LINUX VOICE TUTORIAL

WRITE YOUR OWN PYTHON QUIZ

LES POUNDER

Les Pounder imports functions, defines variables and lists and hones his quizzing skills – all in Python!

WHY DO THIS?

- Program the lazy way, by re-using other peoples' code in your projects.
- Use lists, variables and functions to control a logic flow.
- Display your vast quizzing knowledge to friends and family.

Quizzes are great fun – whether it's a friendly game of Trivial Pursuit at Christmas or a pub quiz down at the Dog & Duck, they're great opportunities to show off your knowledge of trivia. In schools around the world, quizzes are used to test the learning of the students and to consolidate the learning experience.

Creating a quiz is a great way to learn more about structure and control of a program. When writing the code you need to understand how the program will flow: if the player answers the question correctly they progress through the game, but incorrect answers inhibit their progress. The use of programming logic enables the creator to set the pace and the rules for the game while testing their own programming skills.

For our game we will create a quiz with Python-related questions, and to enhance the game we are going to add two libraries to our code:

- **EasyGUI** is an easy way to create a graphical user interface (GUI) for our Python program.
- **Pygame** is a library full of great tools that can enable you to build games and multimedia content. For our game we will use Pygame to add music and sound effects to our project.

Setup

This project can be created using any computer, including a Raspberry Pi. We're using Linux Mint 17, which is based on Ubuntu. We'll also need the Idle development environment, so to install each of the packages open a terminal and type in the following.

To install IDLE

```
sudo apt-get install idle
```

To install EasyGUI

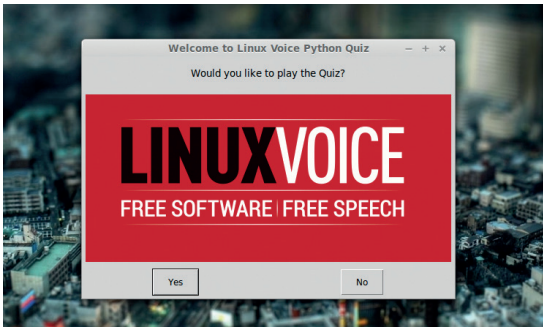
```
sudo apt-get install python-easygui
```

To install Pygame

```
sudo apt-get install python-pygame
```

Once these have been installed, you will need a copy of the project files from **https://github.com/lesp/LinuxVoice_PythonQuiz**. You can also download a Zip archive containing all of the project files from **https://github.com/lesp/LinuxVoice_PythonQuiz/archive/master.zip**.

Idle is an easy-to-use Python editor with an uncluttered and minimalistic interface, enabling you concentrate on writing the code rather than being distracted by fripperies. Idle comes in two versions, one covering 2.x and the 3.x series of Python. For this tutorial I'm using the version for 2.x.



Here's our finished quiz application, written in Python and with nice clickable buttons courtesy of EasyGUI.

When Idle first opens, it presents you with a shell interface that looks very similar to the image bottom right. A shell is an environment where you can prototype new ideas and interact with running programs. The shell is not an ideal environment to write a large program, as it normally works on a line by line basis. If you wish to write much larger programs, which we do, then the best place to work is inside the editor, and to use the editor all you need to do is go to File > New to open a fresh blank editor window.

Plan your logic

Let's open our project in Idle, using the File > Open dialog to navigate to the location where you extracted the project files, so open the file labelled **LV_Quiz.py**.

Once again we will illustrate the intended actions of the project via pseudo code, which is a tool to write the flow of a program in plain English. Here is how we envisage the program will flow.

1. Intro asking the player if they would like to play the quiz
2. If the player wishes to play
3. Reset the score to zero
4. Tell the player their current score
5. Ask the first question
6. If the player answers correctly
7. Add 1 to their score
8. Play jingle
9. Show a dialog box congratulating the player and their current score
10. Else if the player answers incorrectly
11. Play jingle
12. Show a dialog box advising the player of a wrong answer
13. Repeat question twice more to allow player to

answer correctly

14 The question structure continues for three more questions before moving on to the end sequence.

15 Play the intro music

16 if the players score is less than , show a dialog box that commiserates the player and shows their final score.

Else

17 Show a dialog box that congratulates the player and shows their final score.

In order to better understand the project we'll break the code down into sections and step through each section, so let's take a closer look at the code.

