ISSUE 16 - SEP 2013

Get printed copies at themagpi.com

The MagPi

A Magazine for Raspberry Pi Users

BIGGEST ISSUE YET!
44 pages of Raspberry Pi goodness

- USB Arduino link
- Logi-Pi FPGA
- Pi Matrix
- PATOSS
- Pi-Lite
- Bash
- Java
- XML

Skutter: Expanding your senses with I²C

Two competitions! Win a 512MB Raspberry Pi & interfacing goodies

ISSN 2051-9990

The MagPi

http://www.themagpi.com

Welcome to issue 16 of The MagPi.

It's back... Skutter returns to the pages of The MagPi and this time it's more sensitive! Stephen takes you in detail through expanding the number of inputs and outputs which can be controlled from your Raspberry Pi using I²C. This will allow you to add more sensors to your bot while driving the base unit.

In this issue we have some great hardware projects like Jorge's PATOSS for monitoring his injured bird and we learn how to scroll text on the Pi Matrix.

We have more on connecting your Raspberry Pi to an Arduino in Tony's great article on driving a liquid crystal display plus an amazing look into connecting your Raspberry Pi to Logi-Pi by Michael Jones.

After all that, we supplement the above with some fantastic software articles.

We are pleased to provide more on programming in Java by looking at control flow sentences, numbers, strings and booleans with Vladimir. For the cherry on the cake we have more from Bash gaffer tape and building and parsing XML in Python.

Hope you enjoy the biggest issue of The MagPi to date.

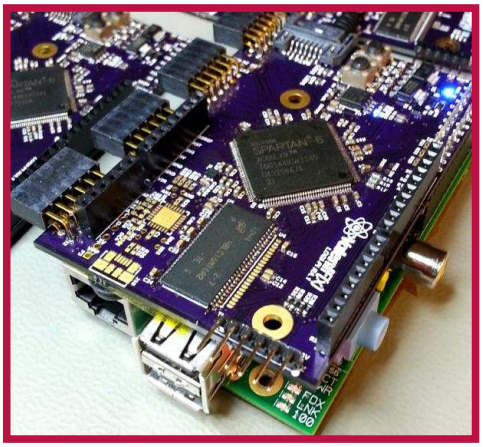
Ash Stone

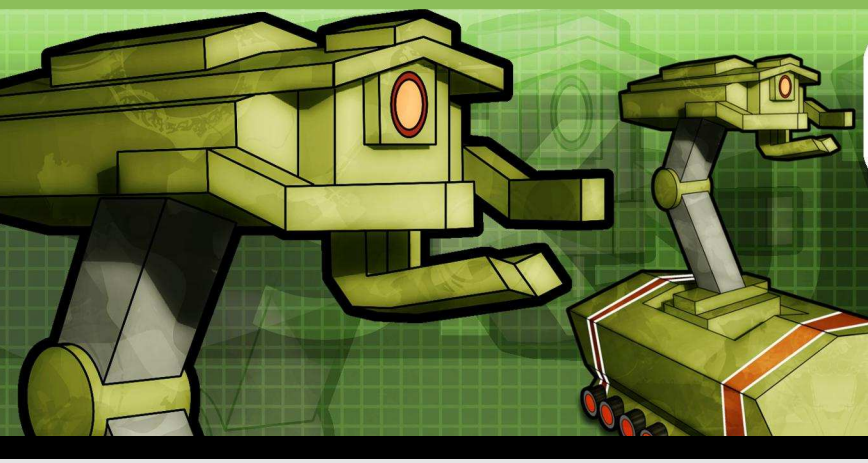
Chief Editor of The MagPi

The MagPi Team

Ash Stone - Chief Editor / Administration / Layout
Aaron Shaw - Issue Editor / Layout / Proof Reading
W.H. Bell - Layout / Administration
Bryan Butler - Page Design / Graphics
Ian McAlpine - Layout / Testing / Proof Reading
Colin Deady - Layout / Proof Reading
Matt Judge - Website / Administration
Shelton Caruthers - Proof Reading
Sai Yamanoor - Tester
Claire Price - Layout / Proof Reading
Courtney Blush - Proof Reading
Amy-Clare Martin - Layout
Matt Weaver - Layout
Gerry Fillery - Proof Reading

Contents





Skutter - Expanding your senses with I²C

Stephen Richards

Guest Writer

DIFFICULTY : ADVANCED

Limitations of the GPIO

We all love the Raspberry Pi. For me the best thing about it, apart from its low cost and low power consumption, is the General Purpose Input and Output header (GPIO). Having that sort of access between the virtual and real world on such a tiny yet powerful computer has allowed me to begin working on a robotics project which until recently I was only able to imagine.

In spite of this, the Raspberry Pi does have some limitations when it comes to building robots. The GPIO has only a limited number of pins to work with. Let's say you have built two simple H-bridge circuits to control motors on your robot. That might take up as much as eight of your GPIOs. Let's imagine you want to include some micro switches on the gripper of a robot arm and some more switches on the robot's bumper (or fender if you are in the US) to detect collisions... before long we've got more inputs and outputs than we've got pins! Things are only complicated further when we consider that if we draw more than 48mA in total from the GPIO at any point, we run the risk of frying the whole thing!

Introducing I²C

So what is a robot builder to do? Happily there is a solution to all of this: something that throws open the doors to a whole galaxy of wonderful electronic devices and sensors. This solution is called the I²C bus (often called the "two wire interface").

Some people pronounce I²C as "eye two see" while others say "eye squared see"; either is acceptable. One of the many available devices that use this is a "GPIO expander" such as the MCP23008. *[Ed: The MCP23017 which is used in the Pi Matrix is the 16-bit version]*. There are many others also available such as "analogue to digital converters", "accelerometers" and many more which I will cover later.

Using the I²C bus can be daunting at first. I²C is unlike USB. USB just seems to work because, although highly complex, it is tightly controlled with a very rigid set of standards. The I²C bus in comparison is much more of a free-for-all. Nevertheless it is extremely effective.

To work with this beast, first we need to understand it. It's known by the nickname "two wire" because it uses two wires to communicate between two or more devices. One of these devices is known as the master with the other devices known as the slaves. In most

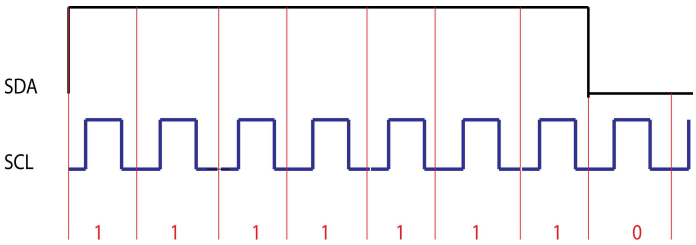
conceivable situations the master would be the Raspberry Pi.

A tale of two wires

On the GPIO header are two pins called SDA and SCL. These two pins are the keys to opening the I²C bus. SDA refers to **S**erial **DA**ta and SCL refers to **S**erial **CL**ock.

The SDA is able to transmit a series of bursts of eight 1s and 0s (called a "byte"). These bytes of data are sent along the SDA wire (bus) and are received by all of the slaves on the bus. The "clock" on the SCL wire is used by both the master and the slaves to allow timings to coincide with these 1s or 0s.

For example, imagine if the byte sent across the SDA bus from the master was "11111110". How would the slave know that this was seven separate "1" bits or just one single "1" bit that has been turned on for a long time? The clock pulses on the SCL bus allows the slave to know this.



When the master sends a byte down the SDA bus, every slave on that bus will receive that byte. "Addressing" ensures that the right device actually uses it. When two devices communicate over I²C they do so in a series of bytes. This series of bytes is called a "message". A message always begins with a special "start" sequence and ends with a special "stop" sequence. This start and stop sequence of bits cannot occur at any other time.

After a start sequence, the next byte of information is a seven bit address. A byte contains eight bits and the eighth bit of the address is used to tell the slave device if it is going to be used to send or receive data.

Once the first address byte has been sent, only the device with that corresponding address continues paying attention to the rest of the message. All the other devices stop paying attention until the stop message sequence is sent.

Disorganisation is the key to success

This seems logical so far, but now we get to the disorganised part. The I²C protocol dictates what the "start" message and "stop" message must be and that the first byte of the message must be the address of the slave device. After that there are no rules at all. Every different I²C device is allowed to use all the data between the address and the stop message in any way it wants. Before you can use an I²C device, you have to study the documentation (called the data sheet) for it to understand how it uses all those bytes of data.

In the rest of this article I will provide an example of this by working through the relevant parts of the datasheet for the MCP23008 I²C 8-bit GPIO expander, plus how to connect this to the Raspberry Pi and how to write a simple Python program to control it.

Controlling the MCP23008

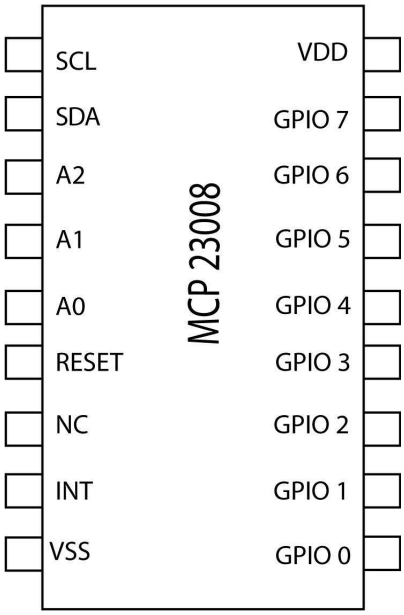
The MCP23008 is very useful to a robot builder because it gives us a relatively simple means of providing more inputs and outputs than the standard GPIO and with more power available.

As stated already, every I²C device works differently. For this reason it is essential to obtain a copy of the datasheet for the device you are working with. To get the most from this article download a copy of the MCP23008 datasheet (`http://ww1.microchip.com/downloads/en/DeviceDoc/21919e.pdf`). From here on in I will be referring to the "MCP23008/MCP23S08 8-Bit I/O Expander with Serial Interface" published in 2007 by Microchip Technology Inc.

The first useful piece of information in this

datasheet is the pin out diagram on page 2. Make sure you look at the pin out for the correct device as several are shown. Using this diagram we can begin to make sense of how we can connect this microchip to our Raspberry Pi.

The first question is which way is "up" on our microchip? If you look carefully you will see a small notch on one end of the device. This signifies the "top".



The key here are the first two pins on the left hand side: SCL and SDA. These can be connected to the corresponding pins on the GPIO of the Raspberry Pi.

The next three pins are A2, A1 and A0. From page 5 these are the address pins.

On page 8 the datasheet states "The slave address contains four fixed bits and three user-defined hardware address bits". This means this part of the address byte can be configured to whatever we want. Let's assume we connect pins A2, A1 and A0 to ground. This is equivalent to making them **000**. On the same page we are given a diagram that shows that the first part of the address for this device must begin with **0100**. Our three pins set the second part of this address to 000, therefore the address for this device configured in this way would be **0100000**.

The very final bit in this address is used to command the MCP23008 to work as an input or an output. The diagram on page 8 of the datasheet shows we must use 0 for write (output) and 1 for read (input).

The next pin along is called RESET. The pinout description on page 5 declares this must be biased externally. For normal use you can get away with just connecting this pin directly to the positive (+ve) terminal of your power supply.

Power requirements

The other two really important pins are V_{SS} and V_{DD}. V_{SS} is ground and V_{DD} is +ve power. (I found that out by checking the pin out description again on page 5 of the datasheet).

Skipping along in the datasheet to page 23 we find the electrical characteristics of the device. From here we can see that this microchip can run on 3.3V logic or 5V logic. Additionally we can see that the device can sink or source 25mA from each GPIO up to a total of 125 mA.

Access to this extra power boost from the GPIO is very useful. Not only does it give us more pins, it supplies more power as well! This is a great advantage when building a H-bridge motor controller, for example. It also means we can use much cheaper, lower valued, current gain transistors than those that were necessary when running one directly from the Raspberry Pi GPIO.

There is another important health warning to consider here however. Although it is possible to run this device on 3.3V or 5V, the Raspberry Pi itself is not tolerant of 5V. Connecting this device to a 5V supply and then trying to use your Raspberry Pi to control it is very likely to cause terminal damage to at least the GPIO, if not the whole Raspberry Pi! Happily it is possible to convert the 3.3V logic of the Raspberry Pi to 5V logic and let the two run safely together using a simple logic level converter circuit. I will describe this circuit later on. For now I will explain how to start using the MCP23008 with the Raspberry Pi.

Setting up the MCP23008 and RasPi

Start off by carefully plugging the MCP23008 into a breadboard. You will see a "gutter" going down the middle of your breadboard. This gutter isolates the two halves of the breadboard from

each other. This means that you can plug in the MCP23008 and know that you are not connecting the pins on either side of the chip to each other. The gutter also makes it easy to remove a microchip. You can carefully work a small flat headed screwdriver under the chip and along the gutter to lift the chip out without bending all the pins.

Before we can start using I²C on the Raspberry Pi we need to make a few changes to some configuration files (assuming you are using a Raspbian distro). The I²C drivers are disabled by default. Let's enable them. From the command line, enter:

```
cd /etc
sudo nano modprobe.d/raspi-blacklist.conf
```

Look for the entry `blacklist i2c-bcm2708` and add a hash '#' at the beginning of the line so it becomes `#blacklist i2c-bcm2708`. Press <Ctrl>+<X> then press <Y> and <Enter> to save and exit.

Next edit the modules file. From the command line, enter:

```
sudo nano modules
```

Add `i2c-dev` on a new line. Press <Ctrl>+<X> then press <Y> and <Enter> to save and exit.

Next install some tools and Python support. From the command line, enter:

```
sudo apt-get update
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
```

Now add the 'pi' user to the i2c group. From the command line, enter:

```
sudo adduser pi i2c
```

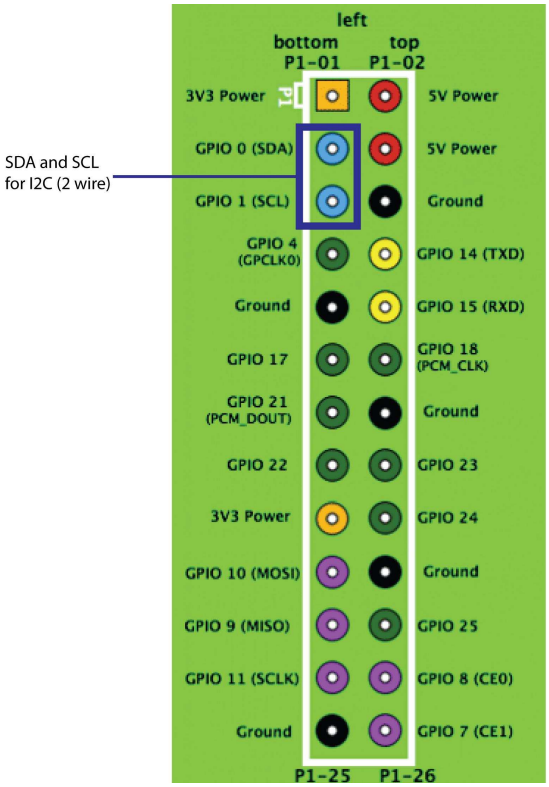
Finally, shutdown your Raspberry Pi. From the command line, enter:

```
sudo halt
```

Plug in the MCP23008

Connect the SDA and SCL on the MCP23008 to the corresponding GPIO connections on the Raspberry Pi.

