Watch videos easily

Les Pounder loves watching cartoons, and now he can choose a random video on demand! Now all he needs is a giant bucket of popcorn!

Stuck on what to watch this evening with the family? Fear not – with this project we can have our Raspberry Pi randomly choose a video for us to watch. So let's start building and coding our own random video player! All of the code, and a detailed high-res circuit diagram can be downloaded from tinyurl.com/ydctn9kh.

Building the circuit for this project is simple. We only require three buttons, connected to the GPIO at pins two, three and four, and each of these buttons also needs to be connected to GND. For this we can use a single GND pin on the Pi, and a female-to-male jumper wire. Connected to the – rail of a breadboard means that we have multiple GND connections that can be connected via the two male-to-male jumper wires. See the diagram in the download for this project for details.

When the hardware is built, attach the accessories required for your Pi and boot to the Raspbian desktop.

TUNE IN

Before we can write any Python code, we need to install two libraries that will help us create the project. Open a Terminal, and type the following:

```
$ sudo pip3 install glob
$ sudo pip3 install keyboard
```

We'll talk more about them later. For now open the Python 3 editor, found in the Programming menu and click File>New to create a new blank file. Immediately click File > Save in the new window. Save the code as VideoPlayer.py and remember to save often!

So now we start writing the code, and our first act is to import the libraries that we require. The first three libraries are used to detect the button presses (GPIO Zero), stop the code from running once and exiting (pause) and to choose a random video (choice.)

```
from gpiozero import Button
from signal import pause
from random import choice
```

The final three imports are a library that we shall use to list the contents of a directory (glob), run terminal commands (subprocess) and the last is a library to emulate a keyboard (keyboard.)

```
import glob
import subprocess
import keyboard
```

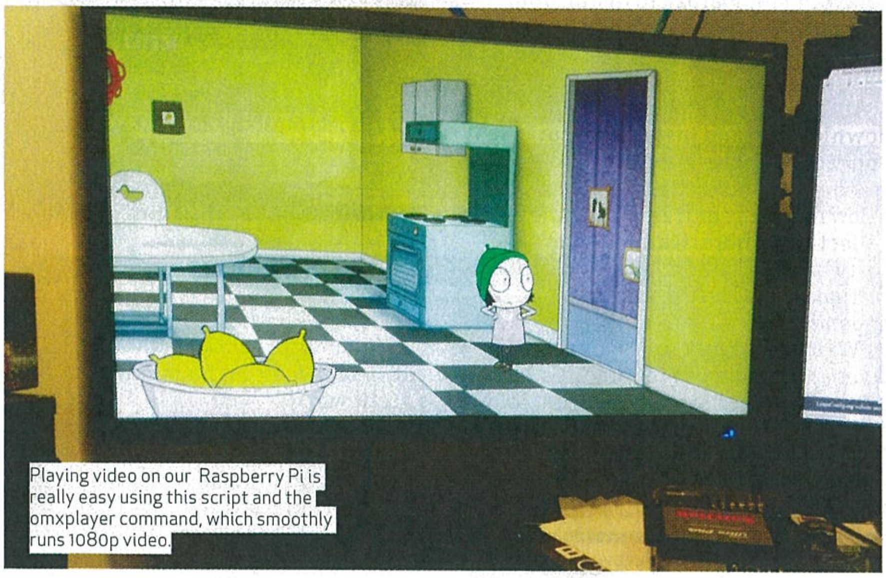
Moving on we create three functions–blocks of code that we can later reuse by calling their name. The first function is called play_video and it first creates a list (a data storage object) called videos .

