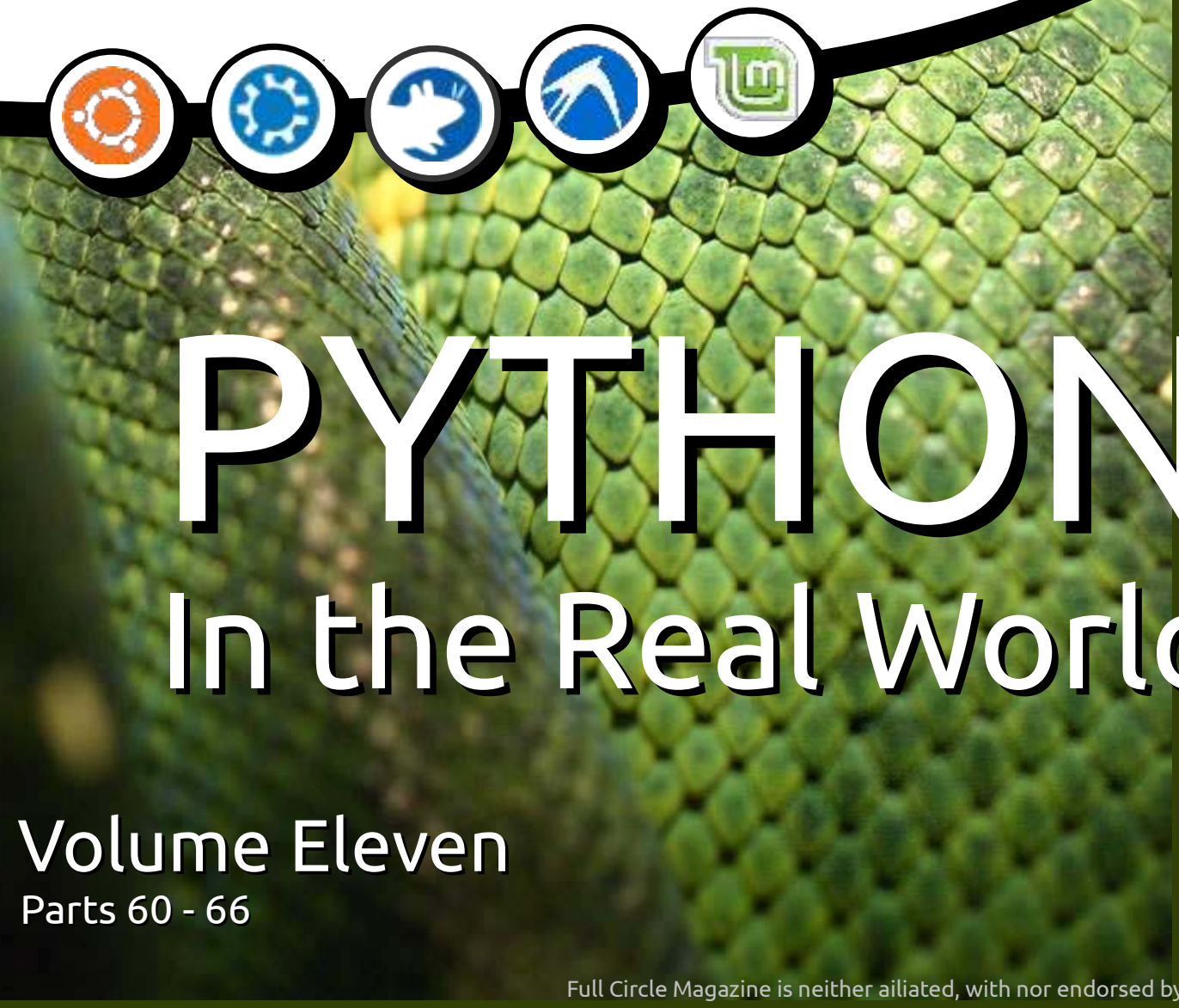
Full Circle

THE INDEPENDENT MAGAZINE FOR THE UBUNTU LINUX COMMUNITY

PROGRAMMING SERIES SPECIAL EDITION

PROGRAMMING SERIES SPECIAL EDITION



PYTHON

In the Real World

Volume Eleven

Parts 60 - 66

Python In The REAL World

Welcome fellow pythoners. As the kids here in the central parts of the U.S. say, "What's Shakin' Bacon?" I'm not exactly sure what that's supposed to mean, but I assume it's a good thing.

You might notice the new header. I decided that I've taught you all the basics of Python that I can for "general" programming, so now we are going to delve into using Python to talk to other types of computers and controllers, like the Raspberry Pi and the Arduino micro controller. We'll look at things like temperature sensors, controlling motors, flashing LEDs and more.

This issue we will be focusing on what we'll need to do this and focus on a few of the projects we will be looking at in the future. Next issue, we will start the first project.

One of the things we will talk about next time will be the Raspberry Pi. The Pi is a credit-card sized computer that natively runs Linux on an SD card. Its output goes to your TV set via HDMI. It also has an Ethernet connection for Internet access.

You can find out more at the official site https://www.raspberrypi.org. If you want to follow along with the projects, you will need a Pi, SD card, Keyboard, Mouse, a 5volt DC power supply like the ones on modern cell phones, and access to an HDMI monitor or TV. Eventually, you should also consider getting a breadboard and some connecting wires for when we start to interface to the outside world. You can find any number of places that sell the Pi on the Internet. Here in the U.S., we can get them for around $35.

One other thing about the Pi is that it provides access to a series of pins that support GPIO (General Purpose Input/Output). Basically, this means that you can write programs that will send signals to the output pins and read the signals from the input pins. This can be used to interface to things like LEDs, sensors, push buttons, etc. Many people have made home automation systems, multiple processor systems (by linking 40 or so Pi computers together to emulate a supercomputer), weather stations, even drones. So you can imagine that the possibilities are endless. That's why I decided to start with it for this series of articles.

After a while, we will begin to work with the Arduino, which according to the official website (https://www.arduino.cc): "*Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects*".

Once again, this is an exciting device to work with. In this part of the series, we will look at talking to the Arduino, first in its native scripting language, and then in Python and eventually interfacing the Pi with the Arduino.

I know this month's article is fairly short, but I've been doing poorly health-wise, so I'm saving my strength for the next article. Until then, grab some electronics and get ready for fun!

Greg Walters is owner of RainyDay Solutions, LLC, a consulting company in Aurora, Colorado, and has been programming since 1972. He enjoys cooking, hiking, music, and spending time with his family. His website is www.thedesignatedgeek.net.

Python In The REAL World

Welcome back to the new direction of my Python series. In case you missed last month, I am changing the direction of this 5 year series from teaching programming in Python to what is called Physical Computing using Python. When you see the phrase 'Physical Computing', think of buttons, LEDs, motors, sensors (temperature, humidity, motion sensors, barometric sensors, etc.) and more. The reason I decided to do this was that after 5 years, I thought I had shown pretty much everything that you needed for "normal" computing, so let's focus on what I consider the future of small computer programming and microcontrollers.

This month, I will be going over selecting a Raspberry Pi (yes there are more) that will fit your goals, installing an operating system onto the SD card and starting the RPi for the first time with the new OS.

Next month, we will start learning to respond to switches and control LEDs. In future articles, we will be interfacing with sensors and the Arduino micro-controller.

A BRIEF HISTORY OF THE RPI

Much of this information comes from the official Raspberry Pi website (http://www.raspberypi.org) and my memory of buying my first RPi. When the Raspberry Pi first came out, there were two models – Model A+ and Model B+. The decision tree was fairly easy since the two different versions fit a "simple or full feature" mindset, as you can see in the gross details presented below (They are now called RPi 1 Models)…

Raspberry Pi Model A+	Raspberry Pi Model B+
256 MB Ram	512 Meg Ram
One USB Port	4 USB Ports (original Model B only had 2 ports)
40 GPIO Pins (original Model A only had 26 pins)	40 GPIO Pins (original Model B only had 26 pins)
No Ethernet Port	Ethernet Port

In February 2015, both of those models were superseded by the RPi 2 Model B. It shares a good deal with the RPi 1 B+, but has a 900 MHZ Quad-core ARM Cortex-A7 CPI and 1GB Ram.

You can find various models of the RPi at any number of web retailers. My humble suggestion is to get the RPi 2 Model B if you can afford the difference in the price between the P1 Model B (it shouldn't be that much of a delta). Any of the code that we create in the next few articles should easily work with any version of the RPi.

While you are searching the web for your RPi, you will see various kits and add-on modules like cameras, servo controllers, motor controllers and so on. At this point, the add-ons won't be needed, but we might use some in the future, so if it is something that you are interested in use your own judgment. As to the kits, here are some things you should consider before you invest in the "ultimate kits". In the next few articles, we will need :

- A Raspberry Pi computer.
- A power supply. For the P1 versions, a 5 VDC 1-1.2 amp cell phone charger with a micro USB connection (normal for many smart phones today) will work well. For the P2 version, I strongly suggest that you get a power supply that has an output of 5 VDC 2.5 amp power supply with a micro USB connector.
- A USB Keyboard and Mouse. While many places offer very small keyboard/mouse combos, for programming work and "normal" computer use, you will want a full size version of both. You can move to the small wireless versions later on if you decide to use the RPi for

other uses like a multimedia centre or expanded home automation. Normally when I work with the Pi, I use a VNC server on the Pi and a VNC client on my linux machine, so I don't have to have multiple keyboards and mice on the top of my desk.

• A 4-8 GB SD Card that is Class 10. Versions P1 A and B used SD cards. P1 Model B+ and above have switched over to a Micro-SD card only support. Keep this in mind when buying a specific version. Of course you can use a bigger card. Officially they say that testing has been done with 32 GB cards and don't see many issues with most of the larger cards. Please Be careful when buying SD cards, since they are not all created equal. Just because a cheap card is marked "Class 10" doesn't actually mean that it is going to work like a more expensive card.

• Some sort of Internet connection, either USB Dongle or Ethernet cable.

• A HDMI monitor/television for output and HDMI cable. If HDMI is not available, the P1 A and B versions provide a RCA Composite Video out and 3.5mm Audio Out connector. The P1 B+ version and later have done away with the RCA Composite Video connector and has replaced it with a 3.5mm jack that combines audio and video in one. You would need a 3.5 mm to 3 RCA connectors to connect to an older TV.

• Speakers or headphones (unless the monitor or device you are using supports HDMI audio).

While this is the "minimum" requirement list for this article, for our first project you SHOULD have the following items available…

• Breadboard – The breadboard will be needed to start working with add on discrete components like LEDs, resistors, switches, etc. without having to do any soldering.