Connect V_{DD} and RESET on the MCP23008 to 3.3V power on the GPIO. Connect V_{DD} on the MCP23008 to Ground on the Raspberry Pi.



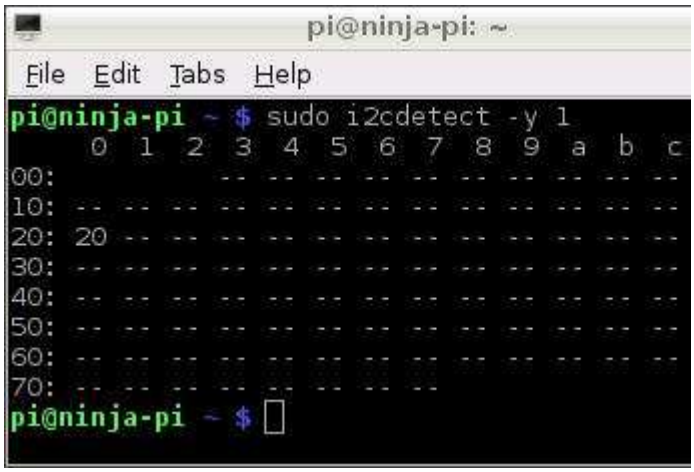
We are now ready to try a few experiments. Turn on the Raspberry Pi.

From the command line, enter:

```
sudo i2cdetect -y 1
```

NOTE: Use 0 instead of 1 for the bus number in the above command if you have an original (revision 1) Raspberry Pi. The revision 1 Raspberry Pi does not have any mounting holes.

If everything is connected up properly you should see output something like the screenshot on the next page.



This means the MCP23008 is communicating with the Pi on address (hex) 0x20.

If we want to use more power than the GPIO on the Raspberry Pi can deliver there are a few more steps we need to take first.

Converting 3.3V to 5V logic

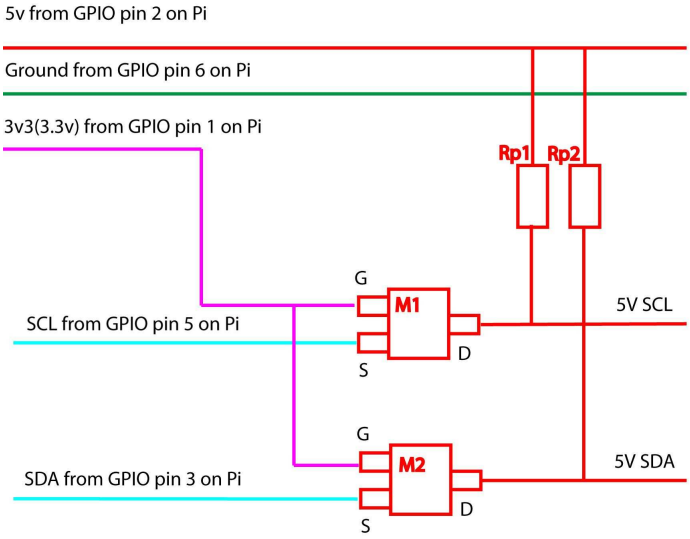
Connecting V_{DD} of the MCP23008 to 3.3V on the GPIO of the Raspberry Pi means that the output of this chip is still affected by the GPIO power limitations. We can connect the device to the 5V supply instead, however that means you are mixing 5V logic with 3.3V logic on the Raspberry Pi and it will not take kindly to this!

The circuit diagram on the right shows a simple circuit that can safely convert 3.3V logic to 5V logic and vice versa. It uses two MOSFETs. A suitable MOSFET is the commonly available BSN20.

Be warned however that the BSN20 is a very small device. To get it working I cut the tracks on

a small piece of stripboard, soldered the terminals of the MOSFET across these and then added some larger connectors to make it suitable for plugging into breadboard.

The symbol used in the circuit diagram is not the conventional MOSFET symbol. I have shown it like this to help visualise the way the little device should be connected.



The purple line signifies 3.3V from the GPIO. This goes to the "Gate" on the MOSFET. The red line signifies the 5V supply. This is connected to the "Drain" of both of the MOSFETs via a "pull up" resistor. (In I²C the natural state of the bus is "low". When a current is applied to the bus via a pull up resistor, it temporarily pulls the signal up to an "on" or "high" condition).

The value of the pull up resistor is not that important. It's generally agreed it should be between 1K and 10K. Many people use 4.7K and report that it works well and this is the value I used for my version of the circuit.

Finally, we need to provide conversion from SCL and SDA on the Raspberry Pi. These lines should be connected to the "Source" of the MOSFET.

Everything on the right hand side of this circuit is now a 5V I²C bus and everything on the left is a 3.3V I²C bus and the two can work safely together in series!

Creating an example Python H-bridge control program

We are now ready to control the GPIO pins 0 to 7 on the MCP23008. Looking at page 6 of the datasheet we can see that the device uses a number of internal registers to control everything it is capable of. The only registers we are really interested in are IODIR and GPIO. Nevertheless it is important to set all the other registers as well to try to prevent anything unexpected happening.

Sometimes the language used in data sheets can be confusing so I have tried to translate into plain English the name, address and function of each register and put it in the table on the next page. I recommend reading the datasheet first and then check your understanding. Being able to read these documents is an important skill.

Register	Address	Description
IODIR	0x00	0 = out (write), 1 = in (read)
IPOL	0x01	Input / output polarity on GPIO bit 7 to bit 0. If bit is set, GPIO value will reflect the inverted value.
GPINTEN	0x02	Interrupt on change on bit 7 to bit 0. If bit is set then it will generate an interrupt if that pin changes.
DEFVAL	0x03	Default value to compare against GPINTEN bit 7 to bit 0. If bit is set, opposite value on corresponding pin will generate interrupt.
INTCON	0x04	Interrupt control register. If bit is set then corresponding IO pin will be compared against the value set in the DEFVAL register.
IOCON	0x05	Setup: bit 5 = sequential operation, bit 4 = slew rate, bit 3 is not used, bit 2 open drain, bit 1 = sets polarity of INT pin. Only functions if bit 2 is clear.
GPPU	0x06	GPPU pull up resistor, bit 7 to bit 0. If bit is set and pin is input then this will pull up the pin with 100k resistor.
INTF	0x07	Interrupt flag, bit 7 to bit 0. If bit is set it means the associated pin will generate an interrupt. A set bit tells us which pin caused the interrupt. READ ONLY.
INTCAP	0x08	Interrupt capture. Captures GPIO value at time of interrupt, bit 7 to bit 0. Remains unchanged until interrupt is cleared via a read of INTCAP or GPIO.
GPIO	0x09	The GPIO, bit 7 to bit 0.
OLAT	0x0A	Output latches.

Python code

The last Skutter article in issue 8 of The MagPi included some simple Python code to control a H-bridge motor controller connected to the GPIO on the Raspberry Pi.

The code on the next page will control two H-bridge controllers connected to the GPIO's on an MCP23008. (Don't forget to change the bus to 0 if you are using a revision 1 Raspberry Pi).

Conclusion

I hope this will help you to begin to understand how to control I²C devices and how useful they can be to a robot builder. If you find an I²C device that can perform a vital function for a robot you are building, (such as an analogue to digital converter, an accelerometer or a distance sensor), hopefully you will now be able to read through the datasheet and make sense of how to control it.

```
#! /usr/bin/python

import smbus import time
address = 0x20

# Define all the registers
IODIR = 0x00
IPOL = 0x01
GPINTEN = 0x02
DEFVAL = 0x03
INTCON = 0x04
IOCON = 0x05
GPPU = 0x06
INTF = 0x07
INTCAP = 0x08
GPIO = 0x09
OLAT = 0x0A

bus = smbus.SMBus(1)    # Change to 0 for revision 1 Raspberry Pi

# Set IODIR as OUTPUT
bus.write_byte_data(address, IODIR, 0b00000000)

# Reset all the other registers
for reg in [IPOL,GPINTEN,DEFVAL,INTCON,IOCON,GPPU,INTF,INTCAP,GPIO,OLAT]:
    bus.write_byte_data(address, reg, 0b00000000)

# Set the GPIO's to turn on/off transistors in H-bridge. See circuit diagram.
#GPIO 0 - 1 = motor 1 fwd.
#GPIO 1 - 1 = motor 1 fwd.
#GPIO 2 - 1 = motor 1 rev.
#GPIO 3 - 1 = motor 1 rev.
#GPIO 4 - 1 = motor 2 fwd
#GPIO 5 - 1 = motor 2 fwd
#GPIO 6 - 1 = motor 2 rev
#GPIO 7 - 1 = motor 2 rev

#---------------------------- IMPORTANT -------------------------------------
# IF GPIO 0, 1 is "1" THEN GPIO 2, 3 must be "0" ELSE transistor short circuit.
# IF GPIO 4, 5 is "1" THEN GPIO 6, 7 must be "0" ELSE transistor short circuit.
#----------------------------------------------------------------------------

# Set all GPIO off
bus.write_byte_data(address, GPIO, 0b00000000))
# Test motor 1 and motor 2 FWD for 3 secs
bus.write_byte_data(address, GPIO, 0b00000011))
time.sleep(3)
# Set all GPIO off
bus.write_byte_data(address, GPIO, 0b00000000))
time.sleep(1)
# Test motor 1 and motor 2 REV for 3 secs
bus.write_byte_data(address, GPIO, 0b00001100)
time.sleep(3)
# Set all GPIO off
bus.write_byte_data(address, GPIO, 0b00000000)
time.sleep(1)
# Test hard right turn for 1 sec
bus.write_byte_data(address, GPIO, 0b11000011)
time.sleep(1)
# Test hard left turn for 1 sec
bus.write_byte_data(address, GPIO, 0b00111100)
time.sleep(1)
# Set all GPIO off
bus.write_byte_data(address, GPIO, 0b00000000)
```

The Pato surveillance system

DIFFICULTY : INTERMEDIATE

Jorge Rancé

Guest Writer

At the start of July my friend and I came across a bird in the street that had apparently sustained an injury to his leg. We took the bird, later named Pato, to the vet for review. It was reported by the doctor that Pato had a broken leg - this required plastering and Pato needed a lot of TLC for a week.

As I do not spend a great deal of time at home the idea of setting up a system in order to monitor Pato was posed. As my occupation is a System Engineer for Linux / Unix systems, I though it wouldn't be so difficult to set-up a monitoring system by using a Raspberry Pi board and some sensors in order to monitor Pato via the internet. Thankfully I had a spare Raspberry Pi lying at home and so I got to work.

What to monitor and how to do it

The system I would create had to be able to send a current picture from Pato's cage, check the temperature of his environment and control the water level. In order to meet these goals I used a webcam for the pictures, a thermometer for the temperature and a liquid level sensor for the water level.

Taking pictures

To set up the webcam was quite straightforward. After plugging it into the USB port and powering up the Raspberry Pi, I ran 'lsusb'. This command checks to see what USB devices the system detects and recognises. In my case the webcam was properly detected as:

```
pi@raspberrypi ~ $ lsusb | grep C270

Bus 001 Device 006: ID 046d:0825 Logitech, Inc.
Webcam C270

pi@raspberrypi ~ $
```

Once the webcam was properly detected, I had to enter some simple commands to install and configure motion:

```
sudo apt-get install -y motion
```

After installation had completed, before starting the webcam, I had to modify three parameters inside /etc/motion/motion.conf - giving them the following values:

```
Daemon = OFF to ON

webcam_localhost = ON to OFF

start_motion_daemon= "no" to “yes”
```

With those changes made I then ensured that the Raspberry Pi was streaming video by entering the following into a browser:

```
http://192.168.x.x:8081
```

(Where x.x should be altered to match the end numbers from your IP address which you can find by typing `ifconfig` into a terminal window.)

Monitoring the temperature

In order to check the temperature of Pato's environment, I bought a USB Temper thermometer via eBay. It required more work than the webcam to get working.

First, I had to ensure all the necessary dependencies were installed. This was completed using the following command:

```
sudo apt-get install -y build-essential libusb-1.0.0 libusbdev
```

The latest version can be cloned from the temper binary via git:

```
git clone https://github.com/bitplane/temper.git
```

Once downloaded it can be compiled with:

```
cd temper/make
```

And then, the new binary must be run using:

```
sudo ./temper 16-Jul-2013 00:02,26.089081
```

I prefer to get rid of the date and time. This can be done using the following script:

```
TEMP=`sudo /home/pi/temper/temper | awk '{ print $2 }' | cut-d, -f2 | cut -c1-5` echo "$TEMP'C"
```

This must then be run using the command:

```
pi@raspberrypi ~ $ ./temperatura.sh 25.63'C
```

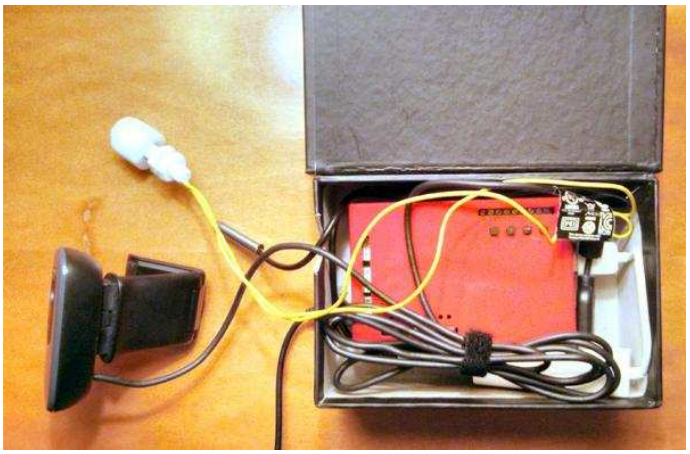
Reading the liquid level sensor

I had recently bought a PiFace board and felt that this would be a good opportunity to use it for the first time.