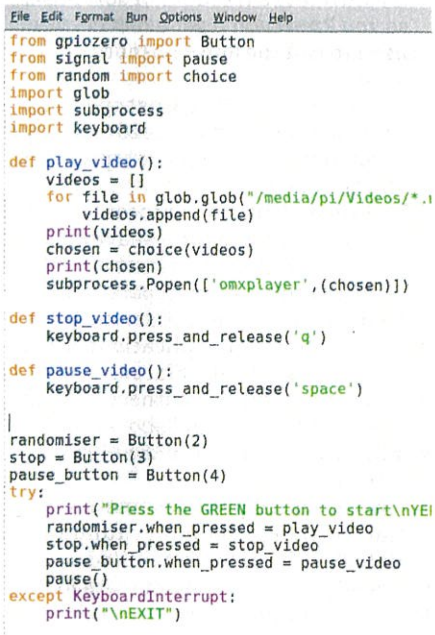
```
def play_video():
    videos = []
```

To fill the list with all the videos we can play, we use a for loop. This will iterate over every file in a directory, as long as it's an mp4 video file. For this we use glob to access the directory / media/pi/Videos, which is really a USB stick called Videos full of mp4 files. Change this to match the name of your chosen directory full of videos. Each time an mp4 is found, it's appended to the videos list that we've just created.

```
or file in glob.glob("/media/
pi/Videos/*.mp4"):
    videos.append(file)
```

Check that the list has been populated with file names by printing the list's contents to the Python shell.


Playing video on our Raspberry Pi is really easy using this script and the omxplayer command, which smoothly runs 1080p video.



```
from gpiozero import Button
from signal import pause
from random import choice
import glob
import subprocess
import keyboard

def play_video():
    videos = []
    for file in glob.glob("/media/pi/Videos/*.
        videos.append(file)
    print(videos)
    chosen = choice(videos)
    print(chosen)
    subprocess.Popen(['omxplayer',(chosen)])

def stop_video():
    keyboard.press_and_release('q')

def pause_video():
    keyboard.press_and_release('space')


randomiser = Button(2)
stop = Button(3)
pause_button = Button(4)
try:
    print("Press the GREEN button to start\nYE
    randomiser.when_pressed = play_video
    stop.when_pressed = stop_video
    pause_button.when_pressed = pause_video
    pause()
except KeyboardInterrupt:
    print("\nEXIT")
```

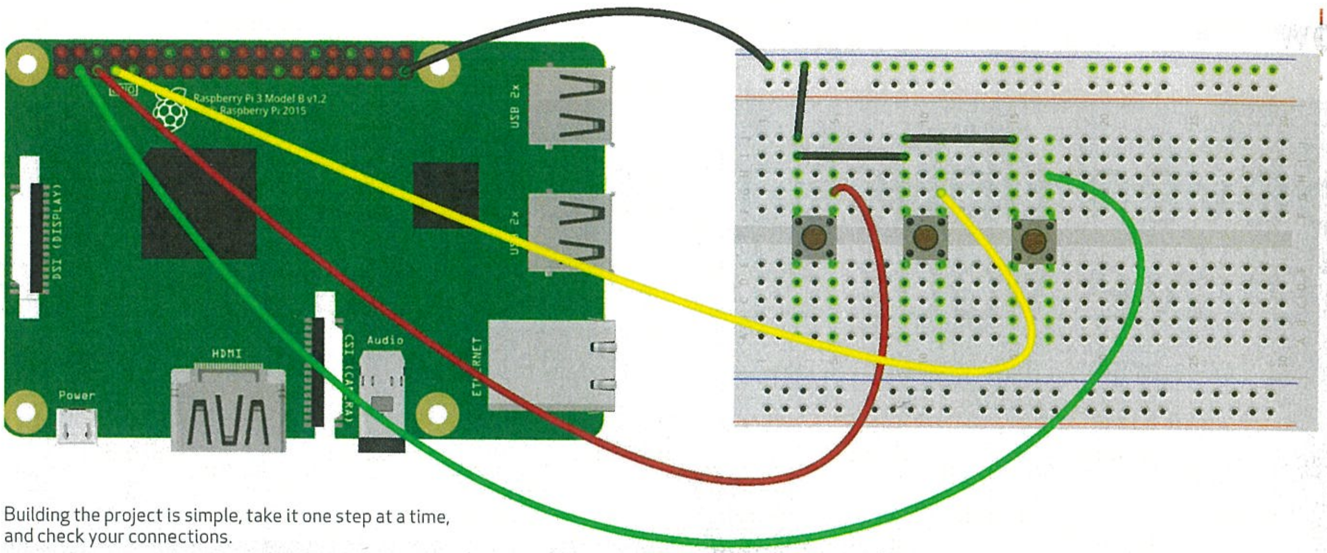
There are only 34 lines of code in this project, and yet we've created a simple media player that can handle 1080p video playback.

```
    print(videos)
```

Choose a random video from the list and store it in a variable called chosen . Print this to the Python shell.

```
    chosen = choice(videos)
     print(chosen)
```

The last line in this function uses the



Building the project is simple, take it one step at a time, and check your connections.