• GPIO interface board (header) and Ribbon cable. This will connect the GPIO pins from the RPi to the breadboard. Check out http://sparkfun.com or http://www.Adafruit.com for this item. The item you will want to look at from Adafruit is called "Pi T-Cobbler Plus". Note that this particular item will NOT work with the RPi V1A or B. It will only work with the later versions. It is currently about $8.00 U.S.. If you are using a model A or B, you should get "Pi T-Cobbler" which is about $7.00 U.S. If you are looking at SparkFun, their item is called the "Pi Wedge". Unless you want to assemble your own (read this as soldering tiny parts), you will want to get the Preassembled version. This one costs about $10.00 U.S.. I believe that they have retired (discontinued) the version for the RPi 1A and 1B. You CAN elect not to get the interface board and ribbon cable and use female (Pi side) to male (breadboard side) jumpers. These will work, however, in some of the things we do later on, if you get the jumper on the wrong pin of the Pi, it could lead to damage to your Pi.

• Various Resistors, LEDs and Mini pushbutton switches. I will give you a list before we need them to give you plenty of time to obtain them. You can get these at many places.

• One other thing you might consider is a case, but only if you have the breakout boards and ribbon cables. This will protect your Pi from your handling of it.

Setup of your RPi

Now comes what must be for me, the most tedious part of the project… the setup. The steps we will perform are:

• Download the OS image.
• Unpack the image file from the archive file. Put it somewhere it's easy to get to.
• Installing OS to the SD Card.
• Getting the RPi hooked up.
• First boot of the RPi with the new OS.

So, let's get the OS image. Go to the downloads page on the official Raspberry Pi website (https://www.raspberrypi.org/downloads). You will be presented multiple versions of various images that you can download, including 2 versions of Ubuntu (The GUI version is Ubuntu Mate), Windows 10 IOT and more. If you have an older model (original models A or B), neither of the Ubuntu images or the Windows image will run on these models. You need the ARMV7 processor and the extra memory to be able to use these images.

The two we are interested in for this project, are the NOOBS and the RASPBIAN images. I will be using the RASPBIAN Wheezy image dated 05-05-2015 for our first few projects, but if you want to have the option of booting into other OS images on the same card,

feel free to download the NOOBS image. Just remember, if you have more than one OS on the card, you have less space available to the RASPBIAN image and you will run into an issue that I always used to, not enough space for all the things you want to try. Assuming that you are doing your work on a Linux machine, you can see the official installation instructions at https://www.raspberrypi.org/documentation/installation/installing-images/linux.md . If you are using a Windows machine or a Mac, follow the links there. I'm going to assume a Linux machine and will give you the instructions here.

Before we get started, you might be asking why, if there is a newer/better version available, am I using the older version. I've had some trouble with the 'Jessie' release and am more comfortable with the 'Wheezy' release at this time. I doubt that this was an issue with the release, probably just a bad download, but I just wanted to let you know. For the purpose of the next few articles, use 'Wheezy' and feel free to play with other versions.

Unpack the archive and have it be sent to a folder that will be easy for you to remember.

Installing the OS Image to the SD Card

If you are using an early version of the Pi, you will be using a standard sized SD card. If you are using a later version you will be using a Micro-SD card. To save me having to type the distinction every time, I will use "SD" in the documentation. One more thing before we start. I STRONGLY SUGGEST that you do not use a device connected to an external USB hub for the imaging of the SD card. I know the specs say you can, but I've never had very good luck doing this.

OK, here we go. Before inserting the SD card into your Linux box, open a terminal and do:

```
sudo -i
```

Most of the commands don't actually need the sudo level permissions, but it won't hurt and neither you or I have to remember when they do. Now run "df -h" to see what devices are currently mounted in the system. My system responds as shown below. Yes, I've named my machine Slartibartfast.

```
Slartibartfast ~ # df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       451G  336G   93G  79% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
udev            3.9G  4.0K  3.9G   1% /dev
tmpfs           796M  1.5M  794M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            3.9G  124M  3.8G   4% /run/shm
none            100M   32K  100M   1% /run/user
/dev/sdd1       2.8T  2.5T  314G  89% /media/greg/TOSHIBA
EXT
/dev/sdb1       1.8T  1.5T  294G  84% /media/greg/extramedia
/dev/sdc1       917G  681G  190G  79% /media/greg/MoreMedia2
Slartibartfast ~ #
```

Notice that I have 4 drives (sda1, sdb1, sdc1 and sdd1). I hope that when I plug in the SD card, it will come up as /dev/sde1. This will be important to know because if we get the wrong /dev/ device, we will corrupt it! Now plug your SD card into the computer and run "df-h" again. My system responds as:

```
Slartibartfast - # df-h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       451G  336G   93G  79% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
udev            3.9G  4.0K  3.9G   1% /dev
tmpfs           796M  1.5M  794M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            3.9G  124M  3.8G   4% /run/shm
none            100M   36K  100M   1% /run/user
/dev/sdd1       2.8T  2.5T  314G  89% /media/greg/TOSHIBA
EXT
/dev/sdb1       1.8T  1.5T  294G  84% /media/greg/extramedia
/dev/sdc1       917G  681G  190G  79% /media/greg/MoreMedia2
/dev/sde1        56M   20M   37M  36% /media/greg/boot
/dev/sde2        30G  3.0G   25G  11% /media/greg/13d368bf-
6dbf-4751-8ba1-88bed06bef77
Slartibartfast - #
```

Thank goodness! However /dev/sde1 has two partitions. This will be important in the next step. If you are me, please write down the drive information so you don't make a mistake. Now you will want to unmount the SD card drive.

```
Slartibartfast ~ # umount
/dev/sde2

Slartibartfast ~ # umount
/dev/sde1

Slartibartfast ~ # df -h
```

Notice that I started yet another "df -h" just to verify that the device is unmounted.

If you have ever used this SD card for anything before, you will want to remove the partitions before proceeding further. Some people might argue that this is not necessary, but why not? It only takes a few seconds and it keeps us from having problems. Use "gparted" to remove all the partitions.

```
Slartibartfast Raspbian # dcfldd bs=4M if=2015-05-05-raspbian-wheezy.img of=/dev/sde
768 blocks (3072Mb) written.
781+1 records in
781+1 records out
Slartibartfast Raspbian #
```

We are about to write the Raspbian image to the SD card. There are two ways to do this. First is to use the "dd" command AS SUDO, which I'm sure will be the first thing that comes to everyone's mind. However, remember when we use "dd", we don't get any information coming back to tell us what is going on and if it takes 5 minutes or longer to write the image, we won't see anything that entire time in the way of progress. While there are other methods I'm going to suggest that instead, you can use the "dcfldd" command (shown top right). Once it gets started (which could take a minute or so) it will give a progress report about how much has been written. Pick your "weapon" of choice. I'm going to show "dcfldd". Now, as SUDO, please change to where ever you have unpacked the image you are going to use.

```
Slartibartfast Raspbian # dd bs=4M if=/dev/sde of=wheezy-
2015-11-07.imgsafe
7609+1 records in
7609+1 records out
31914983424 bytes (32 GB) copied, 1675.51 s, 19.0 MB/s
Slartibartfast Raspbian # truncate --reference 2015-05-05-
raspbian-wheezy.img wheezy-2015-11-07.imgsafe
Slartibartfast Raspbian # diff -s wheezy-2015-11-07.imgsafe
2015-05-05-raspbian-wheezy.img
Files wheezy-2015-11-07.imgsafe and 2015-05-05-raspbian-
wheezy.img are identical
```

I show (below) an "ls" command here. I do this so I can remember the name of the file that I'm just about to work with, and I have the exact spelling.

On my machine, the process took about 10 minutes total.

This next step (above) is totally optional, but if you are like me, you want to verify the write so that you can be sure that this matches the image. We will make an image of the SD card we just did and write it to a temporary image file back to the hard drive. Since your SD card will likely be bigger than the one they used to create the distribution image, we will need to truncate our copy to match the size of the original and finally use diff to verify that both images are the same. Remember this could take a rather long time if you have a card larger than about 8Gb. I'm using a 32Gb card and it looks like it's going to take probably 30+ minutes to copy the image to the drive.

```
Slartibartfast Raspbian # ls -al
total 7424016
drwxr-xr-x 2 greg greg       4096 Oct 31 12:02 .
drwxr-xr-x 3 greg greg       4096 Oct 23 20:11 ..
-rw-r--r-- 1 greg greg 3276800000 May  7  2015 2015-05-05-raspbian-wheezy.img
-rw-r--r-- 1 greg greg 4325376000 Sep 24 16:14 2015-09-24-raspbian-jessie.img
Slartibartfast Raspbian #
```

As you can see, the images are the same, so if there is anything wrong from here until we log in, it's not our fault. This process could be a useful process as you go along and want to make a backup image of your Pi's "drive", just in case something happens.

Finally, we want to run the sync command which will make sure that anything remains uncommitted in the write cache is flushed and that is ok to unmount the SD card.

Now we can move on to some more "exciting" things. Powering on the Pi.

Getting ready to power up your RPi

Notice how I worded the heading for this portion of the instructions. There are certain things you should do before you apply power to your RPi. There are possibilities you can damage your RPi if you don't do the steps in order.

Plug in the Keyboard and Mouse into the USB port/ports.

Plug in the Ethernet cable into the Ethernet port or Wireless dongle into the USB Port.

Switch on your monitor or TV and get it set to the proper mode (HDMI or Composite).

Plug in the video cable (HDMI or Composite).

Put the SD card (or Micro-SD card) into place. It doesn't matter if you are using a full size SD card or a Micro-SD, you will insert it with the label facing down, not up towards the bottom of the Pi. And whatever you do, DO NOT remove the SD card while the RPi is powered on.

At this point, we are ready to plug in the power, so take a deep breath and cross your body parts. Plug it in.

If it worked, then we'll move on. If not, please retry the instructions above.

Once you get Pi booted into a distribution for the first time, you will presented with the raspi-config application. We are going to want to tweak some of the settings. We only really need to do this once.

You will see a screen with 9 options on it. We will work with numbers 1,3 and 4.

- Option #1 - Asks about expanding the file-system. You really want to do this so you can get the most space you can. It will take effect at the next reboot.
- Option #3 - Enable boot to Desktop/Scratch. You should go ahead and set this to Desktop Login as User 'Pi' at the Graphical Desktop.
- Option #4 – This sets various things that we take for granted by our automated setup systems. They include Locale, Timezone and Keyboard Layouts.
  - First select Locale. Since this computer comes from the UK, its default is to select things that someone living there would need. I, on the other hand, need to change some settings. I have to let the window scroll down to EN_US.UTF-8 UTF-8 and select it. Follow the prompts and you'll be fine.
  - Next I need to set my time-zone. Since I live in Colorado, USA, I would select America under the Geographic area, and Denver for the Time Zone.
  - Finally I have to select the keyboard layout I wish to use. It asks a lot of questions, so I would select "Generic", "US", "US", "Default", "No Compose Key" and "No" to Xserver Termination key.

