ISSUE 06 - OCT 2012

The MagPi™

A Magazine for Raspberry Pi Users

The Skutter Has Landed

- Camera Pi
- Win a LCD mount
- Portable power for your Pi
- A pumpkin with a difference

ISSN 2051-9990

The MagPi™

http://www.themagpi.com

The MagPi

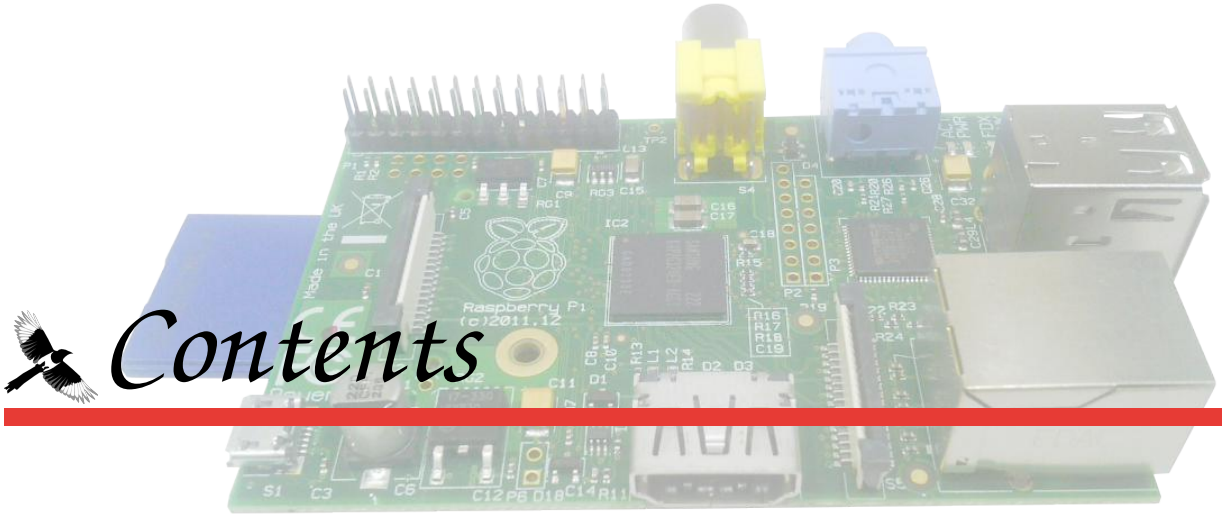
Welcome to Issue 6,

In answer to many requests from readers, the Skutter robot series is back with the next thrilling installment. To get robot projects or field experiments moving, there is an article on powering from batteries. The MagPi also presents a slightly different use for a pumpkin, as well as interviews and programming columns.

If you have not done so already, then we suggest you try out the new turbo Raspbian image, the speed increase is really noticeable.

Ash Stone
Chief Editor of The Magpi

Ash Stone
Chief Editor / Administrator / Header
Jason 'Jaseman' Davies
Writer / Website / Page Designs
Tim 'Meltwater' Cox
Writer / Photographer / Page Designs
Chris 'tzj' Stagg
Writer / Photographer / Page Designs
Ian McAlpine
Page Designs / Graphics
Joshua Marinacci
Page Designs / Graphics
Lix
Page Designs / Graphics
PaisleyBoy
Page Designs / Graphics
Sam Marshall
Page Designs / Graphics
Andrius Grigaliunas
Photographer
Matt '0the0judge0'
Administrator / Website
Bodge N Hackit
Writer
John Ellerington
Writer
Gordon Henderson
Writer
Colin Deady
Writer / Page Designs
Spencer Organ
Writer
Luke A. Guest
Writer
W.H.Bell
Writer
Colin Norris
Editor / Graphics (C Cave Header)
Antiloquax
Writer



Contents

Adding a motorised base Part 1

DIFFICULTY: ADVANCED

So far we have focused on controlling the robot arm part of this project.

The robot arm is a very exciting piece of kit with lots of possibilities, but I would also like to cover another area that could open up the field of robotics to a lot of people, particularly if you are working on a modest budget and can't quite get your hands on one of these yet.

Getting Around

Any robot needs a way to move around. In most cases robots use motorized platforms, but there are a few notable exceptions to this.

In the first part of this article I'm going to devote some space to the different types of "mechanical platforms" that are out there.

I am also going to have a look at some ideas for "do it yourself" and finally I will explain my own solution to this mechanical platform issue.

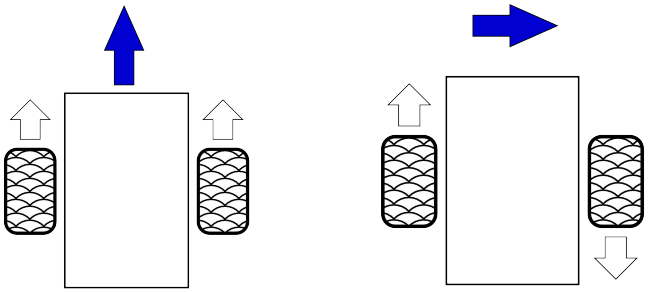
Let's begin by looking at nature. There are insects with multiple legs, mammals with four legs (or two) and then there are snakes which don't have any legs at all! There have been some wonderful developments in the field of robotics which incorporate all of these forms of locomotion.

There are now even swimming robots that hang out with real fish to hunt and identify the bad guys who pollute our oceans!

For the most part, the technology that is used to make these kinds of robots work is prohibitively expensive for the amateur enthusiast, not to mention excruciatingly complicated and almost certainly out of range of those of you who might be considering making a robot on a pocket money budget.

The alternative is to build a motorized platform with some sort of wheels. This instantly brings a robot back down to earth in terms of price and the level of difficulty. The first thing to consider here is – what sort of platform?

There are basically two options. The first is a "differential" platform which works in a similar way to a tank. You have a motor on each side of the robot that's attached via some gears to a wheel.

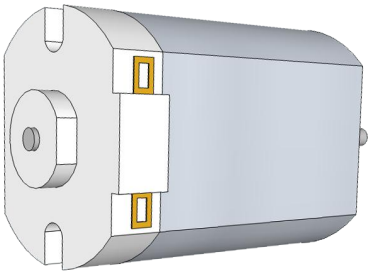


When the wheels are turning in the same direction, the robot will move in that direction. If the wheels rotate in different directions then the robot will turn.

The second option is to use "car" steering, where one motor drives the robot forwards or backwards and a second method, such as a servo, is used to change the angle of the front wheels and thus to steer it. Out of these two methods, the differential platform has the advantage in that it can perform turns in much tighter spaces than the car steering method.

Types of motors

Another consideration is the type of motor that could be used. The simplest and cheapest option is to use a DC motor.



This uses a single coil (containing thousands of turns) of wire attached to the motor shaft, called a “commutator” and two opposing magnets. Applying an electric current to the coil creates an electromagnetic field. The poles of this field are attracted to the opposing fields of the magnets and this makes the shaft move half a turn until the opposing poles are as close to each other as they can get. At this point it switches the commutator and reverses the polarity of the electromagnetic field. The coil is then repeled away from the magnetic poles it was just attracted to, moves the rest of the way around to the opposing fields and the cycle begins again. More power makes it go faster (up to a point, and then it burns out or blows up!). Reversing the current makes the motor run in reverse. This is nice and simple but the drawback is that it is next to impossible to make the DC motor move by an exact amount.

Another option is to use something called a stepper motor. Instead of having a single coil and a commutator, a stepper motor has many coils and a magnet. Each one must be switched on and off in sequence in order to move. This means the motor turns in many tiny little ‘jerks’. You can specify with extraordinary accuracy how far to move a stepper motor but the disadvantages are that they are much more complicated to control and have much less torque (rotational force) than a DC motor.

Whichever motor option you decide to use, it is unlikely that you will simply be able to connect some wheels to it and ‘hey presto’ off you go. The motor will either be too fast or have insufficient torque or “horsepower”. In order to make it useful we will need to use a system of gears or pulleys. For example, a motor that revolves at 10,000 rotations per minute will be rather fast, to say the least. What's more, attaching this directly to any robot with anything more than the tiniest weight and it will be too weak to budge it.

A more sensible (but still nippy) speed might be in the order of 2,500 RPM. In order to achieve this we would need to use gears (or “cogwheels”) with a ratio of 1 to 4. In that way the speed is reduced by a factor of 4 and the torque is increased by the same amount. That means that the gear attached to the motor needs to be four times bigger than the one attached to the wheel. You can achieve the same result by using a combination of smaller gears. Nevertheless this adds another level of complexity to our project.

It is possible to buy electric motors with prefabricated gear assemblies that are specifically made for the robot and hobby market. These have the advantage that a wheel can be directly connected to the axle on the motor, without having to worry about making your own gear assembly. The disadvantage is that they are comparatively expensive, often in the order of £10 or more for a single unit.

Acquiring a motor

You might be reading this and be starting to feel a little bit daunted at the complexity and cost of embarking on a project like this, but don’t worry. I'm going to tell you about another method you can use to make a motorised robot platform. You can cheat!

There are countless reasonably priced motorised toys on the market which, with just a little bit of hacking and bodging that, you can make into a robot platform that can compete with some of the best. Even better is that you don’t even have to pay full price for something like this.

In almost every country in the world there are “car boot sales” or “yard sales” or such like where, with just a little luck and some good hunting, you have a good chance of being able to find a suitable toy which can be converted into a platform – it is likely that such a toy will

use DC motors and you have a greater chance of finding a toy that uses car steering instead of a differential but these sales hold some real gems for the resourceful amateur robot builder.

I have used this cheating method for my own skutter robot. I obtained a Big Trak toy, which you can buy new for about £20 (it's not unlikely that you might find one of these in a car boot sale too, if you are lucky).

To help show you how you can adapt toys into robotic bases I'm going to describe to you how I adapted this Big Trak for my Skutter.

