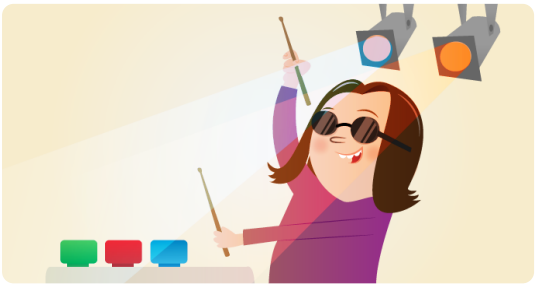
项目

GPIO音乐盒

按下按钮时，制作一个播放音乐的设备。

电子元件 Python



第1步 简介

你会做什么

该项目将向您展示如何将按钮连接到Raspberry Pi的GPIO ()引脚，然后使用它们通过简单的Python应用程序播放声音。

你将学到什么

通过使用Raspberry Pi 创建GPIO ()音乐盒，您将学习如何：

- 用Python播放声音 `pygame`
- 将按钮连接到Raspberry Pi上的GPIO ()引脚
- 使用Python `gpiozero` 库将按钮连接到函数调用
- 在Python中使用字典数据结构
- 确保您的代码可以轻松扩展，以便它对其他项目有用

该资源涵盖了以下**Raspberry Pi数字制作课程的** (https://www.raspberrypi.org/curriculum/)元素：

- **结合编程结构来解决问题** (https://www.raspberrypi.org/curriculum/programming/builder)
- **结合输入和/或输出来创建项目或解决问题** (https://www.raspberrypi.org/curriculum/physical-computing/builder)
- **使用基本材料和工具来创建项目原型** (https://www.raspberrypi.org/curriculum/manufacture/creator)

第2步 你需要什么

硬件

对于这个项目，您将需要：

- 一个覆盆子Pi
- 面包板
- 四（4）个触觉开关，用于制作按钮
- 五（5）个男女跳线
- 四（4）对公跳线引线

软件

你会需要：

- Python 3（预装在Raspbian上）
- `gpiozero` python模块（预装在Raspbian上）
- `libav-tools`，可以通过在终端中()键入以下内容来安装的模块：

```
sudo apt install libav-tools
```

第3步 设置项目

你需要为这个项目提供一些样本声音。Raspbian上有很多声音文件，但使用Python玩起来有点棘手。但是，很容易将声音文件转换为Python可以直接使用的不同文件格式。

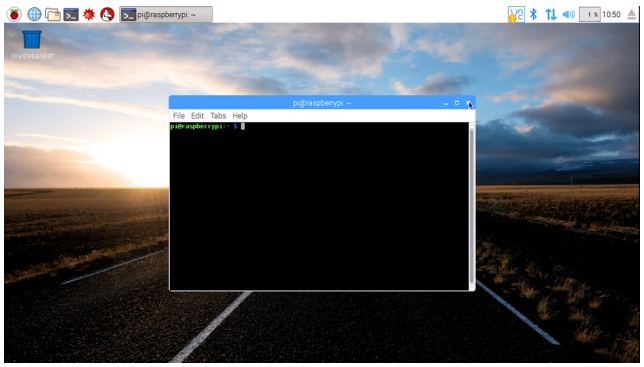
- 您需要做的第一件事是创建一个新目录，用于存储项目的所有文件。创建目录中调用`gpio-music-box`的`home`目录。

> **在Raspberry Pi上创建目录**
>
> 有两种方法可以在Raspberry Pi上创建目录。第一个使用GUI，第二个使用终端。
>
> **方法1 - 使用GUI**

- 单击屏幕左上角的图标打开“文件管理器”窗口

- 在窗口中，右键单击并选择“*新建...*”，然后从上下文菜单中选择“*文件夹*”
- 在对话框中，键入新目录的名称，然后单击“*确定*”

方法2 - 使用终端



- 单击屏幕左上角的图标打开新的终端窗口

- 使用该`mkdir`命令创建一个新目录

```
mkdir my-new-directory
```

- 您可以使用列出当前目录的内容`ls`
- 使用该`cd`命令输入新目录

```
cd my-new-directory
```

复制样本声音

- 因为之前用同样的方法，创建一个名为新目录`samples`的内部`gpio-music-box`目录。
- 存储了大量的样本声音`/opt/sonic-pi/etc/samples`。将所有示例声音复制到您的`gpio-music-box/samples`目录中。