Popen class from the subprocess library to run a command as if we were sat at the terminal, and this command is to open the omxplayer media player, then play the chosen video.

```
    subprocess.
Popen(['omxplayer',(chosen)])
```

The next function is called stop_video and as you can guess it will stop the currently playing video. For this we use the keyboard library, specifically the press_and_release function to simulate pressing Q.

```
def stop_video():
    keyboard.press_and_
release('q')
```

The last function is called pause_video and it emulates pressing the space bar on the keyboard, which is how omxplayer pauses video.

```
def pause_video():
    keyboard.press_and_
release('space')
```

With the functions created, we next need to tell our code where our buttons are connected. We have three buttons connected: randomiser (play), stop and the pause_button. These buttons are connected to the GPIO at pins two, three and four, respectively.

```
randomiser = Button(2)
stop = Button(3)
pause_button = Button(4)
```

PUSH THE BUTTON

On to the last part, which looks for a video when a button is pressed, and reacts accordingly. But first we wrap this section in an exception handler. It'll try to run the code, but if the user presses Ctrl+C then the code will exit. So for the try section our code will first print three lines to the shell – these are the instructions to the user. The \n between each colour is Python shorthand to insert a new line between each instruction:

```
try:
    print("Press the GREEN
button to start\nYELLOW to
pause\nRED to stop")
```

Our three buttons are waiting to be pressed, and using the GPIO Zero library we call the when_pressed class to detect when each of the buttons are pressed. When this happens the corresponding function is run. So when we press the Green randomiser (play) button it'll randomly choose a video from the USB stick and play it. You'll notice that the functions don't have a () at the end of the function name. This is because if they did, then the code would run automatically. Right now they're ready to run on demand.

```
    randomiser.when_pressed =
play_video
    stop.when_pressed =
stop_video
    pause_button.when_pressed
= pause_video
```

The last line in this section is a simple pause(), and this is used to keep the code running, and not just exiting after running once.

```
    pause()
```

But what happens if the user presses Ctrl+C? Well this is a Keyboard Interrupt and if that occurs we shall simply print an empty line and then Exit on the screen.

```
except KeyboardInterrupt:
    print("\nEXIT")
```

So now save your code, and open a Terminal to the directory where you have saved the code. In order to use the code we need to run it with sudo (root powers) because the keyboard library used in the project can only be used as root. To run the code type:

```
$ sudo python3 VideoPlayer.py
```

Press the Green button to play the randomly chosen video, to pause press Yellow, and to stop press Red.

Keyboard shenanigans

In this project we used the Keyboard Python library to emulate a keyboard using just the push buttons that controlled playing, pausing and stopping video playback.

The Keyboard library can be used to press any keys, so we can automate multiple key presses all from one push button. The library can also be used to write text to the screen.

But the most dangerous function of this library is that it can be used to record every key press on the target machine. These key presses can be recorded to a list, then saved to a file and then used for nefarious purposes. Of course, recording the key presses of a user, without their consent is illegal and can land you in a lot of trouble. So don't do it! With that said, it can be used as a powerful tool when debugging how a user interacts with your code, so for software testing (under consent with the general public and in-house testers) you can see what keys they were pressing right before the code locked up and/or went thermonuclear!

The Keyboard library works with Linux and Windows and it's certainly lots of fun to play with. To read a little more about this library head over to this blog post **https://bigl.es/tuesday-tooling-record-replay-keystrokes-with-python** and see a few examples of how it can be used in day-to-day life.