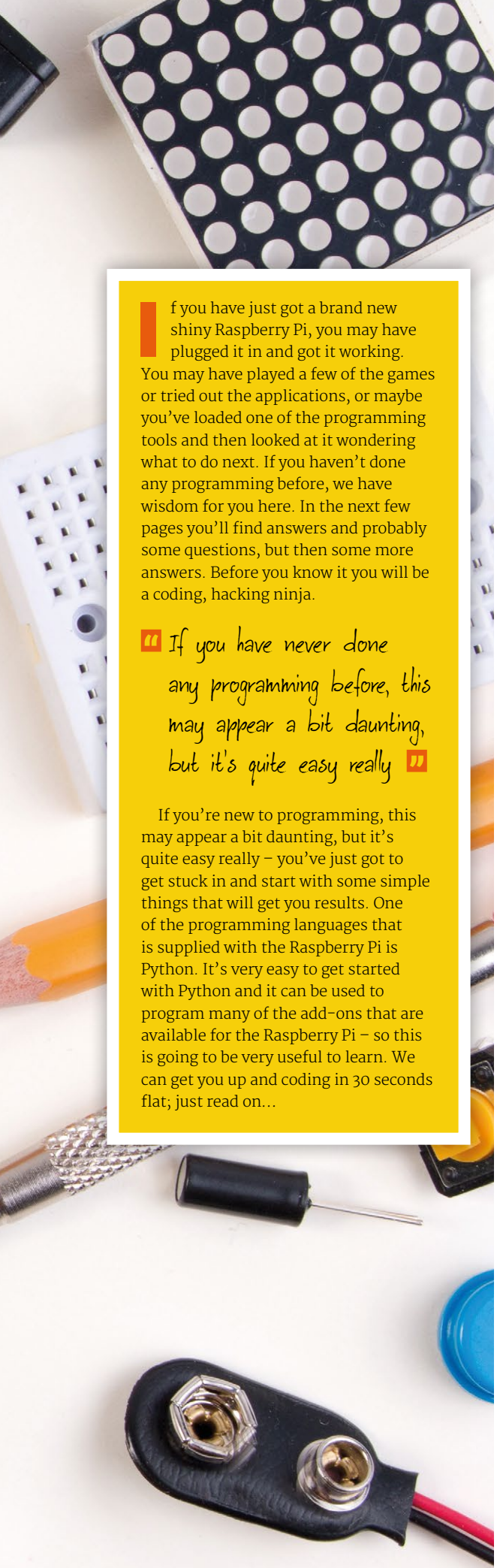
MAKE WITH CODE

If you have bought a Raspberry Pi to learn how to code and hack new electronic gizmos, then you have made a great choice. Get started with this guide...



If you have just got a brand new shiny Raspberry Pi, you may have plugged it in and got it working. You may have played a few of the games or tried out the applications, or maybe you've loaded one of the programming tools and then looked at it wondering what to do next. If you haven't done any programming before, we have wisdom for you here. In the next few pages you'll find answers and probably some questions, but then some more answers. Before you know it you will be a coding, hacking ninja.

> "If you have never done any programming before, this may appear a bit daunting, but it's quite easy really"

If you're new to programming, this may appear a bit daunting, but it's quite easy really – you've just got to get stuck in and start with some simple things that will get you results. One of the programming languages that is supplied with the Raspberry Pi is Python. It's very easy to get started with Python and it can be used to program many of the add-ons that are available for the Raspberry Pi – so this is going to be very useful to learn. We can get you up and coding in 30 seconds flat; just read on...

MAKE AND RUN A PROGRAM

Mark Vanstone

Educational software author from the nineties, author of the ArcVenture series, disappeared into the corporate software wasteland. Rescued by the Raspberry Pi!

magpi.cc/YiZnxL
@mindexplorers

To get started with coding is really easy. Coding is just giving the Raspberry Pi an instruction to do something. The only thing you need to know is what language to talk to it in. In this case we are going to talk in Python 3. We will need to write our instructions somewhere so, to start, open a Terminal window – click the icon in the top left of the screen which is a grey box with a blue bar across the top. This opens a black window with a prompt: **pi@raspberrypi:~ $** in green. If you type `python3` and hit **ENTER**, Python starts and you will see the triple chevron prompt: **>>>**. Now type `print("Hello")` and hit **ENTER**. Bosh! Your first program. You have instructed your Raspberry Pi to print the word 'Hello' and, all being well, it has obeyed.

