Welcome to **Week 2**

Writing your first Python code

In this activity you'll write you first code using the Python 3 shell. Then you'll write a simple program using inputs, outputs and variables to create a reaction game. There's also a hacker level challenge to take this further.

1.11 Getting started with Python

1.12 Simple reaction game using inputs, outputs and variables

1.13 Applied Activity: Working with loops article

1.14 Testing your understanding of Python quiz

1.15 Discussion of quiz questions

Week 2

Getting started with Python

Computers need instructions to be of any use. Without instructions, they are just expensive (or in the Raspberry Pi's case, inexpensive) lumps of plastic and metal. We can write these instructions using a programming language. One of the easiest text-based languages to learn is Python. And just because Python is easy to learn, don't assume it isn't powerful.

On the Raspbian desktop you should see a Raspberry button in the top left. Click on this to see the different types of applications that come pre-installed on Raspbian. Next click on `Programming`, and then on `Python 3 (IDLE)`.

A new window should open up on your desktop. This program is called an IDE, which stands for Interactive Development Environment. This particular IDE is called IDLE.

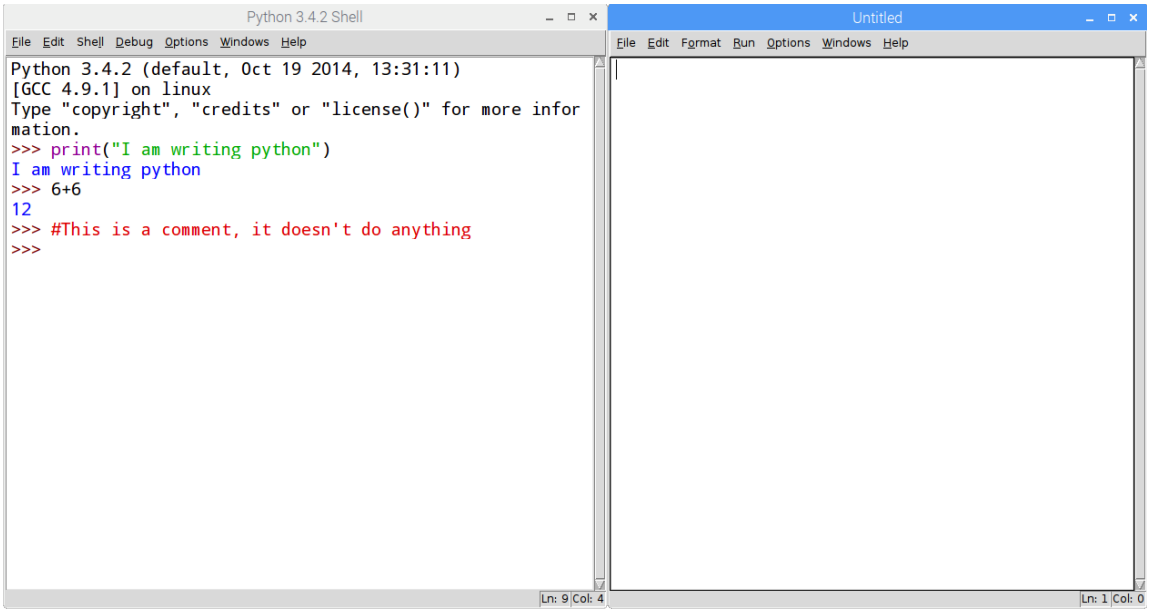
The window that has opened is known as a **shell**. In the shell, you can type Python code, and it will be executed straight away. Have a go at typing the following lines into the shell. Don't worry about the three chevron symbols (`>>>`), as these will be added for you by IDLE.

```
>>> print('I am writing Python')
>>> 6+6
>>> #This is just a comment, it doesn't do anything
```

Use the **comments** section below if you need help writing your first Python code.

The shell is useful for testing a few lines of code, or interacting with programs you have written, but to actually write a computer program, you need to write code into a file.

