Getting to Grips with Raspberry Pi
Pi Camera Module

TechResort Encounters #5138 C_Camera

What do I need?

- Raspberry Pi computer & power supply
- Raspberry Pi camera and ribbon cable
- Cables and connectors to connect to monitor, mouse and keyboard.

We're going to build on the previous Raspberry Pi camera work you've done but code some more sophisticated applications for your camera.

How are we going to be using the Pi Camera?

We're going to learn a little bit of Python

We'll learn how to control the size of our picture files and to apply special effects to them

We're going to create a timelapse film

By the end you might have some other ideas of projects you'd like to try

Turn the page and get stuck in!

Camera Set Up

- Make sure your Raspberry Pi is powered off
- If the camera isn't already attached to your Pi, install it carefully using the instructions from the previous script.
- Connect your Pi to the monitor, keyboard and mouse as usual
- Power it up with the power supply

Introduction to Python and Pi Camera

In one of the earlier sessions you did some basic work with the Raspberry Pi Camera, controlling it from the system command line.

Whilst the command line is a quick way to get things working, if you use the camera on a regular basis for projects you'll probably get quite fed up typing, and retyping the same commands.

In this module we're going to do all our picture-taking with Python.

Hopefully you've already done some Python, but if not, you can learn as you go in this module.

Important things to remember in Python

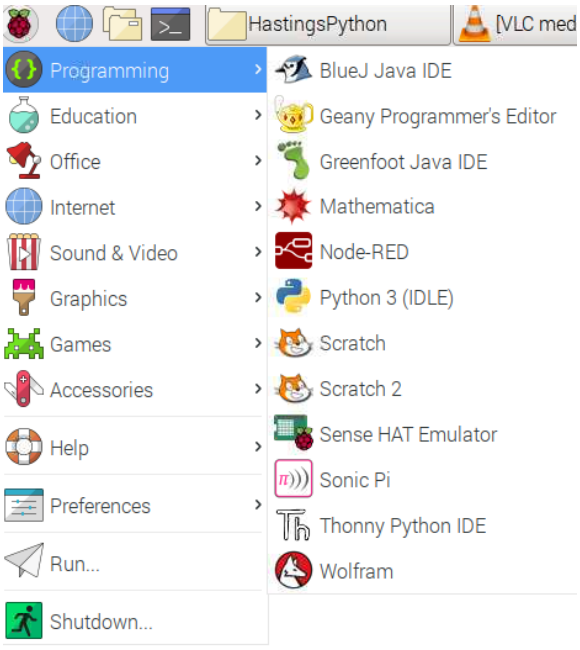
- Conditional and looping statements require text to be indented consistently in order for your program to run, and do what you expect. Pay special attention to indents!
- Commenting code is always good practice – it means you can remember why you did something and also other people know how it's supposed to work.

  The hash (#) symbol starts comments and you need one at the beginning of every comment line.
- It's perfectly fine to have empty lines in your code. The combination of comments and white space can greatly improve the readability of your code. This is really useful for debugging!

Exercise 1 – Getting the Python Editor Underway

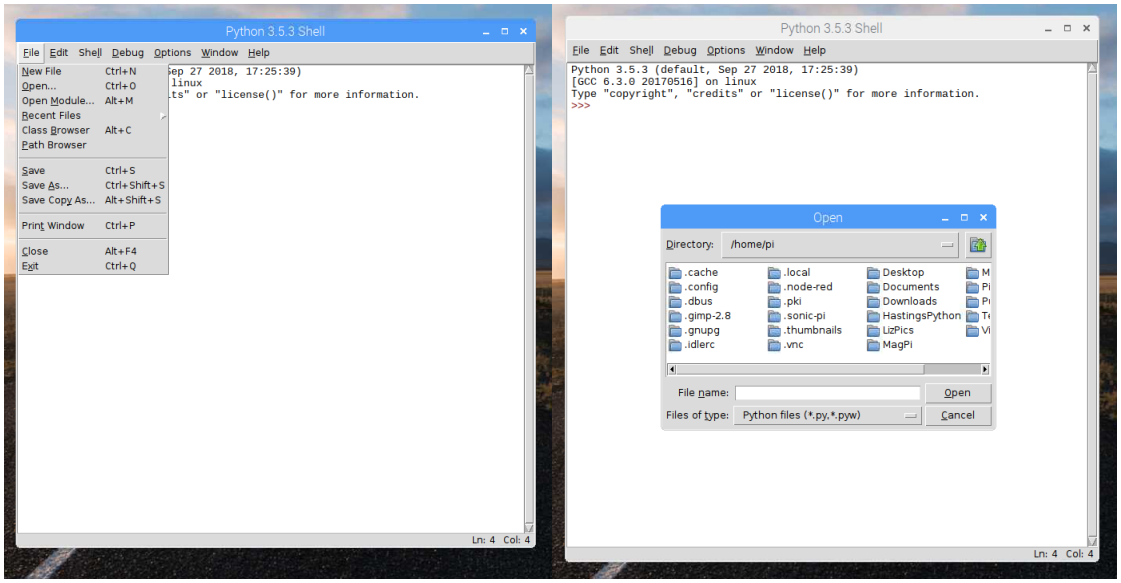
The first thing we're going to do is run a short piece of Python which will allow us to check the camera is working and that we know how to start up and run Python scripts.