Finally I'm ready to set it up, so I select "Finish" and "yes". Your Pi should reboot and you should see the normal desktop. Now we want to update the system to the latest, add a couple of applications that we'll need right away and then let it reboot once again.

Open a terminal off the top menu bar and do:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Now we want to install TightVNCServer. While this is an optional step, I find it much more constructive to use the Remote window on my Linux desktop than be forced to have 2 monitors, keyboards and mice. It always gets me confused about what/where I am.

```
sudo apt-get install tightvncserver
```

Once that's set up, it will ask you to create a password, so no

one can just jump into your screen. Make it easy for you to remember.

The very next thing we want to do is set the tightvncserver to automatically startup on boot. That way we don't have to have a mouse or a keyboard.

• Change to the home directory if you aren't already there.
```
$ cd /home/pi
```
• Next, change to the .config directory.
```
$ cd .config
```
• Now we will make a new directory here called 'autostart'.
```
$ mkdir autostart
```
• Change to the autostart directory we just created.
```
$ cd autostart
```
• Now create a new configuration file. $ nano tightvnc.desktop
And enter the following lines:
```
[Desktop Entry]
Type=Application
Name=TightVNC
Exec=vncserver :1
StartupNotify=false
```
• Save the file (^O) and exit (^X).

Almost done now. The last thing we will need to do is install the IDE we will be using for our code development, which is Geany.

```
sudo apt-get install geany
```

Move over to your normal computer and load VNCViewer software on it. Once that's all done, you will probably want to spend a moment or two by rebooting the computer and making sure that the VNC really did start up and connect. If everything works, you are done.

You will need (as I said earlier) a few things for next month. Some male to male jumpers, female to female jumpers, the breadboard, interface and cable and a handful of things from the electronics store…

• Some small LEDs. Try to get around 10 of each Red, Green, Yellow and Clear.
• Some small ¼ watt resistors. 220 ohm, 4.7K ohm, 10K ohm, and some other "normal" hobbyist resistors. About 10 each will do you and the salesperson at the local shop should be able to get you what you need.
• A couple of small switches (spst) that will fit on the breadboard. (usually comes with 4 pins).

Really that's about all you will need for the next article. In the meantime, enjoy playing with Linux on the Pi. I think you will be surprised by the power of this tiny device.

So until next month, the last thought I will leave you with is something we hear here in the U.S. all the time...

"***But wait … there's more!!!!!!!***"

Greg Walters is owner of RainyDay Solutions, LLC, a consulting company in Aurora, Colorado, and has been programming since 1972. He enjoys cooking, hiking, music, and spending time with his family. His website is www.thedesignatedgeek.net.

The Ubuntu Podcast covers all the latest news and issues facing Ubuntu Linux users and Free Software fans in general. The show appeals to the newest user and the oldest coder. Our discussions cover the development of Ubuntu but aren't overly technical. We are lucky enough to have some great guests on the show, telling us first hand about the latest exciting developments they are working on, in a way that we can all understand! We also talk about the Ubuntu community and what it gets up to.

The show is presented by members of the UK's Ubuntu Linux community. Because it is covered by the Ubuntu Code of Conduct it is suitable for all.

The show is broadcast live every fortnight on a Tuesday evening (British time) and is available for download the following day.

podcast.ubuntu-uk.org

Programming In Python Pt. 62

By the time you read this, it will probably be old news that there is a new Raspberry Pi that was released on November 26, 2015. It's called the Raspberry Pi Zero and the price is an unbelievable $5 U.S. or 4! . I haven't had a chance to find any actual dimensions, but they say it is about the size of a stick of gum. So if you've been holding off getting your new Pi due to cost, now you don't have an excuse. We will discuss the Pi Zero in future articles.

Now back to my Physical Programming series. This time we are going to start actually controlling things. Hopefully, you have been able to procure some LEDs, resistors, switches, jumpers and a breadboard.

As we go through the series, I will be using a free design tool called Fritzing to provide a visual representation of what the project wiring should look like.

You might want to get yourself a copy from their website (http://fritzing.org/home/). Not only can you keep copies of our projects locally, you also can have some fun designing your own circuits.

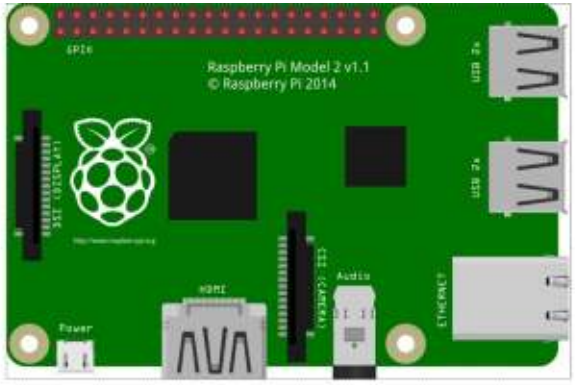
A Quick Discussion Of Our Components

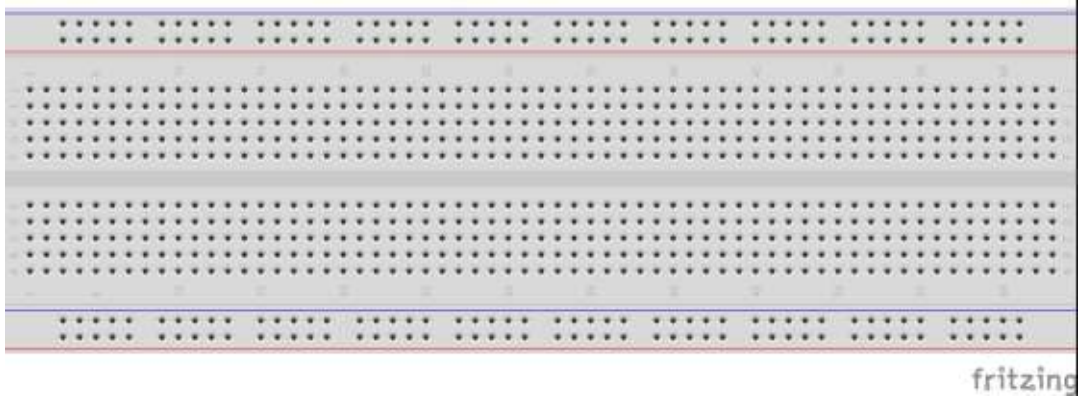
One more thing before we get started, which is a quick discussion on some of the electronic components we will be using this time, Resistors, LEDs and Switches.

Resistors

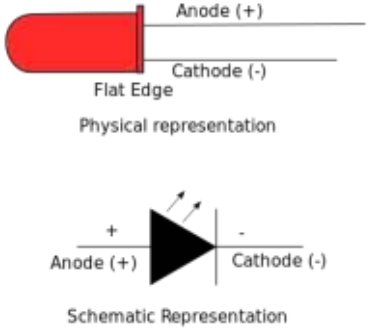
A resistor is a device that 'resists' the flow of electricity to a given extent. This will allow us to limit the amount of electricity that flows through a circuit or part of one. In the case of the LED projects, we will be using resistors so that they will reduce the amount of electricity flowing through the LED (and the GPIO pin), to keep it from burning out.

For a more detailed discussion of resistors, please see: https://learn.sparkfun.com/tutorials/resistors.





LEDs



LEDs are Light Emitting Diodes and are the “standard” replacements for bulbs in just about everything. With a little care in design, they will last almost forever. An LED has two leads/wires called Anode and Cathode. The Anode is the positive

side and the Cathode is the negative side.

If you have a new LED directly out of the package, you will notice that one of the leads is longer than the other. That is the Anode or the positive side. If both leads on a new device are the same length (or if you are recycling parts from an old circuit board), look for the flat edge. That will always show the Cathode or negative lead.

Switches

The switch I chose to use for this project is one that easily mounts in the breadboard or on a circuit board. It is simply square with a small round momentary button on the top. It also has 4 pins. The trick is to know which two pins of the four will be the ones we need. You could take an ohm-meter and run across all the combinations of pins until you find the set that works, or you could just look at the layout of the pins that connect it to the breadboard. The two pins to be used have the leads that grip into the board pointing at each other. You only need one set of pins, so just pick the set you wish.

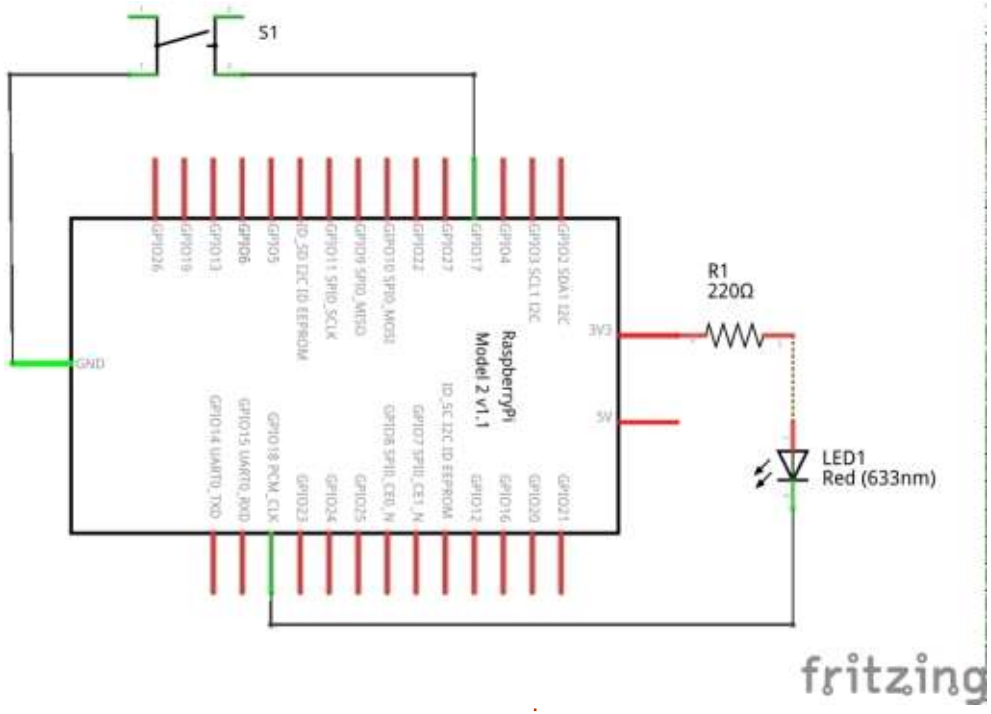
Our First Project...