Click on `File` and `New File`, and a second window will open. It's useful to position them side-by-side like this:



Simple reaction game using inputs, outputs and variables

We'll start with a very simple program that involves **inputs**, **outputs** and **variables**.

Python has many built in **functions** that we can use to build programs, to begin with let's use the *print* and *input* functions.

```
print('Quick, the Enter key')
input()
print('That was fast')
```

Save your file (`File` > `Save As...`) and call it something sensible, like `game.py`. (The `.py` suffix is used for Python files.) To run the file you can just press `F5` on your keyboard.

The `print` function is a type of output. You are instructing the computer to display some text on the screen.

The `input` function takes input from the user's keyboard presses.

Let's make the program a little more interesting.

We could make the program pause for a little bit before it starts the game. Handling time can be a little tricky, but luckily for us the code to interface with the computer's clock has been written for us and packaged up into a nice little module that we can use, called `time`. To start with we only need one function from the `time` module. It's called the `sleep` function. This function isn't available by default, so we'll have to import it into our program.

```
from time import sleep
sleep(5)
print('Quick, press the Enter key')
input()
print('That was fast')
```

Here you have imported the `sleep` function from the `time` module and then instructed the program to pause for five seconds.

Save and run your code to have a play with your new game.

It would be great to know how quick the user was between seeing the message and pressing the key. Again, the `time` module can be used for this, and in particular a function called `time` (confusingly!). We can add this function to our import on the first line, like this:

```
from time import sleep, time
```

The `time` function will get the number of seconds that have elapsed since January 1st, 1970. By storing this value in a variable when the message is displayed, and then checking the time again after the user has pressed a key and subtracting, we can work out their reaction time. Here, the reaction time value is stored in the variable `reaction_time`. You can't have spaces in variable names, which is why we often use underscores.

```
from time import sleep, time
sleep(5)
start = time()
print('Quick, hit the Enter key')
input()
reaction_time = time() - start
```

```
print('You took', reaction_time, 'seconds')
```

Save and run your code then have a play. How fast can you hit that Enter key?

Note: The name of a function in Python is always followed by parentheses. In many cases, like `sleep(5)`, the parentheses can contain more information, such as how long to sleep. When we call the `time` function we don't need to add any information, so we write `time()`.

Applied Activity: Working with loops

A core concept in programming is the ability to **repeat** instructions, either a fixed (finite) number of times, or over and over until some condition is met, possibly forever (infinite). These *loops* are powerful tools to master, and are common in all but the simplest programs.

In Python, loops are defined using two different structures - `while` loops are potentially infinite, and `for` loops repeat a specific number of times. Below you can see examples of both.

```
while True:
    print('Hello')
    input()
```

This is a **while** loop which will repeat the indented lines of code until the condition is no longer true. In this case the condition is just the value *True* which will always be *True*, so this loop can never exit while the program is running. Other conditions could be used which might cause the loop to exit if they were not met. For example, `while lives >0:` would cause a loop to repeat as long as the variable `lives` was greater than 0.

```
for i in range(10):
    print('Hello')
    print(i)
    input()
```

This **for** loop is a finite loop and will loop a fixed number of times. For each value from 0 up to (but not including) 10, it will assign the value to the variable *i* and carry out the indented lines.

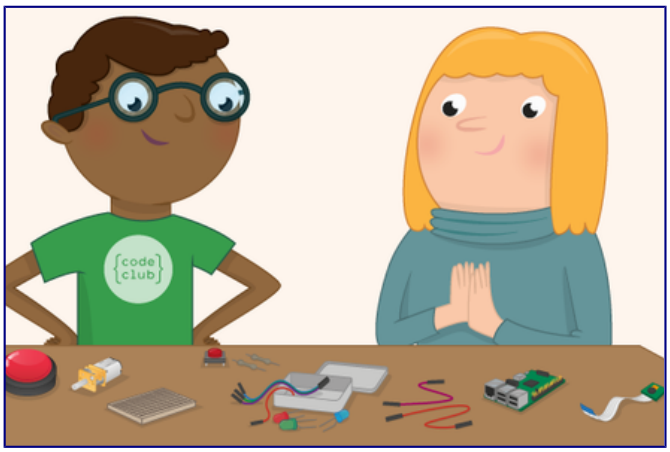
Notice that in both loops, a colon (`:`) is added to the end of the line, and four spaces are then placed before the lines of code that are to be repeated.