Entering programs like this is not going to be very useful most of the time, so now let's look at an app we can write and save a program with. Go to the desktop menu (click the Raspberry Pi logo in the top left of the screen) and in the Programming section, select Thonny (which should then open by default in its Simple mode). Try typing in the following program in the Thonny editor and save it, then run it by clicking the Run button. The output will be displayed in the Shell frame below the program editor.

mwc1.py

Language: Python
magpi.cc/HNJhhd

```
001. import random
002.
003. correct = False
004. r = random.randint(1,10)
005. c = 0
006. while correct == False:
007.     n = input("Guess my number between 1 and 10: ")
008.     c = c + 1
009.     if int(n) == r:
010.         correct = True
011.     else:
012.         if int(n) > r:
013.             print("Sorry, my number is lower. Try again.")
014.         else:
015.             print("Sorry, my number is higher. Try again")
016.
017. print("Well done. The correct answer was " + str(r) + ".
     You got it in " + str(c) + " tries.")
```

LEARN TO CODE

Coding involves using several elements. Let's take a look at a range of them and how they work

▲ After initialising the data the program needs, we loop round, updating the answer variable until the player guesses all the letters or they run out of tries

In the last program, we get input from the keyboard and output text and numbers by using the `print()` function. We have also used a condition structure in the form of `if` and `else`. Python is very particular about how you indent the code with spaces (four per indent level); this shows that the indented code is inside another structure. In the last program, everything indented after the `while` statement will be part of that loop. Let's take a look at a few more coding techniques.

01 Using lists

We are going to write a Hangman-style game, where we start with a secret word and the player has to guess it, letter by letter. If correct, we show them where that letter appears in the word. They are allowed ten wrong attempts before the game ends. See the **mwc2.py** code to follow along. First, we make a list of words to choose from. A list is defined in Python using square brackets, like: `list = ["a","b","c"]`. We'll call our list `WORDLIST`. In this case we're writing the list name in upper case to show that it is a constant, i.e. It is not going to change throughout the program.

02 Pick a word

We can pick a word from our list using the `random` module. We import the module at the top of our code, then we can use the `random.choice()` function to get a word and store it in a variable: `theWord`. When we call a function that is inside a module, we use a full stop between the module name and the function name. Next, we want to get the player to start guessing what the word is. If we look at the bottom of the code, we can see that we call a function called `startGuessing()`. This is our own function that we need to define.

03 Defining functions

Each time we call a function, the code inside it runs. Sometimes functions have outputs, like our function `updateAnswer()` that returns the variable `result`. One of the rules of Python is that you must define a function before you call it, so we will need to define our `startGuessing()` function near the top of the code. To do so, we write `def` and then the name of the function, followed by brackets and a colon. If we want to pass variables as inputs into a function, we can add them inside the brackets.

04 Getting loopy

Now for the code in our `startGuessing()` function. We set the number of tries and dashes, one for each letter of the secret word, then we go into a loop. The code says: "While the player still has some tries left and the answer we have is not the secret word, run the following code." In our loop, we print the answer we have so far and how

Coding guides

Want to learn more about coding? Check out our guides to Python coding and object-orientated programming in *The MagPi* #53 and #54 respectively.

magpi.cc/53
magpi.cc/54

mwc2.py

Language: Python
magpi.cc/RqQdhR

```
001. import random
002.
003. WORDLIST = ["orange", "table", "january", "balloon",
     "mouse", "speaker", "lorry"]
004. theWord = random.choice(WORDLIST)
005.
006. def startGuessing():
007.     triesLeft = 10
008.     answer = "-" * len(theWord)
009.
010.     while triesLeft > -1 and not answer == theWord:
011.         print("\n" + answer)
012.         print(str(triesLeft) + " tries left")
013.         guess = input("Guess a letter:")
014.         if len(guess) != 1:
015.             print("Just guess one letter at a time.")
016.         elif guess in theWord:
017.             print("Yes that letter is in the word.")
018.             answer = updateAnswer(theWord, answer, guess)
019.         else:
020.             print("Sorry, that letter is not in the word.")
021.             triesLeft -= 1
022.
023.     if triesLeft < 0:
024.         print("Sorry, you have run out of tries. The word was: " + theWord)
025.     else:
026.         print("Well done, You guessed right. The word was: " + theWord)
027.
028. def updateAnswer(word, ans, guess):
029.     result = ""
030.     for i in range(len(word)):
031.         if word[i] == guess:
032.             result = result + guess
033.         else:
034.             result = result + ans[i]
035.     return result
036.
037. print("I'm thinking of a word....")
038. startGuessing()
```

Module
A module is another code file. It can contain functions, variables, and data.

List
A list is a collection of variables, in this case string variables.

Loop
A loop is a part of the code that is repeated one or more times, usually depending on an equation being True.

Input
An input is data that a program receives, in this case from the keyboard.

Variable
A variable is a container for data, in this case a string which can change as the program runs.

Function
A function contains code that can be called from elsewhere in the program.

Output
Output is anything that the program produces, in this case words printed to the shell window.

Condition
A condition branches a program to execute one set of code or another. Sometimes there are several branches.

Calling a function
When a function is called, the code inside the function is run and then returns to the next line of code after the function call.

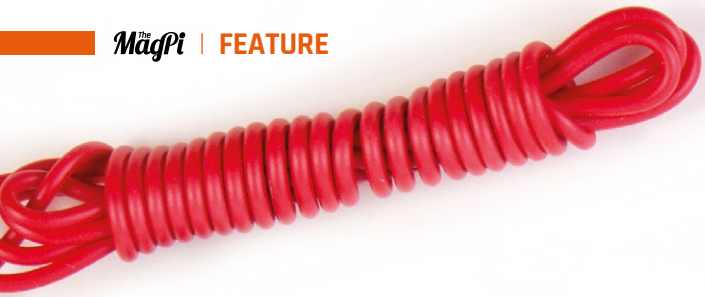
many tries the player has left. Then we use an `if`, `elif`, `else` condition structure to respond to the player, depending on what they typed.