Adding a Big Trak platform

1. Removing the bumper

First disassemble the Big Trak. Most of the screws are easy to access. However one is hidden behind a grey plastic bumper on the rear.

To get to this screw, first unscrew all others to allow Big Trak to be prized open just enough to allow the bumper to be unclipped from inside.

2. Removing the keypad

After removing the lid, unclip all accessories such as the plastic turret etc. Some will be stiff but eventually came off with a bit of patient wiggling.

The keypad and ribbon cable to the electronics beneath prevents the lid being removed. This keypad is glued in place and can be peeled off easily to allow complete removal of lid.

3. Disconnect the motors

Once the lid is removed, unscrew the circuit board below. This reveals a plastic gear box which also houses the two DC motors that power the big trak.

Snip off the wires that are attached to the motors. Solder two double core “bell wire” type cables to each + and – terminal on the two motors.

4. Test the motors

Now test that the motors work. A standard D cell 1.5v torch battery will suffice to see the base trundle forward, reverse and turn depending on how the two wires are held to the battery terminals. In a nutshell that’s it! A fantastic motorized platform for a robot.

To control the motors some reasonably simple electronics are required so that the GPIO on a Raspberry Pi can instruct them to move. (A stern word of warning – do not under any circumstances electrically connect a motor directly to any part of a Raspberry Pi, doing so will certainly cause it to try to handle too much power and will, putting it bluntly, kill it. I will deal with the electronics to safely interface the motors with the GPIO header in another article)

5. Mounting the robot arm

Further adaptions to the Big Trak are required to mount the robot arm inside. The hole for the “turret” on the Big Trak is almost in the perfect location and size to allow the base of the arm to sit inside with the “shoulder” protruding through.

For complete perfection, score around the edge of the hole with a stanley knife or similar and then use a pair of wire snippers to cut out and break off sections of the lip around the turret hole.

Use a file to smooth the rough edges or sandpaper, or even the side of a box of matches.

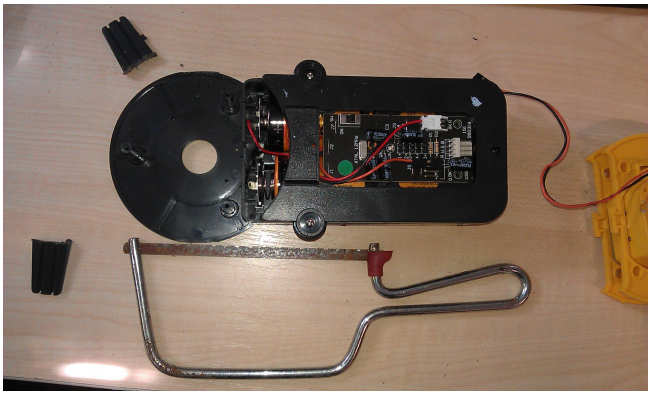
A little more work on the Big Trak body is needed to enable the base of the arm to fit inside.

Use the wire snippers to cut away the plastic loudspeaker mounting on the bottom of the Big Trak.

6. Fitting the arm

Finally, a modification to the robot arm base is required, remove the stabilizers from the base of the arm with a hacksaw. (I don't think the arm really needs these anyway).

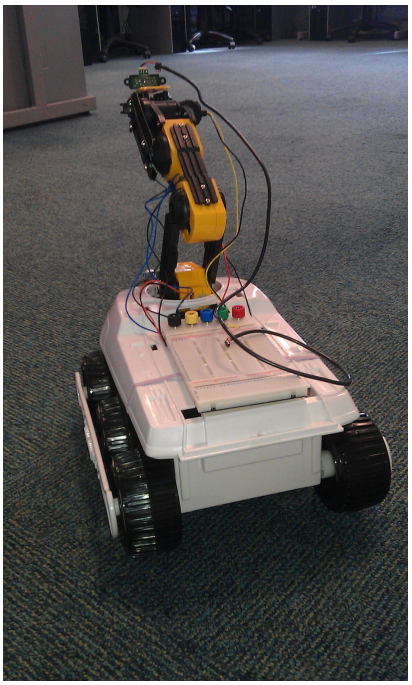
The robot arm base should now be quite a good fit inside the Big Trak.



Some extra work is still needed to make sure the arm will be stable. At the moment for building and testing I am just using tape and rubber bands but in due course I will use something stronger to secure it.

I may go with duck tape for this. Gaffer tape is a favourite "bodging tool" of mine because it is strong but allows you to easily remove it if needed, however there are numerous other methods that you could use with a bit of lateral thinking.

The final product as you can see is a finished skutter body.

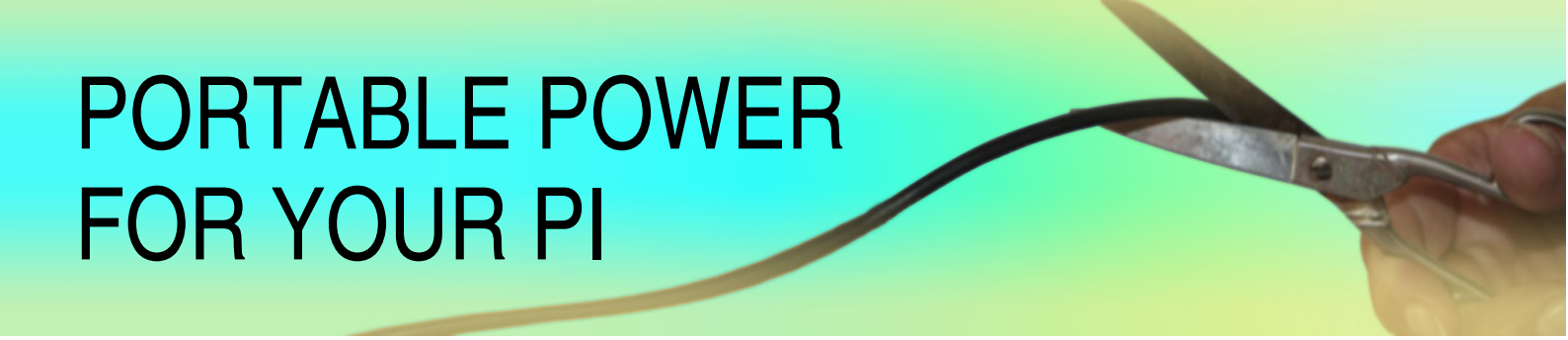


Next Time...

The next stages are to get involved with the electronics and programming for making the motorised platform move under the control of the Raspberry Pi.

Article By Bodge N Hackitt

PORTABLE POWER FOR YOUR PI



Untether your Raspberry Pi with a portable power supply.

There are plenty of interesting projects that require your Pi to be untethered from its mains power supply, or require a separate 5V supply that is capable of providing more power than is available from the Pi's GPIO pins – here's one way to do this.

The main requirement is to provide suitable regulation at 5V, with sufficient current capacity for your project. Although I initially looked at the cheap and popular 7805 regulator, I rejected it because it's not very efficient – a major consideration for a battery-powered supply – and it's only capable of handling 1 Amp – as the Pi uses up to 700mA, that doesn't leave much spare for driving anything else.

I eventually settled for the LM2576T-5.0 switching step-down voltage regulator – this device is much more efficient than the 7805, and is capable of handling up to 3 Amps. It will take any input voltage from 7 – 40V DC, giving you a wide choice of battery pack - I'm using 8 x 1.2v NiMH batteries, which give a 9.6V supply voltage, but you could use a 12v lead-acid battery if that was more suitable for you. Apart from the regulator chip itself, only 4 other components are needed – 2 capacitors, a choke and a Shottky diode.

Here are the RS part numbers:

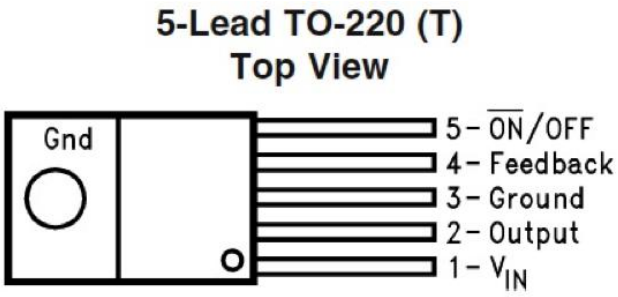
1 x LM2576T-5.0	460-477
1 x 100uF 25v electrolytic capacitor	684-1942
1 x 1000uf 25v electrolytic capacitor	684-1951
1 x 100uH choke (Min 3A Rating)	228-416
1 x Shottky Diode 40V 3A	714-6819
1 x Stripboard	206-5879
1 x Roll of Insulation Tape	513-553
1 x Heatsink (optional, see note opposite)	189-9306

The data sheet for the regulator is here:

http://www.ti.com/lit/ds/symlink/lm2576.pdf

where you will also find the circuit details as shown in diagram, opposite.

And the connections on the regulator chip are:



The circuit is easily built on a piece of stripboard and no track cuts are required; an indicator LED can be added – use a 200 ohm series resistor - and I would recommend 3.25A fuses in series with the positive input and output.Take care to get the 2 capacitors and the diode the right way round.

