Code Project: Fla

PART 3 Improve your programming skills and learn a foreign language with the help of **Mike Saunders** and your very own flash card testing tool...

Our expert

Mike Saunders hacks anything that carries electrons, and is proud of version 1.1 of MikeOS, his very own operating system written in x86 assembly language: **http://mikeos.sf.net**

One of the reasons we love computers is their ability to *just work*. Sure, hardware can break down and programs can have bugs, but when everything's running smoothly your data won't magically disappear. If you download, say, a Norwegian dictionary, your machine will store it for later retrieval – without argument. Your PC won't get bored and try to learn another language, nor will it have a few too many drinks and forget the data it had yesterday. It won't get into an argument with a lady PC and destroy its files out of jealousy. On a solid system, the data is always there.

Here in humanland, though, we're engaged in a constant battle with the foibles and quirks of our brains. We forget stuff, we change our minds – above all, we wish our brains were more like trusty RAM banks than fussy blobs of organic gloop. This is most evident when learning a foreign language: a computer can store millions of words and never 'forget' them, whereas we struggle to remember the German word for 'meeting' even though we used it yesterday. Thanks, brain.

So in this month's coding project, we're going to create a flash card program to help you remember foreign words. It displays an English word and asks you to choose its German equivalent from a list of three randomly chosen options, keeping a score as you progress. But it's not just limited to German – you'll be able to use it for any language, or indeed for anything else you want to learn! You could even set it up to display the name of an animal, having the program test you on its species.

New Python skills

For last month's project (config tools) we used Python, an easy-to-grasp and highly readable programming language. Don't worry if you haven't got the issue or have never written Python code before, though – it's very simple to understand. If you've dabbled in any programming language before, you'll have no problem working with the code.

Our flash card app will need to read text files and generate random numbers. Opening files in Python is a doddle: create a new text file called **foo.txt**, tap a few words into it (one per line), and save it in your home directory. Now create a file called **test.py**, also in your home directory, with the following contents:

```
file = open('foo.txt', 'r')
print file.readlines()
```

To run this Python script, start a terminal and enter:

```
python test.py
```

This code opens **foo.txt** (**'r'** for read-only) and associates the contents with a new object called **file**.

In the second line of code, we call **readlines()** on our **file** object, which scans through **foo.txt** and stores each line in an array. So by printing it to the screen, we see:

```
['hello\n', 'banana\n', 'cupcake\n']
```

Of course, this will vary depending on the words you entered in **foo.txt**. But it demonstrates how Python retrieves text from a file, storing each line as a separate array element. That's all well and good, but what if we want to display the text file normally?

```
file = open('foo.txt', 'r')
for line in file:
    print line
```

This prints out all of the lines in **foo.txt**. Note that indentation is essential in Python – the tab before **print line** shows that it's code to be executed in the **for** loop. That **for** loop essentially says: for every line in the file we opened, print said line until the file ends.

So, we can now retrieve text from files and use word lists for our flash cards. But there's something else we need to do: our program needs to display a list of possible answers when displaying a word. After all, it'd be rather pointless if answer number one was always correct! So in our program, we're going to display three possible answers for the displayed word, one of which will be correct. Here's how to get random numbers:

```
import random
a = random.randint(1, 5)
b = random.randint(30, 100)
print a, b
```

The first line tells Python that we want to use its random number facilities. After that we create two variables, a and b,

» **Last month** Part two: creating GUI-like alternatives for common admin utilities.