05 Changing the answer

If the player guesses a letter correctly, we call another function: `updateAnswer()`. This uses a `for` loop to add the correct letters into our answer variable, then return that string (a variable containing letters/characters rather than a numeric value). This then becomes the answer variable that we print at the start of each loop in the `startGuessing()` function. When the player has guessed all the right letters in our word or they have got it wrong ten times, the program will drop out of the loop to reach the final part of the function.

06 Win or lose

We have an `if` and `else` structure to print congratulations, or let the player know they've run out of tries. A few of the functions are used with variables: `len()` finds the length of a string, and `str()` converts a number variable into a string so it can be added to the start or the end of another string. After the function is complete, it returns to where it was called, which is the end of the program.



CONTROL THINGS WITH CODE

Now we've got the hang of the coding, let's put it to work with some electronic components

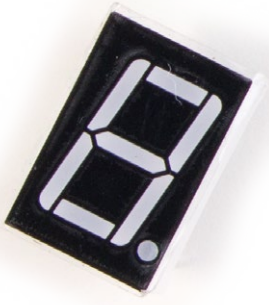
You'll Need

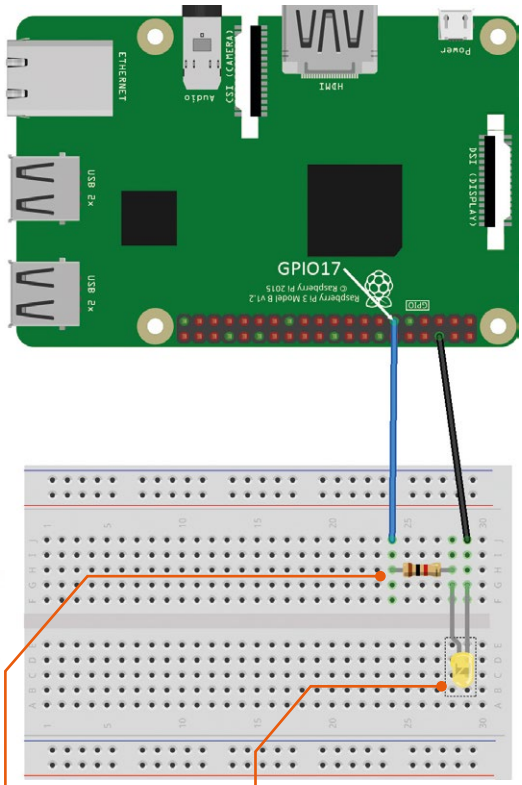
A breadboard
magpi.cc/NtjSiy

An LED
magpi.cc/WBVPxG

A resistor
magpi.cc/iDTFag

2 × male-to-female jumper wires
magpi.cc/OkMyVX

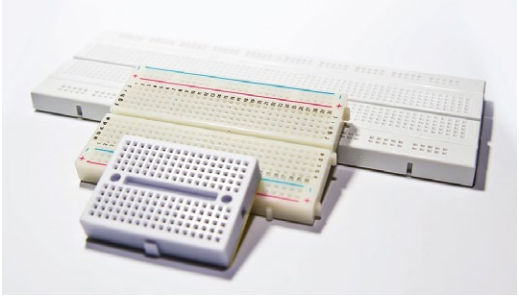




In the previous example, we imported a module into our code and used functions from it. If we want to control electronics from code, there is a very useful module available called `gpiozero`. GPIO stands for 'general-purpose input/output' and the line of double pins on one side of the Raspberry Pi are called GPIO pins. For details about the labels of all the pins, see **pinout.xyz**. If we connect electrical components to these GPIO pins, we can use the `gpiozero` module to make things happen. When we import a module, there are often many different functions inside. We can also make new coding 'objects' with them. Objects are like variables but have their own set of functions and properties inside them that we can call or change, and we do that using the same dot notation (full stop) that we did with the `random` module. For more details on using coding objects (called object-oriented programming or OOP), see issue 54 of *The MagPi*: **magpi.cc/54**.