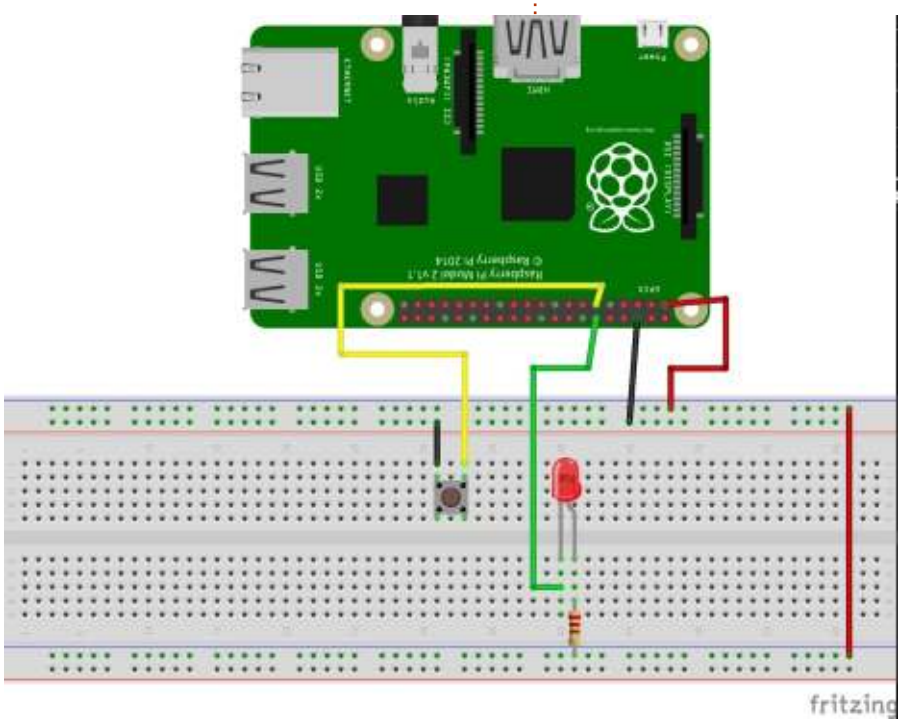
Now let's get started with our first construction project. It's a very simple electronics version of "hello world". We will connect a switch to one of the GPIO pins and monitor it to catch the press of the button.

Shown right is the actual schematic that we will be working with.

So we have a switch that is connected between ground pin and GPIO pin 17 which is physical pin 11. We also have an LED connected with its cathode to GPIO pin 18 (physical pin 12) and its anode connected to a resistor that connects to the 3.3volt pin on the Pi. It is at this point that you need to make a decision. Will you reference the pins by their position on the board, or the GPIO numbers. We'll get back to that in a minute. In the meantime, here's the wiring diagram…

You can see on the breadboard the three components...the switch, the LED and the resistor. The first pin on the RPi is the one on the top right. That pin provides the 3.3





volts DC that we need to power our project. The pin below it is counted as pin #2. Pin #6 is a ground pin. Note that both of those pins have wire connectors that go to the long horizontal buses on the breadboard. Some breadboards have a "+" and "-" on the power bus to help you remember which bus is which. I also have a long jumper from the positive 3.3 volt bus at the top of the breadboard down to the bus on the bottom. It really doesn't matter which bus on the breadboard you use for your power, as long as you are consistent.

There is a short jumper going from the top ground bus to one side of the switch and the other side of the switch connects to physical pin 11 on the RPi (or GPIO pin 17). As for the LED, the Cathode is connected to the physical pin 12 on the RPi (GPIO 18) and the Anode is connected to the resistor, which in turn is connected to the lower 3.3 volt bus. Also notice that the wiring is colour coded. Red will ALWAYS (in my diagrams) be a positive voltage, Black is for ground. Any other colors will mean interconnections for data.

If you have been keeping up so far, you will notice that I am giving both the physical pin number as well as the BCM GPIO pin number. The "BCM" stands for Broadcom, and, in our code, we will have to tell the RPi.GPIO library if we are using board numbering or BCM numbering. This is the decision I was referring to earlier. In our code, we will have to be consistent with one numbering scheme or the other. In the code we are about to look at, I provide both, and you can comment out whichever one you don't want to use. My personal preference is to use the BCM GPIO numbers, but for this project, I will stick with the physical board pin numbers. Now let's get into the code.

As always, I will break the code into parts and discuss each one. First (top right) we have to import the RPi.GPIO library, and we will alias it to the name "GPIO" to make things easier to type. Next, we define two variables; LedPin and BtnPin to the pin numbering scheme we wish to use. Here, I've decided to use the Physical pin numbering, since you probably don't have a breakout wedge yet. I've found the one from SparkFun

```
import RPi.GPIO as GPIO

# If you are using the BCM GPIO pin numbers...
#LedPin = 18
#BtnPin = 17
# Otherwise the physical board numbers...
LedPin = 12
BtnPin = 11
```

to be very nice, but it gives you only the BCM numbers on the pins. Our next bit of code (shown below) will be a function called "setup", where we set up the information for the library to use.

Notice that the first line is commented out since I will be using the board numbering in this example, but it's there to show you how to make the call.

Lines 3 and 4 show how to define what the pins will be, either input or output, and if we use the internal pull up resistors built-in on the RPi or not. So basically this portion of the code says to use physical board pin numbers as references, and it defines the Output pin to drive the LED and the pin that the signal from the button will be coming in on. Also notice that we define the pin for the button to have a pull-up resistor. This means that the signal-line will be at 3.3 volts and when the button is pressed it is pulled down to ground.

Our next function (following page, top right) is called loop, and, as the name suggests, we simply do a loop, checking the button input pin to see if it has been pulled low. If it has, then we turn the LED on, otherwise we set the LedPin to high. That might sound counter-intuitive, but remember

```
def setup():
        #GPIO.setmode(GPIO.BCM)
        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(LedPin, GPIO.OUT)
        GPIO.setup(BtnPin, GPIO.IN,pull_up_down=GPIO.PUD_UP)
```

we have the Anode connected to the 3.3 volt bus through the resistor. This means to turn the LED on, we have to pull the Cathode down to ground (0 volts) level to allow the LED to turn on.

The destroy function (shown below) basically cleans up the states of the pins so we don't get any errors the next time we need to use them.

Finally, we use the "main loop" (shown bottom) to call the routines in the proper order and to allow us an easy way to break out of the loop by pressing the Ctrl-C key sequence.

So, load the program into your RPi and run it. You will notice the text "...LED Off" keeps repeating on the screen until you press the button. That is due to the fact that our loop routine reads the level or status of the button pin and once the voltage goes low, it says "oh...the button input is low, so I need to turn on the LED".

One other thing to notice is that our first and second routines are named "setup" and "loop". It is a good thing to keep this format, because when we get to the Arduino programming, these two routines are required.

We are going to stop here for this month. I want the other authors to have room for their articles. Keep everything close at hand, because we will be using the same hardware setup next time.

Enjoy playing for now and I will see you next month.

```
def loop():
        while True:
                if GPIO.input(BtnPin) == GPIO.LOW:
                        print('...LED On')
                        GPIO.output(LedPin,GPIO.LOW)
                else:
                        print('...LED Off')
                        GPIO.output(LedPin,GPIO.HIGH)
```

```
def destroy():
        GPIO.output(LedPin,GPIO.HIGH)
        GPIO.cleanup()
```

```
if __name__ == '__main__':
        setup()
        try:
                loop()
        except KeyboardInterrupt:
                destroy()
```

Greg Walters is owner of RainyDay Solutions, LLC, a consulting company in Aurora, Colorado, and has been programming since 1972. He enjoys cooking, hiking, music, and spending time with his family. His website is www.thedesignatedgeek.net.

The Ubuntu Podcast covers all the latest news and issues facing Ubuntu Linux users and Free Software fans in general. The show appeals to the newest user and the oldest coder. Our discussions cover the development of Ubuntu but aren't overly technical. We are lucky enough to have some great guests on the show, telling us first hand about the latest exciting developments they are working on, in a way that we can all understand! We also talk about the Ubuntu community and what it gets up to.

The show is presented by members of the UK's Ubuntu Linux community. Because it is covered by the Ubuntu Code of Conduct it is suitable for all.

The show is broadcast live every fortnight on a Tuesday evening (British time) and is available for download the following day.

podcast.ubuntu-uk.org

Python In The Real World Pt. 63

Written by Greg D. Walters

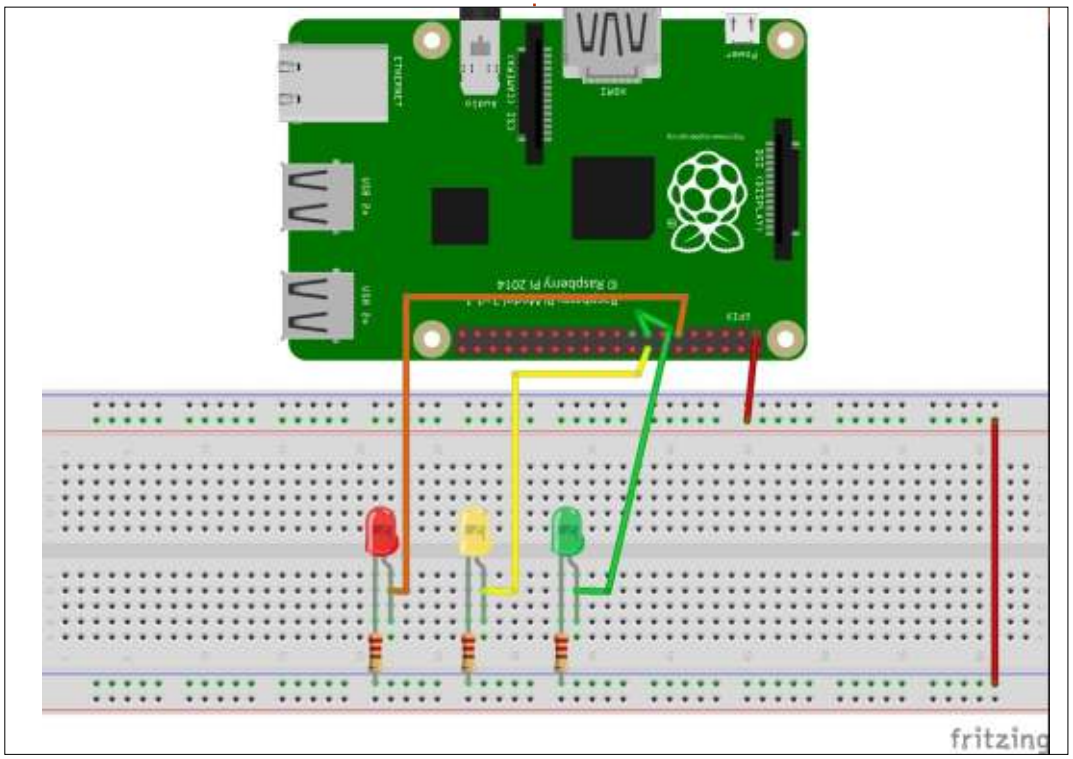
Welcome back to our Real World programming series. Last time, we programmed the RPi to turn on and off an LED when a button was pressed. Very simple, but this got us started. This month, we will do another simple project, a traffic light simulator using 3 LEDs, one Red, one Yellow and one Green. For the most part, the code is very similar to what we used last month, so you shouldn't have any problems. If you have any questions, I suggest you look at last month's article which should answer any of your concerns.

