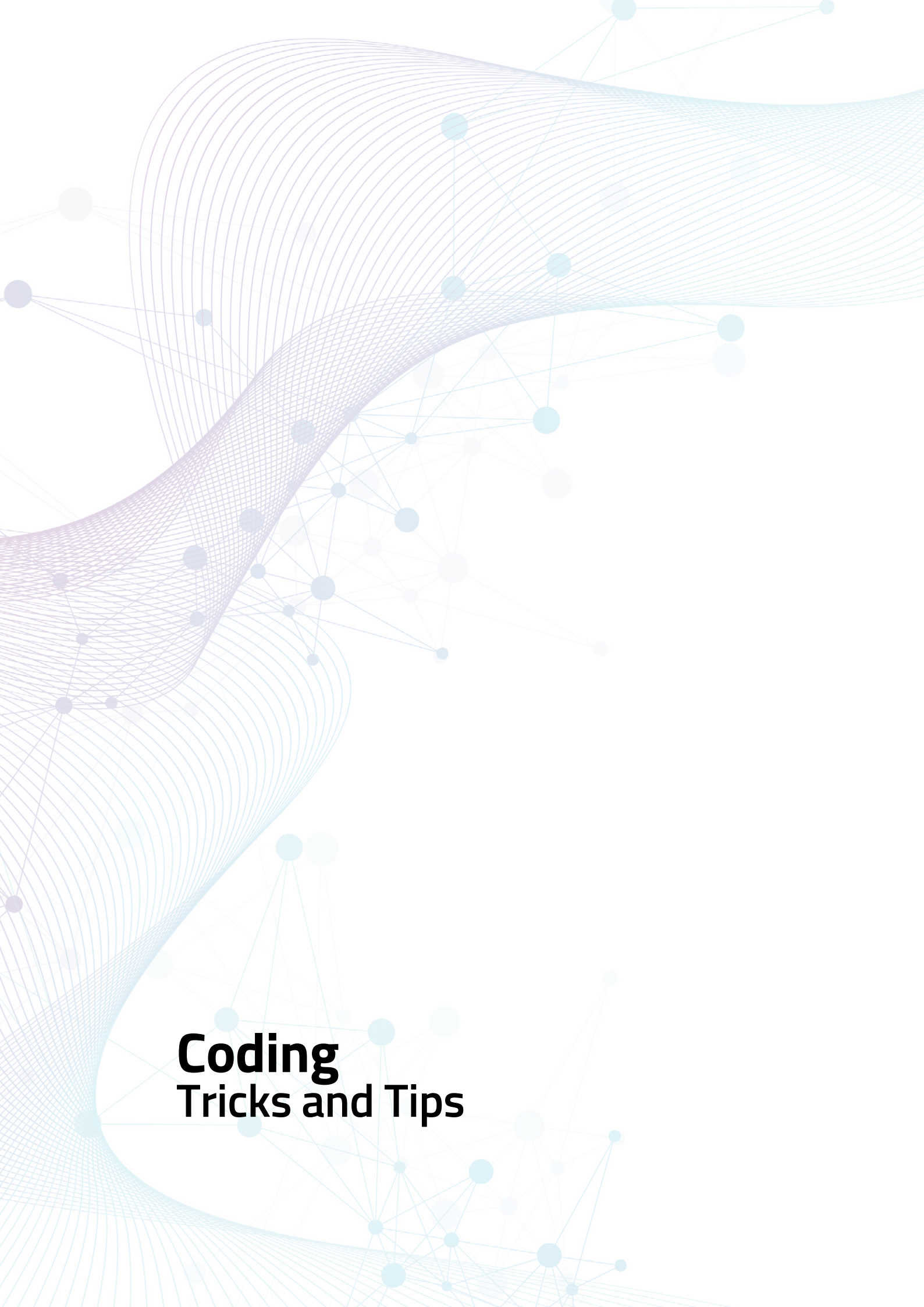
techgo

Coding Tricks and Tips

Next level Secrets & Fixes

Advanced Guides & Tips

Rediscover Your Device



Coding
Tricks and Tips

Coding Tricks and Tips
ISBN: 978-1-912847-53-2

Foreword

Welcome back...

Having completed our exclusive For Beginners digital guidebook, we have taught you all you need to master the basics of your new device, software or hobby. Yet that's just the start!

Advancing your skill set is the goal of all users of consumer technology and our team of long term industry experts will help you achieve exactly that. Over this extensive series of titles we will be looking in greater depth at how you make the absolute most from the latest consumer electronics, software, hobbies and trends!

We will guide you step-by-step through using all the advanced aspects of the technology that you may have been previously apprehensive at attempting. Let our expert guide help you build your understanding of technology and gain the skills to take you from a confident user to an experienced expert.

Over the page our journey continues and we will be with you at every stage to advise, inform and ultimately inspire you to go further.

About the Publisher

From its humble beginnings in 2004, the BDM brand has quickly grown from a single publication produced by a team of just two to one of the biggest names in global tech print and digital publishing, for one simple reason. Our passion and commitment to deliver the very best product to the marketplace.

While the company has grown with a portfolio of over 500 publications crafted by our international staff of respected industry veterans, the foundation that it has been built upon remains the same. That being to create the best quality, fully independent, user friendly and, most essentially, 100% up-to-date content possible.

Delivering not only market leading publications but also piece of mind to our readers, so that they have the very best foundation to build their knowledge, confidence and understanding of their new software and hardware. Our regular readers trust BDM, as should you.

How to use this book

This book has been designed for you to progress through the coreconcepts and fundamentals of use, through to more advanced elements, projects, and ideas. There's something for every style of reader, and for every type of user; there's probably even a few terrible jokes dotted within the pages. So here's how to get the best from it.

Step 1

Don't skip - While it's fun to see what's coming up later in the book, it does make understanding what you're reading more difficult. After all, you wouldn't start reading a book on speaking French, then skip further in without first learning proper grammar, sentence structure and so on. The same applies here. Take your first read-through page by-page, once you've mastered the book, then you can return to key concepts whenever you need.

Step 2

Ever-Changing - While every effort has been made to ensure that this book is up to date, there's no knowing what updates may occur over time. While some companies do offer an accurate roadmap of their future development of a product, it's not always written in stone. For example, an app available for Windows 10 now may not be available with the next update of the operating system. It's up to Microsoft to decide whether they want to drop it for one reason or another. The same, to some extent, applies here. However, we continually update the content in this title, so it's as accurate as possible.

Step 3

Follow the Steps - An obvious one, this. Following the steps from one onwards, in most tutorials in this book, will ensure that you get the result that's intended. If you skip steps, then you may miss out on something important, and not understand how it works later in the book. The temptation to skip something you already know is often too much, but stick with the logical progression of the steps and you'll get the most from what's on offer.

Step 4

Have Fun - Learning a new skill is supposed to be fun. We had fun writing the book, and hopefully you'll have fun reading it and applying new skills. Everyone learns at a different pace, so take your time, digest the tutorials, and keep returning to key concepts if you feel the need to master any element within these pages. The content in this book isn't something we're going to be testing you on, so have fun and enjoy the art of learning something new. And if you create something amazing after reading this book, then let us know.

Contents

BDM's Code Portal
60+ Python programs
21,500+ lines of code
Master Python with the help of our fantastic Code Portal, featuring code for games, tools and more.
Visit: https://bdmpublications.com/code-portal, and log in to get access!

"…you can learn how to start coding using Python, C++, Linux scripting, FUZE BASIC with the Raspberry Pi, Windows batch files and Scratch. We also cover the common pitfalls and mistakes every coder falls into and ways to avoid them…"

@bdmpubs BDM Publications www.bdmpublications.com

Programming with the FUZE

The Raspberry Pi is the powerhouse for many excellent projects. However, one project stands head and shoulders above the rest, the FUZE Project. FUZE is a learning environment for the Raspberry Pi that's amazingly accessible and gets students, teachers and enthusiasts coding and experimenting with the Raspberry Pi quickly and easily.

Used in hundreds of schools across the UK, the FUZE is the perfect combination of Pi potential, imagination, engineering and education, all presented in a cleverly designed retro-themed keyboard case. More importantly, the FUZE also comes with its own programming language, FUZE BASIC. With FUZE BASIC you're able to create simple routines, games, complex algorithms and even interact with robots and other electronics.

Introducing the FUZE Project

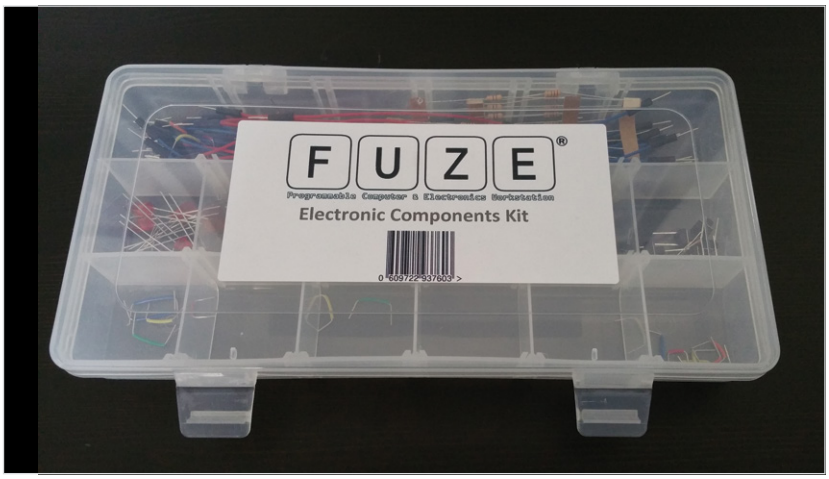
The FUZE Project is a learning environment that's built around the Raspberry Pi and a custom programming language based on BASIC. The FUZE Workstation is the hardware side of the project, incorporating a Raspberry Pi inside a stunning retro-themed case, complete with a full-sized keyboard, IO board and connectivity. The software side is FUZE BASIC, available for both Windows and as a boot image for Raspberry Pi models 2 and 3.

The FUZE Workstation itself, styled on a BBC Micro Model B from the '80s, constructed with a metal body and high quality keyboard complete with programmable red coloured F-keys. Note the holes at the top edges: these are compatible with many popular plastic building blocks and electronic construction kits.

Inside the FUZE workstation is the beating heart of the project, a Raspberry Pi 3. It's connected to the FUZE IO board via the 40-pin GPIO pins with a ribbon cable, with the other connections branching from it: HDMI, power, SD Card and USB. The USB ports have been routed to a powered USB hub, attached to the rear of the workstation, providing the user with four USB ports accessible through the rear IO back plate; there's an Ethernet port too.

The Rear IO back plate is where you plug in your power, connection to a monitor, mouse, SD card, audio and any USB devices you want to include. From left to right, across the back plate, there's an on/off switch, a power port, a bank of four USB ports, the Ethernet port, SD card slot and HDMI port. You can remove the four corner screws of the back plate to gain access to the Raspberry Pi, USB hub and cables inside the workstation if needed.

The FUZE IO board, located in the wide slot at the top of the workstation, is the connection to extended hardware. With it you can connect, program and use sensors, LEDs, robotics and all manner of amazing electronic projects.

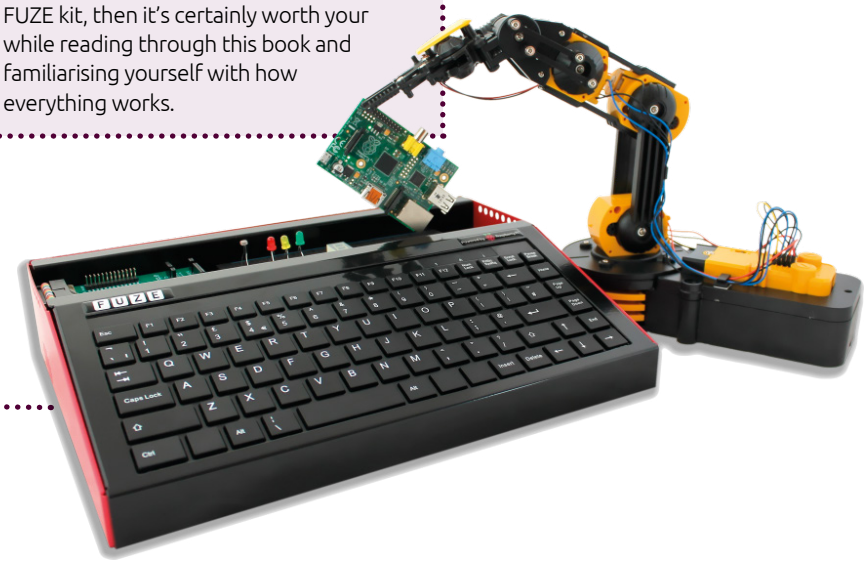


You also receive an electronics kit as part of the FUZE workstation, to help you get started on some of the projects the FUZE is designed to support. Within the kit you can find 24 coloured LEDs, 1 seven-segment LED, 1 light dependant resistor, 8 micro switches, 30 mixed specification resistors, 20 jumper cables and 60 jumper wires.

Complementing the electronics project kit, the FUZE team also bundles an 840-socket solderless breadboard which you can use to wire up interesting projects and use FUZE BASIC together with the Raspberry Pi and the FUZE IO board to control the components from the electronics kit. In case you're wondering why it's called a breadboard, it's because in the early days of electronics users would use a bread board for the base of their projects.

Alongside the other components with the FUZE workstation, you also get either a wired USB or wireless (batteries are included if necessary) mouse and 'FUZE' logo mouse mat.

The kit comes with two ring-bound books containing project ideas for the electronics kit and a programmer's reference guide for FUZE BASIC. If you've purchased the FUZE kit, then it's certainly worth your while reading through this book and familiarising yourself with how everything works.



Depending on which FUZE workstation kit you've purchased, you could also have a robot arm that requires building, along with four D-sized batteries, a BBC micro:bit or even a Capacitive Touch kit. Needless to say, there's plenty of project potential with the FUZE.

Setting Up the FUZE

Thankfully the FUZE Project comes with everything you need to get up and running; you just need to supply the monitor and an Ethernet cable to your network (or you can go Wi-Fi with the Raspberry Pi 3). Before you begin though, let's see how to set up the workstation.

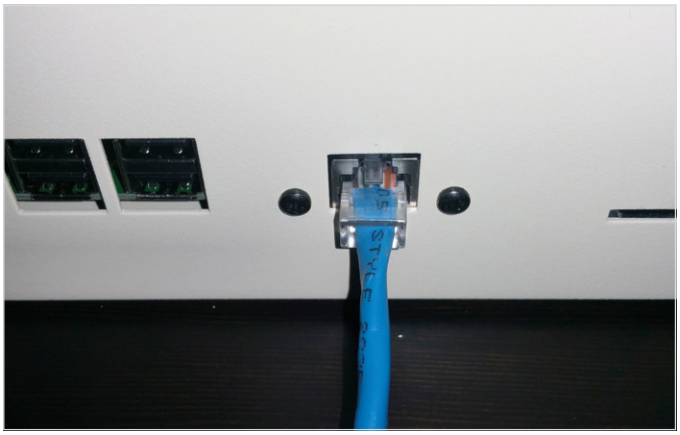
LIGHT THE FUZE

Getting the FUZE up and running is as simple as plugging in a standard desktop computer; but it's always worth running through the process for those who don't know what to do.

STEP 1 Before you power up your FUZE, make sure that the provided SD card is inserted into the SD card slot on the rear IO back plate of the FUZE workstation. The chances are the SD card is already inserted but depending on how the FUZE was packaged, it may be in the electronics kit box.

STEP 2 Grab a spare monitor or if your existing monitor (or TV) can support more than one HDMI connection even better. The FUZE comes with a quality HDMI cable, remove it from its bag and connect one end to the HDMI port on the rear IO back plate of the FUZE and the other to the rear of the monitor or TV.

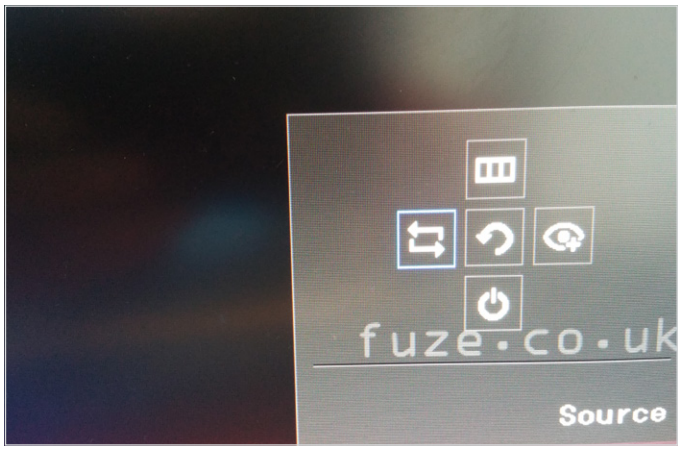
STEP 3 For now, use the Ethernet port, LAN cable, for the FUZE's connection to the home network and ultimately the outside world. You can set up the Wi-Fi but it's always easier to establish a wired connection first if possible. Connect the Ethernet cable to rear IO back plate of the FUZE.



STEP 4 Next, open up the box containing the mouse and plug it into one of the USB ports on the rear IO of the FUZE workstation back plate.

STEP 5 Now open the box containing the power pack and plug it into the power point at the wall and finally to the FUZE workstation itself. The FUZE will power up immediately and start to boot into the custom FUZE Raspbian OS on the SD card.

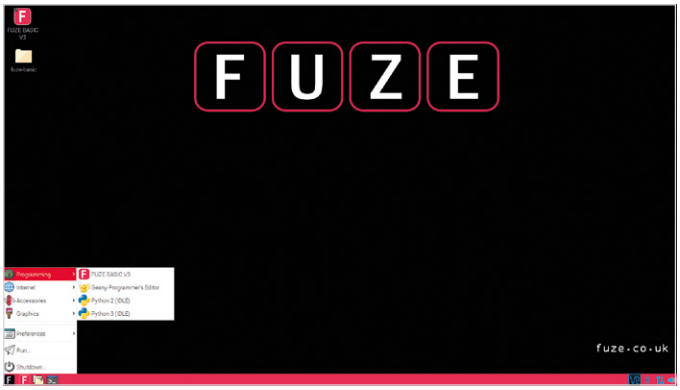
STEP 6 You may need to change the source of the monitor or TV's input to the HDMI, or the numbered HDMI port that you've connected the FUZE to. Once the signal is found by the monitor it displays the FUZE desktop.



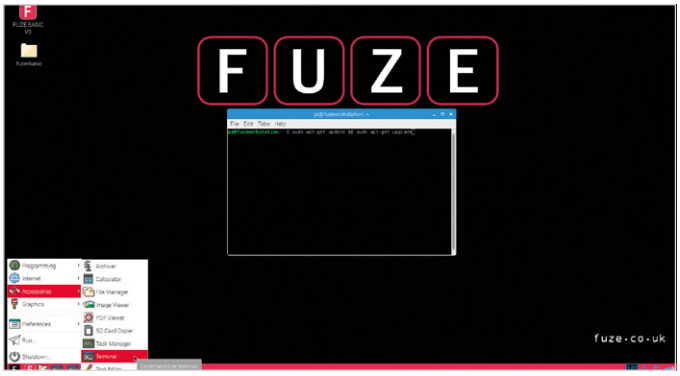
STEP 7 The first thing to notice is that it's significantly different to that of the standard Raspberry Pi Raspbian interface. The launch panel and buttons are located along the bottom of the screen, as with a Windows-type setup, with a couple of icons on the desktop itself.

STEP 8 If you want the Wi-Fi to be the active network connection, look to the bottom right of the desktop for the two arrows (one pointing up, the other down). Click the arrows and the current Wi-Fi access points will be displayed. Connect to yours as you would normally. You can now unplug the Ethernet cable if you wish.

STEP 9 Beyond the different desktop presentation, the FUZE setup works exactly the same as any other Raspberry Pi Raspbian system. You can click the first F (the white F on a black background) to open the system menu detailing the available apps and programs. The second F launches FUZE BASIC, which we'll look at in the next tutorial.



STEP 10 To ensure you're running the latest software and programs, click on the F start button, followed by Accessories > Terminal. In the Terminal enter: `sudo apt-get update && sudo apt-get upgrade` and accept any changes and updates the system has to offer. This will update all your installed software and system files.



Getting Started with FUZE BASIC

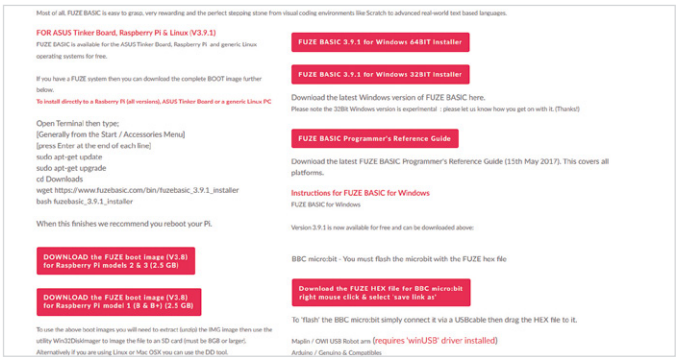
FUZE BASIC is a marvellous programming language to begin learning to code with. It greatly mimics the '80s BASIC versions from the 8-bit machines of the time, such as the Commodore 64, ZX Spectrum and BBC Micro.

BACK TO BASICS

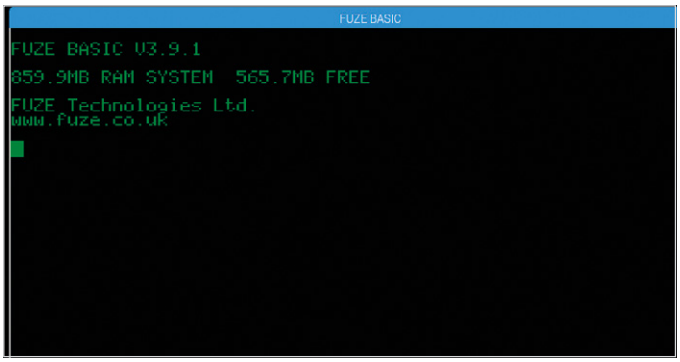
Let's begin our programming journey with FUZE BASIC, an environment where you can create anything, from simple scripts to complex games with graphics and sounds.

STEP 1 In the bottom right panel, the one that's coloured white on a red background? Click it and you launch the FUZE BASIC, complete with a C64-style retro interface. You can also double-click the FUZE BASIC V3 icon on the desktop.

STEP 2 Don't worry if you don't have a FUZE Workstation. FUZE BASIC is available for Windows, the BBC micro:bit and the Raspberry Pi (since it's already running on a RPi). Open a browser to **www.fuze.co.uk/download-fuze.html** and follow the download instructions for FUZE BASIC for Windows and the step-by-step instructions to install it on a Raspberry Pi.



STEP 3 The retro style interface of FUZE BASIC has several themes that you can cycle through, depending on your taste. The default view is that of a Commodore 64 but if you want a different view press the Insert key to cycle through the available interfaces. You'll no doubt recognise some of them, so find one you like.



STEP 4 The screen you're looking at now is called Immediate Mode; pressing the Enter key will reveal a cursor where you can start to enter code. Try this: press Enter, then type: Hello everyone and press Enter again. The output on the screen will display whatever you've typed into the quotation marks.

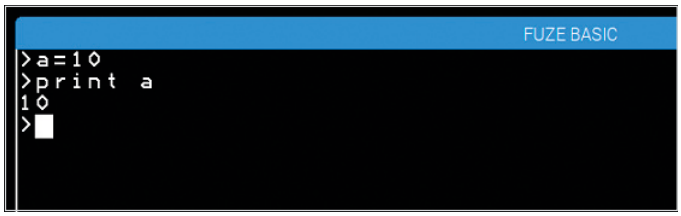
STEP 5

You can also Print the total output of several numbers from within the Immediate Mode. For example, try: `print 10 + 20 + 30`, and press Enter. The sum of the numbers you've entered will now be displayed on the screen, in this case the number 60. Try more numbers and even different mathematical symbols.

STEP 6

If you find the screen getting a little full, enter `cls` to clear the BASIC display. BASIC in Immediate Mode is also capable of storing variables, something which we'll look at in more depth in the next tutorial. For now, try this and press Enter after each line:

```
A=10
Print a
```

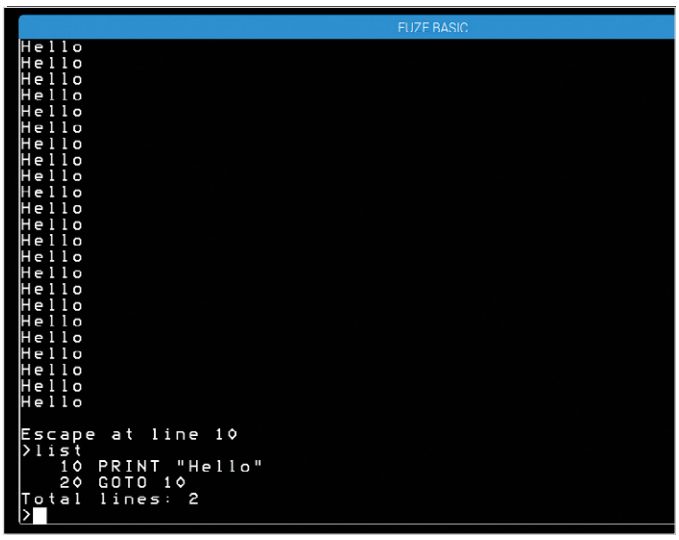


STEP 7

If you're old enough to recall BASIC from the early days of computing, you'll no doubt remember that coding came with line numbers. FUZE BASIC works the same way. Whilst still in Immediate Mode, enter:

```
10 print "Hello"
20 goto 10
```

Now enter `run`. The word Hello should now cycle down the screen. Press the Escape key to exit it.

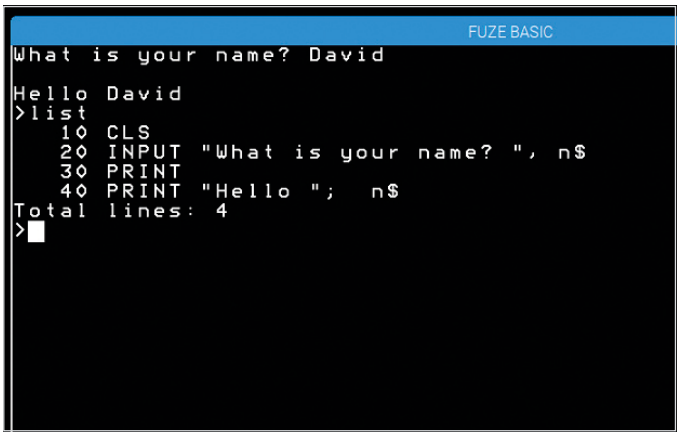


STEP 8

Before we get into variables and other such programming terms, let's have a little play around with a quick listing to ask for user input. Enter this:

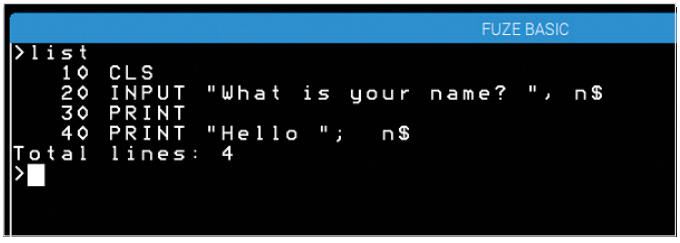
```
10 cls
20 input "What is your name? ", n$
30 print
40 print "Hello "; n$
```

Enter `run` to execute the code.



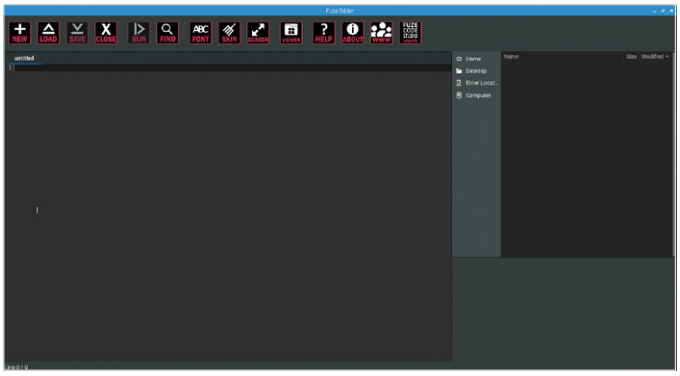
STEP 9

Dissecting the previous code, we have the command to clear the screen `[CLS]`, then the Input command asking for user input and storing the `input` as the variable `n$`. The Print at line 30 puts a blank line on the screen, whilst the `Print` command at line 40 displays the message `Hello` and the contents of the variable `n$`.



STEP 10

There's a lot you can do in Immediate Mode; however, to unleash the full potential of FUZE BASIC you're best working in the Program Editor. To enter the Program Editor type in the command new to clear any programs already stored in memory and press the `F2` key. As you can see, Program Editor looks significantly different to Immediate Mode.



Coding with FUZE BASIC – Part 1

Variables are used in programming to store and retrieve data from the computer's memory. It's a specified location in memory that can be referenced by the programmer at any point in the code, as long as it's created and valid.

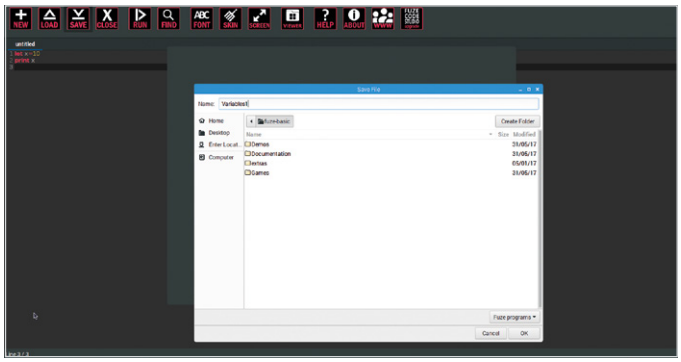
LET THERE BE VARIABLES

We've already looked at assigning some variables in the previous tutorial so let's extend that and see what else we can do with them.

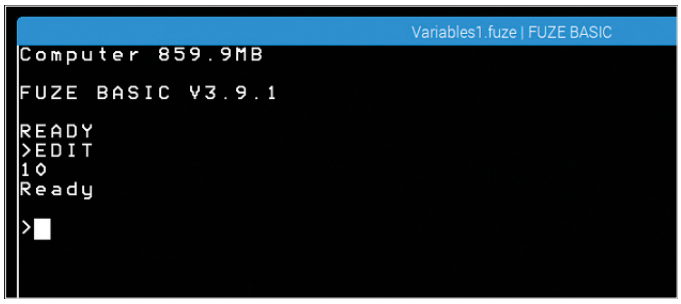
STEP 1 Enter the Program Editor, by pressing the F2 key. Within the Program Editor enter the following, pressing Enter after each line:

```
Let x=10
Print x
```

Now click on the Save button, along the top of the screen and save the program as 'Variables1'. Click the OK button to return to the Editor and the Run button to execute the code.



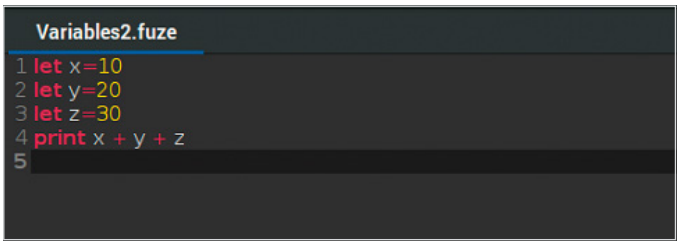
STEP 2 After clicking Run you drop back into Immediate Mode and the display will output the number 10. To break down this simple code, you've created the variable called X, and you've allocated the value 10 to it. The second line simply prints the current value of X – which is of course, 10.



STEP 3 Press F2 to enter Editor mode and click on New. Now let's expand on the simple code. Enter the following:

```
Let x=10
Let y=20
Let z=30
Print x + y + z
```

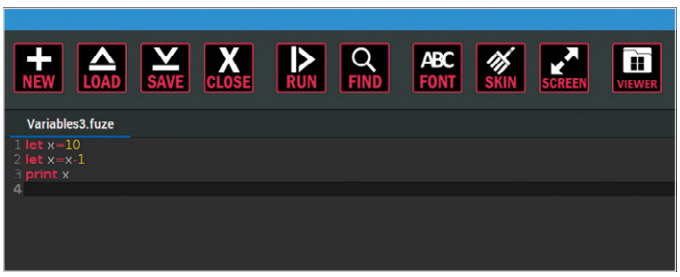
Save as 'Variables2' and Run it. You now have the output of 60 on the screen, as you've assigned X, Y and Z with numerical values, and printed the total.



STEP 4 What if we wanted to change the value of a variable? Enter this listing:

```
Let x=10
Let x=x-1
Print x
```

To begin with X equalled 10 but the next line subtracts 1 making it 9, then prints the current value of X. Imagine this as lives in a game, starting with 10 lives, losing 1 and leaving 9 left.

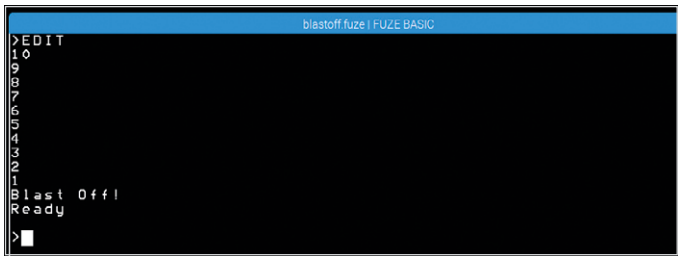


STEP 5

We can extend this further with more commands. Try this:

```
Let x=10
Cycle
Print x
Let x=x-1
Repeat until x=0
Print “Blast Off!”
End
```

This creates a loop that will minus 1 from X until it reaches 0, then prints Blast Off!

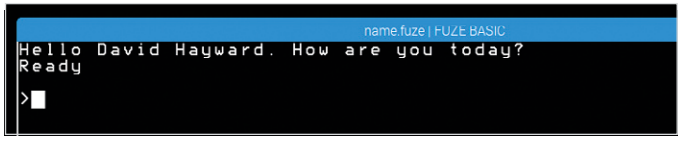


STEP 6

Variables can do more than store numbers:

```
Input “Hello, what is your first name? “, f$
Print
Input “Thanks, and what is your surname? “, s$
Cls
Print “Hello “; f$; “ “; s$; “. How are you today?”
End
```

The variables f$ and s$ store input from the user, then printed it back to them on the same line.

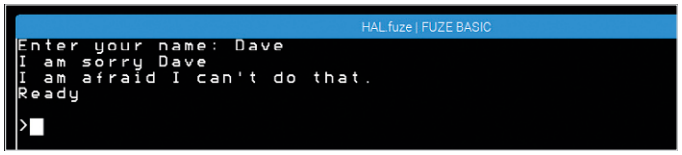


STEP 7

Conditional statements allow you to make your program do different things depending on the user input. For example:

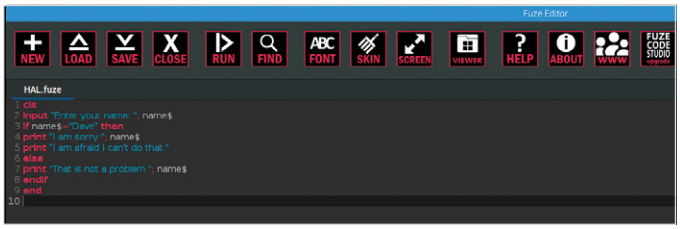
```
cls
Input “Enter your name: “, name$
If name$=”Dave” then
Print “I am sorry “; name$
Print “I am afraid I can’t do that”
Else
Print “That is not a problem “; name$
Endif
End
```

Save as ‘HAL’ and Run.



STEP 8

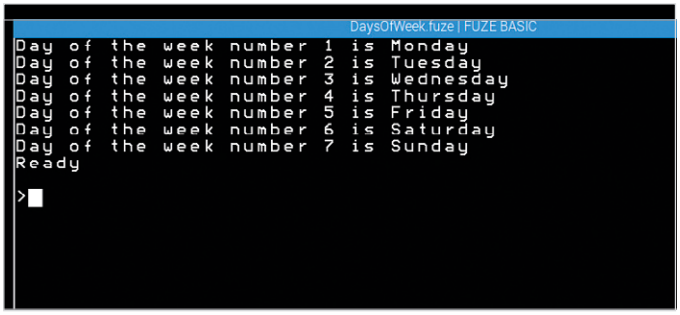
The code from Step 7 introduced some new commands. First we clear the screen, then ask for user input and store it in the variable name $. Line 3 starts the conditional statement, if the user enters the name ‘Dave’ then the program will print HAL’s 2001 infamous lines. If another name is inputted, then it will print something else.



STEP 9

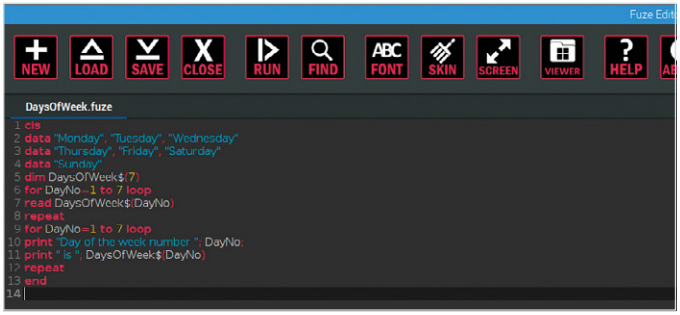
Programs store all manner of information, retrieving it from memory in different ways:

```
cls
Data “Monday”, “Tuesday”, “Wednesday”
Data “Thursday”, “Friday”, “Saturday”
Data “Sunday”
Dim DaysOfWeek$(7)
For DayNo = 1 TO 7 loop
Read DaysOfWeek$(DayNo)
Repeat
For DayNo = 1 TO 7 loop
Print “Day of the week number “; DayNo;
Print “ is “; DaysOfWeek$(DayNo)
Repeat
End
```



STEP 10

The code from Step 9 is beginning to look quite complex, using the Data command to store constant data, creating a variable called DaysOfWeek using the Dim command and assigning it an indexed dimension (7). The code then Reads the stored Data, assigns it a variable dimension from 1 to 7 and prints the result.



Coding with FUZE BASIC – Part 2

Moving on from the previous FUZE BASIC tutorial, let's expand everything you've done so far and see if we can apply it to something other than counting numbers or asking for someone's name. In the grand tradition of BASIC programming, let's create a text adventure.

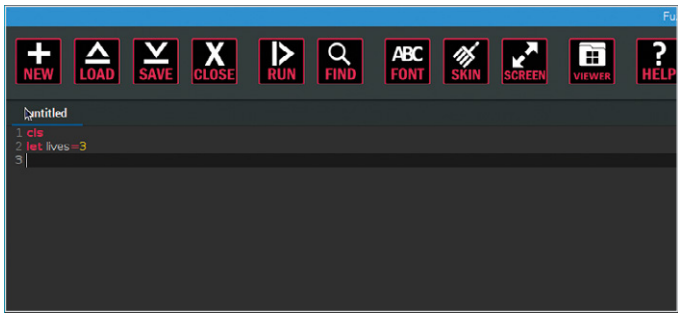
"PALE BULBOUS EYES STARE AT YOU…"

A text adventure game is an ideal genre to explore your BASIC skills in. There are variables, events, user input, counting and if you want, even a few graphics here and there to inject and use.

STEP 1

Enter the Program Editor and begin with a simple clear screen, as it's always a good way to start. What we need to do is set some basic parameters first, so start with the number of lives a player has, for example 3.

```
Cls
Let lives=3
```

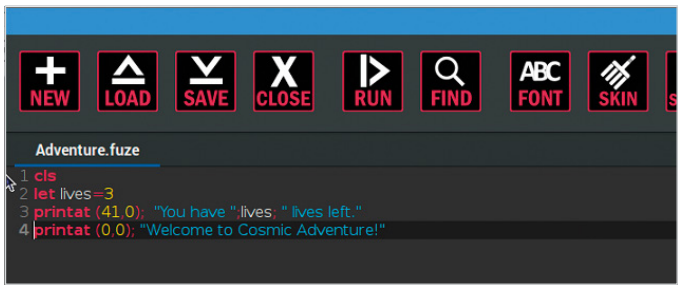


STEP 2

Now you can introduce the game and let the player know how many lives they currently have. You can do this by adding the following to the code:

```
Printat (41,0); "You have "; lives; " lives left."
Printat (0,0); "Welcome to Cosmic Adventure!"
```

The printat command will specify a location on the screen to display the text using x,y.

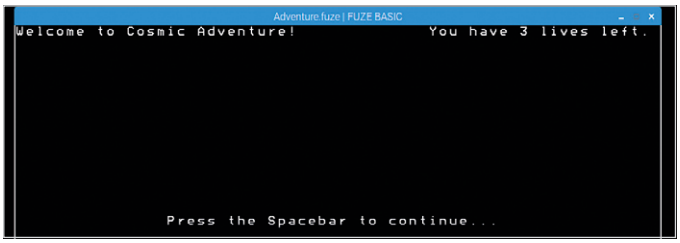


STEP 3

Let's add a way whereby the user is required to press a key to continue, this way you can leave instructions on the screen for an indefinite period:

```
Printat (15,15); "Press the Spacebar to continue…"
While inkey <> 32 cycle
Repeat
```

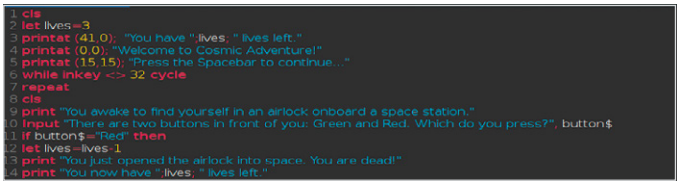
This prints the message whilst waiting for the specific key to be pressed on the keyboard: the Spacebar.



STEP 4

Now we can start the 'story' part of the adventure:

```
Cls
Print "You awake to find yourself in an airlock
onboard a space station."
Input "There are two buttons in front of you:
Green and Red. Which do you press?", button$
If button$="Red" then
Let lives=lives-1
Print "You just opened the airlock into space. You
are dead!"
Print "You now have ";lives; " lives left."
```



STEP 5

Now add:

```
If lives=0 then goto 25
Print “Press the Spacebar to try again.”
While inkey <> 32 cycle
Repeat
Goto 8
Else
Print “The door to the interior of the space station opens, lucky for you.”
```

The Goto command goes to a line number and continues with the code. Here you can use it to start an end of game routine.

```
15 if lives=0 then goto 25
16 Print "Press the Spacebar to try again."
17 while inkey <> 32 cycle
18 repeat
19 goto 8
20 else
21 print "The door to the interior of the space station oper
```

STEP 6

Let’s finish this routine off with:

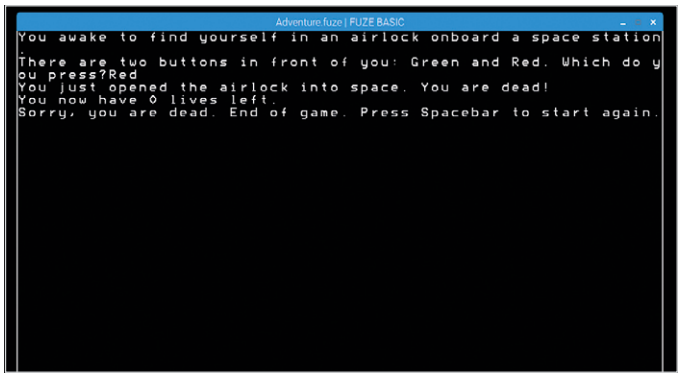
```
Endif
Endif
Goto 29
Print “Sorry, you are dead. End of game. Press Spacebar to start again.”
While inkey <> 32 cycle
Repeat
Goto 1
```

This closes the If statements, then goes to line 29 (if you pressed the Green button) to continue the game, skipping the end of game routine.

```
22 endif
23 endif
24 goto 29
25 print "Sorry, you are dead. End of game. Press Spacebar to start again."
26 while inkey <> 32 cycle
27 repeat
28 goto 1
29
30
```

STEP 7

From line 25 we start the end of game routine as stated on line 15, goto 25. This only works if the variable lives equals 0; the player’s lives have run out. It prints a ‘sorry you are dead’ message and asks to press the Spacebar to start the game all over again from line 1, the goto 1 part.