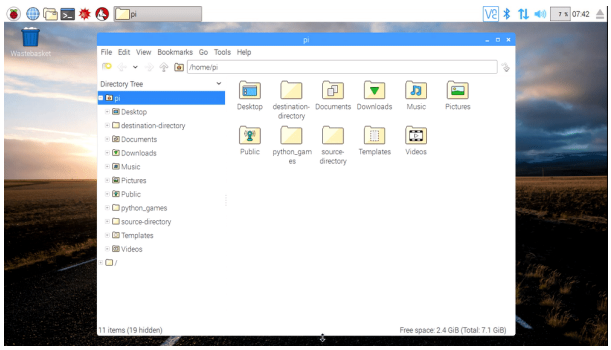
在Raspberry Pi上复制文件

有两种方法可以在Raspberry Pi上复制文件。第一个使用GUI ()，第二个使用终端 ()。

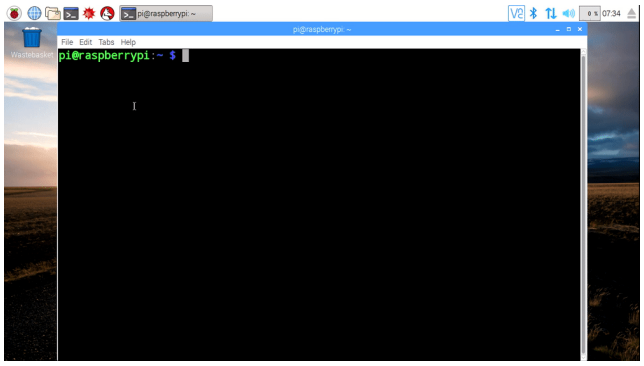
方法1 - 使用GUI ()

- 单击屏幕左上角的图标打开“文件管理器”窗口

- 导航到要复制的文件或目录，然后右键单击它，从上下文菜单中选择“ *复制* ”。
- 导航到要将文件或目录复制到的目录。
- 在目录中单击鼠标右键，然后从上下文菜单中选择“ *粘贴* ”。
- 您还可以使用*套索*选择来选择多个文件和目录。



方法2 - 使用终端 ()



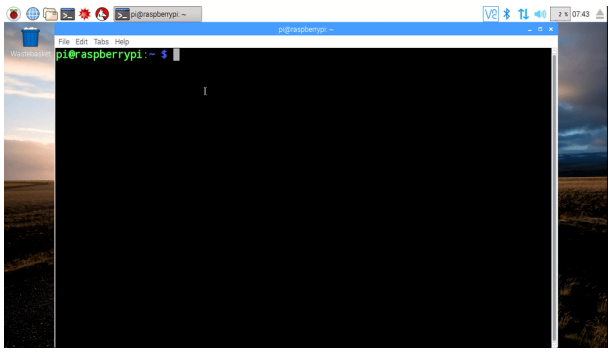
- 单击屏幕左上角的图标打开新的终端()窗口。

- 使用该`cp`命令复制文件。语法如下：

  ```
  cp source-directory / file-to-copy.txt destination-directory /。
  ```

- 如果要复制多个文件，然后你可以使用*通配符*的字符`*`。

  ```
  cp source-directory / * destination-directory /。
  ```



- 要复制包含文件的目录，您需要执行*递归*复制。这可以通过`-r`在`cp`命令后放置标志来完成

  ```
  cp -r source-directory destination-directory
  ```

- 完成后，您应该能够看到`.flac`目录中的所有声音文件`samples`。

以下是终端和GUI方法的视频。

GUI方法

终端方法

第4步 转换样品

- Python难以播放`.flac`文件，因此您需要将它们全部转换为`.wav`格式。
- 请查看以下部分，了解如何转换音频文件以及如何批处理文件。看看你是否可以`.flac`使用**bash**在一批中转换所有样品。

