^CTraceback (most recent call last):
  File "calibrate.py", line 10, in <module>
    sleep(1)
KeyboardInterrupt

Explorer HAT exiting cleanly, please wait...
Stopping flashy things...
Stopping user tasks...
Cleaning up...
Goodbye!
pi$
```

The program you just tested shows the resistance of the LDR. How can you convert that to a light level?

Measuring light levels

Scientists measure the illuminance (brightness) of light in *lux*.

You can usually find out how an LDR's resistance varies with the light level from its datasheet, but we'll use a rough calculation based on a typical LDR.

In total darkness the LDR has a resistance of 1M ohms (one *million* ohms).

When the LDR is indoors in a room lit by bright sunlight, it might be exposed to about 300 lux. Its resistance would be about 3k ohms (3000 ohms).

When the LDR is shaded by your hand it might be exposed to about 1 lux. Its resistance would be about 100k ohms (100,000 ohms).

Those values match fairly well with the measurements shown above in the output from `calibrate.py`.

This table gives a *rough guide* to converting resistance to lux *for that LDR*.

Resistance	Lux
3000	300
6000	100
10000	44.5
30000	7.8
60000	2.6
100000	1.2

This can be calculated in Python by
`300 * (3.0**-log(resistance/3000.0, 2))`.

The ** is Python's power operator.

Mu can plot your data!

One of the great features of the *mu* editor is that it can plot data that's output from your programs.

All you have to do is print the data you want to plot in the right format, and ask *mu* to plot it!

The first thing you need to do is to format the data in the way that *mu* expects.

Mu plots Python tuples, which you met earlier.

To print a tuple, you just need to use an extra pair of round brackets in your `print` statement.

Here's a short terminal session that shows the difference:

```
>>> print(1, 2)
1 2
>>> print((1, 2))
(1, 2)
```

To plot the data from the mini-weather station, you need to

1. Use the same wiring as the weather experiment in the previous chapter
2. Loop forever. In the loop:
   a) Read the two analogue values
   b) Convert the two voltages to a temperature and a light level
   c) Print a tuple consisting of the temperature and light level
   d) Wait until it's time to take the next reading
3. Run the program in *mu*
4. Tell *mu* to plot the data.

```
import explorerhat as eh
from time import sleep
from math import log

def lux(resistance):
    return 300 * (3.0**-log(resistance/3000.0, 2))

r1 = 10000
delay = 1 # for testing
# delay = 600 # every 10 minutes, for production use!
```

