

LEARN TO CODE

WITH **SCRATCH** & **PYTHON**

Anybody can learn to code! Programming a computer is much easier than you think. Come with us and we'll help you get started

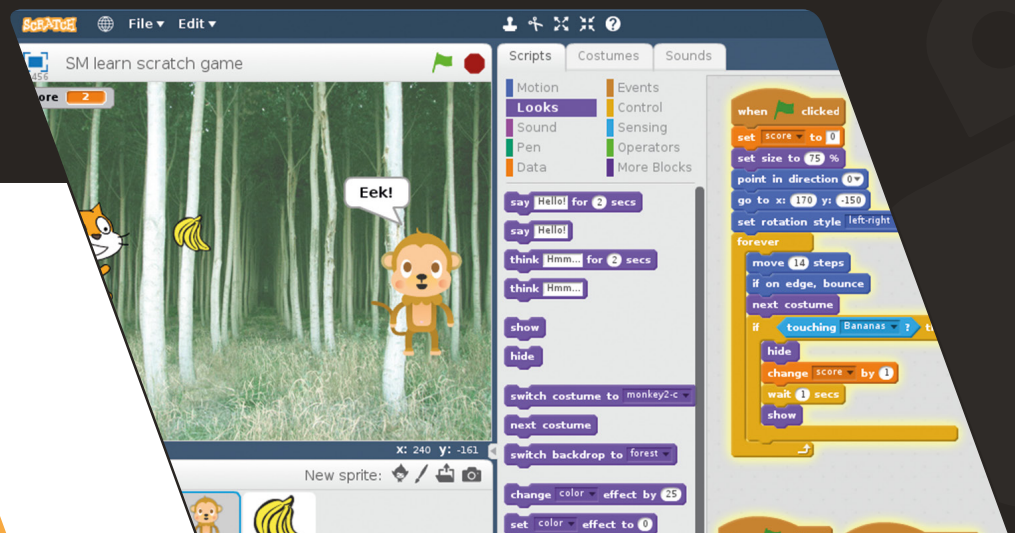


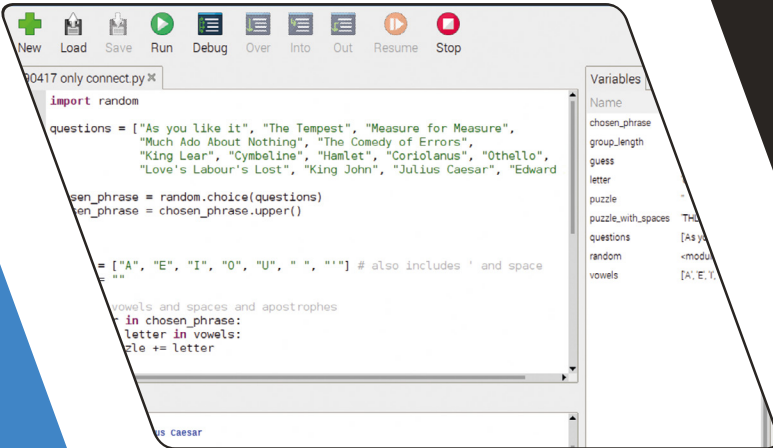
AUTHOR

Sean McManus

Author/co-author of inspiring coding books including Mission Python, Cool Scratch Projects in Easy Steps, and Raspberry Pi For Dummies. Get free chapters at Sean's website.

sean.co.uk





```

import random

questions = ["As you like it", "The Tempest", "Measure for Measure",
            "Much Ado About Nothing", "The Comedy of Errors",
            "King Lear", "Cymbeline", "Hamlet", "Coriolanus", "Othello",
            "Love's Labour's Lost", "King John", "Julius Caesar", "Edward

chosen_phrase = random.choice(questions)
chosen_phrase = chosen_phrase.upper()

shift = 3

letters = ["A", "E", "I", "O", "U", " ", "'", "'"] # also includes ' and space

def caesar_cipher(phrase, shift):
    vowels and spaces and apostrophes
    for letter in chosen_phrase:
        if letter in vowels:
            letter = shift_letters(letter, shift)
            letter += letter

```

Learning to code can be one of the most profound skills you will ever develop. With code, you can control a computer. You can get it to do things for you, and also control gizmos and other computers. Kick back and let your computer do all the work.

Sure, that's cool. But coding is about more than that. It's about understanding how computers work, and getting a better understanding of how technology – and the modern world – works. It's about breaking down problems into little bits and solving them. It's an amazingly helpful life skill. That's why it's profound.

On a more practical level, knowing just a little code can lead to better job opportunities; a little more can open up well-paid and fun jobs. It's an impressive skill to put on your CV and anybody can do it. Anybody.

Coding is a lot easier than you think. And putting the power of computing and digital making into the hands of people is what Raspberry Pi is all about.