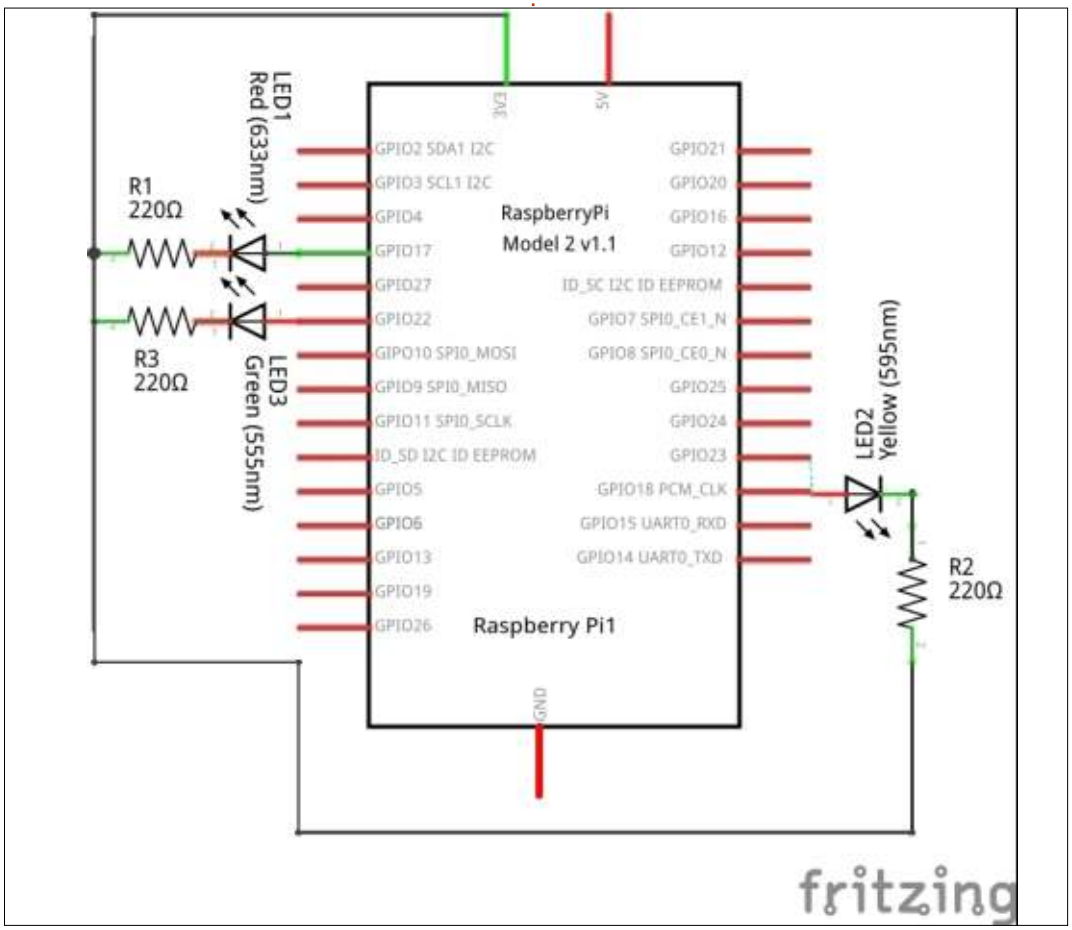
First, let's look at the schematic and the breadboard (below right).

Notice that the wire colours correspond to its 'job', with the exception of the orange wire. The red wires supply 3.3 volts. The green wire controls the green LED, the yellow wire controls the yellow LED, and the orange wire controls the red LED, since the red wire is already being used.

You should also know that the pins being used should work on a RPi v1a/b, RPi v1b+ and RPi v2b.

The red LED cathode is connected to GPIO 17 (Physical pin 11), yellow LED cathode is connected to GPIO 23 (Physical pin 16), and the green LED cathode is connected to GPIO 22 (Physical pin 15). The Anodes of all three LEDs are connected to one side of 220 Ohm resistors and the other sides are connected to a common 3.3





VDC. We don't need the ground voltage for this particular project.

Since I've driven only in the U.S. I've based the simulation on our traffic patterns. Long red light (10 seconds), green light is usually shorter than the red light time (8 seconds), and the yellow light is fairly short (2 seconds). These values are currently hard coded in the time.sleep() function calls. Feel free to change them as you see fit.

Now let's start working through the code.

```
#!/usr/bin/env python

# Traffic Light Simulator
# Written by G. D. Walters

#---------------------------

import RPi.GPIO as GPIO
import os
import time
import datetime

#---------------------------

RedLedPin = 17
YellowLedPin = 23
GreenLedPin = 22
```

The first 9 lines are our standard import statements and a few comment lines. The next three lines define the BCM pin numbers for our LED pins. If you wish to use physical pin numbers, be sure to change the GPIO.setmode() line in the next routine (top right).

```
def setup():
    GPIO.setmode(GPIO.BCM)                  # Numbers GPIOs by physical location

    GPIO.setup(RedLedPin, GPIO.OUT)         # Set the 3 LedPins mode as output
    GPIO.setup(YellowLedPin, GPIO.OUT)
    GPIO.setup(GreenLedPin, GPIO.OUT)

    GPIO.output(RedLedPin, GPIO.HIGH)       # Turn off LEDs
    GPIO.output(YellowLedPin, GPIO.HIGH)
    GPIO.output(GreenLedPin, GPIO.HIGH)
```

As I mentioned above, the GPIO.setmode needs to be changed from 'GPIO.BCM' to 'GPIO.BOARD' if you want to use the physical pin numbers instead of the BCM numbers in our definitions. The next three lines set the LED pins as output pins, and then turn all three LEDs off to start the program by setting the output value to HIGH.

```
def LEDLoop():
    print "Green On..."

GPIO.output(GreenLedPin,0)
    time.sleep(8)

GPIO.output(GreenLedPin,1)
    print "Green Off..."
    print "Yellow On..."

GPIO.output(YellowLedPin,0)
    time.sleep(2)

GPIO.output(YellowLedPin,1)
    print "Yellow Off..."
    print"Red On..."
    GPIO.output(RedLedPin, 0)
    time.sleep(10)
    GPIO.output(RedLedPin,1)
    print "Red Off..."
```

The LEDLoop routine is very simple:

- We print on the console "<color> On…",#
- Turn the LED on by setting the output value to 0 or low,
- Sleep for a designated period,
- Set the output value of the pin back to 1 or high,
- Then print that the LED is now off.

This is then duplicated for the Yellow and Red LEDs. The loop() routine simply forces the

```
def destroy():
    GPIO.output(RedLedPin, GPIO.HIGH)          # led off
    GPIO.output(YellowLedPin, GPIO.HIGH)       # led off
    GPIO.output(GreenLedPin, GPIO.HIGH)        # led off
    GPIO.cleanup()                             # Release resource

if __name__ == '__main__':          # Program start from here
    setup()
    try:
            loop()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy()
will be executed.
            destroy()
```

LEDLoop() routine to be called over and over until the user hits <CTRL> C on the RPi keyboard.

```
def loop():
    while True:
       LEDLoop()
```

The destroy routine and the main loop are the same as last month. We simply set all the LED pins to high, turning them off, and then call GPIO.cleanup().

I'm not sure that we could make a much simpler program to do what we need to do.

If you want, you could duplicate the 3 LEDS and make an intersection simulation before next time.

Next time, we'll have something that is a bit more challenging. Until then, happy programming.

Greg Walters is owner of RainyDay Solutions, LLC, a consulting company in Aurora, Colorado, and has been programming since 1972. He enjoys cooking, hiking, music, and spending time with his family. His website is www.thedesignatedgeek.net.

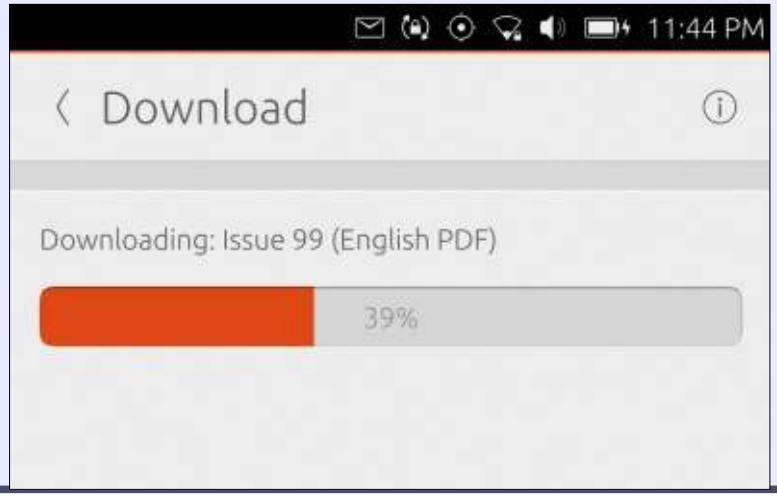
The Official Full Circle App for Ubuntu Touch

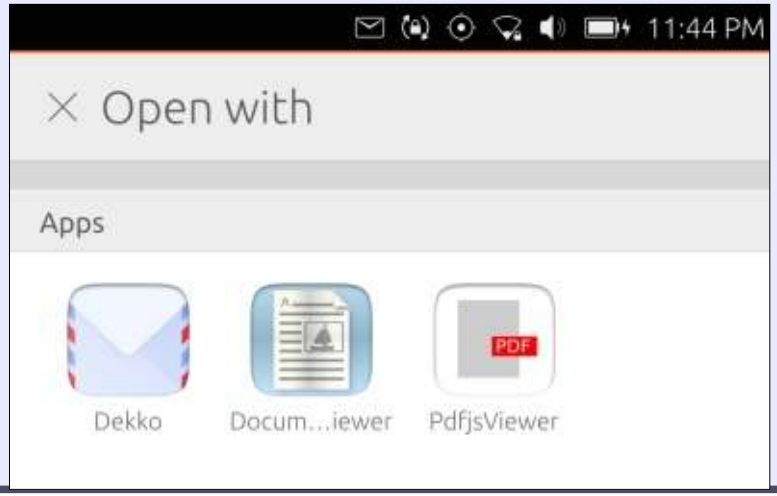
Brian Douglass has created a fantastic app for Ubuntu Touch devices that will allow you to view current issues, and back issues, and to download and view them on your Ubuntu Touch phone/tablet.