It couldn’t be easier. The liquid level sensor was connected to digital input zero. In order to check if there's water or not, I ran the following script. If there's enough water, it would return 1:

```
import piface.pfio as pfio

pfio.init()

print pfio.digital_read(0)
```

And then, run it!



```
pi@raspberrypi ~ $ python boya.py 1
```

Sending tweets from the Raspberry Pi

Once everything was set up from the software

and hardware point of view, now was the time to find out how to send tweets.

I did open a Twitpic account in order to make things easier, so it just had to send an e-mail with an attached picture and the information I wanted to be posted. In order to do that I wrote a really simple script:

```
#!/bin/sh
CPUTMP=`/home/pi/cpu_temp.sh`

ENVTMP=`/home/pi/temperatura.sh`

LIQUID=`python /home/pi/boya.py`

if [ ${LIQUID} -eq 1 ]; then

elif [ ${LIQUID} -eq 0 ]; then
fi

SUBJECT="RPi temp: ${CPUTMP}. Room temp:
${ENVTMP}.

${AGUA}. " echo "" | mutt -a
/tmp/motion/patoss.jpg -

s "`echo ${SUBJECT}`" -- xxxx@twitpic.com
```

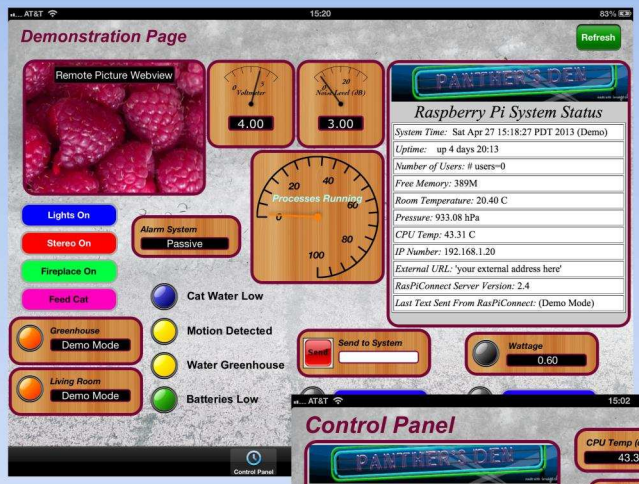
And this is what is being posted on Twitter:

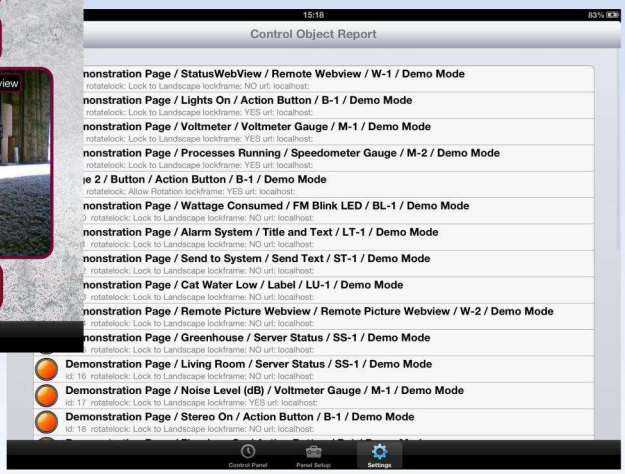
```
AGUA='Water level OK'

AGUA='Pato needs water'
```

Well that concludes the project! I am pleased to write that Pato is doing very well and I can work away from home, safe in the knowledge that he is well looked after.

RasPiConnect

Connect your Raspberry Pi to the World!





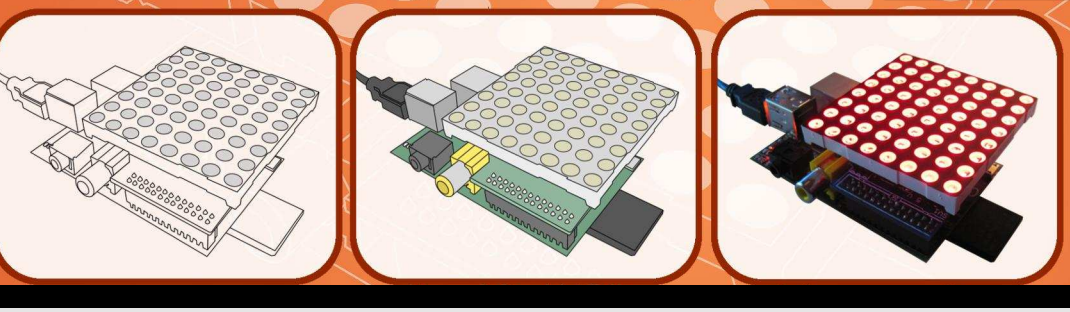
Now Supports Arduino with in-app purchase

Available on the App Store

RasPiConnect connects your Raspberry Pi to the outside world. It allows you to control virtually anything you connect to your Raspberry Pi from your iPad or iPhone.

- EASY to setup - no syncing required
- Buttons, gauges, webpages, webcam pictures and more!
- Build your pages on your iPad/iPhone
- Supports multiple Raspberry Pis
- Five pages of control panels
- Unlimited Controls
- Exchange your panels with friends
- Supports any computer that supports Python (windows, linux, etc.)
- Now allows custom backgrounds

MiloCreek
Where Innovation Flows



Bruce E. Hall
W8BH
Guest Writer

Part 4: Multiplexing and scrolling text messages

DIFFICULTY : MODERATE

Introduction

Previously we looked at how to build the Pi Matrix and how to drive its 16 pins. But 16 pins cannot completely address all 64 LEDs at the same time. We will talk about multiplexing and learn how to scroll text characters across the display, building on sample code from before.

Multiplexing

So far all of our display routines have involved turning on a number of LEDs in a row and repeating the same pattern over any number of rows. But for some applications this is not enough. Suppose we need to turn on 3 LEDs in row 1, a different number of LEDs in row 2 and yet another pattern in row 3. We will need this capability if we want to display complex symbols on the display, like text characters. We need multiplexing.

With multiplexing we do not display all the rows at the same time; we display them sequentially.

For each row:
    Display pattern #n on row #m
    Wait a few milliseconds (at most)

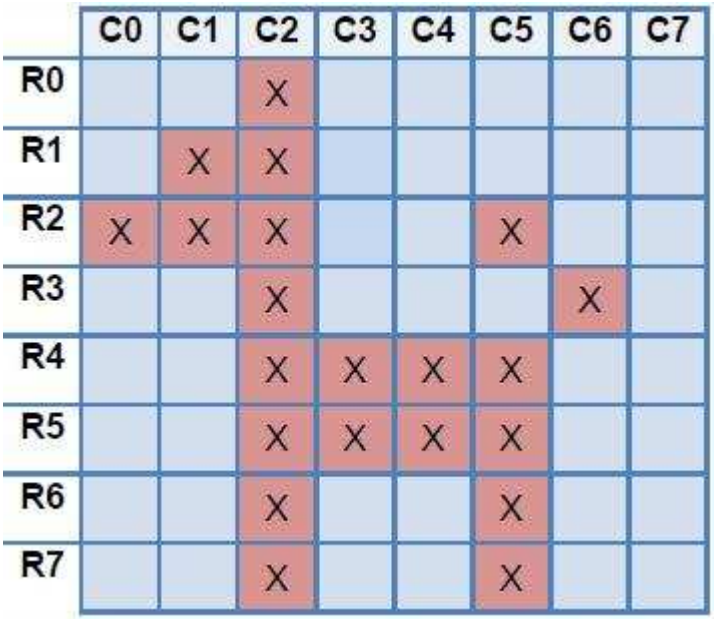
If we do this fast enough our eyes are tricked into thinking that all of the rows are being displayed at the same time (aka persistence of vision). We must refresh the entire display at least 30 times per second, which means that we cannot spend more than about 4 milliseconds on each row. As Python does not run too quickly on the Raspberry Pi the wait between rows can be omitted. The following simple routine will do all the multiplexing we need:

```
def MultiplexDisplay(z, speed):
   for count in range(0, speed):
      for row in range(0, 8):
         SetPattern(1<<row, z[row])
```

The variable 'z' is a list of 8 elements with each element holding a row pattern. z[0] holds the pattern for row 0, z[1] holds the pattern for row 1, etc. The whole display is refreshed (speed) times giving the user enough time to view and interpret the display.

Puppies and fonts

I thought it would be interesting to draw some characters on the Pi Matrix but my 9 year old daughter had a different idea... puppies! She pulled out some graph paper, asked me to draw the right-sized box and then proceeded to shade in the squares of her puppy dog design. She handed the paper back to me and waited for me to put her design on the Pi Matrix. Let's do it!



Starting with the top row, we see that only bit 2 is lit. For this we need binary 00000100, which is 0x04. In the next row, we have bits 1 and 2, which is 0x06. Here are the values that we will need for all eight rows: 0x04, 0x06, 0x27, 0x44, 0x3C, 0x3C, 0x24, 0x24. That's our list input for the routine:

```
def DisplayPuppy():
   z = [0x04, 0x06, 0x27, 0x44, 0x3C,
        0x3C, 0x24, 0x24]
   MultiplexDisplay(z, 100)
```

After this success I was encouraged to tackle the alphabet. A quick internet search for 8x8 fonts gave me exactly what I needed; 128 characters in ASCII order, encoded as 8 rows of 8 pixels (bits). All I needed to do was 'pythonize' the data into one big list of lists, like this:

```
data = [ …
[ 0x0C, 0x1E, 0x33, 0x33, 0x3F, 0x33,
  0x33, 0x00],    # U+0041 (A)
[ 0x3F, 0x66, 0x66, 0x3E, 0x66, 0x66,
  0x3F, 0x00],    # U+0042 (B)
[ 0x3C, 0x66, 0x03, 0x03, 0x03, 0x66,
  0x3C, 0x00],    # U+0043 (C)
    … ]
```

I put this data into its own file called font0.py. You can download this file from `http://w8bh.net/pi/font0.py`. To add it to your program all you have to do is import it. The big list is referred to by its module name, dotted with the list name: font0.data. The index to any character in the list is the ASCII number of the character, which we can get by using the Python function ord(). Try this:

```
import font0
char = raw_input("Enter a character to
display: ")
z = font0.data[ord(char)]
#print char, z
MultiplexDisplay(z, 100)
```

With just a few lines of code we can display any character on the Pi Matrix. I added the '#print char, z' statement for debugging purposes. It is commented out so it doesn't do anything. Remove the hash and it will show you the character's data.

Text

Now let's try displaying words and sentences. Enter a string and display each character in the string.

```
def DisplayString():
   message = raw_input("Enter a message
   to display: ")
   for char in message:
      z = font0.data[ord(char)]
      #print char, z
      MultiplexDisplay(z, 100)
```

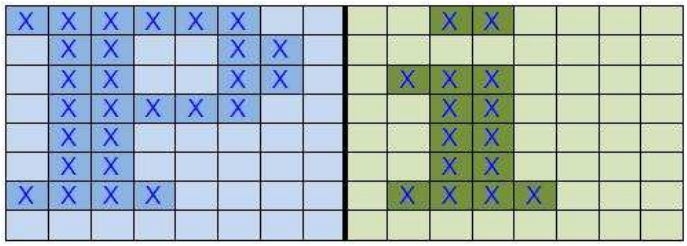
Python is pretty cool here because it is able to iterate over all of the characters in the message without having to grab each character or explicitly set boundaries.

Scrolling

To scroll you need two pieces of data: the data that is currently being displayed and the data that is about to be displayed. For the Pi Matrix this means we will keep track of data for two characters at a time. I call our current character 'z', since this is how I started (above), and call the next (buffered) character 'buf'. To scroll we shift our current display one pixel to the left and

move our buffered data onto the display by the same amount. English is written left-to-right, so left-ward scrolling works the best.

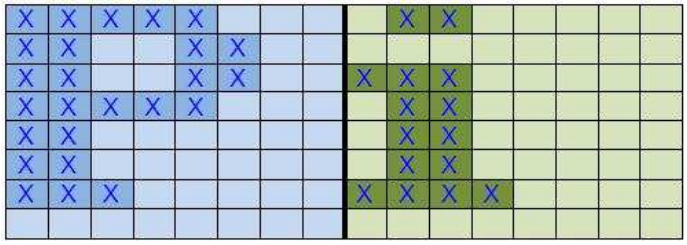
Let's try an example, the word 'Pi'.



z ---------- buf ----------

Here is our data. The blue boxes are 'z' and represent what is being displayed on the Pi Matrix. The green boxes are 'buf', the buffered data waiting to be written to the display.

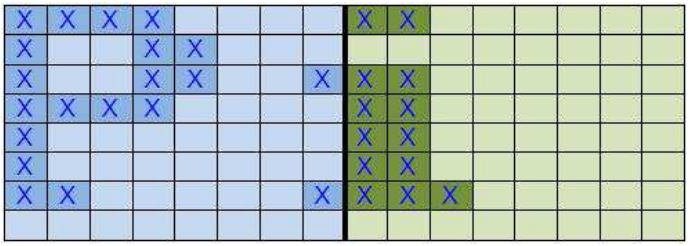
In this example we will scroll the word 'Pi'. The 'P' is being displayed and the 'i' is waiting its turn. If we scroll to the left by one pixel the left-most blue column disappears. Although only the 'P' is visible both characters have shifted slightly to the left.



z <---------- buf <----------

To do this in code, take one row at a time. Shift the bits in the row to the left making sure that the left-most bit in the green box (buf) gets transferred to the rightmost bit in blue (z).

Here's what things look like on the next scroll. The leading part of the 'i' is now appearing on the display.



But what if we were doing a longer word, like 'pie'. We need to add the 'e'. There is too much space behind the 'i' already!

Remember, however, that the green box is just a buffer and isn't being displayed. We'll fill it with a new character as soon as the previous character has been completely shifted into the blue box (i.e is visible).

It is time to code the shift routine, using 'z' and 'buf'.

```
def ScrollLeft(z, buf):
    for row in range(0, 8):
        z[row] >>= 1        #shift current
image row right 1 bit.
        if buf[row] & 0x01: #is bit 1 of
buffer high?
            z[row] |= 0x80   #rotate bit 1
of buffer into bit 7 of current image.
        buf[row] >>= 1      #shift buffer
row right 1 bit also.
```

The '>>= 1' operation shifts the operand one bit to the right; b7 becomes b6 and so on. We do this for each z[row] and buf[row]. The 'if' statement shifts data from buf to z.

Now we have all the parts we need to scroll. Start with a blank display and load the first character into buf. Scroll one bit at a time and display the data. After every 8th scroll, load a new character into buf. Done!