sh cards

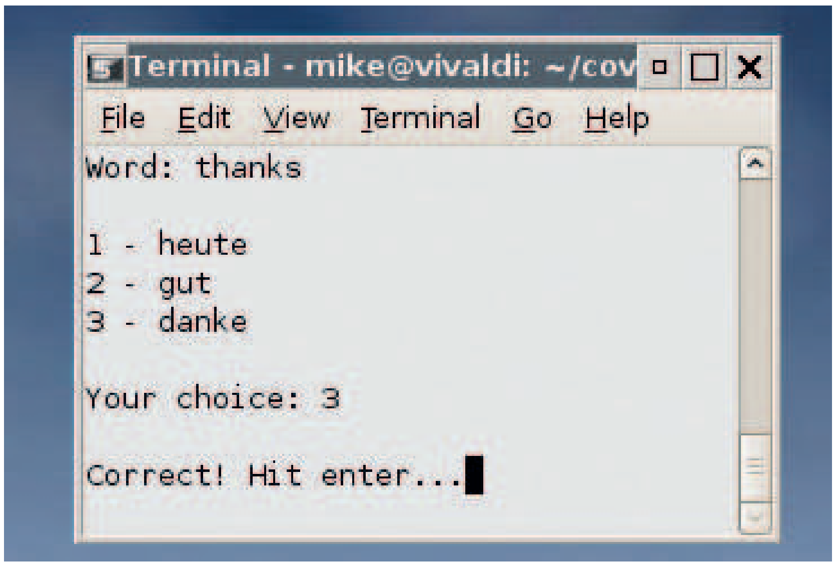
and give them random values via Python's **random.randint()** routine. We specify a range for the values – for variable a, the number will be between one and five (inclusive). For b, it will be between 30 and 100. Easy!

Bring on flashcard.py 1.0

Let's get cracking with the program. We'll need two text files, one containing English words and one containing the corresponding German words. (Or in file one, you could have country names, and in file two you could have capital cities, for instance.) The most important thing is that both files have the same number of lines and the words match up in each. If you're using capital cities, and line seven in file one is 'Japan', line seven in file two must be 'Tokyo'. Otherwise the answers won't match up!

So, create two text files in your home directory and enter ten words in each, one per line. For our example, file one is called **english.txt** and contains 'thanks, good, please' etc; file two is called **german.txt** and contains 'danke, gut, bitte' etc. Now you need the Python code to go alongside these text files in your home directory – here's the listing. You can get this from our DVD as **flashcard.py** in the **Magazine/CodeProject** section, but for now just read it through...

```
import os, random

count = 0
score = 0

file1 = open('english.txt', 'r')
file2 = open('german.txt', 'r')

f1content = file1.readlines()
f2content = file2.readlines()

while count < 10:
  os.system('clear')

  wordnum = random.randint(0, len(f1content)-1)

  print 'Word:', f1content[wordnum], ''

  options = [random.randint(0, len(f2content)-1),
    random.randint(0, len(f2content)-1),
    random.randint(0, len(f2content)-1)]

  options[random.randint(0, 2)] = wordnum

  print '1 -', f2content[options[0]],
  print '2 -', f2content[options[1]],
  print '3 -', f2content[options[2]],

  answer = input('\nYour choice: ')

  if options[answer-1] == wordnum:
    raw_input('\nCorrect! Hit enter...')
    score = score + 1
  else:
    raw_input('\nWrong! Hit enter...')

  count = count + 1

print '\nYour score is:', score
```


```
Word: thanks

1 - heute
2 - gut
3 - danke

Your choice: 3

Correct! Hit enter...
```
Terminal - mike@vivaldi: ~/cov
File Edit View Terminal Go Help

› The first incarnation of our flash card program is a simple text-based affair.

With **flashcard.py**, **english.txt** and **german.txt** in your home directory, open up a terminal and enter **python flashcard.py**. You'll see that the program displays an English word and then three possible German equivalents beneath. These equivalents are numbered, so if you think 3 is the right answer, just press 3 and hit Enter. Then the program will tell you whether you got it right or wrong – it does this for ten questions.