Open Python3 (Idle) from the programming menu item

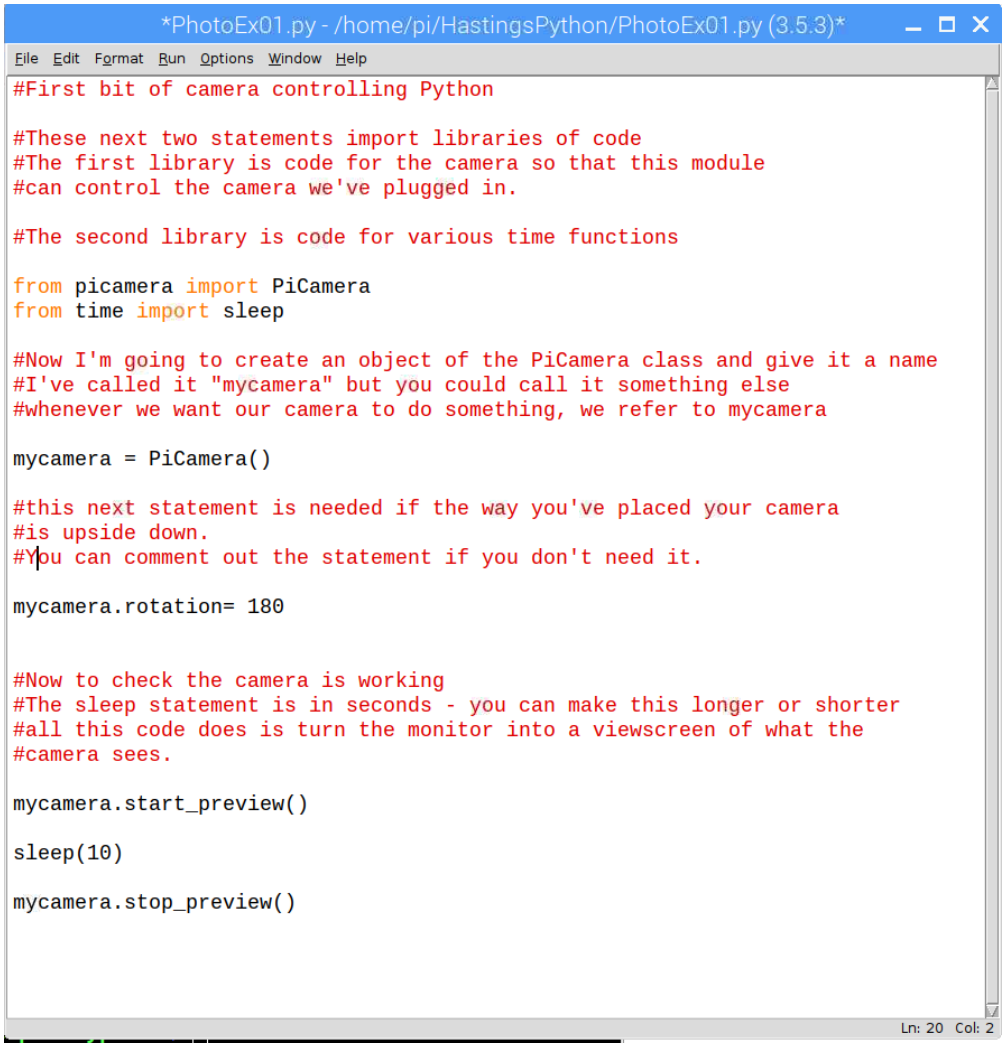


:

And then find the PhotoEx01.py file in the folder with all the camera examples…



This is what the code should look like



Hopefully all this code should be fairly easy to understand but if you don't understand – ask an Elf. Reading the comments should help explain the code.

