

FREE DVD

GREAT GIFT IDEA!



Cool Rasp Pi Projects!

LINUX MAG Special

RASPBERRY PI ADVENTURES

EASY LESSONS IN COMPUTING FOR A NEW GENERATION OF HACKERS AND MAKERS



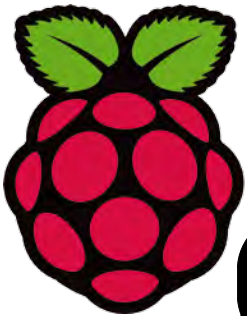
Make your Pi into a web server!

Electronics
Get started with wiring and circuit design

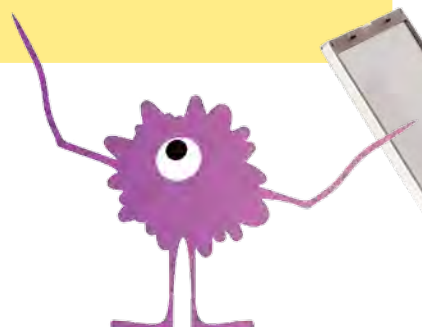
Discover the Rasp Pi camera

LEARN TO PROGRAM!
Take your first steps with:

Compose music with Sonic Pi



- Turtle Art
- Scratch
- Python



£7.99
Issue #27
9 771757 636002
Linux Magazine Special
27

An entire year of **RASPBERRY PI PROJECTS AND OPERATING TIPS**

for one low price!

Two bundles to choose from:



2014



2015

Hurry while supply lasts!

ORDER NOW!



<http://shop.linuxnewmedia.com/us/magazines/raspberry-pi-geek/catchup.html>

THIS SINGLE ISSUE IS YOUR GUIDE TO THE RASPBERRY PI EXPERIENCE!

ATTENTION ADVENTURERS

IF YOU'VE ALWAYS WANTED TO EXPERIENCE THE RASPBERRY PI, but you don't want to wade through pages of complicated documentation, you've picked up the right magazine! *Raspberry Pi Adventures* offers an insightful collection of fun, informative, and easy projects for Raspberry Pi users of all ages.

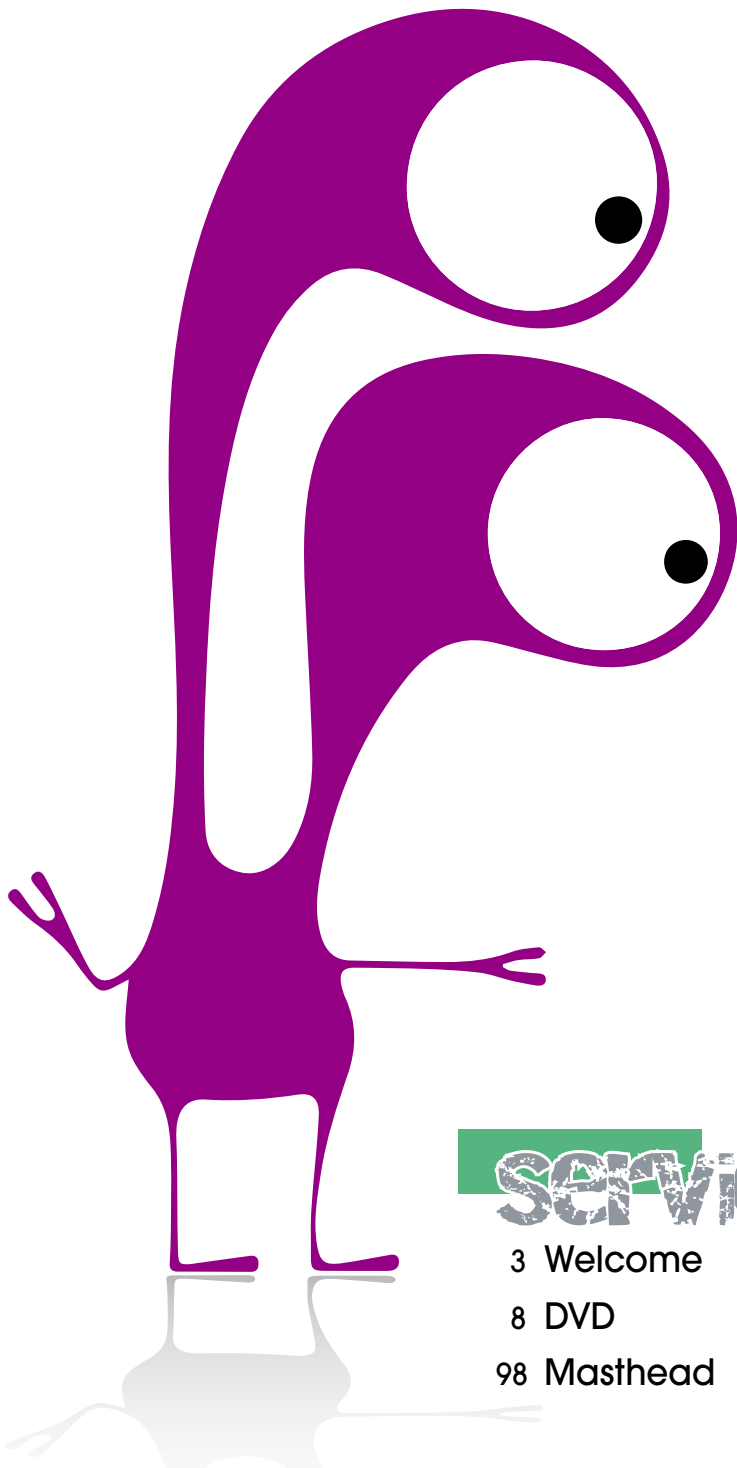
The articles in this issue are written for curious beginners who want to learn about computers and are ready to explore the wonders of the Raspberry Pi. You'll travel deep into the Raspberry Pi experience, exploring the amazing \$35 computer in all its most popular roles. Learn about web servers by building your own home website. Use the Raspberry Pi camera to create a time-lapse video recording. Get started with computer programming, first with easy tools like Turtle Art and Scratch, and then with your first steps in the powerful Python language, a favorite of programmers around the world.

Many Pi experts believe the real fun is creating electronics projects that light up lights, start motors, and ring bells. We'll give you a thorough introduction to Raspberry Pi electronics and help you get started using a cobbler, breadboard, and other electronics tools. You'll also learn to create animations in Scratch, and you'll get the chance to make some music with the amazing Sonic Pi, a music tool created specially for Raspberry Pi environments.

Adventurers get ready! Let the journey begin...



RASPBERRY PI ADVENTURES



INTRODUCTION

- 10 Get Started
Before you start your first adventure, you'll need to set up your Raspberry Pi and install the operating system.
- 18 Discover Raspbian
Customize your system, work with the terminal, and install new applications on your Raspberry Pi.



service

- 3 Welcome
- 8 DVD
- 98 Masthead



Highlights

26 Web Server: Set up your Pi as a mini web server for the home.

52 Scratch: Learn the basics of programming with this easy graphical language.

70 Electronics: Build your Pi into cool projects that interact with the world.

90 Sonic Pi: Compose music and use your Pi as a musical instrument.

PROJECTS

26 Web Server

Install a web server on your Raspberry Pi and build a simple website.

34 Pi Camera

Control a camera with your Raspberry Pi.

42 Turtle Art

Take your first programming steps with turtle graphics.

52 Scratch

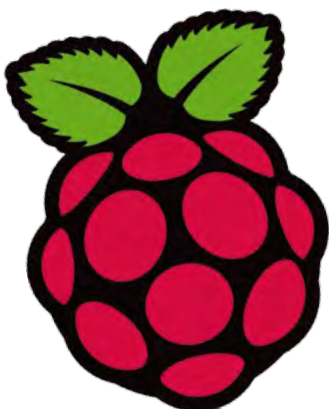
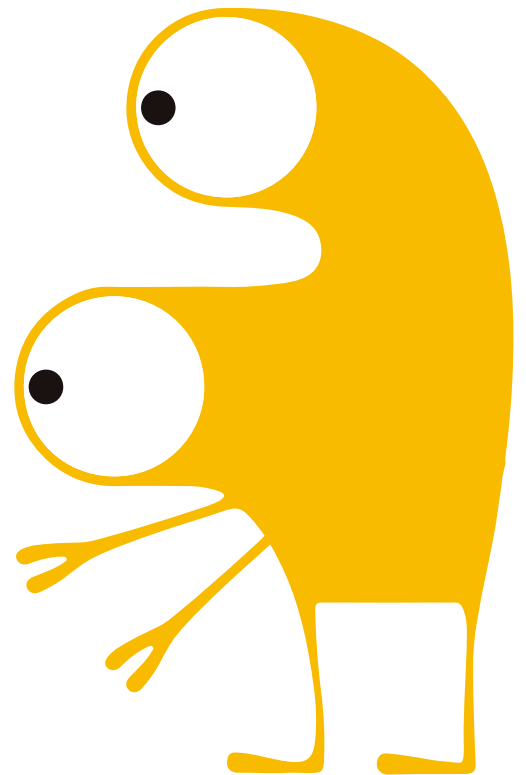
Scratch makes programming fun and easy. We'll show you how to Draw, animate, and create a shark attack game.

58 Python Programming

Now that you've tried Turtle Art and Scratch, we'll show you how to work with the powerful Python programming language.

70 Electronics

Build a scoreboard while you explore the breadboard and discover other tools for integrating your Pi with electrical circuits.



Extras

86 Scratch Animation

Use Scratch's built-in graphics editor to create animations for a racing game.

90 SonicPi

Make music and explore the world of digital sound with Sonic Pi.



See p8 for full details!

Your Roadmap to the Open Hardware Revolution ...

An exciting world of projects, tips, and skill-building tutorials awaits you in every issue of Raspberry Pi Geek.

Order your subscription today and tune in to the revolution!

shop.linuxnewmedia.com

Print Sub

Carry our easy-to-read print edition in your briefcase or backpack - or keep it around the lab as a permanent reference!

Digital Sub

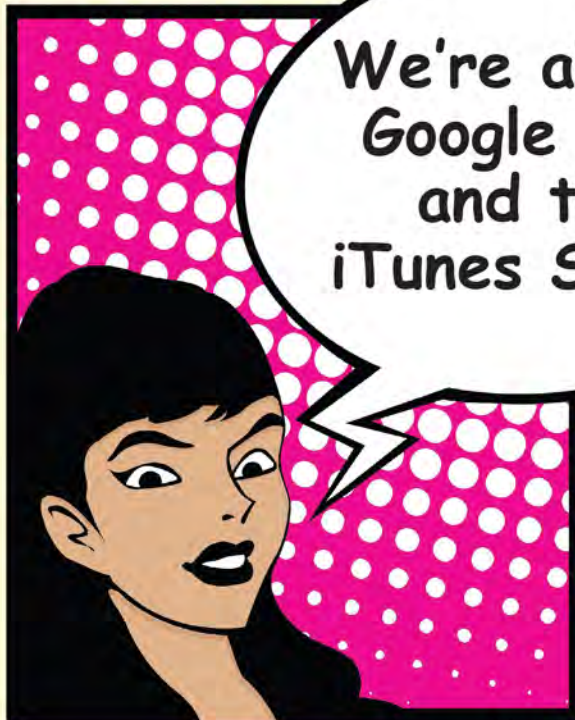
Our PDF edition is a convenient option for mobile readers.

6 print issues with 6 DVDs or
6 digital issues for only

\$59.95 £37.50 €44.90



We're also in
Google Play
and the
iTunes Store!



iTunes Store



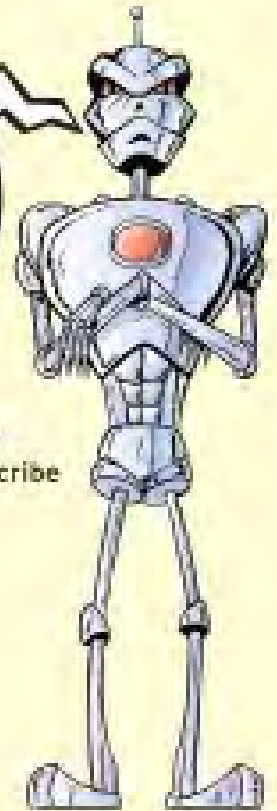
Google US



Google UK



Want to find out what's in the next issue?

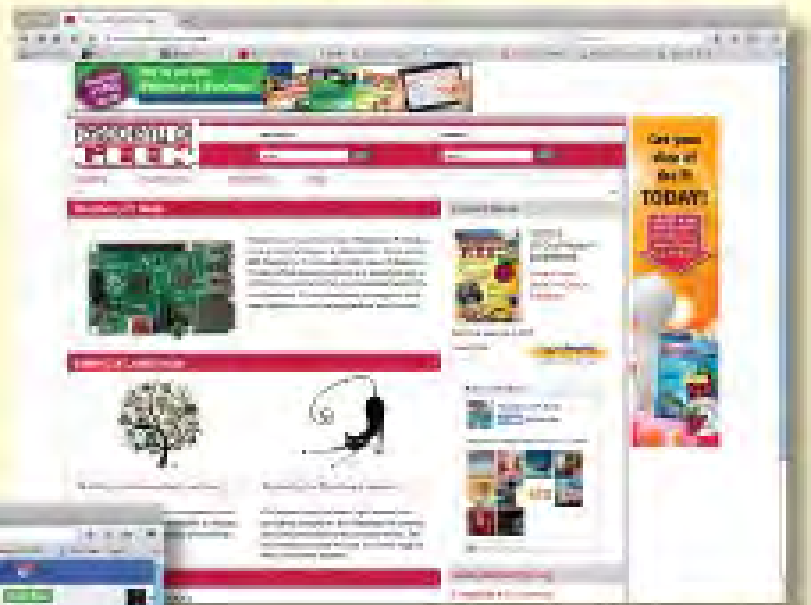


Sign up for our newsletter
www.raspberry-pi-geek.com/mc/subscribe

Visit us online
www.raspberry-pi-geek.com



Like us on Facebook
www.facebook.com/RasPiGeek



Follow us on Twitter
[@RasPi_Geek](https://twitter.com/RasPi_Geek)

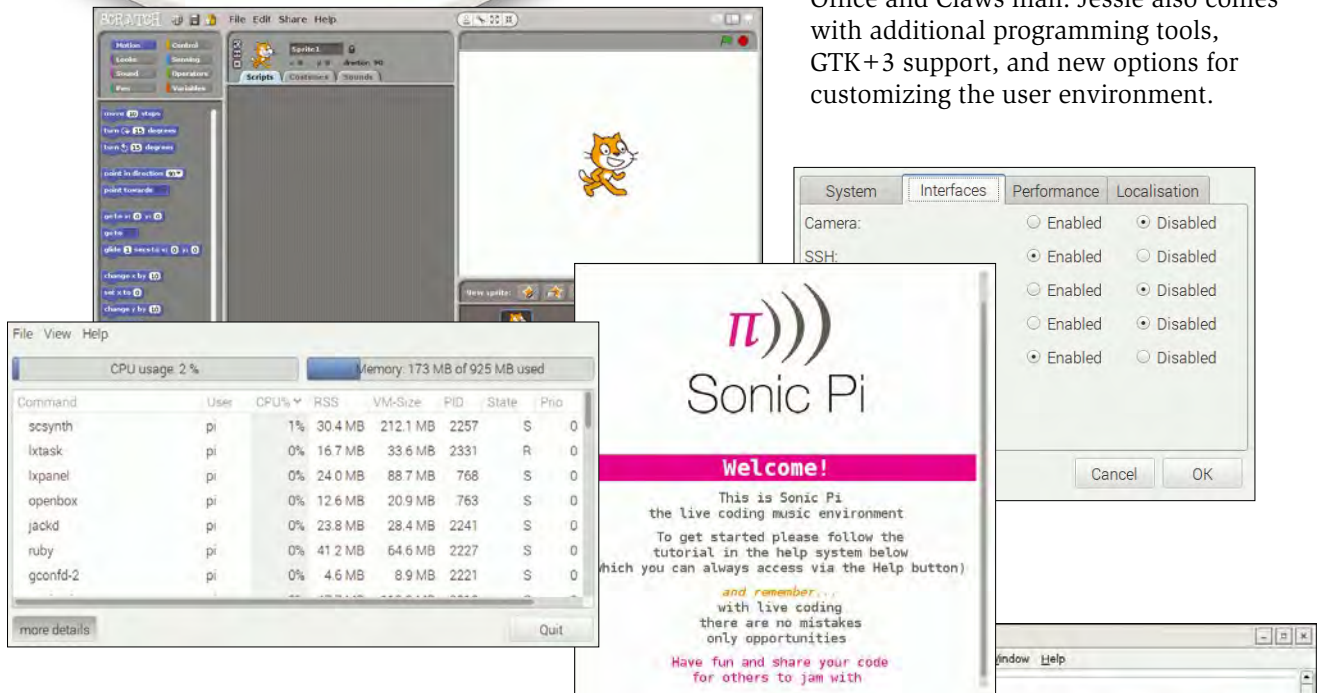


On the DVD



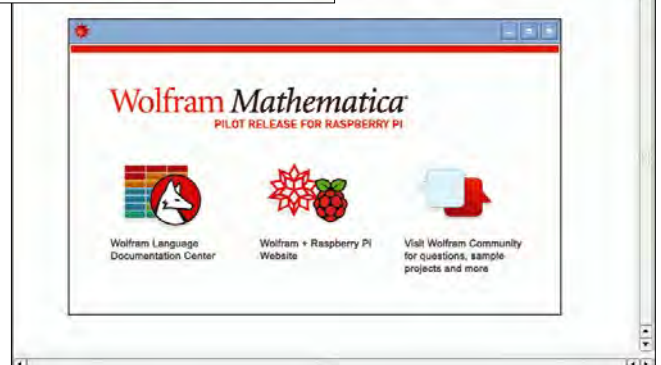
THE DVD ATTACHED to this issue includes the latest release of the Raspbian Linux, the “official operating system” of the Raspberry Pi Foundation. The brand new Raspbian Jessie release is based Debian Linux 8 “Jessie” and comes with many new applications and updates.