Try both of the mini-programs out to see what the difference is between the two loops.

HINT: If you need to stop your program running you can press **Ctrl + C** to stop it.

- Can you use loops in your game to make it repeat?
- Can you use variables to store the total reaction time after several games?
- Can you work out the average reaction time?

Week 2: Let's get physical



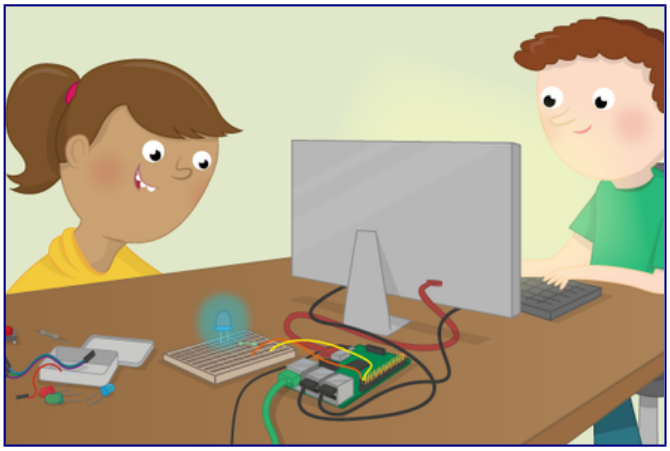
What is physical computing?

Before you delve into controlling the physical world with your Raspberry Pi, let's learn some basics of physical computing and recap last week's activity.

2.1 Week 2 welcome

2.2 General Purpose Input Output on the Raspberry Pi

2.3 What can I do with the GPIO pins?



Programming your first physical output

In this activity you will connect an LED to you Raspberry Pi to create a simple circuit and then control the LED from some Python code.

2.4 Some circuit theory

2.5 Building a simple circuit

2.6 Blinky blinky lights

2.7 Applied Activity : Lights, sequences and coded messages

2.8 Physical Computing Quiz

2.9 Discussion

Week 2 welcome

In Week 1, you learned how to set up the Raspberry Pi. You also looked at how the Raspberry Pi can facilitate learning Physical Computing and much more.

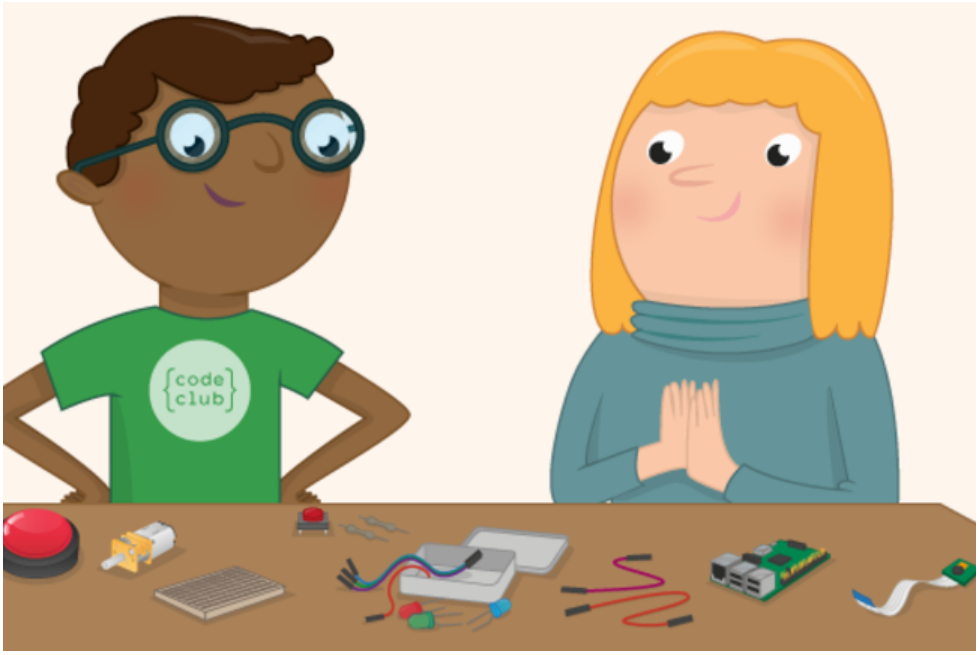
This week you wrote your first few lines of code in Python and you'll be getting physical with the Raspberry Pi, using it to control a simple circuit that you can build quickly.