The Raspberry Pi is 'the little computer that could', and you're 'the person who can'. Don't worry: you've got this. We can help you get started.



p28 START CODING WITH SCRATCH



p31 CODE A QUIZ GAME WITH PYTHON



p34 MAKE AN LED TORCH WITH PYTHON



p35 BUILD AN ELECTRONIC GAME

“ The Raspberry Pi is 'the little computer that could', and you're 'the person who can' ”

Start coding with Scratch

Beginners, arise! It's time to take your first steps with coding, as we introduce you to Scratch and Python, with a sprinkling of twinkling LEDs! By **Sean McManus**

Being able to write programs is like a superpower: it means you can get your computer to do whatever you want. Join us as we show you how to make your first programs using Scratch and Python. You'll also see how easy it is to build simple electronics projects.

A program is just a set of instructions. In Scratch, the instructions are written with visual blocks that lock together to make a sequence called a script.

The blocks are colour-coded to help you find them. To find the brown blocks, for example, click the brown Events button above the Blocks Palette.

Scratch makes coding easier, because you don't need to worry about the spelling of commands. And everything is laid out in front of you.

Understanding coordinates

The Scratch screen is divided into units called steps. When the cat moves 10 steps, it only makes one movement, but that stride shifts it 10 positions across the Stage. The middle of the Stage is at $x=0, y=0$. The x-axis (left to right) runs from -240 to +240, and the y-axis (bottom to top) runs from -180 to +180. The directions the sprite can move in are numbered 0 (up), 90 (right), 180 or -180 (down), and -90 (left). You can use numbers in between those numbers too, so -45 would be a north-west direction. Why not try starting a new project and joining some Motion blocks together to experiment? You can run a script or a block by clicking it, or use the **when flag clicked** block as we did in our program here.

You'll Need

- ▶ Scratch 2
- ▶ Raspbian with Desktop and Recommended Software
- ▶ A good sense of timing!

01 Start Scratch 2

Open Scratch by clicking on the Raspberry Pi Menu icon and choosing Programming > Scratch 2. You will see Scratch interface and a single character in the top-left, known as 'Scratch Cat'.

To control the Scratch Cat, we're going to drag blocks from the Blocks Palette into the Scripts Area and join them together.

Start by clicking on Events and drag the **when clicked** block to the Scripts Area.

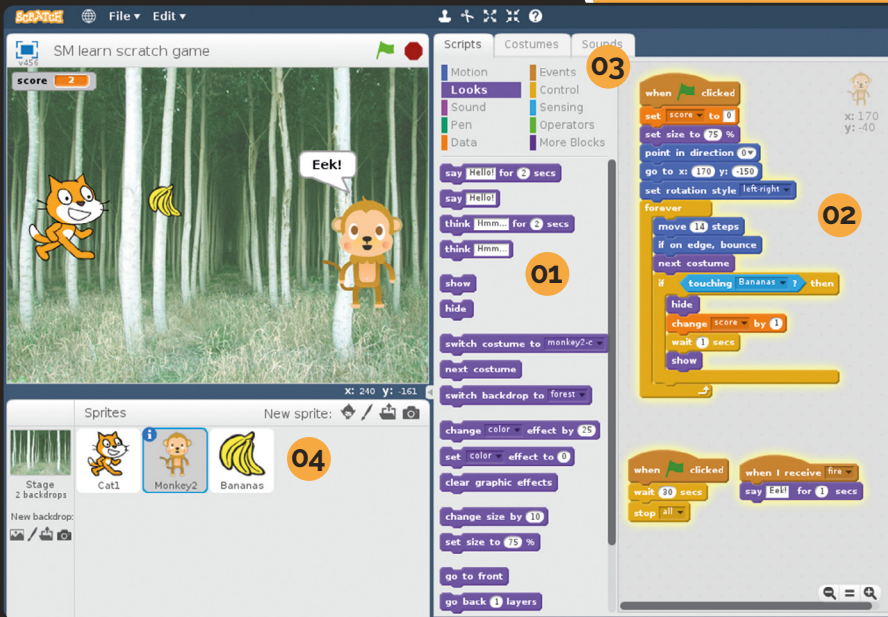
Now click on Motion and drag a **point in direction 90** block and connect it to the bottom of the **when clicked** block.

Click on the fields in the blocks to edit the numbers. Click on '90' and change it to '0'.

Now click and drag the blocks below, and edit their numbers, to build a script for Scratch Cat. This script runs when you click the green flag above the Stage. It sets the cat's movement direction to up, puts the cat in the top-left corner of the Stage, and sets it to always face left or right. Then, the movement blocks inside the **forever** bracket keep the cat moving all the time.

Click the green flag to run your script. Scratch Cat will move to the left side and bounce up and down.





- 01 **Blocks Palette:** Find instruction blocks here
- 02 **Scripts Area:** Drag and drop blocks here to build your script (program)
- 03 **Buttons:** Click to view different types of blocks in the Blocks Palette below
- 04 **Sprite List:** Select and manage sprites here

02 Send a broadcast

The moving objects in Scratch, including the cat, are called sprites. One sprite can send a message to all the other sprites using a broadcast. You can't hear it or see it on screen, but sprites can listen for it, and then start a script when they receive it. We'll use a broadcast to make the cat throw some bananas. Click the brown Events button, and add the two blocks below to the Scripts Area. This new script doesn't join to the existing script (from Step 1) – it sits on its own in the same Scripts Area.