Note that the comments are in the lines that start with a "#" sign and the Python editor turns them red to make them stand out. The code lines are in black but with key words coloured for emphasis.

Make sure it works by selecting "Run Module" from the Run menu. Note that the shortcut for run module is F5. Whenever you want to run the code, you can just press the F5 key instead.

Classes, Objects and Methods

Just a quick note here about the way languages like Python work. They're called "Object Oriented" languages and this informs the way we structure code. We won't be delving too deeply into this but we'll touch on it from time to time.

Here's an example

picamera is a library of code that someone has helpfully defined for us which contains various things. But we're most interested in the object class PiCamera. (note the capitalisation – Python is very fussy about capitalisation) which defines various things we can do with (or methods) our camera.

Firstly we have to tell our Python module that we'll be using that library so it knows where to look to find some of our commands and parameters. Sometimes we also tell our Python module that we don't need the whole library – just the part we're really interested.

Secondly we'll have to set up (aka "define") an object which we'll state is in the class of objects so we can use that library with our object (in this case our PiCamera)

Then, when we want to do something – known as a method – we'll use 'dot notation' to indicate that we're using the method on our object.

It sounds a bit complicated – but you soon get used to it.

In our example code we said we were going to use the PiCamera library in the statement

```
From picamera import PiCamera
```

We set up an object called something like mycamera using the statement

```
mycamera = PiCamera()
```

Now when we want to tell the code to use a method that's contained in the picamera library we can use dot notation and set any values we want to apply (called arguments) by adding "= " and the value like this:

```
mycamera.rotation = 180
```

or

```
mycamera.resolution = (800,600)
```

If we were using several different objects in the class PiCamera we'd call each object a different name and then use methods separately on each camera.

Important note

Whenever you write Python Scripts or use existing ones and it involves reading or writing files it's REALLY important that you ensure you're referring to a valid path name of a folder that exists on your version of Raspbian (that's unless you're specifically creating new folders).

It will probably start **/home/pi/**

We're giving you some example script files but they use the path name for the version of Raspbian that the script author was using.

As the programmer here, it's **your** job to make sure you check for valid path names before you run scripts.

OK – let's move on.

Saving an Image

- If you haven't already done so, create a special directory in /home/pi/ to save all the pictures you take. (make sure the directory is a simple name with no spaces – for example FredPics).

- Now change the Python code to include a statement like this in the line after the sleep statement (which you might want to set to a shorter duration):

```
mycamera.capture('/home/pi/xxxxxxxxx/image.jpg')
```

  Where the 'xxxxxxxx' is the folder you created to put your images in and 'mycamera' is the name you called your camera object.

- Save and run your code. Does it work? Check that the image.jpg file is where you expected it. View the file by double-clicking on it in File Manager.

- You could set your image to be the image on your desktop.
  Hint: Preferences – Appearance Settings

Shooting a Series of Images

- We're going to introduce a simple loop

- Add the following code in the line above the "start_preview" statement

```
for loopcounter in range (5):
```

- This is an instruction to run the next bits of code 5 times. But to identify which bits of code will run repeatedly you need to indent them all by the same amount. To start with indent the next three lines of code (by the same amount)

- If you want to see the difference…run your code once with the "stop_preview" line indented and not indented. What happens?

- "loopcounter" in this code is a variable: a word to represent a value that we'll possibly use several times.

Question

At the end of this code how many images have you written?

Why?

Using a variable in a file name

- You probably noticed that only your final image remained in the folder – that's because the module overwrote the image each time.
- If you want to keep all the images (and you probably do) then you need to make the code automatically change the file name each time.
- You need to tweak the `mycamera.capture` statement to format the file name something like this:

```
mycamera.capture('/home/pi/LizPics/image%s.jpg'
%loopcounter)
```

  be careful to make sure the pathname matches your setup (rather than the set up for author of these notes).

- How does this work?
  the `%s` and the `%loopcounter` are a pair of values.

  Each time the loop runs through the writing the file it'll replace `%s` in the file name with the current value of the loopcounter variable but as a string value (a string is a series of one or more characters – not numeric values).

- Try it.
- Did it work?

Question

If you increased the loop to a number with more than a single digit (that is, 10 or above) what will happen to the file names?