This is what mine looks like:

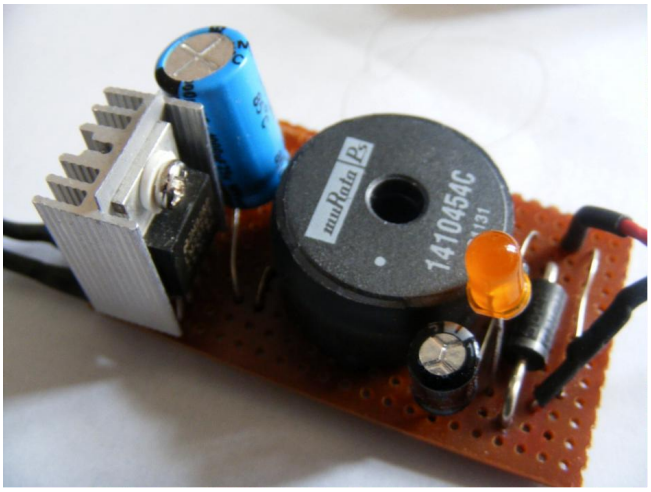
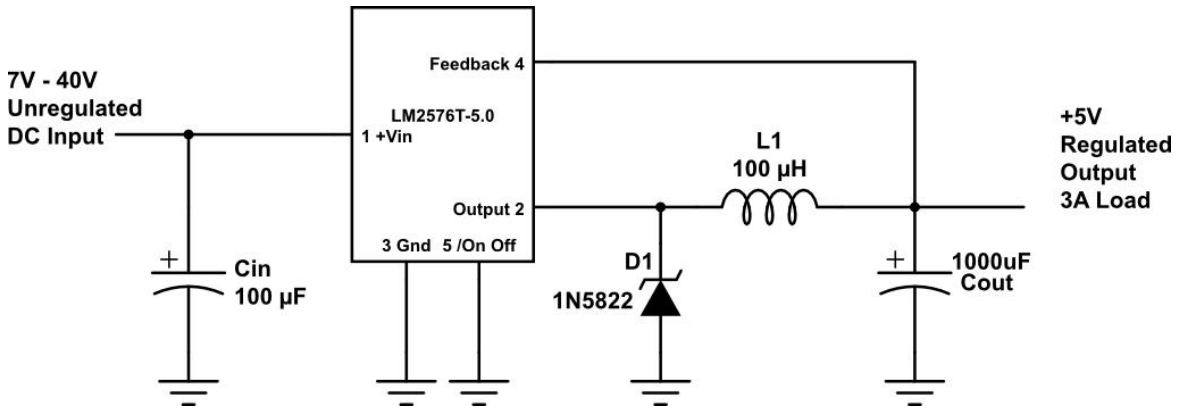
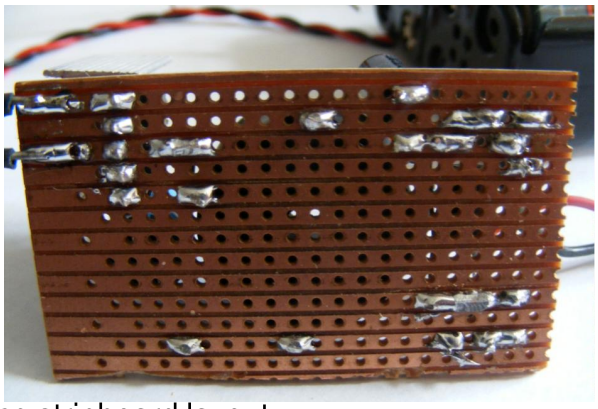


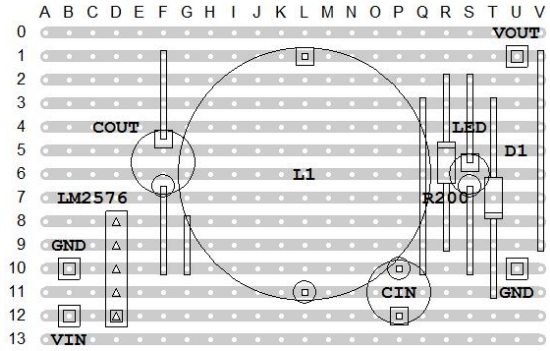
diagram 1:



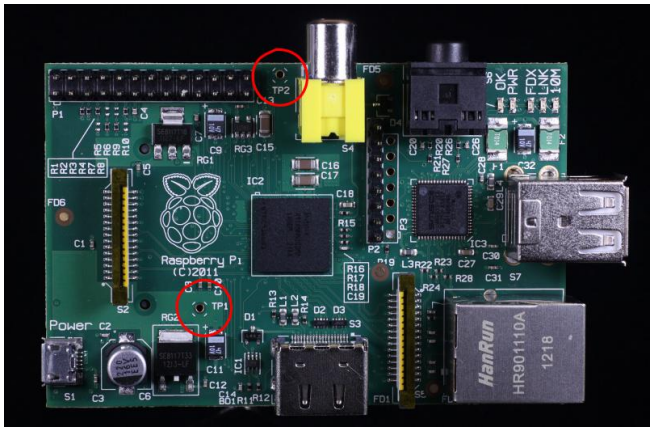
And from the underneath:



The stripboard layout:



Care should be taken to ensure that there are no short-circuits caused by solder bridging adjacent strips of the stripboard. After you have built and tested the supply, if you are going to connect it directly to the Pi, it should be connected via the 2 points marked TP1 and TP2, with the positive feed going to TP1.



The method I have shown does not utilize the micro-usb socket, this is a personal preference and to reduce costs. Alternatively, a cut-up micro-usb connector can be used, the common wire colours are as follows.

Red=5V
Black=Gnd
White=Data + (not used)
Green=Data - (not used)

Remember, the stripboard base is live, so keep it on an insulated surface, possibly use insulating tape (RS part number 513-553) or mount into plastic housing. If using housing, the unit will become warm so ventilation holes would be required to allow for cooling of the device. Remember don't try to connect a mains supply at the same time!

Depending on how much load you put on it, the regulator chip may get warm – if it's getting hot, a heatsink should be added, as I have done – there are plenty to choose from, just search on TO-220 heatsink (RS part number 189-9306).

Note that the metal tab on the regulator is connected to ground / pin 3, so either isolate the heat-sink (most are supplied with a suitable insulating kit) or keep it away from any contact with the other parts of the circuit.

Also note this regulator doesn't mean that you can take more power from the Pi's GPIO pins, any project load should be connected directly to the regulator output.

Article by John Ellerington

Star Letter: An FET Buffer Stage for GPIO Access

In response to the *In Control* article from Issue 4, Clive Tombs shares his own example of connecting to GPIO pins.

Introduction

Following on from the issue 4 article on transistors, I would like to describe my use of the 2N7000 Enhancement FET. I used this device only because I had some on hand from previous projects. Other types could be better suited as I will explain later.

Their use provides some interesting behaviours to buffer circuits which may prove beneficial in some applications.

The data sheet can be found here:
http://pdf1.alldatasheet.com/datasheet-pdf/view/2842/MOTOROLA/2N7000.html

Now, the FET's Gate is, in simplistic terms, insulated from the Source and Drain connections. Only the voltage relative to the Source (Vgs) is important. Once again I state in simplistic terms. Even if the GPIO pin is configured as an INPUT with the Pi's own Pull Up or Down resistors active, the FET will change state due to the extremely high input impedance of the FET.

From the data-sheet it can been seen that at around Vgs of 2.5v at room temperature the device starts to conduct. By 3.3v it can certainly operate an LED or small relay. As I stated above other FETs may be more suitable in their Vgs characteristics.

Now consider the following application: Test all inputs at start-up. Very simple code can be written to test all used inputs at start-up. By pulling the inputs up then down and testing for the condition in software and visually for an LED flash one can verify both the wiring and the buffer FET. This may seem trivial, but if the LED were replaced with the start circuit for some equipment which must be started in a

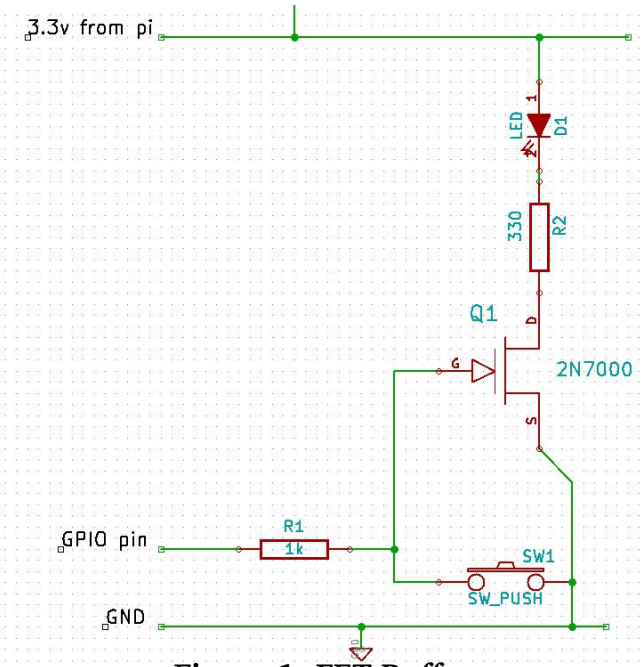


Figure 1: FET Buffer

correct sequence, this code would eliminate the FET as a source of error. As a maintenance engineer I like diagnostics to make my life easier!

It also has the advantage that one GPIO can be used for both input and output with, in Fig 1's case, a visual indication of button press too.

This is my first ever stab at a Python script. It is bound to be very inelegant, but it just about does what we need. It has been tested in Python 3 only. Try running it with a finger on the button to simulate an input being stuck.

Of course one could arrange the switch to pull the input up. That way the LED would not be on all the time. Script adjustments will be necessary.

With a change in resistor values the FET

status can remain unchanged if the button is pressed when the GPIO is set as output.
Eg: if R1 is 330Ω and the switch is connected through about 4k7Ω the Vgs will still be in excess of 3.0v with the button pressed if GPIO pin is output set high.