You need to change 'message1' to 'fire'. Click the down arrow next to 'message1' in the broadcast block and choose New Message. Enter the message name 'fire' and click OK.

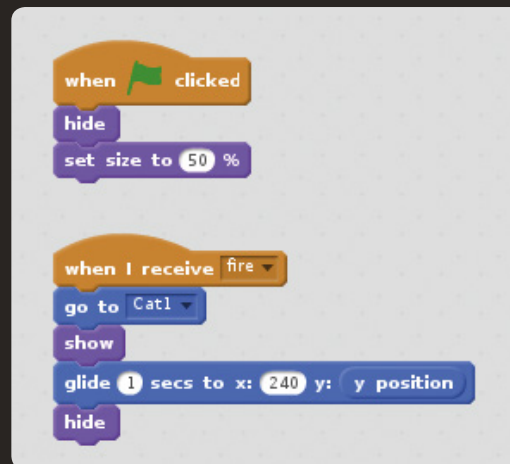
As the program is running, when you tap **SPACE**, the broadcast message is sent silently. Crafty!



03 Add aerodynamic bananas

Scratch enables you to get results fast because it includes its own images and sounds. Click the first New Sprite button above the Sprite List (it will display 'Choose sprite from library' as you hover over it).

Click the Bananas sprite to add it to the Sprite List. Notice that its Scripts Area is blank. We'll give Bananas two scripts. The first one sets the sprite's size and makes it invisible when you click the green flag. The second runs when the cat broadcasts its fire message. Click and drag the blocks below to the Scripts Area.



When adding the **glide** block, drag the **y position** block inside the round 'y:' field to replace the '10' default value.

The 'fire' broadcast makes the bananas jump to the cat (use the go to mouse-pointer block, and choose Cat1 in its menu). Then it makes the bananas visible, glides them across the screen, and hides them again.

Click the green flag, and tap **SPACE** to test. Scratch Cat now throws bananas.

Top Tip

Get the right sprite

Make sure you're adding scripts to the correct sprite. You can select a sprite by clicking it in the Sprite List.

Top Tip

Blocks inside blocks

Some blocks can go inside other blocks. In Step 3, the **y position** block keeps the sprite's y position the same while its x position changes.

```

when green flag clicked
  set score to 0
  set size to 75 %
  point in direction 0
  go to x: 170 y: -150
  set rotation style left-right
  forever loop
    move 14 steps
    if on edge, bounce
    next costume
  
```

04 Add monkey magic

Variables are names used to remember information, such as a score, that might change. Click the Data button and click Make a Variable. Give it the name 'score', select For All Sprites, and click OK. Let's add a moving target for the cat to try to hit. Click the 'Choose sprite from library' icon and choose the Monkey2. Select Monkey2 in the Sprites area and give it the script above.

The monkey's script sets the new score variable to 0 when the game starts, then makes the monkey move up and down. Each sprite can have more than one image: the Next Costume block cycles through them, creating an animation.

05 Add collision detection

Our monkey will react when it's struck by a bunch of bananas. To do that, we use an **if** block; this checks whether

▼ These last two scripts add a timer and make the monkey react when it sees bananas

```

when green flag clicked
  wait 30 secs
  stop all

when I receive fire
  say Eek! for 1 secs
  
```

Where next?

We're huge fans of Scratch at *The MagPi*, so check out our past issues online for more Scratch tutorials. Issue 5 includes a memory game, like Simple Brian. Issue 34 has a multiple-choice quiz, and our 2018 Annual included an introduction to electronics and Scratch. See issue 76 for a roundup of resources to help you learn Scratch, and don't forget there's a Scratch book in *The MagPi's* own Essentials series (magpi.cc/learnscratch) and the *Code Club Book of Scratch* (magpi.cc/ccbook1).

```

if touching Bananas ? then
  hide
  change score by 1
  wait 1 secs
  show
  
```

something is true – in this case, whether the monkey is touching the bananas. If so, the blocks inside its bracket are run. Here, those blocks hide the monkey, add 1 to the score, and wait one second before showing the monkey again.

This whole code chunk goes inside the monkey's **forever** bracket – below the **next costume** block – so the program keeps checking whether the monkey has been hit.

06 Finishing touches

Let's add a simple timer to stop the sprites moving after 30 seconds, and make the monkey react when it sees incoming bananas. Add these two scripts to the monkey sprite. Now the game is complete, why not try experimenting with it? Can you make the monkey move erratically instead of disappearing when it's hit? Can you change the sprites' positions and directions to turn the game sideways, making it more like Space Invaders? What about adding more targets to hit? One of the best ways to learn to code is by experimenting with existing programs.