```
11
12 while True:
13     v1 = eh.analog.one.read()
14     celsius = 100.0 * (v1 - 0.5)
15     v2 = eh.analog.two.read()
16     ldr = r1 * v2 / (5 - v2)
17     level = lux(ldr)
18     print((celsius, level))
19     sleep(delay)
```

Enter the program using `mu`, save it as `weather-plot.py`, then run and stop it.

The first three lines import the libraries you'll use. The `log` function from the `math` library is needed for the calculation in the `lux` function.

Lines 5 and 6 define the `lux` function, which does the calculation you saw above. It turns an LDR resistance into a light level.

Lines 8 and 9 set up a couple of values you will need later.
`r1` is the value in ohms of the fixed resistance that's connected to the LDR.
`delay` is the time to wait between readings (in seconds). For testing, a value of 1 second is fine, but for a real weather station you might want to set it to 600 (10 minutes).

Line 10 is commented out. To set the delay to 10 minutes for slower updates to the plot, uncomment line 10 (remove the #) and comment out line 9 instead (add a # at the beginning of it).

You should always make sure there's a delay between printing values if you want to plot them. If you don't, the plotting code in *mu* will get overwhelmed by the flood of data and things won't work the way you want.

Lines 12-19 contain the main `while` loop.

Lines 13 and 14 read the first analogue voltage and convert it into a temperature.

Lines 15 to 17 read the second voltage, convert it to the resistance of the LDR, and convert that to a light level.

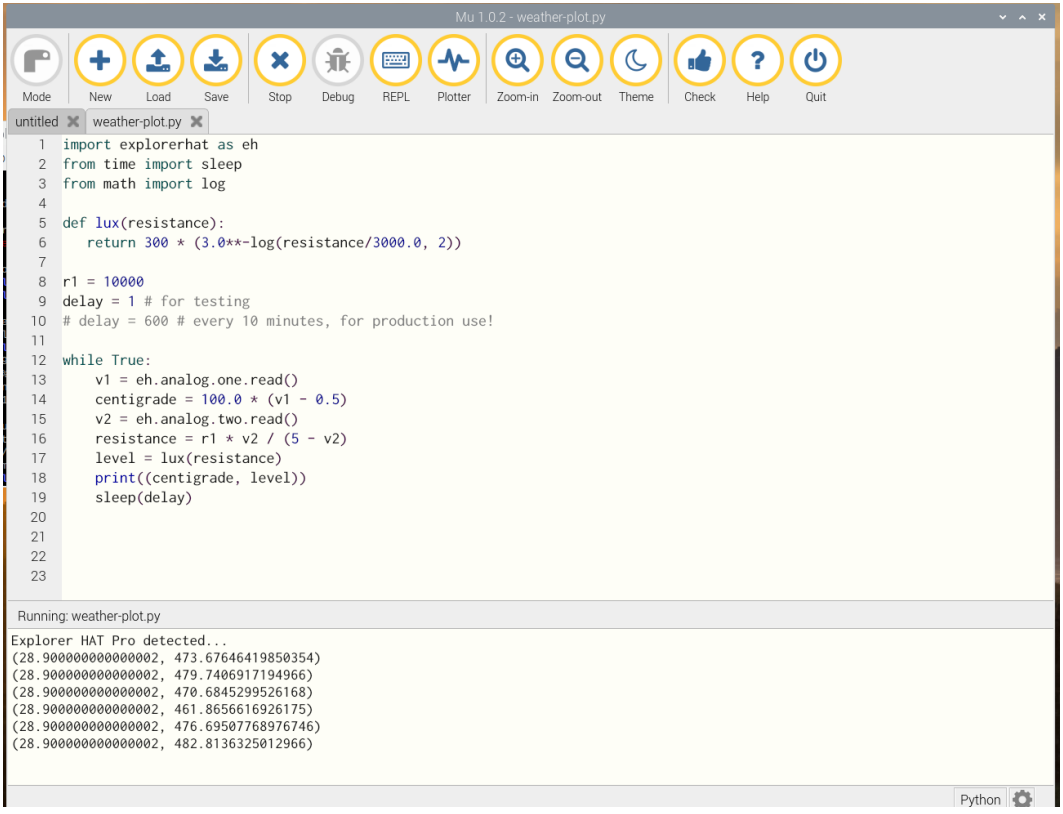
Line 18 prints the tuple, and line 19 waits until it's time to take the next reading.

So far you've run programs using the terminal window, but you need to run this one from *mu* itself to make the plotting work.

With the program open in *mu*, click *mu*'s Run button.

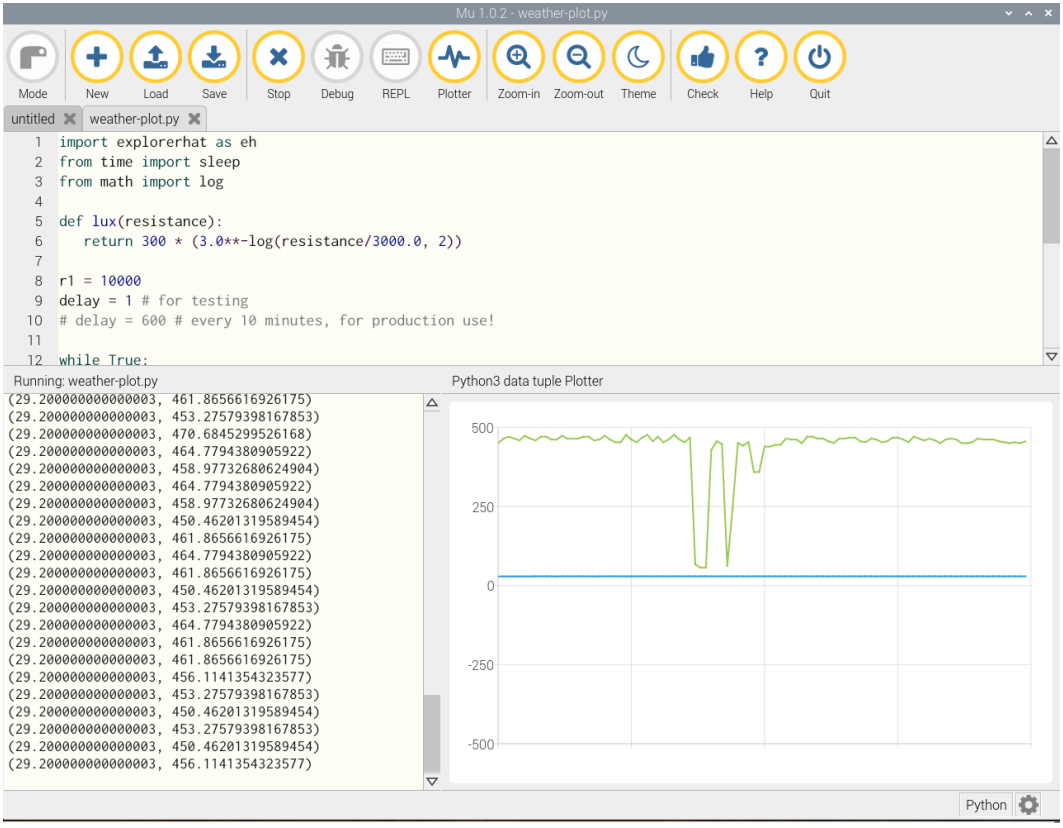
You should see tuples printed once a second. If you shade the LDR or touch the TMP36 you should see the printed values changing.

The mu window should look like this:



Tuples

If you now click *mu*'s Plot button, you should see a plot like this:



Plot 1

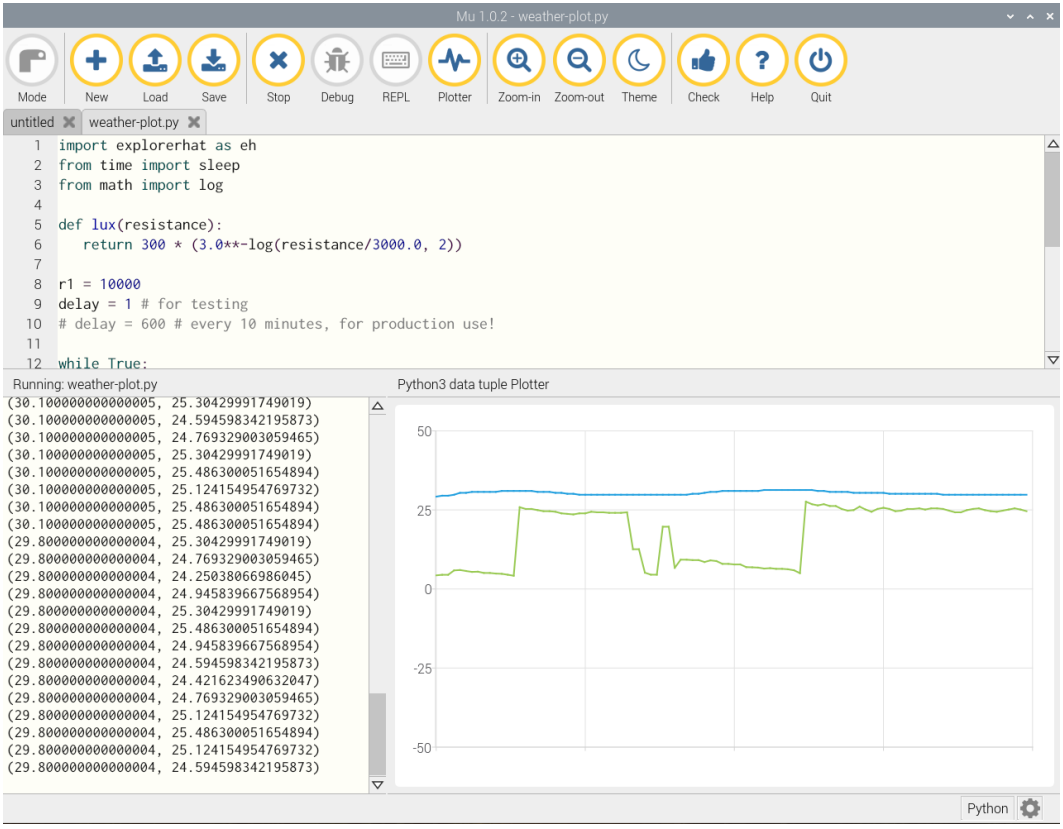
Click on `Stop` to stop the program.

There's just one problem. The light level is around 500, but the temperature is around 29 degrees.

The plot would be easier to read if the two values were similar. You can make one small change to the program to fix that.