转换媒体文件

您可以使用名为**libav-tools**的软件轻松转换Raspberry Pi上的媒体文件。

- 首先要做的是下载软件。打开终端并键入以下内容：

```
sudo apt update && sudo apt install libav-tools -y
```

- 通过在终端中键入以下内容来检查软件是否已安装：

```
avconv -version
```

- 要将声音或视频文件从一种格式转换为另一种格式，您可以使用以下基本命令：

```
avconv -i input.ext1 output.ext2
```

- 例如，要将wav文件（`.wav`）转换为mp3文件（`.mp3`），您可以键入：

```
avconv -i my_video.wav my_video.mp3
```

对文件运行批处理操作

使用bash对文件运行批处理操作

什么是批处理操作

使用*bash*重命名文件非常简单。例如，您可以使用该`mv`命令。

```
mv original_file_name.txt new_file_name.txt
```

如果你需要重命名一千个文件怎么办?

一个*批次*的操作是当您使用一个命令或一组在多个文件中的命令。

- 使用包含少量文件的任何目录，进行第一次基本批处理操作。首先要尝试的是微不足道的。例如，您可以重新创建`ls`命令，以便;
  - 对于目录中的每个文件，
  - 文件名输出到终端。
- 第一部分是告诉bash你想对目录中的所有文件进行操作。

  ```
  for f in *
  ```

- `f`将一个接一个地表示目录中的每个文件名。
- 接下来，您需要告诉bash如何处理每个文件名。在这种情况下，您希望`echo`文件名为标准输出。

  ```
  做
  echo $ f
  ```

- 这里的`$`标志用于表明您正在谈论变量`f`。最后你需要告诉bash你已经完成了。

  ```
  DONE
  ```

- 如果你愿意，你可以在每个命令后按*Enter键*，并且bash在你输入之前不会运行整个循环`done`，所以命令在你的终端窗口中看起来像这样：

  ```
  for f in * .txt
  >做
  > echo $ f
  >完成
  ```

- 或者，您可以使用a 来分隔命令，而不是在每行之后按*Enter键*`;`。

  ```
  for f in * .txt; 回声$ f; DONE
  ```

操纵字符串。

最后一个命令有点无意义，但您可以使用批处理操作做更多事情。例如，如果您想要更改每个文件的名称，那么如果不是被调用，则会调用`0.txt`它们`0.md`。

- 试试这个命令：

```
for f in * .txt; 做mv“$ f”“$ {f％.txt} .md”; DONE
```

- 如果您`ls`是目录的内容，您将看到所有文件都已重命名。那是怎么回事？
- 第一部分`for f in *.txt`告诉bash在每个`$f`带有`.txt.`扩展名的文件（）上运行操作。
- 接下来，`do mv $f`告诉bash *移动*每个文件。接下来，您需要为bash提供已移动文件的新名称。要执行此操作，您需要删除`.txt`并替换它`.md`。幸运的是，bash有一个内置的运算符来删除字符串的结尾。您可以使用`%`运营商。试试这个例子来看看它是如何工作的

```
words =“Hello World”
echo $ {words％World}
```

- 您现在可以在字符串的末尾结束其他内容。

```
回声$ {words％World）月亮
```

- 所以语法`${f%.txt}.md`用`.txt`字符串替换所有`.md`字符串。很明显，如果你使用`#`运算符而不是`%`它，它将从开头而不是结尾删除一个字符串。

- 基本思想如下：`for`每个文件，使用该文件作为`avconv`命令的输入，然后使用文件的名称，`.wav`而不是`.flac`作为输出文件的名称。
- 如果您需要帮助来弄清楚如何执行此操作，请查看下面的提示。