Making it useful

My first python program grabbed user input from a prompt like this:

```
st = raw_input("Enter something: ")
```

There are other ways to get input. One method is to take it from a command line parameter, like this:

```
./matrix4.py "Go Away!"
```

It is easy to get the command line parameters in

Python. First import the sys module then sys.argv will return the list of command line parameters:

```
import sys
print sys.argv
```

From the above example, the program name 'matrix4.py' is contained in sys.argv[0] and our text is in sys.argv[1]. We can also get text from something called 'standard input'. Stdin is the source for command-line programs in all Unix-like operating systems. By default this is the keyboard. If we want to look for other sources from Python we can read from sys.stdin instead. Combining the two gives us lots of input choices:

```
import sys
if len(sys.argv) > 1:
   st = sys.argv[1]
else:
   st = sys.stdin.read()
```

Now we can get do all sorts of neat Linux stuff like pipes and redirects:

```
./matrix4.py "The yellow brown dog"
echo "The MagPi is great" | ./matrix4.py
cat poem.txt | ./matrix4.py
./matrix4.py <poem.txt
```

Python code

The sample code for this month is too long to publish in the magazine, but you can download it from `http://w8bh.net/pi/matrix4.py`. (You may need to change the ORIENTATION constant at the start of the file. Also, Model B Revision 1 owners need to set `bus=smbus.SMbus(0)` near the end of the file).

Conclusion

This concludes our mini-series on the Pi Matrix. As an exercise try to display Conway's "Game of Life" on the Pi Matrix. Have fun!

Back In Time

Because most of the content that we publish in The MagPi is educational, it does not age as fast as content in traditional magazines.

With the 16th issue we have now produced over 500 pages of Raspberry Pi goodness and, unless you've had a Raspberry Pi for some time, there is a good chance you do not know about all of the great articles that we have published in previous issues.

Here is a short list of just some of our earlier articles, with the issue numbers in parenthesis. You can download every issue of The MagPi for FREE from http://www.themagpi.com.

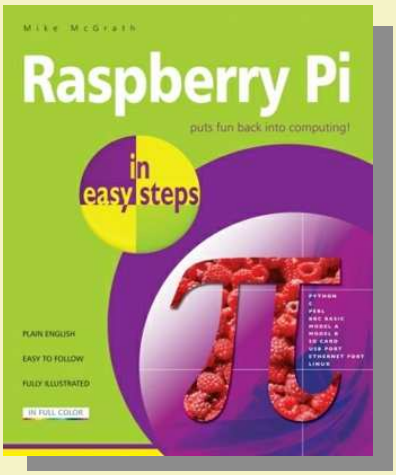
Skutter - build a robot (1, 2, 3, 6, 8, 16)
Play and create computer music (2, 12, 13)
GPIO interfacing for beginners (2, 3, 4, 5, 7)
Command line / Bash (2, 3, 4, 5, 10, 12, 16)
3-axis accelerometer (4)
Customise the LXDE menu (4)
Raspberry Pi media centre (5)
Pumpkin Pi - get ready for Halloween (6)
Arduino and Raspberry Pi (7, 8, 15, 16)
Home automation (8)
SD card backup (9, 10)
RISCOS (9, 11, 13, 15)
Minecraft programming (11)
Printing with the Raspberry Pi (11, 12)
Operating Systems (12)
Raspberry Pi camera module (14, 15)

There have also been many tutorials for a variety of programming languages.

C (3, 4, 5, 6, 9, 13)
C++ (7, 8, 10)
Python (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14)
Scratch programming for kids (1, 3, 4, 5, 13)
Ada (6, 8)
SQL (8)
Charm (10, 11, 14)
Java (14, 16)

Raspberry Pi In Easy Steps

Mike McGrath
In Easy Steps



Raspberry Pi in easy steps does exactly what it says on the tin. As part of the Easy Steps series of how-to guides this book offers a great step-by-step introduction to the Raspberry Pi.

It contains nine chapters of high-quality, illustrated, full-colour pages. It covers everything from purchasing and setting up the Raspberry Pi all the way through to programming in Scratch and Python, developing your own games and applications and controlling the GPIO pins.

Raspberry Pi in easy steps is written in plain English which provides a jargon-free and fun way to find your feet in the exciting world of Raspberry Pi. This clear and easy to follow guide provides a great foundation for new users and those wishing to further their knowledge of this fantastic little computer.

Each chapter includes a number of useful pointers under the headings 'Hot Tips', 'Beware' and 'Don't forget' – these break down key information into easily digestible bite-size chunks. They provide handy tips to spice up your learning, flag something to remember or ward you away from potential dangers.

This book is a clear and fuss-free introduction to the Raspberry Pi – it is great value for money and is suitable for adults and children alike.

The MagPi and Bookaxis are pleased to offer readers a 40% discount. To claim, order from www.bookaxis.com/magpi and quote promo code **MAGPI16**.

Please note: this discount is only valid from the 1st to the 30th September 2013 on the two book titles listed here.

Python for Kids

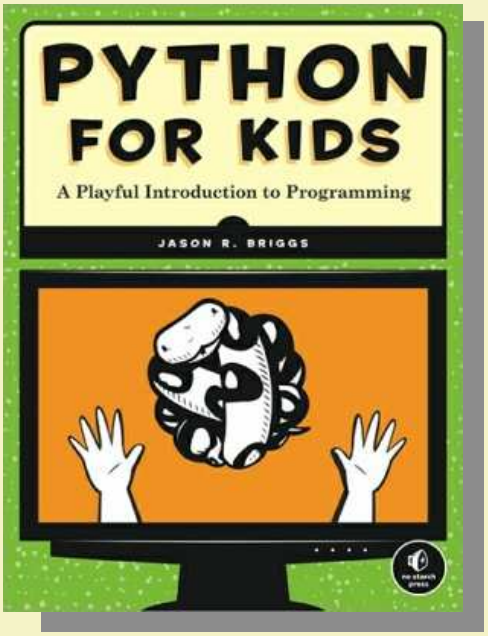
Jason R. Briggs
No Starch Press

Python for Kids is another fantastic volume from No Starch Press – the publishing company which label themselves 'The Finest in Geek Entertainment'. Python for Kids certainly lives up to their reputation as this book provides a refreshingly fun and engaging platform for children to learn about programming.

Having said that, despite its title and obvious target market (the young!) – the quality and technical depth of this book provides a playful introduction to programming to readers of any age.

Starting with basic theory and fundamentals and moving quickly on to creating your own computer games this book is your ticket into the amazing world of computer programming! Each chapter contains a number of fantastic programming puzzles designed to stretch your brain and strengthen your understanding.

Python for Kids puts the fun back into programming and brings Python to life. By the end of this book you will have gained a strong understanding of this powerful and expressive programming language.



There is even a companion website to support your learning, where you will find downloads for all of the examples in this book, solutions and additional programming puzzles.

Bought a Raspberry Pi and wondering what to do with it? This book provides the answer.

A plug and play LED matrix board for the Raspberry Pi

DIFFICULTY : BEGINNER

Jacob Marsh

ModMyPi

The Pi-Lite is a versatile, plug and play, 126 LED (9x14 Grid) matrix display for the Raspberry Pi. Each pixel is individually addressable - so you can display scrolling text, graphics and bar graphs; basically anything that can fit in 126 pixels! It's a great starting place for doing something visual with your Raspberry Pi.

The Pi-Lite comes as a complete, fully assembled board that requires no soldering and it's designed to plug straight into the Raspberry Pi's GPIO ports. The matrix is controlled by an on-board ATMega 328 processor with pre-loaded software and works equally well with a Raspberry Pi using GPIO or with a PC, Mac or Linux machine via the on-board FTDI connector. You'll find a short beginner's guide to set the Pi-Lite up on the Raspberry Pi below.

Step 1 - setting up the Raspberry Pi for basic Pi-Lite functions!

The Pi-Lite is as Ciseco product, so requires a custom Wheezy Image to be loaded onto an SD card and used for this task. This image has reconfigured GPIO pins for serial access and the Minicom terminal emulator that's used to send and receive characters from the serial port is pre-installed. You can set all this up manually on your version of Raspbian; however for ease of this tutorial we'll use the custom image which can be downloaded at the following link:

```
http://openmicros.org/Download/2013-05-25-wheezy-raspbian-ciseco.img.zip
```

Simply unzip the image and load it onto an SD card like the standard Raspbian distribution.

Step 2 - the fun stuff!

Make sure your Raspberry Pi is switched off and then plug the Pi-Lite in. It sits on top of the GPIO ports within the footprint of the Raspberry Pi and fits neatly inside a ModMyPi case. Boot your Raspberry Pi up, log in and you'll be presented with the Raspberry Pi command line. The Pi-Lite will also auto-boot with a very cool sequence!

To access the Pi-Lite module via Minicom and send scrolling text messages, enter the command:

```
minicom -b 9600 -o -D /dev/ttyAMA0
```

Now, simply by typing, you can send any text to the Pi-Lite which will be scrolled across automatically. It's also possible to enter Minicom's command mode to change various settings, such as the scroll speed or pixel state.

To enter command mode type $$$ (three dollar signs) - which will stop all scrolling and Minicom will respond with “OK”. All commands must be sent as one string in UPPER case and terminated with a carriage return (pressing enter). After receiving and carrying out a command the Pi-Lite leaves command mode and returns to scroll mode. If a command is not received within a few seconds or a command is inputted incorrectly, the command control will be terminated and the Pi-Lite will return to scroll mode.

As an example, we’ll increase the scrolling speed using the SPEED command. By default the scroll speed is set to 80, but it can be set anywhere from 1 (very fast) to 1000 (very slow). Let’s slow our scroll speed - simply type:

```
$$$SPEED200
```

Then hit enter. The Pi-Lite will automatically exit command mode and re-enter scroll mode. You can now check to see that your scroll speed has increased!

There’s a full list of commands, as well as the example scripts utilised in Step 3 below, available at the following link. You’ll need these to show bar graphs, turn on/off individual pixels, and generally make your Pi-Lite function:

```
https://www.modmypi.com/pi-lite-raspberry-pi-led-matrix
```

Step 3 - running scripts!

What’s great about the Pi-Lite is that it enables you to run custom Python scripts and subsequently show graphics, repeated text strings, read the weather, run a real-time Twitter feed or display anything else you can imagine! I’ll show you how to download and run some example Python scripts, but you can always edit them or write your own if you’re feeling adventurous! Please note, use upper case in the commands where stated.

The Ciseco Wheezy image will already have a suitable version of Python installed. However, you’ll also need to install the “Git Control System” and the “Python Serial Package”:

```
sudo apt-get install git
sudo apt-get install python-serial
```

We now need to pull the library files from Github and put them in a directory. First ensure you are in your home directory by changing directory to the standard home location:

```
cd /home/pi
```

Then create a directory for the Github example files and browse to it:

```
mkdir git
cd git
```

Now obtain the Pi-Lite source code. This includes the Python examples:

```
git clone
git://github.com/CisecoPlc/PiLite.git
```

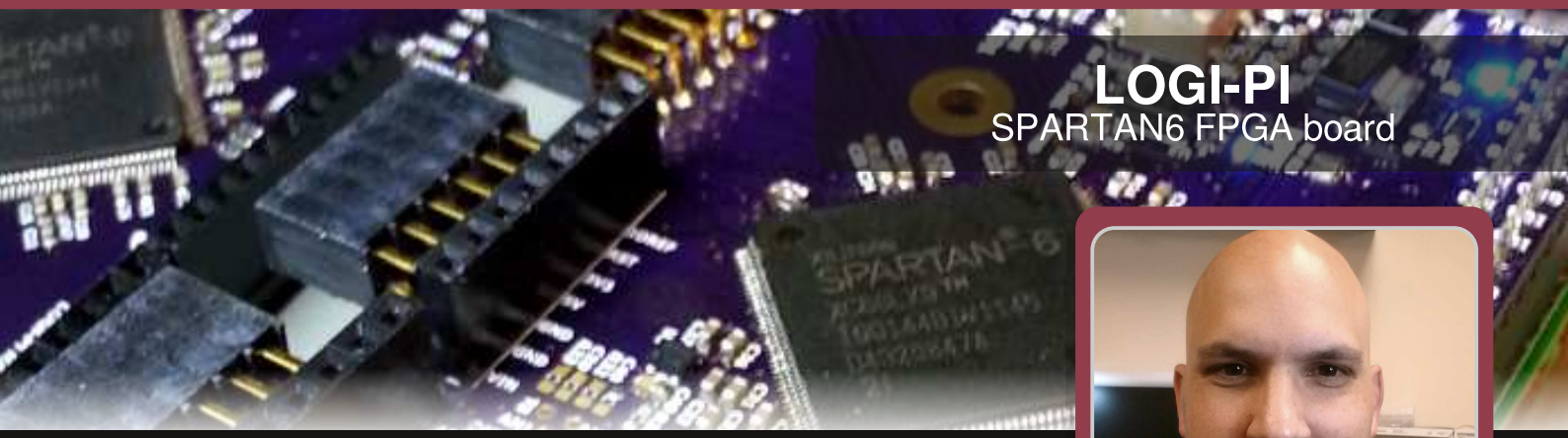
You can now browse to the example scripts:

```
cd PiLite/Python_Examples
```

Some of the scripts can be run straight from the command line via Python (CTRL+C will terminate). For this example we’ll run the Pacman example script, which displays (you guessed it) Pacman on the Pi-Lite!:

```
python Pacman.py
```

As with all Raspberry Pi projects - the best way to learn is to play and a great place to start is the Pi-Lite!



Raspberry Pi meets FPGA

DIFFICULTY : ADVANCED

Michael Jones

Guest Writer

Last month's Arduino - Raspberry Pi usage article introduced the option of directly connecting to, developing, and running an Arduino from the Raspberry Pi. This opens the door for many usages that would not otherwise be available running either of the platforms singularly. Bringing different technologies together in this way enables wider capability in a system. Similar to joining the Raspberry Pi and the Arduino, it is possible to connect many more technological standards to the Raspberry Pi to enable an electronic smorgasbord of functionality. There are so many different electronics standards, interfaces, and processing architectures, and now there's a way to use them together with the LOGi-Pi.

The sea of electronics

We work in a sea of continually expanding electronics boards and modules. Never has there been such an opportunity to work with so many different varieties of modules and processing architectures. There has been an explosion of different electronics options for electronics modules from parts and connectors below .4mm pitch all the way up to the long used standard of 2.54mm pitch. There is the Arduino platform and shields, the Raspberry Pi platform and Plates, the Digilent Inc PMOD modules and Xilinx's high performance FMC expansion module standard and many more. Each of these standards has a myriad of off-the-shelf electronics plug in modules that can be plugged into the corresponding family of development boards enabling the platform to be used in a wide range of different applications. These standards were initially developed to meet the specific needs of the corresponding family only. What if all of these modules could be used by a wider audience and could be used on many different development platforms? The potential for uses and applications would go up exponentially. What if these different module standards could be not only with well known CPU platforms, but with FPGA platforms as well? What if all of these technologies could be used together?