Install

Either search for 'full circle' in the Ubuntu Touch store and click install, or view the URL below on your device and click install to be taken to the store page.

https://uappexplorer.com/app/fullcircle.bhdouglass





Python In The Real World - Pt 64

Written by Greg D. Walters

Welcome back to the crazy world of Python Programming in the real world. Before we get started, I need to make a confession. Last time I goofed. The images in part 63 are wrong. The LEDs are backwards from what they should be. Brian Kelly noted this and was brave enough to point out the old man's errors. Thank you Brian. If you follow the text, you should be good to go.

Secondly, I have to apologize for not making it last month (FCM#106). I'm having more medical issues that are keeping me from sitting for too long. Hopefully this will be taken care of soon.

Enough of that. Now for this month's offering.

The Mystery LED

In the last two articles, we learned how to turn on and off LEDs programmatically. That was simple enough. This is digital output as opposed to analog output. The RPi, unlike the Arduino, cannot do analog I/O. So we are limited to turning a GPIO pin (and in this case, a LED) either on or off. This time we will be using that knowledge to do something pretty interesting.

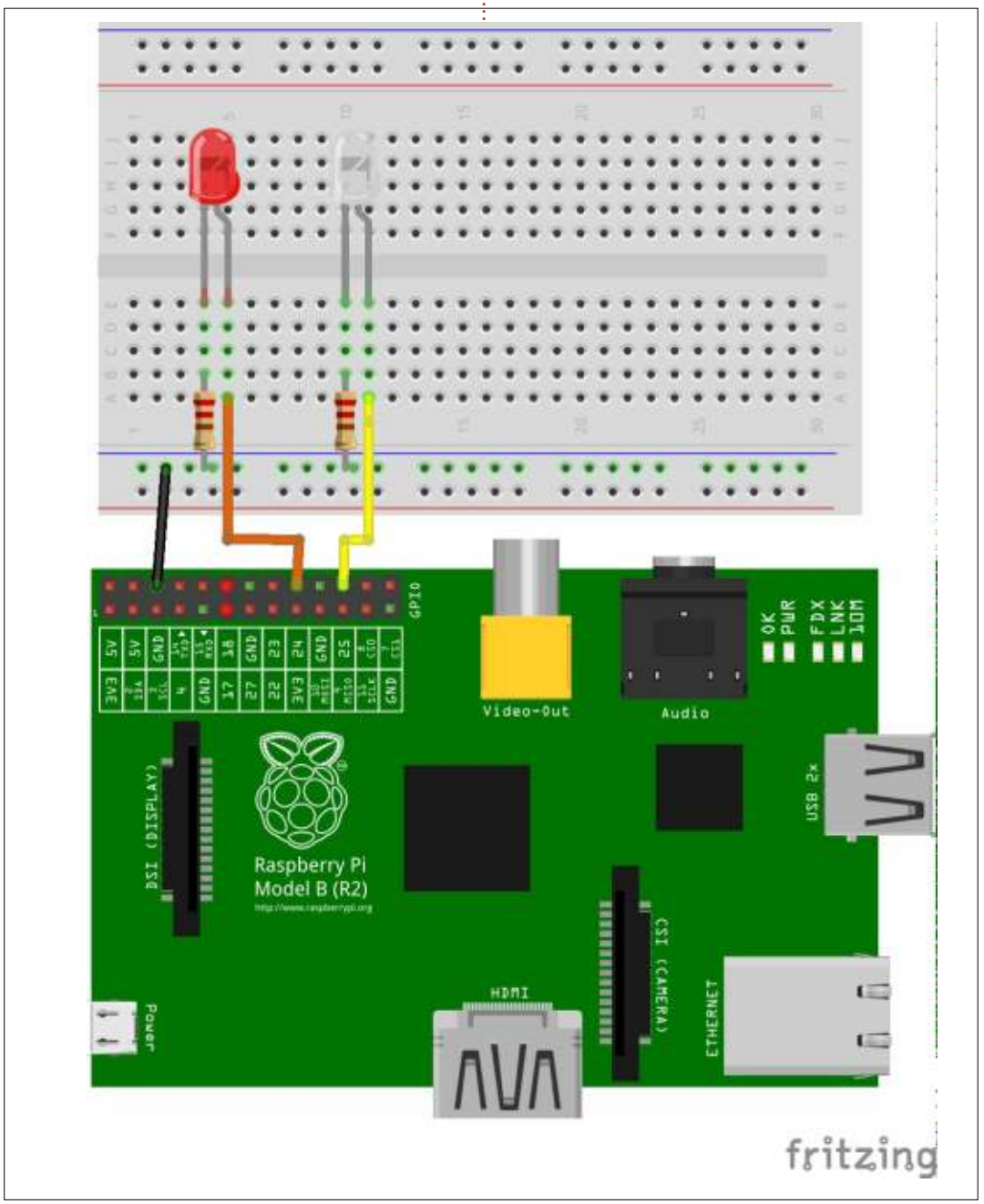
So get your Pi and your breadboard and we'll start working.

The Wiring

You will need a Raspberry Pi, a breadboard, two LEDs - one Red and one White, two 220 Ohm resistors and 3 jumper wires.

I've used the original Pi for this wiring image example. If you have a Pi B+ or 2B (or even the brand new 3), the pins at this point are exactly the same.

Just to avoid confusion (on my side), the Cathodes (Negative side) of the LEDS are connected to the resistors going to ground, and the Anodes (Positive side) are connected through the jumper wires to the Pi pins. The positive



contents

side of the LED is usually marked by the longer lead and the negative side is the one that has the flat spot on the base of the LED.

The Code

I won't explain the code just yet. Just put into the editor as it is. We will discuss it in a bit.

Once you have the code entered correctly, then run it and see what happens.

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setup(25,GPIO.OUT)
GPIO.setup(24,GPIO.OUT)
white = GPIO.PWM(25,100)
red = GPIO.PWM(24,100)
white.start(0) # start white led on 0 percent duty cycle (off)
red.start(100) # red fully on (100%)
pause_time = 0.05
print("Program Starting...Press CTRL+C to exit")
try:
    while True:
        for i in range(0,101): #101 because it stops when it finishes 100
            white.ChangeDutyCycle(i)
            red.ChangeDutyCycle(100-i)
            sleep(pause_time)
        for i in range(100,-1,-1):
            white.ChangeDutyCycle(i)
            red.ChangeDutyCycle(100-i)
            sleep(pause_time)
except KeyboardInterrupt:
        white.stop()
        red.stop()
        GPIO.cleanup()
```

The Reveal

If you have been paying attention over all these years, you probably have figured out what the code is doing. If you can't figure it out, don't feel bad. We'll jump into the explanation.

Instead of the LEDs being either on or off, they pulse on and off. Since I said earlier, we can only send out (or read) a On/Off voltage (or 1/0, or High/Low), so how can this be?

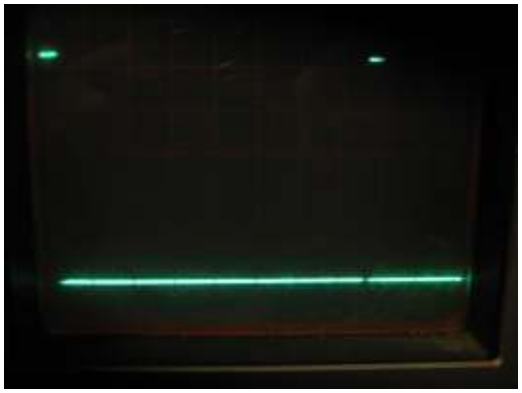
We are using a trick called PWM or Pulse Width Modulation. We are still living with the rules, but we are bending them to our benefit. The pictures below, taken from my oscilloscope connected to the project, should help explain a bit clearer. We will be concerned with only one LED at this point.

If we send out a Low to the GPIO pin to the LED it's zero volts. The LED is getting nothing on the Anode, so it is off. In the last two articles, when we turned the LED on by sending the Anode of the LED a High So we have in the first instance a zero, and in the second a 1. Just like we have assumed... either Off or On.

This time we vary the amount of time that the GPIO signal is high and low. If we do it slowly, the LED would simply flash on and off in response to the voltage. In the case of this version, we are switching it on and off very quickly and at the same time, changing the amount of time it is on compared to off, which is called the duty cycle.

You can see that the signal is on for about 80% of the time and off for about 20%, which would be a 80% duty cycle. By doing this quickly, the LED reacts by dimming

a bit from the 100% on all the time. As the program does its loop, it changes the duty cycle and makes the high longer or shorter depending on what part of the loop it is.



In the picture above, we have a duty cycle of about 5%. In this case the LED is turned on for such a short time, that it is extremely dim and for all intents and purposes it is off.

Now, let's start taking apart the code.

```
import RPi.GPIO as GPIO
from time import sleep
```

As always, we start with our imports. We import the GPIO library, and this time, we import the sleep function from the time library. You will understand the reason for that shortly.

```
try:
    while True:
        for i in range(0,101): # 101 because it stops when it finishes 100
            white.ChangeDutyCycle(i)
            red.ChangeDutyCycle(100-i)
            sleep(pause_time)
        for i in range(100,-1,-1):
            white.ChangeDutyCycle(i)
            red.ChangeDutyCycle(100-i)
            sleep(pause_time)
except KeyboardInterrupt:
        white.stop()
        red.stop()
        GPIO.cleanup()
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(25,GPIO.OUT)
GPIO.setup(24,GPIO.OUT)
white = GPIO.PWM(25,100)
red = GPIO.PWM(24,100)
```

In these five lines, we set the GPIO mode to BCM, and set the GPIO pins 24 (physical pin 9) and 25 (physical pin 11) to be output pins. We have done this before. Now we set the values for the PWM to 100% duty cycle.

```
white.start(0) # start white led on 0 percent duty cycle (off)

red.start(100) # red fully on (100%)
```

We next turn the Red LED on (100%) and the white LED to 0 volts.