2N7000s are available for 10p each. Other, superb devices are now available. Some like the 2SK4043LS can switch pulses of 80A with as little as 2.5v Vgs. A single transistor could never do that as driven by the PI. And the 2SK3018, a surface mount device designed for small Vgs conditions like here in the PI.

```
#input test with visual indication
import RPi.GPIO as GPIO
import time

# change to BCM GPIO numbering
GPIO.setmode(GPIO.BCM)

# (pull_up_down be PUD_OFF, PUD_UP or
# PUD_DOWN, default PUD_OFF)
GPIO.setup(4, GPIO.IN,
pull_up_down=GPIO.PUD_UP)

# test for pin able to go high
if GPIO.input(4):
    print ('Input True Good')
    time.sleep(0.2)
    GPIO.setup(4, GPIO.IN,
         pull_up_down=GPIO.PUD_DOWN)
else:
    print ('Fault Input - pin4')
    time.sleep(1)
    quit()

# test for input able to go low
if GPIO.input(4):
    print ('Faulty Input - pin4')
    time.sleep(1)
    quit()
else:
    print ('Input False Good')
    time.sleep(1)

# if it gets to here, inputs' states
# are both achievable
print ('Inputs tested Good')

# commence the button demo
print ('Press the Button')

while True:
    # set pin high and
    # wait for button press
    GPIO.setup(4, GPIO.IN,
            pull_up_down=GPIO.PUD_UP)

    # button pressed
    if not GPIO.input(4):
        print ('button pressed')
        time.sleep(1)

        # button released
        if GPIO.input(4):
            print ('button released')
            flash = 20
            GPIO.setup(4, GPIO.OUT)

            # flash until
            while flash > 0:
                GPIO.output(4, True)
                time.sleep(0.1)
                GPIO.output(4, False)
                time.sleep(0.1)
                flash -= 1

            print ('press button')
```

There is a lot to be said for the FET in this application.

Clive Tombs

Editors Note

We love hearing from our readers. If you have a comment on an article or a cool Raspberry Pi related tip to share, please send it to us and we'll try to get it in an upcoming issue.

DID YOU KNOW?

The "In Control" series in issues 2, 3 and 4 is a great place to start and learn how to use the GPIO. If you have not started yet but want to have a go, there have been some updates to the RasPi GPIO Python library that you need to know before starting.

1) The RasPi GPIO library can now be easily installed with:

```
$sudo apt-get install python-rpi.gpio
```

or

```
$sudo apt-get install python3-rpi.gpio
```

2) Add the following line to each program:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
```

Pumpkin Pi

A little project to provide some Halloween fun. Wire up a pumpkin with glowing eyes and a motion dector!

The plan is to fit a Halloween pumpkin with a pair of RGB eyes and a nose made from a PIR motion detection sensor. Software will detect movement in the room (or street!) and flicker the eyes. Optionally we can hook up a set of amplified speakers to play some spooky music/effects!

The project can be soldered together relatively easily, or made on a breadboard if you have a selection of jumpers wires, and some insulating tape, heat-shrink, or if you're really stuck, something like BluTak!

Halloween is an old festival or celebration which has been adopted by many cultures and religions over the world. Its roots may originate in acknowledging that autumn is ending and we should take stock for winter, or to remember the passing of souls, or a time to hide from ghouls or the souls of enemies (hence the masks and lanterns!) For us, it'll be a bit of fun, some hot soup and a fancy Raspberry Pi powered lantern!

Ingredients

Like all good pies we need some ingredients:

One Halloween Pumpkin

A big orange pumpkin is best. In Scotland (Where I'm from), you should traditionally use a large yellow neep and make a "tumshie heed". Some English counties may traditionally use a mangelwurzel and if you can get one of suitable size that may be appropriate.

While a fun idea, it's probably not really practical to fit the Raspberry Pi inside the Pumpkin - they are somewhat damp inside, however if it's big enough and you're careful you'll be fine. You can line the base of the Pumpkin with something like bubble wrap, or if you have your Pi inside a case like the Adafruit one with the top removed, or any other case that allows access to the GPIO connector. Just remember to not get the Pi wet!

Electronics

2 x RGB LEDs - I'm using the 276-028 LEDs from Tandy. Any common cathode RGB LEDs will work.

http://www.tandyonline.co.uk/5mm-full-color-rgb-led-common-cathode.html

1 x PIR Sensor - I'm using the 276-135 sensor from Tandy.

http://www.tandyonline.co.uk/pir-motion-sensor-module.html

Note the connector on the PIR sensor - if you're not handy with a soldering iron, then you can use standard breadboard male to male connectors. Black is 0v, red is +5v and brown is our output wire.

PIR means Passive Infra-Red. PIR sensors work by sensing the infra red (heat) in the field of view and "remembering" the pattern. If it changes significantly, then the sensor trips.

They do take a few seconds to initialise, but this is described in the software. They are used in appliances such as burglar alarms and automatic outdoor lights.

Resistors - 4 x 100 Ohm and 2 x 150 Ohm. If you don't have any, I'd suggest buying one of the "starter-packs" of resistors - e.g. a "book" with various values. You'll probably only ever use 10% of them, but it's an easy way to start.

Colour codes for 3-bar resistors:

100 Ohm: Brown, Black, Brown
150 Ohm: Brown, Green, Brown

These are relatively safe values, but the LED may be a little dim, so they can be reduced to 82 Ohm and 100 Ohms respectively.

Construction

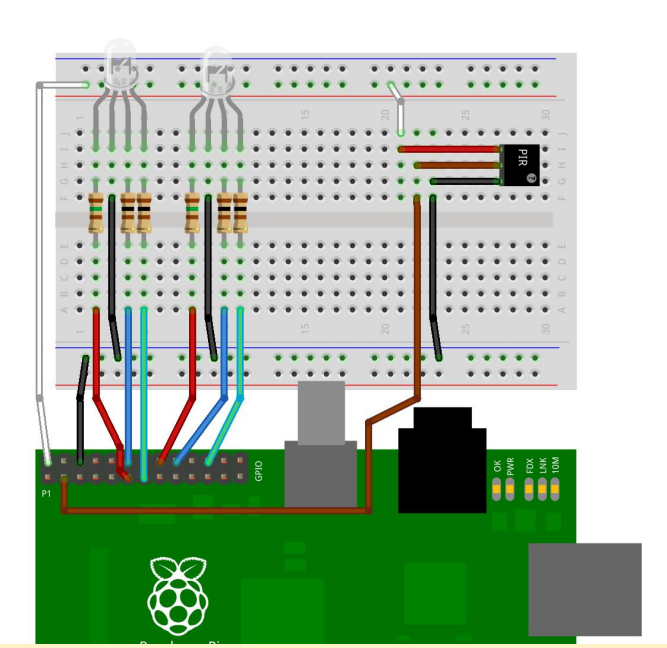
Fairly straight forward. The first thing to do is identify the colours on the LED - Each RGB LED has 4 Legs, with one being longer than the other 3. The longer one is the common cathode, and this is connected to 0v. You can then connect each in-turn to +3.3v via the 150 ohm resistor to identify the colours, however you should end up with:

Red, Cathode (Long Pin), Blue, Green

If you're handy with a soldering iron, what I'd suggest is soldering the resistors directly onto the LEDs, then taking some hook-up wire back to the Pi. If you use a female to female jumper wire, then you can plug it directly into the Pi, however you may want to use a small breadboard as you have 3 devices to connect to the 0v line; although you could simply solder them all together.

I like to use heat-shrink tubing to keep everything separate - it also adds a little mechanical strength the the joints too, but electrical insulating tape will do if you're careful.

We're connecting this to one of the I2C pins on the Pi. Note that although this device is operating at 5v, the output pin is "open collector". This means that there is normally no voltage present on the pin but when the sensor trips, it acts like a switch, connecting the pin to the 0v line. You can simulate the sensor with a simple switch connecting the pin on the Pi to the 0v line. The I2C connectors on the Pi already have on-board 1800 ohm pull-up resistors, making them ideal for this purpose.



See the attached breadboard diagram. Even if not using a breadboard and soldering everything, this is still a good reference to let you see where everything goes.

Carving

Carving your pumpkin... Please be as adventurous as you like, but remember - we'll be putting the LEDs in the eyes and the PIR sensor on it's nose.

Here is a suggestion: With a sharp kitchen knife, cut a ring round the top, but angle the cut towards the centre of the pumpkin. That way, you can remove the lid, hollow it out and put the lid back on and it (hopefully!) won't fall into the excavated pumpkin. You may wish to use zig-zag cuts too - which you can subsequently highlight with a black maker pen to emphasise that sawn-off head look...

Once you have the top off you can scoop out the seeds. Dry these in a low oven for a few hours then feed them to your budgie, hamster, etc.

Scoop out the flesh and put in a large oven-proof dish. Sprinkle with a little olive oil, add a little salt and pepper and roast in a hot oven

(200C / gas mark 7) for about half an hour until just starting to take some colour. Remove from the oven, put in a pan, mash, add in a small pot of single cream and half a vegetable stock cube, blitz or mash, bring to the boil and immediately remove from the heat and serve in a mug with a large piece of crusty bread...

Note: If you're a little bit unsure about sharp knives and hot cookers please do get someone to help you!

Testing

Once assembled, we need to test and for this we need our Raspberry Pi with the LEDs and PIR sensor attached.

Get wiringPi code:

At the command-prompt type the following:

```
cd ~
git clone git://git.drogon.net/wiringPi
```

If this command fails, it means that git is not installed. Install git with:

```
sudo apt-get install git-core
```

and re-run the git command above.Next:

```
cd wiringPi
./build
```

This will build and install wiringPi for you. It may take a minute or two.

The GPIO command.

The `gpio` command is a utility that's part of wiringPi which allows you to manipulate the GPIO pins from the command line.

Try this:

```
gpio mode 0 out
```

That tells the Pi to make pin 0 an output, then:

```
gpio write 0 1
```

This writes 1 (ie. on) to pin 0.

If you've wired it all up correctly, the first LED should now be lit up Red.

Now for more testing run these commands using the default BASH shell:

```
for i in 0 1 2 4 5 6; do gpio mode $i
out; done
gpio write 0 1
gpio write 6 1
```

You should now have a red LED and a green LED. If not, then go back and check your connections. If you get the wrong colours, then just swap the connections, or work out the right pins to use and adjust your program as required (it's sometimes easier to change software than hardware!)

Test the LEDs

Firstly a few handy commands:

This will turn off the first LED:

```
for i in 0 1 2; do gpio write $i 0; done
```

and this will turn off the 2nd LED:

```
for i in 4 5 6; do gpio write $i 0; done
```

Remember you can use the up-arrow keys at the command-line to repeat previous commands.

Then red:

```
  gpio write 0 1
  gpio write 4 1
```

blue:

```
  gpio write 1 1
  gpio write 5 1
```

green:

```
  gpio write 2 1
  gpio write 6 1
```

Remember to run the 'for' sequence above each time to turn them off.

Play with the gpio commands above to see what colour combinations you can create. You ought to be able to get 8 basic colours (Black, Red, Green, Blue, White, Cyan, Yellow and

Magenta). In software, we will use PWM to give us the possibility of generating 1 million different colours!

Testing the sensor

Run this command:

```
gpio mode 8 in
```

The I2c pin we're using is pin number 8, so we set it to an input. This pin is one of the I2C pins on the Pi and has an on-board 1800 ohm resistor pulling the pin to +3.3 volts, so with nothing connected to the pin, it's going to read a logic 1, or high.