The system now boots directly to the desktop and includes the convenient GUI-based Raspberry Pi Configuration Tool for managing configuration settings. Rolled into the new release are convenient desktop tools, such as LibreOffice and Claws mail. Jessie also comes with additional programming tools, GTK+3 support, and new options for customizing the user environment.



RESOURCES

- [1] Raspbian: <https://www.raspbian.org/>
- [2] Raspbian Documentation: <https://www.raspbian.org/RaspbianDocumentation>
- [3] “Jessie is Here”: <https://www.raspberrypi.org/blog/raspbian-jessie-is-here/>
- [4] Raspberry Pi Forums: <https://www.raspberrypi.org/forums/>





RISE HIGHER

EACH ISSUE OF DRUPAL WATCHDOG OFFERS TOOLS, TIPS, AND BEST PRACTICES FOR BETTER DRUPAL WEBSITES.

NOW
PUBLISHED
4 TIMES
PER
YEAR!



Renew or subscribe now!

SUBSCRIPTIONS NOW AVAILABE WORLDWIDE!

Visit <http://drupalwatchdog.com/subscribe>

ASSEMBLING AND STARTING
YOUR RASPBERRY PI



Getting Started

BEFORE YOU START YOUR FIRST ADVENTURE, YOU'LL NEED TO SET UP YOUR RASPBERRY PI AND INSTALL THE OPERATING SYSTEM.

BY PAUL C. BROWN, JOSEPH GUARINO, AND JOE CASAD

THE FIRST TASK you'll face with your new Raspberry Pi is to get the system up and running. You'll need to find the necessary parts, plug in all the cables, and obtain or prepare an SD card with a suitable Raspberry Pi operating system. This article describes how to get your system up and running. If you already have a working Raspberry Pi system, you might want to skip this article and move on to the adventures ahead.

WHICH PI?

The Raspberry Pi comes in several models (see Table 1; the Raspberry Pi Compute Module, which is intended for embedded systems prototyping, is not shown in the table). Most boards are

based on the Model B, with two or more USB ports, Ethernet, and a camera interface. In the fall of 2012, the Raspberry Pi Foundation revised the design of the Raspberry Pi 1; these boards were called Rev 2 systems. They have a slightly different layout, including some differences in the configuration of the GPIO pins.

The version 1 Model A and A+ (RPI1A/A+) are designed for low-resource, low-cost scenarios and are lighter on memory – but lighter on power usage, as well. The Model B+ (RPIB+), introduced in July 2014, became popular for its two extra USB ports and 14 extra GPIO pins, but it was shortly trumped in February 2015 when the Raspberry Pi 2 (RPI2) arrived with a 900MHz quad-core processor and 1GB of RAM. One year later, the Raspberry

TABLE 1: Raspberry Pi Consumer Models

	RPI1A	RPI1A+	RPI1B Rev 2	RPIB+	RPI2	RPI3	RPI Zero
Target Price	\$25	\$20	\$35	\$35	\$35	\$35	\$5
CPU	700MHz ARMv6	700MHz ARMv6	700MHz ARMv6	700MHz ARMv6	900MHz quad-core ARMv7	1.2GHz quad-core ARMv8, 64-bit	1GHz ARMv6
Memory (SDRAM; Shared with GPU)	256MB	256MB	512MB	512MB	1GB	1GB	512MB
USB Ports	1	1	2	4	4	4	1 micro-USB
SD Storage	SD MMC	MicroSDHC	SD MMC	MicroSDHC	MicroSDHC	MicroSDHC	MicroSDHC
Power Rating	300mA (1.5W)	200mA (1W)	700mA (3.5W)	600mA (3.0W)	800mA (4.0W)	800mA (4.0W)	~ 160mA (0.8W)

Lead Image © Alexandr Aleabiev, 123RF.com

Pi 3 (RPi3) offered not only increased speed, with a 1.2GHz 64-bit quad-core ARMv8 processor, but integrated WiFi and Bluetooth to boot.

The very small Raspberry Pi Zero (~2.5x1.25 inches), revealed in November 2015, diverts from the usual RPi form factor, offering a limited number of ports, pre-RPi2 RAM, no camera or LCD display connectors, and no Ethernet. However, it maintains the 40-pin GPIO and costs just \$5. The RPi Zero uses a micro-USB (OTG) port, which supports communication to such things as mice, keyboards, speakers, and video devices through a powered USB hub. Pi Zero users also need to buy a supported WiFi dongle for network connectivity.

BUY SOME PI

You can buy the Raspberry Pi from a number of resellers. The Raspberry Pi website links to a page that lets you browse official vendors around the world [1]. Keep in mind that when you buy a Pi, a few more purchases are required in addition to the unit itself. See the box titled “Parts List” for a shopping list.

GETTING RASPBIAN

You might have acquired a Raspberry Pi in kit form, with a protective case for the Pi, some cables, and an SD card pre-loaded with Raspbian. If you already have an SD card and you’re impatient to get your Pi up and running, you might want to skip this section for the time being and go directly to *First Boot*.

This article describes how to install the Raspbian operating system on your Raspberry Pi. Raspbian is officially sponsored by the Raspberry Pi Foundation, and you can download it from their site [3]. Raspbian is free software, so you can download it for no charge, and once you get it, you are free to copy and distribute it. The DVD attached to this issue also includes Raspbian. The version of Raspbian on the DVD is Raspbian Jessie. If already have a Raspbian system running on your Raspberry Pi, you might be running Raspbian Wheezy, which came before the Jessie edition.

Regardless of whether you get the Raspbian image file from a website or

PARTS LIST

The tiny Raspberry Pi computer-on-a-board won’t do much for you all alone. Be sure you have the following on hand before you start:

- 1 Raspberry Pi
- 1 Power supply
- 1 MMC SD or microSD card – depending on the model of your Rasp Pi.
- 1 USB mouse (optional if you will use the command line only)
- 1 USB keyboard
- 1 DVI- or VGA-to-HDMI converter (if needed) to connect your monitor via HDMI.
- 1 case (optional).
- 1 SD card writer. (Many contemporary computers already have an SD card writer, but if you don’t, buy a card writer or buy an SD card with Raspbian already on it.)

Notes and caveats on the purchase list:

- Power supply – The Pi model B needs a 5V micro USB power supply, but not just any USB cable will do. As with most things, the devil is in the details. If you were to connect your Pi to any average micro USB cable, it would not work. Specifically, model B needs the higher current USB power of 700mA, which is often found in many conventional phone chargers. This is important for system stability.

Failing to pay attention to the details of these power requirements will leave you with lockups and other system problems. See the Raspberry Pi compatible device list [2] for more information.

- Powered USB hub – Old Rasp Pi systems only had two USB ports, which meant no room for additional devices once you connected your keyboard and mouse. RPi2/3 and B+ models have four USB ports, which allow for easier expansion without an external USB hub; however, some users prefer a powered USB hub anyway to help minimize the risk of drawing too much power and causing stability issues.

from the DVD, the steps for setting up your SD card are similar.

If you download an OS file from the Internet, you might want to start with an integrity check to be sure the file arrived in its original state. See the box titled “Integrity Check.”

GETTING AN SD CARD

The Raspberry Pi uses an SD card as a storage medium. Several Raspberry Pi vendors will send you an SD card with Raspbian already installed. If you already have Raspbian on an SD card, you can skip this section. As you’ll see when you read this section, the task of burning an image on an SD card requires several steps and can be confusing. If you’re new



too this kind of thing and want to keep it simple, you might want to consider obtaining a preinstalled Raspbian SD.

The unzipped Raspbian image file is not a standard file but contains the Raspbian OS already installed and half configured. This type of file is called an *image file* because it represents a disk image. You can't just copy the file over to the SD card.

Some SD cards come preformatted. If your card is not formatted yet, you'll need to format it with the FAT32 filesystem. Windows and Mac OS systems have built-in tools for formatting partitions. The SD Association recommends that Windows and Mac OS users use their free formatting tool, which is designed specifically for SD cards. You can find the SD formatter at the SD Association website [5].

Linux users have several options for how to format an SD card. Running the following command with superuser privileges:

```
lsblk
```

will list block devices mounted on the system. (This procedure assumes your Linux system is configured to auto-mount disks.) The SD card will appear in the list as type "Disk." Partitions on the disk will appear below the device in a tree structure. Look for a disk with a

size that is approximately equal to the size of your SD card, and try to verify that this is the correct device. (Be sure you don't choose a hard disk!)

Enter the following command to format the card as FAT32:

```
mkdosfs -F 32 -v <device_name>
```

For instance, if the SD card device has the name `mmcblk0`, enter:

```
mkdosfs -F 32 -v mmcblk0
```

To create a working SD card with Raspbian, you have to do a byte-by-byte copy from the file to the card. Byte-by-byte copies are risky if you are not sure what you're doing because you completely erase the data on the destination drive. If you get confused and choose the wrong destination (e.g., you choose a partition on your hard disk instead of your SD card), you could lose your data. That said, making byte-by-byte copies is not difficult, and you just have to pay attention at the key moments of the process.

If you are using Windows, you can use a program called *Win32 Disk Imager* [6]. Download the program and install it. Plug your SD card into your computer (if you don't have an SD card reader slot, you can purchase a cheap external adapter that will plug into a USB port) and note where Windows mounts the card (see Table 1 for the correct SD card type to use with your Rasp Pi model). If you look at Figure 1, you can see that, in our lab, Windows assigned the card to the `H:` drive.

Now you can use the Win32 Disk Imager program to do a byte-by-byte copy from the Raspbian file to the card. In the *Image File* text box, navigate to the image file, for example, look for `2015-09-24-raspbian-jessie.img`, and in the *Device* drop-down box, choose the letter assigned to your card (in the case shown in Figure 1, you would choose `H:`) and start copying. Please note that all data on the destination drive will be lost, so use an empty card or one that contains data you don't mind deleting.

Both Linux and Mac OS X can use their terminal windows and the `dd` tool to do the copy. You'll need the name assigned to the SD card. See the previous

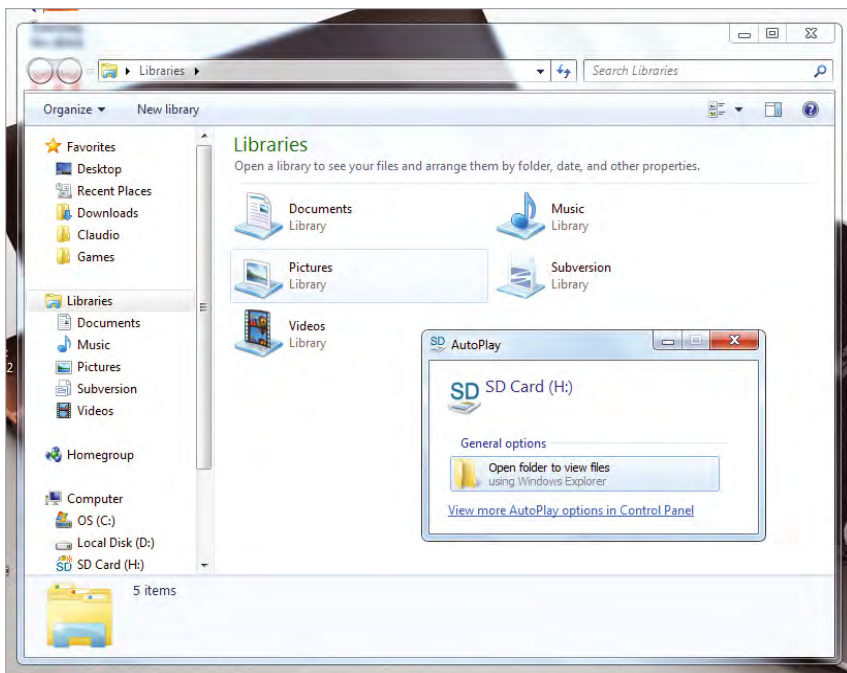


FIGURE 1: In this case, Windows has assigned the SD Card to drive H:.

discussion on finding the name. Another Linux command you can use to find the disk name is `dmesg`. To find the SD card using `dmesg`, plug the card into the computer and immediately run the `dmesg` command from a terminal window:

```
$ dmesg
...
[... ] mmc0: new high speed SDHC card ↗
      at address 59b4
[... ] mmcblk0: mmc:59b4 7.4GiB
[... ] mmcblk0: p1 p2 < p5 p6 > p3
...
```

The last lines will show you where the operating system found your card. Here, the card appears at `/dev/mmcblk0`.

Be sure you have the right drive! The formatting process destroys all data on the drive. To use `dd`, the card must be *unmounted*, that is, not assigned to a directory in the filesystem. You can run

```
sudo umount /dev/mmcblk0
```

to unmount it. If the card was not mounted to start with, `umount` will just display an error and exit.

Now navigate to the directory where you downloaded Raspbian with:

```
cd /Raspbian/download/directory
```

where `/Raspbian/download/directory` is the path to your Raspbian image on Linux. Then, you can carry out the byte-by-byte copy to the card with:

```
$ sudo dd bs=1m ↗
  if=2015-02-16-raspbian-wheezy.img ↗
  of=/dev/sdb
```

On the Mac, first make sure the SD card has been formatted in FAT32 and run the following command:

```
diskutil list
```

Look for the device formatted FAT32 (Figure 2) and note its device name (here, `disk1`). Unmount the disk:

```
diskutil unmountDisk /dev/disk1
```

Next, navigate to the directory where you downloaded Raspbian with:

```
cd ~/Downloads
```

INTEGRITY CHECK

The creators of a file often provide an *SHA-1 hash*, which is the long string of letters and numbers you see under the link on the download page. Running a program on your downloaded file will produce a similar string. If both the string on the site and the string you produce locally are the same, the file is okay. On Windows, you can download a program [4] to use on the command line for checking file integrity. Linux and Mac OS X have preinstalled tools for integrity checks. On Linux, simply type the following in a terminal window:

```
sha1sum 2015-02-16-raspbian-wheezy.zip
```

On Mac OS X, open a terminal and type:

```
openssl sha1 2015-02-16-raspbian-wheezy.zip
```

Once you're happy that the file you have downloaded is okay, you can unzip it and install it on an SD card.

Then, you can carry out the byte-by-byte copy to the card with:

```
$ sudo dd bs=1m ↗
  if=2015-02-16-raspbian-wheezy.img ↗
  of=/dev/rdisk1
```

(Note the use of `rdisk1` rather than `disk1`.) This copy will take a while, and `dd` will not show any sort of progress bar.

ASSEMBLING YOUR PI

Once you have loaded the operating system onto your SD card, assemble your Pi into the case of your choice. Cases are optional, but they do help you keep your

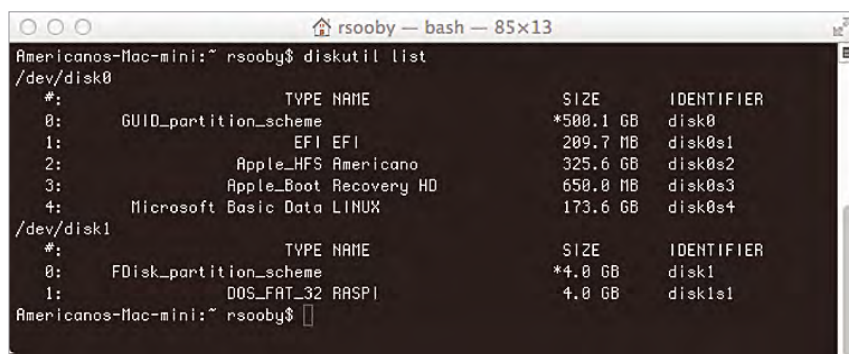


FIGURE 2. The SD card (`/dev/disk1`) was formatted FAT32 and named RASPI with the Mac SD Formatter tool.

Pi safe from accidental damage or electrostatic discharge.

Next, insert the SD card into the bottom of the Rasp Pi. You can only insert the card one way for it to function. Connect your monitor to the Raspberry Pi via the HDMI port. If you have an older monitor that has only VGA or DVI, you will need a cable that converts to HDMI.