Code a quiz game with Python

Make your own text quiz game that mangles famous phrases using the Python language

Many people progress from Scratch to Python, a programming language that is powerful, easy to get started with, and much easier to read and write than other languages.

We're going to make a simple quiz question generator that strips the vowels and shuffles the spaces in a phrase. The player has to work out what that phrase is.

We'll be using Thonny, which provides a friendly single-screen environment for running and testing Python code. Like Scratch, the Thonny IDE (integrated development environment) comes pre-installed in the Raspbian with Desktop and Recommended Software operating system.

01 Create a list of questions

As well as variables, Python has lists, which can store multiple pieces of information. Our program creates a list called `questions`. Each item in the list is a piece of text, known as a string. In

Getting indentation right

Python uses indentation to show which instructions belong to a function, an if statement, or a repeating section. As you can see in Step 3 (overleaf), you can have multiple levels of indentation. The last line belongs to the if instruction, and that is repeated inside the for loop. The best way to get the indentation right is to remember the colon at the end of the previous line. Then, Thonny will add the indentation for you automatically. If you forget, use four spaces at the start of the line to insert the indentation. You'll still need to fix that missing colon, though!

```

1 import random
2
3 questions = ["As you like it", "The Tempest", "Measure for Measure",
4             "Much Ado About Nothing", "The Comedy of Errors",
5             "King Lear", "Cymbeline", "Hamlet", "Coriolanus", "Othello",
6             "Love's Labour's Lost", "King John", "Julius Caesar", "Edward
7
8 chosen_phrase = random.choice(questions)
9 chosen_phrase = chosen_phrase.upper()
10
11
12
13 vowels = ["A", "E", "I", "O", "U", " ", "'"] # also includes ' and space
14 puzzle = ""
15
16 # remove vowels and spaces and apostrophes
17 for letter in chosen_phrase:
18     if not letter in vowels:
19         puzzle += letter
20

```

Shell

```

JLSCS R
What is your guess? Julius Caesar
That's correct!

>>> %Run '190417 only connect.py'

THLL
What is your guess? Romeo and Juliet
No. The answer is OTHELLO

>>> |

```

Python, strings are surrounded by double quotes to show where they start and end. The whole list is enclosed in square brackets, and there are commas between the list items. Type in the code below, save your program, and then click Run. If it worked, you should see no error messages in the Shell window.

```

import random

questions = ["As You Like It",
            "The Tempest", "Measure for Measure",
            "Much Ado About Nothing",
            "The Comedy of Errors",
            "King Lear", "Cymbeline",
            "Hamlet", "Coriolanus", "Othello",
            "Love's Labour's Lost",
            "King John", "Julius Caesar",
            "Edward III"]

```

- 01 Type in and edit your program code here
- 02 Enter direct commands and see program input in the Shell here
- 03 Keep track of the data your program is processing here

You'll Need

- Raspbian with Desktop and Recommended Software
- Thonny

Top Tip

Pesky punctuation!

Take care to add the colons at the end of the `if` and `else` instructions. The code won't work without them.

02 Pick a random question

Python includes modules of prewritten code you can use, such as the `random` module we imported in Step 1. The first new instruction creates a new variable called `chosen_phrase` and puts a randomly chosen question into it. The second line converts the `chosen_phrase` to upper case. Run the program a few times and look at the value of `chosen_phrase` in the Variables pane. You should see different names come up, although names can also repeat.

Add a line of space between the code in Step 1 and add the following code:

```
chosen_phrase = random.choice(questions)
chosen_phrase = chosen_phrase.upper()
```

03 Strip the vowels and spaces

Let's create a new list of forbidden characters, chiefly the vowels, but also the space and the apostrophe. That last list item in the `vowels` list is an apostrophe inside double quotes. We create an empty string variable, called `puzzle`. We're going to go through each letter in the phrase, check whether it's in the `vowels` list, and if not, add it to the end of the `puzzle` string. The `for` instruction sets up a repeating piece of code, called a loop. The instructions that should be repeated are indented from left. Each time around the loop, the variable `letter` is set to contain the next character from the `chosen_phrase` string. The `if` instruction checks whether the letter is in the `vowels`

Where next?

You can find Python code to dissect in most issues of *The MagPi*. Issue 53 (magpi.cc/53) includes a more in-depth beginner's guide to Python, covering variables, looping with `while` and `for`, branching with `if`, and functions, which we'll cover in this issue shortly. Issue 54 (magpi.cc/53) introduces object-oriented programming in both Scratch and Python. Issue 73 (magpi.cc/73) includes a roundup of Python books and online resources. There is a book in our Essentials series too, called *Make Games with Python* (magpi.cc/gameswithpython).

list. If it is not, the letter is added to the end of `puzzle`. The `+=` means 'add at the end'. Run the program, then test it's working by looking at the contents of `puzzle` in the Variables panel. It should contain no vowels, spaces, or apostrophes.