Imports

In Python you can easily add extra functionality to any project via the use of libraries. Libraries come in all shapes and sizes, from the simplest, **time** (which enables you to import various time and date capabilities into your program) to the most complex, such as **numpy** and **scipy** which are used by NASA (and which we used in LV003 to hunt for comets – **www.linuxvoice.com/comet-python**).

As with many other Python projects, we first have to import a few extra libraries to further enhance our project. For this project we will import three libraries, and to do that we use the following code, which is included at the top of our project.

```
from easygui import *
import time
import pygame
```

As you can see, we've imported libraries into our code in two different ways. The most common is used twice and that is:

```
import <name of library>
```

In order to use any of the functions contained inside of a library imported in this manner we must call the library and the function by name. For example, if we want to use the **sleep** function from the **time** library, we would do that like this

```
time.sleep(1)
```

This is the most traditional way of importing libraries and is a great practice to follow, but there is another way, and you can see that we have imported the EasyGUI library in a different way:

```
from easygui import *
```

This changes the previously used method of using functions from a library. Using this method to import the library means that we can omit the leading library name and just call the function.

```
msgbox(arguments for this function)
```

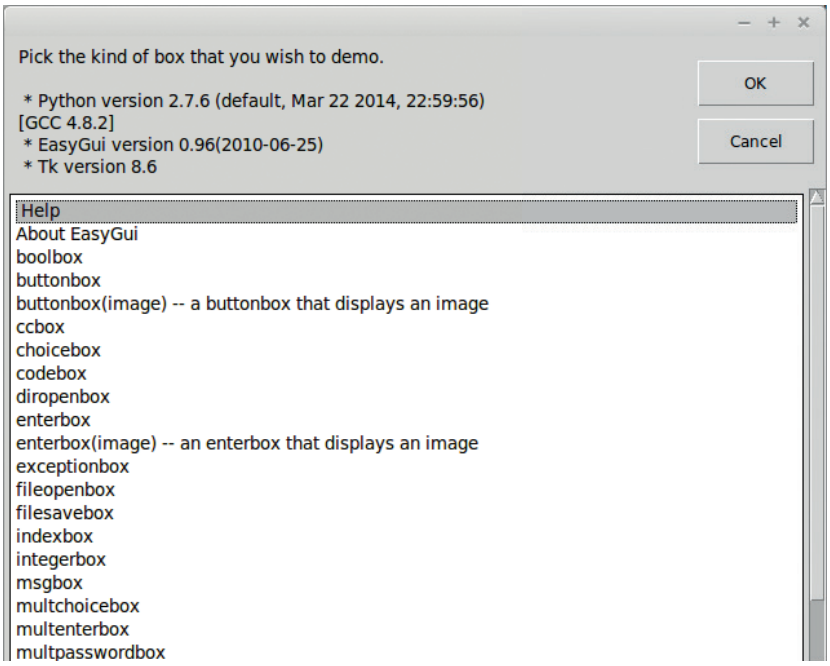
This can be applied to many libraries, and is really useful when working with those new to Python.

There's another method of importing a library, which is to rename a library so that it is easier to use.

```
import time as t

t.sleep(1)
```

As you can see, we have renamed the time library to **t**, which makes it quicker to use. This practice is quite common with Raspberry Pi-based projects, as the Python library **RPi.GPIO** is rather awkward to type and is generally renamed to **GPIO** or **io**.



EasyGUI has an expansive array of many different dialog and menu types. The **egdemo()** function does a great job of showing them all.

Starting Pygame

Pygame is a library full of great tools to create games using Python. With Pygame you can create sprites, characters or objects in the game world, and import video, audio and images into your projects. Entire games can be created using this library, for example the website **https://pyweek.org** showcases many games made in Python including a rather good version of the original Super Mario Bros.

For our quiz we're using the Pygame library to add music and sound effects when certain conditions are met. These audio-based methods of output add a rich atmosphere to a game and provide audio stimulus to the player – think of the jingle you get when you collect coins in Mario or rings with Sonic.

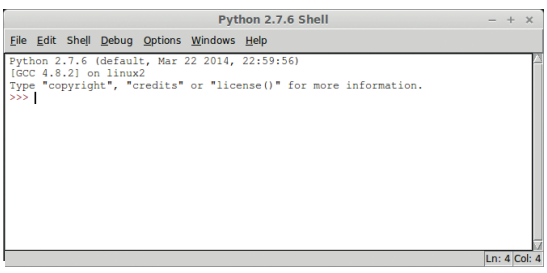
We imported the Pygame library earlier but now we need to start it. To do that you need to initialise the library like so:

```
pygame.init()
```

We then need to initialise the mixer, which controls audio in Pygame.

```
pygame.mixer.init()
```

This is all the setup that Pygame requires at this time. Later in our project we will set up a series of functions that will handle the playback of audio.