Edit line 17 so that it reads `level = 0.04 * lux(ldr)`

Save the file and run it again. Now the plot should look like this:



Plot 2

Mu's plotting capability is really cool!

Summary

You've covered a lot in this chapter. You've looked at

1. How to work out the level of light falling on an LDR
2. How to print tuples in Python, so you can plot them
3. How to plot values using *mu*.

Next

The next chapter covers another really useful technique.
You'll learn how to wire up push-buttons and write Python code that can detect when a button is pushed.

Getting pushy

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

What's this about?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Wiring up a button

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

How *not* to do it!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Use a resistor

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Wiring up the button

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Writing the code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

What we haven't done - debouncing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Next

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Now you're motoring

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

What's this about?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

The motor you'll use

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Controlling motor direction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Next

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Where next?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Support for readers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Next, two favours

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Resources for the future

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/explorerhattricks.

Appendix A

Resistor colour codes

You can read about colour codes on electronic components on Wikipedia.

There's another really useful web page called resistors which you can use to convert resistor values to colour codes.

Wiring colours

I have an additional convention if I'm using components that are connected using something called I2C. Don't worry about that if it means nothing to you - I may add a bonus chapter about I2C when the main book is finished! If you're already familiar with I2C, my convention is to use green for the `SDA` connections and blue for the `SCL` connections.