Take in mind that teaching others reinforces what you're learning, and you might pick up some new information from them too!

This week you will:

- Learn about the GPIO pins on the Raspberry Pi and how they can be connected to real world devices.
- Build a circuit featuring an LED and switch it on and off using some Python code.
- Create light sequences using multiple LEDs or broadcast encoded messages using LED flashes.

Do you have previous experience of using or teaching simple electronics? If so, please tell us about it in the **comments** below.



General Purpose Input Output on the Raspberry Pi

The GPIO pins are one way in which the Raspberry Pi can control and monitor the outside world by being connected to electronic circuits.

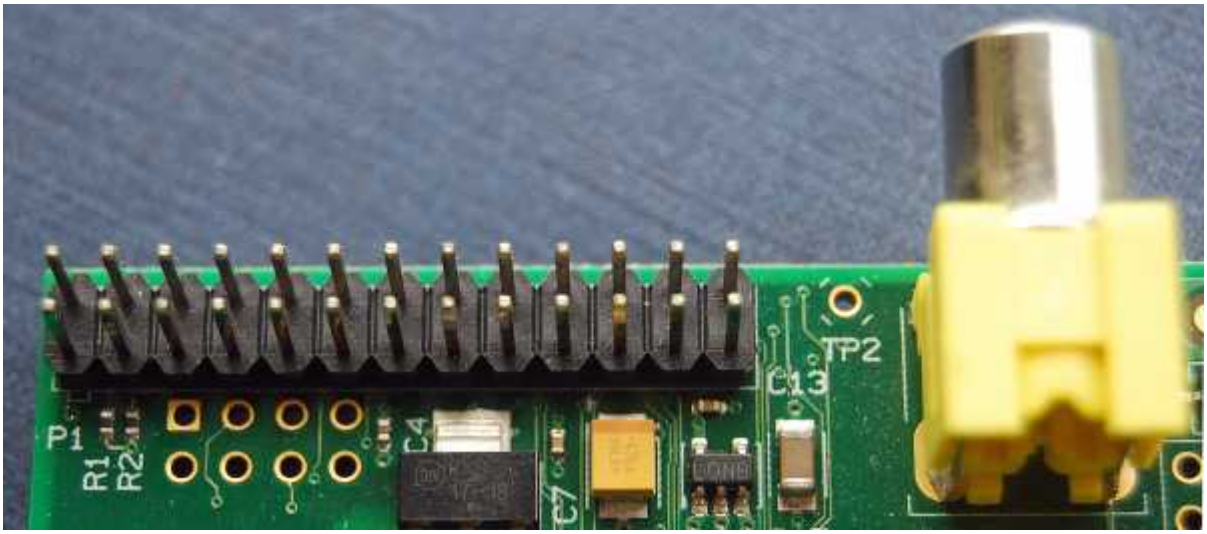
The Pi can control LEDs, turning them on or off, drive motors, and interact with many other objects. It can also detect the pressing of a switch, change in temperature, or light, etc, by attaching kinds of sensors. We refer to all these activities, and more, as **physical computing**.