FPGA technology - Parallel processing

FPGA technology is highly contrasting to the more well known and widely used CPU technology. There a few key differences between the two. At the low level, FPGAs are made of transistor configurations that make up logic gates that combine to make combinational logic or flip flops that can then combine to make up register transfer logic (RTL) and the hierarchy and abstractions keeps going up from there. These combinations and abstractions are put together to create CPUs, custom hardware processing modules that are found in ASICS and/or the combination of the multitude of integrated circuits found on the market today.

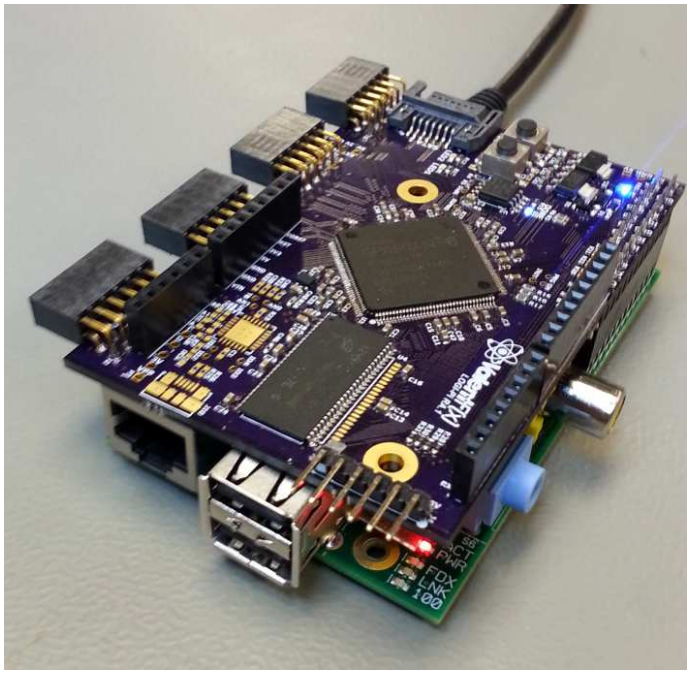
FPGAs are similar to CPUs in that they are able to process data. If a CPU needs to handle tasks A, B and C, the it would typically start with task A and move through task B and C in a linear fashion to finish the tasks. As the CPU is given new tasks, the tasks are added sequentially and are processed until they are all finished. The CPU may become loaded with too many tasks or tasks that take a large number of instructions to complete, slowing down the processing or causing errors. This often happens, for example, in real-time systems. The solution would be to either remove some of the tasks or use a faster processor.

If an FPGA had the same tasks, A,B,C, the tasks could be constructed so that each task was being processed at the same time or in parallel. Any new tasks are also processed in parallel so no additional time is required for the additional processing. Tasks can continue to be added as long as there is enough logic within the FPGA. This example is not to show that FPGAs are superior to CPUs, but to show that the two technologies operate very differently. Both are very good at doing what they do in their respective applications. The FPGA ability to have new functionality added to it without affecting the performance of other functionality makes it great for applications that require flexibility. The complementary nature of operation between CPUs and FPGAs is also the reason that they work very well when used with each other. When used together, these qualities of the CPUs and FPGAs complement each other and deliver outstanding results.

FPGAs are Logic Fabric - FPGAs are like a Chameleon

FPGAs are widely used in the electronics industry because of their unrivaled ability to be fully reconfigured to support any number of contrasting applications while not requiring any modification to the PCB hardware or design. FPGAs are commonly referred to as “logic fabric”. This term implies that the logic fabric can be modified, cut, pieced and applied in many different configurations to create customized applications that otherwise might require the design and fabrication of an ASIC (very expensive); or by using a mixture of different discrete logic and/or processing solutions. Many times FPGAs are used in designs knowing that the design requirements will change and the FPGAs would need to be updated to meet the changes without modifying the existing board hardware. The only change that would be required is for new HDL code to be written and loaded into the FPGA, assuming the inputs and outputs have not changed. In this way, FPGAs are like a chameleon, taking on new colors or disguises to adapt to changing environments.

Introducing the LOGi-Pi



The LOGi-Pi is a member of the LOGi Family of FPGA development boards. It was designed as a solution that unifies the wide range of electronics that are currently available on the market. The LOGi-Pi was designed to provide a plug-and-play experience that supports the most widely used electronics module interfacing standards. This allows the maximum number of off-the-shelf hardware modules to be directly compatible in a plug-and-play manner with the LOGi-Pi. The LOGi-Pi has drivers and applications that create a seamless solution for users who want to use an FPGA, high performance processing CPU. and a multitude of available add-on modules.

Maximum Interfacing Compatibility with existing modules

The LOGi-Pi seeks to allow as much plug-and-play expansion to existing available hardware as possible by using widely available and low-cost, off-the-shelf hardware. PMODs and Arduino Shields were chosen to be used as a standard interface, based on their wide market usage,

availability, and cost. There are currently 59 PMODs and 281 Arduino Shields available that could be used to add direct functionality to the LOGi-Pi. Additionally, high bandwidth SATA interconnects implementing impedance controlled LVDS lines can be interfaced. By using LVDS, such applications as HDMI, SDR, LVDS camera interfaces, and other high bandwidth applications can easily be developed on the LOGi-Pi. All applications are intended to be implemented without needing soldering, jumper wires, or "perf" board expansion as is generally required to interface to many existing FPGA boards.

High Performance Applications

The LOGi-Pi was designed to allow implementation of many high performance applications in a straightforward manner. Many popular applications are best served by the processing capabilities of an FPGA, including SDR (Software defined Radio), quad-copter control, computer vision, and bitcoin mining. Applications have been created for the LOGI-Pi that implement machine vision, and bitcoin mining, and autonomous vehicle controller using GPS, 9-axis IMU, WIFI, and machine vision to run blob detection and color detection algorithms. These applications are representative of a few of the applications that can be developed by using off-the-shelf components and integrating high performance CPU and FPGA technology.

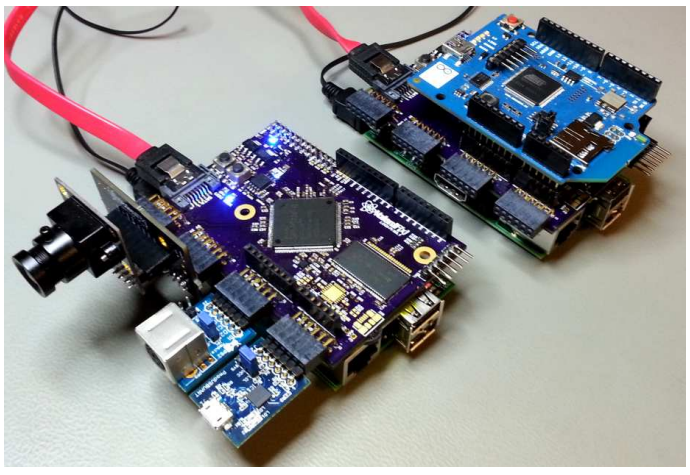
Beginners to experienced FPGA developers

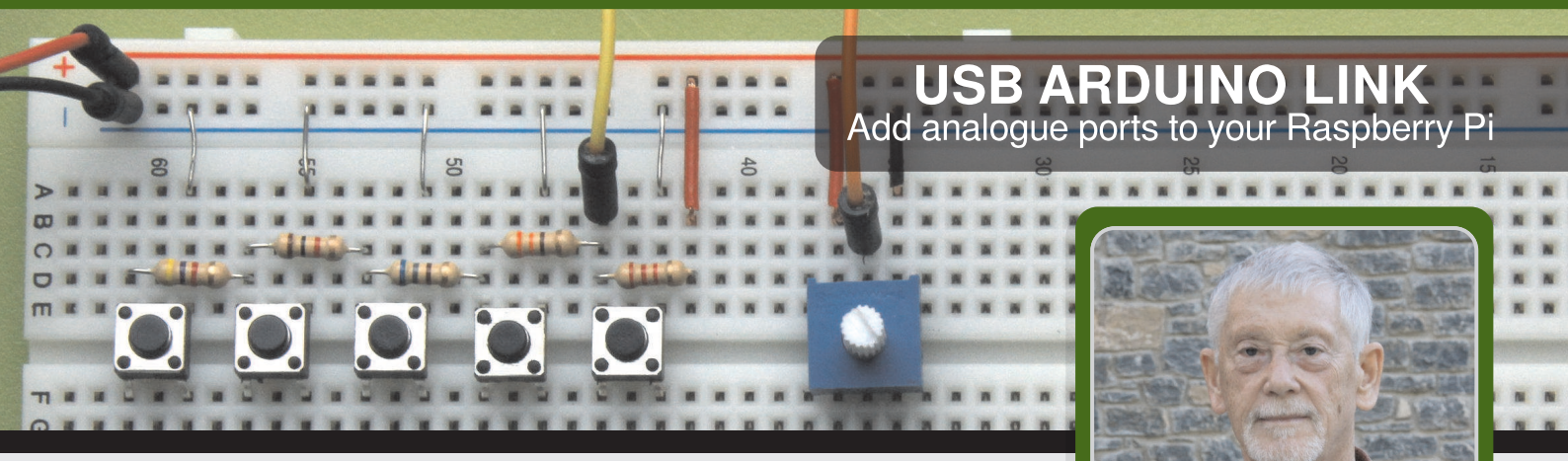
The LOGi-Pi was designed to give beginners an easy way to delve into advanced applications. The LOGi-applications are all open source and are available on github. The applications are organized in such a way that the user will simply need to have git installed on their Raspberry Pi, pull the latest pre-built applications, plug in the hardware modules, including the LOGi-Pi, and then run a shell script that will load the bitstream into the FPGA, setup any needed Raspberry Pi drivers and software, and run the application. The source code is completely open for customizations for those who are interested in getting into the nuts and the bolts of how the applications run.

No matter your experience level, there are fun applications to delve into that use the wide array of available electronics. Let the LOGi-Pi be your gateway into exploring the latest technologies in creating fun, high performance, easy to use applications. Watch for guides in the coming months that walk through the process of using the LOGi-Pi in conjunction with the Raspberry Pi to create these highly functional applications.

You can find information relevant to the LOGi-boards on the logi-blog [1], logi-wiki [2] or logi github repository [3]. Do you have a great idea for a cool project you would like to implement using a LOGi-Pi? We just started the LOGi Contest which will get a free board if your ingenious project idea and plans for implementation are selected as a winner by the LOGi-Team [4]. Want to get involved? We would love to have the help of anyone interested that has some basic skills in HDL and programming (C, C++, python,etc). Want to meet the mad geniuses behind LOGi-projects? Meet Jonathan Piat, principal developer, and the LOGi-Team [5]? Email us with any feedback thoughts or suggestions [6].

[1] http://valentfx.com/logi-blog
[2] http://valentfx.com/wiki
[3] https://github.com/fpga-logi
[4] http://valentfx.com/logi-blog/item/logi-contest
[5] http://valentfx.com/logi-blog/category/logi-team
[6] support@valentfx.com





Using a liquid crystal display and Arduino analogue pins

Tony Goodhew

Guest Writer

DIFFICULTY : INTERMEDIATE

USB Arduino link - Part 2

Last month I wrote about how to set up your Raspberry Pi so that it could communicate with an Arduino via a serial USB cable using Nanpy. In this mode the Raspberry Pi is the master and the Arduino is its slave input/output board, providing protection and extra facilities for the master. I listed simple Python digital input and output programs using a button switch and an LED to test the system. I hope you managed to get it working. I am now going into more detail about driving a liquid crystal display and the 6 analogue pins (A0 – A5) on the Arduino, which can read voltages between 0 and 5 volts.

Using a liquid crystal display (LCD)

Our example uses a 5 volt 16x2 HD4470 compatible LCD (currently about £7 from oomlout.co.uk and from other suppliers). For full circuit information, visit:

```
http://oomlout.com/parts/LCDD-01-guide.pdf
```

It is very easy to control via the built in Nanpy LCD library. When running Nanpy, pins 0 &1 of the Arduino are used for communication with the Raspberry Pi; so I connected my LCD to pins 2 through 7.

LCD pin	Name	Arduino pin	
1	GND	GND	
2	V_{dd} 5V	5V	
3	Contrast	10K ohm potentiometer wiper. Others to 5V & 0V	
4	RS	7	# Not 12
5	R/W	GND	
6	Enable	6	# Not 11
11	Data 4	5	
12	Data 5	4	
13	Data 6	3	
14	Data 7	2	
15	Light +ve	5V	
16	Light -ve	GND	

The following program demonstates the operation by counting up and down:

```
#!/usr/bin/env python
# Basic LCD use via nanpy

from time import sleep
from nanpy import Arduino
from nanpy import serial_manager
serial_manager.connect('/dev/ttyACM0')
from nanpy import Lcd
lcd = Lcd([7,6,5,4,3,2], [16,2])  #LCD set-up
print "### Starting ###\n"
```

```
# Heading - top row
lcd.printString("LCD Demo - Counting", 0, 0)
sleep(1)
for i in range(0, 21):
    position = 4
    if i < 10:
        position = position + 1
    # Clear 2nd row
    lcd.printString("                ", 0, 1)
    lcd.printString(i, position, 1)
    i2 = 20 - i
    position2 = 9
    if i2 < 10:
        position2 = position2 + 1
    lcd.printString(i2, position2, 1)
    sleep(0.7)
print "\n### FINISHED ###"
```

Reading analogue values

The Arduino has 6 analogue ports for reading voltages between 0 and 5 volts with 10-bit resolution giving values from 0 to 1023. The second circuit from the left in the photograph in the header of this article, between breadboard columns 30 and 40, shows the simplest analogue demonstration circuit. It uses a 10K ohm potentiometer with the outer pins connected to GND and 5 volts. The central wiper pin is connected via the orange wire to pin A1 on the Arduino. This pin is also called pin 15. (A0 is 14, A1 is 15 .... A5 is 19.)