STEP 8

We can now continue the game from line 29, adding another press the Spacebar routine, followed by a clear screen ready for the next part of the adventure.

```
Print “Press the Spacebar to continue…”
While inkey <> 32 cycle
Repeat
Cls
```

```
29 print "Press the Spacebar to continue..."
30 while inkey <> 32 cycle
31 repeat
32 cls
33
34
```

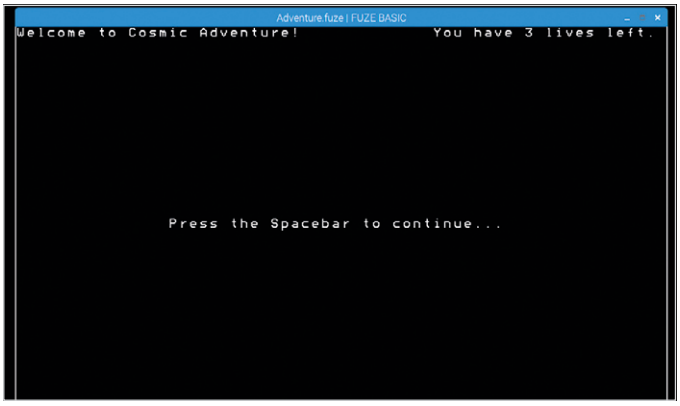
STEP 9

You can now Save the code, call it Adventure (or something), and Run it from the menu. Whilst it’s not the most elegant code you will ever see, it brings in many different elements and shows you what can be done with FUZE BASIC.

```
Adventure.fuze
1 cls
2 let lives=3
3 printat (41,0); "You have ";lives; " lives left."
4 printat (0,0); "Welcome to Cosmic Adventure!"
5 printat (15,15); "Press the Spacebar to continue..."
6 while inkey <> 32 cycle
7 repeat
8 cls
9 print "You awake to find yourself in an airlock onboard a space station."
10 input "There are two buttons in front of you: Green and Red. Which do you press?", button$
11 if button$="Red" then
12 let lives=lives-1
13 print "You just opened the airlock into space. You are dead!"
14 print "You now have ";lives; " lives left."
15 if lives=0 then goto 25
16 Print "Press the Spacebar to try again."
17 while inkey <> 32 cycle
18 repeat
19 goto 8
20 else
21 print "The door to the interior of the space station opens, lucky for you."
22 endif
23 endif
24 goto 29
25 print "Sorry, you are dead. End of game. Press Spacebar to start again."
26 while inkey <> 32 cycle
27 repeat
28 goto 1
29 print "Press the Spacebar to continue..."
30 while inkey <> 32 cycle
31 repeat
32 cls
33
34
```

STEP 10

Before you continue with the adventure, and map the fate of our reluctant space hero, we’re going to improve our code with some graphics. FUZE BASIC has some great graphical commands at its disposal, along with some other useful and interesting extras.



Coding with FUZE BASIC – Part 3

The last tutorial had you creating the foundations for a text-based adventure game. While it works perfectly fine, it would be nice to include some graphics and maybe a few other elements to have it stand out from the usual BASIC programs.

ADDING GRAPHICS

FUZE BASIC employs a variety of different commands to display graphics, either drawn on the screen or by displaying an image file.

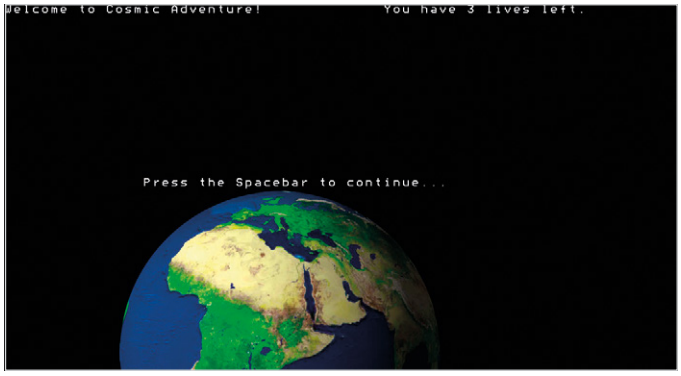
STEP 1 You're going to start by making the game full screen, then adding an appropriate image that sets the theme of the adventure. From line 2 press Enter, to create a new line 3, and type in the following:

```
Fullscreen=1
Spriteindex=newsprite(1)
Earth$=”planetEarth.png”
Loadsprite (earth$, spriteindex, 0)
Plotsprite (spriteindex, 200, 200, 0)
```

Adventure.fuze

```
1 cls
2 let lives=3
3 fullscreen=1
4 spriteindex=newsprite(1)
5 earth$="planetEarth.png"
6 loadsprite (earth$, spriteindex, 0)
7 plotsprite (spriteindex, 200, 200, 0)
```

STEP 2 The code from Step 1 will import and display an image of the Earth; the image itself is already available in the /Desktop/fuze-basic/extras/images folder. It's now classed as a sprite and can be manipulated through the various graphical commands of FUZE BASIC. Any unique images you want to include should be copied to this folder to add to your game.



STEP 3 Now create a new line 13, by getting the cursor to the end of line 12 and pressing Enter. For the new line, type in:

```
Hidesprite (spriteindex)
```

This command will remove the image from the screen, allowing you to include a new image for the next step in the game.

```
10 printat (15,15); "Press the Spacebar to continue..."
11 while inkey <> 32 cycle
12 repeat
13 hidesprite (spriteindex)
14 cls
15 print "You awake to find yourself in an airlock onboard a space station."
```

STEP 4 You may need to source your own images for your game. In our example, we found an image of red and green buttons and copied to the /Desktop/fuze-basic/extras/images folder. Now we need to add it to our code from line 15:

```
buttons$=”buttons.png”
loadsprite (buttons$, spriteindex, 0)
plotsprite (spriteindex, 300, 400, 0)
```

Make sure the image is called before the Input command!

```
14 cls
15 print "You awake to find yourself in an airlock onboard a space station."
16 buttons$="buttons.png"
17 loadsprite (buttons$, spriteindex, 0)
18 plotsprite (spriteindex, 300, 400, 0)
19 input "There are two buttons in front of you: Green and Red. Which do you press?", button$
20 if button$="Red" then
```

STEP 5 Continuing, we can use images of the interior of the ISS if the Green button is pressed. Download the image, put it in the images folder, name it ISS.png and call it from the code whilst hidesprite hides the previous image.

```
Hidesprite (spriteindex)
U;date
Print “The door to space station opens..”
ISS$=”ISS.png”
Loadsprite (ISS$, spriteindex, 0)
Plotsprite (spriteindex, 200, 200, 0)
```

STEP 6 By now your code is getting quite hefty. Don't forget that with each new line you're entering, the original Goto values will be different. It's best to return to the code and update the lines where Goto is referenced.

```
1 CLS
2 LET lives = 3
3 FULLSCREEN = 1
4 spriteindex = newSprite (1)
5 earth$ = "planetEarth.png"
6 loadSprite (earth$, spriteindex, 0)
7 plotSprite (spriteindex, 200, 200, 0)
8 printAt (41, 0);  "You have ";  lives;  " lives left."
9 printAt (0, 0);  "Welcome to Cosmic Adventure!"
10 printAt (15, 15);  "Press the Spacebar to continue..."
11 WHILE INKEY$ <> 32 LOOP
12 REPEAT
13 hideSprite (spriteindex)
14 CLS
15 PRINT "You awake to find yourself in an airlock onboard a space station."
16 buttons$ = "buttons.png"
17 loadSprite (buttons$, spriteindex, 0)
18 plotSprite (spriteindex, 300, 400, 0)
19 INPUT "There are two buttons in front of you: Green and Red. Which do you press?", button$
20 IF button$ = "Red" THEN
21   LET lives = lives - 1
22   PRINT "You just opened the airlock into space. You are dead!"
23   PRINT "You now have ";  lives;  " lives left."
24   IF lives = 0 THEN GOTO 39
25   PRINT "Press the Spacebar to try again."
26   WHILE INKEY$ <> 32 LOOP
27   REPEAT
28   GOTO 13
29 ELSE
30   hideSprite (spriteindex)
31   UPDATE
32   PRINT "The door to the interior of the space station opens, lucky for you."
33   ISS$ = "ISS.png"
34   loadSprite (ISS$, spriteindex, 0)
35   plotSprite (spriteindex, 200, 200, 0)
36 ENDIF
37 ENDIF
38 GOTO 43
39 PRINT "Sorry, you are dead. End of game. Press Spacebar to start again."
40 WHILE INKEY$ <> 32 LOOP
41 REPEAT
42 GOTO 1
43 PRINT "Press the Spacebar to continue..."
44 WHILE INKEY$ <> 32 LOOP
45 REPEAT
46 CLS
```

STEP 7 Additionally we can add an image for the End of Game routine and insert the code from line 39:

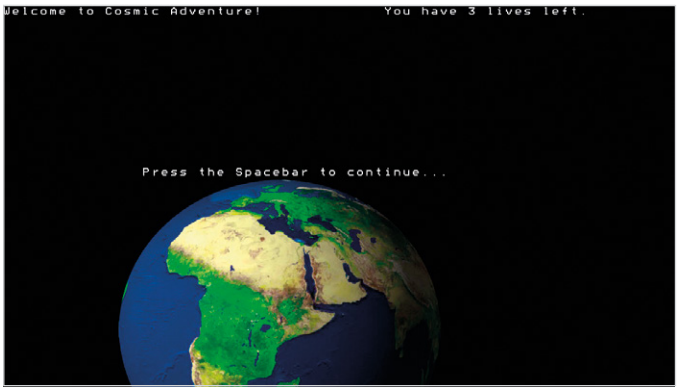
```
Print "Sorry, you are dead."
Gameover$="gameover.png"
Loadsprite (gameover$, spriteindex, 0)
Plotsprite (spriteindex, 200, 200, 0)
While inkey <> 32 cycle
Repeat
Hidesprite (spriteindex)
Goto 1
```

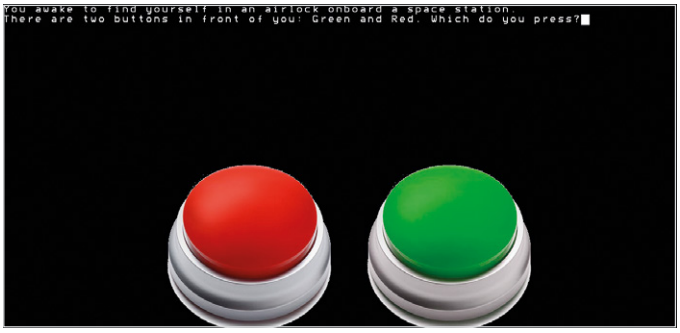
```
39 print "Sorry, you are dead."
40 gameover$="gameover.png"
41 loadsprite (gameover$, spriteindex, 0)
42 plotsprite (spriteindex, 200, 200, 0)
43 print " End of game. Press Spacebar to start again."
44 while inkey <> 32 cycle
45 repeat
46 hidesprite (spriteindex)
47 goto 1
```

STEP 8 Once more, the code has now expanded and as such you need to ensure that any reference to another line is updated to reflect the new numbering; especially lines 24 and 38, which call either End of Game routine or continue the game if the Green Button has been pressed.

```
cls
let lives=3
fullscreen=1
spriteindex=newsprite(1)
earth$="planetEarth.png"
loadsprite (earth$, spriteindex, 0)
plotsprite (spriteindex, 200, 200, 0)
printat (41,0);  "You have ";lives; " lives left."
printat (0,0); "Welcome to Cosmic Adventure!"
printat (15,15); "Press the Spacebar to continue..."
while inkey <> 32 cycle
repeat
hidesprite (spriteindex)
cls
print "You awake to find yourself in an airlock onboard a space station."
buttons$="buttons.png"
loadsprite (buttons$, spriteindex, 0)
plotsprite (spriteindex, 300, 400, 0)
Input "There are two buttons in front of you: Green and Red. Which do you press?", button$
if button$="Red" then
let lives=lives-1
print "You just opened the airlock into space. You are dead!"
print "You now have ";lives; " lives left."
if lives=0 then goto 39
Print "Press the Spacebar to try again."
while inkey <> 32 cycle
repeat
goto 13
else
hidesprite (spriteindex)
update
print "The door to the interior of the space station opens, lucky for you."
ISS$="ISS.png"
loadsprite (ISS$, spriteindex, 0)
plotsprite (spriteindex, 200, 200, 0)
endif
endif
goto 52
print "Sorry, you are dead."
gameover$="gameover.png"
loadsprite (gameover$, spriteindex, 0)
plotsprite (spriteindex, 200, 200, 0)
print " End of game. Press Spacebar to start again."
while inkey <> 32 cycle
repeat
```

STEP 9 Naturally you can continue with Cosmic Adventure yourself, adding choices, graphics and keeping tabs on the number of lives and whatever else you can think of. As we said, it's not the most elegant code and it's as far from a triple-A game as you can imagine; but at least it's given you a head start with FUZE Basic.





STEP 10 Here's a recap of the images we've used for the graphics in our adventure game. The FUZE BASIC manual comes with countless more commands to make better use of the system, so read through it and expand on what you've learned here.

Using a Breadboard

A great way to learn circuits is to use a breadboard. You can use a breadboard with FUZE BASIC, or Scratch and Python, to control LEDs and other simple circuits. Here we'll show you how a breadboard works.

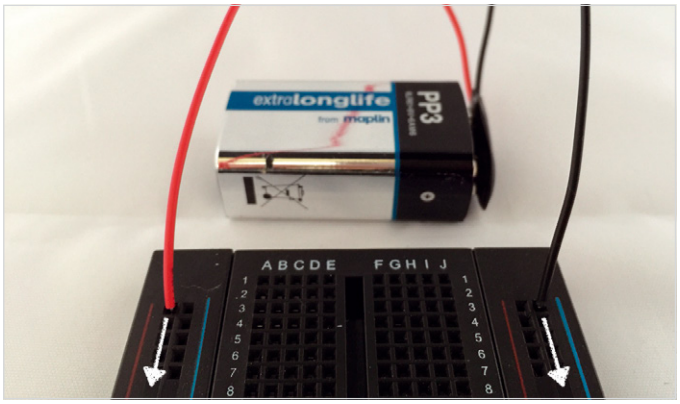
GPIO

The Raspberry Pi enables you to access electronic pins, known as GPIO (General Purpose Input and Output). These are used to interact with external electronics like LED lights and switches. Below you'll learn to build circuits using a Breadboard.

STEP 1 The FUZE Workstation comes with a breadboard and some basic electronics components - you can follow along with this tutorial by getting a breadboard, 1 x blue and 1 x red breadboard wires, a 5mm LED, a 22Ohms 5% resistor, 9V battery, and a 9V snap battery clip. Your local electronics shop will help you out.

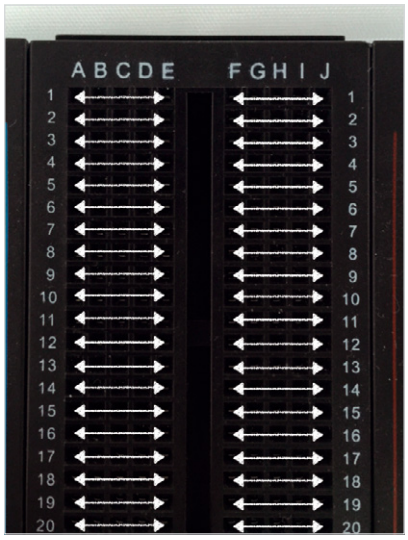
STEP 2 Get out the breadboard, hold it up vertically and take a good look at it. You should see four vertical columns. The two pairs, on the left and right, both have a red and blue line running vertically alongside them. In the middle are vertical columns with letters and numbers. There are typically two main columns, lettered A-E and F-J.

STEP 3 The red and blue lines are power rails: red is for positive and blue is for negative. The holes do not provide any power themselves; instead they just connect to each other. So if you plug an item into one hole, and another item into a connected hole (along the line), then the two are connected as if you'd physically joined the two things together.

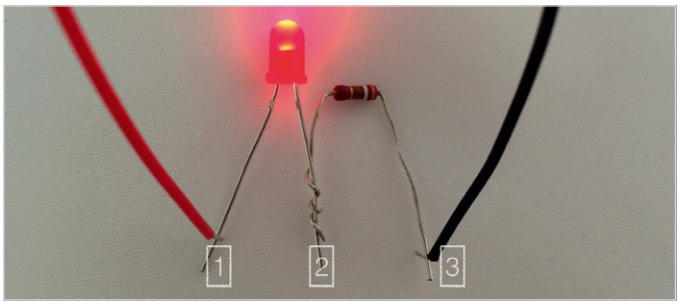


STEP 4 The two columns of holes on the outside are connected all the way down the line from the top to the bottom. Take a 9V battery and attach a snap battery clip. Connect the positive wire (red) to the topmost red hole on the left, it will provide positive power to any wire or component connected in any red hole all the way down to the bottom. Add the blue (negative) wire to the topmost blue hole on the right.

STEP 5 The two columns on the inside of the breadboard work completely differently. They are not wired vertically, but horizontally along the row of each columns. So if you look at row 1, the holes marked A, B, C, D and E are connected; and the holes in rows F, G, H, I and J are connected. What do we mean by "connected"? Let's do it physically first to find out.



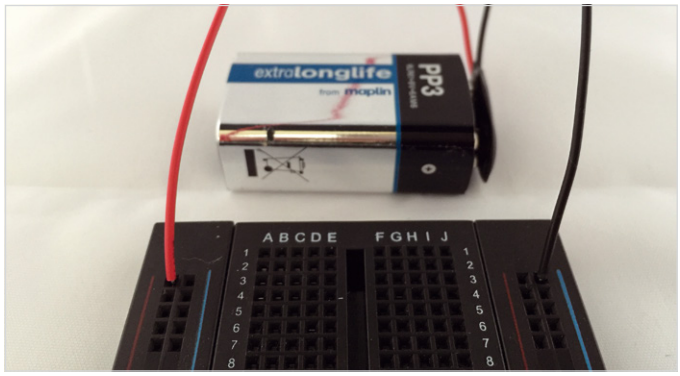
STEP 6 Get the LED and look at it closely. Notice how one leg is longer than the other. That's the positive wire; the shorter one is negative. Take a resistor and wrap one end of it around the shorter wire on the LED. Take the positive wire from the PP3 battery clip and touch the LED; touch the negative wire to the resistor and see the LED light up. We've numbered these 1, 2 and 3 so you can match them in the next steps.



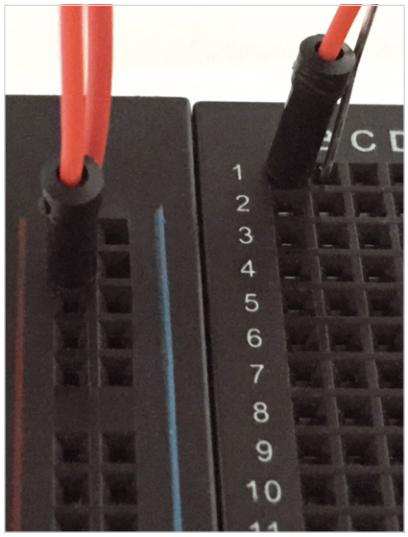
RECREATING THIS IN A BREADBOARD

Wrapping wires and circuits around each other isn't going to be much fun, especially when you're trying to figure out how something works. That's what a breadboard is for: the holes enable you to connect one item to another.

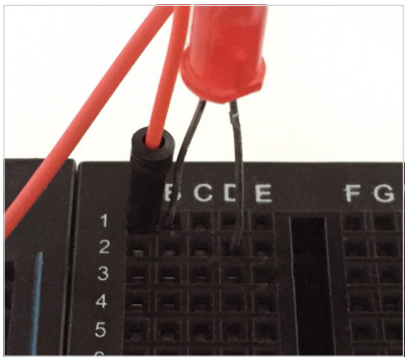
STEP 1 Let's now recreate this simple LED circuit on a breadboard. With the positive and negative cables from the battery connected to the top of the power rails, take a red connector and slot one end into a hole on the red line, and the other end into hole A1.



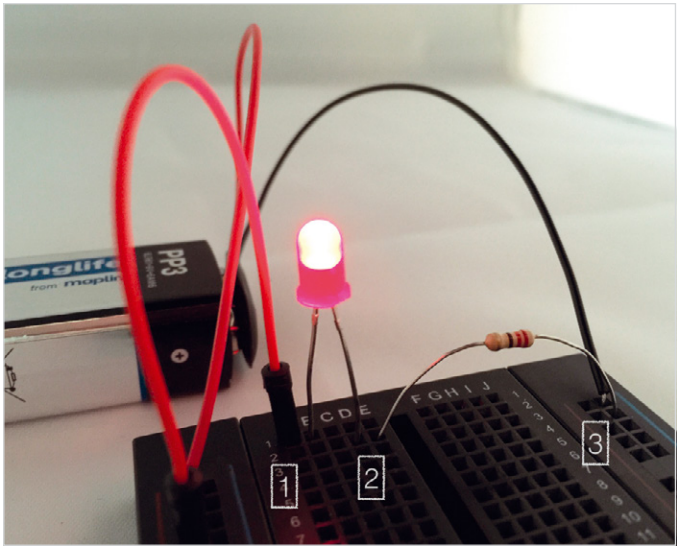
STEP 2 Now take the LED, find the long end, and slot this into hole B1. This is the equivalent of number 1 in our physical connection. The red cable is connected vertically along the powerline, and then to row 1 on the breadboard where it is connected horizontally to the LED on row 1.



STEP 3 This is the part where most people mess up. Take the other leg of the LED and connect it to hole D2. This is the next row down. If you connected it to another hole on line 1, such as D1, it would be the equivalent of touching both LED legs together.



STEP 4 Now take your resistor and place it next to the LED leg in slot E2 (also on the second line). If you look at the photo from Step 6, this is the equivalent of 2 (the part where the LED and resistor are connected. Insert the other end of the resistor in a hole on the negative power rail and your LED will light up.



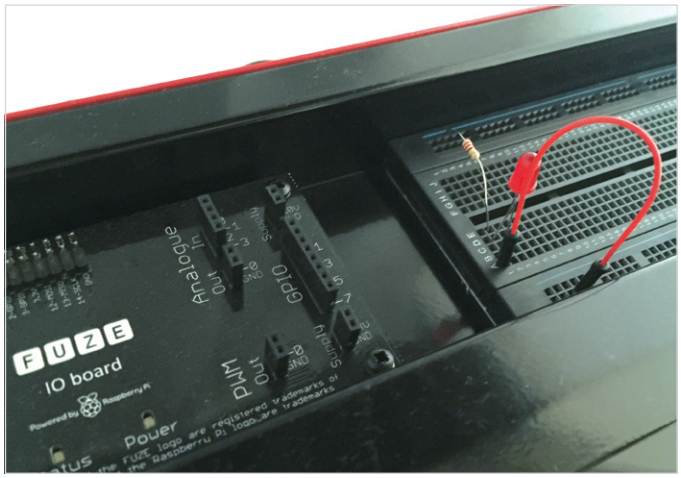
Using the FUZE IO Board

So far our breadboard hasn't been connected to the FUZE or Raspberry Pi in any way, but all that's about to change. We're now going to remove the battery and slot our breadboard into the FUZE.

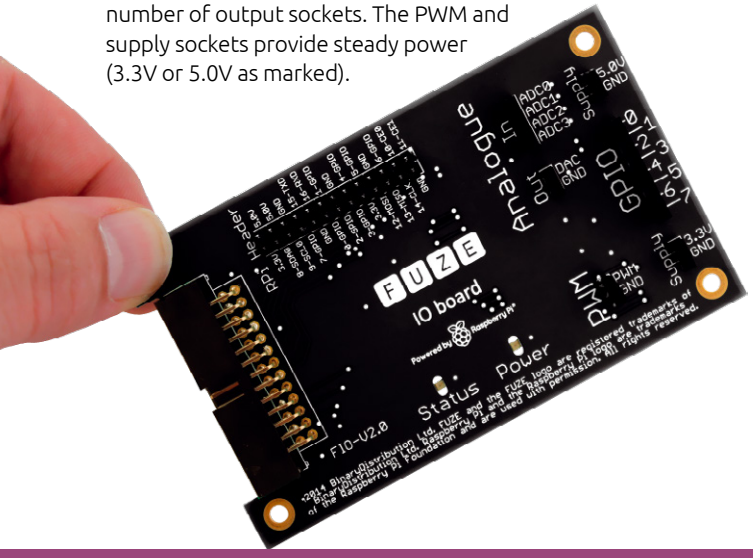
GPIO PINS

The Raspberry Pi's GPIO pins act as a physical interface between the Raspberry Pi and electronic items. On the FUZE these are safely connected to the IO board, and can then be connected to your breadboard.

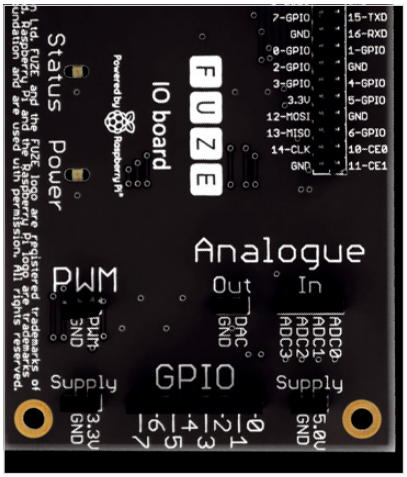
STEP 1 Remove the 9V battery and battery clip if it is still connected to the breadboard, and slide the breadboard into the top of the FUZE so the wires and LED are near the IO board. Let's take a closer look at what the IO board has to offer.



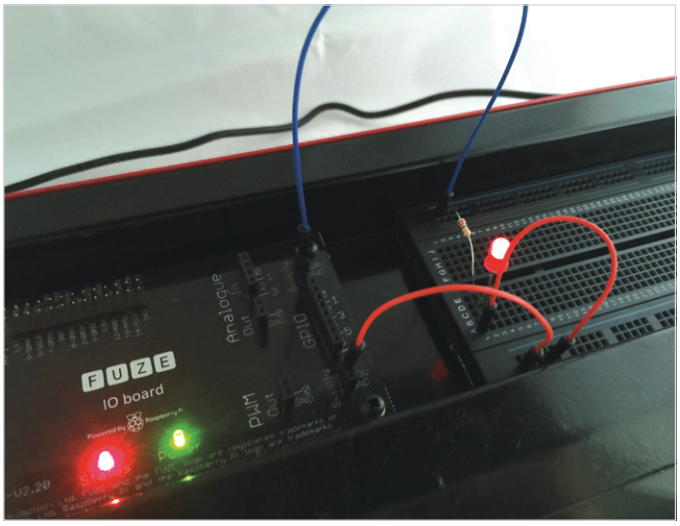
STEP 2 If you look closely at the IO board you'll see a section of pins marked "RPI Header". These match the pins that are on your Raspberry Pi. On the right side of the board are a number of output sockets. The PWM and supply sockets provide steady power (3.3V or 5.0V as marked).



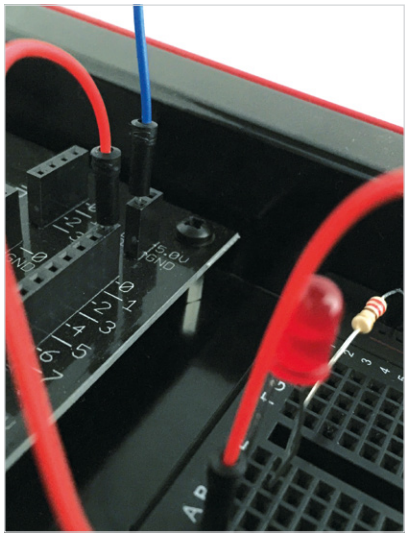
STEP 3 The pins marked "GPIO" and numbered 1-7 are more interesting. These can be turned on or off from inside programs, or at the command line. When turned on they provide 3.3V, and when off they provide nothing. These On/Off switches can be used to activate and deactivate components you attach to the Raspberry Pi.



STEP 4 Start by connecting a blue cable to the pin marked GND in the Supply section on the top right of the FUZE IO board. Connect the other end to the leftmost hole in the blue rail, now running along the top of the breadboard. Connect a red cable to the socket marked 3.3V on the supply section in the bottom right of the IO board. Connect the other end of the cable to first hole in the red rail running along the bottom of the breadboard. The LED will come on.



STEP 5 This isn't any different to what we had before, so let's spice things up. Remove the red cable from the 3.3V IO socket and connect it to the socket marked 0 underneath GPIO. The LED will turn off. This is because this socket won't provide any power until we tell it to.



STEP 6 Start FUZE BASIC and enter:

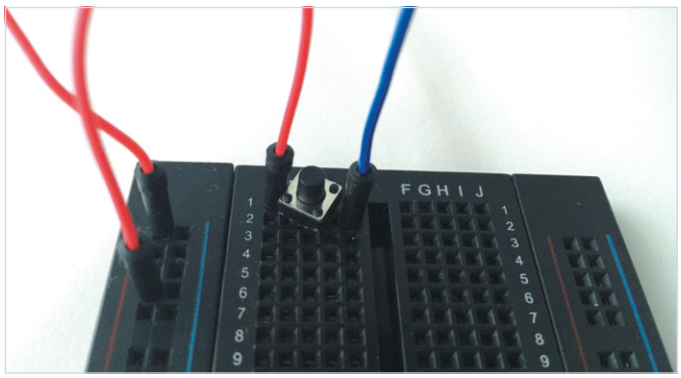
```
PinMode (0, 1)
DigitalWrite (0, 1)
```

The LED turns on. The first part, PinMode, tells the Raspberry Pi that GPIO 0 is going to be used, and the 1 part says it will be output. The DigitalWrite command sets GPIO 0 on. Enter DigitalWrite (0, 0) to turn the LED off.

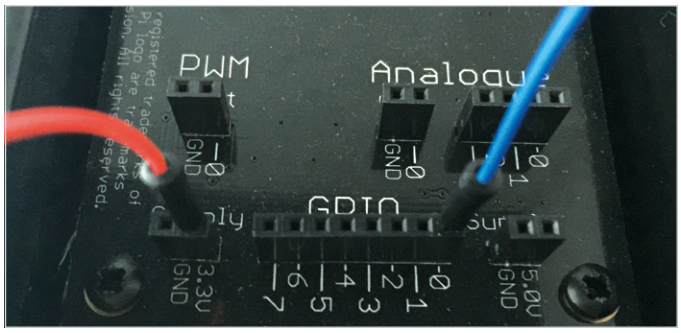
GETTING INPUT

We're now really steaming along. Our Raspberry Pi-powered FUZE is turning on LED lights in the outside world. Next we need to look at input; how we can get information from our breadboard to our Raspberry Pi.

STEP 1 Remove the LED and resistor from the breadboard and remove the GPIO 0 and GND cables. Place the Push button switch in the same place as the LED (B1 and D2) and place the blue cable in the hole next to it (E2). Take a look at the photo if you need help placing the items in the right holes.



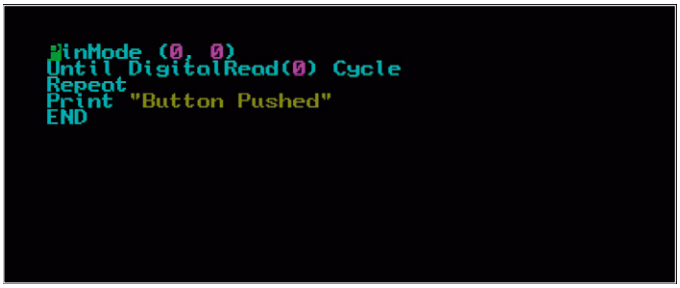
STEP 2 Now take the blue cable in E2 and connect the other end to GPIO 0. Finally, connect the red cable from the first hole in the power rail to 3.3V. Our circuit is complete. Current will go from the 3.3V to the power rail, and from the power rail to our switch. The switch is connected to our blue cable, which connects to GPIO.



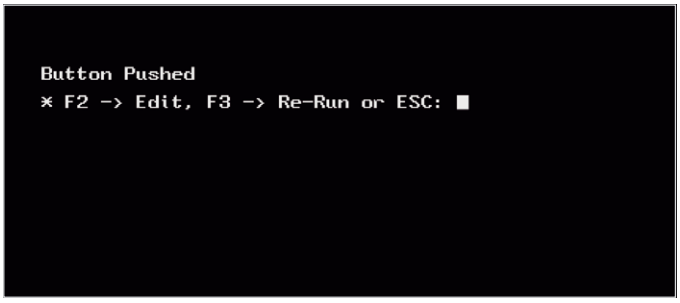
STEP 3 Press F2 to open the Program Editor. Enter the following program:

```
PinMode (0, 0)
Until DigitalRead(0) Cycle
Repeat
Print "Button Pushed"
```

Press F3 to run the program.



STEP 4 Here's what happens. Power is flowing from the 3.3V socket to the switch where it stops. Meanwhile our program has set GPIO 0 to 0 (input mode) and a Cycle Repeat loop is waiting until input comes through on 0 (via DigitalRead). When we push the button a connection is made, power flows to GPIO 0 and it alerts the program. It then prints the message "Button Pushed".



Using a Robot Arm with FUZE BASIC

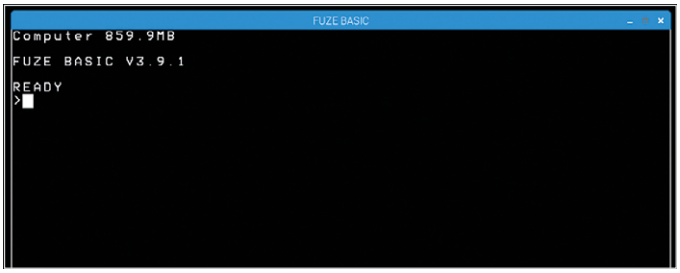
As part of the educational kit, the FUZE Workstation can be purchased with an accompanying robot arm. This is a 149 piece kit-form robotic arm, that requires assembly and is powered by four D-type batteries. It's connected to the RPi or FUZE via a USB cable and is also Windows compatible.

I, ROBOT

We won't go into the construction of the robot arm here, the instructions which come with the arm are easy to follow and it can be completed and ready for use within a couple of hours or so. Let's look at how to get it working.

STEP 1 The robot arm is one of the first external hardware components that was released and fully compatible with the Raspberry Pi; as such, it's an excellent project to get into, from the construction of the arm itself, to operating it via the FUZE Workstation.

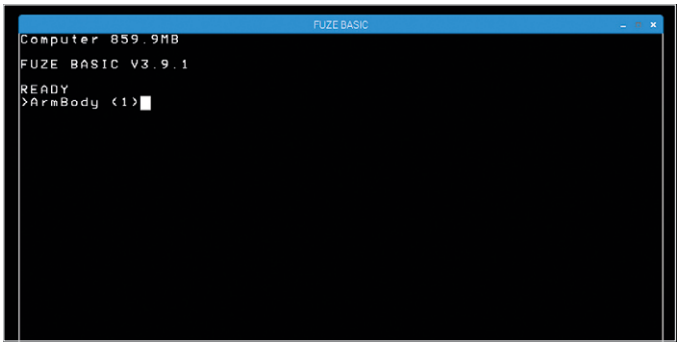
STEP 2 Start by plugging the robot arm into one of the spare USB ports on the back of the FUZE workstation. Ensure that the arm has its batteries correctly in place and that its power switch is On. Now open FUZE BASIC and remain in the Immediate Mode.



STEP 3 To begin with, let's look at a few commands to make the robot arm move. In Immediate Mode, in FUZE BASIC, enter:

```
ArmBody (1)
```

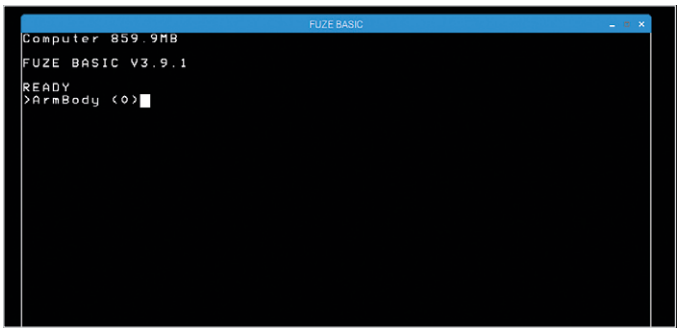
This starts the arm rotating clockwise (looking down on it).



STEP 4 Once the arm begins to rotate clockwise it will get to the limit of its range and start clicking. When it starts this quickly enter the command:

```
ArmBody (0)
```

This will stop the arm from moving.



STEP 5

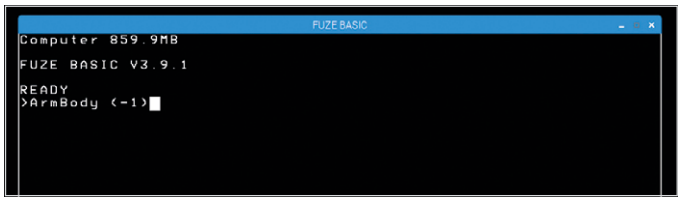
Now enter:

```
ArmBody (-1)
```

This will start moving the arm anti-clockwise. Again, when it starts to click enter the command:

```
ArmBody (0)
```

To stop it from moving.



STEP 6

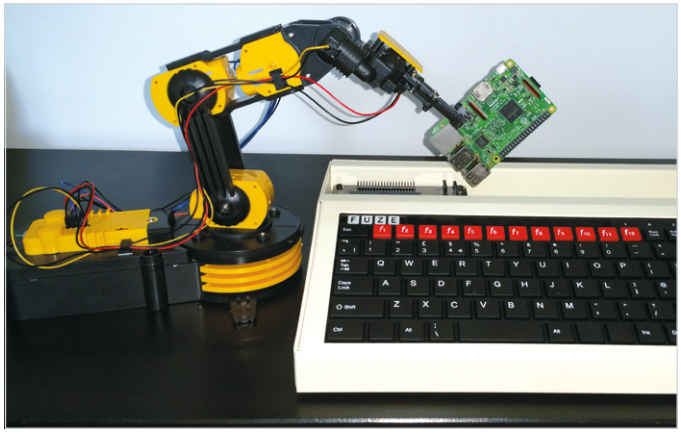
The other commands to make the arm move are:

```
ArmShoulder (x) – where x can be 1, -1 or 0
ArmElbow (x) – where x can be 1, -1 or 0
ArmWrist (x) – where x can be 1, -1 or 0
ArmGripper (x) – where x can be 1 -1, or 0
ArmLight (x) – where x can be 1 or 0
```

Note: you can press the up arrow key to re-enter the previously typed commands, so you can quickly stop the arm's movement when it reaches its limit.

STEP 7

Let's create a program allowing you to move the arm around freely. There are some new commands here: PROC and DEF PROC, that enables BASIC to jump to a PROCedure, another part of the program, then back with ENDPROC. FONTSCALE determines the size of the on-screen print display and HVTAB is an X and Y coordinate system to print on-screen.

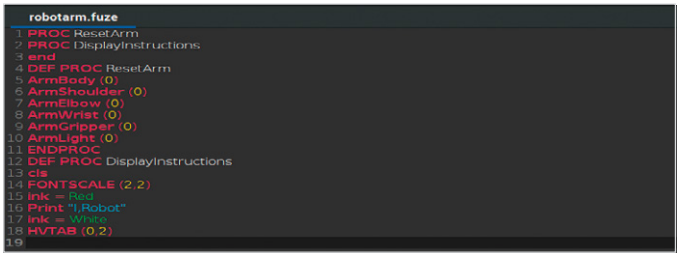


STEP 8

Press F2, and type in the following:

```
PROC ResetArm
PROC DisplayInstructions
End
DEF PROC ResetArm
ArmBody (0)
ArmShoulder (0)
ArmElbow (0)
ArmWrist (0)
ArmGripper (0)
ArmLight (0)
ENDPROC
DEF PROC DisplayInstructions
CLS
FONTSCALE (2,2)
Ink = Red
Print "I, Robot"
Ink = White
HVTAB (0,2)
```

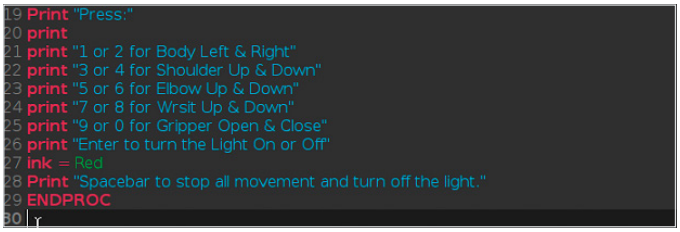
This is the start of the program, resetting the arm and preparing the on-screen display.



STEP 9

Now to expand the program to control the arm:

```
Print "Press:"
Print
Print "1 or 2 for Body Left & Right"
Print "3 or 4 for Shoulder Up & Down"
Print "5 or 6 for Elbow Up & Down"
Print "7 or 8 for Wrist Up & Down"
Print "9 or 0 for Gripper Open & Close"
Print "Enter to turn the Light On or Off"
Ink = Red
Print "Spacebar to stop all movement and turn off the light."
ENDPROC
```



STEP 10

Now we need to process the user input. There's a lot here but type the content as shown in the screenshot. Save the code and Run; you can now control the robot arm using the number keys 1 to 0, the Enter key for the light and Spacebar to reset everything.

FUZE BASIC Examples – Part 1

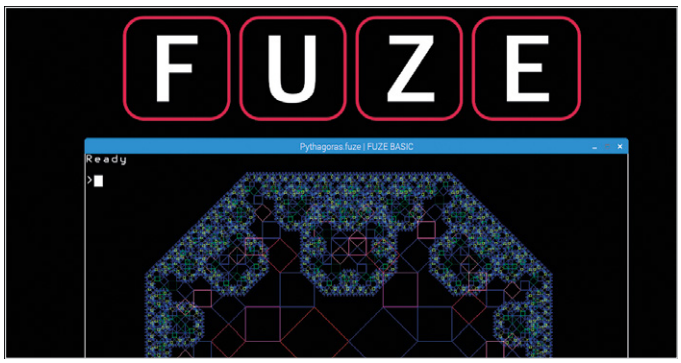
FUZE BASIC has an impressive following of coding experts and enthusiasts who have selflessly provided their code for others to learn from and use. These stalwarts of FUZE BASIC have forged some amazingly detailed examples, which we'll take a look at over the coming pages.

CODE REPOSITORY

From fractal generators to encryption programs and animation, here are ten excellent examples of what others have done with a little patience and a lot of code.

FRACTALS

James Cook's Tree of Pythagoras is an impressive fractal, constructed from squares, that looks remarkably complex but takes up surprisingly little code. You can find it at **www.fuze.co.uk/code-repository**; just enter the code and Run it to be amazed.



CARDIOIDS

Simon Plouffe is a Mathematician who, back in the '70s, created some incredible cardioid images by dividing a circle into prime parts and drawing lines based on mathematical spaced points on the circumference. Anyway, enter this and be amazed:

```
untitled
FULLSCREEN = 1
updateMode = 0
multiplier = 0
m = 200
LOOP
  CLS2
  FOR i = 0 TO m LOOP
    rgbColour (255 - (i / m) * 255, 125, (i / m) * 255)
    x = COS ((i / m) * 360) * (gHeight * 0.8 / 2) + (gWidth * 0.5)
    y = SIN((i / m) * 360) * (gHeight * 0.8 / 2) + (gHeight * 0.5)
    nx = COS (((i + 1) / m) * 360) * (gHeight * 0.8 / 2) + (gWidth * 0.5)
    ny = SIN(((i + 1) / m) * 360) * (gHeight * 0.8 / 2) + (gHeight * 0.5)
    tx = COS (((i * multiplier) / m) * 360) * (gHeight * 0.8 / 2) + (gWidth * 0.5)
    ty = SIN (((i * multiplier) / m) * 360) * (gHeight * 0.8 / 2) + (gHeight * 0.5)
    LINE (x, y, nx, ny)
    LINE (x, y, tx, ty)
  REPEAT
  PRINT multiplier
  UPDATE
  multiplier = multiplier + 0.001
REPEAT
END
```

SCROLLING IMAGES

This code will load any image and make it scroll across the screen from right to left. Put your image either in the /extras/images folder or simply in the same folder as the code itself. Save and Run and enjoy the image moving across the screen. See if you can modify it to full screen, or more.

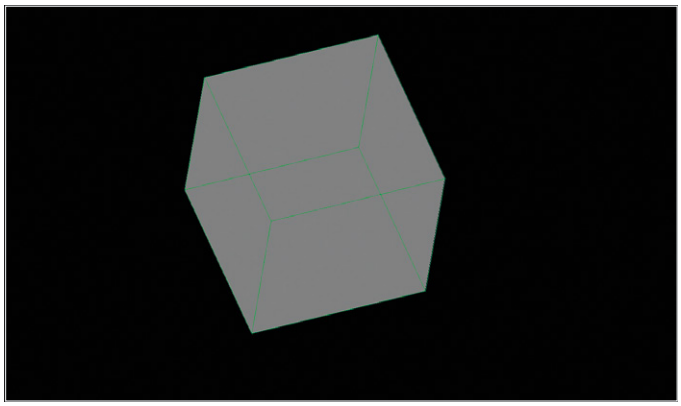
```
Scrolling.fuze
image = loadImage ("monet.png") // load a picture
plotImage (image, 0, 0) // put it on the creen
pixels = 2 // set how many pixels to scroll by
LOOP // start main loop
screen = grabRegion (0, 0, pixels, gHeight) // grab the left of the screen
scrollLeft (0, 0, gWidth, gHeight, pixels) // scroll left by pixel
plotImage (screen, gWidth - pixels, 0) // dump the grab on the right side to make it loop
freeImage (screen) // release the grab from memory
UPDATE // perform a screen refresh
//WAIT (0.005) // adjust this to slow-down or speed-up
REPEAT // do it again, and again and again and...
```

SHOOTER

In the Program Editor click the Load button and browse through the folder /fuze-basic/Demos until you find Shooter.fuze. With the program listing loaded, scroll down to line 247 and change the "player2.png" entry to "Player2.png" – adding a capital P. This is a basic side-scrolling shooter and you've just fixed an error in the code.

3D BOX

While in the same folder as Shooter.fuze, look for Box.fuze. This is a great animation program that displays a rotating three dimensional box on the screen until you press the Escape key. It's a fantastic learning resource and with a little time you can bend it to your will and use it in your own code.