```
pause_time = 0.05
```

```
print("Program Starting...Press CTRL+C to exit")
```

We set the pause_time variable to 0.05 seconds. This makes it fast enough to (hopefully) not allow for a flicker.

In the next block of code, we do our loops. The first loop is to make the white LED get "brighter" and the red LED to get "dimmer". The second is to reverse the process. Just using the first loop as an example, we use a FOR LOOP to set the value of i and then we set the duty cycle for the white LED to i and that of the red LED to 100-i.

Notice that we have wrapped this with a TRY...EXCEPT set. This allows us to continue to run until the user enters CTRL+C. When that happens, we fall out of the TRY side so we can do our clean up code.

So now you know that we can bend the rules to our use.

Next time, we will start to examine a different GPIO library. Until then, have fun.

Alan holds a PhD in Information and the Knowledge Society. He teaches computer science at Escola Andorrana de Batxillerat (high-school). He has previously given GNU/Linux courses at the University of Andorra and taught GNU/Linux syadmin at the OU of Catalunya.

Python In The Real World - Pt 65

Written by Greg D. Walters

Welcome back. This month will be a hodgepodge of information. The main reason is that there are some important advances in tech and you will need time to get some parts for the next few articles.

In the near future, we will be adding the Arduino into our toolbox. I suggest starting off with the UNO or a UNO clone which can be purchased for less than $30 US (£22). We will also need some sensors to really get going. While these are optional and you can just read the article, building these projects are more than half the fun. So, with that said, here is a list of parts…

- One Wire Digital Temperature Sensor - DS18B20
- DHT11 Basic Temperature/Humidity Sensor
- 16x2 LCD Display
- 4.7K and 10K ¼ Watt resistors (3 or 4 of each)
- Large Breadboard (60+ x 10 with power rails)
- 10K Potentiometer (2 or 3)
- Male to Female jumpers (Pi to Breadboard) about 8”
- Male to Male jumpers (Arduino to Breadboard) about 8”
- Male to Male jumpers (Breadboard to Breadboard) small to medium
- Toy/Hobby motor 6 VDC
- L293D or SN754410 Motor Control Chip
- 4 AA Battery Holder and Batteries.

This will pretty much get you going for the next few months. Of course, you could get more and explore on your own. Most everything on the list is less than $10 US. If you shop the internet diligently, you can get very good prices on everything really inexpensively. We’ll leave this for now, but for next time, you will need the DS18B20 temperature sensor and a 4.7K resistor as well as a breadboard and jumpers if you don’t already have one.

Recently, there has been a great stir on the Internet about the Amazon Echo / Alexa device software being ported to run on the Raspberry Pi. The biggest reason for the excitement is that currently the Echo / Alexa is available only in the US and many people in the UK and other countries have been waiting, not so patiently, for it. This gives them a chance to enjoy the technology.

There are at least two projects currently working on getting Echo on the Pi. The first uses Java. You can find the code and instructions at https://github.com/amzn/alexa-avs-raspberry-pi. I have done this project on both a Pi Version 1B and the new Pi 3B. It worked well on both. Many people have problems getting this to work, but I did it in about 4 hours (with small breaks and interruptions), and it worked the first time. The best advice I can give you is take your time, plan on a long weekend, and follow the instructions to the letter. The only problem that I had was that npm and nvm needed to be installed, and, at that time, these installation instructions were not included. I believe this issue has been corrected.

The second project uses Python and is located at https://github.com/lennysh/AlexaPi. To be honest, I tried this, but could not get it to run. I will tell you that I did not spend nearly as much time on this project as I did on the Java version, due to many doctor visits this past week. I intend to spend more time on it to try to get it working.

If you decide to try either projects, PLEASE use a blank SD card and not one that has something you want to keep. Load the Raspbian or NOOBS OS from scratch. That way, if something goes wrong, you can just reload the OS and start fresh.

There are some things you need to know before you attempt to do this project. All of the information below pertains to the java version, but some can be considered to apply to both projects…

- You need to have a USB microphone. Headphone based microphones have issues. I’m using a Logitech webcam with built in microphone and it works well.
- You will also need a set of speakers or headphones attached

to the audio out jack. Many people have had lots of issues with bluetooth audio devices.

• You must push a button to get the Echo / Alexa to listen for your command. It doesn't currently listen for the "wake" word. (more below).

• Some of the features that the actual Echo / Alexa have don't currently work.

• Things like location, weather, traffic, work correctly only in the USA. In any other country, you will get information for Seattle, Washington, USA

• The only supported language currently is English. According to what I was able to find out from my research is that, once the device is being sold in a given country, they will add support for that country's "official" language. I understand that in the UK, the official language is English, and that in the USA, there is no "official" language and that Spanish is a largely spoken language, but is not supported on the device as yet. There are many flame threads on the web – if you wish to voice your ire at the fact that your language of choice is not supported or that the Echo / Alexa is not available there. All I can suggest is that you should be patient. The device was a sleeper for a while and just recently took off well. Amazon, I'm sure, is working on support for other countries right now.

• When you start the app, you have to run two processes. The second one will create a GUI box which has a long URL string that you must copy and paste into a web browser. Once that gets to Amazon properly, then you must click the [OK] button on the screen. You will be presented with a screen that has a [Start Listening] button and some multimedia buttons. To "wake" Alexa up, you click the 'start listening' button and, after you hear the "ding", speak your question or command. When finished, you can click that button again to have it stop listening and process your command, or you can let it timeout (about 5 seconds) then it will start processing. Many people are working on headless operation (no monitor) and a physical button connected to a GPIO pin, and some are actually working on the "wake" word option. You can find more information in the issues section.

• You should (read MUST) use a decent quality SD card. My suggestion is to get nothing less than a Class 10 card that is no smaller than 16 Gig.

• As soon as you boot into the new operating system for the first time, run a 'sudo raspi-config'. Be sure to enlarge the file system to take in the entire card. Be sure to turn SSH on. You will need to reboot here. Next you should then do a 'sudo apt-get update' and then a 'sudo apt-get dist-upgrade' so you are at the latest software revisions.

• There are some steps that require you to enter certain data. Make notes of what you entered, either by a screen shot, into your smartphone, or (HORRORS!!!!!!) on paper. It will make things easier.

• If you have any problems, check the issues section. More than likely someone has already had the same problem and there might be a fix.

• Print the web page with the instructions and work off the print. This way, you can check off those steps you have already completed. Especially helpful if you get interrupted.

• You can find more information, and change certain settings, at alexa.amazon.com. I understand that some people who are not in the USA have problems with this site.

I think that's enough for this month, but next month, we will turn our RPi into a thermometer. The neat thing about using the DS18B20 sensor is that you have more of them on a single line. This way, you could use one in the living room, one outside, etc. We'll use these sensors later on with the Arduino and be able to use the arduino as a remote device so we don't have to try to run a long cable and change the resistance to a point that it won't work.

Until next month, enjoy checking out the Alexa project, and, if you try it / them, hope you have success.

Greg Walters is owner of RainyDay Solutions, LLC, a consulting company in Aurora, Colorado, and has been programming since 1972. He enjoys cooking, hiking, music, and spending time with his family. His website is www.thedesignatedgeek.net.

Python In The Real World - Pt 65

Written by Greg D. Walters

Last month, I suggested you get a number of parts and if you were able to get them, I hope that it didn't cost you too much. If you haven't gotten them, then follow along as best you can, and if there is a particular project you want to try, then get those components that are needed. I'm trying to do this on as little cash outlay for either you or me as possible. Frequently, you can recycle many of the items from older electronic items; many can be found at a local thrift store for pence on a pound. (Hopefully I got that one right. We say pennies on a dollar here, so at least give me an "F" for effort… ok?)

As I was laying last week, waiting for some surgery, I was thinking that if someone were to come up to me and ask directly why I'm doing this, what my answer would be. Before the wonderful chemicals they pumped into my body to make the process less horrible, I realized that the REAL reason is multi-part. First, is to create excitement in "non-programmers" when doing things that seemingly could not be done without a ton of training. Secondly, is to show that the newer technology, like the Raspberry Pi and the Arduino, is not beyond the ken of the "general joe" out there, but that anyone can do things that have real world applications (hence the title of our series). That having been said, making LEDs blink is only the same kind of project for the hardware world as the "Hello World" program is in the programming realm. You have to take small steps before you move to the big race. Believe me, we will be doing some amazing things with all those little parts, whatsits and thingamabobs.

This month, we will be using the DHT11 Basic Temperature/ Humidity Sensor with our Raspberry Pi. Next month, we will be doing the same sort of thing using the Dallas DS18B20 temperature sensor, and if there is time and/or space, we'll also talk about the 16x2 LCD display. In a few months, we'll switch from the Raspberry Pi to using the Arduino. Don't worry, none of the things we are using now will go unused after a single project. For example, once we have a grasp of some of the Arduino basics (which WILL involve learning a small amount of 'C' like programming (sorry about that)), we'll be writing programs in Python on the RPi (or your local computer) to control the Arduino. The sensors we have learned about in our RPi experiments will be re-used when we learn about the Arduino, and many will be incorporated into some larger projects. Very shortly, we will be using DC Motors, Solenoids and Stepper Motors in some really basic projects, but we'll use them in larger projects, including building a Computer Controlled (RPi) Laser Engraver using a Laser Diode recovered from an old DVD Burner.

Enough about the future. Let's start with this month's project.

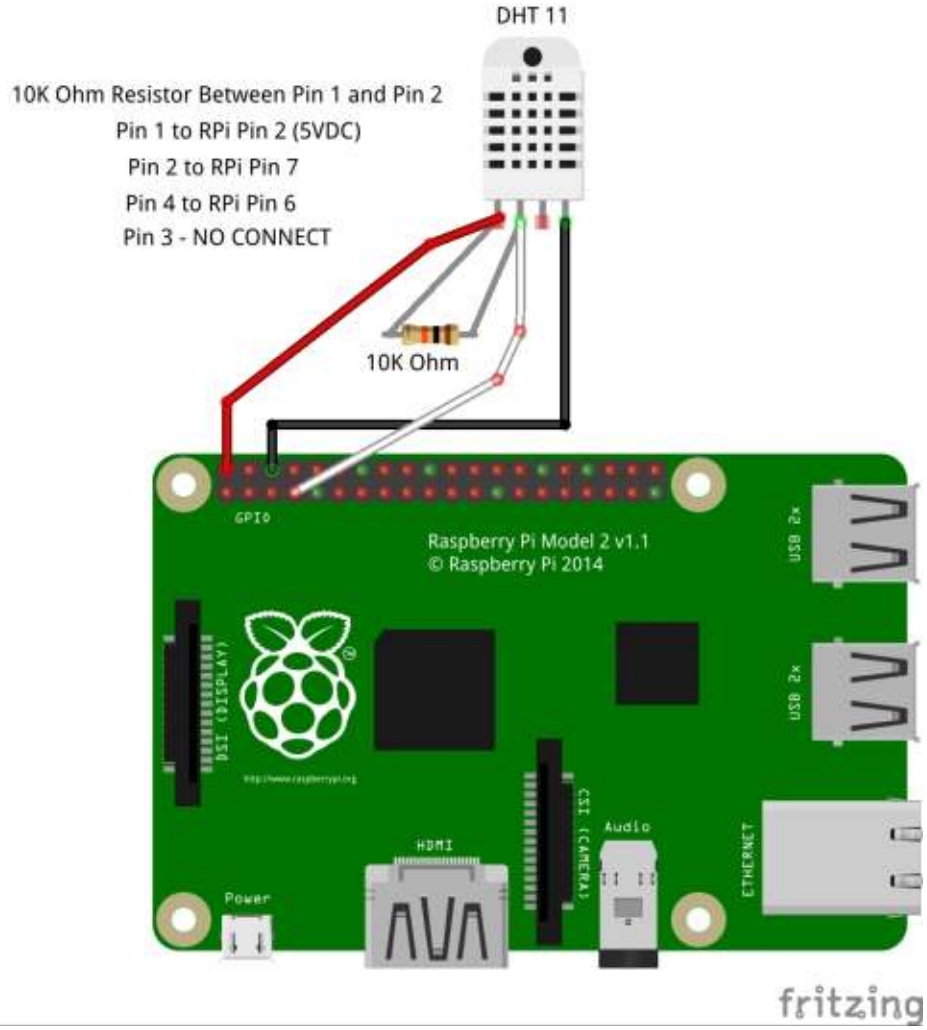
The DHT11 is the least expensive sibling of a series of temperature and humidity sensor sets. The DHT11 has a temperature range from 0º to 50º C with ±2º C accuracy (32º to 122º F, ±3.6ºF) and a humidity range from 20-90%RH ±5%. You can see that it's not the most accurate sensor on the market; there is a DHT22 that is more accurate and has a wider range (-40º to 80º C temp range) but about twice as expensive.

It's a bit of a funny looking thing. A blue rectangular plastic box with holes in it and something shiny inside it. It might come just as a single sensor with 4 pins, or already on a mini-circuit board with 3 or 4 pins. Either way, they are basically the same. For now, we'll use the discrete component (the one without the circuit board) for the sake of the discussion, and I'll address the differences as we go along.

Whenever you want to work with a new sensor, you should get a spec sheet (data sheet). A simple web search should turn up a number of results. Try to get something directly from the manufacturer if at all possible. For the DHT11, a good place to get one of the various data sheets

available is http://www.micropik.com/PDF/dht11.pdf. While this isn't directly from the manufacturer, it is from a company that sells it, and has "translated" the manufacturer's data into a 9-page PDF file.

You might already be asking, why do I need this? There's a bunch of information that, unless you have a PhD in Physics or something, you'll never need. Well, that is true, but there is a lot of information that IS relevant and can potentially keep you from blowing up either the sensor, the controller, or your work bench. In this case, we find that the DC operating voltage is between 3 to 5 volts and it pulls about 0.5mA during "normal" conditions (section 6). We also find that this is a rather slow device and that we should not try to pull data more than once per second. Basically we'll keep it around once every five seconds in our testing program, which is way more than we'll need in reality. Another thing: if the cable that sends the data from the sensor to the microcontroller (our RPi) is less than 20 meters, we should have about a 5K ohm resistor between the data line and the local power supply (at the sensor) as a pull-up. One last thing (I'm going to stop here, but there's much more): Pin 1 is positive voltage, Pin 2 is the data pin, and Pin 4 is the ground pin. This gives us pretty much everything we need to know to safely connect this to our RPi. Below is the wiring diagram for a "raw" DHT11 sensor WITHOUT a breakout board. If you have a sensor with a breakout board, see my discussion below the diagram.



Notice that I said earlier that a 5K resistor was needed as a pull-up. If you are going to use 3.3 VDC as a power source (RPi pin 1), then a 5K resistor works pretty well. If, however, you are going to use 5 VDC as shown in the diagram, use a 10K resistor.

You can see that it's fairly simple, just three wires and a resistor. For our simple project, don't try to make the wiring the entire 20 meters though.

If you have a DHT11 on a breakout board, you will likely have only 3 output pins on it. I have two sensors from different vendors, and (go figure) both have a different pinout. One is laid out [Data] [Positive Voltage] [Ground] and is marked "S -". The other is [Ground] [Data] [Positive Voltage] and is marked as such. Hopefully, yours has some sort of pinout definition printed on it. If not, you can use a multimeter to trace the ground pin and voltage pin directly from the sensor to the breakout pin. You can usually guess that if there are three output pins on the breakout board and you know ground and positive voltage, then the other SHOULD be the data pin.

Now our program code.

For the sake of getting things up and running quickly, we will be

contents

using some code provided by the kind people at Adafruit.com – they provide the library for working with the DHT11. (They found that trying to run straight Python code for the library causes some timing issues, so the library is actually written in 'C'.) There are a number of steps involved, so follow the instructions carefully. I've paraphrased them so if something doesn't work, you can also find the instructions at the Adafruit website at https://learn.adafruit.com/dht-humidity-sensing-on-raspberry-pi-with-gdocs-logging/software-install-updated . Once everything is done, you can run my modified Python example presented at the end of the instructions.

In your "/home/pi" directory, run the following commands:

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git

cd Adafruit_Python_DHT

sudo apt-get update

sudo apt-get install build-essential python-dev python-openssl
```