这是一个动画，展示了如何实现操作以及输出：

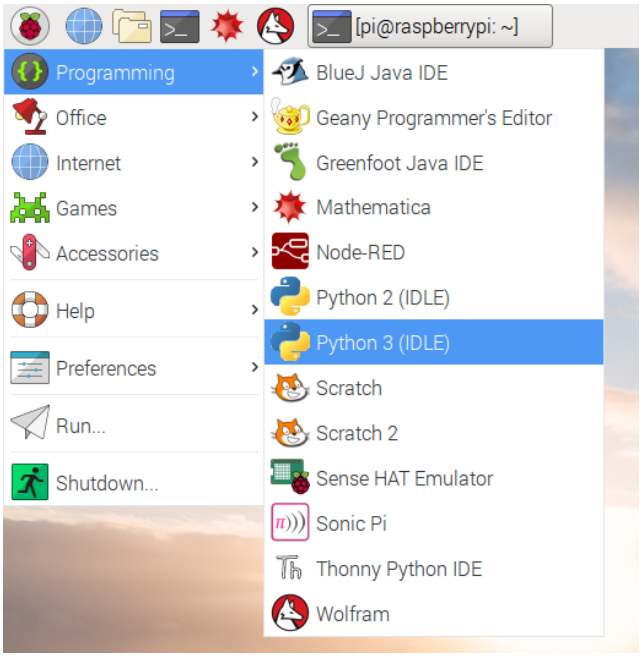
第5步 播放声音

- 现在是时候开始编写Python代码了。您可以使用任何文本编辑器来执行此操作，但IDLE始终是一个简单的选择。

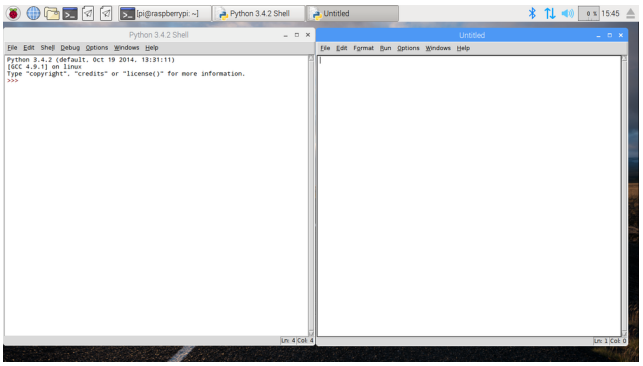
打开IDLE3

IDLE是Python的**I** ntegrated **D** eve **l** opment **E** nvironment，可用于编写和运行代码。

要打开IDLE，请转到菜单并选择`Programming`。
你应该看到IDLE的两个版本 - 确保你点击那个版本`Python 3 (IDLE)`。



要在IDLE中创建新文件，您可以单击`File`，然后`New File`在IDLE的菜单栏中单击。
这将打开第二个窗口，您可以在其中编写代码。



- 创建音乐盒乐器的第一步是测试Python是否可以播放您复制的一些样本。

用Python播放声音文件

To play a sound file with Python, you can use a module called `pygame`. It comes pre-installed on a Raspberry Pi, but if you are on another operating system you may need to use **pip** (https://pip.pypa.io/en/stable/installing/) to install it.
On Linux and MacOS you can open a terminal and type:

```
sudo pip3 install pygame
```

On Windows you can open PowerShell and type:

```
pip3 install pygame
```

Importing and initialising pygame

- First you will need to import the `pygame` module and initialise it.

```
import pygame
pygame.init()
```

Playing a sound

- Next you can create a `Sound` object and provide it with the path to your file.

```
my_sound = pygame.mixer.Sound('path/to/my/soundfile.wav')
```

- Then you can play the sound.

```
my_sound.play()
```

Creating your virtual instruments

- You're going to need four sounds to use in your GPIO music box.
- To do this you can:
  - import and initialise the `pygame` module
  - create four different sound objects using four different `.wav` files

Save your Python file in your `gpio-music-box` directory by clicking on 'File' and 'Save'.

Here's a video showing you the process

Step 6 Connecting your buttons

You're going to need four buttons, each wired to separate GPIO pins on the Raspberry Pi. Check out the steps below and then have a go at wiring up your buttons.

The Raspberry Pi's GPIO pins

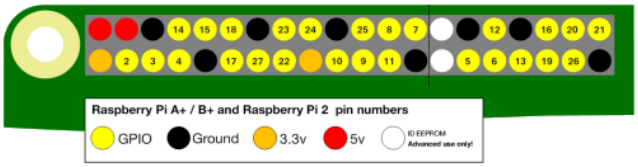
GPIO is an acronym for **G**eneral **P**urpose **I**nput/**O**utput. A Raspberry Pi has 26 GPIO pins. These allow you to send and receive on/off signals to and from electronic components such as LEDs, motors, and buttons.