GPIO

Most models of the Raspberry Pi have 40 pins that look like this:

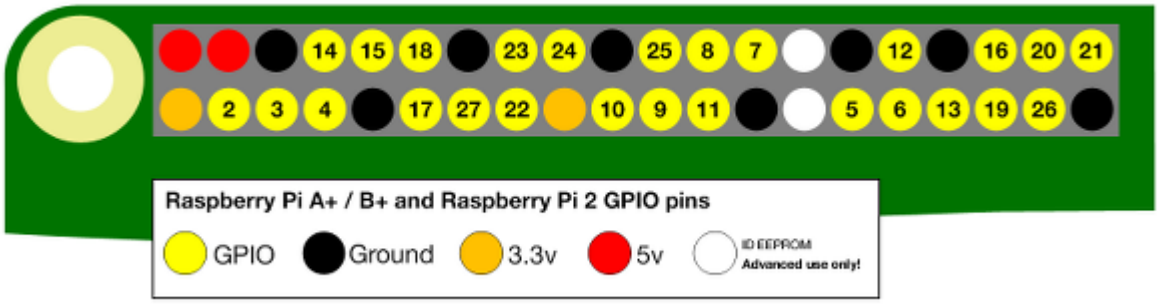
These pins are a physical interface between the Raspberry Pi and the outside world. Using them, you can program the Raspberry Pi to switch devices on and off (output), or receive data from sensors and switches (input). Of the 40 pins, 26 are GPIO pins and the others are power or ground pins (plus two ID EEPROM pins which you should not play with unless you know your stuff!)

Early models A and B have only 26 pins, and look like this:



GPIO Pin Numbering

When programming the GPIO pins there are two different ways to refer to them: **GPIO numbering** and **physical numbering**. Throughout this course (and in all our resources) we will refer to the pins using the GPIO numbering scheme. These are the GPIO pins as the computer sees them. The numbers don't make any sense to humans, they jump about all over the place, so there is no easy way to remember them. However, you can use a printed reference, a reference board that fits over the pins or a website guide to help you out.



What can I do with the GPIO pins? What are they for? What can I do with them?

You can program the pins to interact in amazing ways with the real world.

The output can be anything, from turning on an LED to sending a signal or data to another device.

Inputs could be anything from a simple button to a sensor or a signal from another computer or device.

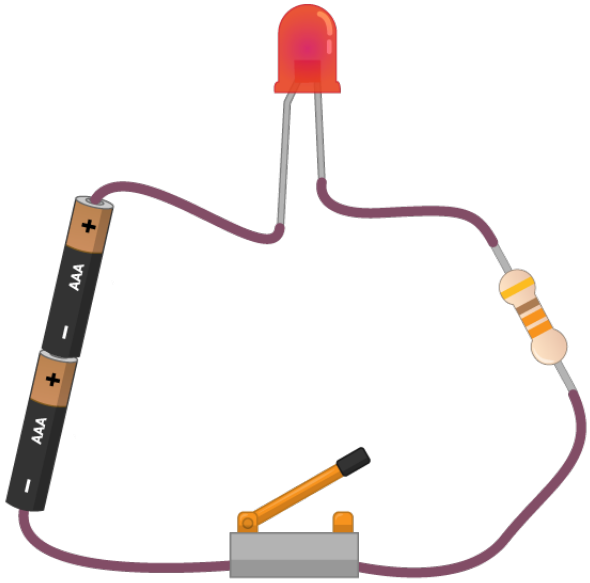
If the Raspberry Pi is connected to the internet, you can control devices that are attached to it from almost anywhere, and those devices can send data back. Connectivity and control of physical devices over the internet is a powerful and exciting thing, and the Raspberry Pi is ideal for this.