Ignore any errors that state a package is already installed.

Next, install the library by running:

```
sudo python setup.py install
```

Once all that is done, you can move on to our sample code.

Above is my modified sample code "borrowed" from the Adafruit sample code.

All of the above can basically be boiled down to three lines of code.

```
#!/usr/bin/python
# simpletest.py
#----------------------------------------------------
# Original code information copyright below.
# Copyright (c) 2014 Adafruit Industries
# Author: Tony DiCola
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
# Code modifications by G.D. Walters for Full Circle Magazine
import Adafruit_DHT
from time import sleep
#----------------------------------------------------
# Sensor should be set to Adafruit_DHT.DHT11,
# Adafruit_DHT.DHT22, or Adafruit_DHT.AM2302.
#sensor = Adafruit_DHT.DHT22
#----------------------------------------------------
sensor = Adafruit_DHT.DHT11
#----------------------------------------------------
```

The two import statements and the assignment of the variable 'sensor' to the class code.

```
pin = 4
sleep(3)
```

Here we define that the sensor is connected to GPIO pin 4 and then we wait 3 seconds for things to settle and be ready to work.

We use a simple loop (next page, top right) to grab the values for humidity and temp over and over. I never got the knack of relating Celsius to "real" temperatures, so I convert it so that I can understand it. If you want Celsius, just comment out the conversion line.

Now (next page, bottom right) we check to see if we got realistic values for both humidity and temperature, then we display them and sleep for 5 seconds.

I must admit, when I run the program with one sensor, it gives some rather wacky results for the first two or three minutes, then

```
#---------------------------------------------------
# Here we loop over and over getting and displaying the
data.
# Use <Ctrl><C> to break out.
#---------------------------------------------------
while 1:
    # Try to grab a sensor reading.  Use the read_retry
method which will retry up
    # to 15 times to get a sensor reading (waiting 2
seconds between each retry).

    humidity, temperature =
Adafruit_DHT.read_retry(sensor, pin)

    # Comment out the next line to display Celsius
    temperature = temperature * 9/5.0 + 32
```

settles down to values that I can trust. The other sensor seems to "lock in" faster, so I'm just writing it off to something in the first sensor.

Well, that's it for this month. Remember, we'll be using the Dallas temp sensor next time, so be ready.

Have a good time and I'll see you next month.

Greg Walters is owner of RainyDay Solutions, LLC, a consulting company in Aurora, Colorado, and has been programming since 1972. He enjoys cooking, hiking, music, and spending time with his family. His website is www.thedesignatedgeek.net.

The Ubuntu Podcast covers all the latest news and issues facing Ubuntu Linux users and Free Software fans in general. The show appeals to the newest user and the oldest coder. Our discussions cover the development of Ubuntu but aren't overly technical. We are lucky enough to have some great guests on the show, telling us first hand about the latest exciting developments they are working on, in a way that we can all understand! We also talk about the Ubuntu community and what it gets up to.

The show is presented by members of the UK's Ubuntu Linux community. Because it is covered by the Ubuntu Code of Conduct it is suitable for all.

The show is broadcast live every fortnight on a Tuesday evening (British time) and is available for download the following day.

podcast.ubuntu-uk.org

HOW TO CONTRIBUTE

FULL CIRCLE NEEDS YOU!

A magazine isn't a magazine without articles and Full Circle is no exception. We need your opinions, desktops, stories, how-to's, reviews, and anything else you want to tell your fellow *buntu users.
Send your articles to: articles@fullcirclemagazine.org

We are always looking for new articles to include in Full Circle. For help and advice please see the Official Full Circle Style Guide: http://url.fullcirclemagazine.org/75d471

Send your comments or Linux experiences to: letters@fullcirclemagazine.org
Hardware/software reviews should be sent to: reviews@fullcirclemagazine.org
Questions for Q&A should go to: questions@fullcirclemagazine.org
Desktop screens should be emailed to: misc@fullcirclemagazine.org
... or you can visit our site via: fullcirclemagazine.org

Please note:

Special editions are compiled from originals and may not work with current versions.

Full Circle Team

Editor - Ronnie Tucker
ronnie@fullcirclemagazine.org

Webmaster - Lucas Westermann
admin@fullcirclemagazine.org

Special Editions - Jonathan Hoskin

Editing & Proofreading
Mike Kennedy, Gord Campbell, Robert Orsino, Josh Hertel, Bert Jerred, Jim Dyer and Emily Gonyer

Our thanks go to Canonical, the many translation teams around the world and Thorsten Wilms for the FCM logo.

For the Full Circle Weekly News:

You can keep up to date with the Weekly News using the RSS feed: http://fullcirclemagazine.org/feed/podcast

Or, if your out and about, you can get the Weekly News via Stitcher Radio (Android/iOS/web):
http://www.stitcher.com/s?fid=85347&refid=stpr

and via TuneIn at: http://tunein.com/radio/Full-Circle-Weekly-News-p855064/

Getting Full Circle Magazine:

EPUB Format - Most editions have a link to the epub file on that issues download page. If you have any problems with the epub file, email: mobile@fullcirclemagazine.org

Issuu - You can read Full Circle online via Issuu: http://issuu.com/fullcirclemagazine. Please share and rate FCM as it helps to spread the word about FCM and Ubuntu.

Magzster - You can also read Full Circle online via Magzster: http://www.magzter.com/publishers/Full-Circle. Please share and rate FCM as it helps to spread the word about FCM and Ubuntu Linux.