```
gpio read 8
```

and it should return 1.

However it may return 0 - and that's because the sensor has tripped.. So point the sensor away from you and wait for it to read 0. You can run a loop as follows:

```
while true; do gpio read 8; done
```

and then move in front of the sensor when it should read 0.

Software

Once we have that working, we need to write some software (or download it from the 'net!). You can clone the code with git or download the files from here;

http://git.drogon.net/?p=halloweenPi

The software here is designed to control the two RGB LEDs triggered by the PIR. There are 3 files of code and a makefile. The makefile is a set of rules which gives instructions about building the code.

To compile and run, type:

```
make
sudo ./halloween
```

If you do not have wiringPi installed, then you will get failures, so install wiringPi:

```
pushd /tmp
git clone git://git.drogon.net/wiringPi
cd wiringPi
./build
popd
```

If you get an error when running the git command, then:

```
sudo apt-get install git-core
```

The three program files -

`halloween.c`: This is the main program - it initialises the sensor and LEDs and performs actions when the sensor is triggered.

`ledControl.c`: This sets up the RGB LEDs and provides a function to program any RGB value to either LED.

`ledPatterns.c`: This is code to create many different patterns on the LEDs.

Now when you run sudo ./halloween you will have a fun Pi powered pumpkin.

If you want to customize the patterns, `ledPatterns.c`, is probably the file you want to play with. You can change one of the existing functions, or add new functions into it.

To add in a new pattern write a new function. Try to use the same style as the existing ones and have it run for a set period of time. Then add an entry into `ledPatterns.h` and call the function from the main `halloween.c` program.

You can watch a short video of the pumpkin at:
www.youtube.com/watch?v=jg8ugFCdJ7I

Gordon Henderson

Editors note; Dear reader, you may have noticed the text refers to carving a real pumpkin but the photo shows a plastic one. Sadly Gordon was unable to find a real pumpkin in stores by the time this issue went to press. Your editor was duly shocked, for in the US pumpkins have been available for weeks and Christmas lights have been sold since June.

Camera Pi

An interview with David Hunt

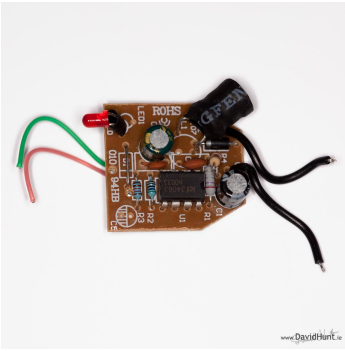
Take a look at the photo to the left and you will see a perfectly normal camera that has been given some extra functionality by the inclusion of a Raspberry Pi in the grip. This is Camera Pi.

Q: When you saw the Raspberry Pi did you immediately think that it was ideal, or did you order one for other projects and then consider the camera grip?

Embedding a Raspberry Pi inside a camera grip was a bit of a Eureka moment. I had looked at the Beagleboard and similar over the last couple of years but the price point was too high each time. The Pi is a perfect combination of price to power ratio, size and features. I saw an announcement on Engadget and decided to get one.

Now, with Camera Pi, I can automatically download photos wirelessly from the camera and preview on an iPad, and remotely issue commands over SSH using gphoto2 to take photos.

Q: How did you discover that the DC-DC circuitry inside an iPhone charger was what you needed to connect the battery?



Someone on the Raspberry Pi Forums mentioned this, and by chance I had a broken iPhone charger in the recycling box. Originally I connected 5V from AA batteries to the DC-DC converter but only got 4V out. I then had the idea to use a 7.2V Canon camera battery and got 5.08V, perfect for powering the Pi. I then connected it to the Pi, and it booted up fine. Both Ethernet and USB were working. I got very excited at this!

I cut the original battery enclosure from the grip in half and used that to mount the battery. This was then located in a cut-out in the grip to the right of the Pi, although it does push on the GPIO a bit. I covered the pins with shrink wrap to ensure they did not touch.

Q: How do you compare your solution to the Eye-Fi SD card (www.eye.fi)?

I have an Eye-Fi card but as my Canon 5D uses Compact Flash an adaptor is needed. I only had limited success with it. I'm not blaming Eye-Fi as they do state this on their website, but it did mean I had to look for an alternative wireless solution to transfer photos. The Pi also gives me a lot of additional functionality by enabling direct control of the camera.

Q: Did you follow a plan for this project from the outset or use a more organic approach adding functionality as you went?

It is definitely an organic project, although I do have a firm idea of what I want to achieve. When I saw the Pi I thought it has lots of possibilities with the hardware ports on board. I had watched "The Mountain" by TSO Photography (www.tsophotography.tumblr.com) and it was inspiring stuff: his camera was connected to a stepper motor and could slowly pan as it took images. This kind of control is a killer feature and it is one of several possible directions Camera Pi may go in as the GPIO possibilities are endless.

Q: In your blog you detail having issues with gphoto2. How did you resolve this problem?

When I first connected the camera via USB gphoto2 would lock up after each image giving an error. I parked the project for a month

because of this. When the next Raspian Release Candidate was released I tried again. In the end I found some C code on the net to reset USB after taking each image, which I wrapped in a shell script. With this in place and after a few hours in Perl I had a working proof of concept.

Q: What other problems did you encounter?

The proof of concept (not held in the camera grip) was fairly straightforward and worked well. To be honest I was amazed that the software packages I need were already available in the repository and easily installed via apt-get, for example: I found a package for writable NTFS support straight away when I went looking.

Modifying the grip was a challenge and the hardest part of the build, taking in the region of 40 hours to carefully cut and fit the components as the grip is made of tough plastic that is difficult to cut.

Q: What do you think of the potential to extend the system via GPIO?

The mind boggles at the possibilities of talking to other equipment over GPIO. For example using the Pi to drive a motorised telescope mount is something I would like to see someone do. To-date I have hooked up some transistors and resistors to the GPIO pins and can wake the camera if it enters sleep mode by emulating a half-press of the shutter release, plus I have connected a broken shutter release cable to GPIO to take photos manually.

Q: What has been the response to Camera Pi?

I have had a lot of incredibly positive feedback through my blog from both amateur and professional photographers: one asked where they can buy a Camera Pi setup. Other enthusiasts are building them and there seems to be a lot of interest. Most people seem to be aware that they can tether their camera to their laptop but had not thought about building their own computer-in-grip solution using gphoto2.

Q: What has been the reaction to the Raspberry Pi amongst your peers?

In the office the Pi has people really excited and we are using it as part of our mentoring program enabling seniors to train up others and encourage creativity. The company is actively encouraging this which is excellent. One guy got up at 4am on the day of the release to get his order in!

Q: Do you think the Pi is an enabling device that brings an affordable development platform into commercial reach of almost anyone?

I am used to working with limited device resources of just 64-128MB RAM and once created an 8MB Linux install on Compact Flash with only a 700KB kernel. With the relatively decent CPU and RAM in the Pi, and configured with minimal RAM dedicated to the GPU (I never start the X GUI) it is perfect for my needs. I am amazed at what the Foundation has managed to achieve with the Pi's hardware for the cost and the low price definitely helps in this regard.

Q: Any final thoughts?

The Raspberry Pi has an emotional factor as it brings back a lot of 80s nostalgia as well as being an excellent, usable machine. It encourages use as a programming tool like the BBC Micro where one would type in games by hand from computer magazine listings. Eben is dead right that there has been a gap of 15+ years in teaching proper computer science in schools, focussing instead on word processing and spreadsheets. Unfortunately, many kids often don't know the fundamentals of programming when they reach university or industry. The Raspberry Pi gives them this opportunity.

David Hunt has worked on embedded systems for about 20 years, programming in C on a variety of devices. Today he works in Ireland for an embedded software company. He is a keen amateur photographer and has won a number of international awards.

Our Raspberry Pi Summer

A School Teacher and his son discover coding like it's the 80s again!

When I first told my wife that I had bought a Raspberry Pi and it was going to be deliveredin 10 weeks she thought I had gone mad.

After some explanation that this was in fact another computer for home I was met with even more disbelief and comments about where would another computer go - we haven't got room.

After exactly 8 weeks the Pi arrived and my seven year old son and I were both really excited by the credit card sized circuit we unboxed.

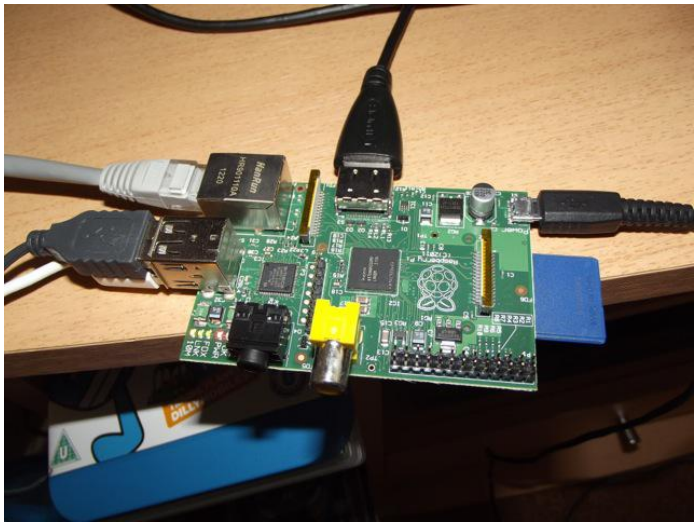
As a teacher and a bit of a geek I was excited at the prospect of giving my son a chance to write some simple games and a project to work on over the summer holidays. Being a child of the 80s I spent many a happy hour writing simple and eventually more complicatedgames in BASIC on my Acorn Electron and wondered if Philip would catch the programming bug.

Following the initial excitement of booting a LINUX machine (thanks again for the many happy memories from Uni) we started up SCRATCH and began to explore. With very few instructions we were in at the deep end.

Within an hour Philip had discovered how to add a piece of graphics (a Sprite) and make it say simple messages. Next came motion and control of the Sprite.

After a few more hours programming and experimenting with different ideas Philip's first game was written.

We decided to make a short video of the game and of using



Plugged in and ready to go!

Boxing our Raspberry Pi

Philip in action

the Pi and uploaded it to YouTube. We could't believe the response - within a few days we had passed 20,000 views and with loads of positive comments flowing in Philip was encouraged to write his second and third games.

Over the next few weeks two more games were written. Philip developed quickly his understanding of the different commands and operations in Scratch mostly through a process of trial and error. There were a number of times he asked me for help and then decided that his code was better and I was making it more complicated.

People have asked me many times what is the point of a Pi? As a teacher I can see amazing potential in this cheap little device that can be simply plugged into a TV and used by anyone. I don't think that my son will become a computer programmer or a game designer - but this amazing summer with the Pi has shown us both what can be achieved with determination and perseverance.

As someone who has recently taught GCSE ICT and had students designing Business Cards or one of the other (banal) activities which make up the GCSE course I thoroughly believe in the call to change teaching ICT back to basics such as programming. The skills that my son has been developing this summer through programming will be invaluable as he moves into Junior school and then beyond. We live in a culture of instant gratification and success and programming has never been like this. Things often don't work and to succeed we need perseverance and a willingness to look at the problem from a different angle. As a parent and a teacher seeing these skills developing is really exciting, a huge change from the "I can't do it so I give up".

Here's to another slice of Pi.

The Game

To start the game:

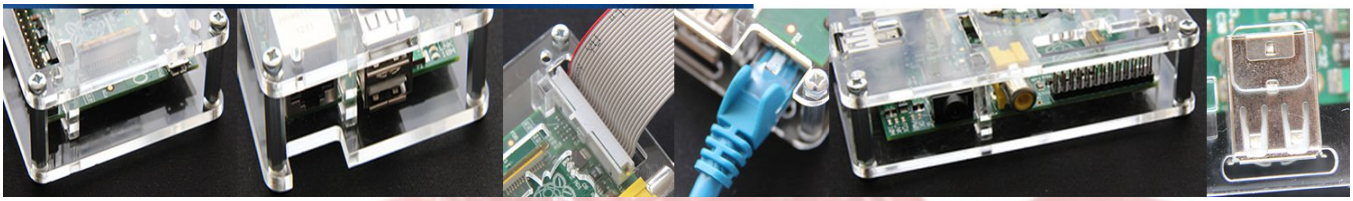
The frog:

The starfish:

The ball:

Spencer Organ is a seasoned Secondary School teacher from the West Midlands. In his spare time, he runs a tech website (home.uktechreviews.com). He is passionate about bringing creativity into teaching and learning and about uisng ICT and multimedia as facilitators for this.

OCTOBER COMPETITION



Once again The MagPi and PC Supplies Limited are proud to announce yet another chance to win some fantastic R-Pi goodies!

This month there will be FIVE prizes!

Each winner will receive a Limited Edition LCD mount by PCSL.

For a chance to take part in this month's competition visit:

http://www.pcslshop.com/info/magpi

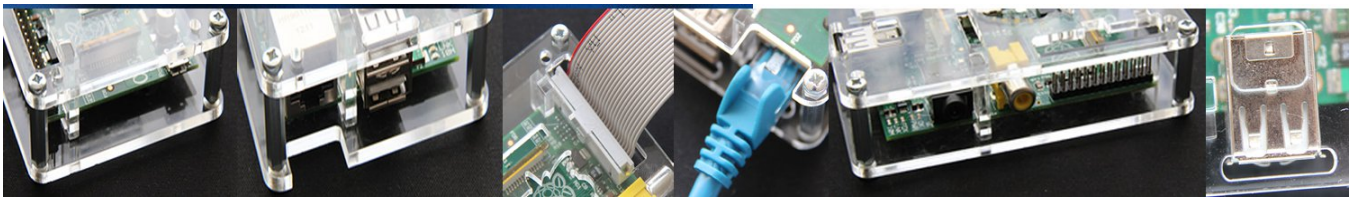
Closing date is 20th October 2012. Winner will be notified in next month's magazine and by email. Good luck!

To see the large range of PCSL brand Raspberry Pi accessories visit
http://www.pcslshop.com

Last Month's Winners!

Winner of the PCSL accessories bundle is **Paul Hargrove (High Wycombe, UK)**
Winners of the PCSL engraved case are **Guido Malfatto (Turin, Italy)** and **Jason Ellmers (Plymouth, UK)**

Congratulations. We will be emailing you soon with details of how to claim all of those fantastic goodies!



Baby steps in a...

BIG WORLD

Ada, a language for everyone

By Luke A. Guest

Introduction

In this article I will introduce the Ada programming language, its history, what it can be used for and also, how you can use it with your Raspberry Pi computer. This article has a number of coloured side-boxes which provide extra information as I introduce Ada, you should read these so you gain a deeper understanding of the language. So let's get started...

You probably know what other programming languages look like, most of these are not very readable and seem to be a jumble of words (sometimes, not even words) and weird symbols. All languages provide various symbols, but Ada was designed so that its programs were easier to read by other people, even many years after they were originally written.

Unlike Python or Ruby, Ada is a compiled language, much like C. This means we have to pass Ada programs (source) through something called a compiler which converts this source into machine language so that it can be run directly on the computer.

In this article, I will be using a Debian based system image, so Debian Squeeze or Raspbian will be fine. Debian provides an Ada compiler, if you are using a different Linux distribution such as Fedora, you will have to check their package managers for the compiler. The compiler is called GNAT so you know what to look for.

Before we get started, I will assume you are running a graphical environment, such as LXDE (after typing startx or booting directly into it, see MagPI issue 3 page 3 for more info), even though in this article we will be using the console only to start.

I have tested the examples in this text by logging into a remote shell on my Raspberry Pi.

Getting started

Before we can start typing in Ada code and running it on the computer, we need to install a few tools. We will need the terminal, a compiler and an editor, start LXTerminal and type in the following commands to install the tools we need:

```
$ sudo apt-get install gnat
```

You will be asked for your password, enter this, when APT asks you if you want to continue press the return key at this point to let APT install its packages.

Then create a directory for this article's source, we will need to change to this directory as we will be running commands directly inside it:

```
$ mkdir -p \
  $HOME/src/baby_steps/lesson1
$ cd $HOME/src/baby_steps/lesson1
```

Let us create a new Ada source file in this window by typing the following in the shell:

```
$ nano -w hello.adb
```

Inside nano, type in the program in Listing 1 (without the line numbers), typing **Ctrl+O** to save the program.

Now inside LXTerminal, create a new terminal tab with **Ctrl+Shift+T**, this will automatically make the new shell's current directory be the same one we are using for this program. We can now compile this program with the following shell command:

```
$ gnatmake hello
```

The program, gnatmake, is the front end to the

```
1   with Ada.Text_IO;
2   use Ada.Text_IO;
3
4   --  Print a message out to the screen.
5   procedure Hello is
6   begin
7      Put_Line ("Hello, from Ada.");
8   end Hello;
```

Listing 1: hello.adb

Line 4: starts with 2 hyphens (or dashes), this is a comment. Anything placed after the hyphens is ignored by the compiler up to the end of the line.

In Ada, a main program can be called almost anything you like, but the filename must match this name (in lower case letters) and have ".adb" appended to the name. In our example, our main program is called "Hello" (lines 5 and 8) and its filename is "hello.adb," the adb means "Ada Body."

Line 5 states our program is a procedure, this is 1 type of Ada's subprograms, the other being a function. Both types of subprogram are used for specific reasons, a procedure does not return any values to the caller whereas a function does. "main" subprograms are procedures.

Line 7 is a call to a subprogram found within a package called Ada.Text_IO, lines 1 and 2. The Put_Line procedure prints whatever is in the string (between double quotes ") to the console.

Line 1 states that we wish to use the facilities provided by the Ada.Text_IO package, and Line 2 tells the compiler we don't want to have to write the subprogram call in full, in other words, if line 2 didn't exist we would have had to type in **`Ada.Text_IO.Put_Line`**. There are reasons to do this, but we will cover this another time.

As all subprograms must have a beginning (line 6), they must also have an ending, line 8. In Ada, all subprograms must state what it is ending by specifying its name again. Ada enforces this as this is an aid to being more readable.

Each Ada program is made up of a number of statements, a statement ends with a semi-colon (;). Every statement you write must have a semi-colon otherwise the program will not compile.

In Ada, there are some words which are defined by the language, these are called keywords, you cannot use these keyword names for your own types, variables or subprograms.

Ada compiler, as you will see when you compile the program, it calls other programs, including gcc, gnatbind and gnatlink.

You can now run the compiled program with the following command:

```
$ ./hello
```

The result is that the program prints what is in the double quotes in the program to the shell, in other words, "Hello, from Ada." You have just written your first Ada program!

Simple types and maths

Unlike other languages, such as C, Ada is a what is called a strongly typed language. What is this type thing? Well, every value in Ada has a type, for example, the number 10 is an integer number, so if we wanted to store values of numbers we would define a variable of type integer, see the sidebar for more information on types. Exit nano using **`CTRL+X`** and create a new file called simple_types.adb and type in the source from Listing 2.

Using what you learnt from the previous example, type in and save this code, then compile it with gnatmake and finally, run the program in the terminal and see what happens.