Add the following code to the program:

```
vowels = ["A", "E", "I", "O", "U", " ", "'", ""]
puzzle = ""

for letter in chosen_phrase:
    if not letter in vowels:
        puzzle += letter
```

04 Insert random spaces

Each character in the string can be referred to by its position number, starting at 0. The number is called an index, and you put it in square brackets after the string. Try this in the Shell (click on the line starting with `>>>`). Instructions in the Shell are carried out immediately. Enter the following:

```
print("Hello"[1])
```

You get 'e' back (because the first character is number 0). You can get a chunk too (called a slice) by giving a start and end index, like this:

```
print("Hello"[1:4])
```

It gives you 'ell' because the last index position (4) is left out. We'll create a new list, called `puzzle_with_spaces`, by adding chunks of the `puzzle` string and a space until there's no `puzzle` string left. The `while` loop repeats the indented instructions below as long as the length of `puzzle` is more than 0. The `group_length` variable is given a random whole number (integer) from 1 to 5. Then that many letters are added to `puzzle_with_spaces` from the front of `puzzle`, plus a space. Those characters are

Debugging in Thonny

You can step through the program slowly to see what it's doing, which can help you to find errors.

Click the Debug button in Thonny, then click the Over button to run through each instruction in turn.

Watch the Variables pane on the right to see how the lists and strings change at each stage of the program. Thonny also helps you avoid errors by highlighting unclosed brackets and double quotes.

“ Python is **easy to get started with** and much easier to read and write than other languages ”

then cut off the front of puzzle. The slicing here only uses one number, so the other one is assumed to be the start or end of the string.

Add this code to your program:

```
puzzle_with_spaces = ""

while len(puzzle) > 0:
    group_length = random.randint(1,5)
    puzzle_with_spaces +=
puzzle[:group_length] + " "
    puzzle = puzzle[group_length:]
```

05 Add collision detection

It prints the puzzle_with_spaces. It then uses the input() function to ask you what your guess is. Your answer goes into the guess variable, and is then converted to upper case to make sure it matches the correct answer if it's right. The if instruction checks whether guess is the same as chosen_phrase. If so, it prints one message. Otherwise, the instruction indented under else runs, to tell you the right answer. In Python, one = is used to put a value into a variable, but two (==) are used to compare items in an if instruction.

Add this code to the end of the program:

```
print(puzzle_with_spaces)
guess = input("What is your guess? ")
guess = guess.upper()

if guess == chosen_phrase:
    print("That's correct!")
else:
    print("No. The answer is ", chosen_phrase)
```

Click the Run button and hopefully you'll see some letters in the Shell and 'What is your guess?' Enter an answer and you'll see 'That's correct!' or 'No. The answer is' and the correct response.

If you've typed the code out by hand, it's likely that you'll see an error message. Go through your code line-by-line and compare it to the full code in `quiz_game.py`.

quiz_game.py

DOWNLOAD
THE FULL CODE:

► Language: Python



magpi.cc/github82

```
001. import random
002.
003. questions = ["As You Like It", "The Tempest",
004. "Measure for Measure", "Much Ado About Nothing",
005. "The Comedy of Errors", "King Lear", "Cymbeline",
006. "Hamlet", "Coriolanus", "Othello", "Love's Labour's Lost",
007. "King John", "Julius Caesar", "Edward III"]
008.
009. chosen_phrase = random.choice(questions)
010. chosen_phrase = chosen_phrase.upper()
011.
012. vowels = ["A", "E", "I", "O", "U", " ", "'"]
013. puzzle = ""
014.
015. for letter in chosen_phrase:
016.     if not letter in vowels:
017.         puzzle += letter
018.
019. puzzle_with_spaces = ""
020.
021. while len(puzzle) > 0:
022.     group_length = random.randint(1,5)
023.     puzzle_with_spaces += puzzle[:group_length] + " "
024.     puzzle = puzzle[group_length:]
025.
026. print(puzzle_with_spaces)
027. guess = input("What is your guess? ")
028. guess = guess.upper()
029.
030.
031. if guess == chosen_phrase:
032.     print("That's correct!")
033. else:
034.     print("No. The answer is ", chosen_phrase)
```


Build an LED torch and electronic game

Discover how easy it can be to get lights blinking and buttons clicking using GPIO Zero and use your new-found skills to build an electronic game

One of the best things about the Raspberry Pi is that you can easily hook up your own electronics projects. Using some electronics components and the GPIO Zero library, you can program a puzzle game where you have to repeat a sequence of lights that gets longer each turn. You might remember a similar electronic game from your childhood, but we call ours Simple Brian. In issue 77 (magpi.cc/77) we showed you how to use Python code to play the game on screen. This issue, we'll show you how to make the electronic game itself, building on your new-found Python skills from Missing Vowels.

First, we're going to show you how to build a torch by lighting up LEDs. Let's get going.

01 Connect your first button

The torch circuit diagram (Figure 1) shows an LED light connected to the GPIO pins of a Raspberry Pi using a breadboard (see magpi.cc/breadboard for a primer on using this piece of equipment).