If you're running a first-generation Pi system, you might want to attach a powered USB hub to increase the available USB ports. Even if you are using a B+ or Pi2B board (with four available ports), you might prefer to use a powered USB hub to avoid potential instability associated with drawing too much power. Next, plug in an Ethernet cable for your local LAN and, finally, plug in your power supply.

FIRST BOOT: JESSIE

Earlier versions of Raspbian launched a complicated configuration dialog at first boot, allowing the user to configure the locale, camera, login, and other settings. (See the next section on booting Debian Wheezy.) With the latest Jessie release, the Raspbian developers have simplified the first login by eliminating the choices and booting the system to a default configuration.

Just slip the SD card into the slot and turn on the power. The system will boot directly to the Raspbian desktop. You can then make the necessary configuration changes through the available configuration tools. In particular, the Raspberry Pi Configuration Tool was developed to encapsulate many of the settings a user might want to address when starting a new system.

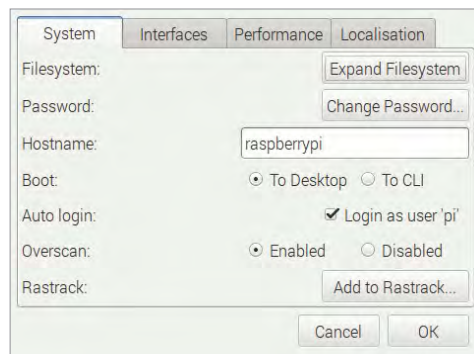
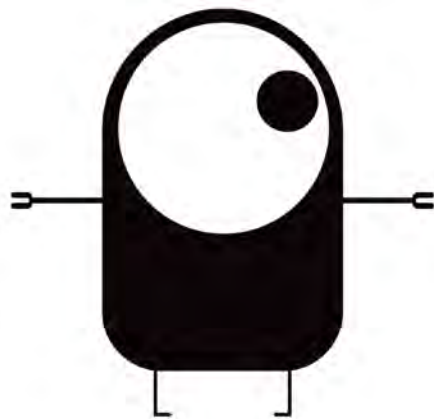


FIGURE 3: The Raspberry Pi Configuration Tool System tab lets you configure login settings and define system parameters.

To open the Raspberry Pi Configuration Tool, click on the *Menu* button in the upper-left corner of the Raspbian desktop and choose *Preferences | Raspberry Pi Configuration Tool*. The tool opens to the *System* tab (Figure 3), which lets you manage the following options:

- **Expand Filesystem** – in some cases, Raspbian is only able to use part of the space on the SD card. This option expands into all the available space. Your configuration might already be using all the space, in which case a message will tell you this option isn't necessary.
- **Change Password** – the default password for the default account is `raspberrypi`, which isn't much of a password and, even if it were, everyone knows it. If you are concerned about the security of your system, change the default password. Be sure you write down your password or have a clear means for remembering it.
- **Boot to desktop or CLI** – The articles in this issue assume you are using a graphic desktop system. If you are an experienced user and would prefer to boot to the command line (CLI means "Command Line Interface"), select the CLI option. A checkbox below lets you choose whether you want the system to prompt the user for login credentials at startup or log in automatically. See the Wheezy discussion in the next section for more on the *Overscan (Under-scan)* option.

The *Interfaces* tab (Figure 4) lets you choose whether to enable the camera and SSH, which you will learn about in later articles. The *Performance* tab lets you choose whether to overclock the Raspberry Pi system. Overclocking increases the speed but can pose issues for stability, cooling, and durability of your

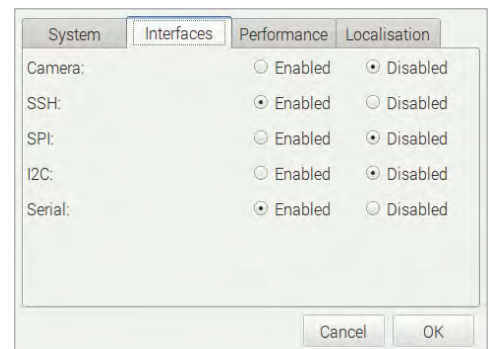


FIGURE 4: The Configuration Tool's Interfaces tab lets you enable or disable the Raspi Pi camera and SSH.

system. You won't need overclocking for the exercises in this issue. Another option in the *Performance* tab lets you set the amount of memory assigned to the Graphics User Interface (GPU). You probably won't need to change this setting either, but if you're using your Raspberry Pi for graphic-intensive applications, such as gaming or video playback, you might want to experiment with increasing the GPU memory.

The *Localisation* tab includes settings that vary based on your location, such as the language, timezone, and keyboard layout.

The next article in this issue discusses some other options for customizing your user environment, such as adding new applications and changing the appearance of your desktop.

FIRST BOOT: WHEEZY

If you created or obtained your Raspbian SD card sometime before you bought this issue, the chances are your SD contains the Raspbian Wheezy edition, which is based on Debian Linux 7 "Wheezy." The first time you boot up Raspbian Wheezy on your Pi, you'll see a bunch of text scroll by before you reach the `raspi-config` tool (Figure 5).

From the configuration tool, you can set up the initial default configuration of your Pi. This might sound like a major decision-making moment, but don't worry: You can run the tool at any time from a command line with `sudo raspi-config` if you are not happy with the settings.

To navigate the menu, use the up and down arrow keys on your keyboard to highlight options, then use the Tab key to skip to the *Select* button to continue to the next screen. All the screens leading off the main menu screen work in the same way.

The first option, *Expand Filesystem*, lets you decide whether you want to expand the root filesystem so that it takes up all the space on the SD card. For now, go ahead and choose this option. The Raspbian image takes up just less than 2GB. Most SD cards are 8GB or more nowadays. If you don't expand the root filesystem, you'll have 6GB or more that you won't be able to use.

The *Change User Password* option allows you to change the administrator user's default password. The administrator

(known as *superuser* or *root* in Linux parlance) has complete control over the computer, and anyone who has access to superuser privileges can cause serious damage. Therefore, if your Raspberry Pi is going to be used by more than one user or is going to be open to a local network or the Internet (e.g., as a server), changing the default password is a good idea. The default administrator's username, by the way, is *pi* and the default password is *raspberry*.

The third option, *Enable Boot to Desktop/Scratch*, lets you boot into the graphical desktop, boot directly into the Scratch programming environment, or continue to boot to the command prompt. This special issue assumes you are running your system using the desktop option, but if you feel like experimenting with running your Pi from a command-line interface, you can always start the desktop later if you need it. (See the box titled "Starting and Stopping from the Command Line.")

Internationalisation Options allow you to change localization settings. The first option in this category, *Change Locale*, sets the language, country, character set, sort order, and so on. When you arrow or page down into the correct box, press the Spacebar to choose.

You can mark more than one locale. If you want to deselect the UK (*en-GB*) English default, find its box marked with an asterisk and press the spacebar to deselect. On the next screen, set the default language from among the choices, then Tab to *Ok*.

Change Timezone lets you choose the time zone you want to use with your Pi. If your Raspberry Pi is connected to the

STARTING AND STOPPING FROM THE COMMAND LINE

If you elect to operate your Raspberry Pi from the command line, you can always start the graphical desktop after you boot by typing

```
startx
```

at the command line.

To power down your Pi from the command line, type

```
sudo halt
```

and hit Enter. If you want to restart your Pi, type:

```
sudo reboot
```

If you just want to log out, type:

```
exit
```

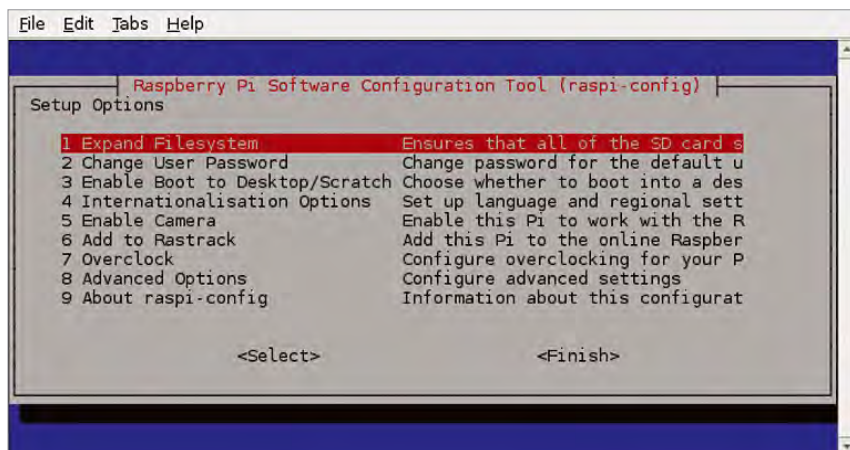


FIGURE 5: The Raspbian configuration tool lets you manage configuration settings for your Pi.

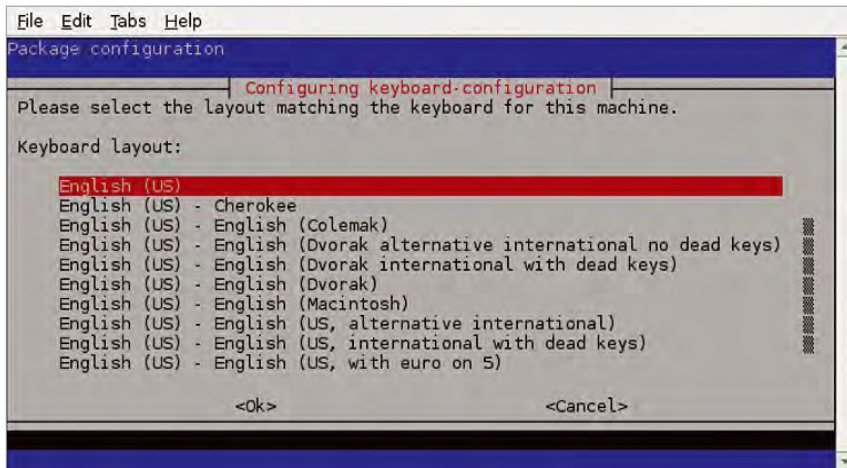


FIGURE 6. You can easily configure the layout and language of your keyboard under *Internationalisation Options* | *Change Keyboard Layout*.

Internet, Raspbian will automatically try to contact an NNTP server to get the correct GMT time and then add or subtract the number of hours to calculate your local time.

The final localization option, *Change Keyboard Layout*, allows you to establish the model, layout, and language of the keyboard you are using. (If the console kicks out errors and shows that *en-GB.UTF-8* is still the preferred language, refer to the “Bug Box” sidebar.) If you don’t see your keyboard in the first screen, arrow down to *Other* and Tab to *Ok*. After a few seconds, you will see a long list of keyboard models. *Generic 105-key (Intl) PC* is the default option

and is the most common keyboard layout. Once you have chosen your keyboard layout, use the Tab key to highlight *Ok* and hit Enter.

In the following screen, you can choose your keyboard’s language, and, in the next screen, the variant of your language (Figure 6). If you select *English (US)*, for example, you will then be able to choose from among a wide range of variants and keyboard models. The next screens let

you define your keyboard’s modifier key and Compose key (Multi-key) and whether you want to use Ctrl + Alt + Backspace to terminate the X server (graphical environment).

The *Enable Camera* option allows you to enable or disable the Raspberry Pi camera add-on [7].

You’ve probably heard of *overclocking* before. Most computers ship with the CPU set to a certain speed, but the microprocessor is capable of running much faster. Overclocking is the technique by which you make a microprocessor run at a higher speed than was originally configured. However, overclocking has its risks. It can lead to instabilities within the system, and higher speeds usually mean that the electronics heat up beyond their design specs, which means that overclocking can lead to components burning out. Overclocking can also shorten the processor’s life, so your Raspberry Pi won’t last as long. My advice is to leave the *Overclock* option alone until you are sure you need the extra speed and understand all the risks.

Advanced Options includes *OverScan*, which allows you to establish a black border around the screen. This option was useful when monitors and TVs had a physical plastic or wooden border that overlapped the viewing area. *OverScan* made the viewing area smaller, thus avoiding information being cut off by the physical border. Most modern monitors don’t have this problem or have their own configuration menus to shrink or move the viewing area. The default is *Disable*, which you should probably leave as is.

The Raspberry Pi has two processors onboard. One is the CPU, which does all the general calculations and executes most of the commands, and the other is the graphics processor, or GPU, which is used to render graphics in games, play videos, and so forth. In bigger machines, each processor has its own, separate memory, but not so with the Pi: On the Model A and Model B Rev 1, you have 256MB of RAM; on the Model B Rev 2, you have 512MB. That RAM has to be shared between the CPU and the GPU. The advanced *Memory Split* option therefore allows you to decide which processor gets what.

For everyday use, the defaults should be okay. But, if you are going to run GPU-intensive programs, such as 3D

BUG BOX

Sometimes, the configuration utility does not change from *en-GB.UTF-8* to your keyboard language of choice (e.g., *en-US.UTF-8*) and throws error messages. If this happens to you, choose *Cancel* to get out of the keyboard configuration window, then Tab to *Finish*, and say *No* to a reboot. You need to edit the `.bashrc` file. To do so, enter:

```
sudo nano /home/pi/.bashrc
```

Arrow down to a blank line and enter:

```
export LC_ALL=C
```

Now press Ctrl+X; hit y[es] to save and Enter to save to the same file name. Next, enter

```
sudo raspi-config
```

to return to the configuration program. You should now be able to change your keyboard settings under *Internationalisation Options*.

games or a Kodi media center application, you might want to give the graphics processor a bit more RAM.

SSH (short for Secure SHell) is a protocol that allows you to log in to a computer from a remote location over a secure and encrypted “SSH tunnel.” If you enable SSH on your Pi, you can administer your Pi from another computer. See the “Networking” box for more details.

Another advanced option is *Update*. This option looks online for updates for the `raspi-config` program and, if they exist, downloads and installs them.

When you’re done configuring, use the Tab key to highlight the *Finish* button and hit Enter to exit the tool. Raspbian will ask whether you want to reboot. Answering Yes reboots immediately. Answering No applies your changes on the next bootup.

Next time you boot into Raspbian, you will see the text login screen or GUI, depending on the choices you just made in the configuration program. Either way, if there is something you would like to change, you can run the Raspbian configuration tool with

```
sudo raspi-config
```

at any time from the command line or from a terminal window on the desktop.

NETWORKING

In the default configuration, Raspberry Pi will join the LAN by requesting an IP address from a DHCP server. (In most home environments, the local router/firewall device acts as a DHCP server, assigning IP addresses to the hosts on the LAN.)

This configuration is fine for simple configurations, but if you want to put your Pi to work as a web server or other server system, or if you want to access the system through SSH without checking the address every time you log in, you might want to set your system up with a permanent static IP address. To begin, open the `/etc/network/interfaces` file with a text editor:

```
cd /etc/network
pi@raspberrypi~$ sudo nano interfaces
```

Replace the line `iface eth0 inet dhcp` with `iface eth0 inet static` and add the IP address, netmask, and gateway address you want to use for the Raspberry Pi. The address, netmask, and gateway will depend on the address configuration for your network. For more informa-

CONCLUSION

Raspbian is designed for easy configuration and installation. Once you get the system on your SD card, the rest is easy, and if you purchased an SD card with Raspbian preinstalled, the task is even easier. Raspbian Jessie lets you change the configuration later using the applets in the *Preferences* window. Jessie also supports the `raspi-config` utility described in the section on Raspbian Wheezy. For either Wheezy or Jessie, you can start up `raspi-config` at any time by entering `sudo raspi-config` at the command line.

Now that your system is up and running, get ready for your first adventure! *

INFO

- [1] RS vendors by country: <https://www.raspberrypi.org/help/faqs/#buyingWhere>
- [2] RPi VerifiedPeripherals: http://elinux.org/RPi_VerifiedPeripherals
- [3] Download Raspbian from the Raspberry Pi’s official site: <http://www.raspberrypi.org/downloads>
- [4] SHA1Sum tool for Windows: <http://www.softpedia.com/progDownload/SHA1Sum-Download-143137.html>
- [5] SD Formatter for Windows and MacOS: https://www.sdcard.org/downloads/formatter_4/



- [6] Win32 Disk Imager: <http://sourceforge.net/projects/win32diskimager/>
- [7] The Raspberry Pi camera: <https://www.raspberrypi.org/products/camera-module-v2/>
- [8] Network configuration in the Debian wiki: <http://wiki.debian.org/NetworkConfiguration>

tion, see your router configuration or consult an online tutorial on TCP/IP addressing. The following example shows a sample entry for an address in the `192.168.77.0` address space:

```
iface eth0 inet static
address 192.168.77.50
netmask 255.255.255.0
gateway 192.168.77.1
```

If you configure a static IP address, you’ll need to tell your Rasp Pi system where to find a DNS server. To set up name resolution, edit `resolv.conf` in `/etc` to point to the DNS server for the network:

```
pi@raspberrypi~$ sudo nano resolv.conf
nameserver 192.168.77.1
```

Several useful tutorials on Linux networking are available online. Raspbian is based on the Debian Linux distribution, so the Debian wiki is a good source for networking information [8].