BEAT 'EM UP

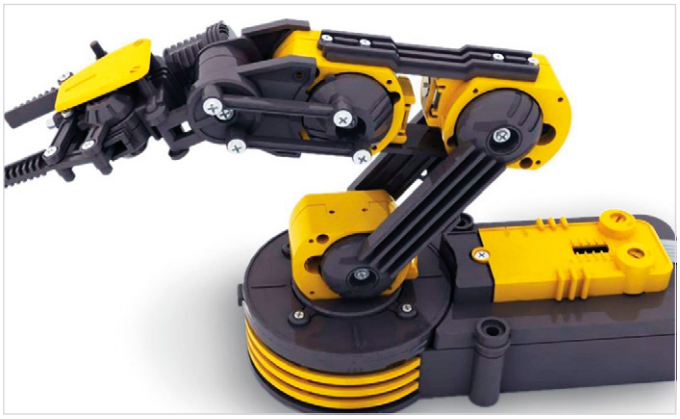
FuzeFighter, also found in the Games folder, is another prime example of what can be done with FUZE BASIC. There's in-game music, sound effects, animations, collision detection, scoring and a two-player element that can be worked into your own routines.

DOGS IN SPACE

Dogs in Space is a fun little game (found in /fuze-basic/Demos) that features in-game music, sprite animation, collision detection, scoring and keyboard controls. Whilst it may not amuse you for too long, it's benefit lies in the code examples that you can turn to your own future programs.

ROBOT CONTROL

The Robot.fuze file, in the Games folder, is an extension to the previous tutorial's robot movement BASIC program that you entered. However, this time there's graphics and animations to help improve the process and make it a more flexible (excuse the pun) program.



SPACE INVADERS

Click the Load button, and browse to /fuze-basic/Games. Open the file silv.fuze and have a look through the 784 lines of code before clicking the Run button. It's quite complex but when you run it you can see why. Those of a certain age will no doubt recall spending a fortune on Space Invaders in the arcades!

SNAKE

Finally, snake.fuze is a good example of a combination of programming elements. Graphics, scoring, collision detection and some interesting routines to help improve your overall program can be found within this code.

FUZE BASIC Examples – Part 2

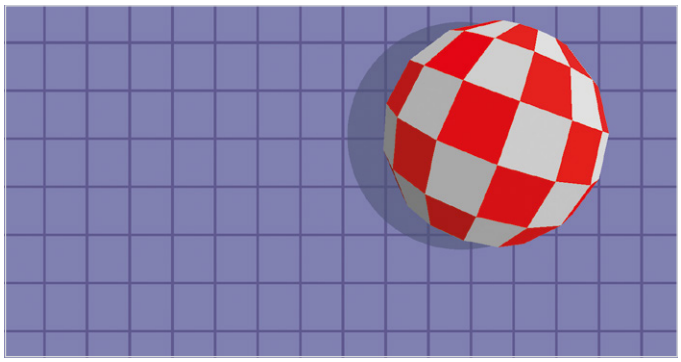
Continuing from the previous pages, here are ten more excellent examples of what can be done with FUZE BASIC. Take what you want from the code, alter it and insert it into your own routines to fine-tune your program.

CODE STRIPPING

Many of the legendary programmers from the golden era of home computing stripped the code from snippets posted in the magazines of the time. They bent the code to their will and created something as close to magic as possible.

AMIGA BALL

Amiga owners will have fond memories of their futuristic computer back in the late '80s and early '90s. The Amiga was a pretty impressive home computer, even by today's standards and its iconic Bouncing Ball routine will forever be remembered by those who grew up with one. Load up aball.fuze from Demos and see what you can use.

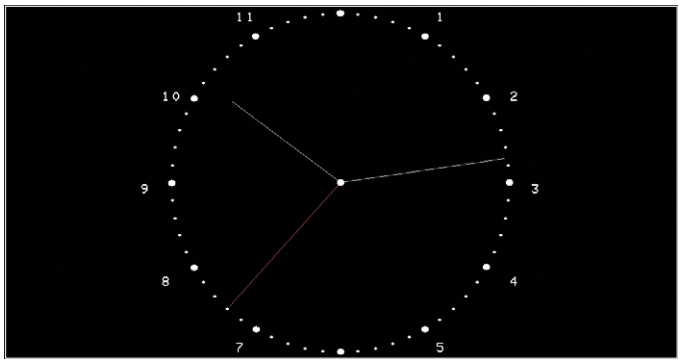


KEYBOARD INPUT

Scankeyboard.fuze is an extremely handy bit of code to load up. It's a simple program that will display the key pressed on the keyboard, which is a great resource when it comes to creating keyboard interactions with the user and the program, such as a game. Just take the key codes you need and insert them in your own programs.

ANALOGUE CLOCK

Whilst an analogue clock on the screen, complete with second hand, may not sound too interesting, there's a surprising amount of useful code within this particular routine. Clock.fuze is in the Demos folder and once loaded up you can strip all manner of handy code snippets from it.



BBC MICRO::BIT

This code snippet will look for and detect any attached BBC micro:bit or Arduino compatible devices that you've attached to the FUZE IO or Raspberry Pi GPIO pins. It's incredibly handy for helping you create the code behind your hardware project.

```
devices=DETECTDEVICES
CLS
IF devices = FALSE THEN
PRINT "No devices found"
END
ELSE
PRINT devices; " devices found"
PRINT
FOR ID = 0 TO devices LOOP
IF DEVICETYPE(ID) = 1 THEN
PRINT "BBC micro:bit :ID="; ID
ENDIF
IF DEVICETYPE(ID) = 2 THEN
PRINT "Arduino or compatible device :ID="; ID
ENDIF
REPEAT
END
```

FONTS

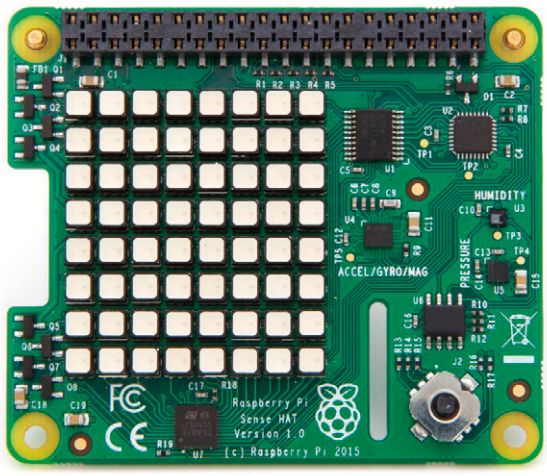
This little snippet of code, although simple, will display some of the available font sizes of FUZE BASIC. The maximum size is 20, so alter as you wish:

```
CLS
FOR size = 1 TO 7 LOOP
INK = RND(30)
FONTSIZE( size )
PRINT “Hello”
REPEAT
END
```

RPI SENSEHAT

If you’re working with a Raspberry Pi SenseHAT, then the following code will return the current value of the HAT’s accelerometer:

```
CLS
LOOP
PRINT “Sense Accelerometer X=”; SENSEACCELX
PRINT “Sense Accelerometer Y=”; SENSEACCELY
PRINT “Sense Accelerometer Z=”; SENSEACCELZ
REPEAT
END
```



HAT COMPASS

And this code will return the value of the SenseHAT’s compass:

```
CLS
LOOP
PRINT “Sense Compass X=”; SENSECOMPASSX
PRINT “Sense Compass Y=”; SENSECOMPASSY
PRINT “Sense Compass Z=”; SENSECOMPASSZ
REPEAT
END
```

untitled

```
1 CLS
2 LOOP
3 PRINT "Sense Compass X="; SENSECOMPASSX
4 PRINT "Sense Compass Y="; SENSECOMPASSY
5 PRINT "Sense Compass Z="; SENSECOMPASSZ
6 REPEAT
7 END
```

MOUSE CONTROL

If you want to incorporate mouse pointer and button use in your code, then this will certainly help you out. It’s a fairly simple bit of code but remarkably effective and it can easily be inserted into your own programs.

```
1 CLS
2 LOCKMOUSE(TRUE)
3 MOUSEON
4 COLOUR = WHITE
5 RECT (100, 100, 150, 50, TRUE)
6 INK = BLACK
7 PAPER = WHITE
8 PLOTTEXT ("Click Me", 105, 125);
9 UPDATE
10 clicked = FALSE
11 LOOP
12 GETMOUSE (x, y, z)
13 IF z <> 0 THEN
14 IF (x > 100 AND x < 250) THEN
15 IF (y > 100 AND y < 150) THEN
16 clicked = TRUE
17 ENDIF
18 ENDIF
19 ENDIF
20 REPEAT UNTIL clicked
21 SETMOUSE (GWIDTH / 2, GHEIGHT / 2)
22 LOCKMOUSE(FALSE)
23 END
```

JOYSTICK CONTROL

Including the use of a gamepad or joystick in your games or code is a great addition to also being able to redefine the keyboard. This code will detect and display the states of each of the axis and buttons.

```
1 UPDATEMODE = 0
2 FONTSIZE (3)
3 SETUPGAMEPAD (0)
4 LOOP
5 CLS2
6 INK = ORANGE
7 PRINTAT (0, 0); "The detected joystick has"
8 INK = BRIGHTGREEN
9 PRINTAT (0, 1); "Buttons: ";
10 INK = YELLOW
11 PRINT NUMBUTTONS (0) - 1
12 INK = BRIGHTGREEN
13 PRINTAT (0, 2); "Analog Sticks & Triggers: ";
14 INK = YELLOW
15 PRINT NUMAXES (0)
16 INK = BRIGHTGREEN
17 PRINTAT (0, 3); "Digital pads: ";
18 INK = YELLOW
19 PRINT NUMHATS (0)
20 INK = ORANGE
21 PRINTAT (0, 5); "Button states"
22 FOR but = 0 TO NUMBUTTONS (0) - 1 LOOP
23 INK = BRIGHTGREEN
24 PRINTAT (but * 3, 6); "#"; but
25 INK = YELLOW
26 PRINTAT (but * 3, 7); GETBUTTON (0, but)
27 REPEAT
28 INK = ORANGE
29 PRINTAT (0, 9); "Analog states"
30 FOR ax = 0 TO NUMAXES (0) - 1 LOOP
31 INK = BRIGHTGREEN
32 PRINTAT (ax * 8, 10); "#"; ax
33 INK = YELLOW
34 PRINTAT (ax * 8, 11); GETAXIS (0, ax)
35 REPEAT
36 INK = ORANGE
37 PRINTAT (0, 13); "Digital pad states"
38 FOR hat = 0 TO NUMHATS (0) - 1 LOOP
39 INK = BRIGHTGREEN
40 PRINTAT (hat * 3, 14); "#"; hat
41 INK = YELLOW
42 PRINTAT (hat * 3, 15); GETHAT (0, hat)
43 REPEAT
44 UPDATE
45 REPEAT
46 PRINT
47 END
```

REACTION TIMER

Finally, if you’re after something a little competitive with your family, then load up reaction.fuze from the Demos folder. When run, this code will test your reaction time by hitting the Spacebar when indicated. See how fast you can get it, and see if you can hack the code.

```
Simple reaction timer
=====================

Press the SPACE bar to start. There will
then be a short delay, then the word GO
will be printed - at that point, you need
to press the SPACE bar again...

Press SPACE to start: Wait for it... Go!
Your reaction time is 420 milliseconds
Ready
```

Coding with Windows 10 Batch Files

Did you know that Windows has its own built-in scripting language? Batch files have been around since the early days of Windows and while they are overshadowed by the might of the modern Windows graphical user interface, they are still there and still just as capable as they were thirty years ago.

Batch file programming is a skill that system administrators still use, so it's worth spending a bit of time learning how they work and what you can do with them. This section introduces batch files and covers user interactions, variables, loops and even a batch file quiz game to inject an element of fun.

What is a Batch File?

The Windows batch file has been around since the early days of DOS, and was once a critical element of actually being able to boot into a working system. There's a lot you can do with a batch file but let's just take a moment to see what one is.

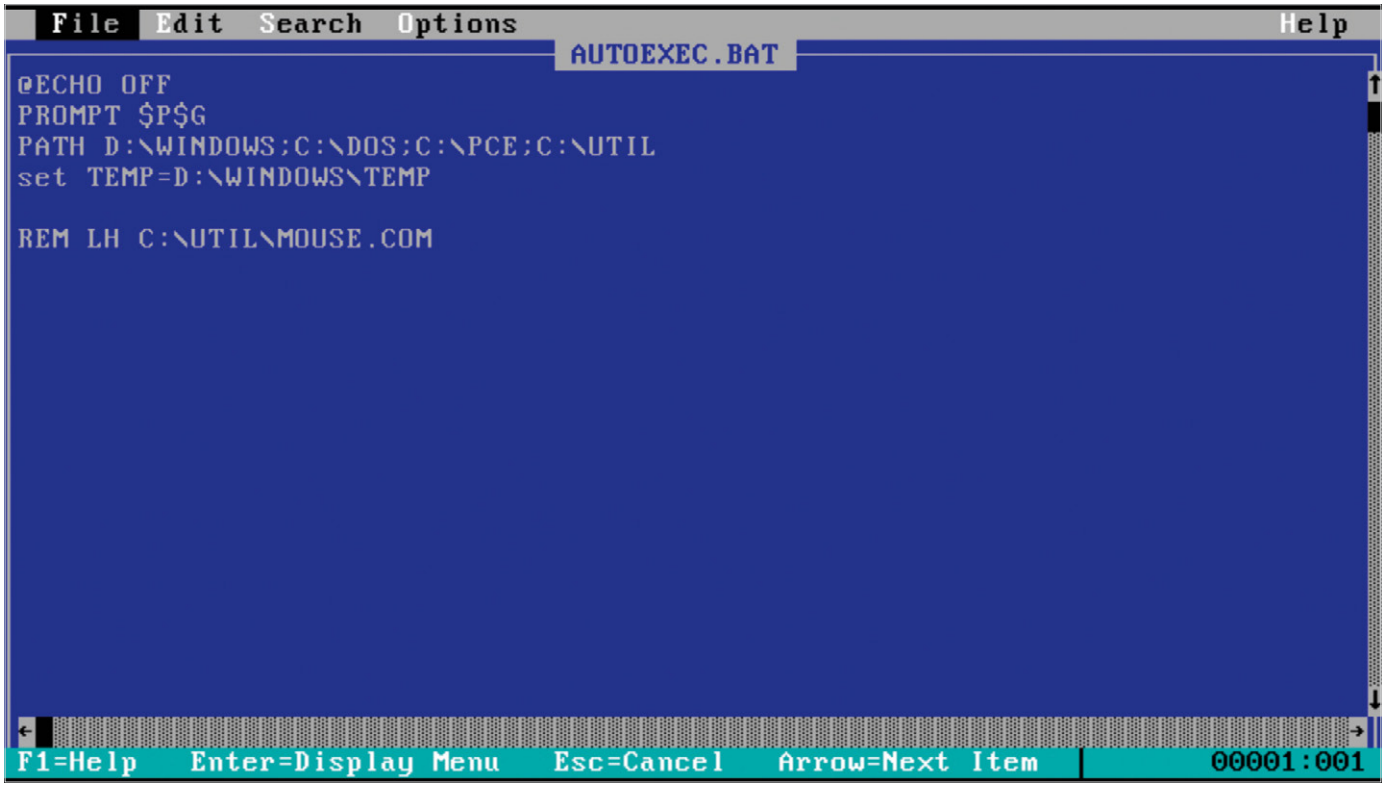
.BAT MAN

A Windows batch file is simply a script file that runs a series of commands, one line at a time, much in the same fashion as a Linux script. The series of commands are executed by the command line interpreter and stored in a plain text file with the .BAT extension; this signifies to Windows that it's an executable file, in this case, a script.

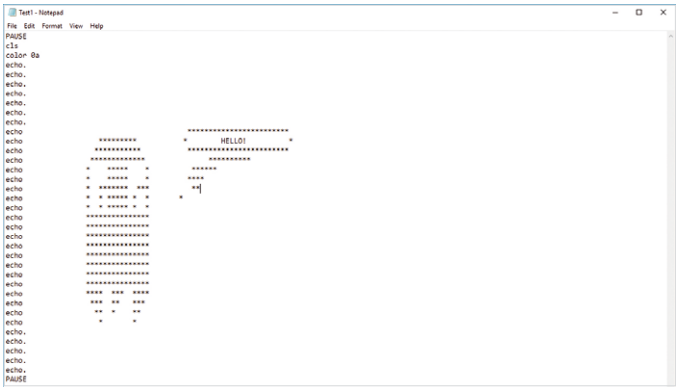
Batch files have been around since the earliest versions of Microsoft DOS. Although not exclusively a Microsoft scripting file, batch files are mainly associated with Microsoft's operating systems. In the early days, when a PC booted into a version of DOS (which produced a simple command prompt when powered up), the batch file was used in the form of a system file called Autoexec.bat. Autoexec.bat was a script that automatically executed (hence Autoexec) commands once the operating system had finished dealing with the Config.sys file.

When a user powered up their DOS-based computer, and once the BIOS had finished checking the system memory and so on, DOS would look to the Config.sys file to load any specific display requirements and hardware drivers, allocate them a slot in the available memory, assign any memory managers and tell the system where the Command.com file, which is the command line interpreter for DOS, was. Once it had done that, then the Autoexec.bat file took over and ran through each line in turn, loading programs that would activate the mouse or optical drive into the memory areas assigned by the Config.sys file.

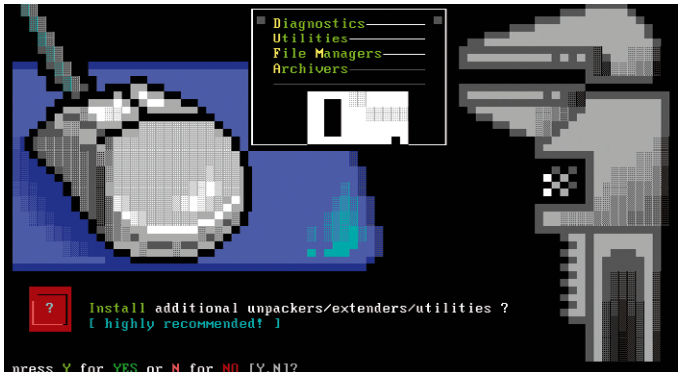
The DOS user of the day could opt to create different Autoexec.bat files depending on what they wanted to do. For example, if they wanted to play a game and have as much memory available as possible, they'd create a Config.sys and Autoexec.bat set of files that loaded the bare minimum of drivers and so on. If they needed



The Autoexec.bat file was a PC user's first experience with a batch file.



Batch files are plain text and often created using Notepad.



Batch files were often used as utility programs, to help users with complex tasks.

access to the network, an Autoexec.bat file could be created to load the network card driver and automatically gain access to the network. Each of these unique setups would be loaded on to a floppy disk and booted as and when required by the user.

The Autoexec.bat was the first such file many users came across in their PC-based computing lives; since many had come from a 16-bit or even 8-bit background; remember, this was the late eighties and early nineties. The batch file was the user's primary tool for automating tasks, creating shortcuts and adventure games and translating complex processes into something far simpler.

Nowadays however, a batch file isn't just for loading in drivers and such when the PC boots. You can use a batch file in the same way as any other scripting language file, in that you can program it to ask for user input and display the results on the screen; or save to a file and even send it to a locally or network attached printer. You can create scripts to back up your files to various locations, compare date stamps and only back up the most recently changed content as well as program the script to do all this automatically. Batch files are remarkably powerful and despite them not being as commonly used as they were during the older days of DOS, they are still there and can be utilised even in the latest version of Windows 10; and can be as complex or simple as you want them to be.

So what do you need to start batch file programming in Windows? Well, as long as you have Windows 10, or any older version of Windows for that matter, you can start batch file programming immediately. All you need is to be able to open Notepad and get to the command prompt of Windows. We show you how it all works, so read on.

BATCH FILE POWER

Just like any other programming interface that can directly interrogate and manipulate the system, batch files require a certain amount of care when programming. It's hard to damage your system with a batch file, as the more important elements of the modern Windows system are protected by the User Account Control (UAC) security; UAC works by only allowing elevated privileges access to important system files. Therefore if you create a batch file that somehow deletes a system file, the UAC activates and stop the process.

However, if you're working in the command prompt with elevated privileges to begin with, as the Administrator, then the UAC won't question the batch file and continue regardless of what files are being deleted.

That said, you're not likely to create a batch file that intentionally wipes out your operating system. There are system controls in place to help prevent that; but it's worth mentioning as there are batch files available on the Internet that contain malicious code designed to create problems. Much like a virus, a rogue batch file (when executed with Administrator privileges) can cause much mayhem and system damage. In short, don't randomly execute any batch file downloaded from the Internet as an Administrator, without first reviewing what it does.

You can learn more about batch files in the coming pages, so don't worry too much about destroying your system with one. All this just demonstrates how powerful the humble batch file can be.

You can create complex batch files or simple ones that display ASCII images on screen.

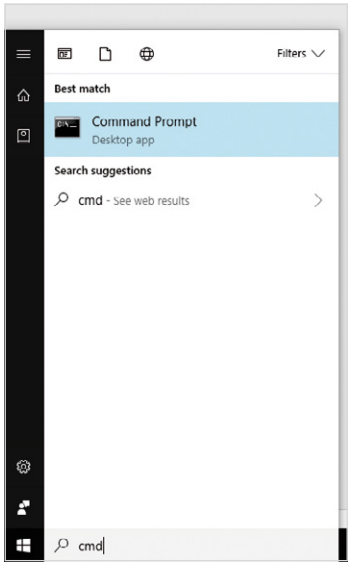
Getting Started with Batch Files

Before you begin to program with batch files, there are a few things you need to know. A batch file can only be executed once it has the .bat extension and editing one with Notepad isn't always straightforward.

A NEW BATCH

Throughout this section on batch files we're going to be working with Notepad, the command prompt and within a folder called 'Batch Files'. To begin with, let's see how you get to the Windows command prompt.

STEP 1 The Windows command prompt may look a little daunting to the newcomer but it's simply another interface (or Shell) used to access the filesystem. You can go anywhere you like in the command prompt, as you would with the graphical interface. To begin, click on the Windows Start button and enter CMD into the search box.

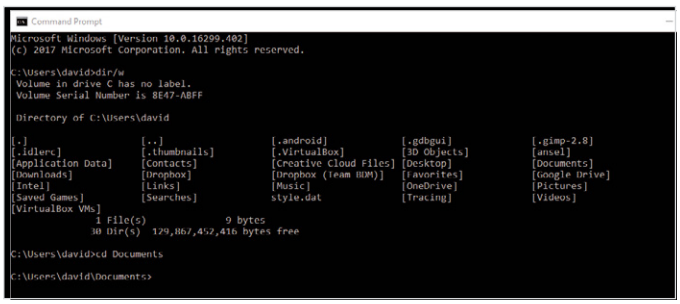


STEP 2 Click on the search result labelled Command Prompt (Desktop App) and a new window pops up. The Command Prompt window isn't much to look at to begin with but you can see the Microsoft Windows version number and copyright information followed by the prompt itself. The prompt details the current directory or folder you're in, together with your username.

STEP 3 While at the command prompt window, enter: `dir/w.` This lists all the files and directories from where you are at the moment in the system. In this case, that's your Home directory that Windows assigns every user that logs in. You can navigate by using the CD command (Change Directory). Try:

```
cd Documents
```

Then press Return.



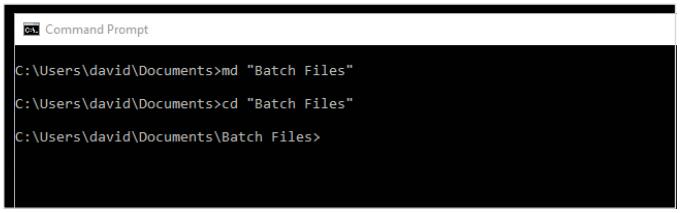
STEP 4 The prompt should change and display `\Documents>;` this means you're in the Documents directory. Now, create a new directory call Batch Files. Enter:

```
md "Batch Files"
```

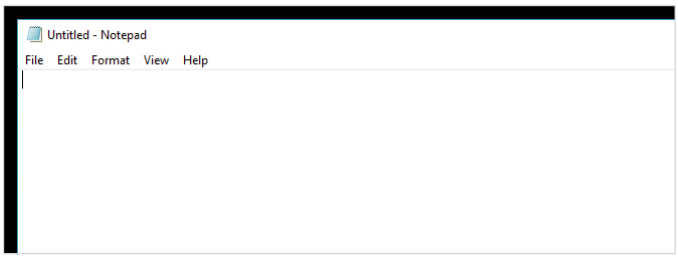
You need the quotations because without them, Windows creates two directories: Batch and Files. Now change directory into the newly created Batch Files.

```
cd Batch Files
```

You won't need the quotes to change directories.



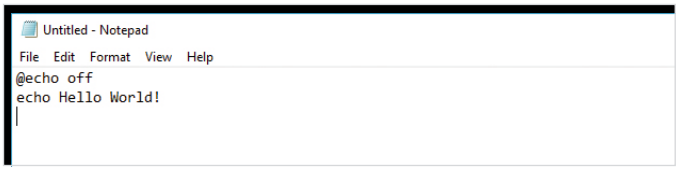
STEP 5 Now that you have the directory set up, where you store your batch files, here is how you can create one. Leave the command prompt window open and click on the Windows Start button again. This time enter Notepad and click on the search result to open the Notepad program. Notepad is a simple text editor but ideal for creating batch scripts with.



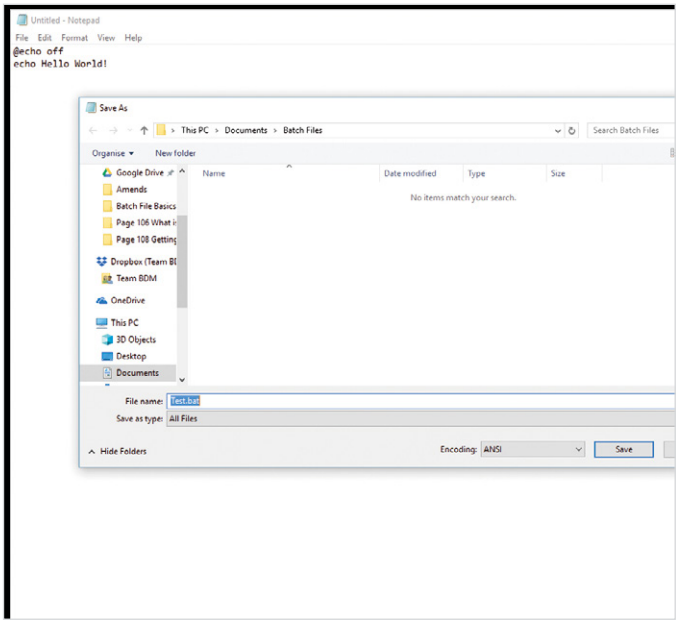
STEP 6 To create your first batch file, enter the following into Notepad:

```
@echo off
echo Hello World!
```

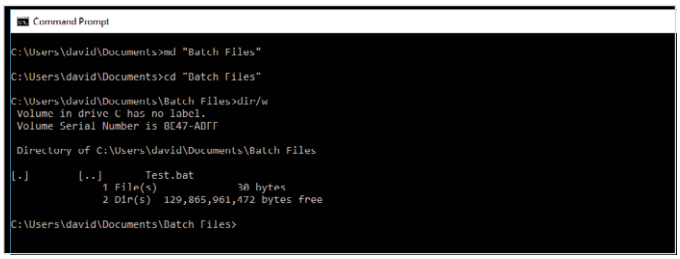
By default, a batch file displays all the commands that it runs through, line by line. What the @echo off command does is turn that feature off for the whole script; with the '@' (at) sign to apply that command to itself.



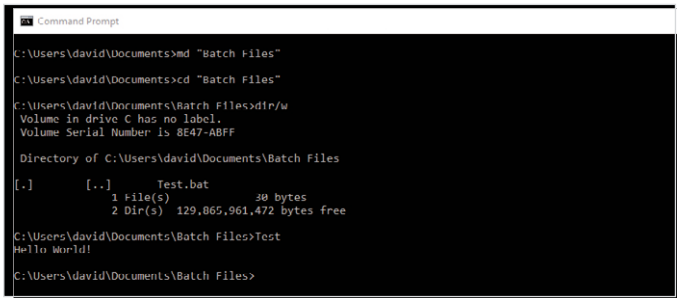
STEP 7 When saving anything in Notepad the default extension is .txt, to denote a text file. However, you want the extension to be .bat. Click on File > Save As and navigate to the newly created Batch Files directory in Documents. Click the drop-down menu Save as Type, and select All Files from the menu. In File Name, call the file **Test.bat**.



STEP 8 Back at the command prompt window, enter: `dir/w` again to list the newly created Test.bat file. By the way, the /w part of dir/w means the files are listed across the screen as opposed to straight down. Enter `dir` if you want (although you need more files to view) but it's considered easier to read with the /w flag.



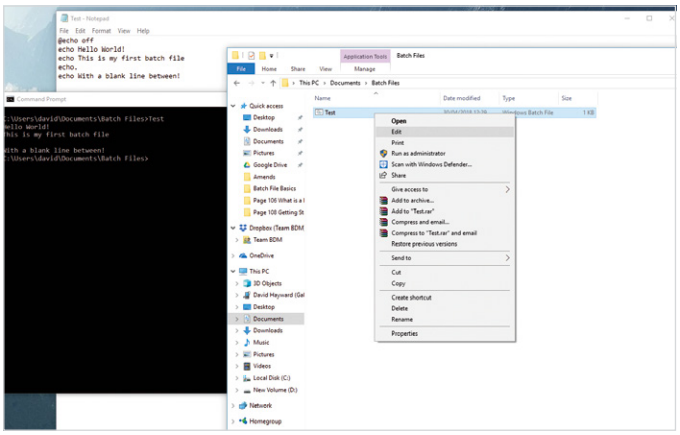
STEP 9 To execute the batch file you've just created, simply enter its name, Test, in the command prompt window. You don't need to add the .bat part, as Windows recognises it as an executable file, and the only one with that particular name in the current directory. Press return and see how you're greeted with Hello World! in the command prompt.



STEP 10 The echo command displays whatever is after it to the screen. Right-click the Test.bat file from Windows Explorer and select Edit to add more echo commands if you like. Try this:

```
@echo off
echo Hello World!
echo This is my first batch file
echo.
echo With a blank line between!
```

Remember to save each new change to the batch file.



Getting an Output

While it's great having the command prompt window display what you're putting after the echo command in the batch file, it's not very useful at the moment, or interactive for that matter. Let's change up a gear and get some output.

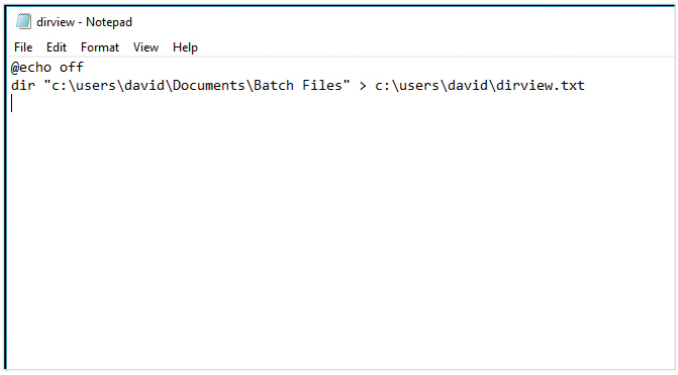
INPUT OUTPUT

Batch files are capable of taking a normal Windows command and executing them, while also adding extra options and flags in to the equation.

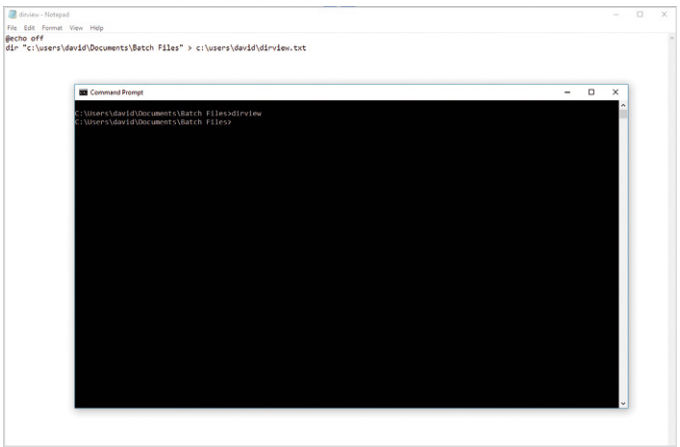
STEP 1 Let's keep things simple to begin with. Create a new batch file called 'dirview.bat', short for Directory View. Start with the @echo off command and under that add:

```
dir "c:\users\YOURNAME\Documents\Batch Files" > c:\users\YOURNAME\dirview.txt
```

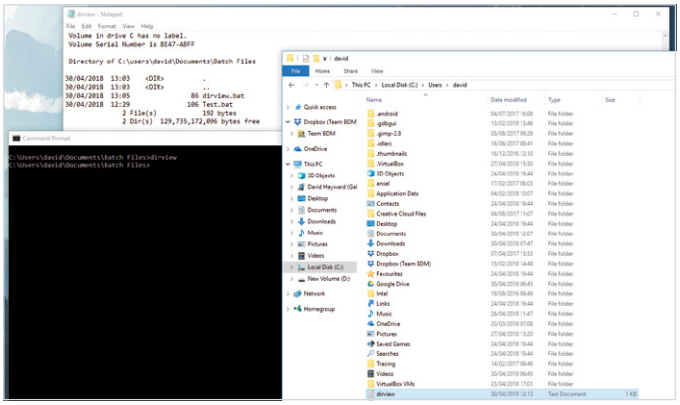
Substitute YOURNAME with your Windows username.



STEP 2 The new line uses the dir command to list the contents of the directory Batch Files, in your Home directory, dumping the output to a text file called `dirview.txt` in the root of your Home directory. This is done, so that the Windows UAC doesn't require elevated permissions, as everything is in your own Home area. Save and run the batch file.



STEP 3 You have no doubt noticed that there is no indication that the batch file worked as there's no meaningful output on the screen. However, if you now open Explorer and browse to `c:\Users\YOURNAME`, remembering to substitute `YOURNAME` with your Windows username, and double-click the `dirview.txt` file, you can see the batch file's output.



STEP 4 If you want to automate the task of opening the text file that contains the output, add the following line to the batch file:

```
notepad.exe c:\users\YOURNAME\dirview.txt
```

Save the file and once again run from the command prompt. This time, it creates the output and automatically opens Notepad with the output contents.

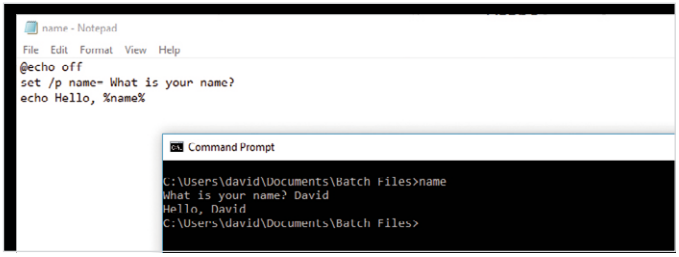
OUTPUT WITH VARIABLES

Variables offer a more interesting way of outputting something to the screen and create a higher level of interaction between the user and the batch file. Try this example below.

STEP 1 Create a new batch file and call it **name.bat**. Start with the `@echo off` command, then add the following lines:

```
set /p name= What is your name?
echo Hello, %name%
```

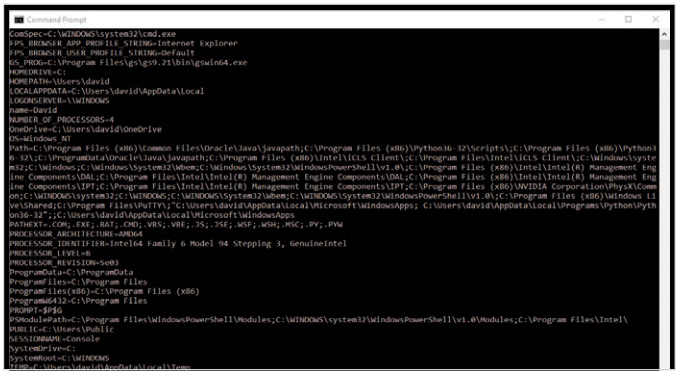
Note: there's a space after the question mark. This is to make it look neater on the screen. Save it and run the batch file.



STEP 2 The set **/p** name creates a variable called name, with the **/p** part indicating that an '**=prompt string**' is to follow. The Set command displays, sets or removes system and environmental variables. For example, while in the command prompt window enter:

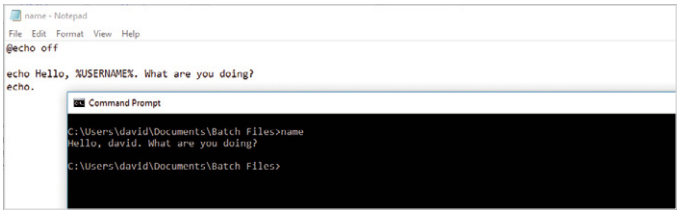
```
set
```

To view the current system variables. Note the name= variable we just created.



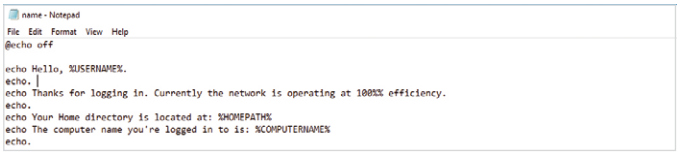
STEP 3 Variables stored with Set can be called with the **%VARIABLENAME% syntax**. In the batch file, we used the newly created %name% syntax to call upon the contents of the variable called name. Your username, for example, is stored as a variable. Try this in a batch file:

```
echo Hello, %USERNAME%. What are you doing?
```

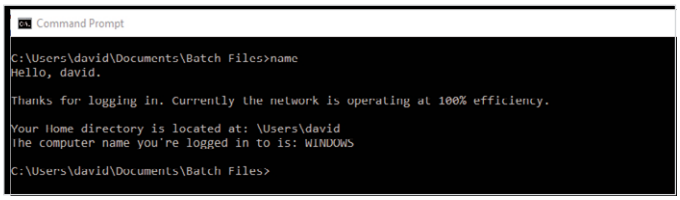


STEP 4 This is extremely useful if you want to create a unique, personal batch file that automatically runs when a user logs into Windows. Using the default systems variables that Windows itself creates, you can make a batch file that greets each user:

```
@echo off

echo Hello, %USERNAME%.
echo.
echo Thanks for logging in. Currently the network is operating at 100%% efficiency.
echo.
echo Your Home directory is located at: %HOMEPATH%
echo The computer name you're logged in to is: %COMPUTERNAME%
echo.
```



STEP 5 Save and execute the batch file changes; you can overwrite and still use name.bat if you want. The batch file takes the current system variables and reports them accordingly, depending on the user's login name and the name of the computer. Note: the double percent symbol means the percent sign will be displayed, and is not a variable.



STEP 6 Alternatively, you can run the batch file and display it on the user's desktop as a text file:

```
@echo off

echo Hello, %USERNAME%. > c:%HOMEPATH%\user.txt
echo. >> c:%HOMEPATH%\user.txt
echo Thanks for logging in. Currently the network is operating at 100%% efficiency. >> c:%HOMEPATH%\user.txt
echo. >> c:%HOMEPATH%\user.txt
echo Your Home directory is located at: %HOMEPATH% >> c:%HOMEPATH%\user.txt
echo The computer name you're logged in to is: %COMPUTERNAME% >> c:%HOMEPATH%\user.txt
echo. >> c:%HOMEPATH%\user.txt

notepad c:%HOMEPATH%\user.txt
```

The > outputs to a new file called user.txt, while the >> adds the lines within the file.

Playing with Variables

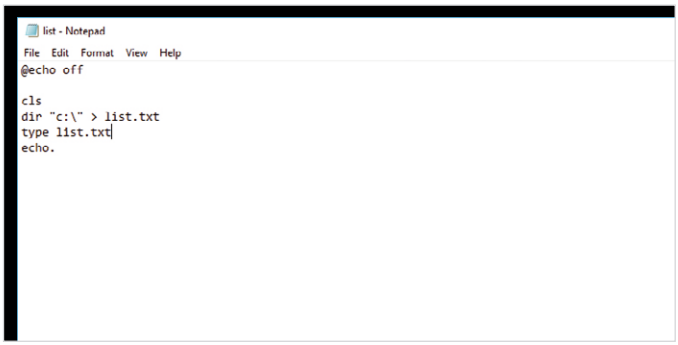
There's a lot you can accomplish with both the system and environmental variables, alongside your own. Mixing the two can make for a powerful and extremely useful batch file and when combined with other commands, the effect is really impressive.

USING MORE VARIABLES

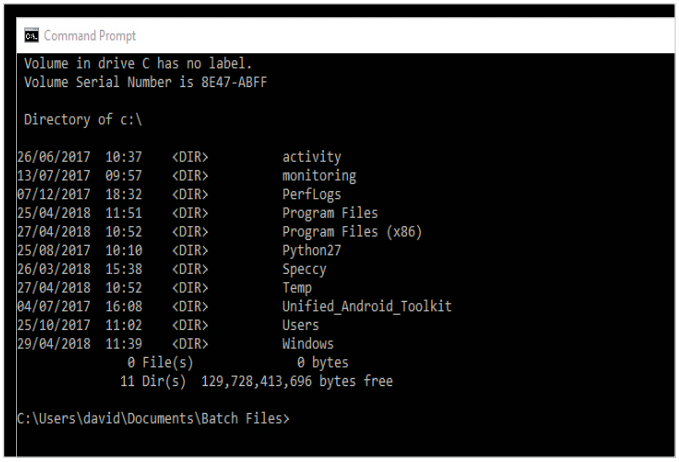
Here's a good example of mixing system and environmental variables with some of your own creation, along with a number of external Windows commands.

STEP 1 Create a new batch file called list.bat and start it off with the `@echo off` command. Begin by clearing the command prompt screen and displaying a list of the current directories on the computer:

```
cls
dir "c:\" > list.txt
type list.txt
echo.
```



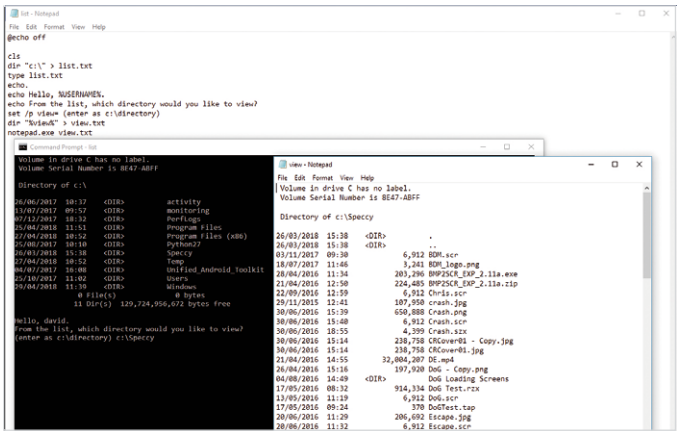
STEP 2 Save and execute the batch file. Within the command prompt you can see the contents of all the files and directories from the root of the C:\ drive; and as any user under Windows has permission to see this, there's no UAC elevated privileges required.



STEP 3 Now, create a batch file that displays the contents of any directory and post it as a text file to the user's screen. Add the following to the list.bat batch file:

```
echo Hello, %USERNAME%.
echo From the list, which folder would you like to view?
set /p view= (enter as c:\folder)
dir "%view%" > view.txt
notepad.exe view.txt
```

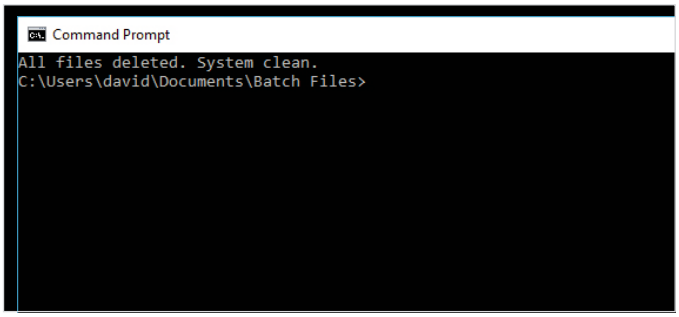
STEP 4 What's happening here is the batch file asks the user to enter any of the directories displayed in the list it generated, in the form of '**c:\directory**'. Providing the user enters a valid directory, its contents are displayed as a text file. We created the view variable here along with %HOMEPATH%, to store the input and the text file.