You'll get filenames like:

image0.jpg

image1.jpg

image2.jpg

…

image10.jpg

image11.jpg

…

image100.jpg

image101.jpg

This isn't a problem because all the file names will be unique but you might want to always make sure you see them in the order they were taken and, in most cases, if the files are named like they are above it can be difficult for software to sort them by the filename. They would sort like this:

image0.jpg

image1.jpg

image10.jpg

image100.jpg

image101.jpg

image11.jpg

image2.jpg

Which is annoying if it's important to have your photos in time-order

Thinking Ahead

When you're writing code and naming directories and files always try and think ahead to how someone will use your files and try and design your solution accordingly.

- Tweak your capture statement to look something like this

```
mycamera.capture('/home/pi/XXXXXXX/image%03d.jpg' %loopcounter)
```

Don't forget to use the path name on your Pi…not just copy the one above!

- Now, instead of a single digit in the file name, there are 3 digits and there will always be 3 digits to represent the number in the series.
- Sorting the files will now keep them in the right order.

Question:

Do you need the "preview" in your code?

No – you can take pictures without a preview but it can be useful for positioning your camera at least at the beginning.

Timelapse/Stop Motion

You've now got all the code you need to build a series of photos to turn into a stop motion film.

Simple Stop Motion Program

Close all the other Python code windows you have open and find the PhotoEx02.py file and open it. You'll need to **check and tweak the path** for saving the files.

Hopefully, since all the code is documented you should be able to work out what it's doing.

Run the code and generate some pictures – does it work OK?

If not, debug and fix it.

Once you've checked it works in principle you're going to generate some images for a simple stop frame animation

- Make a new directory to store your images in
- Adjust the pathname in your code to send your images to the new directory
- Run your module and create a series of about 20 images (don't do more than that at this stage).
- When you've finished, check your images are safely in your new folder.
- To render the images to a video file we're going to use a command line program called *avconv*.
    - Go to a terminal window
    - Type the following command

```
avconv -r 10 -i animation/frame%03d.jpg -qscale 2 animation.h264
```

the switches used here are:

-r which is the frame rate (in frames per second…or fps)

-i which is the direction to the input files

-qscale which is quality scale (from 1 = excellent to 31 = worst)

Make sure your pathname and filename (coloured red above) matches your pathname and file names.

This should now render a video file called "animation.h264" into the home/pi directory.

You can play the file through VLC media player.

You can try tinkering with the frame rate and the qscale so see what happens but because we're only tinkering with a few frames, you probably won't get a good feel for things at this stage.

But you might notice the animation doesn't play very smoothly. This is probably due to the large size of the files.

Question

Can we reduce the size of the files to make it play more smoothly?

The answer is yes and we could do it a number of ways: keeping the image resolution large as we capture them and resize a copy of them. Or we could capture them at lower resolution. That's what we'll do for this exercise so that we can keep things simple

Exercise: Controlling the size of images in Python

- This is really simple, as it turns out. There is a method for the PiCamera class of objects called "`resolution`" which takes two parameters: the horizontal and the vertical resolutions of the image you want.
- Add a statement formatted like this (make sure to use the name of your PiCamera object):

```
yourcameraobject.resolution = ( xxx , yyy)
```

- where xxx is the horizontal resolution (less than 3280) and yyy is the vertical resolution (less than 2464).
- Suggestion: start with something like 800 x 600 and check the file sizes being rendered are 500 Kb or less.
- Now run through the animation exercise again (put the new photos in a fresh directory) with the smaller images to see if the animation runs more smoothly.
- You could try doing this with a larger number of individual images to get a longer animation.

Challenge

You've got a lot of the basics of both Python and using the PiCamera class under your belt. Until now we've really only been dealing with stills.

Here's a challenge we haven't provided you any code for:

You're going to record 15 seconds of video footage and output it to a file

- Start a new Python module
- Import the sleep part of the time library
- Import the PiCamera part of the picamera library
- Set up a new object of the class PiCamera
- Set the resolution of the camera to something like 640 x 480 using the "`.resolution (x,y)`" method
- Start the preview
- Record to a .h264 file using the method "`.start_recording('your path and file name in here')`"
- Record for 15 seconds using the method "`.wait_recording (time in seconds)`"
- Stop recording using the method "`.stop_recording()`"
- Stop the preview

Write (and debug) your code to get this working.

If you get stuck you can compare it to our code in the PhotoEx03.py file (but try before you look at it).

Extending the challenge…

- Can you launch your fixed-time video recording by typing a specific key (with return)?
- Can you change the amount of time you record for by the user typing in a time? (how can you trap potential errors?)
- Can you do a series of short burst recordings and make sure the files don't get overwritten?