Exploring the Raspbian Operating System

FIRST LOOK AROUND

THE ARTICLES in this issue assume you are running the Raspbian operating system on your Raspberry Pi. Raspberry Pi also supports several other operating systems. Most of these alternative systems are also based on Linux, and many of the concepts are similar, however, non-Linux systems such as RISC OS and FreeBSD also support the Raspberry Pi, and you can even find a Raspberry Pi version of Windows called Windows 10 IoT Core.

Raspbian [1] is actually a version of Debian Linux that was developed specifically for the Raspberry Pi. Raspbian is the Raspberry Pi Foundation's "officially supported operating system," and it is recommended for beginning users.

The DVD attached to this issue contains the Raspbian "Jessie" version, which became available just as this issue was going to press. If you already own a Raspberry Pi and have already made a Raspbian SD card (as described in the preceding article), your system is probably running the previous Raspbian Wheezy edition. Many things are similar in "Jessie" and "Wheezy," but some differences exist between the two versions, and you might need to improvise as you

work through some of the instructions you find in this issue.

In general, when you're working with a Linux system, it is a good idea to get comfortable with asking questions and looking for help online. The Help forums at the Raspberry Pi website [2] are a good place to start.

This article provides a brief first look at the Raspbian user interface and introduces some concepts you'll need for the adventures later in this issue: working with the terminal and installing new applications.

DISCOVER RASPBIAN

Boot your Raspbian system, then sit back and wait while a lot of text scrolls over the screen. The text consists of status messages as various components of the system start up.

Raspbian Jessie systems (and older systems configured for desktop startup) will boot directly to the Raspbian desktop (Figure 1). If your system boots to a command line, start up the graphic user interface and desktop system by typing the command `startx` then hitting Enter. (See the preceding article for more on

WE SHOW HOW TO WORK WITH
THE TERMINAL AND INSTALL
APPLICATIONS ON YOUR
RASPBERRY PI
BY JOE CASAD, PAUL C BROWN,
HEIKE JURZIK

Lead Image © Alexandr Alecbiev, 123RF.com

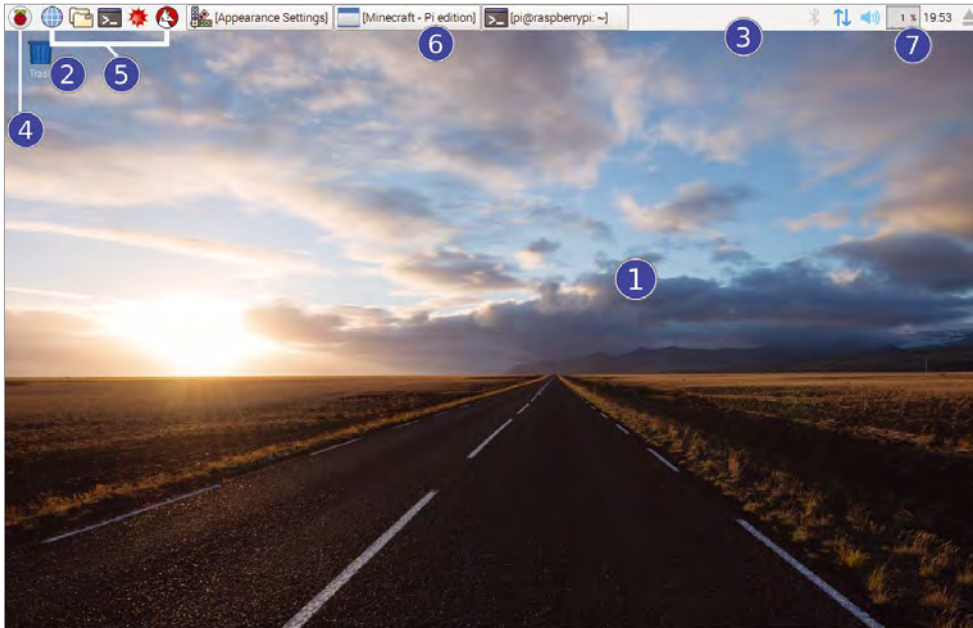


FIGURE 1: The parts of the default Raspbian desktop: (1) wallpaper, (2) trash can, (3) panel, (4) menu, (5) app launcher, (6) taskbar, (7) status bar.

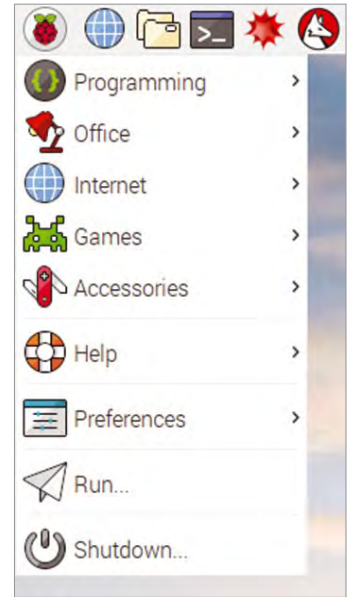


FIGURE 2: The Raspbian launcher menu groups applications into categories for easy access.

how to configure your system to boot directly to the desktop.)

Figure 1 shows some important components of the default desktop window. Right-click an empty space on the desktop for a context menu with options that allow you to modify the appearance. Choose *Desktop Preferences* to change the wallpaper, default fonts, and other features of the display.

A brand-new Raspbian installation comes with a single trash can icon in the upper left-hand corner (Figure 1.2). You can also add other icons to the desktop for easy access to favorite applications. (See the “Personalizing” section later in this article.)

Along the top is a *panel* (Figure 1.3) with a menu launcher in the top left corner (Figure 1.4). Starting from the left, you will see an application bar (Figure 1.5) that contains icons to launch the a web browser, a file manager, a terminal, the Wolfram Mathematica app [3], and a link to the Wolfram scripting language.

The middle section of the panel is reserved for the *taskbar* (Figure 1.6), which shows the windows you have open or minimized. The far right shows your connectivity, volume, CPU usage, and time (with a drop-down calendar), along with an eject button for media (Figure 1.7).

The raspberry icon in the upper-left corner (Figure 1.4) lets you launch applications quickly and easily; it is similar to the *Start* button on many Windows

systems. Click the button to reach a list of preconfigured application categories (Figure 2). You can browse through a selection of preinstalled programming tools, office tools, Internet applications, games, and accessories.

The applications in the *Preferences* menu let you choose options for customizing your Raspberry Pi configuration (Figure 3).

PERSONALIZING

Right-click on the panel and choose *Panel Settings* from the pop-up menu to change

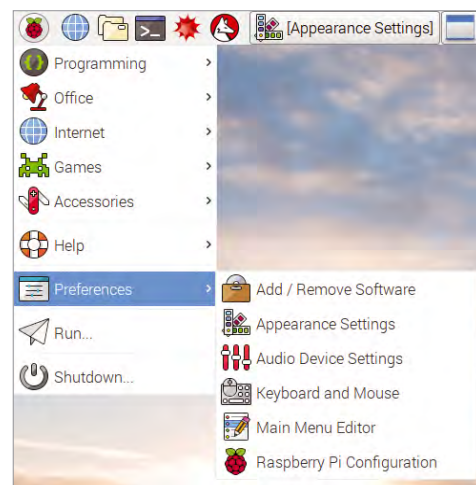
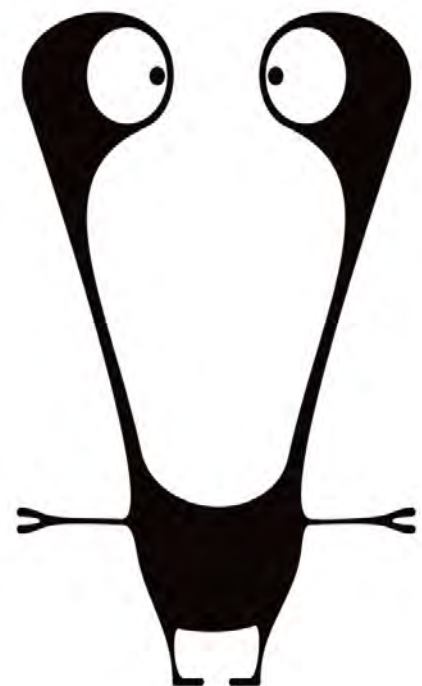


FIGURE 3: The Preferences menu offers tools for configuring your Raspbian environment. Raspbian Jessie provides some configuration options you won’t find in earlier versions.



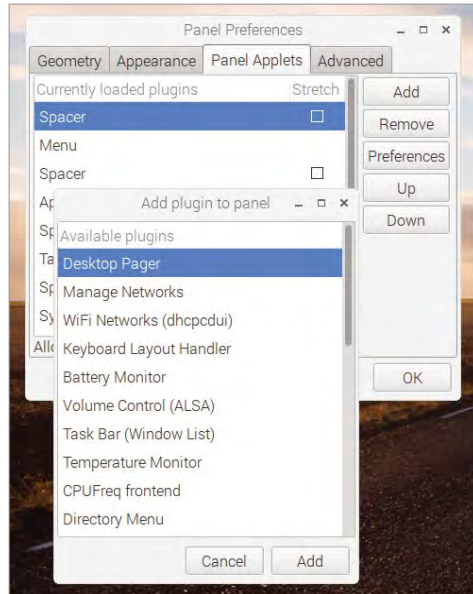


FIGURE 4: Adding new items to the panel is easy.

what the panel looks like, add more items, move items around, and delete the ones you don't want. In the *Geometry* tab, you can choose the size and position of the panel and its icons; in *Appearance*, you can select the color and theme; in *Panel Applets*, you can add more items and sort them using the *Up* and *Down* buttons (Figure 4). (In the default panel, *Up* means "move one position to the left" and down means "move one position to the right.")

Many of the items within the panel are also configurable. For example, right-clicking in the launch bar brings up a menu that allows you to add app launcher, which is very useful for creating shortcuts to apps you use often.

On Raspbian Jessie systems, you can also change some desktop settings using by clicking the menu button and choosing *Preferences | Appearance Settings* (Figure 5).

As with Windows and Mac OS systems, you can add an icon for an application you use frequently to the desktop for easy access. Open your file manager (the second icon from the left in

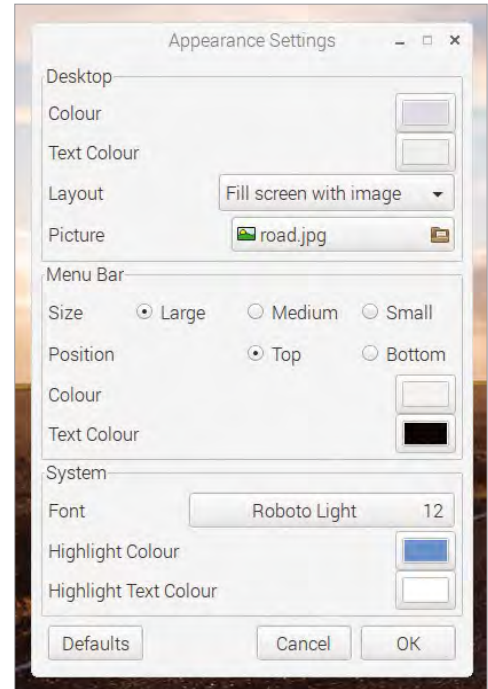


FIGURE 5: Configuring desktop appearance settings in Raspbian Jessie.

the default application bar – it looks like a two folders). Scroll down below the home folder subdirectories to a folder marked with a slash (/). Click the plus button next to the / folder for access to the complete filesystem, and navigate to the `/usr/share/applications` folder. This folder includes icons for all your installed apps. Right-click on the application you want to put on your desktop and select *Copy* from the pop-up menu, then right-click on an

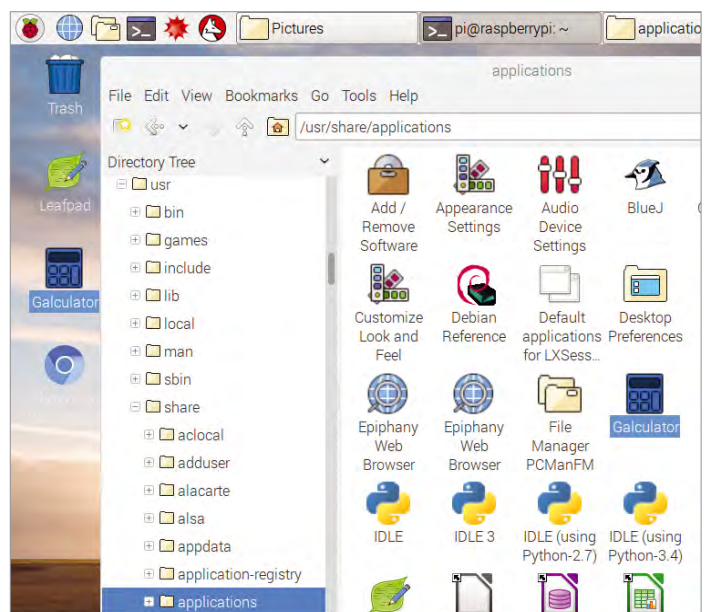


FIGURE 6: You can add icons to the desktop using *Copy* and *Paste*.

empty space on your desktop and select *Paste*. That's it, or at least that's the easy way.

Raspbian also gives you the ability to work with several desktop spaces at the same time. You can configure each desktop as a different workspace, with different applications and different wallpaper. (Picture this feature as being like having two desks in your room – one for homework and one for hobby projects.)

Raspbian lets you have up to 16 desktops, but four is probably overkill, and it could overload your Pi and make it run slowly. Many users are fine with just one desktop, but if you feel like experimenting, *use the scroll wheel* to click in an empty space on the desktop and choose *Add new desktop*.

Also, you can spin the mouse wheel to navigate between desktops, or you can add the Desktop Pager to your panel and then just click on the desktop you want to visit in the pager. To add the desktop pager, right-click on the panel, select *Add/Remove Panel Items*, and click the *Add* button. Choose *Desktop Pager* in the list and click *Add*.

You can move an active window to a different desktop by right-clicking on the title bar and choosing *Send to desktop* (Figure 7). Or, drag the window to the side of the current desktop to move it to another desktop.

Experienced users find it is often much faster to work in a terminal emulator (which is just called *the terminal* for short) than to click around on icons and windows. Perhaps more importantly, building an application for a modern GUI interface requires a lot of additional programming time and effort that doesn't really add much additional functionality.

In the free-wheeling world of Linux, where most of the applications are available for no cost and many are maintained and tested by volunteers, it simply isn't worth the effort to invest a lot of time and energy into building an elaborate GUI-based application to handle a simple task. Consequently, the Linux user environment includes many tools that work best

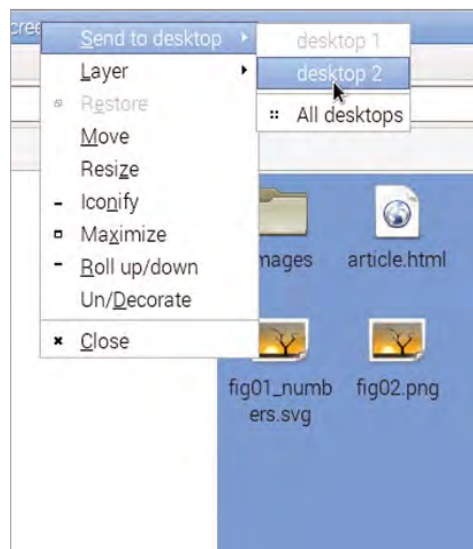


FIGURE 7: You can send a window to different desktops by right-clicking on the title bar.

WORKING WITH THE TERMINAL

Today's users are accustomed to pointing with a mouse and clicking on icons in a graphic user interface like the Raspbian desktop. In an earlier era, however, before computers got as fast and powerful as they are today, users interacted with the computer using text-based commands. A *terminal* was a big, bulky device that allowed the user to enter text-based commands at a keyboard and view text-based output from the computer.

Computers don't use text-based terminals anymore, however, most operating systems (including Windows, Mac OS, and the Linux system that is the basis for Raspbian), support what is called a *terminal emulator*, a window-based application that looks like an old-fashioned terminal and can read and respond to terminal-based commands.

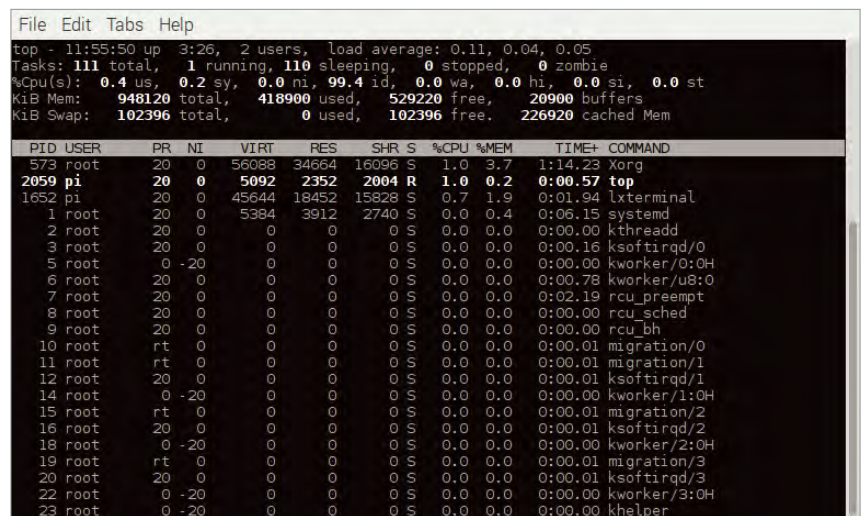


FIGURE 8: Like the terminals of old, the modern-day terminal emulator receives text-based commands from the user and often displays output directly to the screen.