There are lots of brilliant examples of physical computing in our resources.

Now that you're familiar with the GPIO pins of the Raspberry Pi, let's build a simple circuit and program it.

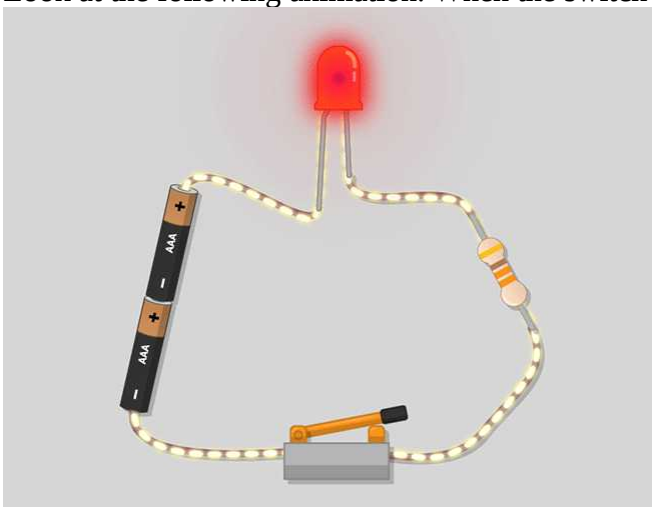
Some circuit theory

The following image shows an extremely simple circuit. It consists of four basic components, all connected up with wires.



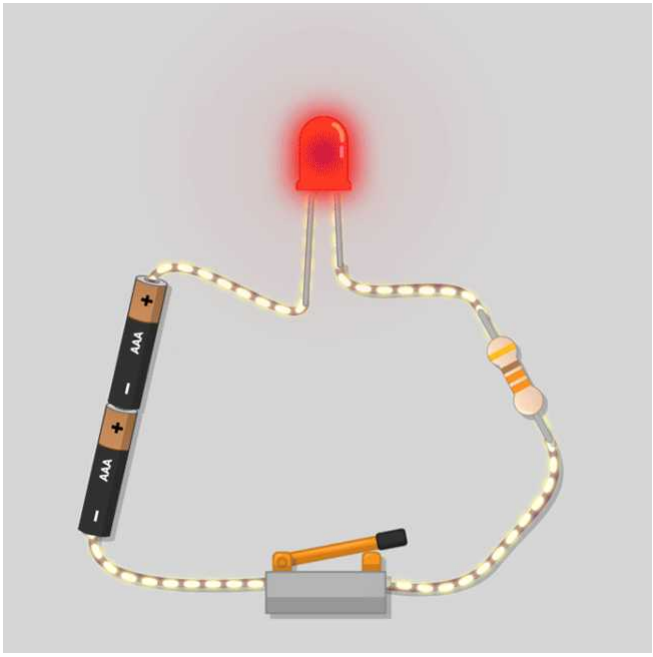
1. The cell (batteries) provides energy to the circuit in the form of electricity. A cell has a positive and a negative side. Electric current flows from the positive side of the cell to the negative side of the cell.
2. The Light Emitting Diode (LED) is a type of **Output Component**. When current flows through the LED, it emits light. Different LEDs can produce many different colors of light, and some can even produce multiple colors.
3. Resistors use some of the energy from the cell, reducing the amount of energy left for the LED. Without the resistor, the LED could receive too much energy for it to cope with, causing it to break and thus stop the current.
4. The switch acts as a break in the circuit. When the switch is *open* then no current can flow through the LED or the resistor. When the switch is closed, the circuit is complete and current can flow, causing the LED to switch on.

Look at the following animation. When the switch is closed, current begins flowing through the circuit.



Correct LED circuit

Now look at this animation showing the LED being swapped round. You should notice that when the LED is placed into the circuit the wrong way around, current ceases to flow, and the LED no longer lights up. The clue is in the name *Light Emitting Diode*. A *diode* is a component that only lets current flow through it in one direction, a bit like a valve in a water pipe. You'll need to remember this when you set up your own circuit.