As well as the shell, Idle has a power editor that is more than capable of handling any size of project.

Comparison operators

One of the key parts of a quiz is making sure that the player has the right answer, and the mechanism to do that is by comparing the answer given to the expected answer. Below is a table of the most common comparison operators in Python, with an example of how to use each of them in your next project.

Operator	Description	Example
==	Checks if the value of two operands are equal or not; if values are not equal then condition becomes true.	q1 == "Float"
!=	Checks if the value of two operands are equal or not; if values are not equal, then condition becomes true.	if game_start != "No":
>	Checks if the value of the left operand is greater than the value of the right operand; if yes, the condition becomes true.	if score > 3:
<	Checks if the value of the left operand is less than the value of the right operand; if yes, the condition becomes true.	if score < 1:
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand; if yes, the condition becomes true.	if score >= 2:
<=	Checks if the value of the left operand is less than or equal to the value of the right operand; if yes, the condition becomes true.	if score <= 3:

Pygame is an impressive and expansive library and in this tutorial we haven't even scratched the surface of what it can do. If you would like to know more about what Pygame can do (and we strongly reccomend it) head over to their website **www.pygame.org**.

Functions

For our quiz we use three functions: **intro()**, **win()** and **lose()**. These three functions were created to handle playback of audio at key points in the game.

But what is a function? Well, a function is a way of executing a block of program code just by calling its name. Let's take a look at one of our functions

```
def intro():
    intro=pygame.mixer.music.load('intro.mp3')
    pygame.mixer.music.play(1)
```

We start with defining the name of the function; in this example it's **intro()**. Next we create a variable called **intro**, which will contain the output from loading the mp3 intro music into Pygame. Finally we instruct Pygame to play the music that has been queued into the mixer, but to only play the music once. Functions are very powerful and can be expanded into much more versatile tools.

Variables

Variables are an important part of many programming languages, and Python is no exception. Variables are a temporary method of data storage, and can store many different types of data for reuse in a project. For example, we can use a temperature sensor attached to a Raspberry Pi to read the temperature and store the value in a variable, or we can store a player's name. Variables are flexible enough to store anything. In our project we use a few different variables to contain the player's score and location of external image files – here are a few examples.

```
score = 0
logo = "./images/masthead.gif"
start_title = "Welcome to Linux Voice Python Quiz"
```

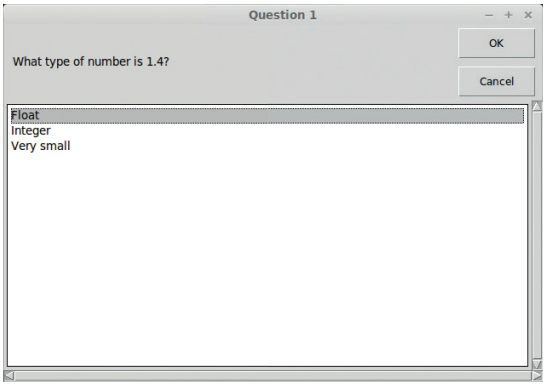
Firstly, our **score** variable is used to track the progress of the player and is updated each time the player answers a question correctly. **logo** and **start_title** are two variables that store a string of text: in **logo**'s case the location of the Linux Voice logo for the intro dialog box, and for **start_title** the text that is displayed at the top of the intro dialog box.

Lists

Another method of storing data in our Python project is to use a list. A list is also known as an array in other programming languages, and by using a list we can store lots of individual items and use them in our code. In our code we use a list to contain the possible answers to questions – for example, we use a list called **play** to contain the answers "Yes" and "No"

```
play = ["Yes","No"]
```

All list contents are indexed, so individual items can be recalled from the list. The first item in a list is



Get the question right and the quiz plays a sound, adds 1 to your score and moves on to the next question.

always index 0. For example, if we wished to print the first item from the **play** list, which is "Yes", then I would do the following.

```
print(play[0])
```

EasyGUI guide

Easygui is a simple method of generating a graphical user interface (GUI). EasyGUI was created by Steve Ferg, who left the project in March 2013. It is now under the maintenance of Alexander Zawadzki, who is keeping the project alive, but the codebase is frozen with little chance of upgrade. Don't let this put you off though – it's exceptionally easy to use.