CHANGE YOUR PASSWORD

To change your password in Raspbian Jessie, click the menu button and select *Preferences | Raspberry Pi Configuration*. In the Raspberry Pi Configuration dialog box, click the *Change Password* button in the *System* tab.

In Raspbian Wheezy, open a terminal window and enter:

```
sudo raspi-config
```

to access the Raspbian configuration tool (see the preceding article for more on configuring Wheezy with the configuration tool.

You can also change your password using the `passwd` command. You'll need to enter your old password before you can change it.

from a terminal, and Linux users are accustomed to working with the terminal, which is also called the “shell” or “command shell” or “console.”

At several places in this issue, you will be asked to “open a terminal” and enter a text-based command. In Raspbian, the terminal window is easy to reach. Just click the icon in the panel that shows a monitor with a black screen (refer to Figure 1.5). The terminal window looks like an old-fashioned terminal (Figure 8).

The cursor sits next to a dollar sign (\$), which marks the space for entering a command. The dollar sign is called a command prompt. If you read or hear the instruction to enter a command “at the command prompt,” that means open a terminal window and type the command. Always hit Enter after you enter a command to send the command to the computer.

Many commands work just fine at the privilege level of the basic user, however, important commands that could potentially damage the system or cause security issues require additional privileges. To run a command at an elevated privilege level, precede the command

with the `sudo` command. For instance, to delete the user *clyde* from the system (Figure 9):

```
sudo deluser clyde
```

Depending on which account runs the command and how your system is configured, you might be prompted to enter a password with the `sudo` command. As you learned in the preceding article, the default user account is called *pi* and the default password is *raspberry*. It is a good idea to change any default password, including your Raspberry Pi's default password – but don't forget it. (See the box titled *Change Your Password*.)

Raspbian uses a structured system of directories similar to what you are used to if you work with Windows or Mac OS. Commands sometimes work best if you run them from a specific directory. If you work with the command line, you'll need to learn to move around within the directory structure.

When you open a terminal window, most likely the terminal will open in your home directory. Type `ls` to list the contents of your directory. You can use the `cd` (change directory) command to move to another directory. You'll also need to mention the path to the target directory:

```
$ cd /home/pi/Documents
```

You can use a dot (.) in the path to represent the current directory. In other words, you could move from your home directory to the *Music* subdirectory by typing:

```
$ cd ./Music
```

A double dot means “go back one level in the directory path,” so if you want to go from the `/home/pi/Music` directory back to your home directory (`/home/pi`), you could type:

```
$ cd ..
```

Many systems also use the tilde character (~) to represent the home directory, so no matter where you are, you can always return to your home directory with:

```
$ cd ~
```

```
File Edit Tabs Help
pi@raspberrypi ~ $ deluser clyde
/usr/sbin/deluser: Only root may remove a user or group from the system.
pi@raspberrypi ~ $ sudo deluser clyde
Removing user `clyde' ...
warning: group `clyde' has no more members.
Done.
pi@raspberrypi ~ $ █
```

FIGURE 9. If the command you want to use requires a higher privilege level, use the `sudo` command.

```
File Edit Tabs Help
DELUSER(8) System Manager's Manual DELUSER(8)
NAME
deluser, delgroup - remove a user or group from the system
SYNOPSIS
deluser [options] [--force] [--remove-home] [--remove-all-files]
[--backup] [--backup-to DIR] user
deluser --group [options] group
delgroup [options] [--only-if-empty] group
deluser [options] user group
COMMON OPTIONS
[--quiet] [--system] [--help] [--version] [--conf FILE]
DESCRIPTION
deluser and delgroup remove users and groups from the system according
to command line options and configuration information in
/etc/deluser.conf and /etc/adduser.conf. They are friendlier front
ends to the userdel and groupdel programs, removing the home directory
as option or even all files on the system owned by the user to be
removed, running a custom script, and other features. deluser and del-
group can be run in one of three modes:
Manual page deluser(8) line 1 (press h for help or q to quit)
```

FIGURE 10. Use `man` to reach a quick summary of help information on a terminal command.

If you get lost when you are navigating around in the directory structure, you can always enter the `pwd` (print working directory) command to display the name of the current directory.

To create a new directory, enter the `mkdir` command with the name you want to give to the directory:

```
$ mkdir /home/pi/Music/Beatles
```

Or, if you were already in the `Music` directory, you could just type:

```
$ mkdir ./Beatles
```

The `cp` command lets you copy files (the angle brackets, `<>`, indicate a parameter that you supply):

```
cp <source_file> <destination_file>
```

The default is to look in the current directory; however, you can include a path with the source or destination to copy to or from a different directory. Of course, you must have the necessary permissions to access the directory.

The `mv` instruction moves files or whole directories from one place to another. If the instruction is used on files or directories that are not moving, it renames them. For example,

```
mv file1 dir/
```

will move `file1` into directory `dir/` hanging off the current directory. But

```
mv file1 file2
```

will change `file1`'s name, renaming it `file2`.

To delete a file, use the `rm` (remove) command, and to delete a directory, use `rm -r` or `rmdir`. Needless to say, you must be careful how you use these commands.

A summary of these basic commands is shown in Table 1. Each of the commands includes additional options. A utility called `man` provides a quick description and a summary of command-line options, which you can see by typing `man <command>`. For instance, to obtain information on the `deluser` command used in Figure 9, enter:

```
man deluser
```

APT PACKAGE MANAGER

Another task you'll need to learn when you're working with Raspbian is adding applications. Because of the space limitations of the sparse, SD-based storage system, Raspbian is preconfigured with only a minimal set of applications. Luckily, it is easy to add new applications to your Raspbian system.

Most Linux systems make new applications available through a system of Internet-based servers known as app repositories. These repositories are the predecessors of the app stores you use to obtain new programs for your smartphone.

Raspbian's APT (Advanced Package Tool) system lets you download and install new applications easily.

Do not go hunting for software on the Internet without checking the Raspbian repositories first. Linux users download and install the majority of their software from approved and reliable repositories and very, *very* rarely stray onto the web to download apps. Furthermore, Raspbian comes with a set of official online repositories already configured, so, if you need to install new programs, you can get started right away.

An application is stored in the repository in the form of package. The package contains the application itself, plus sometimes some supporting software and instructions for

TABLE 1: Some Basic Terminal Commands

Command	Action
<code>ls</code>	List contents of the current directory
<code>cd</code>	Change directory
<code>pwd</code>	Show current working directory
<code>mkdir</code>	Make directory
<code>cp</code>	Copy file(s)
<code>mv</code>	Move or rename a file or directory
<code>rm</code>	Remove file(s)
<code>rmdir</code>	Remove directory

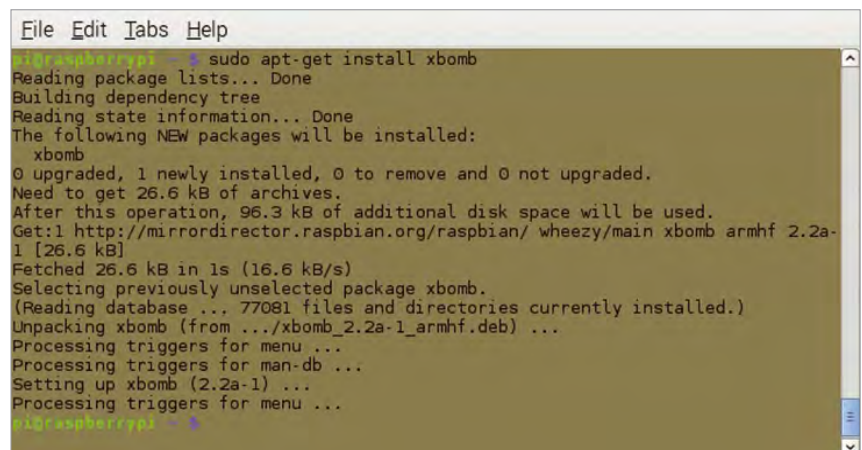


FIGURE 11: The `apt-get` tool allows you to install software from the Raspbian repositories.

the computer on how to configure the application. The basic command for adding a software package is simply

```
sudo apt-get install <package-name>
```

When you enter the command, you usually get a complete summary of what will happen if you go through with the installation, including the dependencies that will be installed, the packages that will be upgraded and removed, and the amount of disk space that will be required (Figure 11). Unless the action can proceed automatically without affecting anything else, you have the choice of continuing the process or not. Just to be sure what you typed doesn't include any unpleasant surprises, you should read the summary carefully before continuing.

As `apt-get` works, it shows which package it is downloading and its progress, as well as the download speed and the amount of time required to finish the operation. The times are only estimates and will change as the Internet connection speed changes. Once the downloads are complete, `apt-get` installs the software, sometimes pausing to ask questions about how you want it installed.

After everything is done, `apt-get` exits with a summary of any problems that it encountered. As a final touch, if the software you just installed is a graphical application, it is added to your desktop menus.

You can remove packages by using

```
sudo apt-get remove <package-name>
```

but this process could leave configuration files behind. If you want to get rid of all traces of a package, use

```
sudo apt-get purge <package-name>
```

The instruction `sudo apt-get autoremove` will remove all old and unused packages.

If you want to check for updates for all the software installed on your Pi, use

```
$ apt-get update
```

and Apt will track down all the newer versions in the online repositories and install them for you.

The Apt system includes several other tools, but by far, the most useful package utility is `apt-cache`, which offers a

treasury of information about packages and your system. For example,

```
apt-cache showpkg <packagename>
```

shows which version is installed, the latest version available in the repositories you are using, and the reverse dependencies of the packages (i.e., which packages depend on it).

Similarly, `apt-cache dump` lists all the packages you have installed, and `apt-cache stats` offers information such as the number of installed packages and the total number of dependencies. An especially useful option is `apt-cache search <string>`, which tracks down the exact name of a package or packages that you might want to install by searching for the `<string>` you provide as an argument. If you wanted to install, say, a Minesweeper-type game, you could type

```
$ apt-cache search minesweep
```

and Apt will return all programs that contain the word "minesweep" in its name or description.

RUNNING HEADLESS WITH SSH

Most of the articles in this issue assume your Raspberry Pi computer is connected directly to a keyboard and mouse. However, many users don't have a spare keyboard and mouse to use for their Raspberry Pi. Also, in some cases, it might be easier to talk remotely to your Raspberry Pi from another computer on the network. SSH (short for "Secure Shell") lets you send commands to your Raspberry Pi system from another computer on the network.

The first step is to make sure your Raspberry Pi is configured to act as an SSH server. The SSH server is enabled

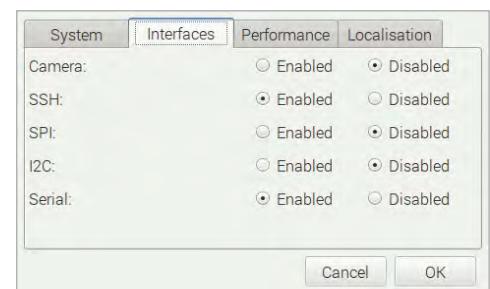


FIGURE 12: Enabling SSH in Raspbian Jessie.



by default on many Raspbian systems. On Raspbian Jessie systems, click the Start button and select *Preferences | Raspberry Pi Configuration*. Click the *Interfaces* tab and be sure SSH is enabled (Figure 12).

On Raspbian Wheezy systems, you have a chance to enable or disable SSH as one of the *Advanced Options* when you start the system for the first time (see the preceding article for more on configuring Wheezy systems). If you disabled SSH initially and want to enable it later, open a terminal window and enter:

```
sudo raspi-config
```

to relaunch the configuration dialog discussed in the previous article. Look for SSH in the *Advanced Options*.

You'll also need to make sure the other computer you are using to connect to the Raspberry Pi is configured to act as an SSH client. Windows, Mac OS, and Linux all support SSH, and often the SSH client is enabled by default in the terminal emulator program. See the vendor documentation for your operating system.

To start the SSH connection to connect to the Raspberry Pi, you need to find out the Rasp Pi's IP address. If you wanted to connect a keyboard and mouse to the Rasp Pi temporarily, you could determine the IP address using the `ifconfig` command. However, since the point of this section is to explain how to connect *without* putting a keyboard and mouse on your Raspberry Pi, I'll offer another approach, although you'll need to know a little about home networking.

My test machine has an Ethernet port and a USB wireless adapter. I've connected the to the local network via a router. The router assigns IP addresses to devices on the network using the DHCP protocol (the default approach for most home routers) [4].

After booting the device, I visited the router's configuration web interface and had a look at the DHCP settings listed for the connected devices. Figure 13 shows my local setup at home: Apart from a Macbook (hostname *lion*, IP 192.168.2.38), the Raspberry Pi with the hostname *raspbpi* is connected via Ethernet (192.168.2.48) and WLAN (192.168.2.49).

To connect via SSH, I use the following command:

```
ssh username@IP-address
```

Replace *username* with the name of your account on the Raspberry Pi (the default login name is *pi*) and *IP-address* with the number you found out earlier:

```
ssh pi@192.168.2.49
password for pi:
```

The standard password is *raspberry* (unless you have changed it, which is a good idea). Note that you won't see any visual feedback when you type it. After you've successfully logged in, you will see the command line prompt:

```
pi@raspbpi ~ $
```

After you have logged in via SSH, you can use most of the terminal commands discussed earlier in this article. In the standard configuration, the user *pi* can gain admin privileges with the *sudo* command and its own password:

```
pi@raspbpi ~ $ sudo -i
[sudo] password for pi:
root@raspbpi:~#
```

Alternatively, you can put `sudo` in front of a single command that needs admin privileges.

Note that, if your system assigns IP addresses using DHCP, the address could change the next time you use your Raspberry Pi. See the preceding article for more on how to configure your IP to have a permanent, static IP address. Note that you'll also have to set up your router to reserve the address. See your router documentation.

CONCLUSION

This article introduced the Raspberry Pi user interface and described how to install applications and work with the terminal. These skills will serve you well as you start your later adventures with the Raspberry Pi. ✨



INFO

- [1] Raspbian <http://raspbian.org/>
- [2] Raspbian Forums: <https://www.raspberrypi.org/forums/>
- [3] Wolfram Mathematica: <http://www.wolfram.com/raspberrypi/>
- [4] DHCP: https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol

lion	192.168.2.38	3C:15:C2:D9:64:22	WLAN 130 Mbit/s
raspbpi	192.168.2.48	B8:27:EB:68:A3:D1	LAN 4 100 Mbit/s
raspbpi	192.168.2.49	74:DA:38:3E:D4:3A	WLAN 150 Mbit/s

FIGURE 13: The test machine with the hostname “raspbpi” is connected to a router and shows up in the list of DHCP clients.

APACHE, HTML, AND MORE

Express yourself!

IN THIS WORKSHOP, YOU WILL LEARN HOW TO INSTALL A WEB SERVER ON YOUR RASPBERRY PI AND BUILD A SIMPLE WEBSITE WITH BASIC HTML COMMANDS.

BY HEIKE JURZIK AND JOE CASAD

THE INTERNET is a vast network of computers reaching across the whole world. Much of the Internet experience you know today occurs through a system of tools and technologies known as the World Wide Web. What you know of as *the web* is really just a huge collection of computers communicating with a common set of rules.

Like many other Internet technologies, the web operates through the basic interaction of a client and server. When you surf to a website, your web browser (which is actually a web client) asks a web server for the contents of a web page (Figure 1). The server sends the information back, and the web browser assembles the parts of the page (text, pictures, and links) to display in the browser window (Figure 2).

The communication system the web client and web server use to talk to each other is called Hypertext Transfer Protocol (HTTP), and the system for encoding information on the format and contents of a web page is called Hypertext Markup Language (HTML).

Busy modern websites incorporate several other additional steps and features. The HTML data is generated dynamically, rather than stored in an ordinary file. Communication between the client and server is often encrypted, and other services, like a database server, might hold all or part of the data. But the classic interaction shown in Figure 1, where a client requests a web page that the server delivers the page in the form of an HTML document, is the fundamental process at the foundation of the World Wide Web.

Like almost any other computer in the world today, your Raspberry Pi system can easily act as a web client. Just click on the globe icon in the Raspbian top menu bar to



open the Epiphany web browser, and enter a web address in the address bar. (You have to be connected to the Internet.) Behind the scenes, the browser application sends a request in HTTP to the web server specified in the address bar. The web server returns a reply in HTTP. Over the course of the communication, the web server will transmit the contents of the web page, encoded in HTML format. The browser will receive the HTML file and will use the information in the file to assemble the contents into the finished page you view in your browser window.