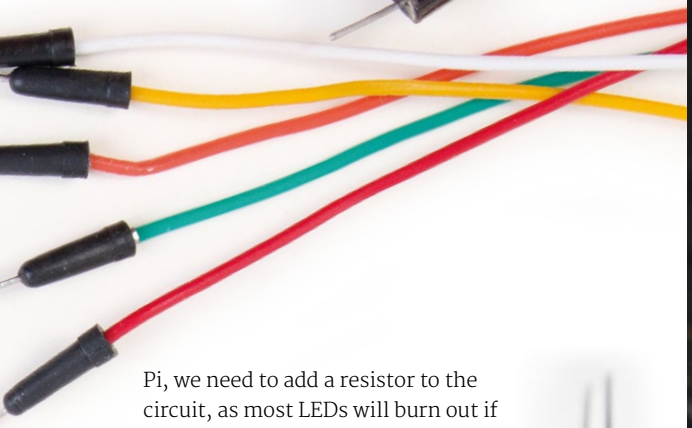
01 The breadboard

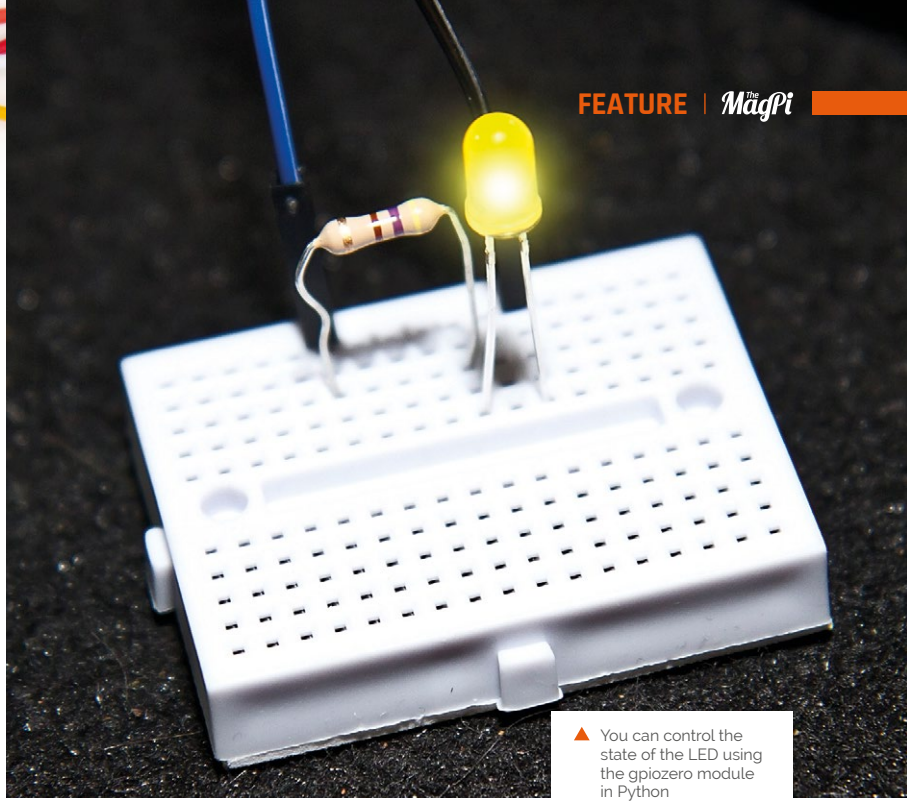
Breadboards come in various sizes, but they all work in the same way. If there are tracks marked red and black/blue (and/or + and −) on the long edges of the board, these are for power and are connected along the length – although sometimes divided into sections. The matrix of holes which make up the main part of the board are connected in lines the other way (vertically in the diagram). There is usually a break in the centre of the board so that the two sides are not connected.



02 Light-emitting diode (LED)

An LED is a bit like a bulb in that if you apply electricity to it in the right way, it lights up. An LED is also a diode, which means that the electricity needs to flow in the correct direction or it will not light. When connecting an LED to the Raspberry





▲ You can control the state of the LED using the gpiozero module in Python

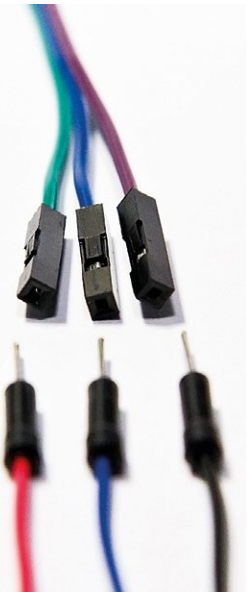
Pi, we need to add a resistor to the circuit, as most LEDs will burn out if we connect them directly to the main power output. In the example, we are using a 1kΩ resistor, but it's fine to use another similar value. To light an LED using `gpiozero`, assemble the components as in the diagram, then write a Python program: start with `from gpiozero import LED`, then create an LED object with `led = LED(17)`, and finally type `led.on()` to light the LED.

03 A resistor

Resistors do what their name suggests: they resist the flow of electricity (current). Some components need to have a certain amount of current in order to operate correctly. Resistors enable us to set the current or voltage to a suitable level for the other components we are using. There are many different types of resistors, with different resistance values. The resistance value can be read from the pattern of coloured stripes on the resistor. You can also get variable resistors, which are known as potentiometers.

04 Jumper wires

We need to connect our components to the Raspberry Pi GPIO pins. For this we use jumper wires. The ones we will be using have a female connector at one end, to go on the GPIO pins, and a male connector to go in the holes of the breadboard. You can also get jumper wires with both male connectors or both female connectors, for different situations. They can be bought in strips all joined together, sometimes known as 'jumper jerky'.

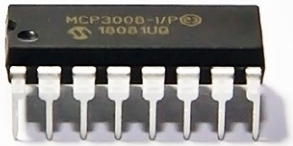


05 Potentiometer

A potentiometer is a variable resistor. It usually has a turning spindle that changes the resistance from one value to another, quite often from no resistance to full resistance (no electricity flowing). A potentiometer can provide us with a variable output voltage which we can measure with the GPIO, but there is a slight problem. The potentiometer provides an 'analogue' output (varies continuously between values) and the GPIO inputs are only digital, i.e. on or off. So we need another component: an analogue-to-digital converter (ADC).