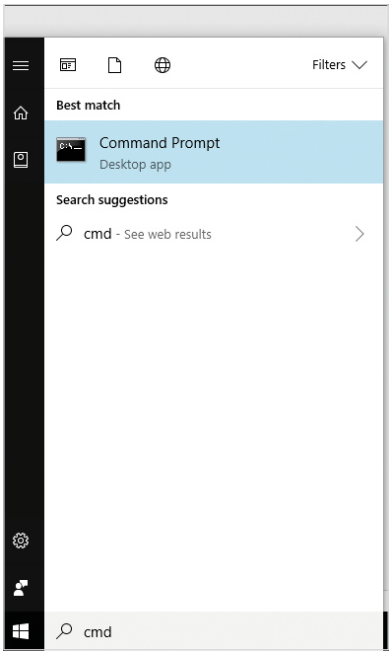
STEP 5 It's always a good idea, when creating text files for the user to temporarily view, to clean up after yourself. There's nothing worse than having countless, random text files cluttering up the file system. That being the case, let's clear up with:

```
cls
del /Q view.txt
del /Q list.txt
echo All files deleted. System clean.
```

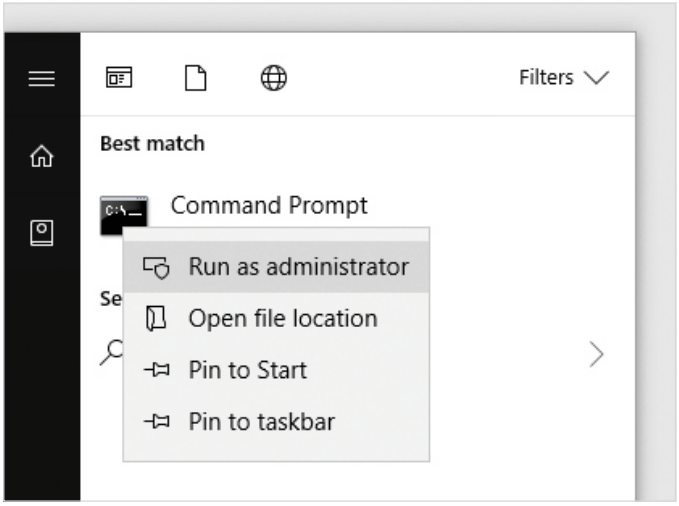
STEP 6 The additions to the batch file simply clear the command prompt window (using the cls command) and delete both the view.txt and list.txt files that were created by the batch file. The /Q flag in the del command means it deletes the files without any user input or notification. The final message informs the user that the files are removed.



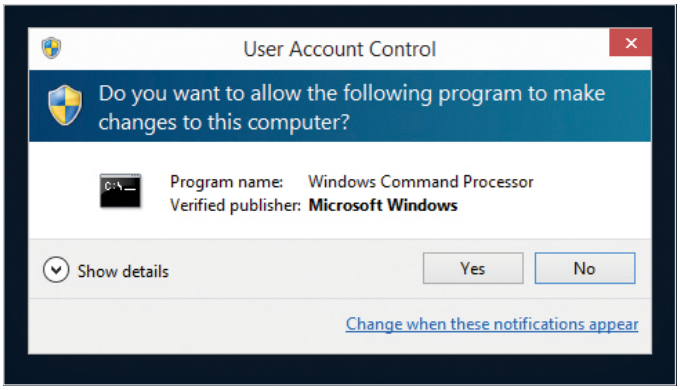
STEP 7 Depending on how your system is configured, you may not get any directory information at all or a message stating Access Denied. This is because the UAC is blocking access to protected areas of the system, like c:\Windows or C:\Program Files. Therefore, you need to run the batch file as an Administrator. Click the Windows Start button and enter **CMD** again.



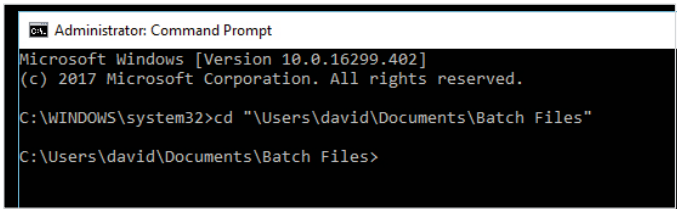
STEP 8 Instead of left clicking on the Command Prompt result, as you did the first time you opened it, right-click it and from the menu choose Run as Administrator. There is a risk that you could damage system files as the Administrator but as long as you're careful and don't do anything beyond viewing directories, you will be okay.



STEP 9 This action triggers the UAC warning message, asking you if you're sure you want to run the Windows command prompt with the elevated Administrator privileges. Most of the time we wouldn't recommend this course of action: the UAC is there to protect your system. In this case, however, click Yes.



STEP 10 With the UAC active, the command prompt looks a little different. For starters, it's now defaulting to the C:\WINDOWS\system32 folder and the top of the windows is labelled Administrator. To run the batch file, you need to navigate to the Batch Files directory with: **cd \Users\USERNAME\Documents\Batch Files**. To help, press the Tab key to auto-complete the directory names.



Batch File Programming

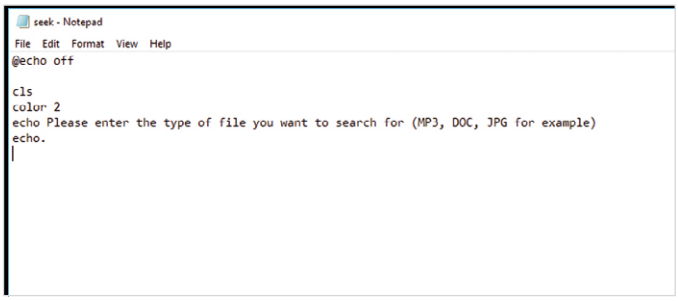
It's the little additions we can make to a batch file that help it stand out and ultimately become more useful. While the Windows graphical interface is still king, the command line can do just as much, and this is where batch files come into their own.

SEARCHING FOR FILES

Here's an interesting little batch file that you can easily extend for your own use. It asks the user for a file type to search for and displays the results.

STEP 1 We are introducing a couple of new commands into the mix here but we think they're really useful. Create a new batch file called seek.bat and in it put:

```
@echo off
cls
color 2
echo Please enter the type of file you want to
search for (MP3, DOC, JPG for example)
echo.
```

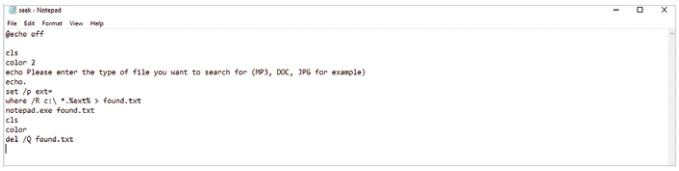


STEP 2 The new command in this instance is color (Americanised spelling). Color, as you already assume, changes the colour of the command prompt display. The color attributes are specified by two hex digits, the first corresponds to the background colour of the Command console and the second to the foreground, and can be any of the following values:

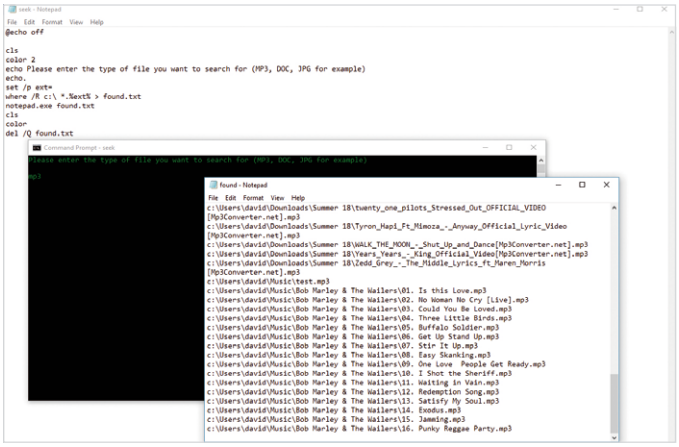
0 = Black	8 = Grey
1 = Blue	9 = Light Blue
2 = Green	A = Light Green
3 = Aqua	B = Light Aqua
4 = Red	C = Light Red
5 = Purple	D = Light Purple
6 = Yellow	E = Light Yellow
7 = White	F = Bright White

STEP 3 Now let's extend the seek.bat batch file:

```
@echo off
cls
color 2
echo Please enter the type of file you want to
search for (MP3, DOC, JPG for example)
echo.
set /p ext=
where /R c:\ *.%ext% > found.txt
notepad.exe found.txt
cls
color
del /Q found.txt
```



STEP 4 Another new command, Where, looks for a specific file or directory based on the user's requirements. In this case, we have created a blank variable called ext that the user can enter the file type in, which then searches using Where and dumps the results in a text file called found.txt. Save and run the batch file.

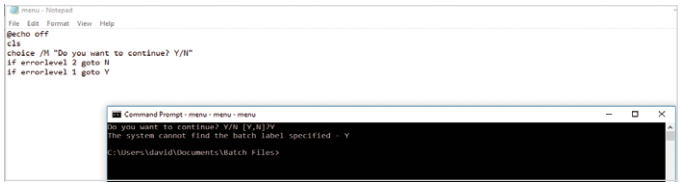


CHOICE MENUS

Creating a menu of choices is a classic batch file use and a good example to help expand your batch file programming skills. Here's some code to help you understand how it all works.

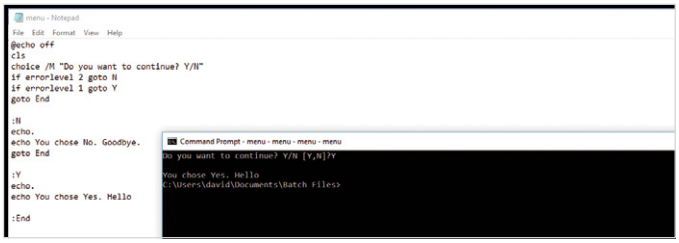
STEP 1 Rather than using a variable to process a user's response, batch files can instead use the Choice command in conjunction with an ErrorLevel parameter to make a menu. Create a new file called menu.bat and enter the following:

```
@echo off
cls
choice /M "Do you want to continue? Y/N"
if errorlevel 2 goto N
if errorlevel 1 goto Y
goto End:
```

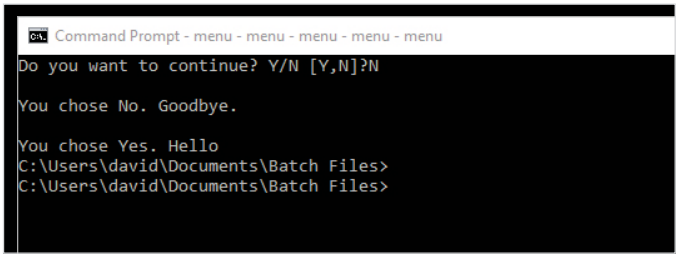


STEP 2 Running the code produces an error as we've called a Goto command without any reference to it in the file. Goto does exactly that, goes to a specific line in the batch file. Finish the file with the following and run it again:

```
:N
echo.
echo You chose No. Goodbye.
goto End

:Y
echo.
echo You chose Yes. Hello

:End
```



STEP 3 The output from your choice is different depending on whether you pick Y or N. The :End part simply signifies the end of the file (also known as EOF). Without it the batch file runs through each line and display the Y response even if you enter N; so it's important to remember to follow your Goto commands.



STEP 4 ErrorLevels are essentially variables and the /M switch of Choice allows a descriptive message string to be displayed. Extend this menu with something new:

```
@echo off
cls
echo.
echo ------------------------------------------
echo.
echo Please choose a directory.
echo.
echo Press 1 for c:\Music
echo.
echo Press 2 for c:\Documents
echo.
echo Press 3 for c:\Pictures
echo.
echo Press 4 for c:\Videos
echo.
echo ------------------------------------------
choice /C 1234
if errorlevel 4 goto Videos
if errorlevel 3 goto Pictures
if errorlevel 2 goto Documents
if errorlevel 1 goto Music
```

STEP 5 Now add the Goto sections:

```
:Videos
cls
CD %HOMEPATH%\Videos
echo You are now in the Videos directory.
goto End

:Pictures
cls
CD %HOMEPATH%\Pictures
echo You are now in the Pictures directory.
goto End

:Documents
cls
CD %HOMEPATH%\Documents
echo You are now in the Documents directory.
goto End

:Music
cls
CD %HOMEPATH%\Music
echo you are now in the Music directory.
goto End

:End
```

STEP 6 When executed, the batch file displays a menu and with each choice the code changes directory to the one the user entered. The %HOMEPATH% system variable will enter the currently logged in user's Music, Pictures and so directories, and not anyone else's.

Loops and Repetition

Looping and repeating commands are the staple diet of every programming language, including batch file programming. For example, you can create a simple countdown or even make numbered files or directories in the system.

COUNTERS

Creating code that counts in increasing or decreasing number sets is great for demonstrating loops. With that in mind, let's look at the If statement a little more, alongside more variables, and introduce the Else, Timeout and eof (End of File) commands.

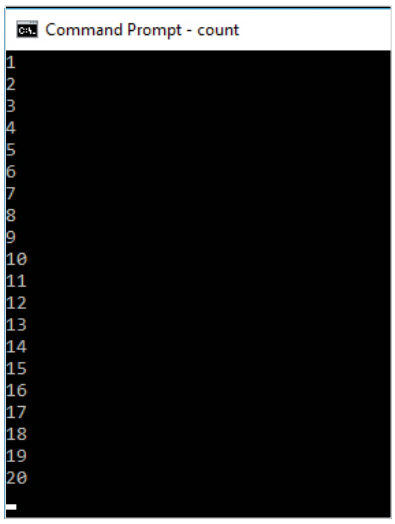
STEP 1

Start by creating a new batch file called count.bat. Enter the following, save it and run:

```
@echo off
cls
set /a counter=0

:numbers
set /a counter=%counter%+1
if %counter% ==100 (goto :eof) else (echo %counter%)
timeout /T 1 /nobreak > nul
goto :numbers
```

STEP 2

The count.bat code starts at number one and counts, scrolling down the screen, until it reaches 100. The Timeout command leaves a one second gap between numbers and the Else statement continues until the counter variable equals 100 before going to the eof (End Of File), thus closing the loop.



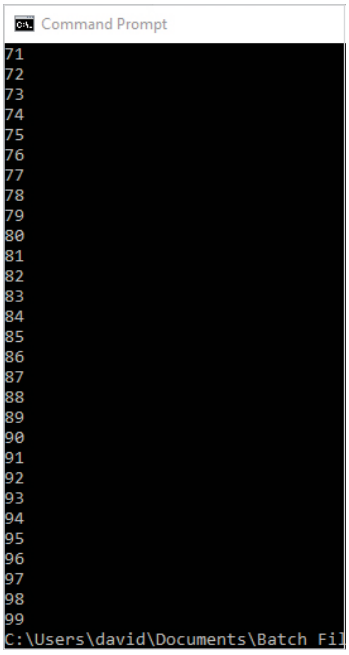
STEP 3

The count.bat is a rough way of demonstrating a loop; a better approach would be to use a for loop. Try this example instead:

```
@echo off
for /L %%n in (1,1,99) do echo %%n
```

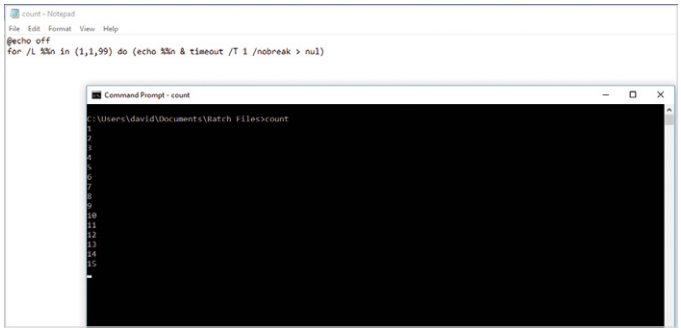
STEP 4

Breaking it down, there's For, then the /L switch, which handles a range of numbers. Then the parameter labelled as %%n to denote a number. Then the in (1,1,99) part, which tells the statement how to count, as in 1 (start number), 1 (steps to take), 99 (the end number). The next part is do, meaning DO whatever command is after.



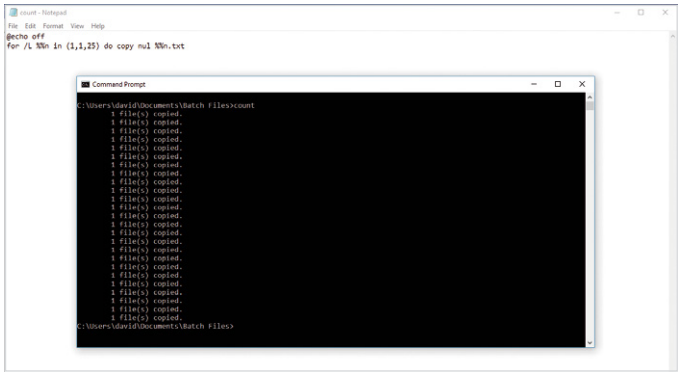
STEP 5 You can include the pause between the numbers easily enough within the far simpler For loop by adding multiple commands after the Do For loop. The brackets and ampersand (&) separate the different commands. Try this:

```
@echo off
for /L %%n in (1,1,99) do (echo %%n & timeout /T 1 /nobreak > nul)
```



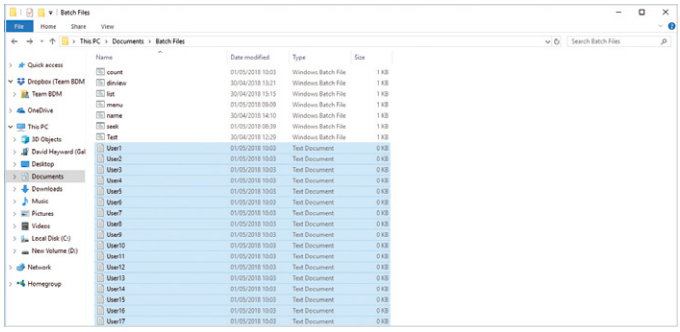
STEP 6 One of the great time saving uses of batch files is to create multiple, numbered files. Assume that you want twenty five text files within a directory, all numbered from 1 to 25. A For loop much like the previous example does the trick:

```
@echo off
for /L %%n in (1,1,25) do copy nul %%n.txt
```



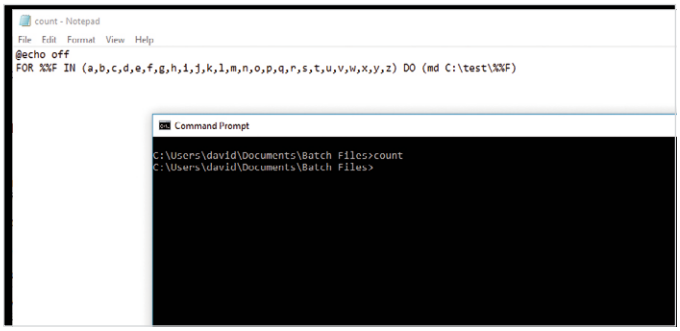
STEP 7 If you open Windows Explorer, and navigate to the Batch Files directory where you're working from, you can now see 25 text files all neatly numbered. Of course, you can append the file name with something like user1.txt and so on by altering the code to read:

```
@echo off
for /L %%n in (1,1,25) do copy nul User%%n.txt
```

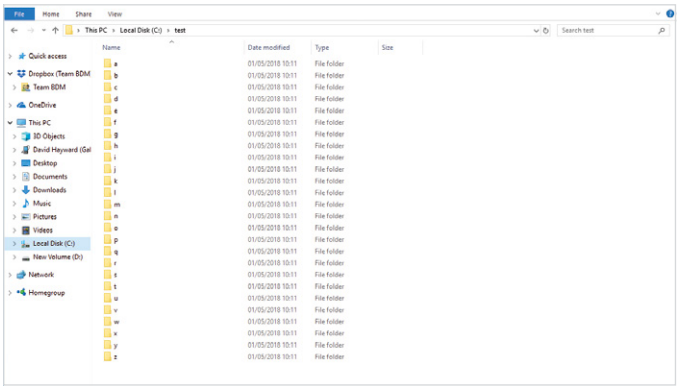


STEP 8 There are different ways of using the For loop. In this example, the code creates 26 directories, one for each letter of the alphabet, within the directory c:\test which the batch file makes using the MD command:

```
@echo off
FOR %%F IN (a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,
s,t,u,v,w,x,y,z) DO (md C:\test\%%F)
```

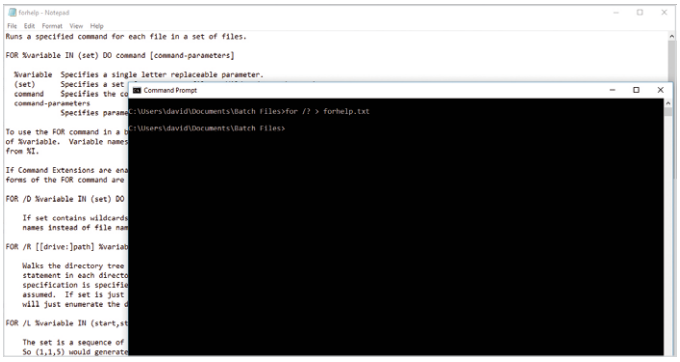


STEP 9 Loops can be powerful and extremely useful elements in a batch file. While creating 26 directories may not sound too helpful, imagine having to create 1,000 users on a network and assign each one their own set of unique directories. This is where a batch file saves an immense amount of time.



STEP 10 Should you ever get stuck when using the various commands within a batch file, drop into the command prompt and enter the command followed by a question switch. For example, for /? or if /?. You get an on-screen help file detailing the commands' use. For easier reading, pipe it to a text file:

```
For /? > forhelp.txt
```



Creating a Batch File Game

Based on what we've looked at so far with batch files, you can probably come up with your own simple text adventure or multiple-choice game. Here's one that we created, that you're free to fiddle with and make your own.

Make up your own questions but how about also including an introductory or loading screen? Make your loading screen in a separate batch file and save it as screens.bat (for example). Then, from the main game batch file, you can load it at the beginning of the file with the call command followed by color to reset the game's colours:

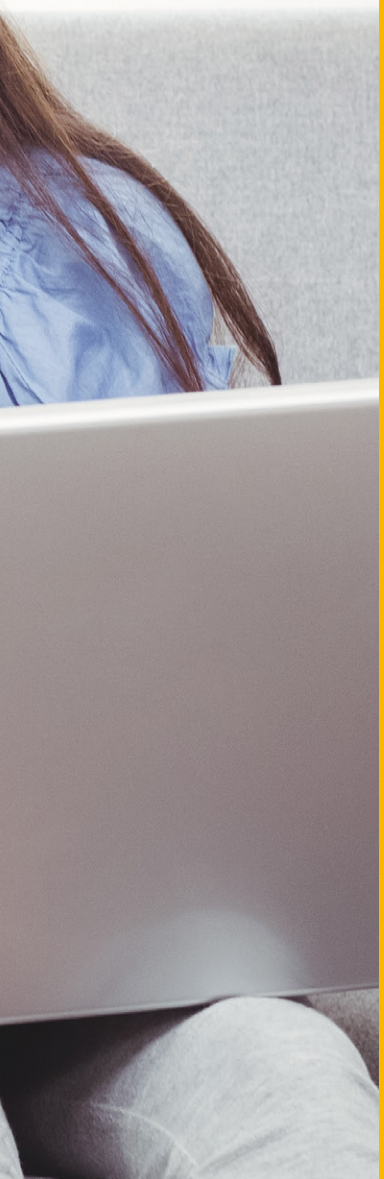
```
@echo off
Cls
Call screens.bat
color
:start
set /a score=0
set /a question=0
cls
set /p name= What is your name?
```

DIGCLOCK.PY

This is a surprisingly handy little script and one that we've used in the past instead of relying on a watch or even the clock in the system tray of the operating system.

```
@echo off
Cls
:start
set /a score=0
set /a question=0
cls
set /p name= What is your name?

:menu
cls
echo.
echo ******************************************************
echo.
echo Welcome %name% to the super-cool trivia game.
echo.
echo Press 1 to get started
echo.
echo Press 2 for instructions
echo.
echo Press Q to quit
echo.
echo ******************************************************
choice /C 12Q
if errorlevel 3 goto :eof
if errorlevel 2 goto Instructions
if errorlevel 1 goto Game

:Instructions
cls
echo.
echo ************************************
echo.
echo The instructions are simple. Answer the
   questions correctly.
echo.
echo ************************************
pause
cls
goto menu

:Game
set /a question=%question%+1
cls
if %question% ==5 (goto end) else (echo you are on
   question %question%)
echo.
echo get ready for the question...
echo.
timeout /T 5 /nobreak > nul
if %question% ==5 (goto end) else (goto %question%)
```

```
:1
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following version of Windows is the best?
echo.
echo A. Windows 10
echo.
echo B. Windows ME
echo.
echo C. Windows Vista
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:2
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following version of Windows is the
   most stable?
echo.
echo A. Windows 10
echo.
echo B. Windows 95
echo.
echo C. Windows ME
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:3
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following Windows version is the latest?
echo.
echo A. Windows 10
echo.
echo B. Windows 98
echo.
echo C. Windows 7
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:4
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following Windows uses DirectX 12?
echo.
echo A. Windows 10
echo.
echo B. Windows 3.11
echo.
echo C. Windows XP
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:Wrong
cls
echo ******************
echo.
echo WRONG!!!!
echo.
echo ******************
set /a score=%score%-1
pause
goto :game

:correct
cls
echo ******************
echo.
echo CORRECT. YIPEE!!!
echo.
echo ******************
set /a score=%score%+1
pause
goto :game

:end
cls
echo ******************************
echo.
echo Well done, %name%, you have answered all the questions
echo.
echo And your final score is....
echo.
echo %score%
echo.
echo ******************************
choice /M "play again? Y/N"
if errorlevel 2 goto :eof
if errorlevel 1 goto start
```



Programming with Scratch and Python

Scratch is a free programming language and online community that's targeted primarily at young people but also useful for older users too. It's a visual language created by MIT (Massachusetts Institute of Technology) and designed to help teach the building blocks of programming.

It's extremely versatile and as such can be used in conjunction with Python code to create interesting and useful programs. With the pairing of Scratch and Python you can make games, system utilities and even control external sensors, robots and motors.

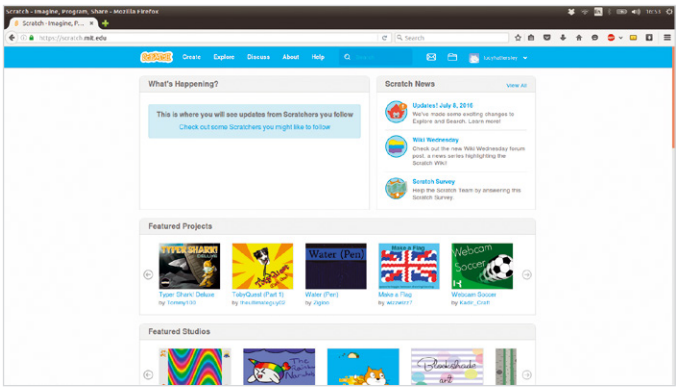
Getting Started with Scratch

If you are completely new to programming then Scratch is the perfect place to start. With Scratch you can learn the building blocks of programming and important programming concepts in a highly visual interface.

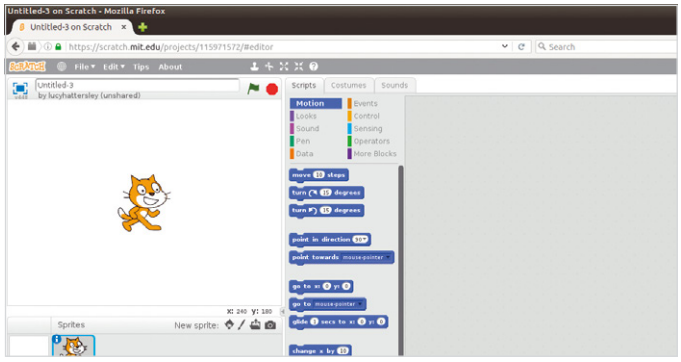
INSTALLING SCRATCH

Scratch can be run inside your web browser at scratch.mit.edu. You need to have Flash installed in your browser; if isn't already, it can be installed from get.adobe.com/flashplayer. Sign up for an account with Scratch so you can save your programs.

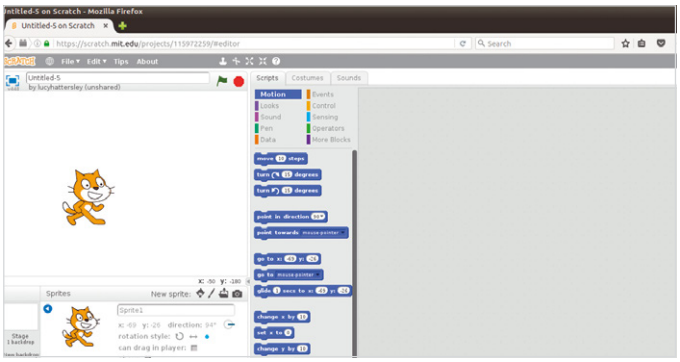
STEP 1 Scratch runs from inside the web browser. Click Create to open a new document. The Scratch interface opens in the web browser, click the maximise button on your browser so you have plenty of space to view the window and all its contents.



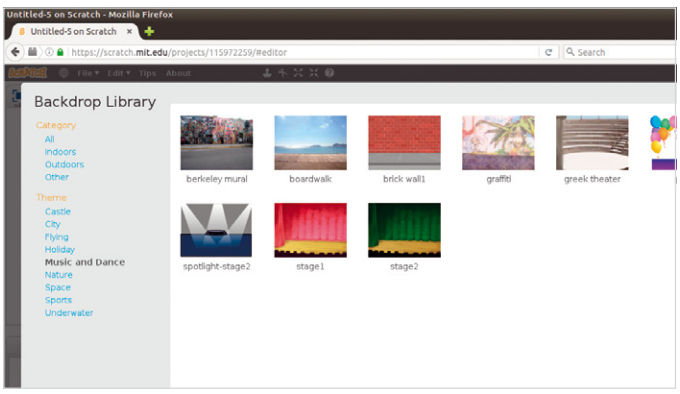
STEP 2 You can see the Scratch interface with a list of blue items in the Block Palette, an empty Script Area and a Stage. On the Stage will be an orange cartoon cat, known as "Scratch Cat". This is the default sprite that comes with all new projects; you will also see smaller versions of the sprite above the Script Area and in the Sprites Panel.



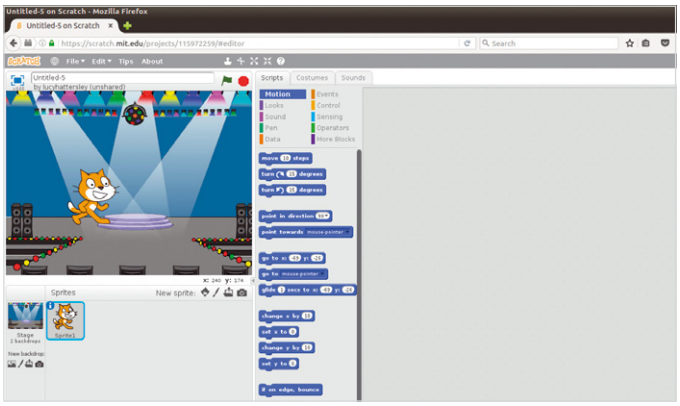
STEP 3 Let's take a look at Scratch Cat. Use click and drag with the mouse to position him on the Stage. At the top, just above Scripts, you'll see two icons for Grow and Shrink. Click one and click the cat to resize him. Shift-click on Scratch Cat and choose Info to access rotation controls. Click the blue back button to get back to the Sprites pane.



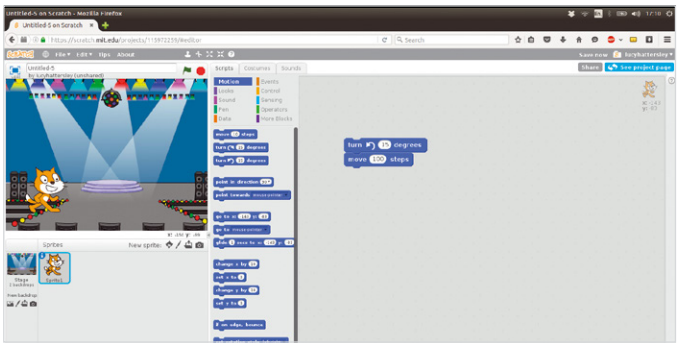
STEP 4 Scratch Cat looks a little lonely on his white space, so let's give him a background. Click the Stage icon to the left of the Sprites Pane. The Script Area switches to Backdrop Library displaying the available backgrounds. Click Music and Dance and choose spotlight-stage. Click OK.



STEP 5 The background appears on the Stage and Scratch Cat looks a lot happier. Let's create a script that moves him to the stage. Click Sprite1 in the Sprites Pane to select the cat and click the Scripts tab to return to the Script Area. Now click the blue Motion tab at the top of the Block Palette.



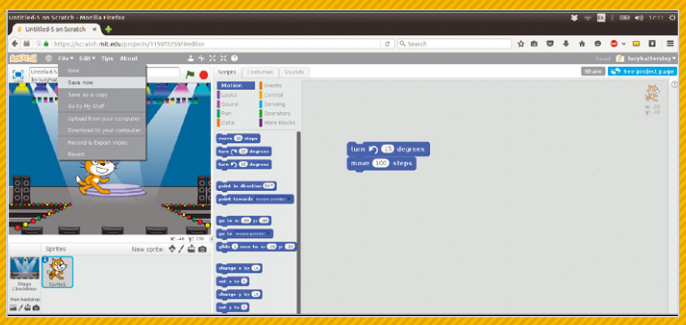
STEP 6 Drag the `turn [15] degrees` block (with an anti-clockwise symbol) from the Block Palette to the Script Area. Now drag the `move [10] steps` block and connect it to the bottom of the Turn 15 Degrees block. They will snap together. Change the `10` in `move [10] steps` to 100. Our program is now ready. Click the script (the two blocks) to see what it does. Scratch Cat will rotate and move towards the stage.



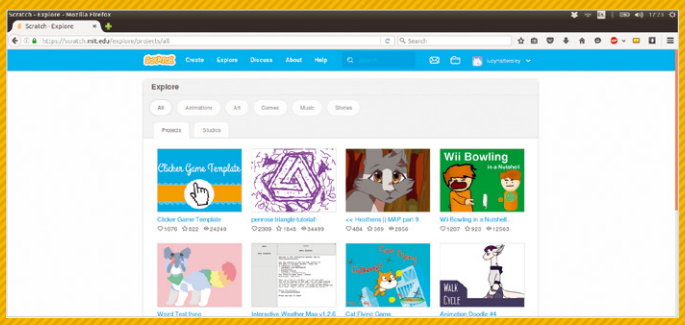
SAVING SCRATCH FILES

Take the time to learn how to open and save your files, and open other test programs, before you get stuck into programming with Scratch. There are lots of Scratch programs available so it's easy to learn alongside other users.

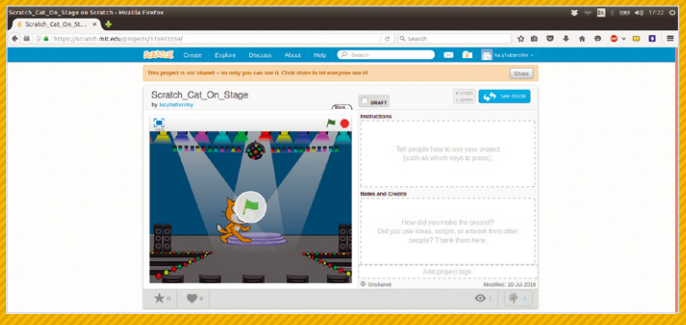
STEP 1 Click File > Save Now to save your project. Enter a name in the New Filename box; we called ours "Scratch_Cat_On_Stage". As we mentioned in both Python and Unix tutorials, it's important to avoid any special characters and spaces in your filenames. Use underscores "_" instead.



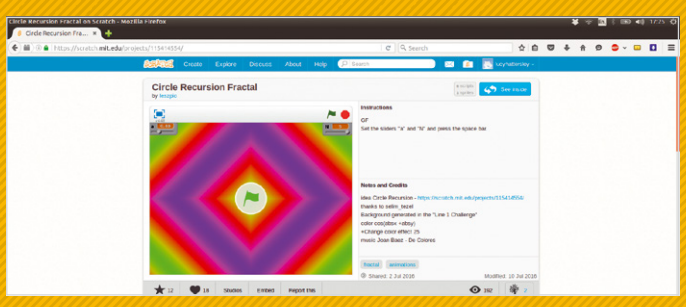
STEP 3 Lots of example Scratch files can be found on the MIT website by clicking Explore. Here you can see a huge range of projects built by other users and you'll also be able to share your own projects. Choose a project from Explore to open it.



STEP 2 Choose File > Go to My Stuff to exit the stage and view the saved file. Click the Scratch_Cat_On_Stage link to view your file. You can add Instructions, Notes and Credits here, and Tags. Click See Inside to head back to viewing your code again.



STEP 4 You can run the project directly inside the main window by clicking the Green Flag icon. Click the Star icon to Favourite or bookmark the project and the Heart icon to like it. More importantly, click the See Inside to see the code used to create the project. This is a good way to learn how Scratch code is being used.



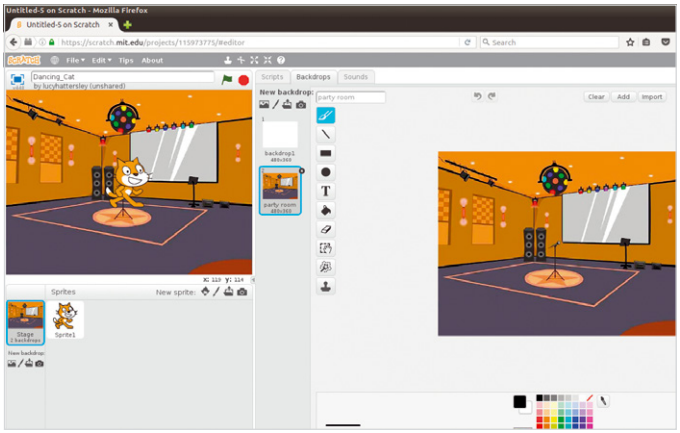
Creating Scripts in Scratch

The program you make in Scratch is a script, or a bunch of scripts. In this tutorial we'll take a look at how you construct your scripts to build up a great program. This is great starter practice for creating objects in Python.

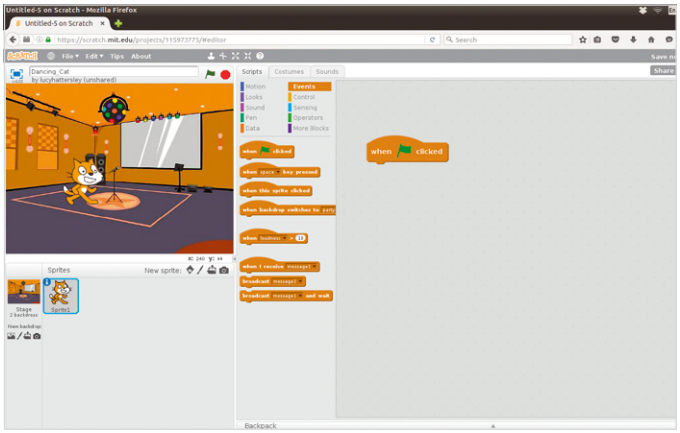
VISUAL CODING

The scripts in Scratch are created by snapping together blocks. These blocks are similar to the code you find in more complex programming languages, such as Python, but much easier to understand.

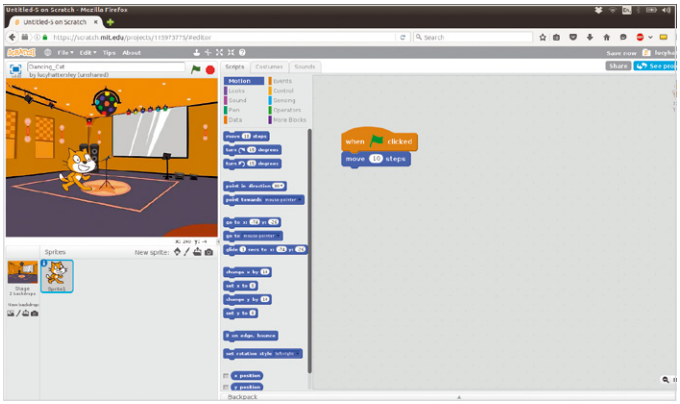
STEP 1 Click Create to start a new Scratch project and name it Dancing Cat. You're going to put your cat and some other characters on a dance floor and get them to bust some moves. Click Stage, then Music and Dance and choose party-room. Drag the Scratch Cat graphic around the Stage to find a good starting position.



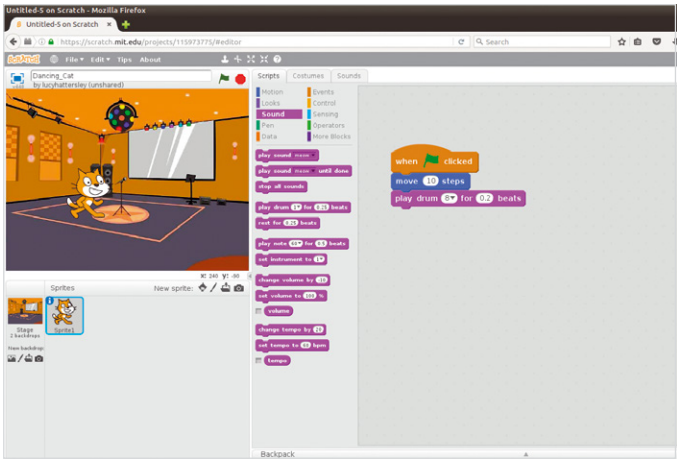
STEP 2 Click on Sprite1 in the Sprites Panes and click the Scripts tab above the Scripts area. Now click Events in the Blocks Pane and drag the `when flag clicked` block into the Scripts area. This block represents the start of your program. It tells Scratch to run through the blocks below it when we click the Green Flag icon above the Stage window.



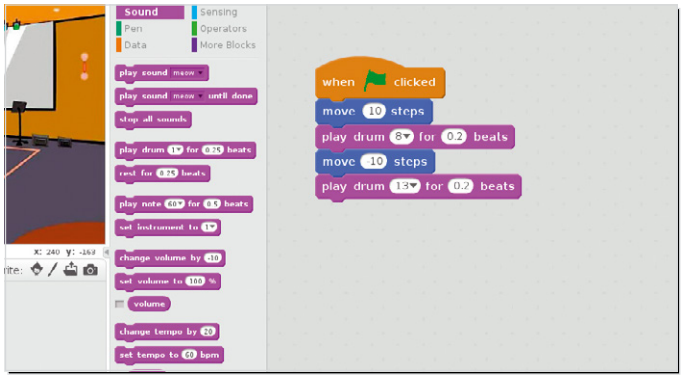
STEP 3 Click the Motion tab and drag the `move [10] steps` block and connect it beneath the `when flag clicked` block. A quick word about that [10]. When you write a number or word inside those square brackets, that's the way of saying you can choose a value. It's the equivalent of a variable", because it varies. We'll tell you which number or selection we're using but you can use any you want. Play around.



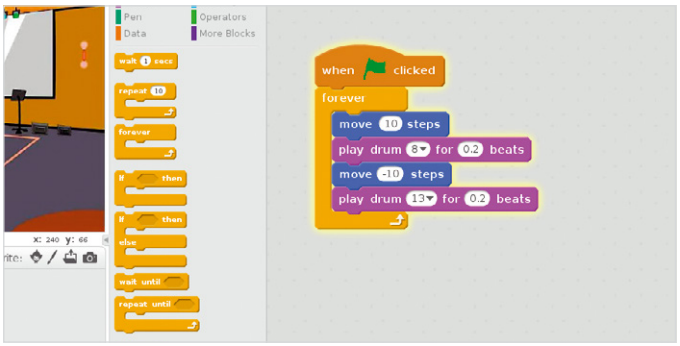
STEP 4 It's not much of a disco, so let's add some sound. Click the sound tab and drag `play drum [8] for [0.2] beats` and connect it to the bottom of the stack of blocks. Click on the blocks and Scratch Cat will move and a sound will come from your speaker.



STEP 5 Dancing is a back and forth affair, so let's get Scratch Cat moving back. Drag another `move [10] steps` block to the bottom of the stack. Now click the `10` and change it to `-10` (minus 10). Entering minus figures moves the cat backwards. Drag another `play drum` block to the bottom of the script. Pick a different drum sound. We chose `13`.



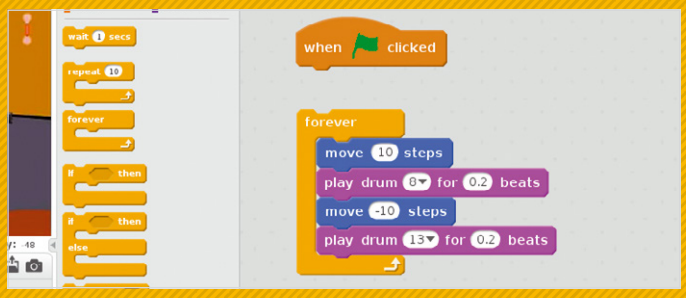
STEP 6 Scratch Cat only moves back and forth once, which isn't much of a party. Click Control and drag the `forever` block to the Script Area. Carefully position it beneath the `when [flag] clicked` block but above the `move [10] steps` block. The script should nest within the two prongs of the `forever` block. Click the Green Flag icon to start the disco. Click the red Stop icon to end the program.