Your Raspberry Pi can also act as a web server. Of course, your tiny Pi is not fast enough or powerful enough to be one of those big servers providing content to thousands of users on the open Internet. Also, the complexity of the security configuration for a real Internet web server means the Pi is not a suitable candidate for building an actual Internet-ready production server system. But you can still set up a Rasp Pi computer as a web server for your own home network. You can experiment with creating your own web pages and use your Pi web server to post information for your family, like schedules, notes, and favorite recipes.

As you work on building your Rasp Pi website, you will learn how web servers work and get some first-hand experience with HTML.

UNDERSTANDING URLS

Everything on the web has an address. The familiar address string you have seen in the address bar of your browser is known as a Universal Resource Locator (URL). The URL is actually a special case of a more general form called a Universal Resource Indicator (URI); however, the term URL has persisted and is still in common use.

The basic form for a URL is:

```
scheme://domain/path
```

The `scheme` specifies the protocol used for the communicating with the resource. The web uses the HTTP protocol, so the scheme is `http`.

The domain is typically the domain name of the network where the resource resides, as described by the Internet's DNS naming system. Familiar names

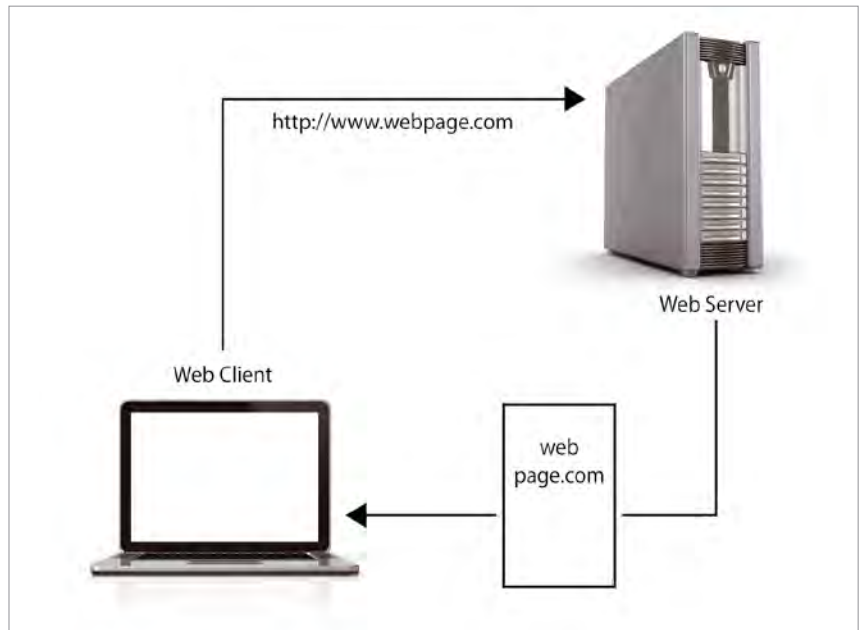


FIGURE 1: Communication on the web follows a client-server model. The client asks for a resource such as a web page, and the server transmits that resource in the form of an HTML document.

like *www.google.com* and *www.whitehouse.gov* are DNS domain names. The tasks of configuring and registering a DNS name is not necessary for a little Rasp Pi web server on a home network. Instead, you can just use the IP address of the Raspberry Pi system.

The path is the series of subdirectories leading to the file, along with the filename. If the URL does not include a path, the web server will look for the de-

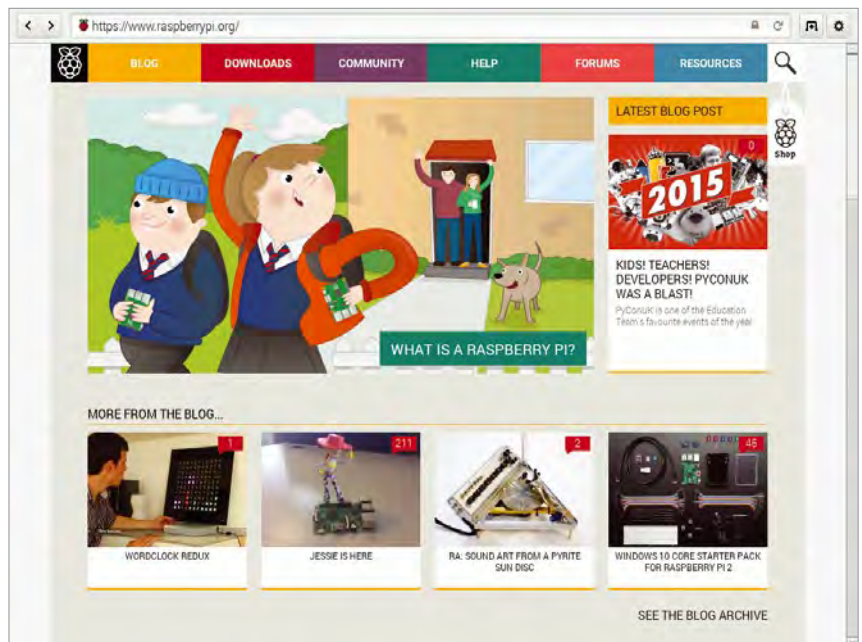


FIGURE 2: To request a web page, the user enters a URL in the address bar of a web browser. The web server sends back an HTML document, which the browser assembles into the graphical display the user sees in the browser.

```

pi@raspbpi: ~ $ sudo apt-get install apache2
[sudo] password for pi:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap liblua5.1-0 ssl-cert
Suggested packages:
  apache2-doc apache2-suexec-pristine apache2-suexec-custom openssl-blacklist
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap liblua5.1-0 ssl-cert
0 upgraded, 10 newly installed, 0 to remove and 2 not upgraded.
Need to get 1746 kB of archives.
After this operation, 5235 kB of additional disk space will be used.
Do you want to continue? [Y/n]
* raspbi * | huhn | 12:43 Thu | 0-$ bash | 1$* bash
    
```

FIGURE 3: To set up the Apache web server, simply install the package “apache2” and confirm the package manager’s selection and suggestions.

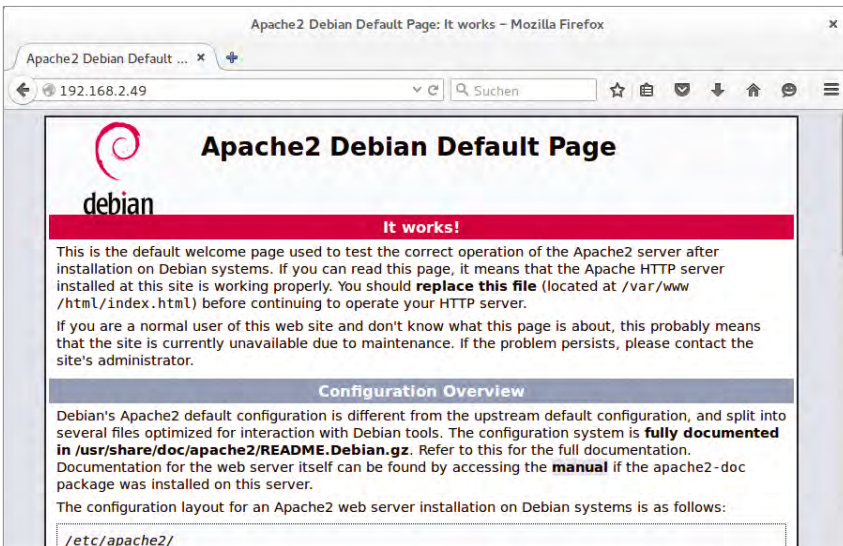


FIGURE 4: Open the default website either on the Pi, or (as in this example) in a web browser on another computer on the network. The image shows the default page for Raspbian Jessie.

fault filename `index.html`. The “home page” for many websites is actually the `index.html` in the web server directory.

From your local network, you’ll be able to reach your Rasp Pi web server by entering the URL:

```
http://IP_address
```

`IP_address` is the IP address of your Raspberry Pi system. For example:

FINDING THE IP ADDRESS

To find the IP address of your Raspberry Pi system, open a terminal window on your Raspberry Pi and enter the `ifconfig` command (Figure 5). If you’re using a standard Ethernet cable to connect your Raspberry Pi with the network, use the IP address for the `eth0` interface, labeled “inet addr” in Figure 5.

```
http://192.168.2.49
```

Several other optional parameters can also be part of a URL. You could include a login name and password, as well as a port number, search terms, and other information. If you decide to continue to expand and develop your Raspberry Pi website, you could experiment with adding login information and creating a network path with access to other files and directories.

INTRODUCING APACHE

Apache is the most popular web server, and it runs on millions of computers around the world. Apache is also easy to install and use, and it is extremely well documented. The Apache web server is actually an application that runs on the web server computer.

Apache listens for incoming requests from browsers on the network requesting web pages. When Apache receives a request from a web client, it finds the HTML file specified in the URL that came with the request and transmits the HTML data back to the web client. Of course, as I mentioned before, big modern web servers behave in much more complicated ways, but this simple scenario is all you need for your Raspberry Pi web server system.

To install the web server, be sure your Raspberry Pi is connected to the Internet and type the following command:

```
sudo apt-get install apache2
```

As you can see in Figure 3, the package manager resolves all dependencies and offers to install additional software packages. Hit Y and Return to continue. After APT has downloaded, unpacked, installed, and configured the software, if all goes well, your new web server will start automatically.

Once the application has started, it will listen on the network for incoming HTTP requests. You can easily confirm

that the web server is running. In a browser window, on the Raspberry Pi, visit the address `http://localhost` to see the default web page. Alternatively, if you are accessing the web page from another computer on the local network, use the IP address of the Raspberry Pi system `http://IP-address`. For example, enter:

```
http://192.168.2.49
```

to visit the new site (Figure 4). See the box titled “Finding the IP Address” for more on finding the IP address for your Raspberry Pi system. The “Managing Apache” box provides some additional information of managing your Apache server system.

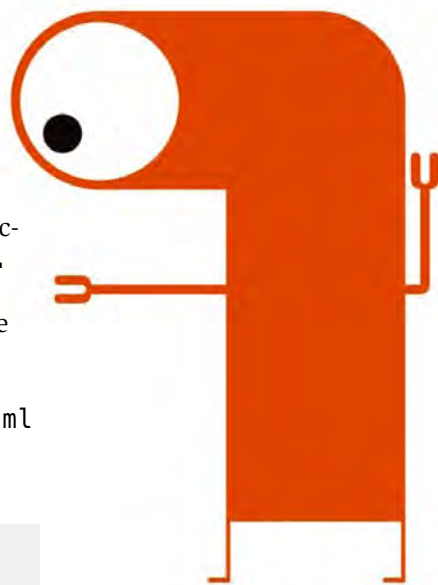
DISCOVERING HTML

Once your Apache web server is up and running, you can experiment with building your own web page. The default web page (refer to Figure 4) is located in `/var/www/index.html` if you are using Rasp-

bian Wheezy or in `/var/www/html/index.html` if you are running the more recent Raspbian Jessie edition. If you are using a different operating system on your Raspberry Pi, consult your vendor documentation (or just search for `index.html`).

The `index.html` file belongs to the user and group `root`. Before you change the `index.html` file, change into the directory with the `index.html` file and make a backup copy of the original:

```
cd /var/www/html
sudo cp /var/www/html/index.html \
/var/www/html/index.html.orig
[sudo] password for pi:
sudo nano index.html
```



```
pi@raspbpi: ~
pi@raspbpi ~ $ ifconfig
eth0    Link encap:Ethernet  HWaddr b8:27:eb:68:a3:d1
        inet addr:192.168.2.49  Bcast:192.168.2.255  Mask:255.255.255.0
        inet6 addr: fe80::ba27:ebff:fe68:a3d1/64 Scope:Link
        inet6 addr: 2002:542c:c9a4:0:ba27:ebff:fe68:a3d1/64 Scope:Global
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:7755 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1641 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:9524843 (9.0 MiB)  TX bytes:208792 (203.8 KiB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

FIGURE 5. Enter the `ifconfig` command on the Raspberry Pi system to discover the Pi’s IP address.

MANAGING APACHE

Once you install and reboot, the Apache server should start automatically. If you need to manually start, stop, or restart Apache, use the `systemctl` command. The following command:

```
sudo systemctl stop apache2.service
```

stops all Apache processes. This command:

```
sudo systemctl restart apache2.service
```

first stops and then restarts Apache (it will start the web server if it wasn’t running before). Another useful `systemctl` option is `status`, which gives you more information about the web server and even informs you if there are syntax errors in the configuration (Figure 6).

For your Raspberry Pi home web server, you should be able to use the standard Apache configuration settings. However, if you feel like experimenting with some of the parameters that define Apache’s behavior, the main Apache configuration file is `/etc/apache2/apache2.conf`. See the notes in the file for more on what the parameters do and how they affect your Apache installation.

LISTING 1: Hello, World!

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <title>Hello, World!</title>
05   </head>
06   <body>
07     <h1>This is a Heading</h1>
08     <p>This is a paragraph. Hello, World!</p>
09   </body>
10 </html>
```



FIGURE 8: HTML describes the structure of a document. The code in Listing 1 produces this simple web page.

```
This is a link to <a href=
"http://www.linuxpromagazine.com/">
Linux Pro Magazine</a>.
```

Experiment with adding links to your `index.html` file. You can link to an outside website, or, if you are in a building mood, create additional pages for your Raspberry Pi website by adding the `path` to the URL:

```
http://192.168.2.49/path
```

where `path` is a structure of directories leading to the file followed by the file-name.

For instance, you could create a subdirectory in the directory with your `index.html` file called `photos` and place a file in the `photos` subdirectory called `photos.html` with links to photos of your latest vacation. If the IP address of the Raspberry Pi machine is 192.168.2.49, the URL for the photos page would be:

```
http://192.168.2.49/photos/photos.html
```

The link on the `index.html` home page could say something like:

```
<P>Check out the photos of my
<a href="http://192.168.2.49/photos/
photos.html">wild and crazy</a>
vacation to Delaware!</p>
```

The `photos.html` file might contain links to the actual image files (as described in the next section), along with notes and commentary on favorite moments of the journey.

The photo files themselves could also go in the `photos` subdirectory for a convenient and portable solution. See the box titled “Absolute or Relative

Directory” for more on linking to directories in HTML.

TAGS AND ATTRIBUTES

An HTML attribute provides additional information, for example, a URL to an external website. Attributes come in handy when you want to embed a picture into a website.

First, you need the tag ``. Inside of the `` tag, you specify the source file (`src`), alternative text to add a short description for people who use a text browser or screen reader (`alt`), and two attributes that define the width and height:

```

```

Note that the `` tag has no end tag.

ABSOLUTE OR RELATIVE DIRECTORY

For clarity and simplicity, the example of the link to the photos subdirectory:

```
<a href="http://10.0.0.119/photos/photos.html">wild and crazy</a>
```

gives the full URL of the new page. This type of link is known as an *absolute link* because it provides the complete path to the resource from anywhere on the network. In practice, web designers are more likely to use a *relative link* to link to a subdirectory. A relative link just shows the part of the path that below the current directory to the location of the file. For instance:

```
<a href="/photos/photos.html">wild and crazy</a>
```

is the same link expressed as a relative link.

A relative link saves a little typing. Perhaps more importantly, a relative link is more portable: if you decide to move your web server to another computer, you can copy the accompanying structure of subdirectories, and the links will still work.

TABLE 1: HTML Tag Reference

Tag	Description
<!-- -->	A comment that is not visible on the website.
<!DOCTYPE>	Describes the document type.
<html>	Defines the root of the HTML document.
<head>	The header contains information about the document.
<title>	Sets a document title.
<body>	Definition of the document's body.
<h1>, <h2>, ... <h6>	Up to six different headings are possible.
<p>	Defines a paragraph.
<hr>	Prints a horizontal line.
	Bold text.
<i>	Italic text.
	Inserts an image.
<a>	Describes a hyperlink.
	Inserts an ordered list.
	An unordered list.
	Defines a list item for both list types, ordered and unordered.

Sometimes, you might want to add a comment to your HTML document that is only visible in the source code, not in the browser:

```
<h1>This is a Heading</h1>
<!-- will insert correct heading later -->
```

Although HTML was mainly designed to describe the structure of a website, it supports simple formatting tags. To indicate bold or italic text, use `` or `<i>`:

```
<p>This is <i>italic</i>;
this is <b>bold</b>.</p>
```

To insert a horizontal line in an HTML page, use `<hr>`. Like the `` tag, `<hr>` has no end tag. It's also possible to define unordered (bulleted) and ordered lists. Both require the outside tags for the list definition (`` and ``) and the `` tag for the list items:

```
<ul>
  <li>apples</li>
  <li>pears</li>
  <li>bananas</li>
</ul>
<hr>
<ol>
  <li>apples</li>
  <li>pears</li>
  <li>bananas</li>
</ol>
```

Of course, the HTML specification has many more tags, and if you combine HTML with CSS (Cascading Style Sheets) or a scripting language, you can build complex websites with dynamically created content. W3Schools [5] offers some interesting tutorials with lots of examples for HTML, CSS, and more. Table 1 shows a list of interesting tags and their meanings. *

INFO

- [1] Raspbian: <https://www.raspbian.org/>
- [2] PuTTY download: <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>
- [3] Apache: <http://httpd.apache.org/>
- [4] Upgrading Apache from version 2.2 to 2.4: <http://httpd.apache.org/docs/2.4/upgrading.html>
- [5] W3Schools: <http://www.w3schools.com/>



MORE UBUNTU!