Press the button into the board, then use jumper wires to form a circuit with your Pi, as shown by

the yellow wires in the diagram. The first button connects to the GPIO 2 pin on one side, and to the ground rail on the other side. We'll connect the latter to a ground pin on the Pi, so anything plugged in that row of holes connects to ground.

02 Connect your first LED

Always use a resistor when you connect an LED to your Pi, to prevent the LED drawing too much current and getting damaged. Both the LED and the resistor plug straight into your breadboard. The current flows from GPIO 18, through the resistor, through the LED (lighting it up), to the breadboard's ground rail. LEDs only work one way around: the short leg is the negative side, which you connect to ground. The LED won't light up yet.

03 Make an LED torch

You've made your first circuit! Let's test it by coding a torch. The `torch.py` code shows how to use an LED and a button. It imports the relevant parts of the GPIO Zero library, then sets up an LED called `light`, connected to GPIO pin 18. The button on pin 2 is set up with the name `button`. The `while True` loop checks whether the button is pressed forever. If so, the light is turned on. Otherwise, it's turned off. Pay attention to the capitalisation of LED and Button when setting them up.

04 Add the other buttons and LEDs

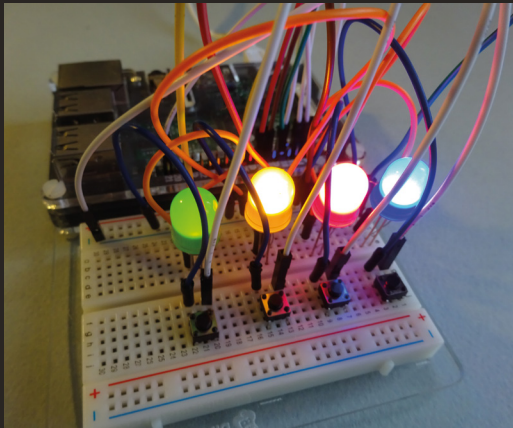
Take a look at Figure 2 (overleaf). It looks complex at first, but the other three buttons and LEDs are connected in the same way as

You'll Need

- ▶ 4 × LEDs (ideally different colours)
- ▶ 4 × 330 Ω resistors
- ▶ 4 × 6 mm Tactile momentary button switches
- ▶ 400-point breadboard
- ▶ 9 × Male-to-female jumper cables
- ▶ 8 × Male-to-male jumper cables
- ▶ PiBow case with Breadboard Base pimoroni.com

Where next?

The online documentation for GPIO Zero (magpi.cc/DPyUc) provides more code examples, including a button-controlled camera, an LED bar graph, and a motion sensor. We surveyed useful resources for basic electronics in issue 77, and there's a book in our Essentials series called *Simple Electronics with GPIO Zero* (magpi.cc/gpio-zero).



▲ The finished game, with all the lights lit up for testing

the first ones, just using different pins on the Pi. All the buttons (yellow wires) connect to the Pi's inner row of pins, and the LEDs (blue wires) to the outermost row. We've separated the components in this diagram a bit so it's easier to see how to wire it up, but try to line up your LEDs and buttons on the breadboard so it's easier to play the game.

05 Test them all

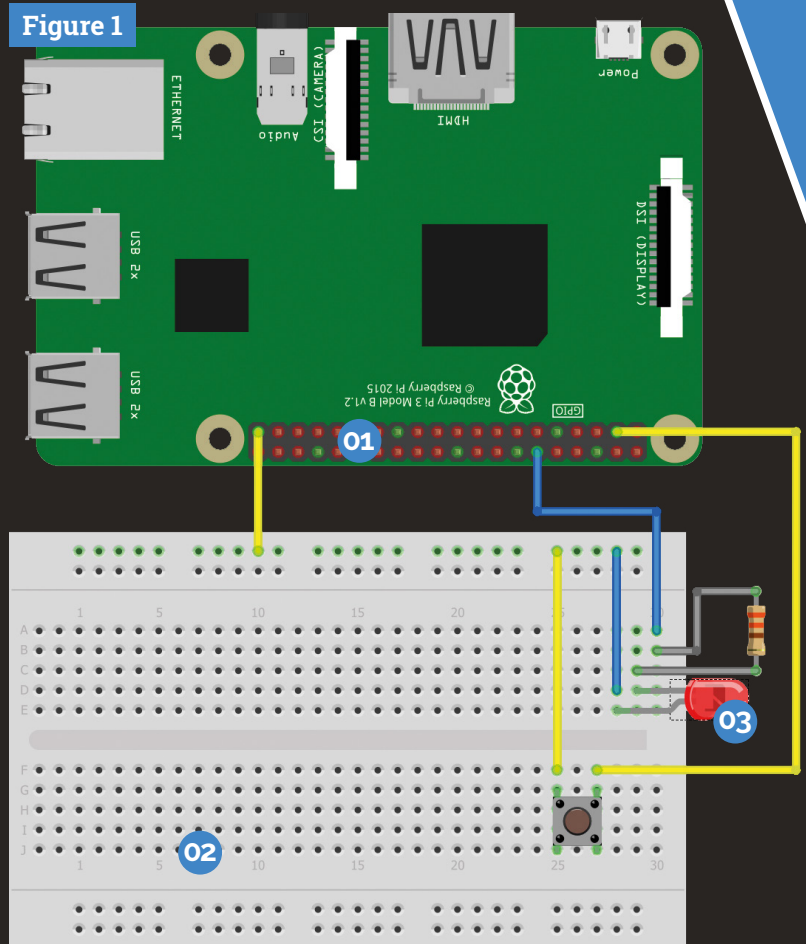
Now you can test these LEDs and buttons too. Modify your torch code to use LED(23) and Button(3) and then run the program to test the next light switch works. Then check LED(24) and Button(4), and finally LED(25) and Button(17). The buttons should be next to the LED they illuminate.