Let's look at the code in a bit more depth. We start off with an **import** line, which tells Python which facilities we're going to use. In this case, we need to call an OS function (to clear the screen) and generate random numbers. Then we set up two variables, **count** and **score** – the first is used to show ten questions, and the second stores how many you've got right. Then we have:

```
file1 = open('english.txt', 'r')
file2 = open('german.txt', 'r')

f1content = file1.readlines()
f2content = file2.readlines()
```

Here we open two files, assigning them to two variables called **file1** and **file2**. These variables are like handles for the files – they represent the files stored in memory. But we don't want just the raw files; we want their contents, so the second two lines in this code chunk grab the actual text data into two arrays called **f1content** and **f2content**. Now we have a list of English words in **f1content** and a list of German words in **f2content**. Next, we start our main program loop: »

» **If you missed last issue:** Call 0870 837 4773 or +44 1858 438795.

```
while count < 10:
  os.system('clear')

  wordnum = random.randint(0, len(f1content)-1)

  print 'Word:', f1content[wordnum], ''
```

We want to ask ten questions, so we execute all the indented code ten times (the count variable is incremented each time). The first line of this loop clears the screen by calling the normal **/usr/bin/clear** tool, and then we get a random word to display. We're saying: **wordnum** needs to be a random number representing a line in the word files, so give me a number between zero and the length of the file (in lines). Even though the word files may have line numbers from one to ten, arrays start at zero, hence why we get a random number between zero and file-lines minus one. So, line one in **f1content** (**english.txt**) is actually the zero-th part of the array, and line ten is the ninth bit. Then we display that word from the **english.txt** file.

```
options = [random.randint(0, len(f2content)-1),
  random.randint(0, len(f2content)-1),
  random.randint(0, len(f2content)-1)]

options[random.randint(0, 2)] = wordnum
```

Next, we create an array of three possible answer numbers called **options**. We set each answer to a random number, limited to the maximum number of lines in **f2content** (**german.txt**). We now have three random German words – but hang on, one of them needs to be correct, right? Otherwise we could have three totally wrong answers! So we choose one of our **options** answers to be **wordnum** – that is, the right one. And instead of the correct answer always being, say, number one, we position it at random in the three-choice list of possible answers.

From here, the remainder of the code is very easy to understand. It prints the three possible answers to the screen, then gets a response from the user – ie entering **1**, **2** or **3**. When the user enters an answer, we check to see if it corresponds with the correct word. So, if the English word is 'thanks' and option number two is 'danke', when the user enters '2' our program says: Ah! Option two is the seventh word in the German file. And the original **wordnum** answer was also seven, so that's correct! The words match. Bingo.

Quick tip

If you're experimenting with some Python code and your program gets stuck in a loop, you can kill it from the terminal with Ctrl+C. You may need to press this a couple of times, especially if your program is waiting for some user input. If your program still won't quit, enter **ps ax** in a terminal, find the Python process number for your script, and enter **kill -9 <number>**. That's the ultimate program halting command.

"This program can be morphed to involve words and pictures."

A pictorial version

We've now got a text-based flash card tool, which is great for many purposes, but how about a graphical version? You may want something that shows a picture of an animal, and gives three possible names – ideal for kids. Or perhaps you want to brush up on national flags, and that's exactly what we're going to do here. Like before, however, this program can be morphed into anything that involves words and pictures: foodstuffs and their names in Spanish, sitcom star photos and their screen names...

For this, we need to venture beyond the command line and use a graphical layer. Thankfully, we have the perfect choice for our Python adventures: *PyGame*, a library that links Python with the popular SDL media layer. *PyGame* lets us create windows and display proper images on the screen, all with minimum hassle. Indeed, much of our existing code can remain the same – we just need to make it show pictures instead of plain text.

Whereas our first version of **flashcard.py** used two text files with corresponding words, this graphical incarnation will use a list of words and a list of associated pictures. So, for flags, **file1.txt** may contain 'Nepal, Canada, Finland...' and **file2.txt** would have 'nepalflag.png, canadaflag.png, finlandflag.png...' and so on. We get the list of words from the **file1**, and a list of corresponding pictures to display from **file2**.