06 Analogue-to-digital converter

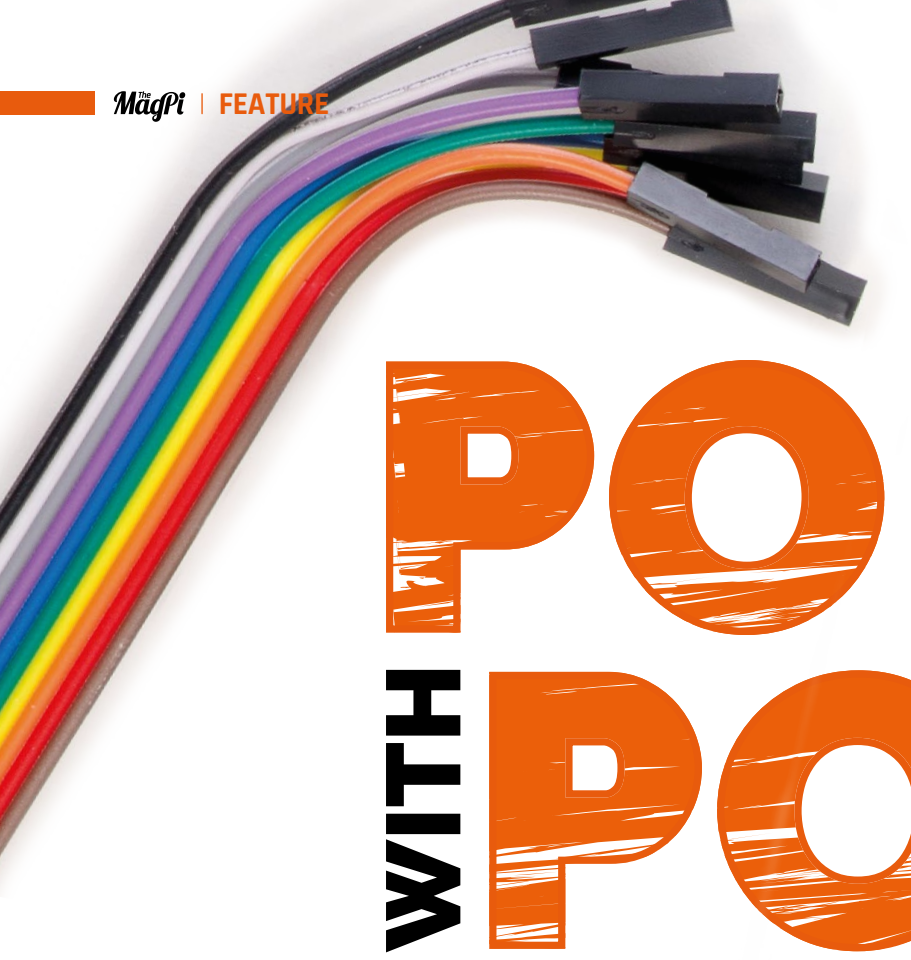
ADC components come in various forms, but the one we have in this example is called an MCP3008. It's an integrated circuit (IC), meaning that it's a box with some circuitry inside it. We don't need to know what is inside it – we just need to know what to connect to each of the legs of the IC. We will need to wire up several of the legs to GPIO pins and provide the IC with power; when we've done that, we can connect the potentiometer to the IC and then read values in showing the position of the potentiometer spindle using the `gpiozero` module. We'll cover the code in the next section.



Electronics guide

For more details about using electronic components with the Raspberry Pi, check out our Electronics Starter Guide in issue 64 of *The MagPi*.

magpi.cc/64



PONG WITH POTS

Now to put what we have covered to the test: we will make a retro game and control it with our own homemade controllers

You'll Need

Breadboard
magpi.cc/vhZkGh

6 × male-to-female jumper wires &
10 × male-to-male jumper wires
magpi.cc/ffZNwL

2 × potentiometers
magpi.cc/oZRFEe

MCP3008 integrated circuit
magpi.cc/aCvXuo

On the previous page we talked about potentiometers and analogue-to-digital converters, and this is where we get to use them. It's a bit more complicated than lighting up an LED, but only a little. First, we are going to write a program which has two rectangular bats at each side of the screen that can be moved up and down by two players. A ball bounces backwards and forwards between the bats until one player misses the ball and the other player scores a point. That's right, you guessed it, the game is Pong and we are going to create a controller for each player from a potentiometer and wire it into the Raspberry Pi.