FREE INSIDE!
\$39.99 VALUE

UBUNTU USER ARCHIVE
3,000 ARTICLES FROM 2004-2016

All Ubuntu User ever in one Massive Archive!
Celebrate 25 years of Linux with every article published in Ubuntu User on one DVD

UBUNTU user
EXPLORING THE WORLD OF UBUNTU

**SET UP YOUR VERY OWN ONLINE STORAGE
YOUR CLOUD**

- Choose between the best cloud software
- Access your home cloud from the Internet
- Configure secure and encrypted connections
- Set up synchronized and shared folders
- Add plugins for more features

PLUS

- Learn all about **Snap** and **Flatpak**, the new self-contained package systems
- Professional photo-editing with **GIMP**: masks and repairs
- Play spectacular **3D games** using Valve's **Steam**
- Discover **Dasher**, the accessible hands-free **keyboard**

DISCOVERY GUIDE

- New to Ubuntu? Check out our special section for first-time users! p. 83
- How to install Ubuntu 16.04
- Get all your multimedia working
- Go online with NetworkManager
- Package management

Pretty Polly
Discover Parrot, the Ubuntu-based security distro

USE \$7.99
CASH \$9.99
Fall 2016

FALL 2016 WWW.UBUNTU-USER.COM

Can't get enough Ubuntu? We've got a whole lot more!

Ubuntu User is your roadmap to the Ubuntu community. In the pages of **Ubuntu User**, you'll learn about the latest tools, best tricks, and newest developments in the Ubuntu story.

Ubuntu User helps you explore the treasures of open source software within Ubuntu's expansive repositories. We'll bring you exclusive interviews with Ubuntu leaders, keep you current on the exciting Ubuntu community, and answer your most perplexing Ubuntu questions. Learn how to choose a video editor, find the perfect tool to customize your desktop, and configure and manage Ubuntu systems using the best admin tools.

DON'T MISS ANOTHER ISSUE!

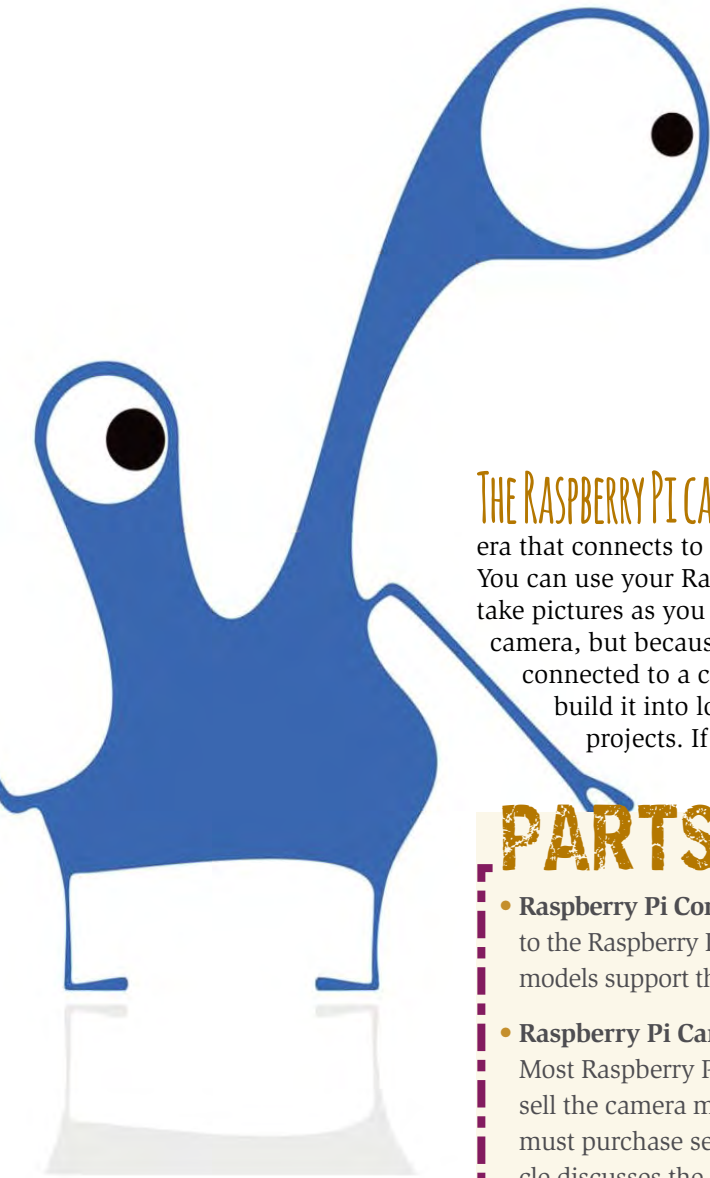


HUGE SAVINGS OFF THE NEWSSTAND PRICE!

SUBSCRIBE NOW: SHOP.LINUXNEWMEDIA.COM

EXPLORING THE RASPBERRY PI CAMERA

HELP YOUR RASPBERRY PI SEE THE WORLD WITH THE PI CAMERA MODULE.
BY DMITRI POPOV



THE RASPBERRY PI CAMERA module is a special camera that connects to your Raspberry Pi. You can use your Raspberry Pi camera to take pictures as you would any other camera, but because your Pi camera is connected to a computer, you can build it into lots of other kinds of projects. If you know a little

programming, you can put your Pi camera to work for projects such as balloon aerial photography, video monitoring, or a scary Halloween pumpkin that lights up automatically when someone enters the room.

This article shows how to install and configure your Raspberry Pi camera and looks at some basic commands for taking pictures with your Pi. You'll even learn to use your Pi for time-lapse photography, and you'll get a quick look at how to embed camera commands into a Python script.

The camera module is no match for a proper camera or even a decent smartphone camera. But it's tiny, cheap, and configurable, so it's a great little gizmo for all sorts of fun and useful projects.

PARTS LIST

- **Raspberry Pi Computer** – According to the Raspberry Pi Foundation, all models support the camera module.
- **Raspberry Pi Camera Module** – Most Raspberry Pi distributors also sell the camera module, which you must purchase separately. This article discusses the standard Rasp Pi Camera Module, *not* the Pi NoIR camera, an infrared-sensitive device for low-light photography.

GETTING THE RASPBERRY PI CAMERA

The camera doesn't come with the Raspberry Pi: you'll need to order it separately. Most of the online stores that sell

Lead Image © Alexandr Aleabiev, 123RF.com

HOW

the Raspberry Pi sell the Raspberry Pi camera module as well. Consult your favorite Raspberry Pi vendor for more information on obtaining the camera. The camera typically sells for less than US\$30.

Connecting the camera module to Raspberry Pi is easy, but there is one important thing you need to keep in mind. Did you notice that the camera comes packed in a silver bag? This bag protects the camera from static electricity that can damage its delicate electronics. So, before you unpack the camera, you need to make sure that no static electricity is lurking in your body. The easiest way to do this is to touch something metal, such as a radiator, in your home. You should always avoid touching the camera module's contacts and electronic components.

Raspberry Pi has a special CSI camera connector that sits between the HDMI and sound ports. To unlock the camera connector, use your fingers to grab the connector's plastic tab and gently lift it up. Take the camera module, make sure the contacts at the end of the ribbon cable face the HDMI port, and insert the

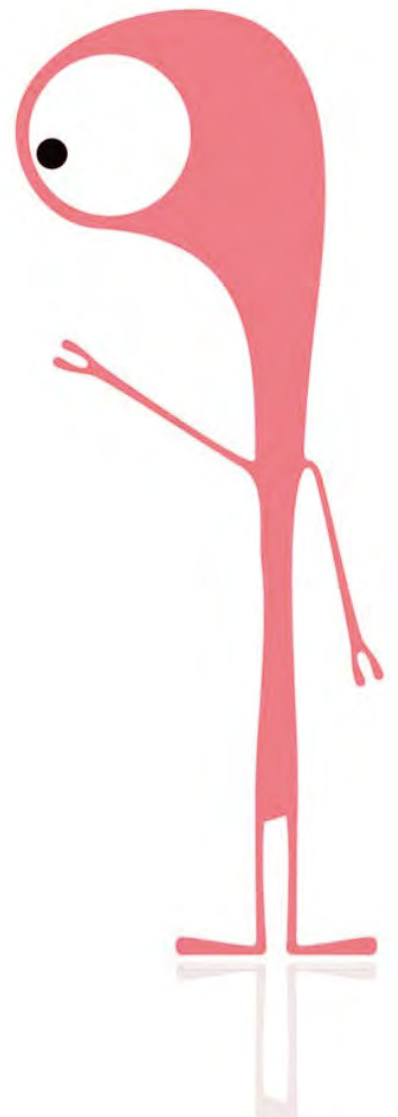
cable into the connector. Hold the cable in place and gently lock the tab by pushing it back down. If you did it correctly, the cable should sit straight and firmly in the connector. If you don't get it right, no problem: Lift the tab and try again. Check the official camera setup video [1] to make sure you do this right.

ENABLING THE CAMERA

The next step is to enable the camera in Raspberry Pi, so the machine can actually detect and control it. On Raspbian Jessie systems, click the *Menu* button, chose *Preferences* | *Raspberry Pi Configuration*, and select the *Interfaces* tab to enable the camera.

On Raspbian Wheezy, `raspi-config` offers an option for configuring the Raspberry Pi camera. Boot your Raspberry Pi to the Raspbian desktop. Click the *Menu* button and choose *Accessories* | *Terminal* to open a terminal window. In the terminal window, type:

```
sudo raspi-config
```



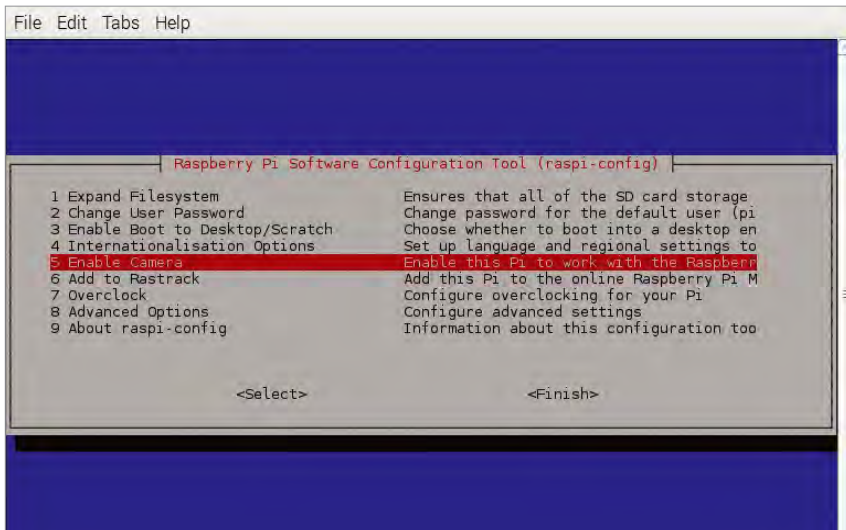


FIGURE 1. The Raspbian configuration utility offers several configuration options, including support for the Raspberry Pi camera.

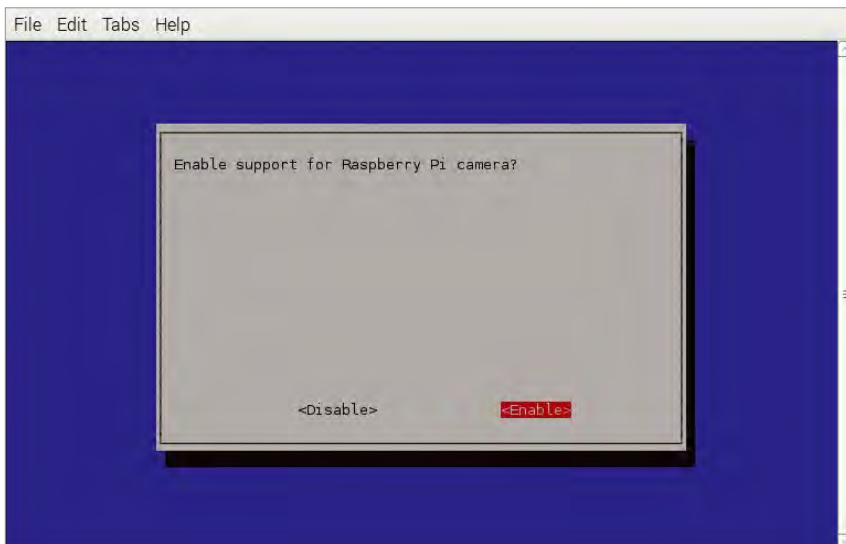


FIGURE 2. The config utility asks if you're sure you want to enable; scroll to Enable and hit the Enter key.

and press *Enter*. In the Raspbian configuration utility (Figure 1), use the arrow keys to scroll down to the *Enable camera* item and hit *Enter*. In the next dialog box (Figure 2), press the right arrow key and scroll to *Enable*, then hit *Enter*, and scroll to *Finish* in the main configuration window. Hit *Enter* and select *Yes* to re-boot your Raspberry Pi.

TAKE A PIC

The camera is connected to the Raspberry Pi, so to get the camera to take a picture, you need to tell the Raspberry Pi to take a picture. The Pi camera is capable of taking video as well as still photos. The two primary commands for operating the Raspberry Pi camera are:

- `raspistill` – for still photos
- `raspivid` – for videos

Once your Rasp Pi camera is connected, and your Raspbian system is configured to talk to the camera (see the preceding section), enter one of these commands in a terminal window to take a picture. The commands come with several options and flags that let you pass additional information to the Raspberry Pi about how to take the picture and where to store the image file.

Open a terminal window and enter the following command:

```
raspistill -o photo.jpg
```

The `-o` flag specifies the name of the resulting photo. The red LED on the camera should turn on, indicating that the camera is taking a picture. When the LED turns off, you should see the `photo.jpg` file in the home directory. Congratulations, the camera works! By the way, if the camera is mounted upside-down, the photo may come out topsy-turvy. Fortunately, this is easy to fix using the `-vf` and `-hf` flags:

```
raspistill -vf -hf -o 001.jpg
```

The camera can take videos, too. To record a video, run the command:

```
raspivid -o movie.h264 -t 5000
```

The `-t` flag specifies the duration of the recording in milliseconds. See the box called “Exploring Raspberry Pi Camera Options” for more on the various options for taking picture with the Raspberry Pi camera.

TIME-LAPSE PHOTOGRAPHY

Have you ever seen a movie that shows how a plant grows from a seed to a blooming flower in just a few minutes? The trick used to create these kinds of videos is called time-lapse photography, meaning that you take photos at regular intervals (e.g., every 5 seconds) over a long period of time, then assemble the photos into a movie. Creating time-lapses requires time and patience, and usually involves some serious gear and software. However, you can use a Raspberry Pi with the Pi camera module and an open source

video tool to create impressive time-lapse movies.

For your first time-lapse project, you may want to try something simple, like creating a short time-lapse clip of a sky with moving clouds. Place your Raspberry Pi in the window and point the Pi camera in the right direction (you may need to take a couple of test shots to find the best position).

Next, you need to do some calculations to find out how many photos to

take and at what intervals. To do this, use the

Timelapse Calculator [2] (Figure 3).

First, choose *Shooting Interval* from the *Calculate* drop-down list. Specify the desired length of the final movie in the *Clip length* field: 30 seconds should be good enough for the first project. In the *Event duration* field, enter the duration of the



EXPLORING RASPBERRY PI CAMERA OPTIONS

The `raspistill` command supports several flags that can help you explore the camera's capabilities and learn a few important basics for taking good photos.

If you think the photos taken by the camera module look dull – washed out with not very bright colors – you can adjust three parameters that can improve the output picture. The `--contrast` flag increases the difference between dark and light areas of the image, and the `--brightness` flag brightens up the whole picture. Both flags accept values between 0 and 100. So, if you want to increase contrast and brighten the output picture, the `raspistill` command might look something like this:

```
raspistill --contrast 5 2
--brightness 51 -o photo.jpg
```

To make colors look more vivid, you can increase the difference between colors using the `--saturation` flag:

```
raspistill --saturation 35 -o photo.jpg
```

Unlike the `--contrast` and `--brightness` flags, `--saturation` accepts values between -100 and 100. So, you can both increase and decrease saturation. For example, if you want to convert the output image into a black-and-white picture, set `saturation` to -100.

Normally, the camera module chooses the optimal settings automatically. However, any camera can sometimes become confused and pick the wrong settings, and the Pi camera is no exception. Fortunately, you can specify important options manually to get better results. Use the `--exposure` flag to manually select the capturing mode. Suppose you want to photograph a dark object on a bright background (e.g., a blackbird sitting in a window). The camera has the *backlight* mode just for this kind of situation:

```
raspistill --exposure backlight 2
-o photo.jpg
```

If you need to take a picture in low light, use the *night* mode, and use the *antishake* mode can be useful for reducing camera shake.