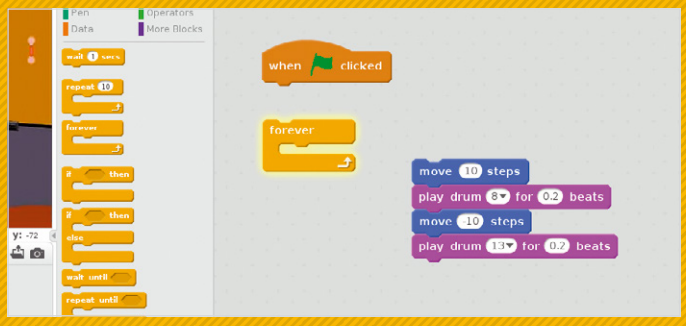
EDITING SCRIPTS

Nothing is set in stone, and you can move your blocks in and out of scripts and even have several scripts or parts of scripts in the Script Area. Scratch is far more forgiving than other programming languages for experimentation.

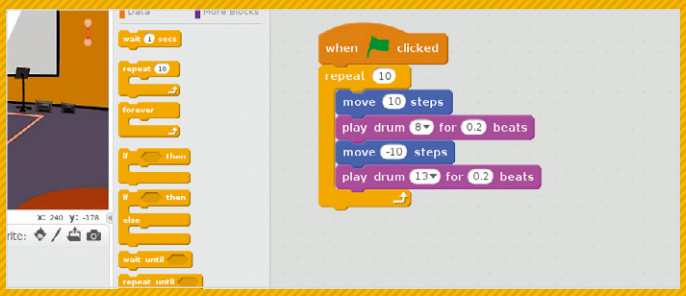
STEP 1 It's pretty bad form to use the `forever` block or a forever loop in programming. Programs are supposed to run from start to a finish. Even programs like Scratch have an end point when you quit the program. You want to replace the `forever` block with a `repeat` one. Click the `forever` block in your script and drag it down to separate it from the other blocks.



STEP 2 Your `move` and `play drum` blocks are still nested within the `forever` block though and you want to keep them. Click the topmost `move` block and drag it out of the `forever` block. It's now good to get rid of the `forever` block so drag it to the left and back to the Blocks List to get rid of it.



STEP 3 Now drag a `repeat [10]` block from the block list and connect it to the `when [flag] clicked` block in the Script Area. Now drag the top `play drum` block of the stack inside the `repeat [10]` block. If you drag the top block all the blocks underneath move with it and the whole lot will be nested inside the `repeat [10] stack`.



STEP 4 You can position the stack anywhere on the Script Area and even keep the unused blocks around, although we think it's good practice to keep only what you are using in the Script Area and remove any unused blocks. Click the Green Flag icon above the Stage Window to view Scratch Cat doing a short dance.

Interaction in Scratch

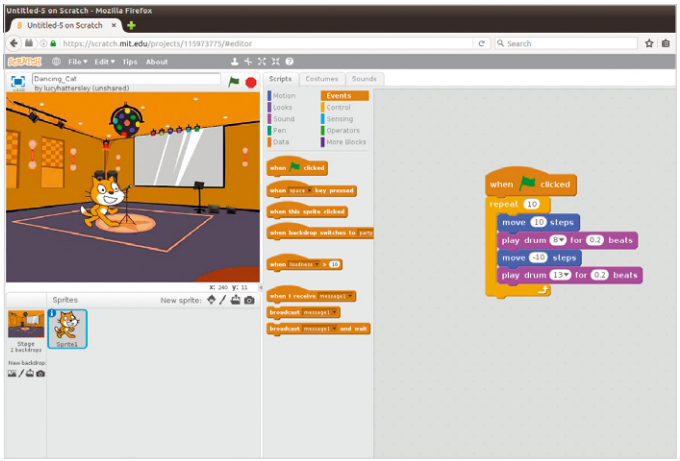
Your Scratch Cat is now dancing back and forth but wouldn't it be great if you could control him. In this tutorial you're going to look at creating keyboard interactions in Scratch. Let's get our disco cat really grooving!

INTERACTIVE CONTROL

The only Control option we've really looked at so far is the `when [flag] clicked` block, which starts the program. Once the program is running it does its thing, right up until it finishes. You're going to use the other Control blocks to do something more interesting.

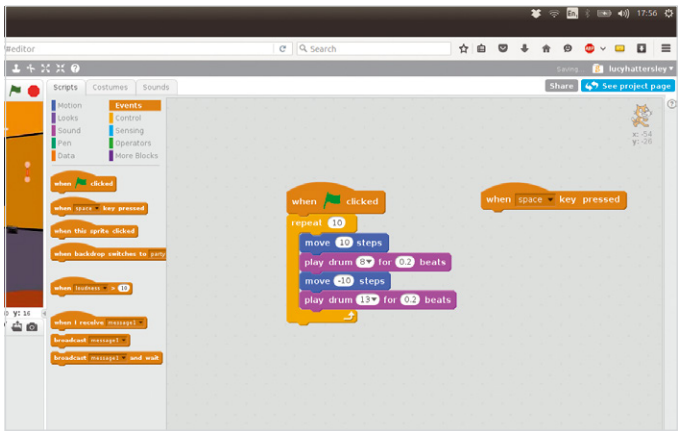
STEP 1

Open the Dancing Cat program from previous tutorials. Select Sprite1 and click on Events so you can see the `when [flag] clicked` script. Now click Control in the Block Palette and drag the `when [space] key pressed` block to an empty part of the Script Area.



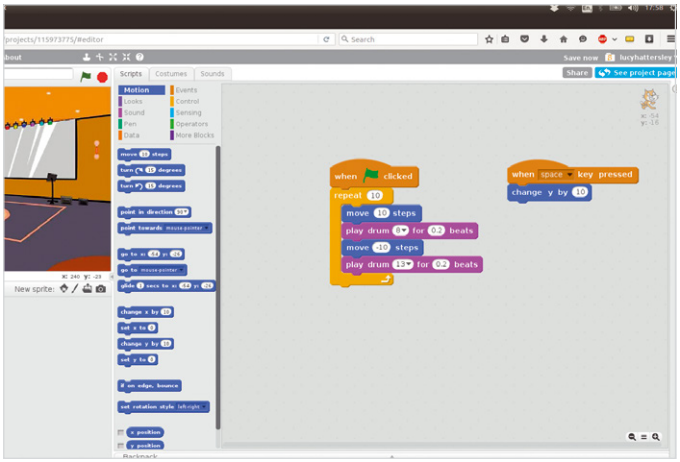
STEP 2

You can drag and rearrange the block scripts to any part of the Script Area. We like to have our `when [flag] clicked` scripts in the top left but it really doesn't matter where they are. It's also worth spotting that we now have more than one script for Sprite1; you can have multiple scripts for each sprite in your program.



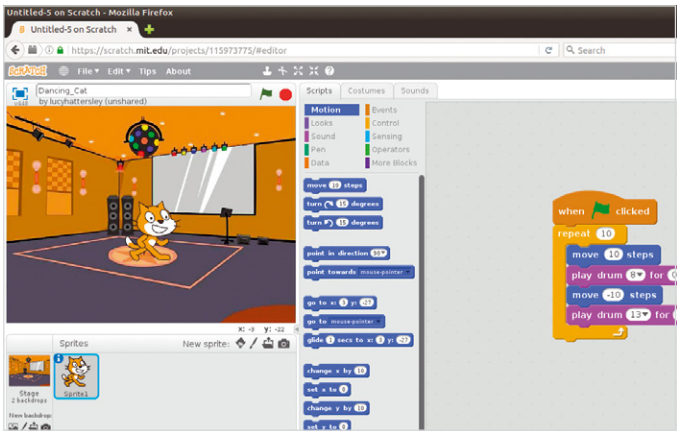
STEP 3

We're going to make Scratch Cat jump up and down when we press the space bar. Click Motion and drag `change y by [10]` and clip it to the `when [space] key pressed` block. What's with the "y"? This is what's known as a "coordinate".

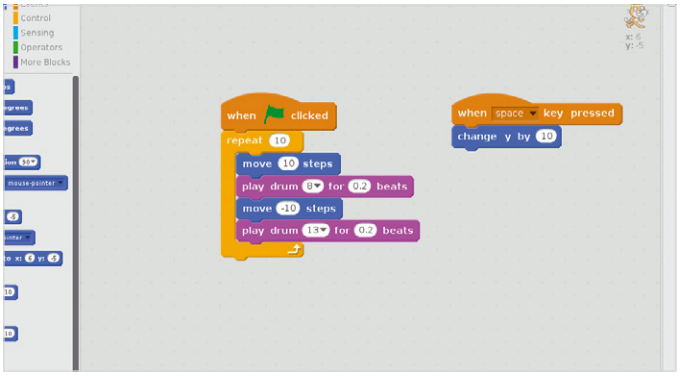


STEP 4

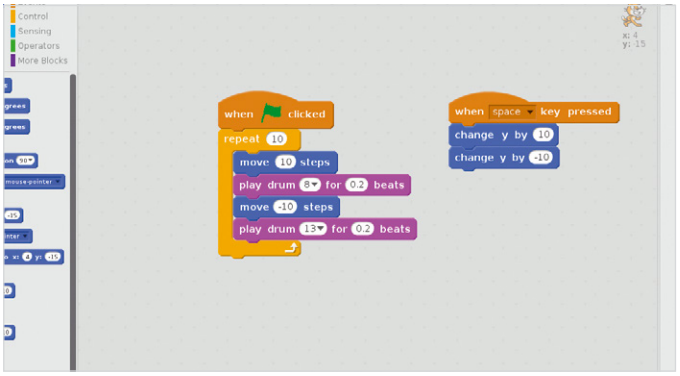
The position of each sprite on the stage is shown using two variables, x and y. These are referred to as the "coordinates". The x is the sprites horizontal position on the stage whilst the y coordinate is the vertical position. Click and drag the sprite around the Stage and you'll see the x and numbers change.



STEP 5 The centre of the Stage is x: 0 and y: 0. As you move the sprite up and to the right the numbers increase and as you move it left and down they decrease (going into negative numbers). So when we use the `change y by [10]` block it says, take the current value of y (the vertical position) and increase it by 10. That makes our cat jump up.



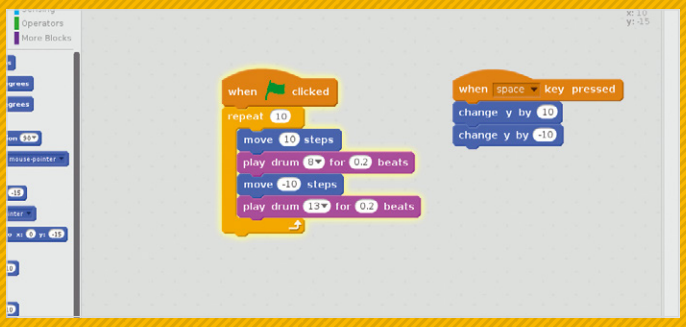
STEP 6 What goes up must come back down. So drag another `change y by [10]` block and attach it to the bottom of the `when [space] key pressed` script. Now change `[10]` to `[-10]`. Click the Green Flag and run the program. Now press the space bar and… oh no, nothing happens. We've just encountered our first "bug".



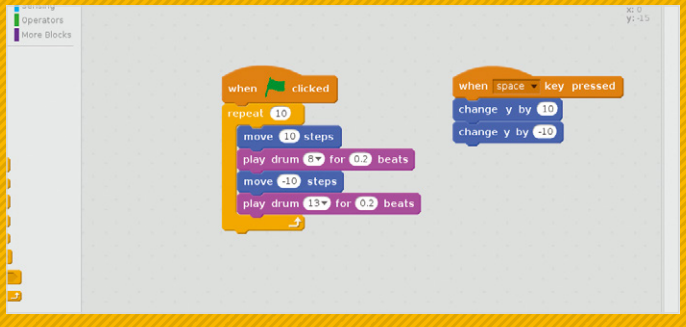
FIXING YOUR SCRIPT

We know that there's something wrong with our script and we want to see Scratch Cat jump when the space bar is pressed. So let's quickly squash this bug and see it working.

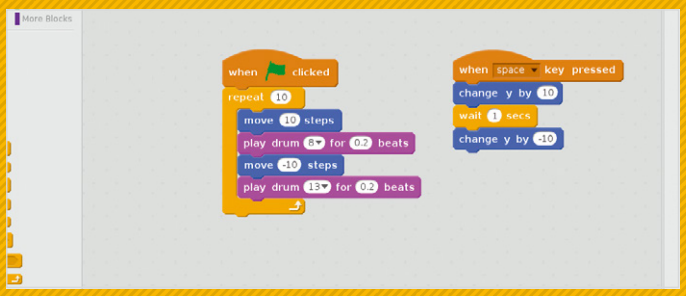
STEP 1 The problem is that programs are super-fast and highly visual programs like Scratch can move in the blink of an eye and that's what is happening here. If you tap the space bar repeatedly while the program is running you'll see Scratch Cat flickering as it jumps up and down.



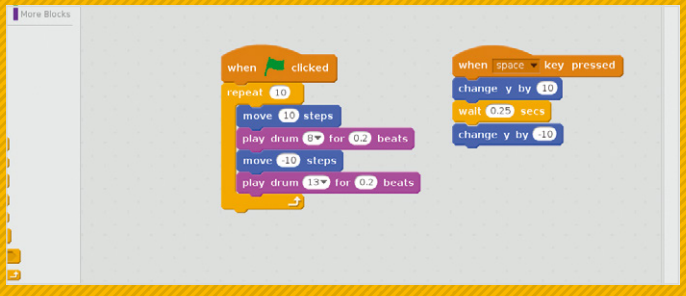
STEP 2 The challenge is that our motion controls move the cat instantly from one place to another, so fast that we can't see. Sometimes this is fine, like our back and forth dance, but obviously we need to slow down the jump. Help is at hand. Click the Motion tab to view the Motion blocks.



STEP 3 Drag a `wait [1] secs` block from the Blocks Palette and insert it underneath the `change [y] by 10` block. Now press the space bar on the keyboard to see Scratch cat jump up, and then back down. Notice that you don't need to press the Green Flag icon to run the program; the Green Flag starts our other script.



STEP 4 We think Scratch Cat stays in the air a bit too long. We want a jump, not a levitation effect. Change the `wait [1] secs` variable to `[0.25]`. This is a quarter of a second and will give us a more fun hop. Press the Green Flag to start the script and tap the space bar whenever you want Scratch Cat to jump.



Using Sprites in Scratch

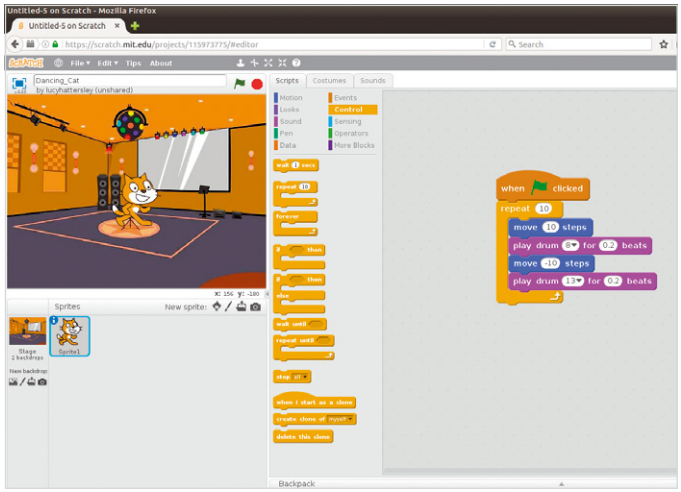
So far we've just got the one sprite in Scratch, the eponymous Scratch Cat. In this tutorial we're going to add a second sprite and see how to make the two sprites interact with each other.

LOOK SPRITE

Sprites are 2D (flat) graphics drawn on top of a background. They are commonly used to display information in games such as health bars, scores or lives. Older games are composed entirely of sprites, just like our Scratch project.

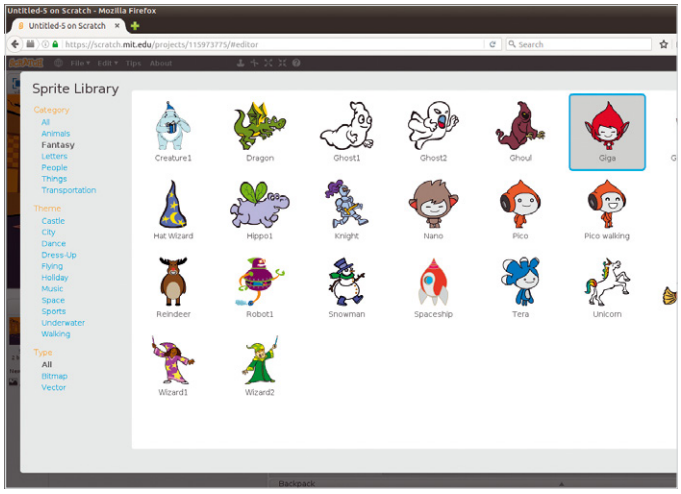
STEP 1

We're going to add another sprite to our project and a second character to the scene. Click the Choose Sprite From Library button, just above the Sprites pane. This opens the Sprite Library that displays all the different characters available.



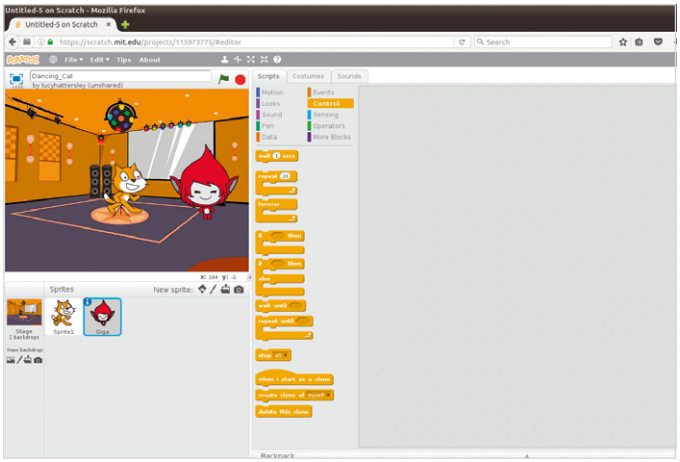
STEP 2

Click on the Fantasy link in the sidebar and choose Giga. Click OK to add the character to the stage. Click and drag the sprite to reposition Giga to the right of Scratch Cat. Notice that a Giga icon has joined Sprite1 in the Sprites pane.



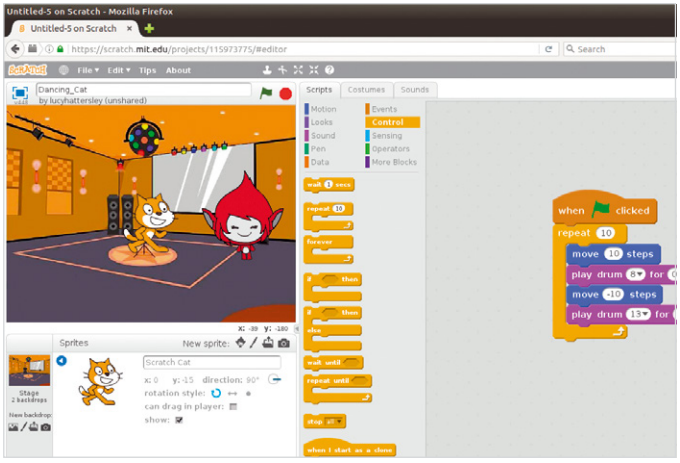
STEP 3

All the blocks on the Script Area have vanished. The scripts we built for Scratch Cat relate to that object, not to our new sprite. Click on Sprite1 in the Sprites pane to view the Scratch Cat scripts again. Then click Giga to return to your Giga character.

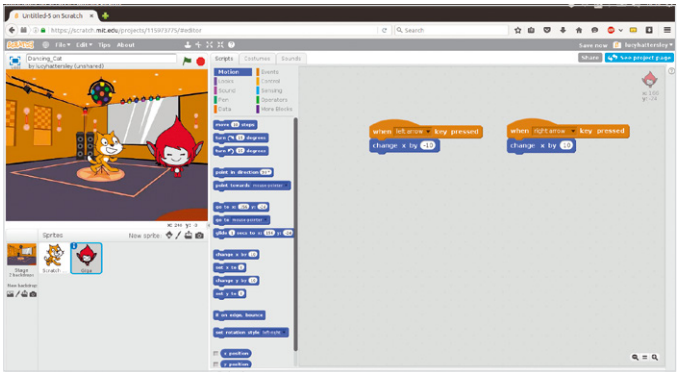


STEP 4

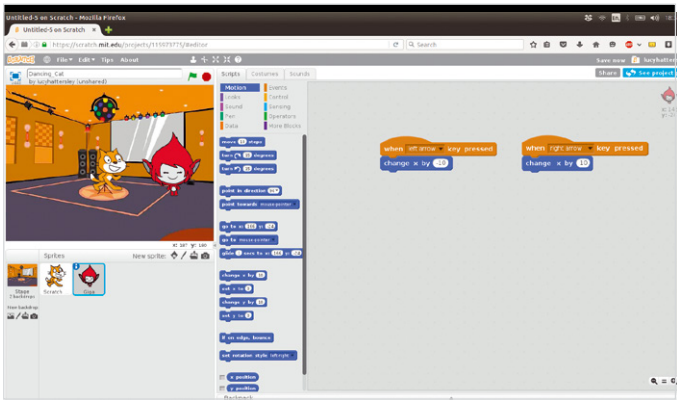
Now that we have more than one sprite, it's a good idea to name them. Click the "i" icon next to Sprite1. Change Sprite1 in the text box to Scratch Cat. Take a look at the other options here. You can remove a sprite from the stage by unticking the Show checkbox, without deleting it from the project.



STEP 5 Select Giga and choose Events. Drag a `when [space] key pressed` and change `[space]` to `[right arrow]`. Add a `change x by [10]` block beneath. Shift-click the script and choose duplicate to create another. For the second script change the when `[right arrow] key pressed` to `[left arrow]` and `change x by [10]` to `[-10]`.



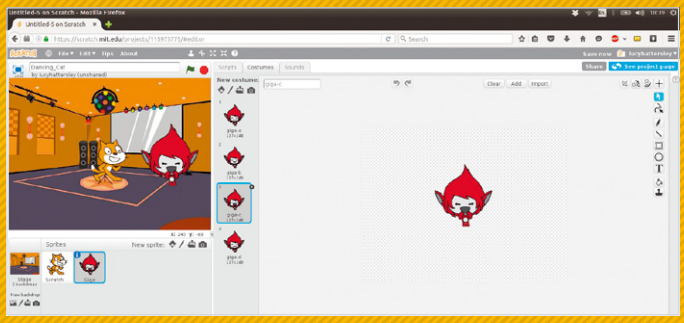
STEP 6 We're quite sure you can see where this is going. Press the Green Flag icon and our Scratch Cat object will start to dance and still jumps with space, whilst our other object, Giga, can be moved left and right using the arrow keys on our keyboard.



CHANGING COSTUMES

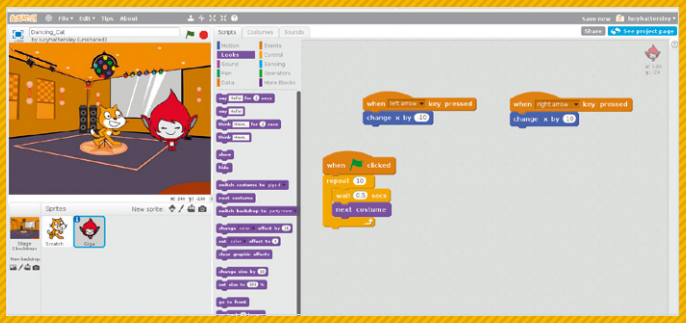
Our two objects, Giga and Scratch Cat don't have to look like the original characters. That's just the name we've given to each sprite. The visual appearance is a costume and our objects can change their costume and look completely different.

STEP 1 Select Giga in the Sprite Pane and click the Costumes tab. The Scripts Area now displays the costumes being used by Giga, including the current look. Choose giga-c in the list on the sidebar; this gives our sprite a different pose. Switch back to giga-a for now.

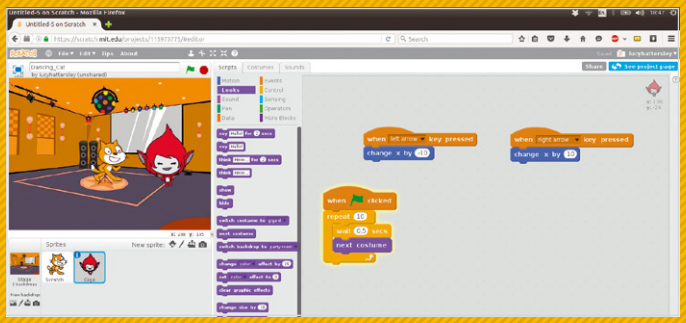


STEP 2 Let's use costume changes to animate Giga. Click Events and drag `when flag clicked` block to the Scripts Area. This block will activate at the same time as the Scratch Cat scripts when the Green Flag icon is clicked.

STEP 3 Attach a `repeat [10]` block and inside it place `wait [1] secs`. Change `[1]` to `[0.5]`. Click Looks and drag a `next costume` block into the `repeat` block. Now Giga will switch to the next costume every half a second, creating an animation effect.



STEP 4 Click the Green Flag icon to start the animation. Scratch Cat now moves back and forth, tapping out a beat, and Giga animates through four different poses. You can move Giga left and right using the arrow keys. It's starting to form into an interactive scene.



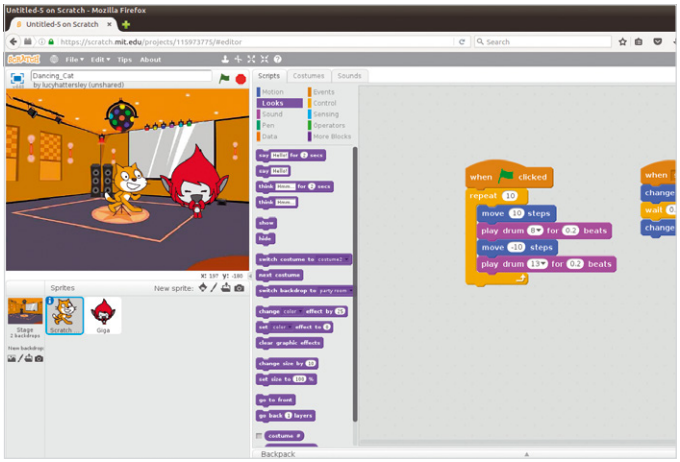
Sensing and Broadcast

Get your sprites and scripts to communicate with each other. The concept of Sensing and Broadcast is similar to the way objects communicate in Python. This tutorial will walk you through the process.

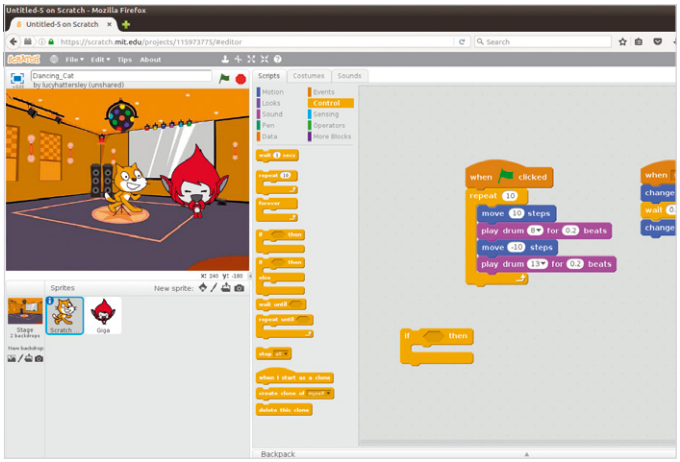
MORE INTERACTION

Earlier we looked at how you could interact with scripts, using the keyboard to move the sprites but scripts and sprites can interact with each other, sending messages and responding to events.

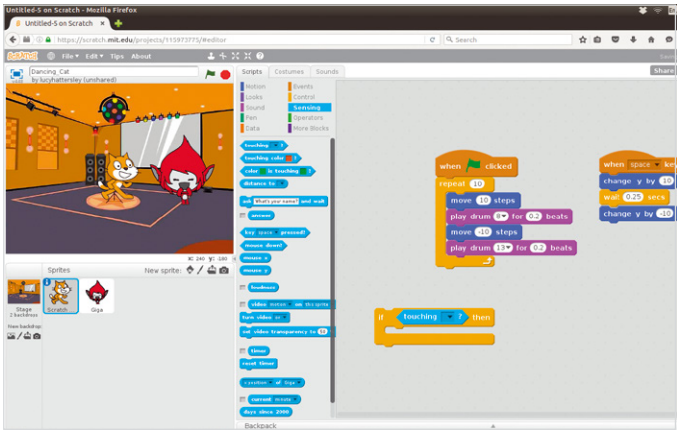
STEP 1 Scripts can broadcast messages to each other. These can be used to start other scripts or respond to events. You're going to get Scratch Cat to respond to the Giga and say "Watch Out!!!" if the two sprites touch. Select Scratch Cat in the Sprites Pane so you can view its Script Area.



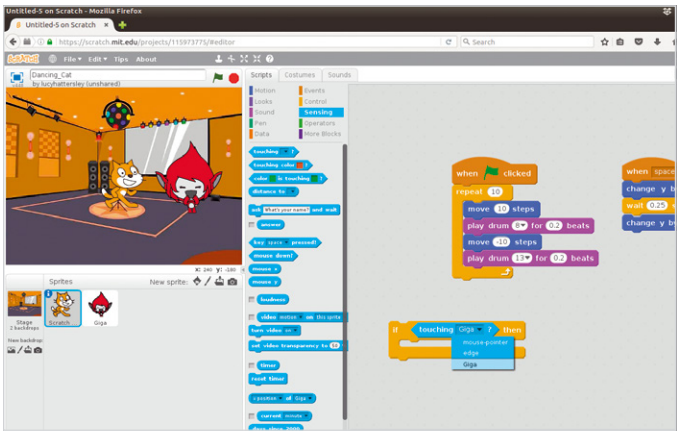
STEP 2 You 've looked at 'if' and 'else' in Python, so now you need to look at them in Scratch. Programs work around a few simple terms and 'if this then do that' is one of them. On our stage, you want to say, "if Giga touches Scratch then Scratch says 'Watch Out!!!'" Click Control and drag an `if` block to the Script Area.



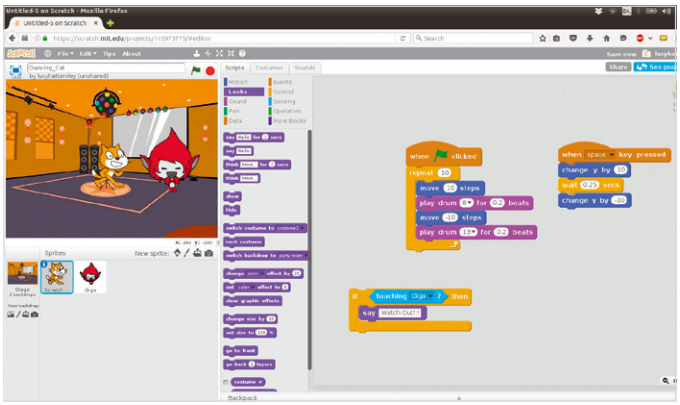
STEP 3 The options for interaction between the sprites are found in the Sensing part of the Block Palette. The one we are looking for is touching, at the top. Notice that this is a different shape to ones you are used to. It is designed to fit in the slot next to our `if` block. Drag the `touching` block into the spare slot in the `if` block.



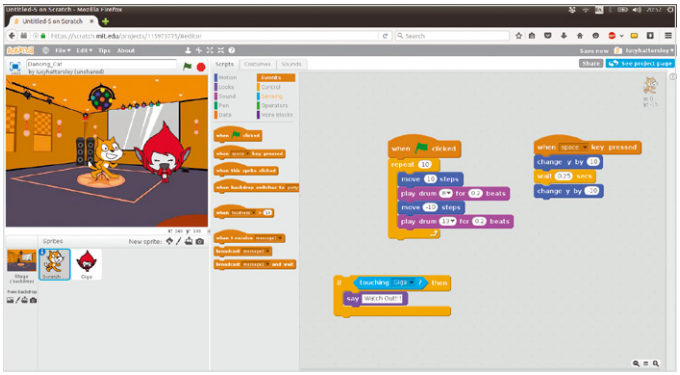
STEP 4 The `touching` block responds to the condition of our sprite; in this case if it's touching another sprite. However, we need to tell it which one. Click the arrow in the touching block and choose Giga from the list. There are a couple of default options, mouse pointer and edge. These are handy if you're making games.



STEP 5 We've got our `if`, what we now need is a response. Click Looks and drag a `say [Hello!] for 2 secs` block and insert it inside the `if` block. Change `[Hello!]` to `[Watch Out!!!]`. Try to run the program. Nothing happens! That's because our `if` script isn't part of the `when flag clicked` script.



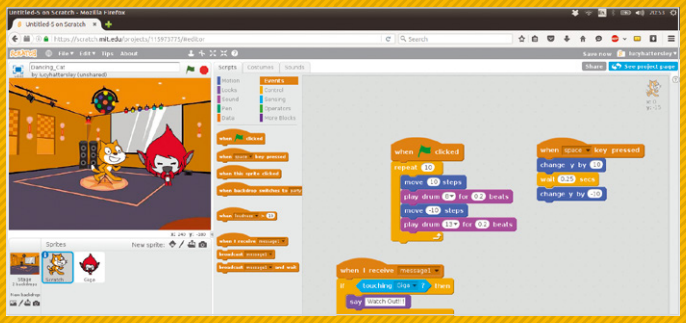
STEP 6 How do we get our 'if' script to run alongside the other scripts? We could put the `if` block inside `when flag clicked` but it would look big and ugly and do two different things. We could add a second when [flag] clicked block to the `if` block but that would only work if they were touching at the start. The answer is to use a `broadcast` block.



BROADCASTING

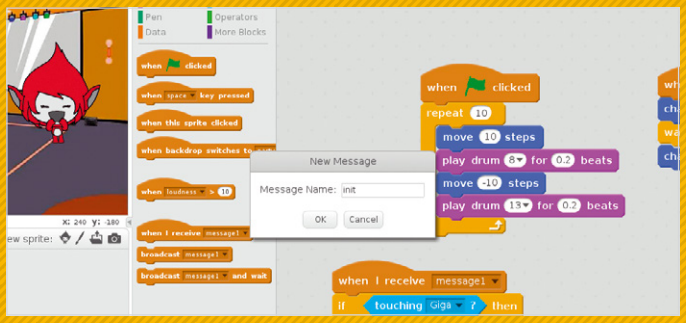
The broadcast block enables one script to interact with the others. It can be used to tell scripts to run and is ideal for bigger projects where each sprite does several things at once.

STEP 1 Click the Events tab in Scripts and look for the `when I receive` block. Drag this and connect it to the top of the `if` block. We need to set the receiver, i.e. the message it gets from the other script. Click the arrow next to receive to reveal the Message name window.

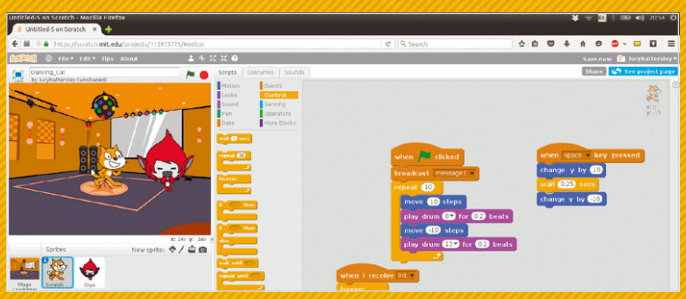


STEP 3 Now we need to broadcast that 'init' message from our when [flag] clicked script so it runs at the same time. Drag a `broadcast` block to the Script Area and insert between the `when flag clicked` block and the `repeat [10]` block. Now click the arrow in `broadcast` and choose 'init'.

STEP 2 You can call the message anything you want but it should start with a lowercase letter and in this instance you are going to use the word "init". This stands for "initialize" and getting the lingo right now will make your life much easier when you move to a more complex language. Click OK.



STEP 4 Our program is almost ready but our `if` script only works once, when the Green Flag icon is clicked. We're going to place it inside a Forever block. This is okay though because it's not our main program and we are also going to add a `stop all` block to the end of our `when flag clicked` block. Click the Green Flag to see your program run.



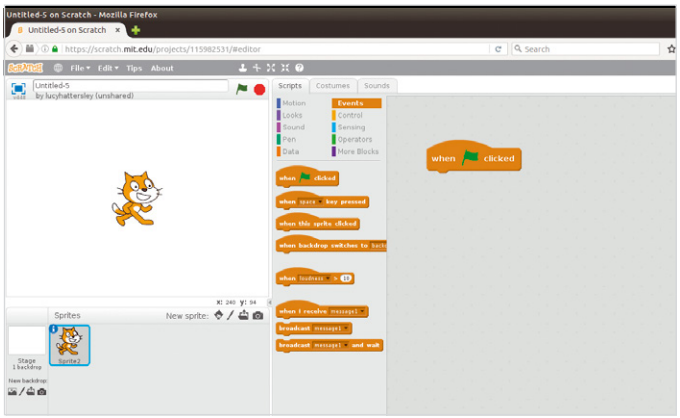
Objects and Local Variables

You've already met variables but Scratch makes them easy to understand. What isn't so obvious in Python is how variables are stored locally to objects. This is where Scratch helps. You're going to create a dice game to help understand this.

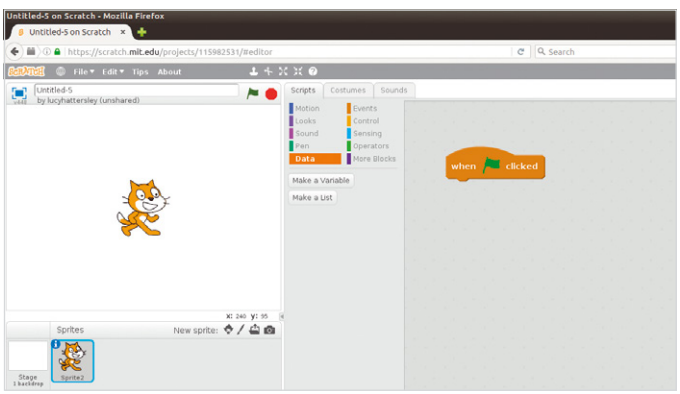
ARE YOU LOCAL OR GLOBAL?

Objects, like your Scratch Cat sprite, can have their own variables. This could be the player score or the amount of ammunition left. These are stored inside the object and are known as "local". Variables used by all objects are known as 'global'.

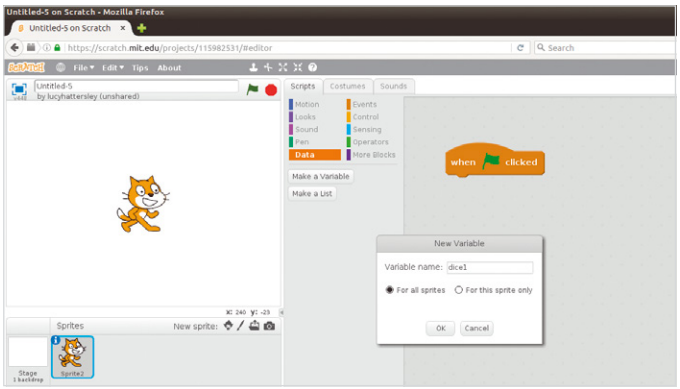
STEP 1 You're going to leave the disco behind. Choose File > New. You start with a blank stage containing a single Scratch Cat sprite. Click Control and drag a `when flag clicked` block to the Script Area.



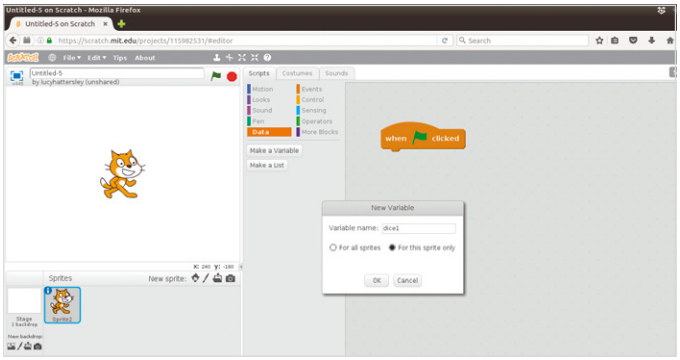
STEP 2 You 're going to create a simple game where Scratch Cat rolls two dice and wins if they're both the same number. Click Data. Unlike other sections there are no blocks here; instead we have to create the variables we need. We need two, one for each dice.



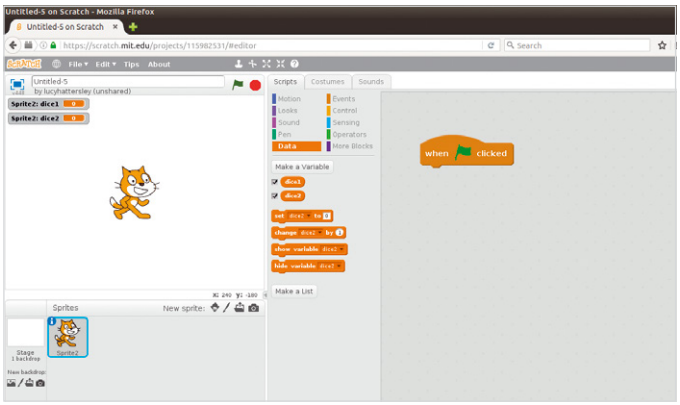
STEP 3 Click Make a variable and enter `dice1` into the ? window making sure that your variable starts with a lowercase letter. There are two options here: `For all sprites` and `For this sprite only`. `For all sprites` allows every sprite to use the dice; this is known as a global variable.



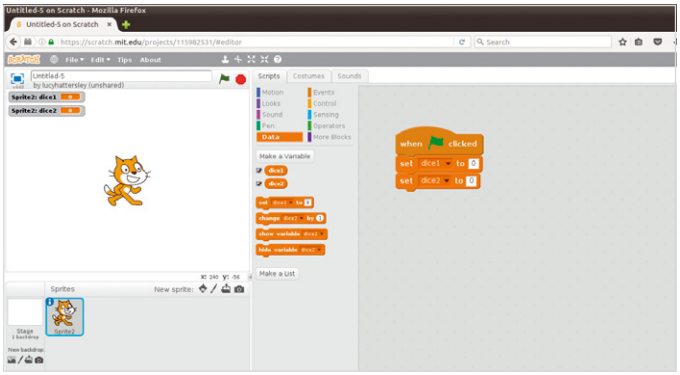
STEP 4 `For this sprite only` means that only this sprite can use the two dice variables. this is known as a local variable. This is useful if you want to create another character with their own set of dice and play against each other. We're doing that in the next tutorial, so choose For this sprite only and click OK.



STEP 5 A whole bunch of blocks appears in the Block Palette. We can now use our dice1 variable but we want two dice, so click `Make a variable` again and this time enter `dice2`. Remember to choose `For this sprite only` and click OK.



STEP 6 Both the dice1 and dice2 variables are currently empty. They could be anything we wanted, but we want them to be a random number between 1 and 6. Drag the `set [dice1] to [0]` block and click it underneath the `when flag clicked` block. Drag another `set [dice1] to [0]` block underneath and change the `[dice1]` setting to `[dice2]`.

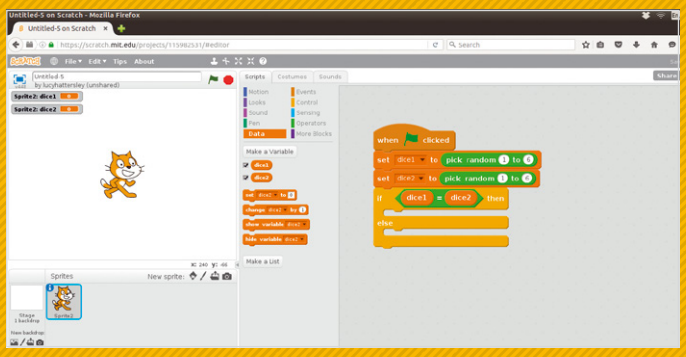


SMOOTH OPERATORS

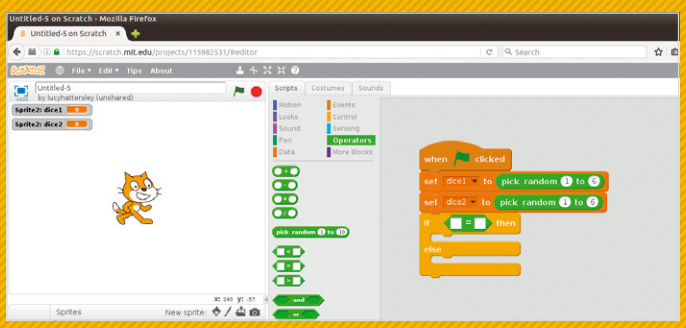
Operators are used to change the values of variables. Some of these will be familiar; you've used the addition operator '+' to add two numbers together. Programs can also check if numbers are equal, bigger or smaller than each other or even not equal.

STEP 1 Click the Operator tab and drag a pick `random 1 to 10` block and drop it into the `[0]` in `set dice1 to [0]`. Change the `[10]` to a `[6]` so it reads `set dice1 to pick random [1] to [6]`. When we click the Green Flag it will pick a number between 1 and 6 and store it in the dice1 variable. Add a `pick random [1] to [10]` block to dice 2 and also set it to `[6]`.