```
1    with Ada.Text_IO;
2    use Ada.Text_IO;
3
4    procedure Simple_Types is
5       X      :         Integer := 10;
6       Y      : constant Integer := 20;
7       Result :         Integer := 0;
8    begin
9       Result := X + Y;
10
11      Put_Line ("Result = " & Integer'Image (Result));
12   end Simple_Types;
```

Listing 2: simple_types.adb

So, we've already seen lines 1, 2, 4, 8 and 12. So what's new? We have not seen the variable and constant definitions before, these are on lines 5, 6 and 7. Here we define 2 variables, **`X`** and Result which are integer types and 1 constant, **`Y`**, which is also an integer type.

The difference between a variable and a constant is that you can assign a value to a variable within the program, see line 9, where we assign **`X + Y`** to **`Result`**. In Ada the symbol **`:=`** means assign or give the variable on the left the value of what is on the right, in our case this is 10 + 20 which makes Result equal 30. To make something constant we use the keyword "constant" before the type name (integer).

So what happens if you try to assign a value to **`Y`** in the program source? Try this yourself and see what happens when you compile the program. It will not compile, because you cannot assign to a constant once it has already been assigned to.

On line 11, there is something strange **`Integer'Image`**. What is this? This is an attribute of Integer. See the sidebar entitled "Cool features: Attributes" for more on these.

Also, on line 11, we have another symbol, **`&`**, which means string concatenation. This means we can "add" strings together, the left side of & is added to the right side of **`&`** and then **`Put_Line`** prints everything to the screen.

Exercises

1. Change line 9 to each of the following, compile and run, what is the value of Result?

```
a) X – Y
b) Y - X
c) X * Y
d) X / Y
e) Y / X
```

2. Use this time to play around with various numbers, variables and constants and see what results you get in the console.

Cool features: Types

Types enable the compiler to make sure that only variables of the same type can be used together, for example, **`X := Y + 10`**; X, Y and 10 are all integers, if X was something else, say of type Boolean, this program would not make sense and it would not compile; the compiler would give you a very helpful message to find your error.

Numeric types

Along with Integer types there are two more types which are based on the Integer type called Natural and Positive types; these are subtype's of Integer in that they restrict the range of values allowed to be assigned to variables of these types.

Tutorial 4 - Bitwise operators and system commands.

Did you manage to solve last month's challenge problem? Here is the solution to compare with,

Challenge solution

```
#include <stdio.h>
#include <stdlib.h>
int newMask() {
  int mask = (double)rand()/RAND_MAX*254+1;
  return mask;
}

int main(int argc, char *argv[]) {
  int seed = 0xA3, mask = 0;
  char c;
  FILE *inputFile = 0, *outputFile = 0;

  srand(seed); /* Set the seed value. */

  /* Check the number of arguments */
  if(argc!=3) {
    printf(" Usage: %s <input file> <output file>\n",argv[0]);
    return 1; /* Report an error */
  }

  inputFile = fopen(argv[1],"r"); /* Open the input file. */
  if(!inputFile) return 2;

  outputFile = fopen(argv[2],"w"); /* Open the output file. */
  if(!outputFile) return 3;

  c = fgetc(inputFile); /* Get the first character. */

  /* Loop until end-of-file is reached. */
  while(c != EOF) {
    mask = newMask(); /* Get a new mask value. */
    printf("mask = %d\n",mask);
    c ^= mask; /* Exclusive-OR with the mask. */
    fputc(c,outputFile); /* Write to the output file. */
    c = fgetc(inputFile); /* Get another character. */
  }

  /* Close the files. */
  fclose(inputFile);
  fclose(outputFile);

  return 0;
}
```

The solution uses a new mask to encrypt each character. The numbers returned from `newMask` follow a series, which is repeatable for a given value of the input seed. Therefore, the encryption key is the random number seed.

Bitwise operators

The main bitwise operators are summarised in the table below.

Condition	Meaning	Condition	Meaning
a & b	a 'and' b	a >> n	right shift a by n
a | b	a 'or' b	a << n	left shift a by n
a ^ b	a 'exclusive or' b		

These operators are typically used with integer variable types or signal bytes stored in `char` variables. They act on the binary form of the number and are typically used for bit packing or testing packed bits. For example, if the status of several switches needs to be read, their input could be stored in one integer variable.

As revision of the second tutorial, the decimal values of each bit can be printed with the program below:

```
#include <stdio.h>
int main() {
  int bit = 0, i = 1;
  while(i>0) { /* Loop until the sign bit is set */
    printf(" pow(2,%2d) = %11d\n",bit,i);
    i = i<<1; /* Shift value of i left by one. */
    bit++; /* Increment the counter by one. */
  }
  return 0; /* Return success to the operating system. */
}
```

In this example, the value stored in the variable `i` is shifted one place to the left. The left shift operator has the effect of moving all of the bits in the variable `i` one place to the left. If a bit is shifted outside the memory allocation of the variable `i`, the bit is lost. In this case, `i` only contains one. Therefore, the action of the left shift operator is to move to the next power of two. When the bit in the variable `i` is moved into the sign bit the number becomes negative which causes the while loop to stop.

The & operator is very useful for testing if bits are set. This can be combined with the left or right shift operator to test every bit in a integer variable,

```
#include <stdio.h>
int main() {
  char str[33]; /* Declare a character array to hold the output. */
  int bit, i = 235643; /* Declare a number to convert to binary. */
  for(bit=31;bit>0;bit--) { /* Loop from left to right */
    if(((1<<bit) & i) == 0) str[31-bit] = '0'; /* False */
    else str[31-bit] = '1'; /* True */
  }
  str[32]='\0'; /* Add the string terminator */
  printf("%d (decimal) = %s (binary)\n", i, str);
  return 0; /* Return success to the operating system. */
}
```

In this example program, each character in the char array is set according to the binary value. Then to complete the string, the string terminator is added. Finally, the binary form of the integer number is printed.

System commands

It can be useful to be able to run shell commands or other programs without directly linking to an associated library. This can be accomplished with the `system` function,

```
#include <stdlib.h>
int main() {
  system("ls ./"); /* List the files in current directory. */
  return 0; /* Return success to the operating system. */
}
```

The `system` function evaluates the string argument passed to it as if it had been typed at the command line. The standard output from the command is not captured by the program and is instead printed on the screen.

The standard output from a system command or program can be captured using a pipe. Pipes follow the same syntax as regular file functions, allowing reading, write and bidirectional connections. For example, the contents of the current directory can be read into a program using,

```
#include <stdio.h>
int main() {
  int c;
  FILE *ptr = 0; /* Create a null FILE pointer */
  ptr = popen("ls ./", "r"); /* List the files in the directory and listen */
  if(!ptr) return 1; /* If the command fails return failure. */
  while((c=fgetc(ptr)) != EOF) { /* Read each character. */
    printf("%c",(char)c); /* Print the characters. */
  }
  pclose(ptr); /* Close the pipe */
  return 0; /* Return success to the operating system. */
}
```

In this case, each file name returned is available within the program.

Any command that can be typed at the command line can be executed using `system` or `popen`. Rather than just call simple shell functions, these command can be used to plot data using gnuplot,

```
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
  int x_min = 0, x_max = 4; /* Set the range for the plot. */
  char commandStr[100], systemCmd[200];
  if(argc < 2) {
    printf("Usage %s <function>\n", argv[0]); /* One argument is needed.*/
    return 1; /* Return error. */
  }
  /* Build the command in two steps to show what is going on. */
  sprintf(commandStr, "plot [x=%d:%d] %s(x)", x_min, x_max, argv[1]);

  /* Run the command so that gnuplot stays open. */
  sprintf(systemCmd,"echo \"%s\" | gnuplot --persist",commandStr);
  system(systemCmd); /* Tell gnuplot to plot it. */
  return 0; /* Return success to the operating system. */
}
```

Before trying this example, gnuplot should be installed by typing:

```
sudo apt-get gnuplot-x11
```

Then once the program has been compiled try, `./gplot sin` The `--persist` flag causes the gnuplot window to stay open after the program has finished. More information on the gnuplot program is available at, http://www.gnuplot.info/

Monitoring a LINUX system

There are several useful functions which are available under LINUX, but are not implemented in the same way on other operating systems. For example, the status of the memory can be retrieved using the `sysinfo`,

```
#include <stdio.h>
#include <sys/sysinfo.h>
int main() {
  struct sysinfo info; /* Create a sysinfo instance to hold the result. */
  sysinfo(&info); /* Get the system information */
  printf("Memory used = %d\n",info.totalram - info.freeram);
  return 0; /* Return success to the operating system. */
}
```

where the sys/sysinfo.h is available on LINUX, but not OSX or MS Windows. Before the system information can be retrieved, a struct variable of sysinfo type is created. This is not a simple variable, but contains several variables. The member variables of the struct are accessed using the "." operator. When sysinfo is called, the address of the struct variable of systinfo type is passed to the function. The function then writes the status into the member variables of the struct.

In the final example for this tutorial, gnuplot is used to plot the memory usage as a function of time:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/sysinfo.h>
int main() {
  int i, ramUsed;
  char gnuplotCmd[250], systemCmd[350];
  FILE *outPtr = 0;
  char fileName[50];
  sprintf(fileName,"data.txt"); /* The name of the output file. */
  struct sysinfo info; /* A sysinfo struct to hold the status. */
  outPtr = fopen(fileName,"w"); /* Open the output file. */
  if(!outPtr) return 1; /* If the output file cannot be opened return error */
  for(i=0;i<60;i++) {
    sysinfo(&info);  /* Get the system information */
    ramUsed = info.totalram - info.freeram;
    fprintf(outPtr,"%d %d\n", i, ramUsed); /* Write the ram used. */
    usleep(500000); /* Sleep for 1/2 a second. */
  }
  fclose(outPtr); /* Close the output file. */

  /* Now plot the data */
  sprintf(gnuplotCmd, "plot \'%s\'\n", fileName); /* Build the plot command. */

  /* Create the full command, including the pipe to gnuplot */
  sprintf(systemCmd,"echo \"%s\" | gnuplot --persist",gnuplotCmd);

  system(systemCmd); /* Execute the system command. */
  return 0; /* Return success to the system. */
}
```

where the sys/sysinfo.h header file is available on LINUX and the unistd.h header file is available on LINUX or OSX. The program writes the memory usage to an output file every half a second. Then gnuplot is run to plot the memory usage as a function of time.