Special Camera Settings

So far we've only done still and video capture using the PiCamera's automatic settings.

You probably know that with most cameras you can override automatic settings to get the particular effects you're after.

Pi Camera is no different. We can use code to change override the automatic settings. We're only going experiment with a couple of settings but there's scope to do loads more.

Here's a snippet of code to show you the effect of changing the image's brightness and annotating the image to tell you what the brightness setting was for the image.

```
mycamera.start_preview()
for counter in range(100):
      mycamera.annotate_text = "Brightness: %s" %counter
      mycamera.brightness = counter
      mycamera.capture
('home/pi/LizPics/BrightnessTest/image%03d.jpg' %counter)
      sleep(0.5)
mycamera.stop_preview()
print ('ALL DONE')
```

Check you understand how it's working and write your own module (or tweak one of your existing modules) to incorporate this.

Notes:

- Be sure to name your camera consistently
- Make sure your path for saving files exists already…and adapt your code to include the correct path.
- The "Print" statement will just show you in the terminal window when the program is finished.
- You can also change the image contrast in the same way, using something like "mycamera.contrast"
- If you're building up lots of directories of pictures which aren't very useful you might want to do some housekeeping to make sure you don't run out of space (hint: use file manager)

Enhancing your application

You might have found it a bit annoying to have to go and write a new directory (or check the name of an existing one) to put in your code. You might rather create a new directory within your program.

It's not that hard.

Use an input statement and use it to prompt the user to input the name of the directory they want to create.

```
newdir = input('Where shall I put your images?  ')
```

This statement translates to

> Print on screen 'Where shall I put your images?' and wait for the user to type some text and press return then save the text they typed to the variable "newdir".

Now we know what the user wants to call the new directory we have to make a string with the whole path name (including the new directory name).

```
path = "/home/pi/" + newdir
```

Which means: take the path name /home/pi/ and adds the new directory to it in the form of a string (the technical word for this is "concatenation") then save this new string in the variable "path".

These two statements prepare the way to make the directory.

The statement to make the directory is

```
os.mkdir(path)
```

The final piece in the puzzle is to ensure the new files get written to the new directory - it should be something like this ,using string concatenation with "+" again… (using your own camera name and looping variable)

```
mycamera.capture(path+'image%03d.jpg' %counter)
```

BIG WARNING

There's no error checking for valid path names or directories that already exist.

So be careful when you're testing.

Question:

What error checking could you do?

More sophisticated use of the range statement.

In the modules you've been tinkering with we used the "range" statement to determine the number of time a loop executes.

When we used it for brightness we probably didn't need to see all the steps and we could certainly have predicted that a brightness of zero was pretty useless.

We can improve the range statement where the looping value itself is as important as the number of times the loop happens.

If you give it only one parameter it will take it as the stopping point with a start point of zero and a step size of 1.

Because the check for range happens at the beginning of the loop a statement of

```
range(10)
```

will stop the loop immediately the looping variable reaches 10 and doesn't do the code in the loop when it has the value of 10.

Using the brightness example, range(10) will take pictures with brightness 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

And then stop.

But, range can take 3 parameters to allow you to control things with more sophistication

The first is the starting point

The second is the finishing point

The third is the size of the steps

What values would counter use to loop with here?

```
for counter in range(20, 80, 5)
```

Can you tweak your latest module to control the brightness on your photos without taking too many, and starting and ending at sensible values?

Give it a try and ask an elf if you get stuck.

Then go and compare it with my specimen code in PhotoEx04.py

Special effects

There's a special attribute you can use to apply weird and wonderful effects to your pictures.

Load up the PhotoEx05.py file to see the effects in use.

You'll see the names of the different effects and you can apply them individually. For example, to apply the sketch effect you would use the following statement before the image capture:

```
Mycamera.image_effect = 'sketch'
```

Note that you need to put the name in quotes since it's a value, not a variable.

What's next?

That's really up to you.

You might decide you've had enough of photography for now. That's fine. Go and try one of the other modules (Sonic Pi, Electronics or Python Games).

But if you want to build on your knowledge to write your own application for the camera you could try one of these challenges:

- Try the electronic button controlled camera module script
- Use what you've learned here to develop an application which acts as a photobooth perhaps previewing different photo effects and then capturing the one of the subject's choice.
- Create a short timelapse or video film of the workshop (make sure people are happy to appear in it).
- Or you might have an idea of an application with a camera that you'd like to try and build. Ask an elf for a PDF copy of the Pi Camera manual if there's something you're not sure how to do.