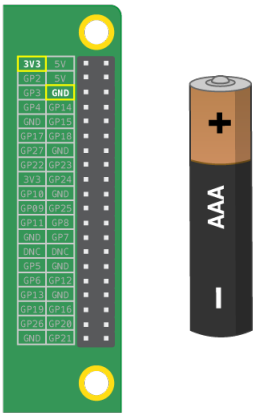


Building a simple circuit

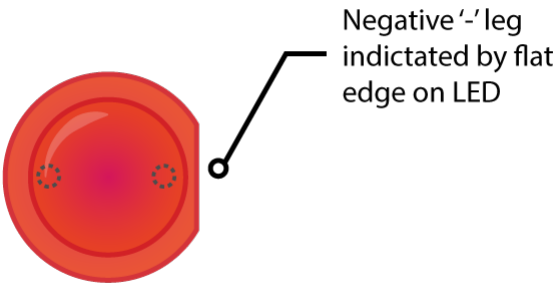
To build the circuit you just learned about, in this next step, you're going to need an LED, a 330 ohm resistor, and two female-to-male jumper leads.

The first thing to do is to have a look at the GPIO pins on the Raspberry Pi. The two pins we are going to use to begin with are labelled on this diagram. They are called 3V3 and GND.

- 3V3 means 3.3 **volts**. You can think of this as the amount of energy that is available to be provided to the circuit. This pin acts a bit like the positive side of a standard battery.
- GND means Ground. For a circuit to be complete, electric current must always be able to flow to a ground pin. This is like the negative side of a standard battery.



- Take one of your female-to-male header leads and connect the female end of it to the 3V3 pin on the Raspberry Pi, an other female-to-male header leads to the ground pin.
- Now have a good look at your LED. This is not a symmetrical component. You should see that one leg is longer than the other. The long leg is sometimes called the **anode**, and this leg should always be connected to the positive side of a circuit. One way to remember this is to imagine the longer leg as having had something added and the shorter leg has had something taken away. Sometimes the LEDs might have legs the same length, in which case you can tell which side is the anode because the plastic rim of the LED will be round, except near the negative leg (called the **cathode**) where it is slightly flattened.



- Find the long leg of the LED
- Connect the resistor to the LED (on the right side, look at picture "Correct LED circuit")
- The last step is to connect all your components both to the 3V3 pin and to the ground pin. Make sure that your Raspberry Pi is powered on and then take the female end of the jumper lead and plug it into your ground pin.
- If your LED doesn't light, then try the following things:
    1. Check your Raspberry Pi is on.
    2. Check all your components are firmly in the breadboard.
    3. Check your LED is the right way around.
    4. Try another LED.

What was the point?

Testing is hugely important when working with physical hardware for a few reasons.

- Using small components like this requires a certain level of dexterity. Testing gives you experience, and for some will be a challenge in itself.
- Testing gives you an early success that they can be pleased with, motivating you to move on to controlling the LED with code.
- You're introducing another layer of complexity and another area where things could go wrong. By testing, you know that your LED works, and if you encounter problems later you should be able to eliminate the LED as the cause.
- Once you know how to test, you are equipped to solve problems in your projects by yourself.

But what about the switch?

We mentioned earlier that the Raspberry Pi could act like both the cell and the switch, but in your circuit the pin we're using (3.3V) is always on. However, other pins can be switched on and off. By making a subtle change to your circuit and writing some code, you can control the behavior of your LED.

The first thing to do is to remove the female-to-male jumper lead from the 3V3 pin. This pin provides 3.3 volts all the time; you can't write code to control it. However, there are plenty of GPIO pins on the Raspberry Pi that *can* be controlled. For the purposes of this exercise, we're going to use pin 17, but you can choose any numbered pin. Connect the female-to-male jumper lead to your pin, as shown in the diagram.