06 Build an electronic game

Now we're ready to start making the Simple Brian game (see [brian.py](#)). This starts by setting up a list of buttons, and a list of their associated LEDs. It also creates an empty list called `sequence`, which we'll use for the sequence of lights the player must repeat. With each turn, it'll get longer.

07 Add functions

Functions enable you to bundle up a set of instructions so you can reuse them. You have to define a function before you can use it. To define a function, you use `def`, followed by the function name, `()`, and a colon. The brackets are there to hold any info you're sending to the function, but we don't need to send any so they're empty. You can tell which instructions belong to a function, as they're indented. The `lights_on()` and `lights_off()` functions use a loop to go through all the items in



- 01 Count the pins to see where to connect your wires
- 02 The breadboard makes it easy to plug in components and quickly set up circuits
- 03 Using this simple circuit (with an LED and a button), you can make a push-button torch

torch.py

DOWNLOAD THE FULL CODE:

magpi.cc/github82

Language: Python

```
001. # Torch demo
002. from gpiozero import Button, LED
003.
004. light = LED(18)
005. button = Button(2)
006.
007. while True:
008.     if button.is_pressed:
009.         light.on()
010.     else:
011.         light.off()
```

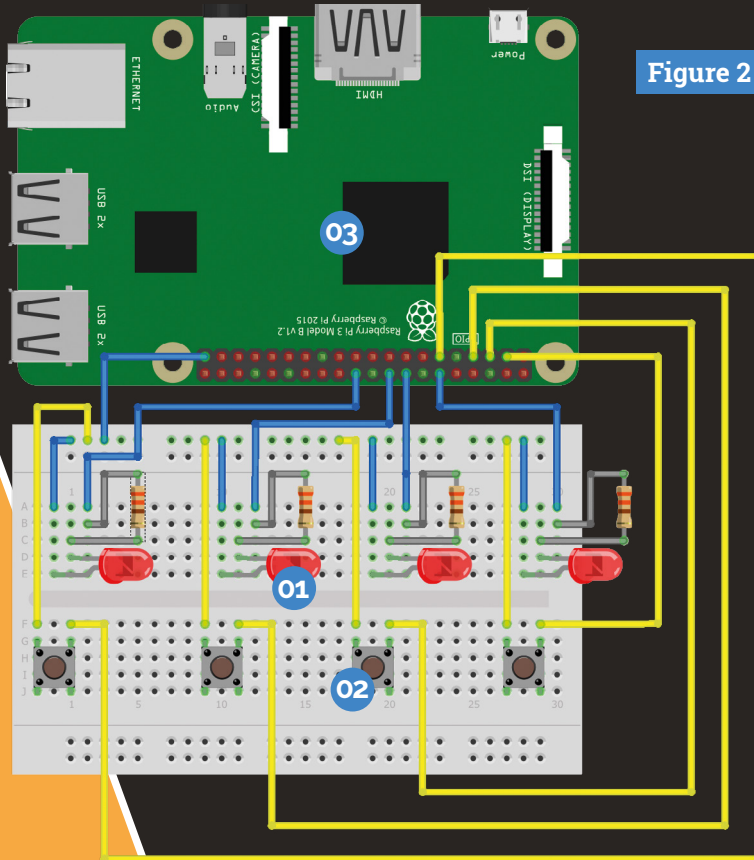
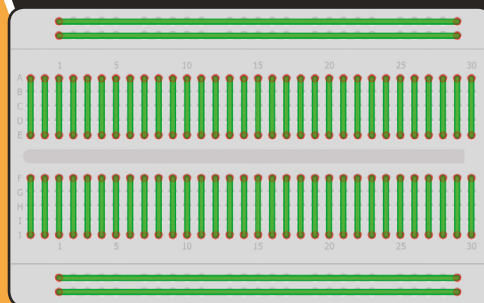


Figure 2

- 01 Just like its famous namesake, our Simple Brian game features four LED lights that flash up a sequence for you to memorise
- 02 The circuit also contains four buttons. We'll push these to repeat the sequence of lights that we've memorised
- 03 At the heart of our project is the Raspberry Pi. The buttons and lights are connected to our computer with wires. Code in the computer flashes the lights and keeps an eye on the buttons we're pushing



▲ On a breadboard, the rails along the edges are connected in a long line, and the short lines of dots in the middle are connected to each other

the `leds` list, putting them into `led`, then turning `led` on or off. The `flash_all()` function shows how to repeat a set number of times, in this case 3. The loop turns the lights on and off, with a 0.25 second pause after each change using `sleep(0.25)`.