01 Super-fast game coding

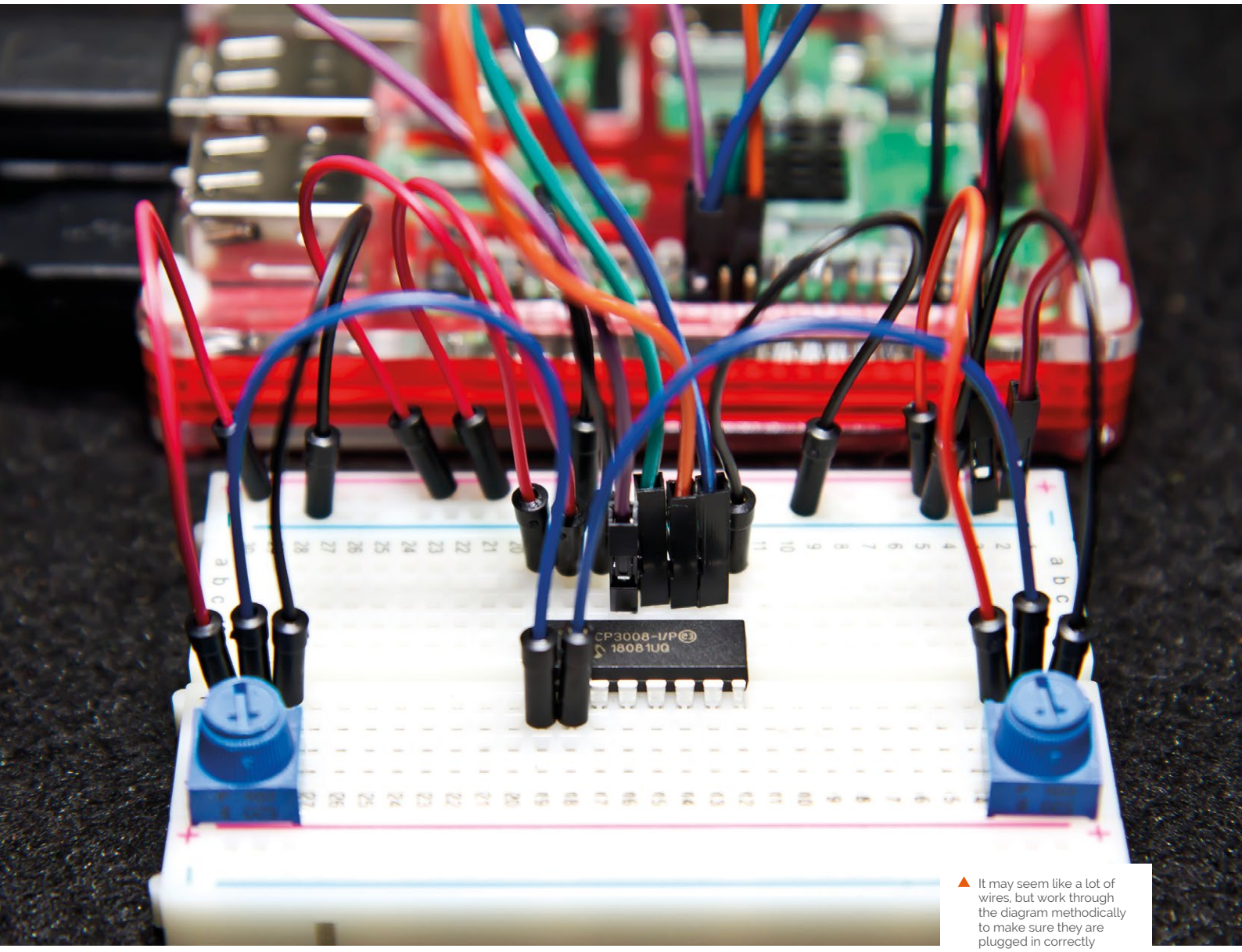
If you have been following other coding articles in recent issues of *The MagPi*, you will know that when writing a quick game on the Raspberry Pi, Pygame Zero is your friend. We can make the basics of the game code very quickly by importing the `pgzrun` module, which holds all the Pygame Zero code. We need to call `pgzrun.go()` at the end of our code, and that's our game window sorted. As with all Pygame Zero programs, we have a `draw()` function to write the graphics to the window, and an `update()` function to update the game items between draw cycles.

02 Running the code

The listing **mwc3.py** provides all the code you need for the game to work. There is some code in the `updatePaddles()` function for the keyboard to control the paddles or bats, just in case you want to test it before making the proper controllers. We import several modules with this code. We have covered `pgzrun`, but we also need `random` so that the ball will move in a random direction each time it starts. In addition, we need `gpiozero` to deal with the input from the controllers, and we need the `math` module for calculating the direction of the ball.

03 Wiring it all up

One thing to bear in mind when connecting any electronics to a computer is that if the wires are connected in the wrong way, you may cause damage to the computer or the electronic



▲ It may seem like a lot of wires, but work through the diagram methodically to make sure they are plugged in correctly

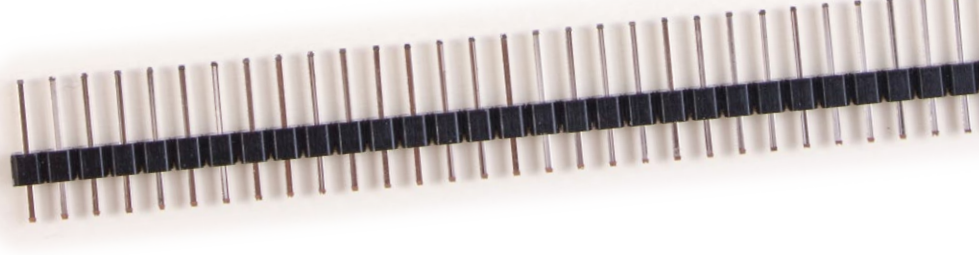
components, so it's always a good idea to power off your Raspberry Pi before connecting anything to the GPIO pins. Follow the wiring diagram (overleaf) carefully, making sure that the jumper wires are connected to the right GPIO pins and to the correct places on the breadboard. Once you have put everything in place, it's a good idea to have another check just to make sure.