You should see that your LED has turned off. This is because at the moment, pin 17 is in an **off** or **low** state. To turn on the LED, we'll need to set pin 17 to be **on** or **high**. When the pin is in a high state, it will provide 3.3 volts to the circuit. To change the state of the pin, you're going to need a few lines of Python code, which you'll write in the next step.

Blinky blinky lights

1. Just like last week, we're going to use the 'development environment' **IDLE**, to write and run Python code. Open IDLE by clicking on `Menu` > `Programming` > `Python 3 (IDLE)` on your Raspberry Pi.
2. As before, you're going to need an empty file to start writing your code. Click on `File` > `New File` to open a new window. Save this straight away (`File` > `Save As..`) and call it `blink.py`.
3. Last week you used the `time` module to help write a simple program, and you imported the `sleep` function. This week you're going to use a module called `gpiozero`. This module gives you access to the GPIO pins on the Raspberry Pi. You don't need to use the whole of the `gpiozero` library, just the bit that allows you to control LEDs. Inside the library are loads of blueprints for objects you can control; these blueprints are called *classes*. You want to import the *LED* blueprint from the *gpiozero* library. At the top of your file write:

   ```
   from gpiozero import LED
   ```

4. The next step is to create a new **object** for the specific component and pin we are using. Here we've called the object `red_led`, but the name you give it doesn't really matter. This object can then receive different commands to control its behaviour.

   ```
   red_led = LED(17)
   ```

   (This assumes you have connected the LED to pin 17 on your Raspberry Pi.)
5. Save and run your program. You can do this by clicking on `File` > `Save` and then `Run` > `Run Module`, or by using the shortcut keys - `ctrl` + `S` and then `F5`.
6. Nothing will happen right away. Click into the shell (the first window that opened in IDLE) and you can now pass some different commands to your `red_led` object. Try the following two commands.

   ```
   >>> red_led.on()
   >>> red_led.off()
   ```

7. Your LED should turn on and off each time your type the lines. Let's try and automate that a little. Back in the `blink.py`, file you can import the `time` module, and specifically the `sleep` function. Alter you code so it looks like this:

```
from gpiozero import LED
from time import sleep
red_led = LED(17)
```

8. Now you can instruct your program to repeatedly turn the LED on and off again, by using an infinite loop.

```
while True:
  red_led.on()
  sleep(1)
  red_led.off()
  sleep(1)
```

9. Save and run your program, and see what happens. What does changing the amount of time passed to the `sleep` function do? What's the smallest `sleep` time you can use?

A little bit of abstraction

The `gpiozero` module has some built in **methods** to give you additional control of your LED. These are **abstractions**. Complexity has been hidden away, making your code easier to write and more readable.

Either remove your `while True:` loop or copy and past your code into a new file so that it looks like this:

```
from gpiozero import LED
from time import sleep
red_led = LED(17)
```

Now you can try adding the following code snippets to your program. Try them one at a time and play with the numbers to see what they do.

Snippet 1

```
red_led.blink(0.1, 0.2)
```

Snippet 2

```
red_led.blink(0.2, 0.1, 5)
```

Snippet 3

```
red_led.toggle()
sleep(1)
print(red_led.is_lit)
red_led.toggle()
sleep(1)
print(red_led.is_lit)
```

Applied Activity : Lights, sequences and coded messages

Now that you can control a single LED with code, there's lots more you can do.

Here are a few ideas of extra challenges you could set yourself and use with students:

1. Experiment with the frequency of the LED flashes. What's the fastest you can make it flash? Can you make it flash randomly?
2. Can you create short (dot) flashes and long (dash) flashes, giving you the basics of Morse code? Can you use that to broadcast a message?
3. Can you add extra LEDs to your breadboard and control them with other GPIO pins?
4. With multiple LEDs, you could write code to create a simple light sequence. This could be something functional like a traffic light sequence, or something fun like some blinky disco lights.