Using EasyGUI you can easily add a GUI to most Python projects. If you would like to see the full library of GUI elements you can use the inbuilt demo function, remembering to import the library to start with **easygui.egdemo()**.

For this project we're using three different types of GUI elements.

- **Buttonbox** To ask if the player would like to play.
- **Choicebox** To ask questions and capture answers from the player.
- **Msgbox** To update the player on their score.

EasyGUI has an easy-to-learn syntax which is common across all of the many different types of GUI elements it provides. Here for example is the syntax to create a message box.

```
msgbox(title="Title of dialog box",msg="Message to the player",image="Location of the GIF")
```

Providing all of this information each time can be long winded, so to make things a little easier we have created variables that store the various details for each question.

Question structure

Each question is inside a loop that will only repeat if the player answers the question incorrectly, and the player will only have three chances to answer each question before they are automatically progressed to the next question. Using a **for** loop with a range of 0 to 3 we can have the question repeated three times unless the loop is broken by a correct answer.

Under the **for** loop you can see the question being formed using variables such as **msg** and **title**, and there's also a list labelled **q1choices**, which contains the potential answers. All of these variables and the list are then used to create the contents of our first question. To ask the question we first create a variable to store the answer chosen by the player (in this case the variable is **q1**). Here is the code

```
#Question 1
for i in range(0,3):
    msg = "What type of number is 1.4?"
    title = "Question 1"
    q1choices = ["Integer","Float","Very small"]
    q1 = choicebox(msg,title,q1choices)
```

Now that we have asked the question we need to use conditional logic to compare the answer given to the correct answer. To do this we compare the variable **q1** with the hard-coded answer "Float". If the answer given matches the expected result then the **win()** function is called, which plays the audio. We then increment the score by one point. Finally we set up the variables necessary for our GUI dialog box. Once these steps are complete we break this loop and move on to question 2.

```
    if q1 == "Float":
        win()
        score = score + 1
        correct = ("Well done you got it right. Your score is "+str(score))
        image = "./images/tick.gif"
        msgbox(title="CORRECT",image=image,msg=correct)
        break
```

But let's say that our player gets this question wrong – in this scenario we would move to the **else** section of our logic. This triggers our **lose()** function to play audio and then creates two variables that will contain the contents of a dialog box informing the player that they chose the wrong answer.

```
else:
    lose()
    wrong = "I'm sorry that's the wrong answer"
    image = "./images/cross.gif"
    msgbox(title="Wrong Answer",image=image,msg=wrong)
```

Project files

All of the files used in these projects are available via my GitHub repository. GitHub is a marvellous way of storing and collaborating on code projects. You can find my GitHub repo at **https://github.com/lesp/LinuxVoice_Pibrella**.

If you're not a Github user, don't worry you can still obtain a zip file that contains all of the project files. The Zip file can be found at **https://github.com/lesp/LinuxVoice_Pibrella/archive/master.zip**.

Expansion activity

Our quiz is playable, but the code is quite large, with lots of repetition. How can we enhance our code so that we have a much smaller project? The answer may be to use a function with arguments.

Earlier we used functions to control the playback of audio in the quiz. These functions took no arguments and simply ran when executed. A function that takes an argument expects to see one or more additional pieces of information before it runs. Here is a basic example of defining a function that takes an argument.

```
def func(x,y):
  print(x*y)
```

To use this function, we call the function by its name and then substitute the x,y with the values that we wish to use, as so.

```
func(2,3)
```

This will then print the answer to the equation **2 * 3**. For our project we can create a function for each of the different types of EasyGUI elements used, and then use the arguments to dictate what is displayed.

```
def msg(title,msg,image):
  msgbox(title=title,msg=msg,image=image)
```

With this function created we can now test to see if it works.

```
msg("This is the title","This is a message to the player","./images/image.gif")
```

The above code will set the title to be "This is the title" with a message reading "This is a message to the player" and the location of the image is used to grab the image and display it in the dialog box.

So using this new function syntax, do you think that you could make a function for each of the dialogs made in our quiz?

Les Pounder is a maker and hacker specialising in the Raspberry Pi and Arduino. Les travels the UK training teachers in the new computing curriculum and Raspberry Pi.