The following code demonstrates how to read analogue values in Python:

```
#!/usr/bin/env python
# Read values from Analogue pin A1
from nanpy import Arduino
from nanpy import serial_manager
serial_manager.connect('/dev/ttyACM0')
from time import sleep
pot = 15        # Pot on A1 - Analog input

print "Turn the pot - 10 bit ADC input"
for i in range(0, 40):
    val = Arduino.analogRead(pot)
    print val
    sleep(0.3)
```

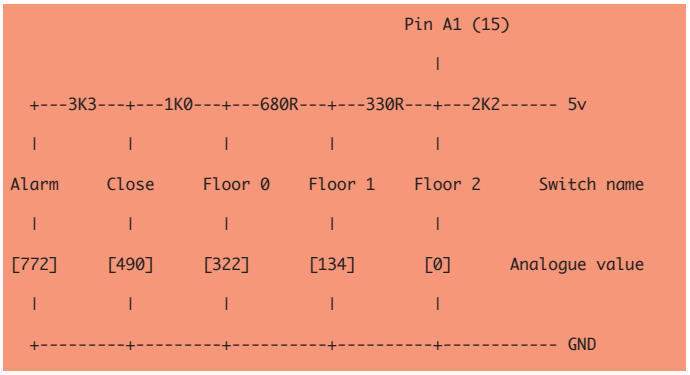
As you turn the spindle on the potentiometer the values change within the range 0 to 1023.

You can connect many different components and devices to the analogue pins: to read temperatures, measure distances, etc.

Once you start building more ambitious projects you will find that you can quickly run out of pins. As you have seen, a liquid crystal display uses up 6 of your digital pins. Imagine that you were building a model lift. You need digital pins to control the motor, an LCD and several LEDs. You also need buttons to call the lift on each floor and others in the cage. One trick is to use a single analogue pin to monitor the 5 switches on the panel in the lift cage.

The circuit below is shown in the photograph in breadboard columns 42 to 61. It has five resistors in series and five button switches connecting the resistor junctions to GND. The circuit is connected to Arduino pin A1 via the yellow wire from the junction between the right most resistors.

The circuit is shown below:



Here the 2K2 ohm resistor acts as a pull up resistor to 5 volts. With no button pressed the reading from the analogue pin is 1023. If the button for floor 2 is pressed the junction is shorted to GND and the reading changes to 0, just like a digital switch. As each button is pressed in turn, from right to left, the resistance to GND increases (0, 330R, 1K1, 2K1, 5K4 Ohms) and the voltage at the main junction changes. Unfortunately, resistors with the same nominal value can vary significantly but still be within their quoted tolerance. It is not really worth

trying to calculate the analogue value for the other switches as each board will be slightly different. It is much easier to re-run the last program a few times with the yellow wire connected to pin A1, press the buttons in turn and record the values for each button. My values are shown in the square brackets and listed in the program.

There is usually at least one 'difficult' button whose value oscillates between two adjacent values. My reading for the 'Close Door' button was either 490 or 489. This problem is easily overcome by testing for a range between upper and lower limits rather than a specific value. There are very large gaps between the button values leaving plenty of room for the 'fudge factor'.

For example: (value > 485 and value < 495) rather than (value == 490).

Here is the basic code to read the switches:

```
#!/usr/bin/env python
# Read 5 switch/resistor array on pin A1 (15)
# Lift cage control panel simulation

from time import sleep
from nanpy import Arduino
from nanpy import serial_manager
serial_manager.connect('/dev/ttyACM0')

pot = 15   # Pot on A1 - ADC input

# You may want to change these values to match
# your board or increase fudge value
floor2 = 0
floor1 = 134
floor0 = 322
close  = 490
alarm  = 772
fudge  = 5

print "Press the buttons"
print "\nCTRL-C to stop program\n"

old_val = -1
val = Arduino.analogRead(pot)

while True:
    try:   # Trapping CTRL-C
        # Has val changed > ADC 'wobble' ?
        if abs(old_val - val) > 3:
            old_val = val
            if val < fudge:
                print"Floor 2"
            if val > floor1 - fudge and val <
floor1 + fudge:
                print"Floor 1"
            if val > floor0 - fudge and val <
floor0 + fudge:
                print"Floor 0"
            if val > close - fudge and val <
close + fudge:
                print"Close doors"
            if val > alarm - fudge and val <
alarm + fudge:
                print"Alarm"

        val = Arduino.analogRead(pot)

    except KeyboardInterrupt:
        print "\nProgram interrupted.\n"
        break

print "*** Finished ***"
```

This is an 'endless loop' program – lifts run all the time. Stopping it with <CTRL>+<C> normally results in a mess of red error messages. Notice how this has been trapped, which results in a 'clean' termination.

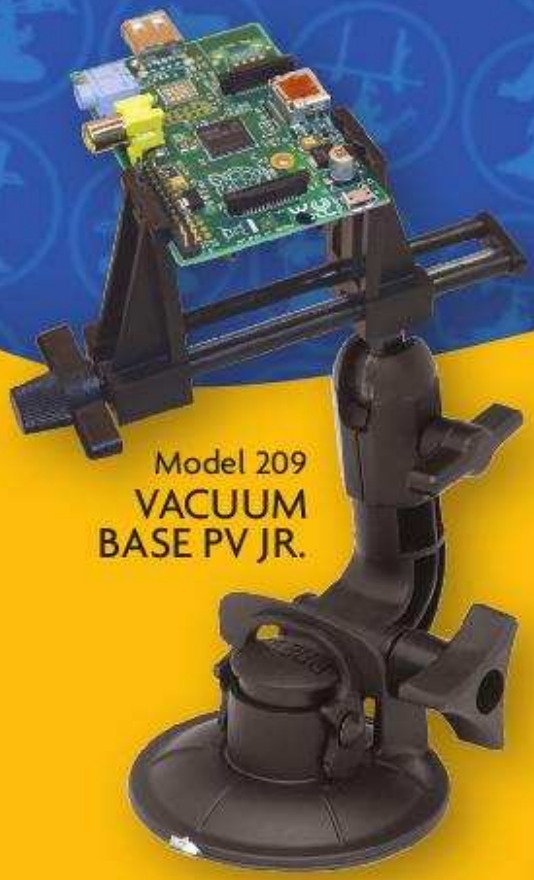
Conclusion

You can contact me with any feedback, suggestions or questions, via email at arduinolink@gmail.com. I enjoy experimenting with hacking projects and robots and hope to encourage others to take up this great hobby (especially in Leicestershire, where I taught computing for 23 years. How about a Raspberry Jam? Lancashire is doing so much more!).

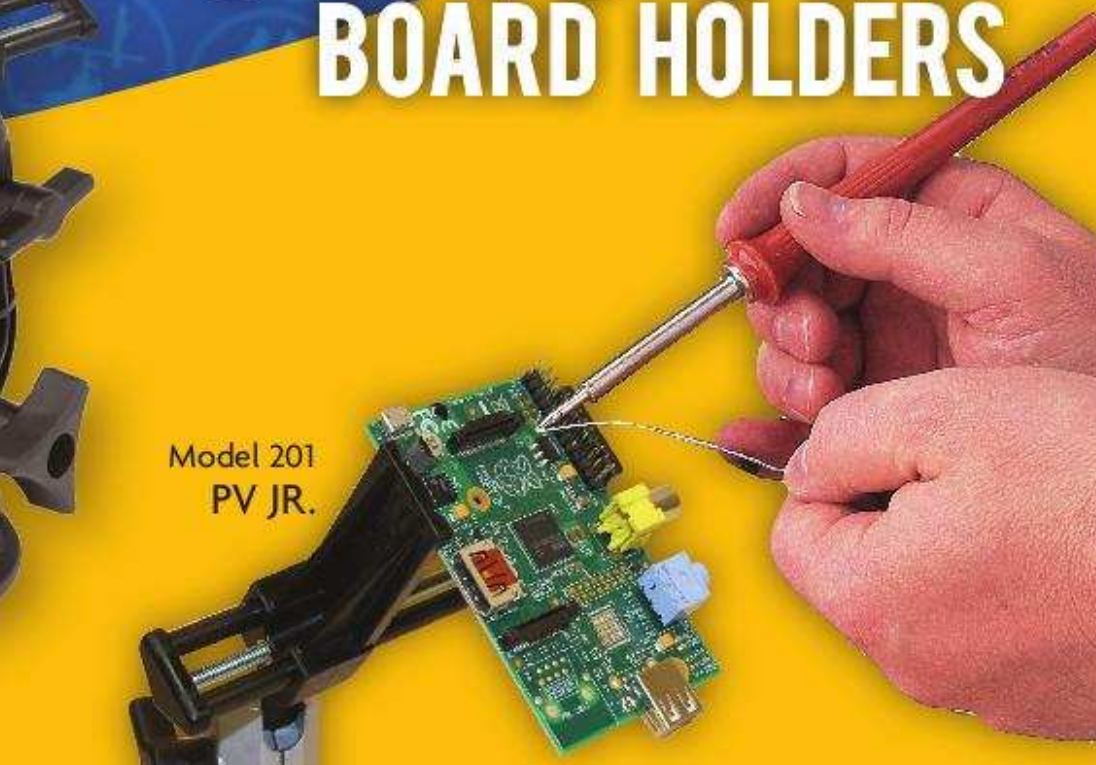
Thanks again to Andrea Stagi for producing Nanpy.

VISIT US ON

WORLD'S MOST VERSATILE CIRCUIT BOARD HOLDERS

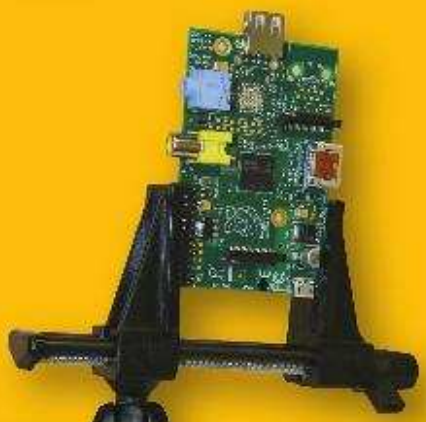


Model 209
VACUUM
BASE PV JR.



Model 201
PV JR.

- Work-holding tools for electronics projects
- Circuit board holders make soldering easy & fun
- Versatile hobby vises for any project



Model 207
VISE BUDDY JR.

PANAVISE®
Innovative Holding Solutions

www.panavise.com
00 1 800 759 7535
7540 Colbert Drive • Reno • Nevada • 89511 • USA

Expand your Pi

Stackable Raspberry Pi expansion boards and accessories

IO Pi

32 digital input/output channels for your Raspberry Pi. Stack up to four IO Pi boards to give you 128 I/O channels.

£16.99

RTC Pi

Real-time clock with battery backup and 5V I²C level converter for adding external 5V I²C devices to your Raspberry Pi.

£9.75

ADC Pi

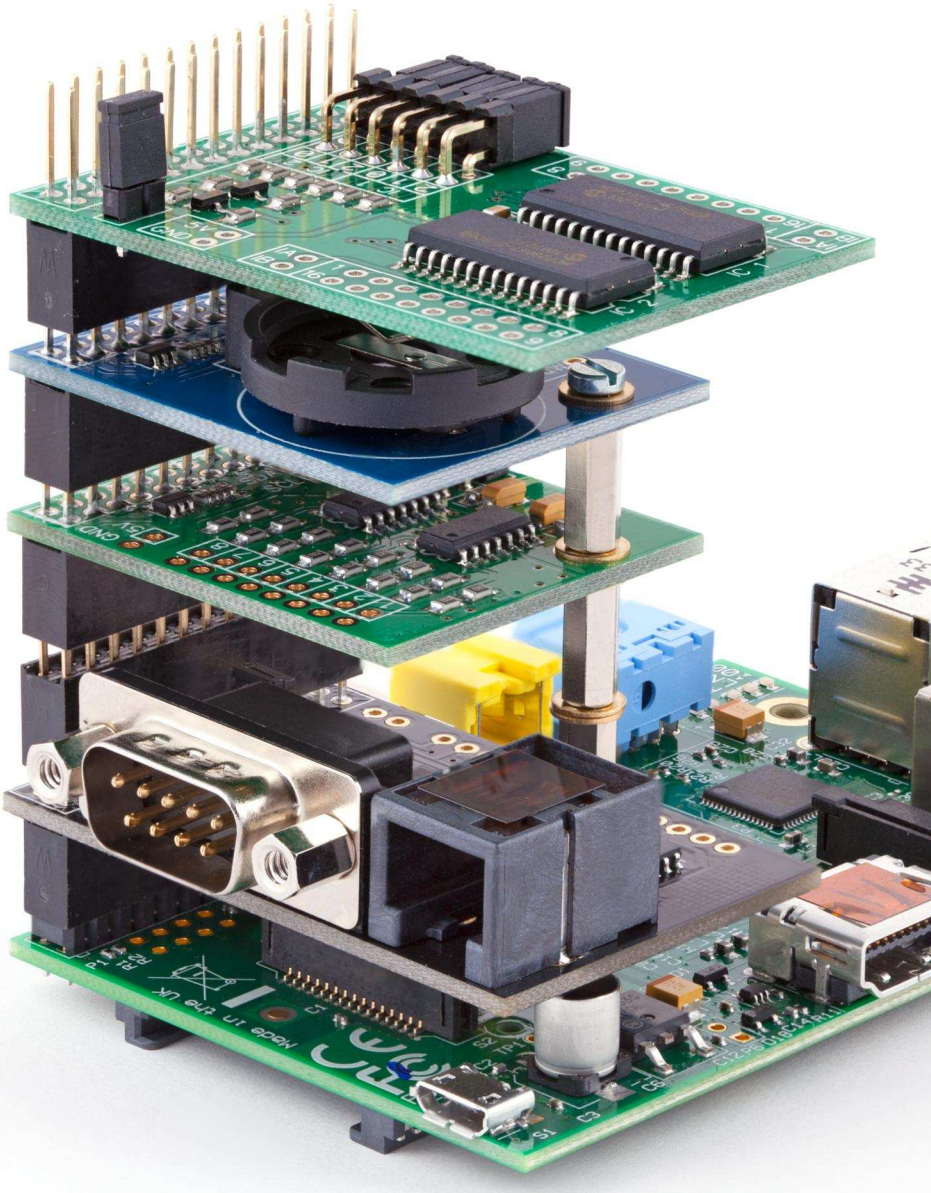
8 channel analogue to digital converter. I²C address selection allows you to add up to 32 analogue channels on your Raspberry Pi.

£17.99

Com Pi

RS232 and 1-Wire® expansion board adds a serial port to your Raspberry Pi. Ideal for the Model A to enable headless communication.

£19.99



AB electronics UK

www.abelectronics.co.uk

Win a selection of Raspberry Pi expansion boards from AB Electronics UK

Enter our competition and you could win a selection of Raspberry Pi expansion boards including an IO Pi, RTC Pi, ADC Pi and Com Pi.

Visit our website for more information.

For your chance to win visit

www.abelectronics.co.uk/magpi/

The MagPi What's On Guide

Want to keep up to date with all things Raspberry Pi in your area?

Then this section of The MagPi is for you! We aim to list Raspberry Jam events in your area, providing you with a Raspberry Pi calendar for the month ahead.