If you look at a Raspberry Pi with the USB ports facing towards you, the layout of the GPIO pins is as follows.

3V3	5V
GPIO2	5V
GPIO3	GND
GPIO4	**GPIO14**
GND	**GPIO15**
GPIO17	**GPIO18**
GPIO27	GND
GPIO22	**GPIO23**
3V3	**GPIO24**
GPIO10	GND
GPIO9	**GPIO25**
GPIO11	**GPIO8**
GND	**GPIO7**
DNC	DNC
GPIO5	GND
GPIO6	**GPIO12**
GPIO13	GND
GPIO19	**GPIO16**
GPIO26	**GPIO20**
GND	**GPIO21**

Each pin has a number, and there are additional pins that provide 3.3 Volts, 5 Volts, and Ground connections.

Here's another diagram showing the layout of the pins. It shows some of the optional special pins as well.



Here's a table with a brief explanation.

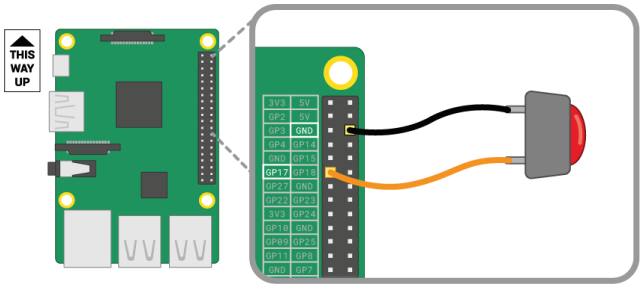
Abbreviation	Full name	Function
3V3	3.3 volts	Anything connected to these pins will always get 3.3V of power
5V	5 volts	Anything connected to these pins will always get 5V of power
GND	ground	Zero volts, used to complete a circuit
GP2	GPIO pin 2	These pins are for general-purpose use and can be configured as input or output pins
ID_SC/ID_SD/DNC		Special purpose pins

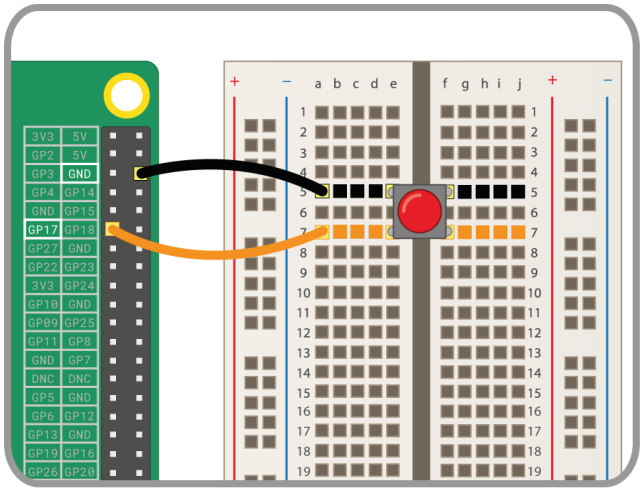
Using a button with a Raspberry Pi

A button is one of the simplest input components you can wire to a Raspberry Pi. It's a non-polarised component, which means you can place it in a circuit either way round and it will work.

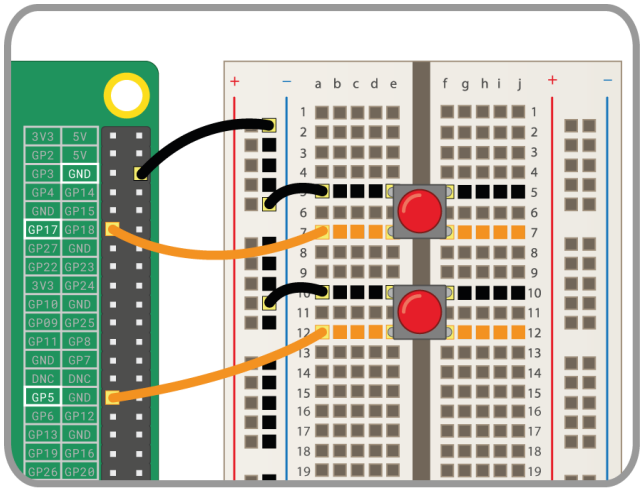
There are various types of buttons - they can for example have two or four legs. The two-leg versions are mostly used with flying wire to connect to the control device. Buttons with four legs are generally mounted on a PCB or a breadboard.