The code for this is a bit longer than before, and to avoid wasting space with setup bits we'll just include the main chunk here. Still, this is the majority of the code, and it should show you how a graphical version works. We have the full code listing – with lots of comments (denoted by # marks) – on our DVD in the **Magazine/CodeProject/Graphical** section.

Note that you need the *PyGame* library installed before you run it: most distros have this in their package repositories (search your package manager), but if not you can get the source in our DVD's **Development** section.

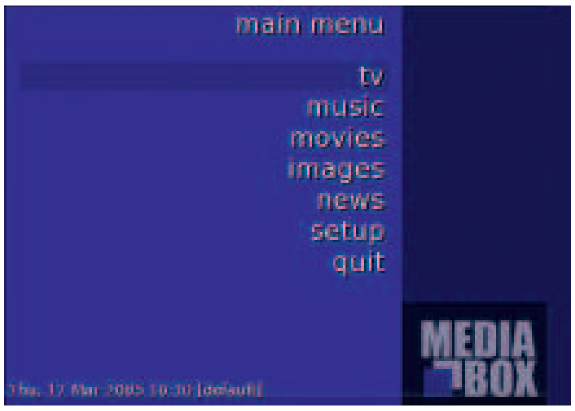
Here's the main chunk of code that we're using. There's some new stuff in here, but in typical Python fashion, it's largely self-explanatory – open an image file, draw it to the screen at a specified location, and so forth.

```
init()
screen = display.set_mode((640, 480))
display.set_caption('Flashcard')

font = font.Font(None, 48)

while count < 10:
  screen.fill(0)

  wordnum = random.randint(0, len(f2content)-1)

  mainpic = image.load(f2content[wordnum].rstrip('\n'))

  screen.blit(mainpic, (255, 50))

  options = [random.randint(0, len(f1content)-1),
    random.randint(0, len(f1content)-1),
    random.randint(0, len(f1content)-1)]

  options[random.randint(0, 2)] = wordnum

  text1 = font.render('1 - ' + f1content[options[0]].rstrip('\n'),
    True, (255, 255, 255))
  text2 = font.render('2 - ' + f1content[options[1]].rstrip('\n'),
    True, (255, 255, 255))
```

Do more with PyGame

Our second flash card program merely scratches the surface of what's possible with *PyGame* (**www.pygame.org**). This library provides a wealth of facilities for loading images, moving sprites around, handling keyboard/mouse input and playing back sound effects. It's a popular choice for open source game programmers – see **www.pygame.org/tags/arcade** for examples of what can be achieved.

Best of all, *PyGame* has copious documentation, including tutorials for complete beginners and a detailed API reference. At **www.pygame.org/docs** you'll find getting-started tutorials on initialising *PyGame*, moving sprites and using pixel effects.



› ***PyGame* isn't just limited to video games – you can use it to create media players too.**

```
text3 = font.render('3 - ' + f1content[options[2]].rstrip('\n'),
   True, (255, 255, 255))

screen.blit(text1, (30, 200))
screen.blit(text2, (30, 300))
screen.blit(text3, (30, 400))

display.update()

```

The first four code lines tell *PyGame* to set up the screen. We initialise *PyGame*, then tell it to create a new 640x480 pixel window, setting the window title bar to some appropriate text. Then we create a new font: we create an object called **font** using *PyGame*'s own font system, and with 'None' we say: it doesn't matter what font we use – just choose the system default. We also want a 48-point font size.

Then the main loop kicks in, running ten times for ten questions. **screen.fill(0)** simply fills the screen with the zero-th colour, which is black – it just clears the screen for each question. Next we choose the line for the random word that'll be the answer, as in our text-based version, and then we have:

```
mainpic = image.load(f2content[wordnum].rstrip('\n'))
```

This is quite a big instruction, so let's parse it. We're creating a new *PyGame* picture object called **mainpic** for drawing to the screen. However, we need to load the picture from somewhere – and we need it to be the correct answer picture. You'll remember that **wordnum** now contains the file line number of the correct answer – we've told Python to choose it at random.