Challenge problem

Modify the previous example program to write an output file using the return value of the command hostname to form a file name. Then plot the memory used and the system load while running one or more other programs. The loads[3] member variable of the sysinfo struct holds the, one, five and fifteen minute load averages. Try using,

```
fprintf(outPtr,"%d %f %d\n", i, ramUsed/10240.0, info.loads[0]);
```

to write the data file. Then plot the data using two strings,

```
sprintf(gnuplotCmdOne, "plot \'%s\' using 1:2 title \'%s\'", fileName, "Ram used");
sprintf(gnuplotCmdTwo, ", \'%s\' using 1:3 title \'%s\'\n", fileName, "Load");

/* Create the full command, including the pipe to gnuplot */
sprintf(systemCmd,"echo \"%s%s\" | gnuplot -persist",gnuplotCmdOne,gnuplotCmdTwo);
```

The solution to the problem will be given next time.

Article by W. H. Bell

THE SCRATCH PATCH

The Bubble Sort Algorithm

This month, I thought we'd do something a little different: an algorithm.

Algorithms are step-by-step instructions for doing a certain task. If you learned the "grid method" for multiplying numbers at school, then you are using an algorithm. If you follow the steps correctly, you will get the right answer.

When we write a computer program, we want an algorithm that is as fast as possible and that uses as little memory as possible. It also needs to produce the correct result, of course!

As usual, if you get stuck you can download the project from: http://scratch.mit.edu/forums/

My user name is "racypy".

The Bubble Sort is an algorithm for sorting lists of numbers.

It's actually not one of the most efficient sorting methods: it can be quite slow on a really jumbled list.

However it is very fast at sorting lists that are almost in the right order.

Here's the start of our program. It simply announces what the program is and then calls two processes - "Make_Array" and "Bubble_Sort".

Fact:

Algorithms get their name from a Persian mathematician:

Al-Khwārizmī (c.780 – c. 850).

Make_Array

This part of the program is the "Make_Array" procedure. First we use a loop to make a list (called "ordered") holding the numbers 1 - 15.

Then we makes random choices from this list, adding them to the new list "Array". We delete the ones that have been used, so that we only get each number once.

Bubble_Sort

Here we loop over the array for the same number of times as there are numbers in the list**.**

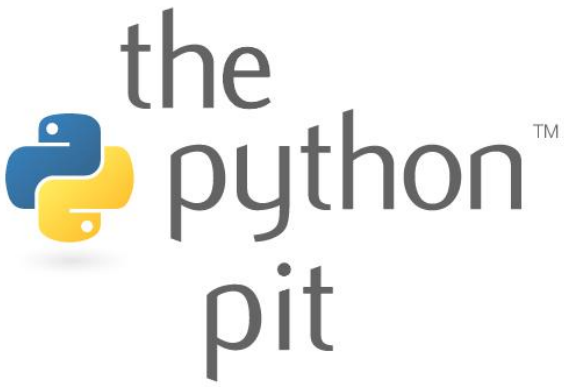
Nested within this loop is another in which we loop over the list, checking to see if any number is greater than the one next to it.

If it is greater, then the numbers get swapped over.

As it's running, you can see the numbers getting swapped until the list is fully sorted.

If you enjoyed this, you might want to adapt the program so that it sorts numbers that the user provides, instead of this randomised list.

Scratch On!



When programming, it is sometimes useful to be able to read and write to and from external text files. The first example shows you how to use python to create a html web page.

The second program displays fading titles pulling data from an external text file.

```
# HTML Writer

# By Jaseman - 16th September 2012

import os

# Creates a file and opens it for writing (w)
f = open('/home/pi/test.html', 'w')

# Write lines of code into the file
# Note: avoid using " quotations, use instead '
f.write("<html>"+"\n")
f.write("<head>"+"\n")
f.write("<title>A Webpage Created by Python</title>"+"\n")
f.write("</head>"+"\n")
f.write("<body bgcolor='#ffffdd'>"+"\n")
f.write("<font face='verdana' color='#000000'>"+"\n")
f.write("<center>"+"\n")
f.write("<h1>THE HEADING</h1><p>"+"\n")
f.write("<hr>"+"\n")
f.write("</center>"+"\n")
f.write("<h3>A Subheading</h3><p>"+"\n")
f.write("This is the text of the first paragraph.<p>"+"\n")
f.write("<hr>"+"\n")
f.write("<center>"+"\n")
f.write("<font size='2'>"+"\n")
f.write("<b><a href='mailto:editor@themagpi.com'>EMAIL</a></b><p>"+"\n")
f.write("<b><a href='http:www.themagpi.com'>WEBSITE</a></b><p>"+"\n")
f.write("</body>"+"\n")
f.write("</html>")

# Close the file
f.close()

# Open the html file with Midori browser
os.system("midori /home/pi/test.html")
```

This program was written for Raspbian Wheezy, but could be adapted for Windows PC's by changing the file name path and the browser name in the os.system call.

PYTHON VERSION: 2.7.3rc2
PYGAME VERSION: 1.9.2a0
O.S.: Debian 7

TESTED!

First open Leafpad. Type in the text shown to the right and save the file as 'settings.txt' in the same location where your python code will be saved.

```
screen width:1024
screen height:600
window caption:Fading Titles
text size:100
title 1:Jaseman Presents...
title 2:A Python Pit Production
title 3:FADING TITLE DEMO
```

```
# Import Settings

# By Jaseman - 22nd September 2012

f = open('settings.txt', 'r') # Opens a text file to read settings from (r)
settings = [] # Create a variable array to hold the settings

for line in f: # Loop to get each line of the file into the array
      settings.append(line)

f.close() # Close the file

# This part splits each line at the colon (:) and defines variables
screenx=settings[0].split(':');screeny=settings[1].split(':')
windowcaption=settings[2].split(':');textsize=settings[3].split(':')
title1=settings[4].split(':');title2=settings[5].split(':')
title3=settings[6].split(':')

import os,pygame; from pygame.locals import *; pygame.init()
os.environ['SDL_VIDEO_WINDOW_POS'] = 'center'

pygame.display.set_caption(windowcaption[1].strip())
screen=pygame.display.set_mode([int(screenx[1]),int(screeny[1])],0,32)
fadesurf=pygame.Surface((int(screenx[1]),int(screeny[1])))
titlesurf=pygame.Surface((int(screenx[1]),int(screeny[1])))
nexttitle=1;run=1

while run==1:
      # Print the next title
      font = pygame.font.Font(None,int(textsize[1]))
      if nexttitle==1:
            text = font.render(title1[1].strip(),True,(255,255,255))
      if nexttitle==2:
            text = font.render(title2[1].strip(),True,(255,255,255))
      if nexttitle==3:
            text = font.render(title3[1].strip(),True,(255,255,255))
      tgr=text.get_rect
      tp=tgr(centerx=screen.get_width()/2,centery=screen.get_height()/2)
      titlesurf.blit(text,tp)
      # Increase the transparency of fadesurf
      for t in range(255,0,-20):
            fadesurf.set_alpha(t); screen.blit(titlesurf,(0,0))
            screen.blit(fadesurf,(0,0)); pygame.display.update()
      # Decrease the transparency of fadesurf
      for t in range(0,256,20):
            fadesurf.set_alpha(t); screen.blit(titlesurf,(0,0))
            screen.blit(fadesurf,(0,0)); pygame.display.update()
      titlesurf.fill((0,0,0)); screen.blit(fadesurf,(0,0))
      pygame.display.update()
      nexttitle+=1
      if nexttitle>=4: nexttitle=1
```

Try changing the values in the 'settings.txt' file and then run the python program again. By this method you can change how the program runs without having to alter the python code itself.

Feedback

I'm Faizal from Malaysia. Just ordered a Raspberry Pi and waiting to reach me. Saw this link today for your mag. Great effort but if can have more images (more pictures) view would be great.

Faizal

Your magazine is great. I'm using it to teach python to my kids. One possible suggestion, would it be possible to produce a .mobi or .epub version, as this would save paper, and i could just put all the magaines on the Kindle for them to read and copy from?

Adam

I am a Pi owner and just today found out about the magazine. I am planning on starting at issue 1 however I would have found it sooner had it been on Zinio (http://za.zinio.com/) I use this Android app to read all my magazines. So if you can publish it there, it would be awesome, and I can get the updates the moment the next one comes out.

Q

Still a great mag. Just one question have you stopped the skutter articles as this is the second month with nothing regarding the skutter? I hope you have not as I was looking forward to the articles.

Andy

While I enjoy reading the MagPi, I'd like to read it on my Kindle, if you already offer that as an electronic download. The two column layout is a pain to read on the Kindle. Would you consider one of the following options:

1. Master the MagPi for one column half-size-pages PDF
2. Create an edition for the Amazon Kindle shop

Kirill

If only a magazine had existed in the early days of the Sinclair ZX80 my life would have been so much easier. My Raspberry Pi is due to be delivered by Santa but I have your excellent magazine to keep me informed and interested till then. Many thanks for an excellent and innovative magazine.

Ron

Just a thanks and to let you know, I'd pay for this.

Newell

The MagPi™

editor@themagpi.com