Are you in charge of running a Raspberry Pi event? Want to publicise it?
Email us at: editor@themagpi.com

Cambridge Raspberry Jam

When: Saturday 21st September 2013, 12.30pm until 6.00pm
Where: Institute of Astronomy, Madingley Road, Cambridge CB3 0HA

This event is currently sold out but a waiting list exists. Further information available at http://cambridgeraspberry-es2.eventbrite.com

CAS Teachers North London Hub Meet

When: Wednesday 25th September 2013, 3.30pm until 5.30pm
Where: Central Enfield City Learning Centre, 113 Parsonage Lane, Enfield, UK

A session for teachers to share their experience of using Raspberry Pi's. Further information: http://pienfieldclc-es2.eventbrite.co.uk

At-Bristol Raspberry Pi Boot Camp

When: Saturday 28th September, 10.30am until 4.30pm
Where: At-Bristol Science Centre, Anchor Road, Harbourside, Bristol, BS1 5DB, UK

The event will include live demonstrations, workshops and hands on tutorial sessions. Sponsored by ARM who will also be in attendance. http://pibootcampsept2013.eventbrite.com

The Raspberry Pi at CERN

When: Saturday 5th October 2013, 9.30am until 4.30pm
Where: CERN Microcosm, Route de Mayrin, 1211 Geneva, Switzerland

The event will include live demonstrations, motivational talks and a hands on tutorial session. Guest speakers from Google and Ibisense will also be present. http://cern-raspberrypi.eventbrite.fr

2 - Control flow and basic variables

DIFFICULTY : MEDIUM

Vladimir Alarcón & Nathaniel Monson
Guest Writers

The main goal of this second article is to give you more elements of the Java programming language, so you'll be able to write much more powerful programs. The first article in the series appears in Issue 14. It might prove helpful to read over the C cave articles too, since basic Java syntax is in many cases the same as C.

In this article, I first explain in detail Java control-flow statements. Then, I focus on numbers, strings, and booleans, and I also include examples that combine them. With all these new elements you can write complex logic to your code. Well... let's go for it!

Control-flow statements

In the first article in Issue 14 we looked at two control flow statements: `if` and `while`. I know you already know, but let me remind you anyway. An `if` condition executes the following statement only when the condition in between parenthesis is true. For example:

```
if (a > 3) { c = c + a; }
```

This example adds `a` to `c`, only when `a` is greater than 3. The section in between the round parenthesis is called the condition and must evaluate to true or false, i.e. it's a boolean expression (I'll explain booleans later in this article).

An `if` statement can, additionally, include a section to be executed when the condition is false. For example:

```
if (score > 50) {
  System.out.println("You won! :)");
} else  {
  System.out.println("You lost. :(");
}
```

The other statement we saw before was `while`. The `while` statement executes the following statement zero, one or many times, as long as the condition in it remains true. For example,

```
int f = 10;
while (f < 20) {
  System.out.println(f);
  f++;
}
```

will print all the integer numbers bewteen 10 and 19. It won't print 20, because at that point the condition will not be true anymore. Ah... did you notice the fourth line? Good catch! The two + signs after a variable increase its value. That is a short way of typing:

```
f = f + 1;
```

Similarly,

```
f--;
```

decreases the value of the variable by one and assigns the

result to the varaible. It is equivalent to:

```
f = f - 1;
```

Now, let's look at the `for` statement. `for` is similar to a `while` statement, but it combines the condition, the starting point, and the change statement into one line. The numbers between 10 and 19 can be also written using `for`:

```
for (int f = 10; f < 20; f++) {
  System.out.println(f);
}
```

Compare both examples. Yes, I would agree that a `for` statement looks more compact than a `while`. A `while`, though, can be easier to read some times.

In a `for`, the section between parenthesis is divided in three parts: the first one is executed only once when starting; the second one is the condition to check on every cycle; the third one is to be executed at the end of every cycle.

Now open your editor and try the following program:

```
public class Countdown {

  public static void main(String[] args)
     throws InterruptedException {
    for (int t = 10; t >= 0; t--) {
      System.out.println("-> " + t);
      Thread.sleep(1000);
    }
    System.out.println("-> Fire!");
  }

}
```

Compile it using the `javac` command and run it using the `java` command (see previous article for details). This program will count down from 10 seconds down to zero and then prints "Fire!". Do you see how the value of the variable `t` changes? We use the double minus sign here to decrease its value. Also, did you notice the line with `Thread.sleep(1000);`? That line makes the program wait for 1 second (1000 milliseconds) each time, so the numbers don't show up all at once. To use the `Thread.sleep` method, the `for` loop is within a `throws` statement. Trust me on the `throws` statement for now; I'll explain exceptions later on.

Challenge #3: Create a new program `Countup.java` that will count from 1 to 20, using a one second sleep. Then when reaching 15, show a message saying "Five to go...".

Last but not least, let's look at the `switch` statement. A `switch` checks for the value of a variable and allows to execute different code for different values of it. For example:

```
import java.util.Random;

public class CardinalDirection {

  public static void main(String[] args) {
    Random r = new Random();
    int dir = r.nextInt(4);
    switch (dir) {
    case 0:
      System.out.println("North");
      break;
    case 1:
      System.out.println("West");
      break;
    case 2:
      System.out.println("South");
      break;
    default:
      System.out.println("East");
      break;
    }
  }

}
```

In this example, an integer number is randomly chosen between zero and three (four possible cases). The interesting part is that the number is not shown as a number, but as a cardinal direction. Each case statement specifies what to do if the variable has the specified value. Please note that I don't use a case for the number three, but I use a default statement. It's always a good practice to use a default statement, so any unspecified value will be shown anyway.

Well, that's it for now with control flow statements. Of course there are other more exotic ones like `continue` and `do-while` but they are less used and I'll leave them for you to study.

Let's now move on to a totaly different subject.

A little bit about numbers

Sooner or later you'll need to do some maths, so here are the basics. In Java every numeric variable must be declared as one of the types built in the language itself. Java provides four integer types (without decimal places), two floating points (with decimal places) and two very high precision types. When using numbers you need to declare the variable once, and then you can assign a value to it. For example:

```
int trees; // declaration
trees = 17665785; // assignment
```

You can assign and reassign the value of the variable many times, but you must declare the variable only once. These two lines can be combined as:

```
// declaration & assignment
int trees = 17665785;
```

By the way, did you notice the //? The double slash allows you to write a comment. Anything in the line after it is ignored by Java, so you can add notes or reminders. These are typically very useful several months later, when you need to look at your program and you don't remember why you did something.

Now, if you want to use integer numbers you have four options:

```
byte:  -128 to 127
short: -32768 to 32767
int:   -2,147,483,648 to 2,147,483,647
long:  -9,223,372,036,854,775,808
     to 9,223,372,036,854,775,807
```

You'll probably use the third one (int) most of the time, unless you really need big numbers or small numbers. On the other hand, if you want decimal numbers (floating point) you have two options:

```
float:  ±1.401298e-45 to ±3.402823e+38
double: ±4.94065645841246e-324
     to ±1.79769313486231e+308
```

Normally, if you want to do a lot of maths you will probably prefer the type double (15-digit precision) over float (7-digit precision).

The four maths operators are written as +, -, *, and /, and you can use parenthesis to group sections as you need. The following program shows examples of maths operations:

```
public class Numbers {

  public static void main(String[] args) {
    int x = 5;
    System.out.println(7 + x * 3); // 22
    System.out.println((7 + x) * 3); // 36
    System.out.println(7 / 3); // 2
    // 1 (remainder)
    System.out.println(16 % 3);
    // 2.33333333
    System.out.println(7.0 / 3.0);
    System.out.println(3.14159265 * x * x);
  }

}
```

Notice that multiplications and divisions have priority over addition and subtraction, but these priorities can be changed using parentheses.

Need more maths functions? Well, there is a class called `Math` that provides many maths functions. For example a more complex formula like: $f = \frac{1 - x^3}{\sqrt[4]{y - \pi}}$

can be written as:

```
double f = Math.abs(1.0 - Math.pow(x, 3.0)) /
   (4.0 / 3.0 * Math.sqrt(y - Math.PI)));
```

When dealing with floating point calculations make sure that you write literal numbers with a decimal point: for example, instead of 3 write 3.0.

The following program adds all numbers from 1 to 10 that cannot be divided by 3:

```
public class Sum {

  public static void main(String[] args) {
    int total = 0;
    for (int n = 1; n <= 10; n++) {
      if (n % 3 != 0) {
        total = total + n;
      }
    }
    System.out.println("Total: " + total);
  }
}
```

Running this program will show the numbers 1, 2, 4, 5, 7, 8, and 10, with a total of 37.

Challenge #4: Change the previous program to also skip the numbers that can be divided by 4. With your change the program should now show 1, 2, 5, 7, and 10, with a total of 25. Tip: you'll need to add a second if statement.

If you are curious, you can find the full list of math functions at:
http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

But... what if 15 decimal places is not enough for you? Well... `BigInteger` and `BigDecimal` come to the rescue. These classes can handle very high precision. A hundred decimal places, or a thousand, or a million if you want. Really? Yes, of course! For example:

```
import java.math.BigDecimal;

public class HighPrecision {

  public static void main(String[] args) {
    BigDecimal a = new
BigDecimal("2.7182818284"
     + "5904523536028747135266249775572"
     + "4709369995"); // 50 decimal places
    BigDecimal b = new
BigDecimal("3.1415926535"
     + "8979323846264338327950288419716939937"
     + "5105820974944"); // 60 dec. places
    System.out.println("e*pi="
     + a.multiply(b));
  }

}
```

Run this program and see for yourself. You'll get at least 50 digits of precision with it.

A little bit about strings

You have seen strings many times so far in examples given in these tutorials. A string is a sequence of letters, numbers and/or symbols (all called characters). Usually you see them as literals like `"Hi John!"`. However, you can use them in more complex expressions, to search within them, to get sections of them, or when building bigger strings from several parts. For example, the + sign builds (concatenates) several strings into a single one, like in the example below:

```
String qty = "50";
String message = "I found " + qty + " books.";
```

If you want to get a part of a string you use the `substring` method. Every letter (character) on a string has a position. The first one is 0, then 1, then 2 and so on. For example, to get the word `"house"` (position 4 to 8 in the following example) you'll write:

```
String line = "The house is blue.";
String word = line.substring(4, 8 + 1);
```

Now, if you want to find where the word `"house"` is in the string you can use `indexOf`:

```
String line = "The house is blue.";
int position = line.indexOf("house"); // 4
int red = line.indexOf("red)"; // -1
```

Notice that `indexOf` is written with an upper case `"O"`. In the last case, the word `"red"` is not found in the line of text and, therefore, Java returns -1.

Also, to get the length of the string you use the method `length()` and to get a single character in a string you use the method `charAt()` as shown in the following example that prints a string letter by letter.

```
public class LetterByLetter {

  public static void main(String[] args) {
    if (args.length == 0) {
      System.out.println("Please type"
       + " a word.");
    } else {
      String word = args[0];
      for (int i = 0; i < word.length(); i++){
        System.out.println(word.charAt(i));
      }
    }
  }

}
```

Once compiled using `javac`, run the program specifying a word you choose on the command line. For example:

```
java LetterByLetter telephone
```

As I said in the previous article, the variable `args` represents the parameters you type in the command line (`"telephone"` in this case). See how the program tells you if you forget to type a word.

If you look at the for statement it uses both methods to first get the length of the string and then to get each character one by one.

Anyway, strings have many more useful methods. The full list can be found at http://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Challenge #5: Change the previous program to show every letter in upper case. Tip: once you get the whole word, change it immediately to uppercase using one string method. Look for this method in the web page shown above.

A little bit about booleans

Booleans are variables that can only have two values: true or false. They are very useful to use as indicators, or as conditions used inside the `if`, `while` or `for` statements. For example, let's declare a boolean variable;

```
boolean painted = false;
```

Now, if you want to, you can change the value as in any other variable:

```
painted = true;
```

or you can use it in a `while` statement:

```
while (painted) {
  // do something
}
```

You can make complex expression using booleans with `!` (meaning NOT), && (meaning AND), and `||` (meaning OR). A `!` returns the opposite of a value. An && returns true only if both sides are true, and a `||` returns true if either side is true. For example:

```
boolean a = true;
boolean b = false;

boolean c = !a; // false
boolean d = a || b; // true
boolean e = a && b; // false
boolean f = (!a && b)  || a; // true
```

Also, numbers and strings can be compared to return a boolean. That boolean, in turn, can be used as a condition for an `if`, `while` or `for`, as you may have noticed in several examples before. For example:

```
int total = 156;
if (total > 100)  {
  // do something
}
```

The section total > `100` is a boolean expression, since its value as a whole is a boolean (either true or false). To compare two numbers you can use six operands:

```
==        is equal to
!=        is different from
>         is greater than
<         is less than
>=        is greater or equal than
<=        is less or equal to
```

All of them return a boolean, and as such you can combine them in any boolean operation as you wish. In the following example:

```
int sides = 6;
boolean painted = false;
if (sides >= 4 && !painted) {
  // paint it!
}
```

you can see a more complex condition.

A final note

I know I covered a lot of topics in this second article. It's a little bit more complex than the first one, but my goal was to give you as many tools as I could as soon as possible. With them you can now write basic and not so basic programs. You now have much more power in your hands.

On the next article we will cover Java methods, classes and objects, as well as the extensive Java Class Library included in Java ...and we'll write a program to generate a random map.

SEPTEMBER COMPETITION

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

This month there is one MASSIVE prize!

The winner will receive a new Raspberry Pi 512MB Model B, an exclusive Whiteberry PCSL case, 1A PSU, HDMI cable, 16GB NOOBS memory card, GPIO Cobbler kit, breadboard and jumper wires!

For a chance to take part in this month's competition visit:
http://www.pcslshop.com/info/magpi

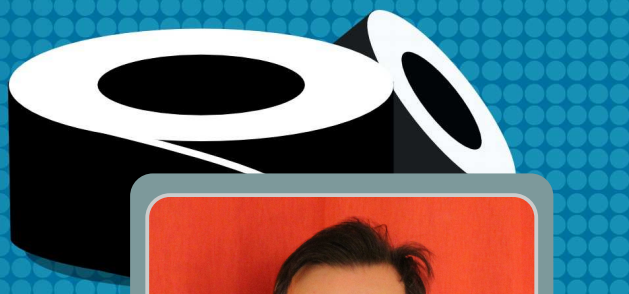
Closing date is 20th September 2013. Winners will be notified in next month's magazine and by email. Good luck!

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

July's Winner!

The winner of a new 512MB Raspberry Pi Model B plus an exclusive Whiteberry PCSL case, 1A PSU, HDMI cable, 16GB NOOBS memory card, GPIO Cobbler kit, breadboard and jumper wires is **Ellis Howe (Peterborough, UK)**.