The diagrams below shows how to wire a two-leg or four-leg button to a Raspberry Pi. In both cases, **GPIO 17** is the input pin.





If you are using multiple buttons, then it is often best to use a *common ground* to avoid connecting too many jumper leads to **GND** pins. You can wire the negative rail on the breadboard to a single *ground* pin, which allows all the buttons to use the same ground rail.



- Place your four buttons into your breadboard.
- Wire each button up to a different GPIO pin.

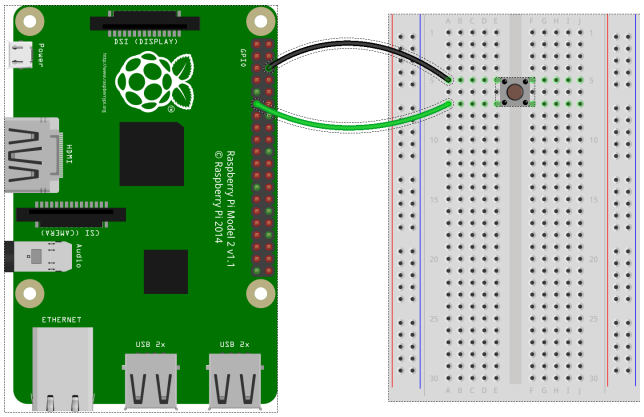
Here's a video showing how the buttons can be wired.

Step 7 Playing sounds at the press of a button

- Have a look at the section below to see how a function can be called using a button push.

Triggering functions with buttons

- In the diagram below, a single button has been wired to pin 17.



- You can use the button to call functions that take no arguments:
- First you need to set up the button using Python 3 and the `gpiozero` module.

```
from gpiozero import Button
btn = Button(17)
```

- Next you need to create a function that has no arguments. This simple function will just print the word `Hello` to the shell.

```
def hello():
    print('Hello')
```

- Finally, create a trigger that calls the function.

```
btn.when_pressed = hello
```

- Now each time the button is pressed, you should see `Hello` being printed to the Python shell.
- Your function can be as complex as you like - you can even call functions that are parts of modules. In this example, pressing the button switches on an LED on pin 4.

```
from gpiozero import Button, LED

btn = Button(17)
led = LED(4)

btn.when_pressed = led.on
```

The function you want to call when the button is pressed is, for example, `drum.play()`. However, when using an event, such as a button push, to call a function, you don't use the brackets `()`. This is because you don't want the function to be called straight away, but rather be called only when the button is pushed. So in this case you just use `drum.play`.

- Now see if you can make your buttons trigger different sounds. Test and run your code to make sure all four buttons make sounds play. If something's not working, then have a look at the hints below.

Here's a video showing how all the sounds can be triggered by pressing buttons. The buttons are wired to pins 4, 17, 27 and 10.

Step 8 Improving your script

The code you have written should work without any problems. However, it's generally a good idea to make your code a little cleaner once you have a working prototype. The next steps are completely optional. If you're happy with your script, then just leave it as it is. If you want to make it a little cleaner, then try the following steps.

- You can store your button objects and sounds in a dictionary, instead of having to create eight different objects.
- Have a look at the steps below to learn about creating basic dictionaries and then looping over them.

Dictionaries in Python

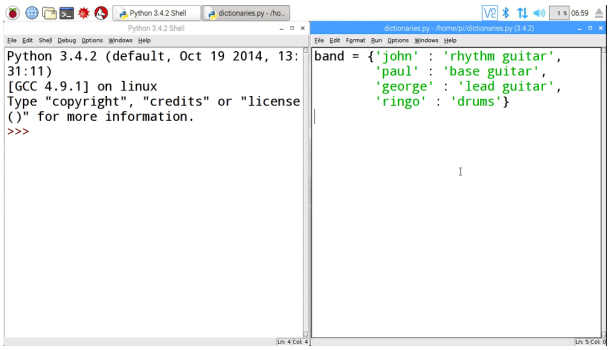
A dictionary is a type of data structure () in Python. It contains a series of key () : value () pairs. Here is a very simple example:

```
band = {'john' : 'rhythm guitar', 'paul' : 'bass guitar',
'george' : 'lead guitar', 'ringo' : 'bass guitar'}
```