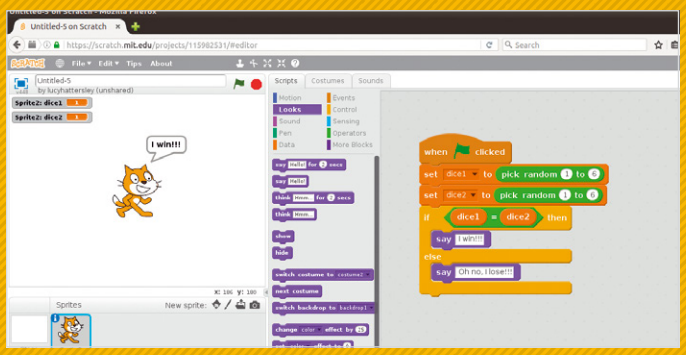
STEP 3 The `=` operator checks if two things are the same but we need to tell it to check our variables. Click Data and drag `dice1` to the space on the left of the `=` block. Drag `dice2` to the space on the right of the `=` block.



STEP 2 We need to check the two dice. Click the Control tab and drag the `if else` block to the script. This block is like the `if` block we used before but it says, "if this happens, do this; if not, do this instead." Now click operators and look for the `=` block. Drag it into the space next to `if`.



STEP 4 Finally click the Looks tab and drag `say [Hello!]` blocks into the `if` and `else` spaces. Change the say `[Hello!]` text in `if` to `[I win!!!]` and in `else` to `[Oh no. I lose!]`. Click the Green Flag to play the game.



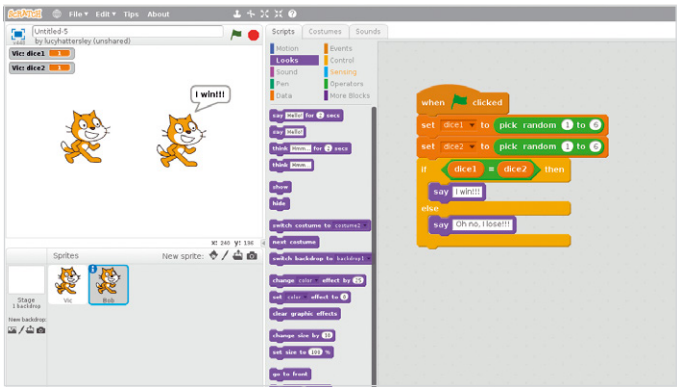
Global Variables and a Dice Game

We're going to create a game where two sprites play dice with each other and the sprite with the highest score wins. This lets us examine the idea of more than one object, each with its own set of local variables.

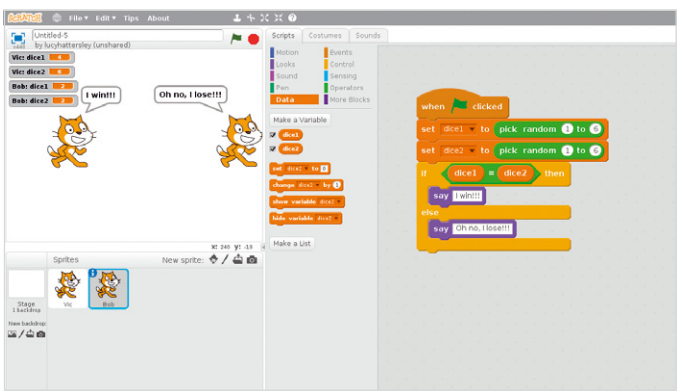
DOUBLE TROUBLE

We're going to start with the dice game from the previous tutorial, but create a second character. The fancy OOP word for this is "instantiation": creating an instance of something – in this case another instance of Scratch Cat.

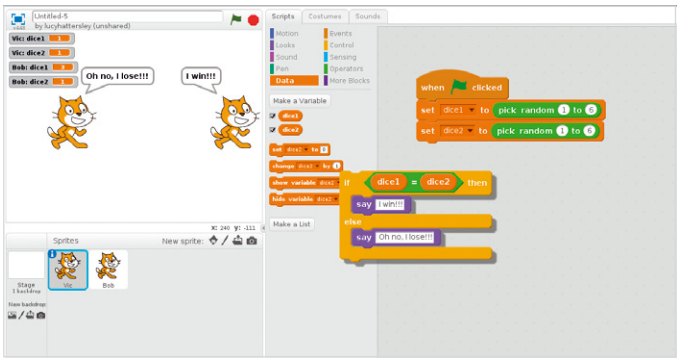
STEP 1 Let's add a second character to the Stage. Shift click on Scratch Cat on the stage and choose Duplicate. Click OK. Rearrange the two characters on the stage so they're stood side by side. Use the Sprite Info Window to give them names: "Vic" and "Bob".



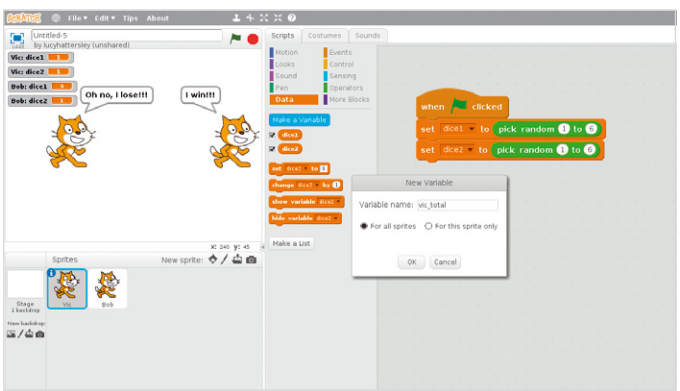
STEP 2 Select Bob and click Data in the Script Palette and you'll see dice1 and dice2. Bob received his own set of dice when we duplicated him but he doesn't share dice1 and dice2 with Vic. Place checks in the boxes next to dice1 and dice2 so you can see them on the Stage.



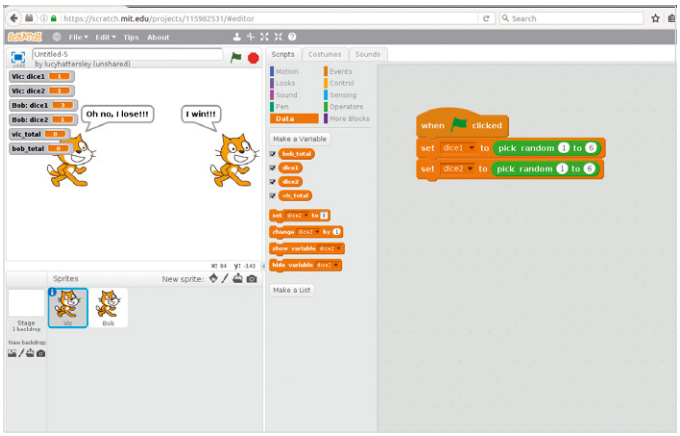
STEP 3 You can check this out by clicking the Green Flag icon. Vic and Bob will both roll dice1 and dice2 but each gets its own random result (shown in the Stage). We're going to change the game to one where the two dice are added together (highest score wins). Disconnect the If block from the script and drag it to the Block Palette.



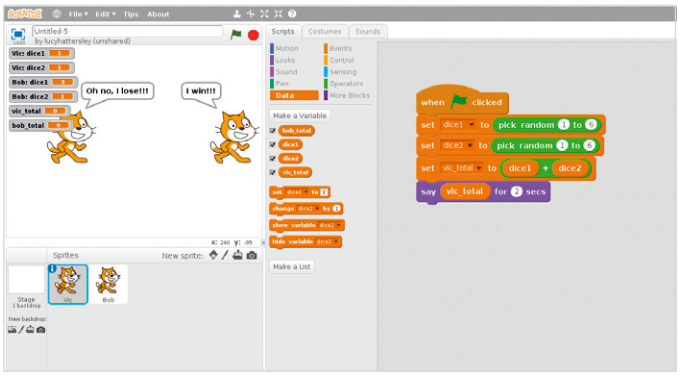
STEP 4 We need two new variables: one for Vic's total, and one for Bob's. Click `Make a variable` and enter `vic_total`. This time choose `For all sprites` then click OK. With Vic still selected, click `Make a variable` again and enter `bob_total`. Click OK.



STEP 5 Imagine both characters writing their totals on a blackboard that they share, but each has its own pair of dice. The totals are global and shared across both characters; the dice are local to each object.



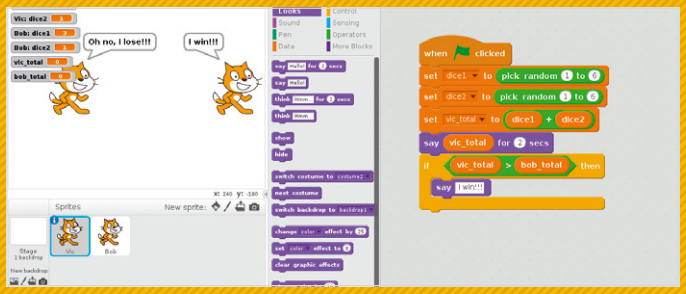
STEP 6 Drag `set [dice2] to [0]` from the block palette and connect it to the script. Change `dice2` to `vic_total`. Now click Operators and drag the `+` block to the `[0]`. Click Data and drag `dice1` to the left side of the `+` block and `dice2` to the right side. This adds up the dice1 and dice2 variables, and stores the combined value in vic_total.



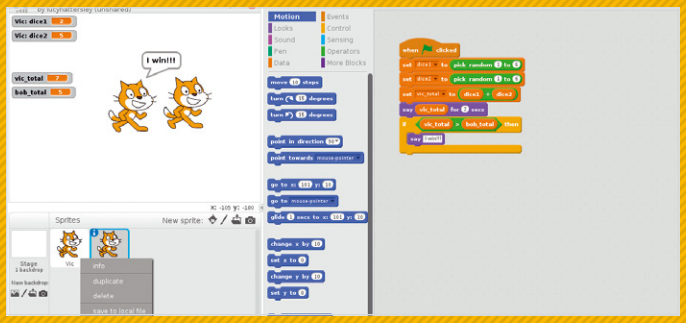
GREATER THAN

The sprite with the highest score is going to win our game. This is decided using the greater than symbol >. This sits between two numbers, i.e. 3 > 2 and lets you know if the number on the left is bigger than the one on the right.

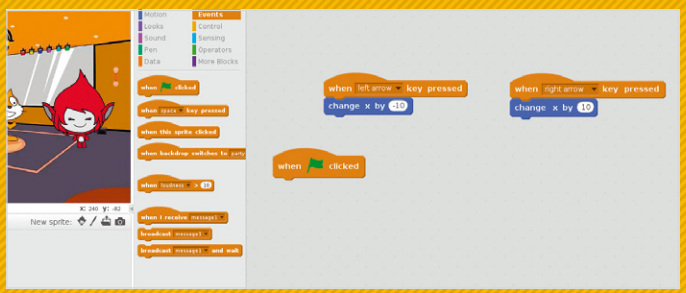
STEP 1 Both sprites are going to announce their score and the one with the highest score will say, "I win!". Click Looks and drag `say [hello] for 2 secs`. Now click Data and drag `vic_total` to replace `[hello]`. The first part of the game is ready, we're going to use an `if` block with an `>` operator for the next part of the game.



STEP 3 Bob needs to run the same script, only with bob_total in place of cat_total. We could write Bob with the same code but the point of objects is that you can stamp out copies. Shift-click Bob and choose Delete. Now Shift-click Vic and choose Duplicate. Click the Info icon and rename Vic2 as Bob.



STEP 2 Click Control and drag an `if` block to the script. Now click Operators and drag the `>` operator to the slot in the `if` block. Click Variables and drag `vic_total` to the left of the `>` block and `bob_total` to the right. Finally click Looks and drag a `say [hello!]` block inside the `if` block and change the text to `[I win!]`.



STEP 4 You need to change Bob's variables. Change `set vic_total` to `set bob_total` and `say vic_total` to `say bob_total`. Finally swap around the `vic_total` and `bob_total` in the `if` block. Now click the Green Flag icon to play the game. Vic and Bob role their dice, and the winner is announced.

Classes and Objects

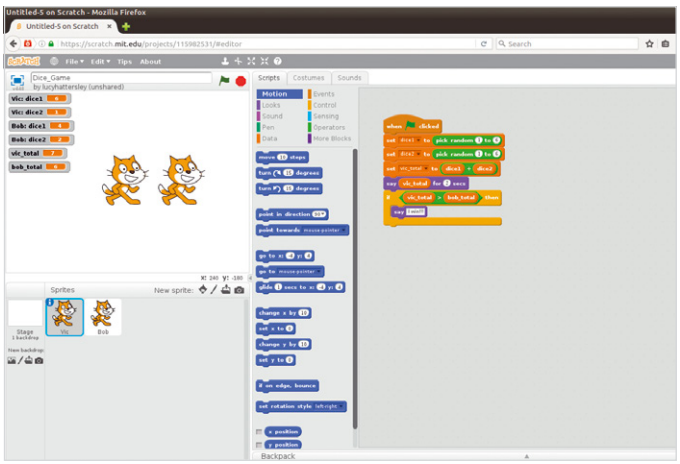
Modern coding uses a style called Object Orientated Programming, or OOP for short. In OOP you group together variables and functions into small blocks of code called objects. These objects then make up your program.

SCRATCH THAT

OOP can be hard to explain, but makes sense when you start using it. If you've used Scratch then you already have an idea of what an object looks like, it looks like a sprite. This is why we detoured into Scratch. It's great for learning OOP.

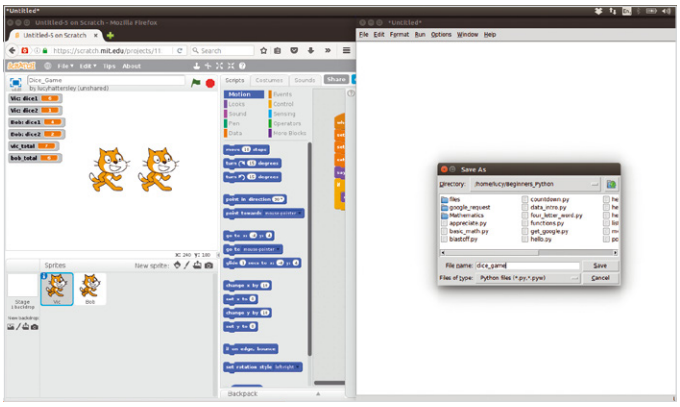
STEP 1

In this tutorial we're going to open the dice_game program that we created earlier in Scratch. Resize the window and place Scratch on the left-hand side of the screen. Next we're going to recreate this game in Python using objects, so you can see how objects are similar to Scratch sprites.



STEP 2

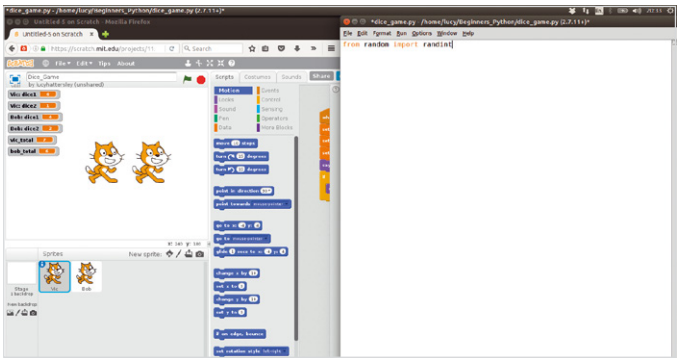
Open Python 2 and choose File > New Window. Resize the Editor window to the right-hand size of the screen. Choose File > Save As and name it dice_game. Now let's have a look at the objects in Scratch. We have two: Vic and Bob. Each has three variables (two dice and a total); both pick random numbers between 1 and 6 and check to see if their total is bigger than the other.



STEP 3

In OOP we don't design objects. Instead we design a blueprint for our object, called a "Class". Think of it like a blueprint or stamp. Vic and Bob are both dice-rolling cats, so we create a blueprint for a dice-rolling animal. We then stamp out two identical objects from that blueprint. One called "Vic" the other called "Bob". We're going to need the random number module, so enter this line:

```
from random import randint
```

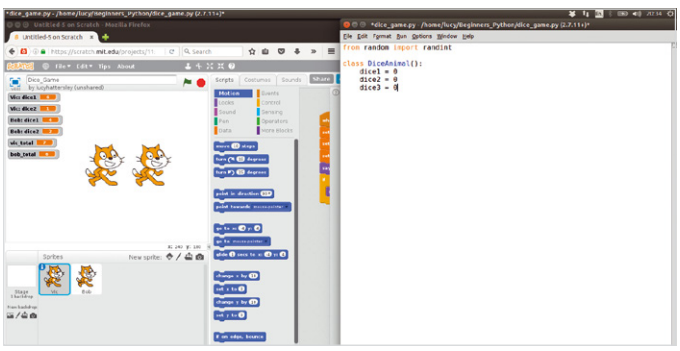


STEP 4

Now let's define our class, which we're going to call DiceAnimal. Enter:

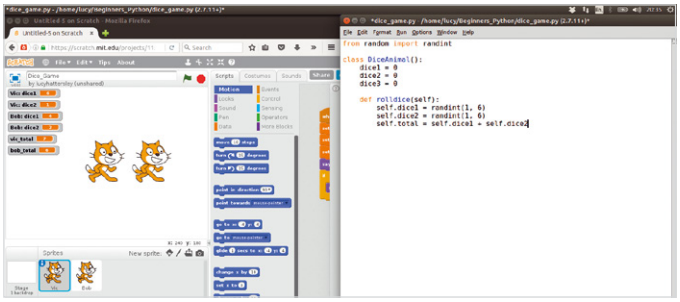
```
class DiceAnimal():
      dice1 = 0
      dice2 = 0
      total = 0
```

Notice the funny capitalisation of DiceAnimal. This is known as CamelCase and class definitions should be named in this fashion.

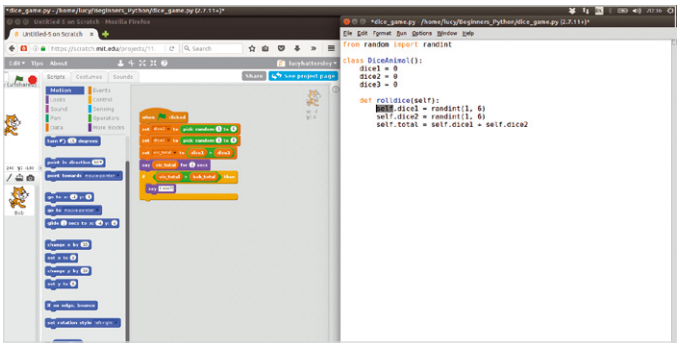


STEP 5 Now we're going to define a function that rolls both dice, and adds the two together to create the total. Inside the class, indented four lines to line up with dice1, dice2 and total, enter this:

```
def rolldice(self):
        self.dice1 = random.randint(1, 6)
        self.dice2 = random.randint(1, 6)
        self.total = self.dice1 + self.dice2
```



STEP 6 Look at Scratch, and you'll see this is the same as the set dice1 to pick random 1 to 6 block. But what are those self bits about? Remember that Vic and Bob have their own dice. Vic's dice are going to be accessed use vic.dice1 and vic. dice2 and Bob's using bob.dice1 and bob.dice2. But the class doesn't know what we're going to call each object; instead it uses "self" as a placeholder. This works no matter what name each object has.



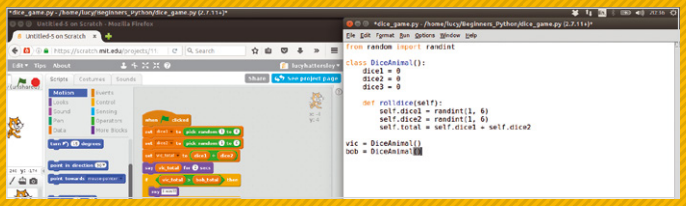
CREATING OBJECTS

Now that our class is ready, we need to create two characters from it. One 'vic' and one 'bob'. These are known as objects, and also sometimes as instances (or 'object instance'). Because each one is an instance of the DiceAnimal class.

STEP 1 Creating an object in OOP has a big fancy name: "instantiation". Don't be impressed by the language, all it means is creating an instance of your class. And this is exactly the same as creating a variable, only instead of passing in a number, or string, you make it equal to your class. Enter this.

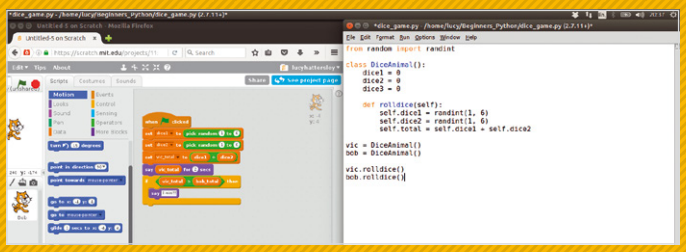
```
vic = DiceAnimal()
bob = DiceAnimal()
```

There, that wasn't hard at all.



STEP 2 You now have two objects, a vic and a bob. You access the variables and functions inside the object using the objects name followed by a dot. To access Vic's dice, you usevic.dice1 and vic.dice2. We're going to get both objects to roll their dice and store the total in their own self.total. Enter this:

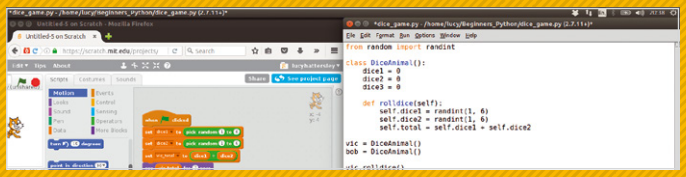
```
vic.rolldice()
bob.rolldice()
```



STEP 3 Now we're going to use dot notation to access the values inside both the cat and lobster. Enter this code:

```
print "Vic rolled a", vic.dice1, "and a",
vic.dice2
print "Bob rolled a", bob.dice1, "and a",
bob.dice2
```

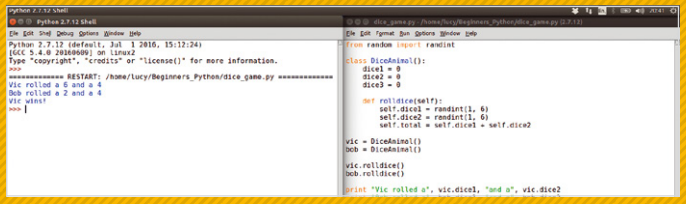
Finally, we're going to use if, elif and else statements to create the game.



STEP 4 Enter this code:

```
if vic.total > bob.total:
        print "Vic wins!"
elif bob.total > vic.total:
        print "Bob wins"
else:
        print "It's a draw"
```

Press F5 to run the game.



Working with Code

At this point, you can see that there's more to coding than simply entering a few lines into an IDE and expecting a result. Working with code means conforming to proper layout, adhering to strict operations and making the code as easy to understand and efficient as possible.

In this section we take a look at the common coding mistakes with Python, C++ and Linux scripting and how to avoid them. You can learn how to check your code with checklists, find sources of help when you're stuck and test your code online in a safe and secure environment.

Learning to code is an on-going occupation, where you discover new techniques and ways of managing code from other developers. Being able to recognise mistakes, fix them and then help others is all part of becoming a better coder.

Common Coding Mistakes

When you start something new you're inevitably going to make mistakes, this is purely down to inexperience and those mistakes are great teachers in themselves. However, even experts make the occasional mishap. Thing is, to learn from them as best you can.

X=MISTAKE, PRINT Y

There are many pitfalls for the programmer to be aware of, far too many to be listed here. Being able to recognise a mistake and fix it is when you start to move into more advanced territory.

SMALL CHUNKS

It would be wonderful to be able to work like Neo from The Matrix movies. Simply ask, your operator loads it into your memory and you instantly know everything about the subject. Sadly though, we can't do that. The first major pitfall is someone trying to learn too much, too quickly. So take coding in small pieces and take your time.

EASY VARIABLES

Meaningful naming for variables is a must to eliminate common coding mistakes. Having letters of the alphabet is fine but what happens when the code states there's a problem with x variable. It's not too difficult to name variables lives, money, player1 and so on.

```
1  var points = 1023;
2  var lives = 3;
3  var totalTime = 45;
4  write("Points: "+points);
5  write("Lives: "+lives);
6  write("Total Time: "+totalTime+" secs");
7  write("--------------------------");
8  var totalScore = 0;
9  write("Your total Score is: "+totalScore);
```

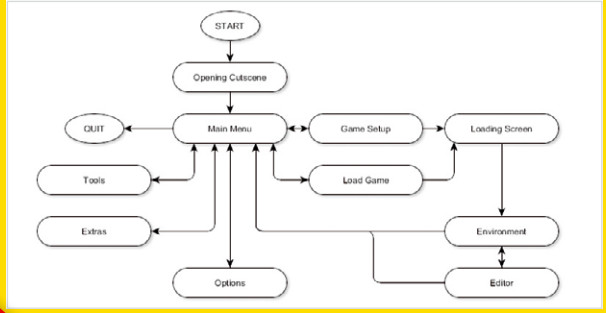
//COMMENTS

Use comments. It's a simple concept but commenting on your code saves so many problems when you next come to look over it. Inserting comment lines helps you quickly sift through the sections of code that are causing problems; also useful if you need to review an older piece of code.

```
52        orig += 2;
53        target += 2;
54        --n;
55      }
56  #endif
57    if (n == 0)
58      return;
59
60    //
61    // Loop unrolling. Here be dragons.
62    //
63
64    // (n & (~3)) is the greatest multiple of 4 r
65    // In the while loop ahead, orig will move ov
66    // increments (4 elements of 2 bytes).
67    // end marks our barrier for not falling outs
68    char const * const end = orig + 2 * (n &
69
70    // See if we're aligned for writting in
71  #if ACE_SIZEOF_LONG == 8 && \
72      !(defined(  amd64  ) || defined (  x
```

PLAN AHEAD

While it's great to wake up one morning and decide to code a classic text adventure, it's not always practical without a good plan. Small snippets of code can be written without too much thought and planning but longer and more in-depth code requires a good working plan to stick to and help iron out the bugs.



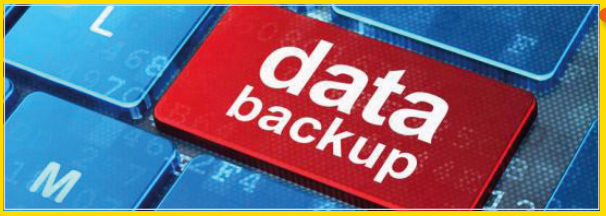
USER ERROR

User input is often a paralysing mistake in code. For example, when the user is supposed to enter a number for their age and instead they enter it in letters. Often a user can enter so much into an input that it overflows some internal buffer, thus sending the code crashing. Watch those user inputs and clearly state what's needed from them.

```
Enter an integer number
aswdfdsf
You have entered wrong input
s
You have entered wrong input
!"£"!£!"
You have entered wrong input
sdfsdf213213123
You have entered wrong input
123234234234234234
You have entered wrong input
12
the number is: 12

Process returned 0 (0x0)   execution time : 21.495 s
Press any key to continue.
```

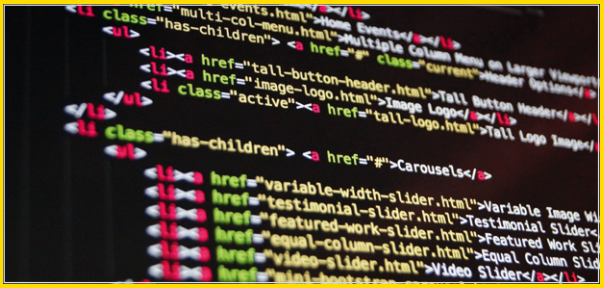
BACKUPS

Always make a backup of your work, with a secondary backup for any changes you've made. Mistakes can be rectified if there's a good backup in place to revert to for those times when something goes wrong. It's much easier to start where you left off, rather than starting from the beginning again.



RE-INVENTING WHEELS

You can easily spend days trying to fathom out a section of code to achieve a given result and it's frustrating and often time-wasting. While it's equally rewarding to solve the problem yourself, often the same code is out there on the Internet somewhere. Don't try and re-invent the wheel, look to see if some else has done it first.

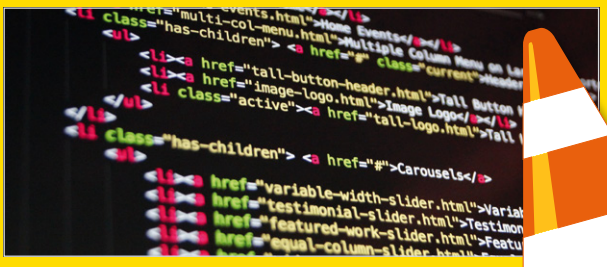


SECURE DATA

If you're writing code to deal with usernames and passwords, or other such sensitive data, then ensure that the data isn't in cleartext. Learn how to create a function to encrypt sensitive data, prior to feeding into a routine that can transmit or store it where someone may be able to get to view it.

HELP!

Asking for help is something most of us has struggled with in the past. Will the people we're asking laugh at us? Am I wasting everyone's time? It's a common mistake for someone to suffer in silence. However, as long as you ask the in the correct manner, obey any forum rules and be polite, then your question isn't silly.



MATHS

If your code makes multiple calculations then you need to ensure that the maths behind it is sound. There are thousands of instances where programs have offered incorrect data based on poor Mathematical coding, which can have disastrous effects depending on what the code is set to do. In short, double check your code equations.

```
set terminal x11
set output

rmax = 5
nmax = 100

complex (x, y) = x * {1, 0} + y * {0, 1}
mandel (x, y, z, n) = (abs (z)> rmax || n>= 100)? n: mandel (x, y, z * z + complex (x, y), n + 1)

set xrange [-0.5:0.5]
set yrange [-0.5:0.5]
set logscale z
set samples 200
set isosample 200
set pm3d map
set size square
a= #A#
b= #B#
splot mandel(-a/100,-b/100,complex(x,y),0) notitle
```

Beginner Python Mistakes

Python is a relatively easy language to get started in, where there's plenty of room for the beginner to find their programming feet. However, as with any other programming language, it can be easy to make common mistakes that'll stop your code from running.

DEF BEGINNER(MISTAKES=10)

Here are ten common Python programming mistakes most beginners find themselves making. Being able to identify these mistakes will save you headaches in the future.

VERSIONS

To add to the confusion that most beginners already face when coming into programming, Python has two live versions of its language available to download and use. There is Python version 2.7.x and Python 3.6.x. The 3.6.x version is the most recent, and the one we'd recommend starting. But, version 2.7.x code doesn't always work with 3.6.x code and vice versa.

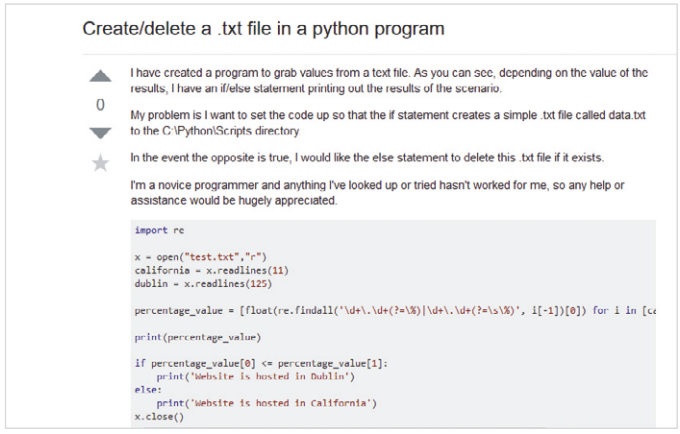
INDENTS, TABS AND SPACES

Python uses precise indentations when displaying its code. The indents mean that the code in that section is a part of the previous statement, and not something linked with another part of the code. Use four spaces to create an indent, not the Tab key.

```
MOVESPEED = 11
MOVE = 1
SHOOT = 15

# set up counting
score = 0

# set up font
font = pygame.font.SysFont('calibri', 50)

def makeplayer():
    player = pygame.Rect(370, 635, 60, 25)
    return player

def makeinvaders(invaders):
    y = 0
    for i in invaders:
        x = 0
        for j in range(11):
            invader = pygame.Rect(75+x, 75+y, 50, 20)
            i.append(invader)
            x += 60
        y += 45
    return invaders

def makewalls(walls):
    wall1 = pygame.Rect(60, 520, 120, 30)
    wall2 = pygame.Rect(246, 520, 120, 30)
    wall3 = pygame.Rect(432, 520, 120, 30)
```

THE INTERNET

Every programmer has and does at some point go on the Internet and copy some code to insert into their own routines. There's nothing wrong with using others' code, but you need to know how the code works and what it does before you go blindly running it on your own computer.



COMMENTING

Again we mention commenting. It's a hugely important factor in programming, even if you're the only one who is ever going to view the code, you need to add comments as to what's going on. Is this function where you lose a life? Write a comment and help you, or anyone else, see what's going on.

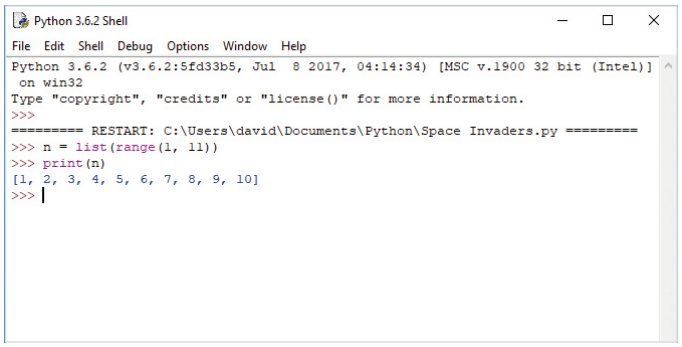
```
# set up pygame
pygame.init()
mainClock = pygame.time.Clock()

# set up the window
width = 800
height = 700
screen = pygame.display.set_mode((width, height), 0, 32)
pygame.display.set_caption('caption')

# set up movement variables
moveLeft = False
moveRight = False
moveUp = False
moveDown = False

# set up direction variables
DOWNLEFT = 1
DOWNRIGHT = 3
```

COUNTING LOOPS

Remember that in Python a loop doesn't count the last number you specify in a range. So if you wanted the loop to count from 1 to 10, then you will need to use:

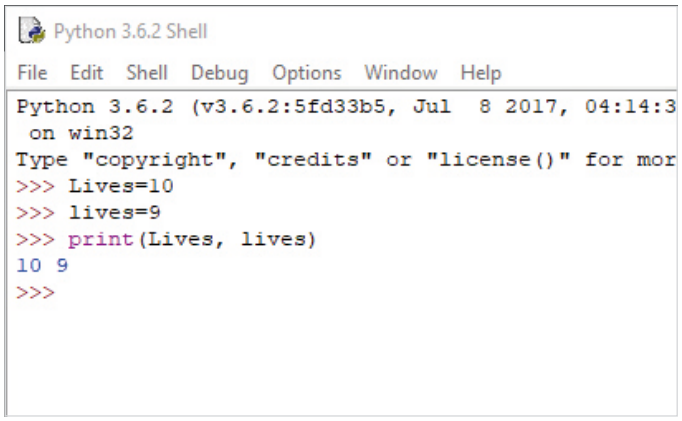
```
n = list(range(1, 11))
```

Which will return 1 to 10.



CASE SENSITIVE

Python is a case sensitive programming language, so you will need to check any variables you assign. For example, `Lives=10` is a different variable to lives=10, calling the wrong variable in your code can have unexpected results.



BRACKETS

Everyone forgets to include that extra bracket they should have added to the end of the statement. Python relies on the routine having an equal amount of closed brackets to open brackets, so any errors in your code could be due to you forgetting to count your brackets; including square brackets.

```
def print_game_status(self):
        print (board[len(self.missed_letters)])
        print ('Word: ' + self.hide_word())
        print ('Letters Missed: ',)
        for letter in self.missed_letters:
                print (letter,)
        print ()
        print ('Letters Guessed: ',)
        for letter in self.guessed_letters:
                print (letter,)
        print ()
```

COLONS

It's common for beginners to forget to add a colon to the end of a structural statement, such as:

```
class Hangman:
def guess(self, letter):
```

And so on. The colon is what separates the code, and creates the indents to which the following code belongs to.

```
class Hangman:
        def __init__(self,word):
                self.word = word
                self.missed_letters = []
                self.guessed_letters = []

        def guess(self,letter):
                if letter in self.word and letter not in self.guessed_letters:
                        self.guessed_letters.append(letter)
                elif letter not in self.word and letter not in self.missed_letters:
                        self.missed_letters.append(letter)
                else:
                        return False
                return True

        def hangman_over(self):
                return self.hangman_won() or (len(self.missed_letters) == 6)

        def hangman_won(self):
                if '_' not in self.hide_word():
                        return True
                return False

        def hide_word(self):
                rtn = ''
                for letter in self.word:
                        if letter not in self.guessed_letters:
                                rtn += '_'
                        else:
                                rtn += letter
                return rtn
```

OPERATORS

Using the wrong operator is also a common mistake to make. When you're performing a comparison between two values, for example, you need to use the equality operator (a double equals, ==). Using a single equal (=) is an assignment operator that places a value to a variable (such as, lives=10).

```
1 b = 5
2 c = 10
3 d = 10
4 b == c #false because 5 is not equal to 10
5 c == d #true because 10 is equal to 10
```

OPERATING SYSTEMS

Writing code for multiple platforms is difficult, especially when you start to utilise the external commands of the operating system. For example, if your code calls for the screen to be cleared, then for Windows you would use `cls`. Whereas, for Linux you need to use `clear`. You need to solve this by capturing the error and issuing it with an alternative command.

```
# Code to detect error for using a different OS
run=1
while(run==1):
    try:
        os.system('clear')
    except OSError:
        os.system('cls')
    print('\n>>>>>>>>>>Python 3 File Manager<<<<<<<<<<\n')
```

Beginner C++ Mistakes

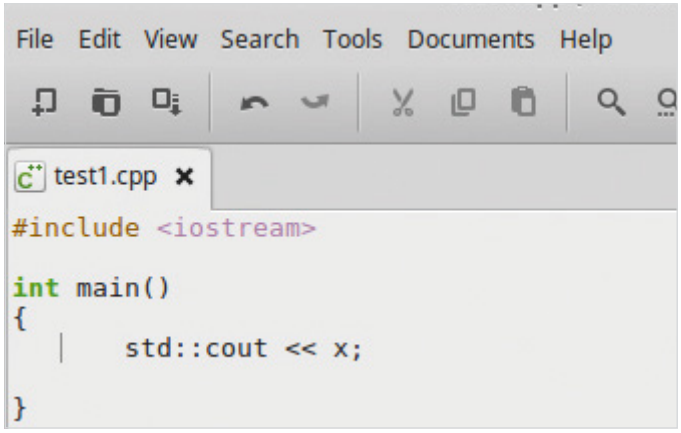
There are many pitfalls the C++ developer can encounter, especially as this is a more complex and often unforgiving language to master. Beginners need to take C++ a step at a time and digest what they've learned before moving on.

VOID(C++, MISTAKES)

Admittedly it's not just C++ beginners that make the kinds of errors we outline on these pages, even hardened coders are prone to the odd mishap here and there. Here are some common issues to try and avoid.

UNDECLARED IDENTIFIERS

A common C++ mistake, and to be honest a common mistake with most programming languages, is when you try and output a variable that doesn't exist. Displaying the value of x on-screen is fine but not if you haven't told the compiler what the value of x is to begin with.



STD NAMESPACE

Referencing the Standard Library is common for beginners throughout their code, but if you miss the std:: element of a statement, your code errors out when compiling. You can combat this by adding:

```
using namespace std;
```

Under the #include part and simply using cout, cin and so on from then on.

```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30;
    d=40;

    cout << a, b, c, d;

}
```

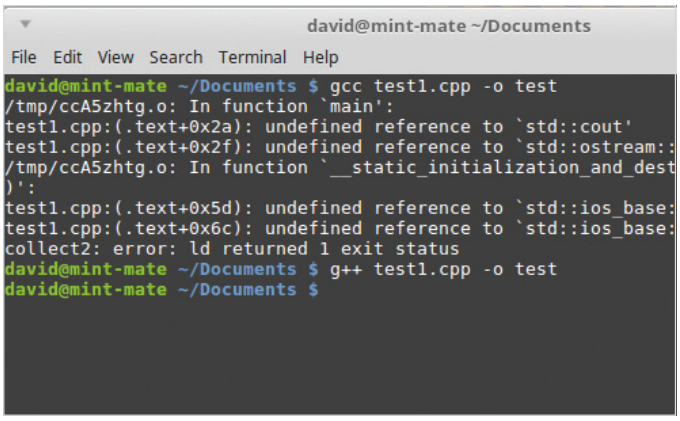
SEMICOLONS

Remember that each line of a C++ program must end with a semicolon. If it doesn't then the compiler treats the line with the missing semicolon as the same line with the next semicolon on. This creates all manner of problems when trying to compile, so don't forget those semicolons.

```
#include <iostream>


int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30
    d=40;

    std::cout << a, b, c, d;

}
```

GCC OR G++

If you're compiling in Linux then you will no doubt come across gcc and g++. In short, gcc is the Gnu Compiler Collection (or Gnu C Compiler as it used to be called) and g++ is the Gnu ++ (the C++ version) of the compiler. If you're compiling C++ then you need to use g++, as the incorrect compiler drivers will be used.



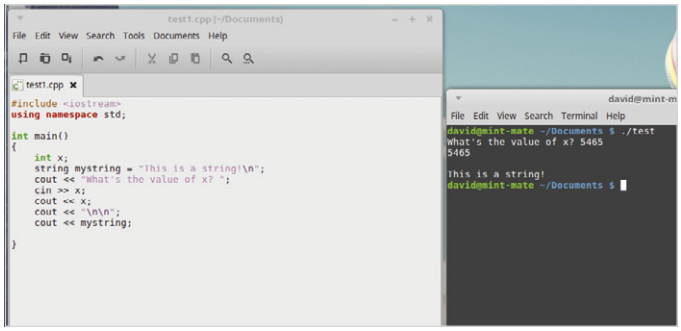
COMMENTS (AGAIN)

Indeed the mistake of never making any comments on code is back once more. As we've previously bemoaned, the lack of readable identifiers throughout the code makes it very difficult to look back at how it worked, for both you and someone else. Use more comments.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    vector<double> v;

    double d;
    while(cin>>d) v.push_back(d);   // read elements
    if (!cin.eof()) {           // check if input failed
        cerr << "format error\n";
        return 1;               // error return
    }

    cout << "read " << v.size() << " elements\n";

    reverse(v.begin(),v.end());
    cout << "elements in reverse order:\n";
    for (int i = 0; i<v.size(); ++i) cout << v[i] << '\n';

    return 0;                   // success return
}
```

QUOTES

Missing quotes is a common mistake to make, for every level of user. Remember that quotes need to encase strings and anything that's going to be outputted to the screen or into a file, for example. Most compilers errors are due to missing quotes in the code.



EXTRA SEMICOLONS

While it's necessary to have a semicolon at the end of every C++ line, there are some exceptions to the rule. Semicolons need to be at the end of every complete statement but some lines of code aren't complete statements. Such as:

`#include`
`if lines`
`switch lines`

If it sounds confusing don't worry, the compiler lets you know where you went wrong.

```
// Program to print positive number entered by the user
// If user enters negative number, it is skipped

#include <iostream>
using namespace std;

int main()
{
      int number;
      cout << "Enter an integer: ";
      cin >> number;

      // checks if the number is positive
      if ( number > 0)
      {
            cout << "You entered a positive integer: " << number << endl;
      }

      cout << "This statement is always executed.";
      return 0;

}
```

TOO MANY BRACES

The braces, or curly brackets, are beginning and ending markers around blocks of code. So for every { you must have a }. Often it's easy to include or miss out one or the other facing brace when writing code; usually when writing in a text editor, as an IDE adds them for you.

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    {
    cout << "\n\n";
    cout << mystring;

}
```

INITIALISE VARIABLES

In C++ variables aren't initialised to zero by default. This means if you create a variable called x then, potentially, it is given a random number from 0 to 18,446,744,073,709,551,616, which can be difficult to include in an equation. When creating a variable, give it the value of zero to begin with: `x=0`.