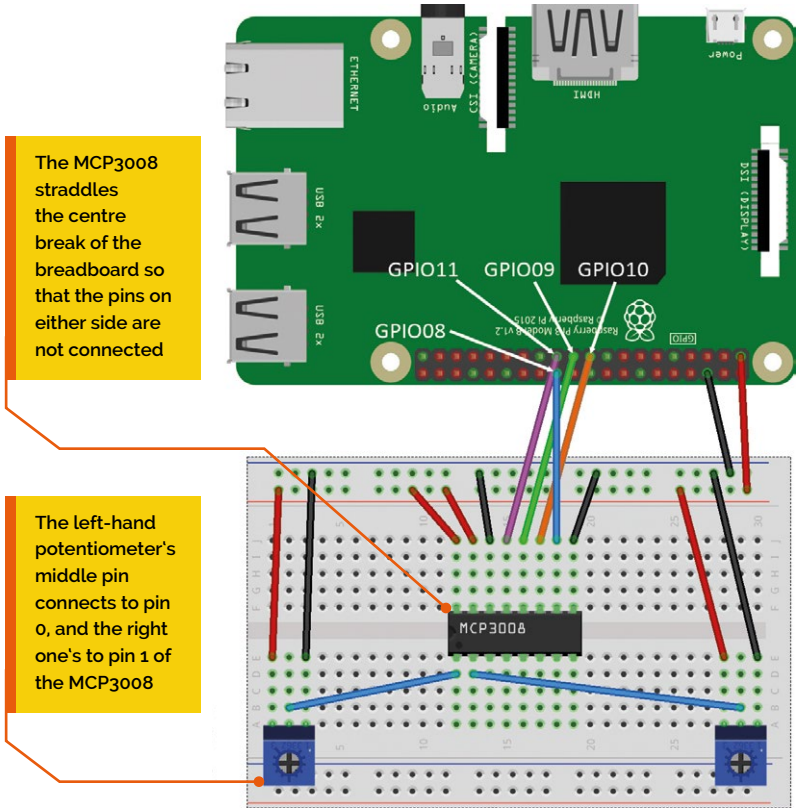
04 The MCP3008 IC

The MCP3008 converts the voltage from our potentiometers into a number with the help of the `gpiozero` module. It has eight channels for input, but we are going to just use two of them in this case. You will see from the diagram that all the legs on the top side of the IC are connected to either GPIO pins or to power lines. There are two red connections going to the positive power track, then a black lead to the negative or ground track. Then there are four coloured wires that go to: purple – GPIO 11; green – GPIO 09; orange – GPIO 10; and blue – GPIO 08. Then there is one last connector to the ground track.

05 The inputs

All the MCP3008 pins on the bottom side of the IC are input channels. We will use the first two pins, which are channel 0 and channel 1. We





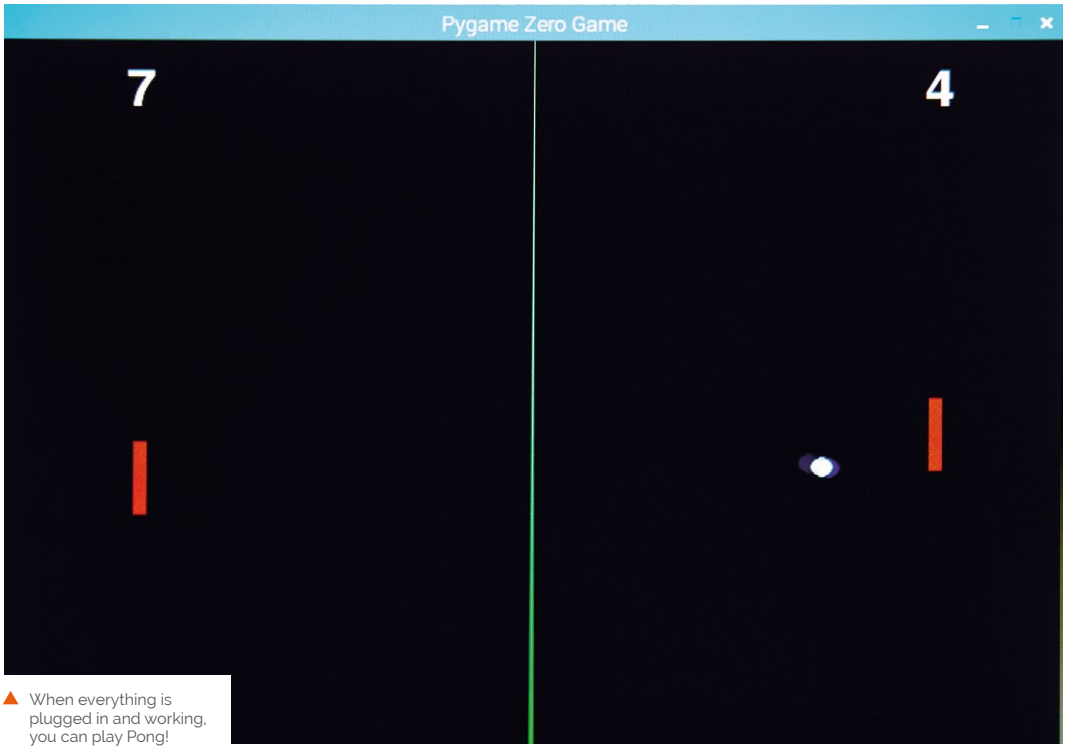
The MCP3008 straddles the centre break of the breadboard so that the pins on either side are not connected