The dictionary has a name, in this case `band`, and the data in it is surrounded by curly brackets (`{}`). Within the dictionary are the key () : value () pairs. In this case the **keys** are the names of the band members. The **values** are the names of the instruments they play. Keys and values have colons between them (`:`), and each pair is separated by a comma (`,`). You can also write dictionaries so that each key () : value () pair is written on a new line.

```
band = {
    'john' : 'rhythm guitar',
    'paul' : 'bass guitar',
    'george' : 'lead guitar',
    'ringo' : 'bass guitar'
    }
```

- Open IDLE, create a new file, and have a go at creating your own dictionary. You can use the one above or your own if you like. When you're done, save and run the code. Then switch over to the shell to have a look at the result by typing the name of your dictionary.



- You'll probably notice that the key () : value () pairs are no longer in the order that you typed them. This is because Python dictionaries are **unordered**, so you can't rely on any particular entry being in any specific position.
- To look up a particular value () in a dictionary you can use its key (). So for instance, if you wanted to find out what instrument `ringo` plays, you could type:

```
band['ringo']
```

- Dictionaries can store all types of data. So you can use them to store numbers, strings, variables, lists or even other dictionaries.

Looping over dictionaries with Python

Like any data structure in Python, you can iterate over dictionaries.

- Remember, the order of keys in a dictionary can be unpredictable.
- If you simply iterate over a dictionary with a `for` loop, you will only be iterating over the keys.

Take this dictionary for example:

```
band = {
  'john' : 'rhythm guitar',
  'paul' : 'base guitar',
  'george' : 'lead guitar',
  'ringo' : 'bass guitar'
  }
```

- You can iterate over it with a `for` loop like this:

```
for member in band:
  print(member)
```

This will produce:

```
>>> ringo
john
george
paul
```

- If you want to get the keys and values, you'll need to specify this in your `for` loop

```
for member, instrument in band.items():
  print(member, '-', instrument)
```

This will give the following:

```
>>> ringo - base guitar
john - rhythm guitar
george - lead guitar
paul - base guitar
```

- Can you create a dictionary that contains your `Button()` objects as keys and your `Sound()` objects as values?

Storing functions in Python dictionaries

Using functions in dictionaries

Dictionaries can store all types of data, which makes them useful for many things.

You can store custom or inbuilt functions and methods within a dictionary. There are plenty of times when this comes in handy, for example when, in another programming language, you would use a **CASE** statement.

Have a look at the bit of code below. It's an example of a menu system, where different functions are run depending on whether the user types in the values 1, 2 or 3.

```
def option1():
  print('You chose 1')

def option2():
  print('You chose 2')

def option3():
  print('You chose 3')

choice = input():

if choice == '1':
  option1()
elif choice == '2':
  option2()
elif choice == '3':
  option3()
```

Rather than chaining `if` and `elif` conditionals together, you can use a dictionary of functions. For instance:

```
options = {'1': option1, '2': option2, '3': option3}
```

This now has the possible user inputs as the keys, and the functions to be called as the values. With an additional line, the user's choice can be prompted for and the appropriate function can be called.

```
options[input('Choose 1, 2 or 3 ')]()
```

So the completed code looks like this:

```
def option1():
  print('You chose 1')


def option2():
  print('You chose 2')


def option3():
  print('You chose 3')
```

```
options = {'1': option1, '2': option2, '3': option3}

options[input('Choose 1, 2 or 3 ')]()
```

- If you have a dictionary of `Button()` and `Sound()` objects, you can now loop over them with a `for` loop, so that each button is tied to a different sound.

Here's a video showing how the code can be written:

Step 9 Challenge: a mobile disco

- Can you use different sounds in your program to change things up?
- Why not add more buttons for a greater variety of instruments?
- Can you add some LEDs to your project and make them blink whenever a sound is played?