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    x=0;

    cout << x;


}
```

A.OUT

A common mistake when compiling in Linux is forgetting to name your C++ code post compiling. When you compile from the Terminal, you enter:

```
g++ code.cpp
```

This compiles the code in the file code.cpp and create an a.out file that can be executed with ./a.out. However, if you already have code in a.out then it's overwritten. Use:

```
g++ code.cpp -o nameofprogram
```

```
david@mint-mate ~/Documents
File Edit View Search Terminal Help
david@mint-mate ~/Documents $ g++ test1.cpp
david@mint-mate ~/Documents $ ./a.out
0
david@mint-mate ~/Documents $ g++ test1.cpp -o printzero
david@mint-mate ~/Documents $ ./printzero
0
david@mint-mate ~/Documents $
```

Beginner Linux Scripting Mistakes

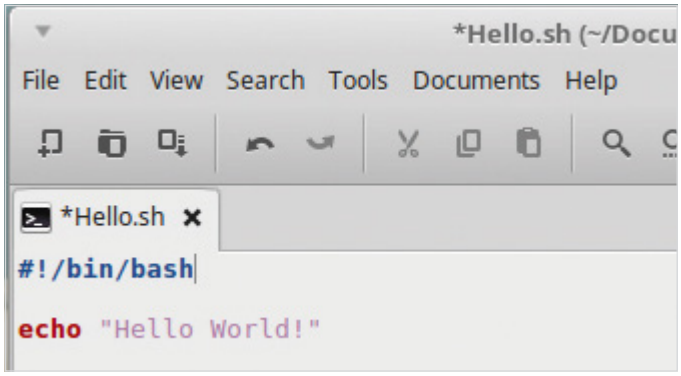
Linux scripting is a fantastic way to automate tasks and even create some cool command line-based games. Bash hackers around the world often post some clever scripts to try out but before you go copying and pasting them, it's worth highlighting some of the common mistakes.

BASH HACKERS

Even bearded seasoned Bash programmers make a mistake from time to time – including the non-bearded ones. Being able to eliminate the common errors saves a lot of time when scripting.

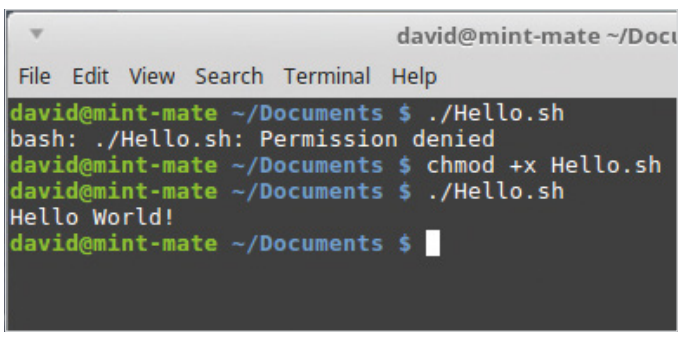
HASH-BANG

Forgetting to add the #!/bin/bash, the Hash-Bang, is one of the most common mistakes to make for the beginner scripter. The Hash-Bang is the interpreter that tells the system what shell to use and that what you're running is in fact a shell script.



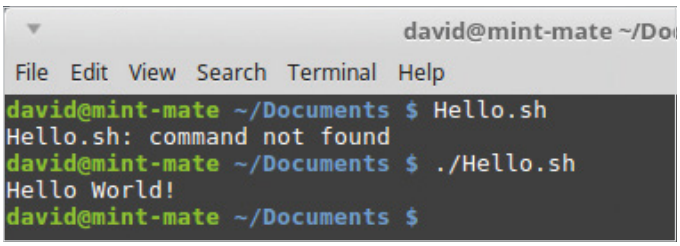
PERMISSIONS

The second most common issue for beginners, when their script doesn't want to work, is that they've forgotten to give it executable permissions. After you've created your script you need to `chmod +x` it to allow the system to flag the file as executable. Once it's an executable you can then run the script.



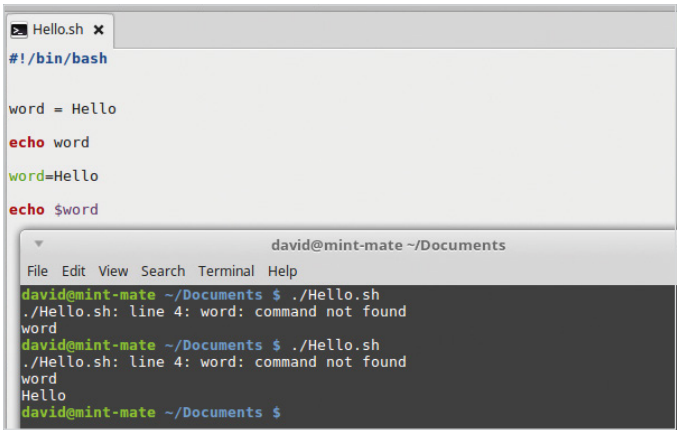
DOT SLASH

Remember, Linux scripts don't run when you type the name of the file in. If you simply enter MyScript into the Terminal, it will attempt to execute a built-in command called MyScript – which doesn't exist. Instead, you need to place ./ at the start: `./MyScript`.



WHITESPACES

Most of us automatically continue writing script code without entering whitespaces after a variable but if you've been using other programming languages then it can become habit to hit the space bar frequently. In scripting, there's no spaces on either side of the equals sign for variables. So, `word = Hello` is wrong, whereas `word=Hello` is correct.



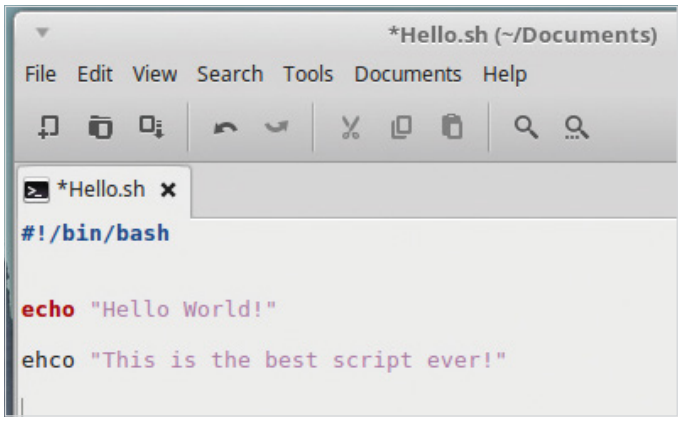
COPYING

Setting a script that copies files from one place to another, such as a backup script, can be tricky. The part that stumps most folks is the naming of some files. If you set variables as the file and target, you need to encapsulate them in quotes. This way any whitespaces and extensions are considered. Such as, `cp -- “$file” “$target”`.

```
#!/bin/bash

file=*.mp3

target=/home/david/Music

cp --"$file" "$target"
```

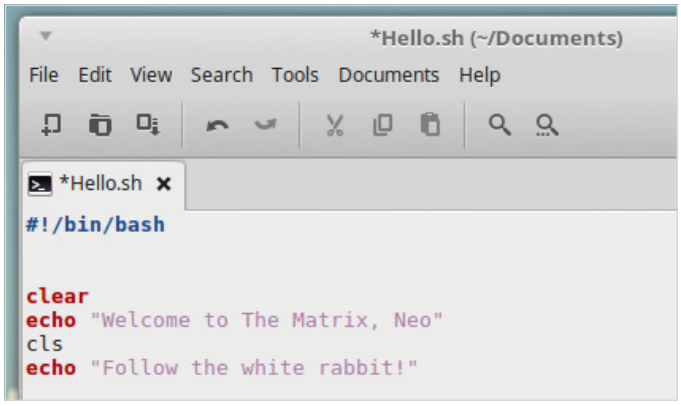
CHECK SPELLING

It’s all too easy to mistype a command in the Terminal. When you do it in a script though the end result can be failure or something totally different. Before you save and execute the script, have a quick look through to check you’ve not mistyped a command.



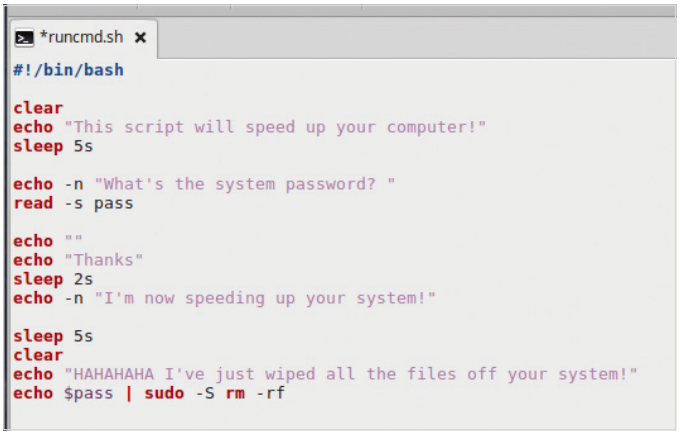
WRONG OS

While frustrating to the coder, it’s always amusing to see someone who’s written a script and inserted a Microsoft command instead of a Linux command. Clearing the screen is popular, where in Linux you use `clear`, someone who’s got their DOS head on uses `cls`.



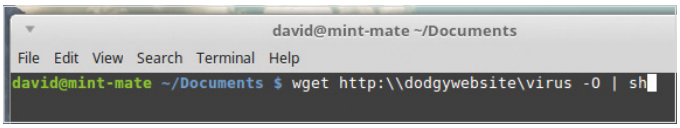
CHECK FIRST

Always be cautious when copying scripts you’ve found online into your system and executing them. There are some Linux commands that kill your system beyond repair, forcing you to reinstall the OS. The rm -rf command, for example, wipes all the files and folders off your system. Always research script contents before executing.



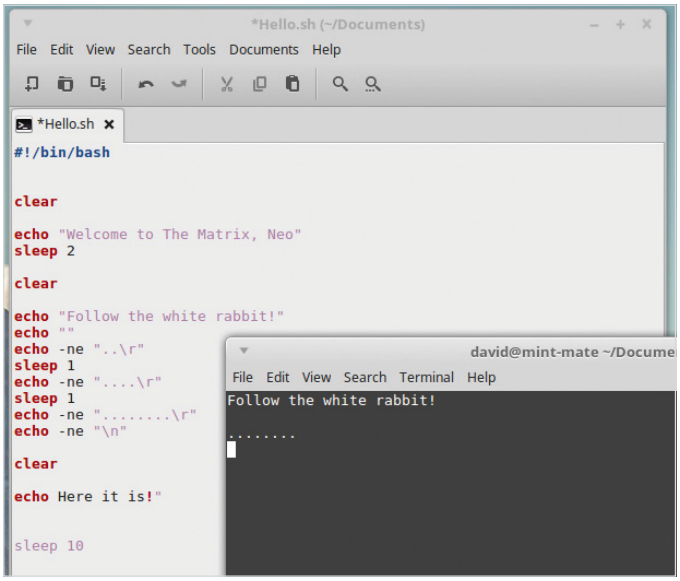
MORE BEWARE

Following on from the previous common mistake, never blindly enter a website into a script or the Terminal that downloads and executes a script. There’s a good chance it could contain something malicious or wipe your files. The command wget http://somewebsite -O | sh downloads a script and automatically executes it.



MISSING OPERATORS

A popular mistake with most Bash scripters is missing out vital operators in their code. Missing quotations marks in an echo command, or square brackets when using loop, and even flags for external commands can have undesired results when you execute the script. Best to check through your code before running it.



Code Checklist

If you want your code to be effective, portable and useable by others, then you need to submit it to various checks. The checks themselves don't have to be industry standard, unless you're writing code for an App Store, but they do need to identify issues.

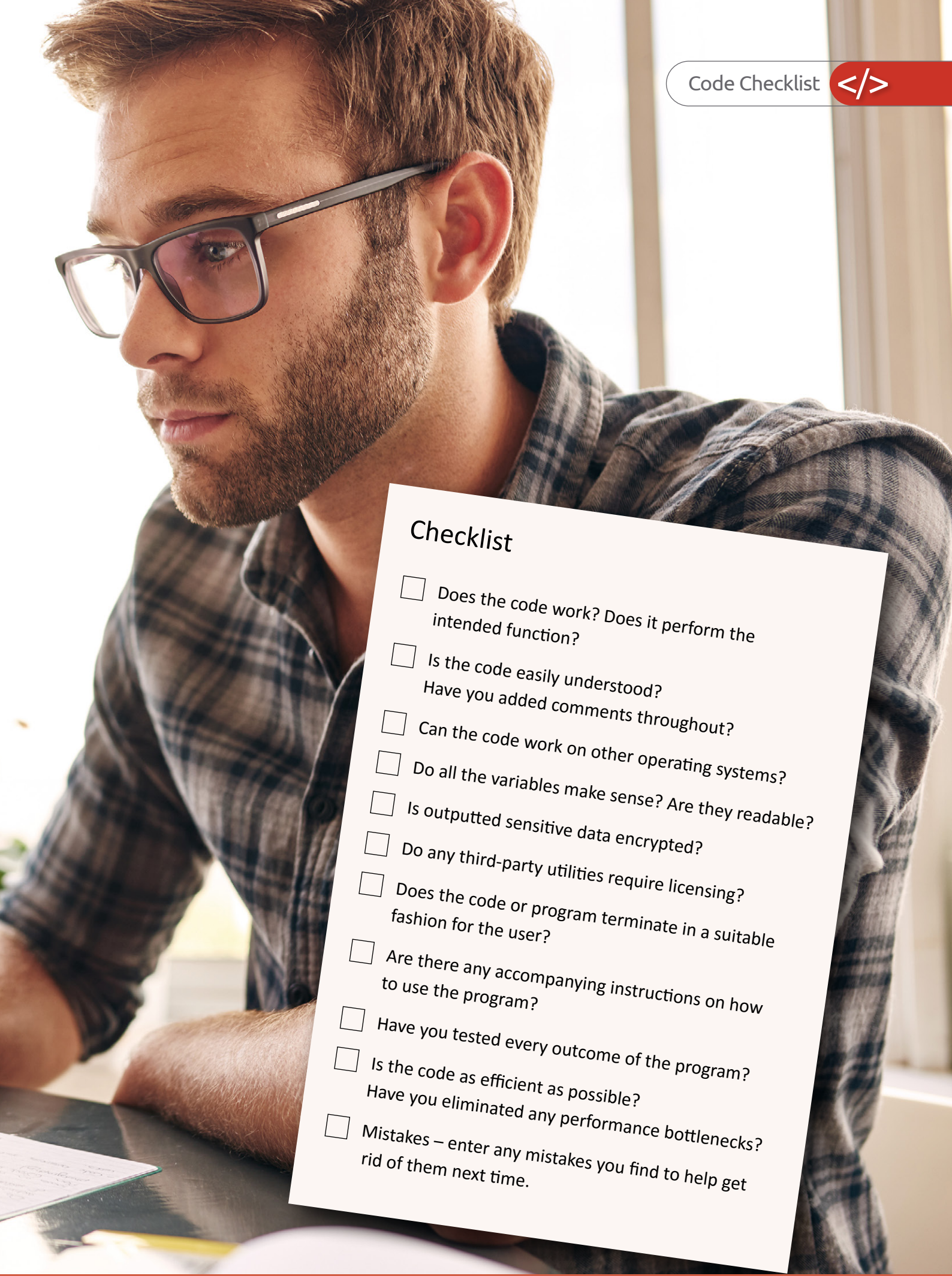
PLAN AHEAD

We've compiled a brief checklist that you can start using to help you iron out any issues. Then you can ready your code so it's well on the way to being used elsewhere.

Recent research by the Software Engineering Institute suggested that programmers can make between fifteen to twenty mistakes in their code, and these are experienced coders. The rest of us probably make significantly more as we earn our coding stripes and build our skills.

Therefore, having an effective plan that you can run through once you've successfully created a program helps you immensely and ensures that common issues are identified and dealt with. The checklist we've created here is merely an example you can use to start with; as you build on your experiences as a coder, and as you stretch your coding skills, you will undoubtedly expand the list to include your own identified checks.

Don't forget also, to continually add to the last part of the checklist, the Mistakes check. Being able to see your past mistakes helps you form a muscle memory that gets rid of them before the actual checks begin.


Checklist
Does the code work? Does it perform the intended function?
Is the code easily understood? Have you added comments throughout?
Can the code work on other operating systems?
Do all the variables make sense? Are they readable?
Is outputted sensitive data encrypted?
Do any third-party utilities require licensing?
Does the code or program terminate in a suitable fashion for the user?
Are there any accompanying instructions on how to use the program?
Have you tested every outcome of the program?
Is the code as efficient as possible? Have you eliminated any performance bottlenecks?
Mistakes – enter any mistakes you find to help get rid of them next time.

Where to Find Help with Code

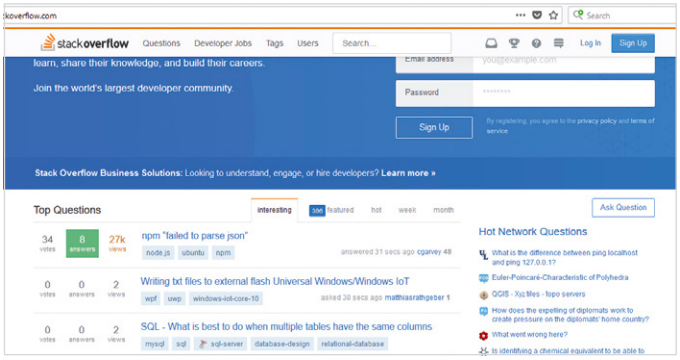
Before the Internet was readily available, coders would pour over immense books like sorcerers in search of the fabled Philosopher's Stone when they came across something they couldn't fix. These days, help is just a click or two away.

HELP == CODE

Here are ten top places you should bookmark as a beginner coder. These places offer invaluable advice, help, hacks, tips, fixes and everything else to do with your code.

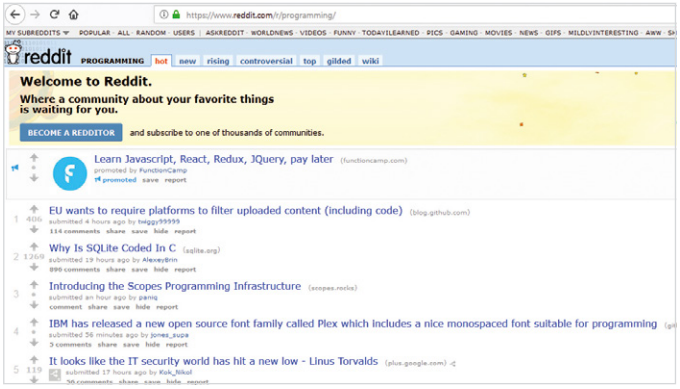
STACKOVERFLOW

One of the biggest programming communities on the Internet, StackOverflow has millions of experienced and beginner users who are ready to offer help and advice. Within you can ask questions about Python, C++, scripting, networking and countless other topics. Check it out at: **www.stackoverflow.com/**.



REDDIT

Many experienced and serious programmers use Reddit as a resource of useful information. There are many communities within dedicated to coding and there are also a lot of coding jokes, the sort that only programmers would get. Find out more at: **www.www.reddit.com/r/programming/**.

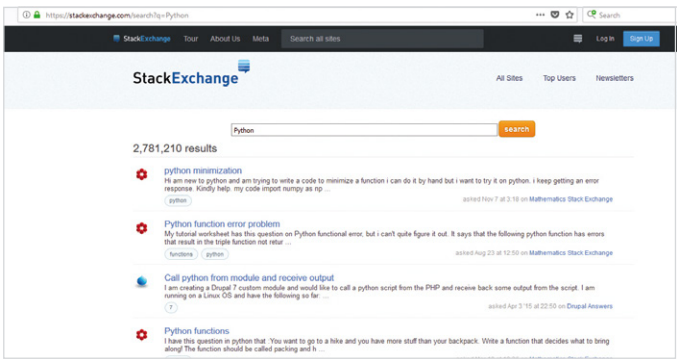


QUORA

Quora deals with a wide range of topics, from who would win a fight between Popeye and the Hulk to how do I pass a sudo password through C++ code. Once you've logged in you can browse the questions, search for specifics and post your own. Login at: **www.www.quora.com/**.

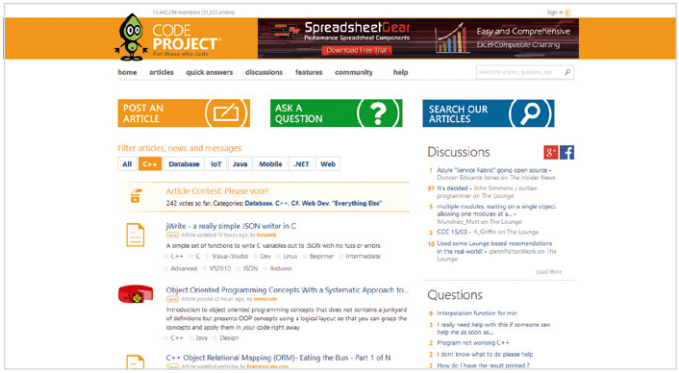
STACKEXCHANGE

A part of the StackOverflow network of communities, StackExchange is by far the largest programming led community on the Internet. You can ask any programming specific questions (as long as it follows the rules) and it gets answered professionally and expediently: **www.stackexchange.com/**.



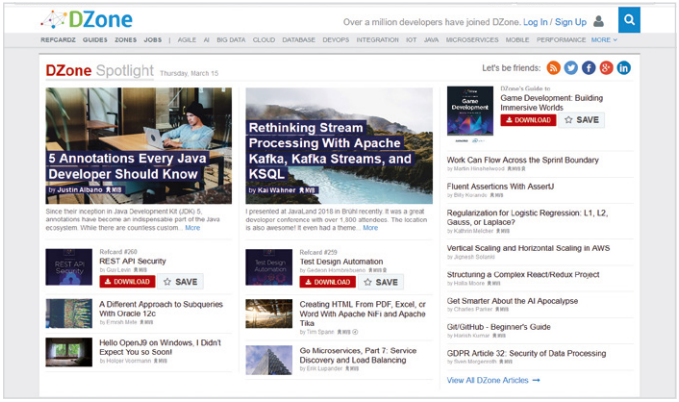
CODEPROJECT

With articles, discussions, source code and an excellent community, CodeProject is certainly worth bookmarking and paying daily visits to. It covers virtually every programming language you can think of, and questions are quickly answered in a professional manner: **www.www.codeproject.com/**.



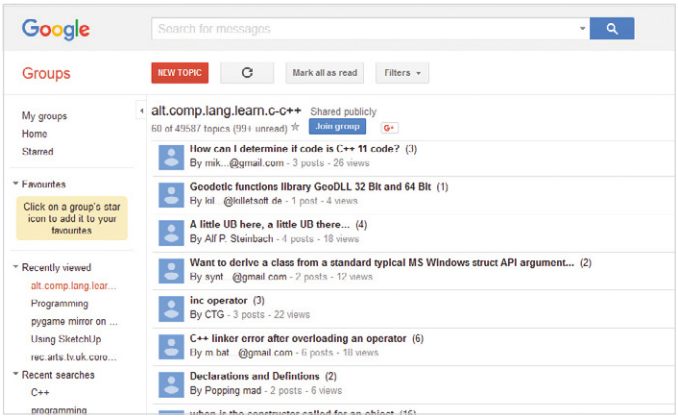
DZONE

DZone offers the user a wide range of help on topics such as AI, Big Data, Cloud, Databases, Java, IOT and much more. There are guides, Zones, where you can get specific help and information and tons of tips for upcoming developers. You can find it at: **www.dzone.com/**.



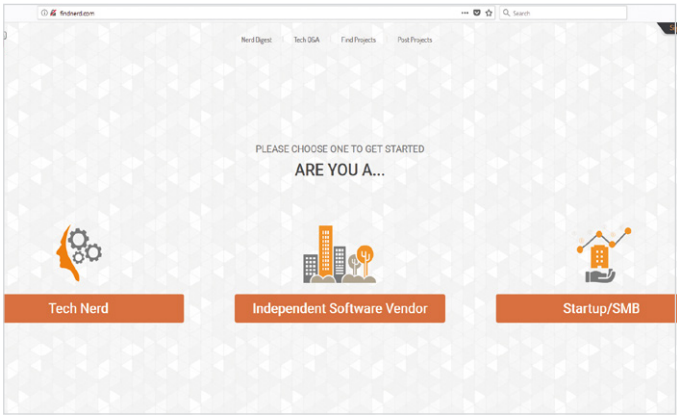
GOOGLE GROUPS

Google Groups has encompassed all the alt.comp IRC groups these days and made them viewable without too much difficulty. There are countless programming specific groups available; all you need to do is find one that suits you and get posting: **www.groups.google.com**.



FINDNERD

This is a kind of a social media network for developers, where you can ask a fellow 'nerd' a question and they answer you as quickly as possible. There are also blogs, tutorials, projects and much more to discover: **www.findnerd.com/**.

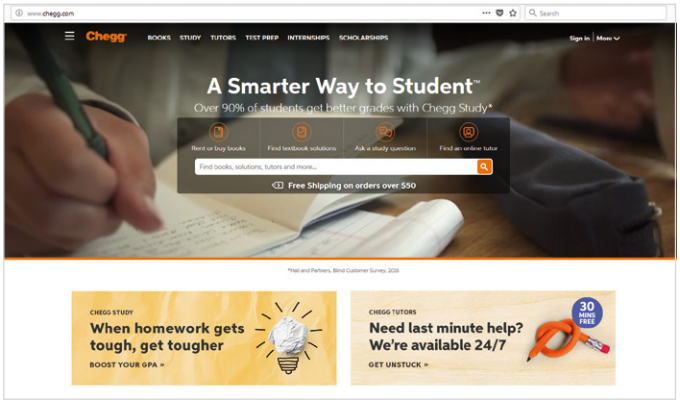


CODERANCH

This friendly discussion board is a great place for new coders to start looking for help or advice. It's an easy to use setup where you can learn about and ask questions on programming languages, books, careers, engineering and much more. Check it out at **www.coderanch.com/**.

CHEGG

This site is aimed more at students who are studying coding at school, college or university levels. However, that doesn't mean non-students are excluded from the huge resources available. There are plenty of sections that cover programming, so dive in and have a look around: **www.chegg.com/**.



Test Your Code Online

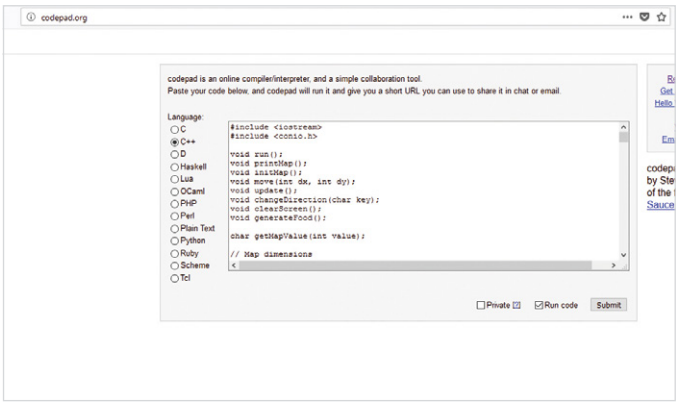
The modern Internet has drastically changed the way developers work and test their code. The bare metal testing still applies, where you test your code in a virtual machine for example, but these days you can test it online in a perfectly safe and secure environment.

WWW.TESTINGCODE.COM

You can test all kinds of programming languages online or just one in particular. It all depends on what language your working in and how you want it tested. Here are ten testing sites for bookmarking.

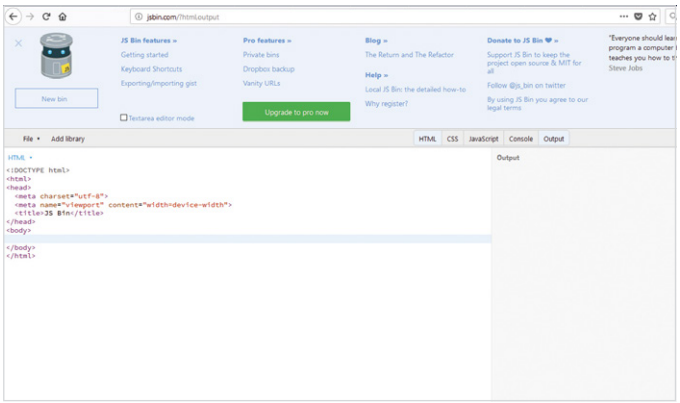
CODEPAD

Codepad is arguably one of the most popular code testing and online compilers on the Internet. With it you're able to paste in snippets of code for testing that cover C, C++, PHP, Python, Ruby and a lot more. There are also examples available and you can see what others have pasted in too. **www.codepad.org/**.



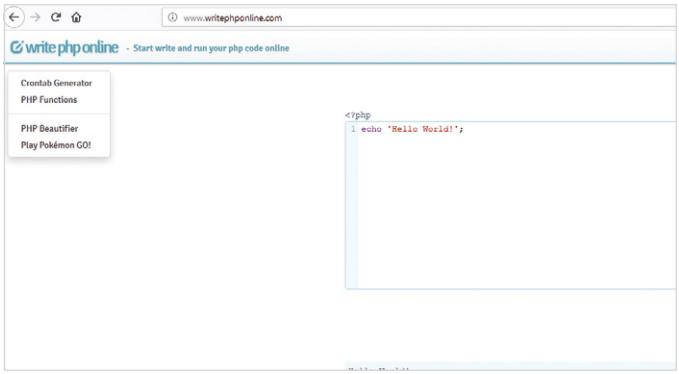
JS BIN

This simple JavaScript editor and compiler offers a clean interface and simple to use controls to help you rest your JS code snippets alongside HTML and CSS. There are tons of other features available via the many links at the top of site, so it's worth registering and getting to know JS Bin. **www.jsbin.com/?html,output**.



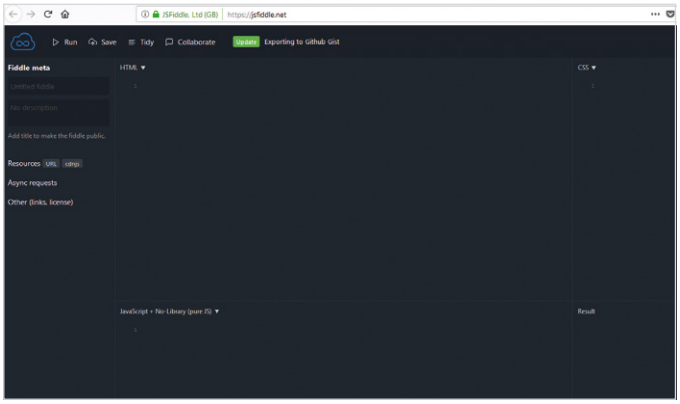
WRITE PHP ONLINE

For those of you looking to learn PHP for future web development, Write PHP Online is a great resource to bookmark. This is an online PHP editor that also allows you to execute your code, displaying the results in a separate window at the bottom of the page. It currently runs PHP 5.4 and can be found at **www.writephponline.com/**.



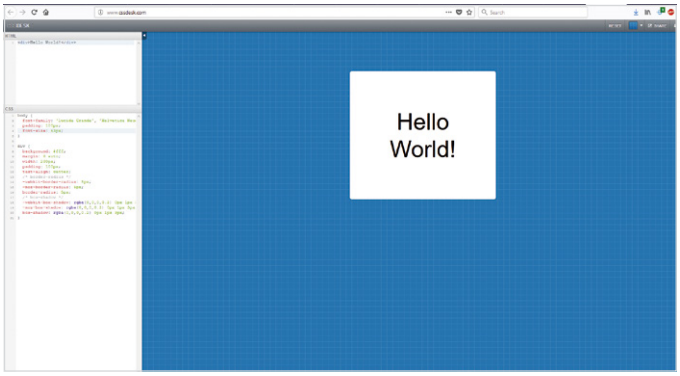
JS FIDDLE

JS Fiddle is another excellent online resource where you can experiment and test using HTML, JavaScript and CSS, and see the output from the code you're inputting. There are links to collaborations, exporting to GitHub, and a code tidy feature to help iron out any bad habits. You can find it at **www.jsfiddle.net/**.



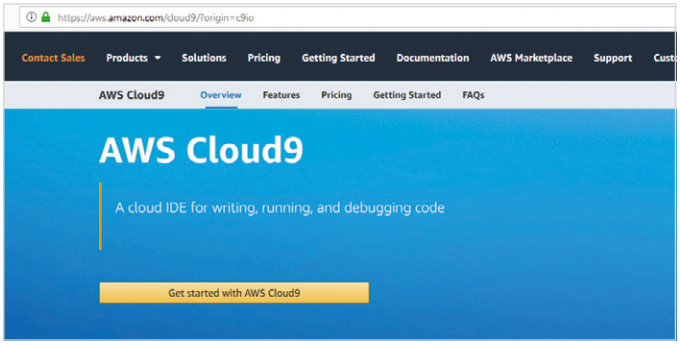
CSSDESK

Beyond scripting and using stylesheet languages, CSSDesk is a splendidly developed site that displays HTML, CSS and the output from both in a large screen area to one side. You can set certain options and either share or download your finished code when you're done. **www.cssdesk.com/**.



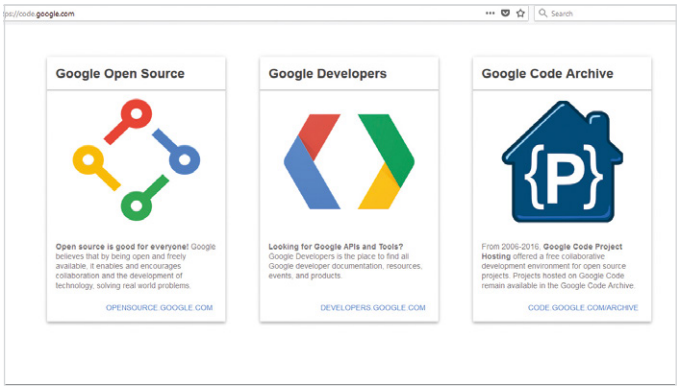
CLOUD9

Using the Amazon Web Services (AWS) cloud infrastructure, Cloud9 is an excellent IDE designed for collaborations, testing, debugging and editing. It includes essential tools for the major programming languages including Python, PHP, JavaScript and more. Being cloud based, you can quickly share your work with others and create a development environment. **www.aws.amazon.com/cloud9/?origin=c9io**.



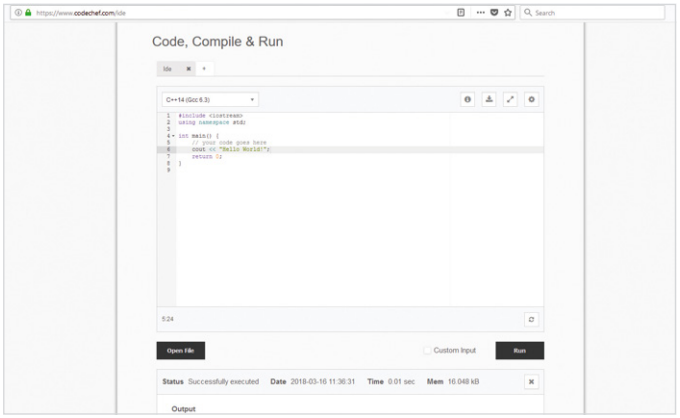
GOOGLE CODE

Google's commitment to creating and maintaining good code is certainly commendable, regardless of what you may think of the company as a whole. Its embraced the open source community and created Google Code, where you can test code, see great examples, get hold of APIs and tools and much more. **www.code.google.com/**.



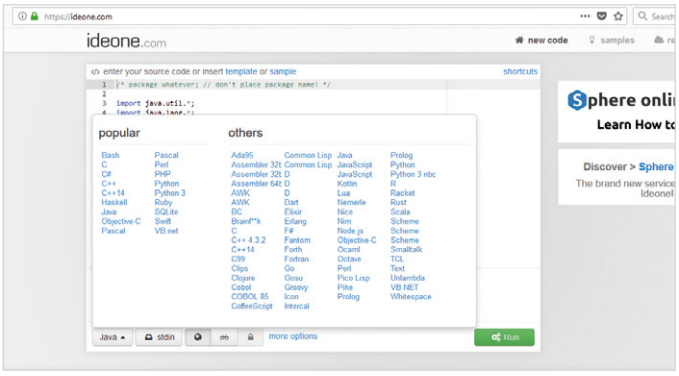
CODECHEF

This online platform helps developers work through and test their code in a huge range of supported languages. There's a dedicated section to help hone your coding craft and even a monthly contest and Code Cook-offs with cash prizes available. **www.codechef.com/ide**.



IDEONE.COM

This online code editor and compiler supports over 60 different programming languages for you to test and debug. There's Linux Bash, C++, Python and Python 3, Java, JavaScript and many more to try out. There are samples available and you can view recent code entered. **www.ideone.com/**.



JSBEEB

Purely for fun, those of you old enough to recall the heady days of the '80s when the C64, ZX Spectrum and other marvellous 8-bit computers ruled will certainly warm to jsbeeb. In the '80s, British classrooms were filled with BBC Microcomputers. Jsbeeb fills the nostalgic gap with a BBC Micro emulator where you can code, save, share and more. **www.bbc.godbolt.org/**.

Python OS Module Error Codes

The Python OS module enables the coder to interact with the operating system. You can read files, write to them, manipulate paths and even run built-in operating system specific commands. It's one of the most used modules, and one of the most problematic.

OS.ERROR

The interaction between Python and the operating system brings about its own set of problems and error codes. Filename too long, path not valid and so on are represented as errno. values. Here's a list of common ones.

errno.EPERM	Operation not permitted
errno.ENOENT	No such file or directory
errno.ESRCH	No such process
errno.EINTR	Interrupted system call
errno.EIO	I/O error
errno.ENXIO	No such device or address
errno.ENOMEM	Out of memory
errno.EACCES	Permission denied
errno.EEXIST	File exists
errno.EISDIR	Is a directory
errno.EINVAL	Invalid argument
errno.EMFILE	Too many open files
errno.ETXTBSY	Text file busy
errno.EFBIG	File too large
errno.ENOSPC	No space left on device
errno.EROFS	Read-only file system
errno.ENAMETOOLONG	File name too long
errno.ENOTEMPTY	Directory not empty
errno.ENONET	Machine is not on the network
errno.ENOPKG	Package not installed
errno.ECOMM	Communication error on send
errno.EPROTO	Protocol error
errno.ELIBBAD	Accessing a corrupted shared library
errno.EUSERS	Too many users
errno.EDESTADDRREQ	Destination address required
errno.EMSGSIZE	Message too long
errno.ENOPROTOOPT	Protocol not available
errno.EPROTONOSUPPORT	Protocol not supported
errno.EADDRINUSE	Address already in use
errno.ENETDOWN	Network is down
errno.ENETUNREACH	Network is unreachable
errno.ECONNRESET	Connection reset by peer
errno.ENOBUFS	No buffer space available
errno.ETIMEDOUT	Connection timed out
errno.ECONNREFUSED	Connection refused
errno.EHOSTDOWN	Host is down
errno.EHOSTUNREACH	No route to host
errno.EALREADY	Operation already in progress
errno.EINPROGRESS	Operation now in progress
errno.ESTALE	Stale NFS file handle
errno.EREMOTEIO	Remote I/O error


button.data-api',
(e.target)
asClass('btn')) $btn = $btn.cl
($btn, 'toggle')
arget).is('input[type="radio"]
ckbox"]'))) e.preventDefault()
button.data-api blur.bs.butto
ton"]', function (e) {
closest('.btn').toggleClass(

Python Errors

It goes without saying that you'll eventually come across an error in your code, where Python declares it's not able to continue due to something being missed out, wrong or simply unknown. Being able to identify these errors makes for a good programmer.

DEBUGGING

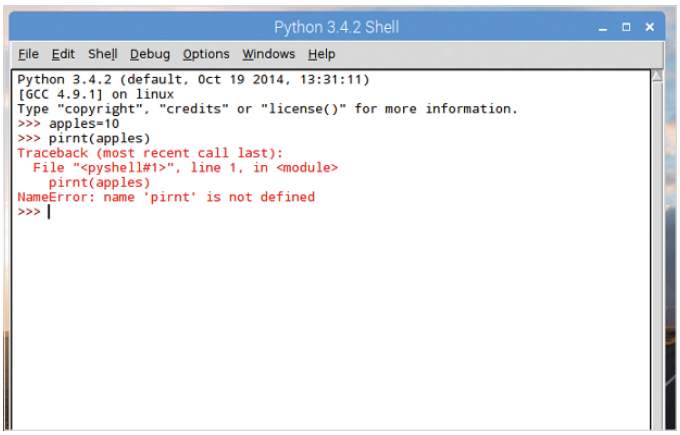
Errors in code are called bugs and are perfectly normal. They can often be easily rectified with a little patience. The important thing is to keep looking, experimenting and testing. Eventually your code will be bug free.

STEP 1

Code isn't as fluid as the written word, no matter how good the programming language is. Python is certainly easier than most languages but even it is prone to some annoying bugs. The most common are typos by the user and whilst easy to find in simple dozen-line code, imagine having to debug multi-thousand line code.

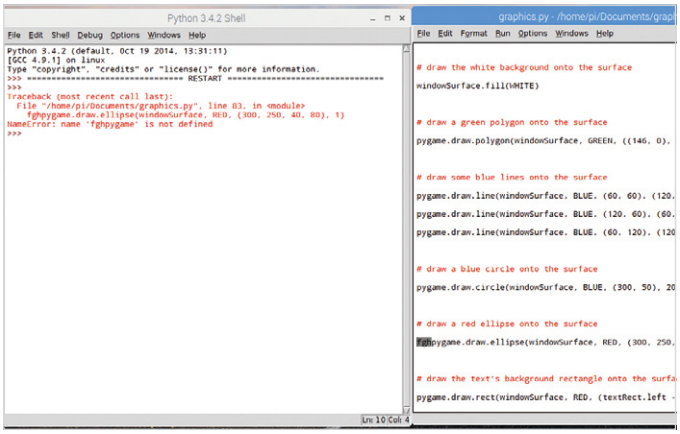
STEP 2

The most common of errors is the typo, as we've mentioned. The typos are often at the command level: mistyping the print command for example. However, they also occur when you have numerous variables, all of which have lengthy names. The best advice is to simply go through the code and check your spelling.



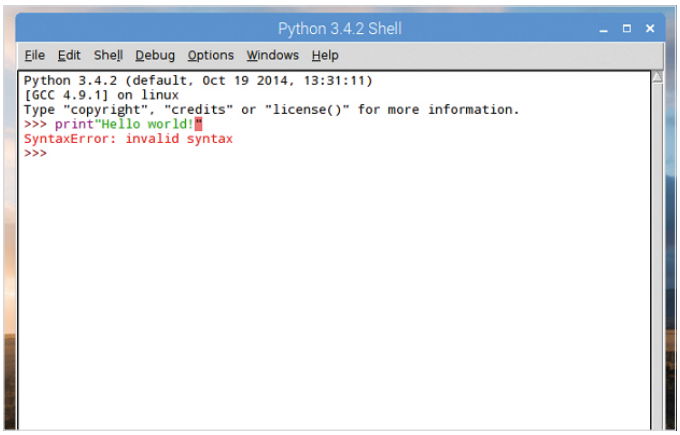
STEP 3

Thankfully Python is helpful when it comes to displaying error messages. When you receive an error, in red text from the IDLE Shell, it will define the error itself along with the line number where the error has occurred. Whilst in the IDLE Editor this is a little daunting for lots of code; text editors help by including line numbering.



STEP 4

Syntax errors are probably the second most common errors you'll come across as a programmer. Even if the spelling is correct, the actual command itself is wrong. In Python 3 this often occurs when Python 2 syntaxes are applied. The most annoying of these is the print function. In Python 3 we use **print("words")**, whereas Python2 uses **print "words"**.



STEP 5

Pesky brackets are also a nuisance in programming errors, especially when you have something like:

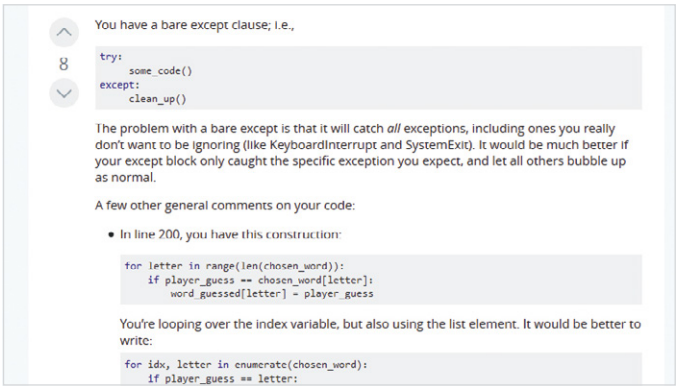
```
print(balanced_check(input()))
```

Remember that for every '(' there must be an equal number of ')'.