The left-hand potentiometer's middle pin connects to pin 0, and the right one's to pin 1 of the MCP3008

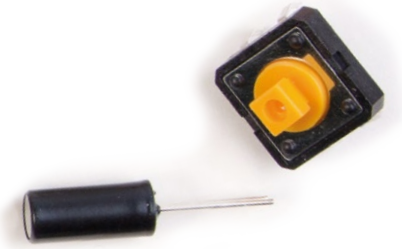
connect the middle pins of our potentiometers to those channel pins, which will read the positions and convert that to a value in our program. If you want to know exactly what all the pins are for on this IC, you can do a web search for 'MCP3008 pinout' and that will give you descriptions of each. ICs are very useful in electronics as they mean that we can reduce how complex our circuits are and we don't need to know exactly how they work inside. They are a little bit like modules in Python.

06 Finishing the job

When you have checked that all the connections are correct, you can switch the Raspberry Pi back on and reload your program. You may want to initialise the SPI interface on your Raspberry Pi by going to the main desktop menu > Preferences > Raspberry Pi Configuration, and go to the Interfaces tab. It will work without this, but may cause a few warnings in the Thonny shell window. So, all being well, when you run your program you will have a game of Pong which can be controlled by two players with the potentiometers. If it doesn't work first time, check your code and then your wiring, and try again.



▲ When everything is plugged in and working, you can play Pong!



mwc3.py

Language: Python
magpi.cc/umUcfq

```
import pgzrun
import random
from gpiozero import MCP3008
import math

pot1 = MCP3008(0)
pot2 = MCP3008(1)

# Set up the colours
BLACK = (0  ,0  ,0  )
WHITE = (255,255,255)
p1Score = p2Score = 0
BALLSPEED = 5
p1Y = 300
p2Y = 300

def draw():
    screen.fill(BLACK)
    screen.draw.line((400,0),(400,600),"green")
    drawPaddles()
    drawBall()
    screen.draw.text(str(p1Score) , center=(105, 40), color=WHITE, fontsize=60)
    screen.draw.text(str(p2Score) , center=(705, 40), color=WHITE, fontsize=60)

def update():
    updatePaddles()
    updateBall()

def init():
    global ballX, ballY, ballDirX, ballDirY
    ballX = 400
    ballY = 300
    a = random.randint(10, 350)
    while (a > 80 and a < 100) or (a > 260 and a < 280):
        a = random.randint(10, 350)
    ballDirX = math.cos(math.radians(a))
    ballDirY = math.sin(math.radians(a))

def drawPaddles():
    global p1Y, p2Y
    p1rect = Rect((100, p1Y-30), (10, 60))
    p2rect = Rect((700, p2Y-30), (10, 60))
    screen.draw.filled_rect(p1rect, "red")
    screen.draw.filled_rect(p2rect, "red")

def updatePaddles():
    global p1Y, p2Y

    p1Y = (pot1.value * 540) +30

    p2Y = (pot2.value * 540) +30

    if keyboard.up:
        if p2Y > 30:
            p2Y -= 2
    if keyboard.down:
        if p2Y < 570:
            p2Y += 2
    if keyboard.w:
        if p1Y > 30:
            p1Y -= 2
    if keyboard.s:
        if p1Y < 570:
            p1Y += 2

def updateBall():
    global ballX, ballY, ballDirX, ballDirY, p1Score, p2Score
    ballX += ballDirX*BALLSPEED
    ballY += ballDirY*BALLSPEED
    ballRect = Rect((ballX-4,ballY-4),(8,8))
    p1rect = Rect((100, p1Y-30), (10, 60))
    p2rect = Rect((700, p2Y-30), (10, 60))
    if checkCollide(ballRect, p1rect) or checkCollide(ballRect, p2rect):
        ballDirX *= -1
    if ballY < 4 or ballY > 596:
        ballDirY *= -1
    if ballX < 0:
        p2Score += 1
        init()
    if ballX > 800:
        p1Score += 1
        init()


def checkCollide(r1,r2):
    return (
        r1.x < r2.x + r2.w and
        r1.y < r2.y + r2.h and
        r1.x + r1.w > r2.x and
        r1.y + r1.h > r2.y
    )

def drawBall():
    screen.draw.filled_circle((ballX, ballY), 8, "white")
    pass

init()
pgzrun.go()
```