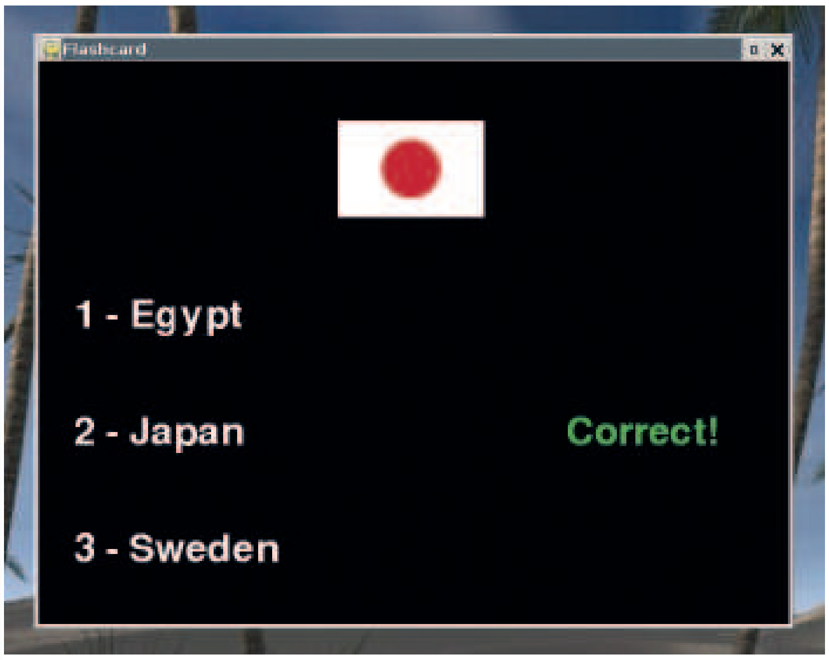
Like before, in the full code we load two text files into **f1content** and **f2content**. **f1content** contains the word list – in our case, 'Nepal, Canada, Finland...'. **f2content** contains a list of corresponding image file names: 'nepalflag.png, canadaflag.png, finlandflag.png...'. Our **wordnum** points to the word in **f1content**, and also to the associated image filename in **f2content**. If **wordnum** is 2, it may point to 'Nepal' in **f1content** and 'nepal.png' in **f2content** – this is why your files should match up!

So, we load our picture from the correct filename in **f2content**, stripping off the newline character with **rstrip** – otherwise *PyGame* gets confused! Next up, we choose one of the three options to be the correct one, as before. Then we create three text snippets to display to the screen. Here's the first:

```
text1 = font.render('1 - ' + f1content[options[0]].rstrip('\n'),
   True, (255, 255, 255))
```

This creates a new image called **text1**, containing the first random option from our **f1content** list of words. The **True** means that we want this text to be anti-aliased, and the **(255, 255, 255)** is the text colour in red, green, blue format. So, our text here is white. We do this for the other two options, then 'blit' (draw) the text to the screen, and update the display to make sure everything is shown.

The remainder of the code, which you can read with full comments on our DVD, handles user input at this point. We check to see if the user has pressed 1, 2 or 3, and react accordingly – print 'Correct!' and update the score, or print 'Wrong!'. Then we wait for Enter to be pressed and restart the loop.



› **With Python and *PyGame*, we can now create graphical tests like this one.**

Finishing off

Hopefully this has inspired you to delve deeper with your flash card project, perhaps expanding it to include five possible options instead of three. Or maybe you could capture the system time at the start of the test, and compare it to the system time at the end, thereby rating how quickly the test was completed!

There's lots to do with these programs, and if you come up with something cool, please do let us know and we'll put it on our DVD. If you have any questions about this tutorial, hop over to the **Programming** section of our forums at **www.linuxformat.co.uk/forums**. Have fun! LXF

» Next month Jaded? Despondant? Fall in love with programming for St Valentine's Day!