08 Lights test flash all

After you've entered the functions (down to line 23), you can test the program by adding `flash_all()` as the last line and then running it. All the lights should flash together, three times. Delete that test line before you carry on. In line 25, the program runs the `lights_off()` function to ensure the lights are all off before the game begins.

09 Add to the sequence

Now we enter the main game loop, under `while True` (line 27). Everything from here on in is indented to show it belongs to that loop, repeating endlessly. The game sequence starts as an empty list, so the first thing we do is to add an LED. We pick a random LED using `random.choice()` and add it to the end of the sequence list using the `append()` list method. A list method is a built-in Python function that you can apply to a list. Other methods are available to insert and remove items, and sort the list, among other things.

10 Play the list sequence

The lights all flash three times using the `flash_all()` function before the sequence begins, to show this is the start of the sequence. Then a loop is used that takes each LED from the sequence list, and puts it into `light`, in turn. It's turned on, there's a short pause, then it's turned off. There's another short pause so it's obvious there are multiple flashes of the same light if it repeats in the sequence. In round one, there's only one light in the sequence list, but as the game progresses, this loop will get longer. You can run the program at this point to see the light sequence gradually extend, without the player getting a chance to guess.

11 Get the player's guess

Getting the player's guess uses a similar loop to the one that plays the lights sequence. It

brian.py

[DOWNLOAD THE FULL CODE:](#)
[▶ Language: Python](#)
[!\[\]\(23d9fc146e83b5c3013cfa32c784f8d5_img.jpg\) magpi.cc/github82](https://magpi.cc/github82)


```

001. from gpiozero import Button, LED
002. from time import sleep
003. import random
004.
005. buttons = [Button(2), Button(3), Button(4),
006.            Button(17)]
007.
008. sequence = []
009.
010. def lights_on():
011.     for led in leds:
012.         led.on()
013.
014. def lights_off():
015.     for led in leds:
016.         led.off()
017.
018. def flash_all():
019.     for _ in range(3):
020.         lights_on()
021.         sleep(0.25)
022.         lights_off()
023.         sleep(0.25)
024.
025. lights_off()
026.
027. while True:
028.     # Add a new light to the end of the sequence
029.     new_light = random.choice(leds)
030.     sequence.append(new_light)
031.
032.     # Flash all before playback
033.     flash_all()
034.
035.     # play the sequence
036.     for light in sequence:
037.         light.on()
038.         sleep(0.5)
039.         light.off()
040.         sleep(0.25)
041.
042.     # get the player's input
043.     for light in sequence:
044.         guess = None
045.         while guess == None:
046.             for button in buttons:
047.                 if button.is_pressed():
048.                     # convert button push to list
049.                     index number
050.                     guess = buttons.index(button)
051.
052.             if leds[guess] == light:
053.                 light.on()
054.                 sleep(0.5)
055.                 light.off()
056.                 sleep(0.25)
057.             else:
058.                 print("You failed at level ",
059.                       len(sequence))
060.                 for _ in range(10):
061.                     light.on()
062.                     sleep(0.15)
063.                     light.off()
064.                     sleep(0.15)
065.                 sequence = []
066.                 break

```

works its way through the sequence list, accepts a guess, and checks whether it matches the current item in the sequence. There are three loops inside each other here. The program sets the `guess` variable to `None`, a special value in Python. Then a `while` loop keeps repeating until the `guess` variable changes. Inside that, a loop goes through the buttons list, checking each one in turn to see whether it's pressed. If so, the `guess` variable is changed, ending the `while` loop. The program converts the button the player pressed into its index number in the list and puts that into the `guess` variable. That way, we can match the button to its LED, which will be at the same position in the `leds` list. You can find the position of an item in a list using `listname.index(item)`.

12 Check the player's guess

We're still inside the loop going through the sequence here, as the indentation of line 51 shows. Now we check whether the light the player guessed (`leds[guess]`) matches the current light in the sequence. If so, the light is turned on and then off again. If the two lights don't match, the player made a mistake. We can tell how long a list is using `len(listname)`. We use the length of the sequence list to tell the player which level they got to. The correct light is then flashed quickly ten times. The sequence list is emptied to start a new game, and the `break` instruction breaks out of the `for` loop that's getting player input. When the player has either guessed all the lights, or failed, the game repeats from line 28, adding a new light to the sequence. 

Top Tip



Underscoring repetition

The line `for _ in range(3)` repeats the indented instructions three times. The `_` shows we don't need to use the loop number. Often, you'd use a variable name instead.