```
import sys

def balanced_check(data):
    stack = []
    characters = list(data)

    for character in characters:
        reference = {
            '(': ')',
            '{': '}',
            '[': ']'
        }
        if character in reference.keys():
            stack.append(character)

        elif character in reference.values() and len(stack) > 0:
            char = stack.pop()
            if reference.get(char) != character:
                return "NO"
        else:
            return "NO"
```

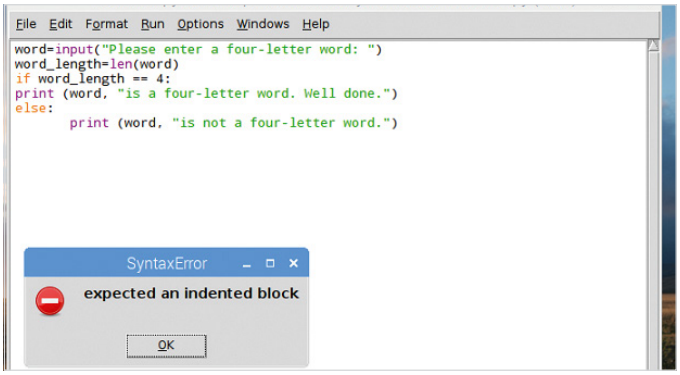
STEP 6

There are thousands of online Python resources, code snippets and lengthy discussions across forums on how best to achieve something. Whilst 99 per cent of it is good code, don't always be lured into copying and pasting random code into your editor. More often than not, it won't work and the worst part is that you haven't learnt anything.



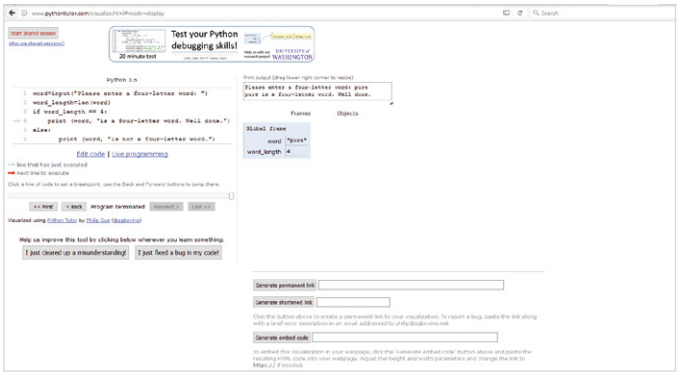
STEP 7

Indents are a nasty part of Python programming that a lot of beginners fall foul of. Recall the IF loop from the Conditions and Loops section, where the colon means everything indented following the statement is to be executed as long as it's true? Missing the indent, or having too much of indent, will come back with an error.



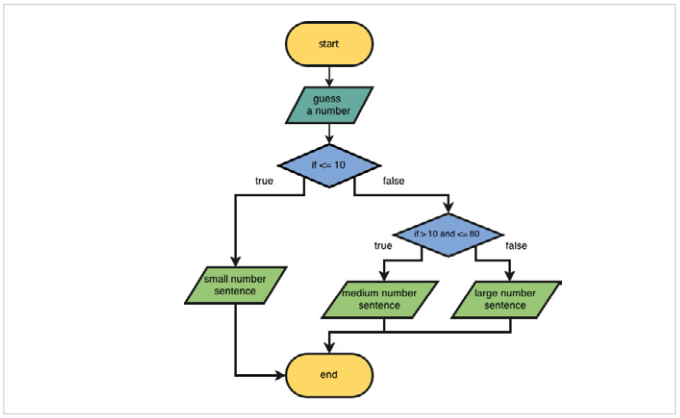
STEP 8

An excellent way to check your code step-by-step is to use Python Tutor's Visualise web page, found at **www.pythontutor.com/visualize.html#mode=edit**. Simply paste your code into the editor and click the Visualise Execution button to run the code line-by-line. This helps to clear bugs and any misunderstandings.



STEP 9

Planning makes for good code. Whilst a little old school, it's a good habit to plan what your code will do before sitting down to type it out. List the variables that will be used and the modules too; then write out a script for any user interaction or outputs.



STEP 10

Purely out of interest, the word debugging in computing terms comes from Admiral Grace Hopper, who back in the '40s was working on a monolithic Harvard Mark II electromechanical computer. According to legend Hopper found a moth stuck in a relay, thus stopping the system from working. Removal of the moth was hence called debugging.

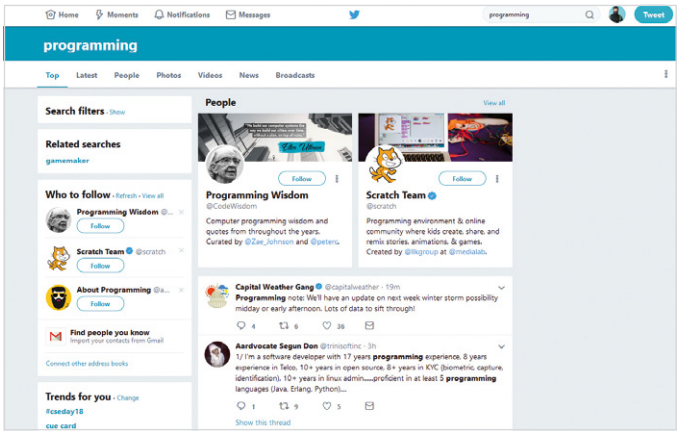
Where Next?

Coding, like most subjects, is a continual learning experience. You may not class yourself as a beginner any more but you still need to test your code, learn new tricks and hacks to make it more efficient and even branch out and learn another programming language.

#INCLUDE<KEEP ON LEARNING>

What can you do to further your skills, learn new coding practises, experiment and present your code and even begin to help others using what you've experienced so far?

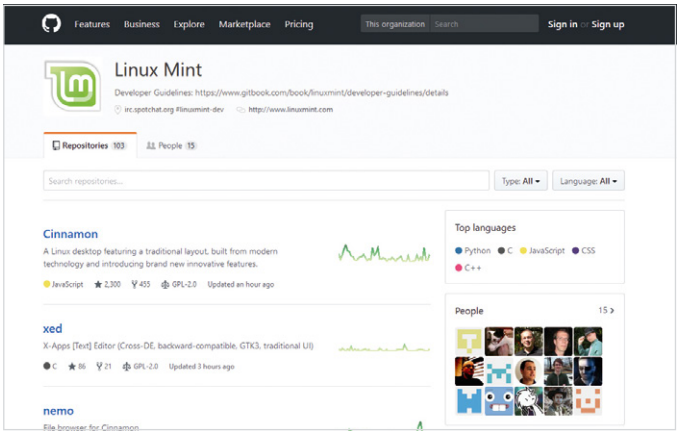
STEP 1 Twitter isn't all trolls and antagonists, among the well publicised vitriol are some genuine people who are more than willing to spread their coding knowledge. We recommend you find a few who you can relate to and follow them. Often they post great tips, hacks and fixes for common coding problems.



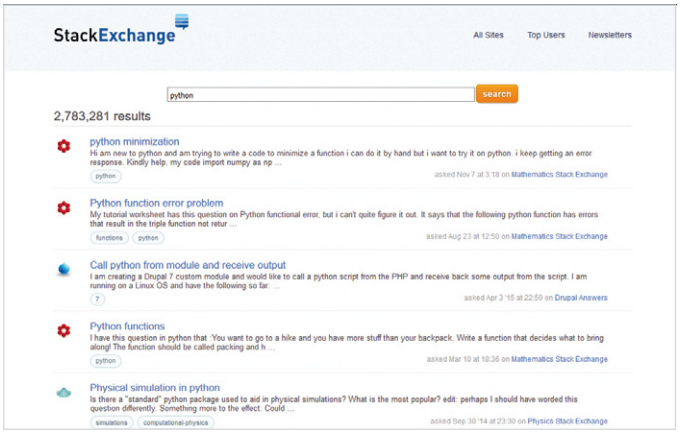
STEP 2 If you've mastered Python fairly well, then turn your attention to C++ or even C#. Still keep your Python skills going but learning a new coding language keeps the old brain ticking over nicely and give you a view into another community, and how they do things differently.

```
#pragma once

#define EXPAND_ME __FILE__ " : " __LINE__

constexpr auto pi = 314LLu;
thread_local decltype(pi) rage = 0b10;

[[deprecated("abc")]] char16_t *f() noexcept { return nullptr; }

template <typename T> struct Foo {
  static_assert(std::is_floating_point<T>::value,
                "Foo<T>: T must be floating point");
};

struct A final : Foo {
  A() = default;
  [[noreturn]] virtual void foo() override;
};

template <typename T> concept bool EqualityComparable = requires(T a,
  { a == b } -> bool;
};
```

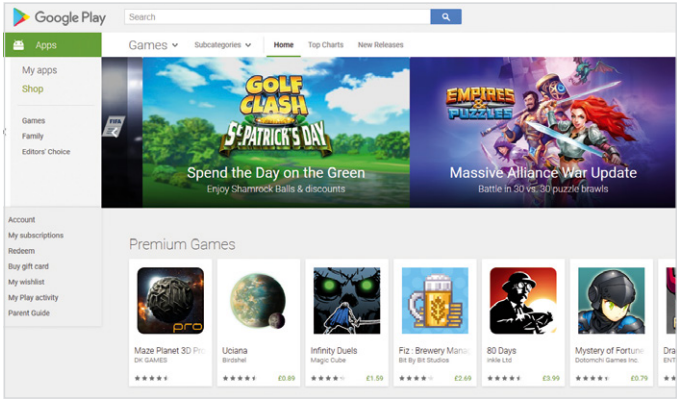
STEP 3 Look for open source projects that you like the sound of and offer to contribute to the code to keep it alive and up to date. There are millions of projects to choose from, so contact a few and see where they need help. It may only be a minor code update but it's a noble occupation for coders to get into.



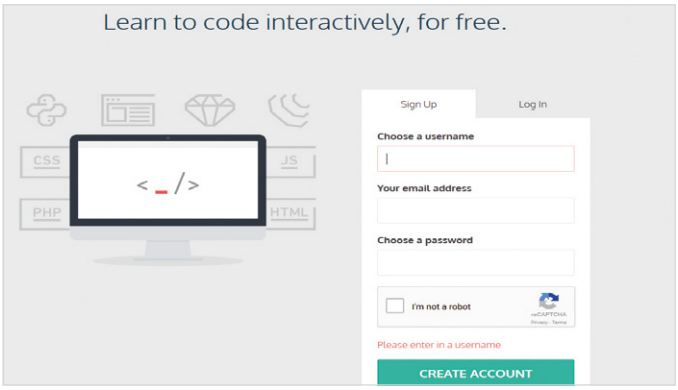
STEP 4 Become more active on coding and development knowledge sites, such as StackExchange. If you have the skills to start and help others out, not only will you feel really good for doing so but you can also learn a lot yourself by interacting with other members.



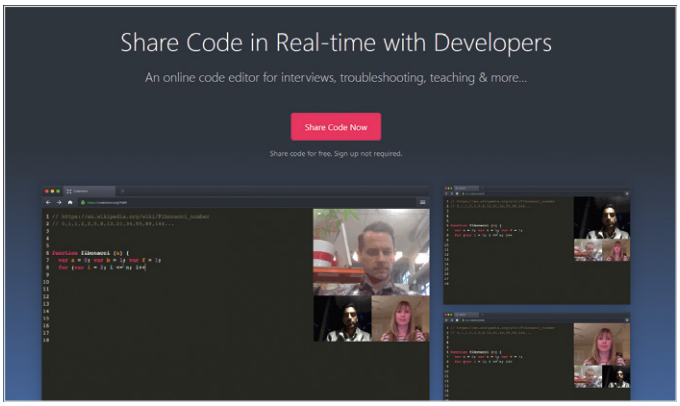
STEP 5 The mobile market is a great place to test your coding skills and present any games or apps you've created. If your app is good, then who knows, it could be the next great thing to appear on the app stores. It's a good learning experience nevertheless, and something worth considering.



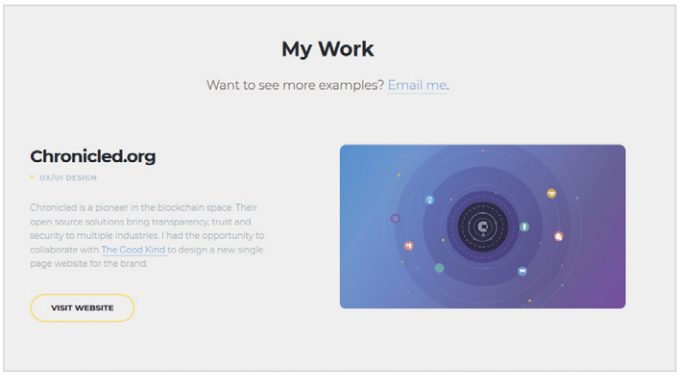
STEP 6 Online courses are good examples of where to take your coding skills next, even if you start from the beginner level again. Often, an online course follows a strict coding convention, so if you're self-taught then it might be worth seeing how other developers lay out their code, and what's considered acceptable.



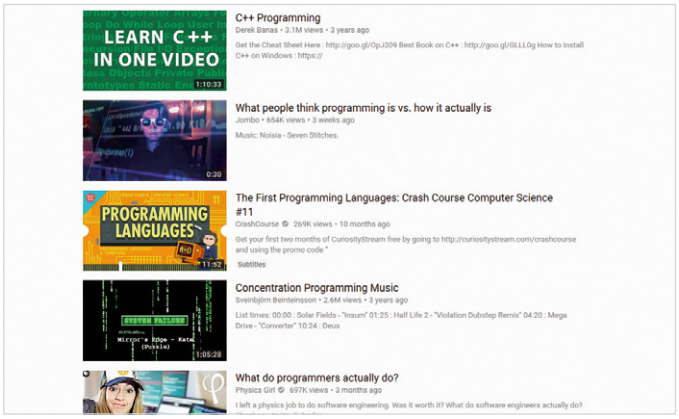
STEP 7 Get sharing, even if you think your code isn't very good. The criticism, advice and comments you receive back help you iron out any issues with your code, and you add them all to your checklist. Alternatively your code might be utterly amazing but you won't know unless you share it.



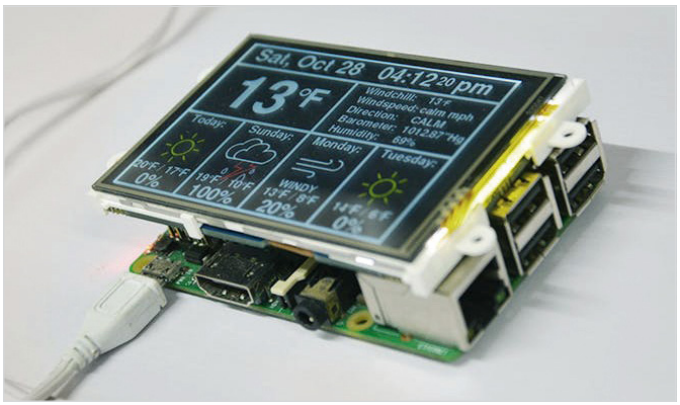
STEP 8 If you've learned how to code with an eye for a developer job in the future, then it's worth starting to build up an online portfolio of code. Look at job postings and see what skills they require, then learn and code something with those skills and add it to the portfolio. When it comes to applying, include a link to the portfolio.



STEP 9 Can you teach? If your coding skills are spot on, consider approaching a college or university to see if they have need for a programming language teacher, perhaps a part-time or evening course. If not teaching, then consider creating your own YouTube how to code channel.



STEP 10 Contributing to hardware projects is a great resource for proving your code with others and learning from other contributors. Many of the developer boards have postings for coders to apply to for hardware projects, using unique code to get the most from the hardware that's being designed.



Glossary of Terms

Trying to include definitions for every programming language would require many more pages than we have here. However, we have created a list of some of the most common terms you will encounter as you get started on your coding journey. As you gain experience and try new things, your coding vocabulary will naturally expand.

A

ALGORITHM
A process or set of rules to be followed in calculations or other problem solving operations, especially by a computer.

ANGULAR.JS
Angular.js is an open source web application framework maintained by Google.

APACHE
Apache is an open source Unix-based Web server. It was created by the Apache Software Foundation.

AJAX
AJAX stands for: asynchronous JavaScript and XML. It is a set of web development techniques utilising many web technologies on the client side in order to create asynchronous web applications.

API
An API is an application programming interface. It is a set of routines, protocols and tools for building software applications. APIs express software components in terms of their operations, inputs, outputs and underlying types.

B

BACKBONE.JS
Backbone.js is a JavaScript framework with a RESTful JSON interface and is based on the model-view-presenter (MVP) application design paradigm.

BOOLEAN SEARCHING
Boolean searches allow you to combine words and phrases using the words and, or, not (Boolean operators) to limit, broaden or define your search.

BUG
A mistake in the program. A point of error that causes the program to stop, or behave differently than expected.

BRACKETS
Characters often used to surround text. The different types of brackets are: Parenthesis, Curly Brackets, Angle Brackets and Square Brackets.

C

CALL
To run the code in a function; also referred to as "running", "executing" or "invoking" a function.

CLASS
In Python, a template for creating user-defined objects. Class definitions normally contain method definitions which operate on instances of the class.

CODING
Coding is the act of computer programming in a given coding language.

COFFEESCRIPT
CoffeeScript is a programming language that trans compiles to JavaScript.

COMPILER
This is a program that takes the code you have written and translates it into the binary ones and zeros of actual machine code.

CONCATENATION
Combining two things together, such as two lists or strings of text.

CONSTANT
A variable that never changes its value. Example: the PI constant has the value 3.14.

D

DATA STRUCTURES
A data structure is a method of organisation of data in a computer so that it can be used efficiently.

DEPLOYMENT
Software deployment is all of the activities that make a software system available for use.

DJANGO
A free open source web application framework written in Python that follows the model-view-controller (MVC) framework.

DUMP
A list of data that is saved if a program crashes, often as a text file. It is very useful for diagnosing problems.

E

EXECUTABLE
A program, usually a single file, ready to be run.

EXPRESSION
A piece of syntax which can be evaluated to some value; An accumulation of expression elements like literals, names, attribute access, operators or function calls.

EXPRESS.JS
Express.js is a Node.js web application server framework, designed for building single-page, multi-page and hybrid web applications.

F

FLASK
A micro web application framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine.

FRAMEWORK
A framework is often a layered structure indicating what kind of programs can or should be built and how they would interrelate.

FULL STACK
A full stack, also known as a software stack or bundle, is a set of software components needed to create a complete web application.

FUNCTION
A set of instructions that are written once to obtain a particular result and can then be used whenever necessary by 'calling' it.

G

GIT/GITHUB
A micro web application framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine.

GUI
General User Interface, refers to the 'front end' of a piece of software that the end user actually sees and interacts with.

H

HAML
HTML Abstraction Markup Language is a lightweight markup language that's used to describe the HTML of a web document.

HASHABLE
An object is hashable if it has a hash value which never changes during its lifetime (it needs a __hash__() method), and can be compared to other objects.

HTML
HyperText Markup Language, commonly referred to as HTML, is the standard markup language used to create web pages. This is often the very first technology that beginners to web development will learn.

HTTP REQUEST
HyperText Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

I

INTEGRATED DEVELOPER ENVIRONMENT (IDE)
An Integrated Development Environment is a basic editor and code interpreter which allows you to work with a specific coding language. The Python IDE is known as IDLE.

INTERPRETED
Python is an interpreted language, as opposed to a compiled one, though the distinction can be blurry because of the presence of the bytecode compiler.

INTERPRETER
Some languages do not need a compiler but instead use an 'interpreter' that translates to machine code as the program is run.

IOS SWIFT
iOS Swift is a multi-paradigm compiled programming language created by Apple Inc. for iOS, macOS and watchOS and tvOS development.

ITERATION
A sequence of instructions that are repeated. For example, to perform an action for every item in a list you would 'iterate' over that list. Each time it is repeated is one iteration.

J

JQUERY
jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery is the most popular JavaScript library in use today.

JSON
A format for transmitting information between locations that is based on JavaScript. Many APIs use JSON.

L

LAMP STACK
LAMP is an archetypal model of web service solution stacks: Linux operating system, the Apache HTTP Server, MySQL relational database management system and the PHP programming language.

LBYL
Look before you leap. This coding style explicitly tests for pre-conditions before making calls or lookups.

LINUX
Linux is a Unix-like computer operating system assembled under the model of free and open source software development and distribution.

LOGICAL OPERATION
The use of simple Boolean logical such as and, or and not.

LOOP
A piece of code that keeps running until a certain condition is fulfilled; or isn't fulfilled in the case of an 'infinite loop' that will crash the system running it.

M

MONGODB
MongoDB is a cross-platform document oriented database. Classified as a NoSQL database.

MVC
Model-view-controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts.

MYSQL
MySQL is an open source relational database management system (RDBMS).

N

NESTED
When one thing is contained within another it is said to be 'nested'.

NODE.JS
Node.js is an open source, cross-platform runtime environment for developing server-side web applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on multiple systems.

O

OBJECT ORIENTED PROGRAMING (OOP)
OOP is a programming paradigm based on the concepts of 'objects' that are data structures containing data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.

OBJECT RELATIONAL MAPPER (ORM)
ORM is a programming technique for converting data between incompatible type systems in object-oriented programming languages.

P

PHP
PHP is a server-side scripting language designed for web development but also used as a general purpose programming language.

PYTHON
Python is a widely used general purpose, high level programming language. Its design philosophy emphasises code readability and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java.

R

RECURSION
When something refers to itself. For example a variable may add something to itself for each iteration of a code loop.

RUN TIME
The time during which a program is actively running.

S

SANDBOX
A place to run a program for testing and experimenting.

SASS
Sass is a scripting language that is interpreted into Cascading Style Sheets (CSS). SassScript is the scripting language itself and consists of two syntaxes.

SLICE
An object usually containing a portion of a sequence. A slice is created using the subscript notation, [] with colons between numbers when several are given.

SOFTWARE DEVELOPMENT KITS
A 'software development kit' or SDK is a bundle of software tools for the creation of new applications for a specific platform or framework.

SUBROUTINE
A function or other portion of code that can be run anywhere within a program.

SYNTAX
Programming languages are just like human languages, they have their own 'syntax' or rules to describe how statements should be written.

T

TYPE
The type of a Python object determines what kind of object it is; every object has a type.

V

VALUE
A piece of data that can be contained inside a variable. Every value has a type.

VARIABLE
A way, used by many programming languages, to store a piece of data that can then be modified at any time.

W

WRITE
To send output data values to an external destination, usually to a file. Can also refer to sending data over a network.

The Ultimate Jargon Buster

Avoid tech confusion, either when reading this book or when talking to friends, with this glossary of technology terms and phrases. We have looked across the tech boarders to bring to you the definitive jargon buster, but it should help you to understand the common terms people use when talking about their devices and their software.

0-9

3G
The third generation of mobile data networking used by both the iPhone and iPad. This connection is slower than Wi-Fi, but is more readily available and is used to transfer data from your device when you are on the go. It uses the mobile phone network.

4G
The fourth generation of mobile data networking.

5G
The fifth generation of mobile data networking offers increased speed when transferring data on the go but it is still in its early stages of adoption by mobile phone networks.

A...

Accessibility
A series of tools and features designed to make an Apple device such as the Mac and mobile devices easier to use by those with disabilities such as vision or hearing impairments. You can find the Mac's Accessibility features and customise them in System Preferences.

ADB
Android Debug Bridge. Part of the Android Software Development Kit, used to send commands from a computer to an attached phone.

Adobe Bridge
Bridge is a browser application produced by Adobe Systems as part of the Creative Suite and is usually installed alongside Photoshop. Its main function is as the file management hub of the Creative Suite. It can be used to open, manage, rate and rename files as well as edit their metadata.

Adobe RGB
A device independent colour space developed by Adobe. It provides a relatively large range of colours, i.e. grey-balanced and perceptually uniform. It is widely used for image editing.

ADSL
Asymmetric Digital Subscriber Line. It's a means of connecting to the Internet through your telephone line. Sometimes just called 'DSL'

Airplane Mode
All airlines warn you to turn off mobile electronic devices when on board an aircraft, so this iPad setting turns off all incoming and outgoing signals to your device, including data, Bluetooth and Wi-Fi.

AirPlay
A protocol for streaming sounds and video from an Apple device to a set of compatible speakers or a device such as an Apple TV. It's wireless and easy to use.

AMOLED
Active Matrix Organic Light Emitting Diode. A bright and colourful display technology popular on smartphones (although it has now been superseded by Super AMOLED and qHD.)

Android
The name of the operating on your smartphone (we are assuming you own an Android phone if you are reading this magazine). There have so far been eleven versions/updates released.

Android Market
The previous name for the Google Play Store. The place to go to find apps, books and movies to install on your phone.

Anti-Aliasing Filter
This is an optical filter, also known as low-pass filter, which is placed on the camera sensor to create a slight blur or softening that helps counteract aliasing or Moiré interference.

Apk (.apk)
The file extension of Android applications.

Apps (Applications)
The programs, such as Angry Birds, Facebook or Soundhound, that you install and run on you Android phone.

App Store
The App Store is where you can download free and paid programs to your device using your Apple ID. You can access it through the application found on your home screen.

App Inventor
A web-based system that lets anyone develop apps for Android. Originally created and run by Google, but now run as an open-source project.

Apple ID
This is the email address and password that you have registered with Apple. It will be required to access most online applications on your iPad, including iTunes, App Store and Books.

Apple Menu
The menu that's opened by clicking on the Apple icon in the left of the menu bar, when using a Mac or MacBook computer. It gives access to system functions such as Preferences, App Store, Force Quit and more.

Archos
A manufacturer of Android tablets.

ASUS
A well-known manufacturer of Android smartphones and tablets.

AT&T
One of the "Big Four" of American carriers

B...

Bit
A contraction of binary digit, the smallest unit of information storage or digital information that can take on one of two values, 0 and 1.

Bit Depth
Defines how many bits of colour data are used to describe each pixel or channel. For example, 2 bits per pixel only allows for black or white. 8 bits provides 256 colours. When referring to an 8-bit colour image, 256 is multiplied by the three primary channels (red, green and blue) to create what is commonly called 24-bit colour, with a possible 16,777,266 colours.

Black Point
In image editing, the black point is a tonal adjustment that sets the point at which the deepest shadow detail in the histogram is clipped to black.

Bloatware
The name given to unwanted applications preloaded onto your phone. Bloatware cannot usually be removed by the end user unless they decide to root their handset.

BlueTooth
Short range file data system built into almost every Android smartphone ever made. Can be used to send files and connect speakers or headphones wirelessly to your phone.

Books
Apple's eBook reader, available from the App Store. It handles the standard electronic publishing formats protected by FairPlay DRM and PDF. It was introduced in 2010 along with the iPad.

Bootloader
A normally hidden mode in Android that helps with flashing ROMs when rooting an Android phone.

Broadband
Wide bandwidth data transmission, that is, fast Internet as opposed to the older, dial-up services.

Browser
An app used to access websites found on the worldwide web. The iPad and iPhone come with Apple's Safari browser preinstalled but others are available in the App Store. Android devices use the Chrome browser

BSI
Backside Illumination. Sometimes used to improve smartphone camera performance.

C...

Calendar
This is one of several preloaded apps found on most devices. Use it to keep track of events, invitations and reminders on your phone and tablet.

Camera Raw
Proprietary raw file formats designed to hold image data and metadata generated by digital cameras. These formats are non-standard and undocumented, although they are usually based on the TIFF/EP file format standard.

Carrier
Another name for a mobile network provider (Vodafone, AT&A, Sprint, etc.)

Casting
The process of converting one data-type into another. For example, sometimes a number may stored as text but need to be converted in to an integer.

CCD (Charged Coupled Device)
A type of image sensor found in digital cameras and scanners. It is a light-sensitive chip that converts light into an electrical charge that is then processed by an analogue to digital converter. CCD differs from the other common sensor type (CMOS) in the way that it processes the electrical charges captured by sensor elements.

CDMA
One of the two main cell phone communication standards. Not often used in phones outside of the U.S.

Chromatic Aberration
Known also as colour fringing, chromatic aberration is caused when a camera lens does not focus the different wavelengths of light onto the exact same focal plane. The effect is visible as a thin coloured halo around objects in the scene, often the border between dark and light objects.

Class
A class provides a means of bundling data and functionality together. They are used to encapsulate variables and functions into a single entity.

Clipping
The loss or either highlight or shadow details when tone information is forced to pure white or black. For example, over-exposure can produce clipping by forcing highlights that should contain detail to register as pure white. Clipping can also be caused either intentionally as a creative effect or unintentionally because of excessive corrections. Saturation clipping can occur when colours are pushed beyond the range of a colour space.

Comments
A comment is a section of real world wording inserted by the programmer to help document what's going on in the code. They can be single line or multi-line and are defined by a # or '".

Constant
A number that does not change. It is good practice to name constants in capitals e.g. SPEED_OF_LIGHT

CMOS (Complementary Metal Oxide Semiconductor)
A type of image sensor found in digital cameras and scanners. It is a light-sensitive chip that converts light into an electrical charge, which is then processed by an analogue to digital converter. CMOS differs from the other common sensor type (CCD) in the way that it processes the electrical charges captured by sensor elements.

CMYK
Also commonly referred to as process colour, CMYK is a subtractive colour model using cyan, magenta, yellow and black inks in colour printing.

Colour Profile
Also known as an ICC profile, the Colour Profile defines the information required to by a colour management system (CMS), to make the colour transformations between colour spaces. They can be device specific such as monitors, scanners or printers or abstract editing spaces.

Compression
The process of re-encoding digital information using fewer bits than the original file or source. This reduces transmission time and storage requirements. There are a number of different algorithms that provide either "lossy" or lossless compression. JEPG is a common file format that employs lossy compression to achieve smaller file sizes at the expense of image quality.

Cupcake
The nickname for Android version 1.5.

CyanogenMod
One of the best known and most often used series of custom ROMs.

D...

DECT
Digital Enhanced Cordless Telecommunications. It's a wireless standard used mostly for cable-free telephone systems.

DLNA
Dynamic Living Network Alliance. A technology found on some high-end Android phones that lets users stream photos and videos from their phone to a compatible TV.

DNG (Digital Negative)
An open standard file format developed by Adobe Systems that provides an alternative to proprietary camera raw files. The DNG specification incorporates rich metadata along with embedded previews, camera profiles and editable notes. DNG uses lossless compression that can result in a significant file size reduction over the original proprietary raw.

Download
The term used when taking a file from the Internet or from a connected device such as a computer, to your phone or tablet.

Dock
The opaque strip at the bottom of the home screen. Apps in the dock remain in a special row of icons (or Folders post iOS 4) along the bottom of iPhone, iPod touch and iPad screens and do not change when you swipe between home screens.

DPI (Dots Per Inch)
The measurement of print resolution expressed in how many dots of ink are laid down either horizontally or vertically per inch. A higher number indicates a greater amount of output resolution. Not to be confused with pixel per inch (PPI). There is not necessarily a direct correlation between DPI and PPI.

Dream (HTC Dream or G1)
The very first phone to use the Android operating system.

Dynamic Range
In the context of photography, dynamic range describes the difference between the brightest and darkest light intensities of a scene. From capture to output, there can be a large difference in the size of the dynamic range that each device is capable capturing or reproducing. Dynamic range is commonly expressed in the number of f-stops that can be captured or the contrast ratio of the scene or device.

E...

Eclair
The nickname for Android version 2.0/2.1.

Emoticon
A small drawing used to augment a message or text. Typically these are yellow faces showing a variety of expressions.

Escape Sequence
When characters that have certain meanings in the Python coding language are required in strings they have to be "escaped" so that the computer knows they do not have their usual meaning. This is done by putting a slash in front of them e.g. \"

Ethernet
The format used for local cabled networks (LAN). Your router comes supplied with Ethernet cables and has ports for plugging them in.

Exposure
The total amount of light that strikes the sensor or film during an image capture. An optimal exposure takes full advantage of the dynamic range of the sensor without under-exposing the shadows or over-exposing the highlights.

Extender
A device that extends the range of a wireless network by creating a second entry point, which may, or may not, merge with the main one.

F...

Facebook
Currently the most popular social networking site on the Internet; there are currently over 835 million registered users.

FaceTime
Apple's video calling service. Requires a Wi-Fi connection and is currently only supported via a phone number on iPhone and Apple ID email address on iPod touch 4 and Mac.

Factory Reset
An option on your Android phone that allows you to return it to the state it was when it left the factory.

File Format
The structure of how information is encoded in a computer file. File formats are designed to store specific types of information, such as JPEG and TIFF for image or raster data, AI for vector data or PDF for document exchange.

Folder
An icon representing a container for a group of apps, files or icons.

Force Quit
In the Fast App Switcher, tapping and holding an app will put it in 'jiggly mode' and tapping the x badge will force it to quit. Built-in apps like Mail and Messages will automatically restart while third-party apps will restart the next time you launch them.

Froyo
The nickname given to Android version 2.2.

G...

G1
The very first phone to run the Android operating system. Also known and the HTC Dream.

Game Center
Apple's gaming service, where you can discover new games and share your game experiences with friends from around the world.

Gamut
The range of colours and tonal values that can be produced by a capture or output device or represented by a colour space.

Galaxy
A range of hugely popular handsets from Samsung, the biggest smartphone manufacturer in the world.

Geotagging
The act of digitally attaching your location to photos taken on your phone.

Gingerbread
The nickname given to Android version 2.3.

Gmail
Google's web-based email software. Comes pre-installed on every Android smartphone.

Google
Owner (although not the original creator) of Android. Also own a fairly well known search engine...

Google Now
An enhanced Google search app which bases the information displayed on current location. Currently only found in Jelly Bean.

Google Play
Previously known as Android Market, this is where you go to download Android compatible apps, books, music and movies.

Gorilla Glass
Increasingly popular scratch-resistant glass used for smartphone displays.

GPS
Global Positioning System. A system that uses satellites to pinpoint your current location.

Grayscale
A monochromatic digital image file with pixel values that use shades of grey to represent tonal information. The term is often used to describe digital black and white photographs.

GSM
One of the two main cell phone communication standards. Used in most countries outside of the U.S.

H...

Hacking
Most often means rooting when talking about Android.

Hard Reset
Also called Factory Reset. Returns the phone to its post-factory state.

HDR (High Dynamic Range)
A process that combines multiple exposure variations of an image to achieve a dynamic range exceeding that of a single exposure. Algorithms are used to blend the exposures into a high-bit file format that can then be converted to either 8 or 16 bit for printing or web presentation.

Histogram
A graphical representation of the tone and colour distribution in a digital image. This is typically based on a particular colour or working space by plotting the number of pixels for each tone or colour value. It can be used to interpret photographic exposure and reveal shadow or highlight clipping.

Home Button
The physical hardware button on the front of early models of the iPhone, iPod touch, iPad and many Android devices, located just below the screen. It's used to wake the device, return to the Home Screen and several other functions.

Home Screen
The front end of your smartphone or tablet. The screen you see, containing app icons, widgets, etc., when you first unlock the device.

Honeycomb
The nickname given to Android version 3.0. The only version designed specifically for tablets, but now superseded by ICS.

HTC
A large Taiwanese smartphone manufacturer.

HTTP
Hypertext Transfer Protocol, the protocol used by the World Wide Web (Internet) that defines how messages are sent, received and read by browsers and other connected software layers.

HTTPS
Hypertext Transfer Protocol Secure, an encrypted and far more secure version of HTTP.

I...

Ice Cream Sandwich
The nickname given to Android version 4.0/4.1. The majority of new Android tablets now use this.

IMEI
International Mobile Equipment Identity. This is a unique identification number assigned to every phone.

Intel
Well known PC processor manufacturer. Has now started producing smartphone processors.

Internet
A global system of interconnected computers and networks which use the Internet Protocol Suite (TCP/IP) to link online devices.

Indentation
The coding language Python uses indentation to delimit blocks of code. The indents are four spaces apart, and are often created automatically after a colon is used in the code

iOS
Apple mobile operating system and the software that powers the iPhone, iPod touch, iPad and Apple TV.

IPS
In Plane Switching is a type of display used on some phones that increases the viewing angle of the screen.

I-Tunes
Mac and Windows music playing software, also used to activate and sync iPhone, iPod touch and iPad. It is also used to purchase and manage music, movies, TV shows, apps, books and other media.

ISO (International Organisation for Standardisation)
In photography, ISO refers to the standard for measurement of the sensitivity of film or digital sensors to light.

J...

Jelly Bean
The nickname given to Android 4.2, the latest version of the operating system.

JIT
The Just In Time compiler was introduced in Android 2.2. It helps to speed up apps on Android.

JPEG, JPG (Joint Photographic Experts Group)
A standard created by the Joint Photographic Experts Group for the compression of photographic images and the accompanying file format. It employs lossy compression that can reduce file size but at the expense of image quality and detail.

K...

Kernel
The basic Linux building block of Android.

Keyboard
Tablets and smartphones can feature either a physical or software keyboard.

Keyword
An element of metadata that is used to make a file more easily discoverable to searches. Keywords can be individual words or short phrases and can have a hierarchical structure.

L...

Landscape Mode
This describes a phone or tablet when you hold it horizontally; this is when it's wider than it is tall and the Home button is on the right or left of the screen.

Launcher
This is the part of the Android user interface on home screens that lets you launch apps and make phone calls.

LAN
Local Area Network. Devices that are connected to your router using Ethernet cables, are part of the LAN (see also WLAN).

LG
A large Korean electronics and smartphone manufacturer.

Linux
An open-source operating system that is used as the basis of Android.

Live Wallpapers
Animated wallpapers introduced in Android 2.1.

Loop
A piece of code that repeats itself until a certain condition is met. Loops can encase the entire code or just sections of it.

LTE
Long-Term Evolution. A name sometimes given to 4G data networks.

Luminance
The intensity of light as emitted or reflected by an object or surface. This is usually expressed in candelas per square meter (cd/m2). It is a measurement of the brightness of an object or light source.

M...

Magic
HTC phone also known as the MyTouch 3G. The first phone to use an Android operating system.

Mail
Built-in Apple app for handling POP3, IMAP, MobileMe and Exchange/ActiveSync email accounts.

Messages
One of Apple's built-in iPhone apps that handles SMS text messages and MMS multimedia messages. SMS messages are also more generally called "messages" on most devices.

MMS: (MultimediaMmessages)
MMS supports images, videos, sound, contact cards and location data. Sent and received via the Messages app.

Megapixel
A term used to describe digital camera resolution, 1 megapixel equals one million pixels or sensor elements. To calculate the megapixel value for a camera, multiply the horizontal by the vertical pixel counts of the recorded image.

Mesh
A means of combining two wireless access points into one, so they use the same settings and appear as a single network to devices that join it.

Metadata
Embedded or associated information describing a file's contents, used in digital photography to hold exposure information, GPS location data, copyright information and more. There are several metadata formats such as EXIF, IIM, IPTC Core, Dublin Core, DICOM and XMP.

Modem
Short for modulate-demodulate, a modem converts data into a signal that can be transferred over a phone line, and does so in reverse for incoming data.

Motorola
A large manufacturer or electronics and smartphones.

N...

News
Is an app in iOS that collected together magazine and newspaper apps and allowed the automatic downloading of new stories.

Nexus
A range of smartphones and tablets developed by Google. The Nexus range runs a pure version of Android.

NFC
Near Field Communication. A technology which allows data to be between phones or between your phone and another device.

Noise
The unwanted colour or luminance variations of pixels that degrade the overall quality of an image. Noise can result from several different sources including a low signal to noise ratio, the use of high ISO settings, long exposures, stuck sensor pixels and compression artefacts. It can appear as random colour speckles, a grain-like effect or banding.

Notification Centre:
A pull-down list of recent notifications, accessible from any iOS Home Screen or from within any iOS app. Similar to the Notification Panel found on Android.

O...

OEM
Original Equipment Manufacturer. A company which manufacturers devices for another brand (e.g. ASUS is the OEM of Google's Nexus 7.)

One Series
A range of smartphones from HTC. Includes the One X, One V and the One S.

Open GL
An open source graphics library, used on some smartphones.

Open Source
Software which is available to be studied, used and adapted by anyone. Android is open source software.

Operating System
Also OS. The program that's loaded into the computer after the initial boot sequence has completed. The OS manages all the other programs, graphical user interface (GUI), input and output and physical hardware interactions with the user.

Optimus
A range of smartphones from LG

OTA
Over The Air. A method which upgrades are wirelessly sent to smartphones.

Output
Data that is sent from the program to a screen, printer or other external peripheral.

P...

Pantech
A South Korean smartphone manufacturer.

PDF (Portable Document Format)
Developed by Adobe Systems, PDF is an open standard file format for cross-platform document exchange. PDF is highly extensible, preserves the integrity of the original document, is searchable and provides document security.

Photos
Built-in Apple app that handles your photo albums on your iPhone and iPod touch 4, and synced photos and videos for iPhone and all generations of iPad and iPod touch.

Photo Stream
Part of iCloud, Photo Stream stores your last thirty days or 1000 photos online and on your iOS devices, and all your photos on your Mac.

Pixel
Derived from the term picture element, this is the smallest unit of information in a digital image. It is also commonly used to describe the individual elements on a capture device such as a camera sensor.

PIN
Stands for Personal Identification Number. Used to lock smartphones and SIM cards.

Plug-In
A software application or module that provides extended and specific functionality from within a larger host application.

Portrait Mode
This describes a smartphone or tablet when you hold it vertically; this is when it's taller than it is wide and the Home button is at the top or bottom of the screen.

PSD
The .psd (Photoshop Document) format is a popular proprietary file format from Adobe Systems, Inc. It has support for most of the imaging options available in Photoshop, such as layer masks, transparency, text and alpha channels. In addition, spot colours, clipping paths and even duotone settings can be saved if you are preparing images for printing.

Project Butter
Software enhancements introduced in Android 4.1. Designed to smooth out frame rates and animations.

Q...

QR Code
A type of barcode which can be scanned by smartphones to reveal information such as text and website URL's.

QuickTime
Apple's 2D video and graphics player, used to play movies and other video on iOS.

R...

Raw Files
A Raw file is the unprocessed data captured by a digital camera sensor. In most cases, cameras write Raw files using a proprietary file format. Raw files give the photographer the advantage of managing image processing during post-production rather than letting the camera make the processing decisions, as happens when shooting in JPEG format. See also: DNG.

Recovery Mode
A separate operating mode of Android. Mainly used for device administration and repair.

Retina Display
Super sharp display available on Mac computers and iOS devices.

Resolution

A measurement of the ability of an optical, capture, or output system to record and reproduce detail. It can be defined in several different metrics such as Line Pairs, PPI, DPI, SPI and LPI. Also see DPI and PPI.

RGB
A colour model that uses the three primary additive colours (red, green, blue) that can be mixed in different ratios to make all other colours.

ROM
Read Only Memory. In Android a ROM is used to load software updates. Custom ROMs are software updates developed by third parties.

Root
In Android, to Root means to unlock the device to allow more access to the core software (or root).

Router
A device that manages and organises your home network devices, whether they connect to the router using a cable (LAN), or wirelessly (WLAN).

S...

Safari
Apple's web browser, both for Mac OS X and iOS (sometimes called Mobile Safari). Based on KHTML/ WebKit renderer and the Nitro JavaScript engine.

Samsung
A huge Korean smartphone and electronics manufacturer.

SD Card
A small memory card which can often be inserted into smartphones to increase storage capacity.

Sense
The user interface designed by and used on HTC phones.

Sharpening
The process of increasing or emphasising contrast around the edges of details in an image, to give the impression that the image is sharper than it really is.

Sideload
The process of installing an app onto your phone outside of the Google Play store.

SIM Card
The small plastic chip required in all GSM phones to connect to the mobile network.

Siri
Apple's intelligent virtual assistant, that replaces Voice Control on the iPhone.

Sleep/Wake Button
Physical hardware button. Used to power on, wake from sleep, put to sleep and power down most smartphones and tablets.

Sony Ericsson
The company formed by Sony and Ericsson to manufacture and distribute mobile devices.

Sprint
One of the large US mobile carriers.

SSID
Service Set ID. In a nutshell, this is the 'name' of your wireless network, and can be changed using your router.

Super AMOLED
An improvement of AMOLED displays, providing brighter, less power hungry and less reflective screens.

T...

T-Mobile
Large US mobile carrier and manufacturer of smartphones.

Tegra 2
NVIDIA's dual-core mobile processor.

Tegra 3
NVIDIA's newer, quad-core, mobile processor.

Tethering
Using your smartphones data connection to provide internet access for another device (laptops, etc.)

Text Field
Any area where you can add text. For example, the search field is where you type something you're looking for. Tap on a text field to bring up the virtual keyboard.

Thumbnail Image
A small, low-resolution image preview used on the web to link to a high-resolution version of the file. Thumbnails can also be embedded in file formats such as TIFF and PSD.

TIFF or TIF (Tagged Image File Format)
An open standard file format specifically designed for images. TIFF can incorporate several types of compression, including LZW, JPEG and ZIP. The format is suitable for the storage of high quality archive images. The DNG format is based on the main TIFF standard.

TouchWiz
Samsung's custom user interface.

Twitter
One of the most popular social networks built around a follower and following system rather than friends.

U...

UPnP
Universal Plug and Play. A protocol used by digital media players for enjoying video, music, and pictures over your home network.

URL
Uniform Resource Locator. This is a web address, used to access a web page on the Internet, and usually starts 'www' and ends in '.com', or some other top-level domain.

USB
Universal Serial Bus. The connection method now used by most smartphones to connect to a computer or power source (MicroUSB).

V...

Vanilla
Sometimes used to describe Android without any custom user interface applied.

VDSL
Very High Speed Digital Subscriber Line. It's another protocol for getting on the Internet using your phone line, and is sometimes shortened to DSL.

Verizon
One of the four large US mobile carriers.

Virtual Environment
A cooperatively isolated runtime environment that allows Python users and applications to install and upgrade Python distribution packages without interfering with the behaviour of other Python applications running on the same system.

Virtual Machine
A computer defined entirely in software. Can be used to test/run/create code that won't affect the host system.

VPN
(Virtual Private Network):
This provides secure access over the Internet to private networks, such as the network at your company or school.

W...

While Loop
A coding loop that repeats code while a comparative statement returns the value True.

White Balance (WB)
In digital photography, white balance establishes the colour balance of the image in relationship to colour temperature of the lighting conditions. Most digital cameras have several built-in white balance presets (tungsten, daylight, cloudy, fluorescent, etc.) along with an auto setting and the ability to set a custom WB.

Widgets
The name given to the home-screen gadgets which allow you to see app updates, news, etc.

Wi-Fi
A group of backwards-compatible radio technologies used to connect peripherals to a network wirelessly.

WLAN
Wireless Local Area Network. Your network of wireless devices, as opposed to devices connected with a cable (see LAN).

World Phone
A device which works on both CDMA and GSM networks outside of its home country.

WPS
Wi-Fi Protected Setup, an easier way of connecting wireless devices to your router.

XYZ...

Xperia
A range of smartphones developed by Sony Ericsson. Includes the Xperia T and the Xperia Play, the PlayStation smartphone.

YouTube
Google-owned, web-based video streaming service. A YouTube app is usually pre-installed on Android devices.