Congratulations. We will be emailing you soon with details of how to claim your prizes!



3 - Strings & arithmetic operations

DIFFICULTY : INTERMEDIATE

W. H. Bell

MagPi Writer

Challenge solution

Did you manage to solve the problem in Issue 12? Here is a solution:

```
#!/bin/bash
failed=0
if [[ $# != 4 ]]; then
  failed=1
elif [[ $1 != *"x"* ]]; then
  failed=1
fi
if [[ $failed == 1 ]]; then
  echo "Usage: $0 <f(x)> <min> <max> <step>"
  exit 1
fi
fcn=$1
min=$2
max=$3
step=$4
python <<EOF
import math
from math import *
x = $min
while x < $max:
  print "%d %f" %(x,$fcn)
  x = x + $step
EOF
```

Try running the solution by typing,

```
./solution002.sh "pow(x,2)" -4 4 0.1
```

This will print out the values of the square of x between -4 and 4, in steps of 0.1.

String operations

While it is possible to use command evaluation and sed to replace string elements, Bash also contains functionality for some simple operations.

```
#!/bin/bash
with="avec"
str="Raspberry pi with Linux"
str_fr="${str/with/$with}"
str_fr="${str_fr#Raspberry pi}"
str_fr="tarte aux framboises$str_fr"
str_fr="${str_fr%Linux}Raspbian"
echo "$str_fr"
```

As for the previous examples, save the script in a text file and make the file executable. Then run the script and look at the output. The first line of the script defines a variable to hold the French word for with. Then the `str` variable is operated on by the commands that follow. The `${//}` command includes the variable name, the string to remove, and then the string to replace the former string. The string to replace is optional. The command `${#}`, takes the variable name and then the string prefix to remove. The command only removes the start of the string. Next "Raspberry Pie" in French is appended to the string `str_fr`. Finally, the suffix is removed using the `${%}` command. The `${%}` command takes the variable name and then the string to remove. The

match is only made with the end of the string. Try moving the `echo` command to an earlier line in the script to see the effect of each step, or try to change the commands themselves.

The Bash native string functions are rather limited, but there are other functions available via `expr` that can easily be added to scripts.
In this script, a space is used within the string to encourage the correct use of quotes. The `distro`

```
#!/bin/bash
distro="Raspbian Linux"
search=" "
offset=$(expr index "$distro" "$search")
echo "String: \"$distro\""
echo -n "String length: ${#distro}"
echo ", found space at: $offset"
echo "Substring: \"${distro:$offset}\""
```

string is declared. Then the `expr` command is used to search for the space within the string and find its position. The return value from the `expr` command is stored in the `offset` variable. The `offset` and the substring up to the offset is then printed, along with the length of the string. Notice that the `-n` in the `echo` command does not append a newline character. More `expr` commands can be found in the `expr` man page. Try typing,

```
expr man
```

to view the manual page. Type `q` to leave the manual page.

Arithmetic operations

Bash provides simple integer mathematical operations. These can either be directly accessed or accessed through the `let` function.

```
#!/bin/bash
i=1
j=4
i=$((++i*j))
echo $i
```

In this example script, two integer variables are defined. Then the value of `i` is incremented before it is multiplied by `j`. Notice that integer mathematical operations are within the `$(())` brackets. The operations will work as hoped with or without a dollar sign in front of the variable names.

```
#!/bin/bash
a=3
b=2
c=$(($a/$b))
echo "$a / $b = $c"
c=$(($a*$b))
echo "$a x $b = $c"
c=$(($a+$b))
echo "$a + $b = $c"
c=$(($b-$a))
echo "$b - $a = $c"
```

This time the script demonstrates the limitations of integer operations by including a division. The division rounds down, since the floating point part of the number is lost. The behaviour is similar for the `let` command:

```
#!/bin/bash
a=9
b=4
let a--
let "a*=2"
let "a/=b"
echo $a
```

The `let` command uses variable names. When the arguments are more complex, quotes are needed around the arguments. The result is automatically put back into the Bash variable.

If floating point numbers are required, the basic calculator (`bc`) shell can be used to perform simple calculations:

```
#!/bin/bash
radius="2.5"
pi="3.14159"
area=$(echo "$pi*$radius^2" | bc)
echo $area
```

Try using the `bc` manual page and test out some more of the functions. If `bc` is not good enough, then try some inline Perl or an embedded Python script.

Challenge problem

A program writes output files into different subdirectories, but the output files themselves have the same name. Write a program to rename the output files using a number suffix.

XML for the Raspberry Pi: Part 2

DIFFICULTY : INTERMEDIATE

John Shovic

Guest Writer

Introduction

This series of articles discusses the use of XML in applications for the Raspberry Pi. Part One introduced XML and the format of its data structures. In Part Two we cover building and parsing XML in Python, while in Part Three we will show how XML is used as a communications protocol for a client / server application, RasPiConnect. RasPiConnect is an iPad/iPhone app that connects and displays information for any number of Raspberry Pi's via a defined XML interface.

What do we mean by parsing?

Parsing refers to the syntactic analysis of the XML input into its component parts in order to facilitate executing code based on the result of the analysis. In other words, the program is "reading" the XML to find values that it is looking for, paying attention to proper syntax and form. XML syntax includes a nested hierarchy of elements. This means that each level of the hierarchy is included as a fully enclosed subset of the previous level. In our example below, each object <XMLCOMMAND> is fully enclosed ("nested") in the <XMLObjectXMLRequest>. You can extend this nesting as far down as you like. When you write parsing code this nesting usually results in `for` loops in Python iterating through all the objects at a level in the hierarchy.

Options for Parsing XML in Python

There are many different packages for parsing XML in Python. We will be using xml.etree.ElementTree. ElementTree is a simple to use, fast XML tree library built into Python. It is somewhat limited in features, but for straightforward XML message parsing it is hard to beat.

What do you need to know about ElementTree? Very few commands are needed to parse simple XML. These few will be illustrated below.

Python Example Code

```
import xml.etree.ElementTree as ET

incomingXML = """
<XMLObjectXMLRequests>
 <XMLCOMMAND>
  <OBJECTSERVERID>W-1</OBJECTSERVERID>
  <OBJECTNAME>StatusWebView</OBJECTNAME>
  <OBJECTTYPE>1</OBJECTTYPE>
  <OBJECTID>7</OBJECTID>
 </XMLCOMMAND>
 <XMLCOMMAND>
  <OBJECTSERVERID>M-2</OBJECTSERVERID>
  <OBJECTNAME>Processes</OBJECTNAME>
  <OBJECTTYPE>64</OBJECTTYPE>
  <OBJECTID>0</OBJECTID>
 </XMLCOMMAND>
```

```
</XMLObjectXMLRequests>"""

root = ET.fromstring(incomingXML)
print incomingXML

# iterate through all the values
for element in
root.findall('XMLCOMMAND'):
    print 'XMLCOMMAND'
    print 'OBJECTNAME:',\
        element.find('OBJECTNAME').text
    print 'OBJECTTYPE:',\
        element.find('OBJECTTYPE').text
    print 'OBJECTSERVERID:',\

element.find('OBJECTSERVERID').text
    print 'OBJECTID:',\
        element.find('OBJECTID').text
```

Setup the ElementTree data

After the import of the ElementTree code and writing the XML to a string (note: You could be reading this from a file or a web request), we first set up the root of the XML hierarchy. The root of this XML code is <XMLObjectXMLRequests>.

Iterate through the list

We know from looking at the XML file, that <XMLObjectXMLRequests> consists of a number of <XMLCOMMAND> objects. We use a for loop to do this (each element inside the root is a <XMLCOMMAND> object) using the ElementTree command `findall` (finding all XMLCOMMAND objects in this case).

Parse the individual items

In the interior of the `for` loop, we now parse the individual elements of the <XMLCOMMAND> object. Here we use the ElementTree `element` command with the text attribute. Note that the <XMLCOMMAND> elements are not in the same order! XML does not care if elements on the same level are in any particular order. Furthermore, it is not guaranteed that the first <XMLCOMMAND> element will be the first one retrieved by ElementTree.

Expected elements can be missing from objects. In the case of missing elements in Python (using ElementTree) you absolutely must use an `if` statement to deal with the missing element. If you do not then you risk causing a Python exception when operating on the returned value as ElementTree returns a None and not a valid value. If you are using strings as values, you will probably want to set your string variable to a "" (empty string) rather than allowing it to be set to a Python None. This is a very common mistake in writing ElementTree code.

```
if (element.find('XXXX').text == None):
                #do something
```

Uses for XML in Python programs

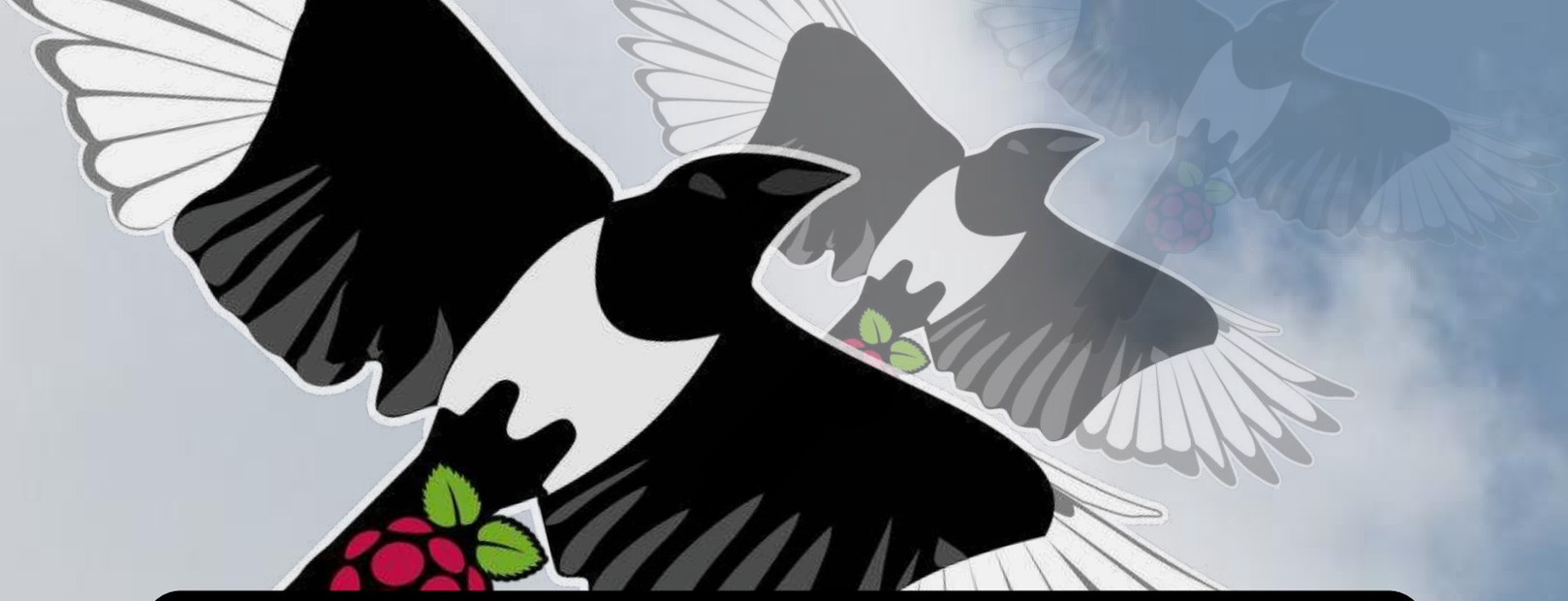
XML is used extensively in the software industry, ranging from HL7 messages in Healthcare, Simple Object Access Protocol (SOAP) for client-server information exchange, and even XML is used in Microsoft Word files. The key advantages of using XML are cross system use, readability, expandability and the ability to edit the XML in a text editor.

Programmers often use XML to read and write configuration files to the disk, speeding debugging and development. This makes it easier to set up test suites for programs as you can read the same XML structures from the disk as you would send across the Internet in a web request. The expandability of XML allows you to add new parameters and structures in your Python programs while maintaining backwards compatibility. Part Three of this series will show how this is done in Python.

Conclusion

XML is wordy and as a result uses a fair bit of disk space to store and memory to process. In the Raspberry Pi world and across the Internet this generally does not matter. However, in microcontrollers such as the Arduino, RAM memory space is at a premium, so a more "dense" and simple protocol such as JSON is more appropriate. If disk space is at a premium, XML will compress extremely well because of all the duplication of keywords and descriptions.

XML is easy to read, parse and debug for beginners and seasoned programmers alike.



Feedback & Question Time

Just received my magazines and binder today. Wow! They are beautiful. This whole project has been great. I had selected the Signature Kit and am I ever so glad. The hardware kit came some time ago and the whole thing was top quality. It meant a lot to have those books signed by Liz and Eben. Congratulations for a well executed project.

Tony Guerich

Just found @TheMagP1 - loads of Raspberry Pi info, lessons, resources and projects.

Jim Leese

I love The MagPi! I remember me as a young nerd - I was reading a technical magazine full of source code, I learned a lot by copying those sources.

Fabrizio

Just received my binder and mags - they look great. Thanks for all the effort you have put into this. And you just know that we all want Volume 2 now don't you :-)

Mark Pearson

I am enjoying the Raspberry Pi and am trying to learn some programming. I want to write my own program with some graphics. The MagPi is great - it has helped me with my Raspberry Pi experience.

ab

First I wanted to congratulate you and the staff for putting together a very well done publication. I've downloaded each one and I'm in the process of reading them along side my Raspberry Pi and I do try the programs that you have. All in all a great job and you've made it easy for anyone to learn about the Raspberry Pi.

I do have one comment though. I still belong to the old school and I like to print each issue out and read it and make notes in the margins. I like the colours chosen, but would ask that you reduce the amount of solid colours especially the black bars at the top and bottom of each page. Having to print them only uses up more ink which become an expensive process for each issue.

Richard

[ed:The PDF is a little expensive on printer ink, so we have made hard copies up to issue 9 available as bound magazines. These are available from our licensed retailers who are listed on our website. We will be printing issues 10+, but there is a large workload involved in getting them to a stage where they can be sent to the printers, so we are slightly behind.]

288 pages of goodness just arrived! The mags look awesome! Love the binder and stickers! It's been a bit of a wait but it's definitely been worth it! The whole bundle is fantastic and the magazine quality is perfect, they look and feel like a premium magazine you would buy from a newsagents.

You can see right away the hard work that has gone into these and it is very much appreciated.

THANK YOU!! to the whole MagPi team!!

Time to crack on and start reading!

Nial Pearce