The `--metering` option also can help you deal with difficult lighting situations. The spot metering mode, for example, measures light only in the small area in the center of the frame instead of automatically evaluating the entire scene. This tool can be useful when you need to “focus” metering only on a certain object in the scene:

```
raspistill --metering spot -o photo.jpg
```

If the output image has unnatural colors, the camera probably used the wrong white balance settings. You can try to fix that by choosing the white balance mode manually. On a cloudy day, for example, try using the *cloud* white balance:

```
raspistill --awb cloud -o photo.jpg
```

time-lapse session (how long you want the camera to take photos). One hour is a good starting point. Set the *Frame per second* field to 24.

The higher the frame rate you choose, the “smoother” the final movie will be. Usually 24 and 30 frames per second produce good results. Finally, enter 2.5 (that’s the average size of a photo taken by the Pi camera module) into the *Image size* field to calculate how much space the photos will occupy. Now, note the calculated values. For a

30-second movie at 24 frames per second, you will need to take 720 photos at 5-second intervals. The entire session will take one hour, and the photos will require 1.76GB storage space.

Of course, you don’t have to take 720 photos at exactly 5-second intervals manually. Using the `raspistill` command with the `--timelapse` and `--timeout` flags, you can automate the whole process. The `--timelapse` flag specifies the interval between photos in milliseconds, whereas the `--timeout` flag specifies the duration of the time-lapse in milliseconds. Here is the command that takes photos at 5-second intervals (which equals 5000 milliseconds) for one hour (1 hour = 60 minutes, 1 minute = 60 seconds, 1 second = 1000 milliseconds, so 60x60x1000 is 3600000 milliseconds):

```
raspistill --timeout 3600000 Z
--timelapse 5000 -o photo%04d.jpg
```

If you look closely, you’ll notice that the name of the output image has the `%04d` part in it. This string acts as a 4-digit

TIMELAPSE CALCULATOR

Calculate	Shooting interval					
Clip length	0	h	0	m	30	s
Event duration	1	h		m	0	s
Frames per second	24		fps			
Image size	2.5		MB			
Shooting interval						
Number of photos						

FIGURE 3: Use the Timelapse Calculator to determine timelapse settings.

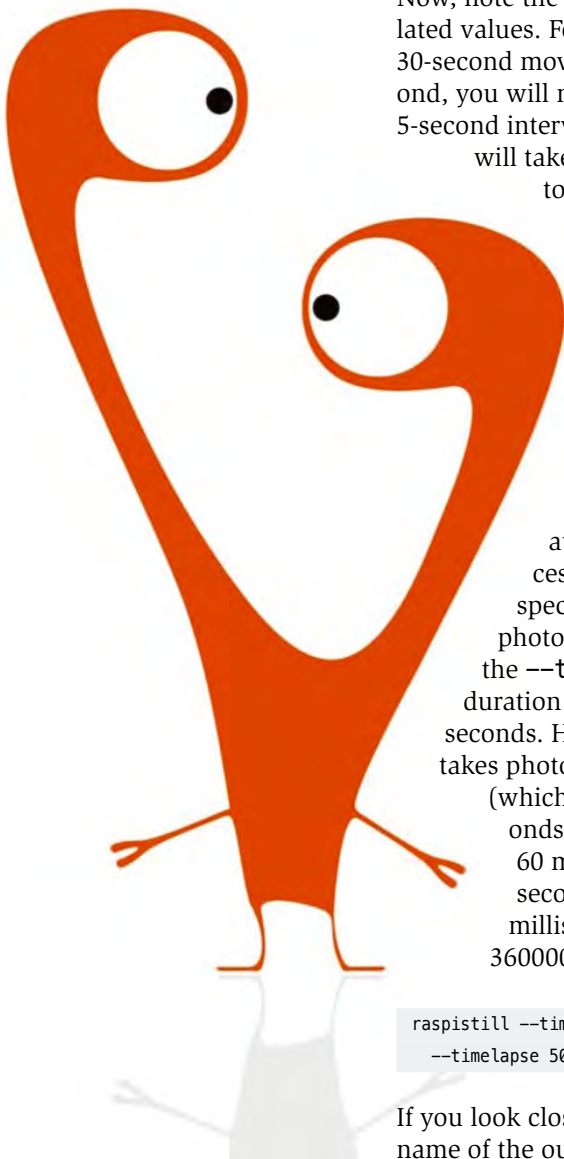
counter with a leading zero. When the command saves the image file, it replaces the string with a number sequence: `photo0001.jpg`, `photo0002.jpg`, `photo0003.jpg`, and so on. To keep all time-lapse photos neatly organized on your Raspberry Pi, create the `timelapse` directory, and switch to it in the terminal:

```
mkdir timelapse
cd timelapse
```

Run then the `rasptill timelapse` command and wait till it finishes taking photos.

The final step is to assemble the photos into a movie. The easiest way to turn the photos into a time-lapse movie is to use the `mencoder` tool. To install `mencoder` on Raspberry Pi, run the `sudo apt-get install mencoder` command (be sure you’re connected to the Internet, so your Pi can download the `mencoder` software).

`Mencoder` is a clever little tool that can do all kinds of tricks with videos: converting them from one format to another, tweaking video quality, and editing movie clips. You can also use `mencoder` to turn a photo set into a movie. Before you launch `mencoder`, however, you need to create a text file containing a list of all photos you wish to assemble in a movie. In the terminal, switch to the `timelapse` directory and run the command `ls *.jpg > photos.txt`. Use



the command below to generate a time-lapse movie:

```
mencoder -nosound -ovc lavc -lavcopts 2
vcodec=mpeg4:aspect=16/9:vbitrate=2
8000000 -vf scale=1920:1080 66 2
-o timelapse.avi -mf type=jpeg:fps=24 66
mf://@photos.txt
```

See the `mencoder` documentation for more on the flags and parameters used in this command. Basically, the command uses the `photos.txt` list to generate a movie clip without sound in MPEG4 format. Once the movie has been generated, you can watch it on your Raspberry Pi using the `omxplayer` media player. This command-line video player is optimized for Raspberry Pi, so it's perfect for watching videos. To view the time-lapse video,

simply run the `omxplayer timelapse.avi` command from the `timelapse` directory.

GOING FURTHER WITH PYTHON

The `raspistill` and `raspivid` commands offer many options for controlling your camera interactively from a terminal window. Once you get used to the camera and get some basic experience with writing programs, you'll have lots of options for how to use the camera in your Raspberry Pi projects. You could place your Rasp Pi camera near a bird's nest to record images of the birds. Hook up your Pi to a motion detector or send your Rasp Pi for a balloon ride.

Raspberry Pi users often use the Python scripting language to automate their Rasp Pi projects. (See the article on

WEBCAM

Suppose you want to use your web browser to take a photo and immediately view it. To do this, you can build a simple web app using the Python Bottle module. The bottle module provides a framework that lets you use your Raspberry Pi as a mini-web server. This exercise assumes your Raspberry Pi is connected to your local network and your local router has a means for assigning IP addresses through DHCP. Start with installing the module by running these commands in the terminal:

```
sudo apt-get update
sudo apt-get install apt-get install python-pip
sudo pip install bottle
```

Create a new text file in a text editor and enter the code in Listing 1 into it. Save the file as `webpicam.py`. Create another text file, enter the code in Listing 2 into it, and save it as `takephoto.py`. Then, run the following commands to make both scripts executable (so the system treats them as programs and not as text files) and then create the `static` directory for saving photos:

```
chmod +x webpicam.py
chmod + takephoto.py
mkdir static
```

Open a terminal window on your Raspberry Pi and enter the `ifconfig` command to determine the IP address of your Raspberry Pi system (Figure 4). Find the IP address for the `eth0` interface, labeled `inet addr` in Figure 4.

Now run the `./webpicam.py` command to launch the app.

Go to another computer on your local network, open a browser, and enter the following in the address bar of the browser window:

```
http://pi_IP_addr:8080
```

Replace `pi_IP_addr` with the IP address of your Raspberry Pi. For example:

```
http://10.0.0.119:8080
```

Press the *Take Photo* button, and you should see the taken photo (Figure 5). Refresh the page if the photo doesn't appear.

Python elsewhere in this issue.) The latest version of Raspbian comes with a Python module that can communicate and control the camera, so if you have a little programming knowledge, you can start writing Python scripts right away.

A brief look at how to integrate the camera into Python will help you get started. To begin, create a simple script that takes a photo. On the Raspberry Pi, use a text editor to create a new text file, and enter the code below into it:

```
#!/usr/bin/python
import picamera
camera = picamera.PiCamera()
camera.capture('photo.jpg')
```

Save the file as `takephoto.py`. After you save the file, open the terminal,

and run the following command in the terminal window:

```
python takephoto.py
```

The preceding command tells Python to execute the script you just saved to the file `takephoto.py`. The script just takes a picture and saves the image to the filename `photo.jpg`. You might be wondering how this script is any different from simply taking a picture with the `raspistill` command, and in this case, it is really the same. However, you get lots of extra powers once you put the command in a Python script. For example, if the image you wanted to capture required lots of additional options for light and color settings, putting the command in a script could reduce the amount of typing and eliminate the chance for a typing error.

When you learn more about Python, you could add additional input, such as triggering the photo with a motion detector or a conditional loop that would take a photo only under predefined conditions.

See the box titled “Webcam” for a look at how to turn your Rasp Pi camera into a web camera that you can operate from another computer on your home network. (BTW: You’ll need to know a little about computer networking to try this experiment!)

GRAPHIC INTERFACE

Sometimes it’s more convenient to operate the camera using a graphical interface instead of running commands in a terminal window. The `RPICameraGUI` [3] application is a graphical web application for the Raspberry Pi camera.

To install `RPICameraGUI`, you need to add a couple of components using the `sudo apt-get install -y python-wx-tools git-core` command. Next, run the `git clone https://github.com/sixbacon/RPICameraGUI.git` command to fetch the application. Switch to the `RPICameraGUI` directory and run the application with the command `./RPICameraGUI.py`.

This simple application (Figure 6) makes it easier to operate the camera, but here is the clever part: You can access and control `RPICameraGUI` from another computer. How can this be use-

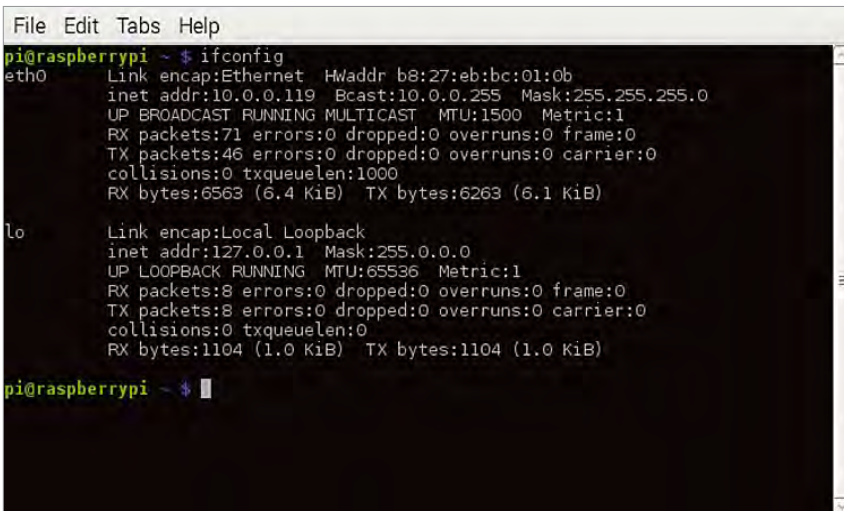


FIGURE 4: Use the `ifconfig` command to determine the IP address for your Rasp Pi.

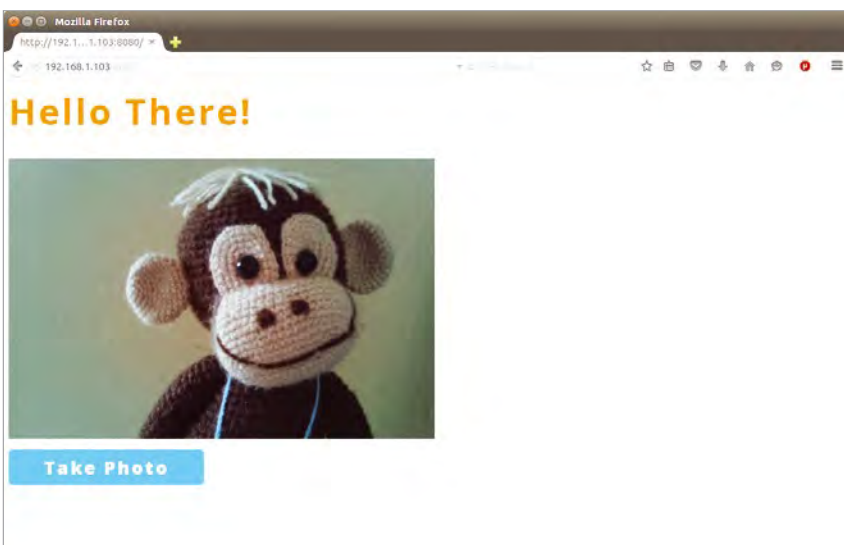


FIGURE 5: You can use a simple web app to control the camera module with a browser.

ful, you might ask? Imagine that you want to take photos of birds in your garden. You can place the Raspberry Pi with the camera module connected to it next to a bird feeder and then use your laptop to control and trigger the camera remotely.

To get started, you need to enable SSH on the Raspberry Pi. Run the `sudo raspi-config` command, switch to the *Advanced Options* section, choose *SSH*, and select *Enable*. After you reboot the Raspberry Pi, you should be able to connect to it from another Linux machine using the command `ssh -X pi@raspberrypi`. (Don't forget to replace `raspberrypi` with the Raspberry Pi's IP address.) Switch to the `RPICameraGUI` directory, launch the application using the `./RPICameraGUI.py` command, and you should see the camera GUI as if it were running on your machine.

FINAL WORD

The camera module is a fantastic addition to your Raspberry Pi. After you've mastered the basics of using and controlling the camera, you can use Python to build whatever great camera-based application you can think of. Good luck, and happy coding! ✨

INFO

- [1] Camera Module Setup: www.raspberrypi.org/help/camera-module-setup

LISTING 1: Simple Web Camera App

```
01 #!/usr/bin/python
02
03 from bottle import post, route, request, static_file, run
04 import os
05
06 @route('/')
07 @route('/', method='POST')
08 def index():
09     if (request.POST.get("takephoto")):
10         os.system("/home/pi/takephoto.py")
11     return '<h1>Hello There!</h1>
12         <p></p><p>
13         <form method="POST" action="/"><input name="takephoto"
14         value="Take Photo" type="submit" /></form></p>'
15
16 @route('/static/:path#.#+', name='static')
17 def static(path):
18     return static_file(path, root='static')
19
20 run(host="0.0.0.0",port=8080, debug=True, reloader=True)
```

LISTING 2: Helper Script for the Web Camera App

```
01 #!/usr/bin/python
02 import picamera
03 camera = picamera.PiCamera()
04 camera.hflip = True
05 camera.vflip = True
06 camera.capture('static/photo.jpg')
```

- [2] Timelapse Calculator: www.photopills.com/calculators/timelapse
- [3] RPICameraGUI: github.com/sixbacon/RPICameraGUI

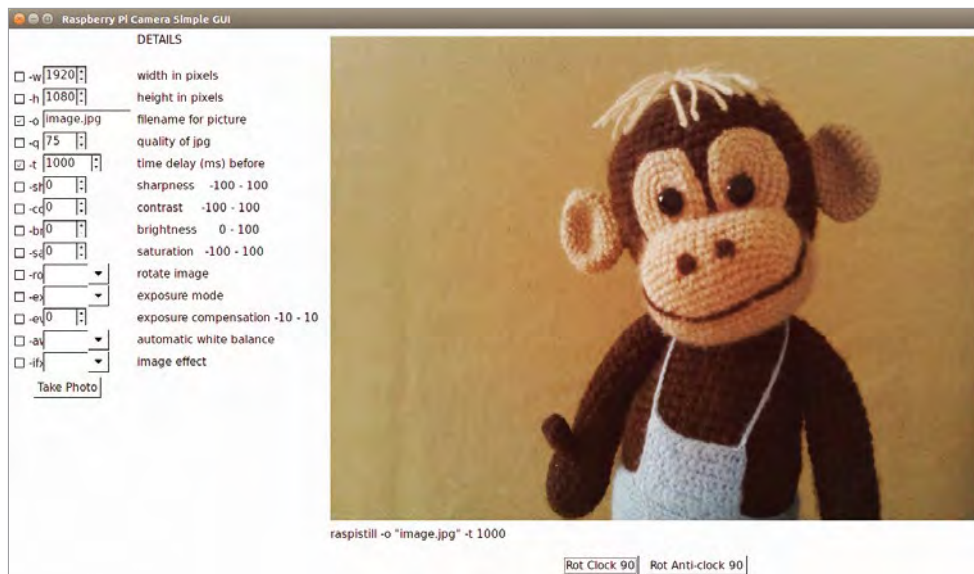


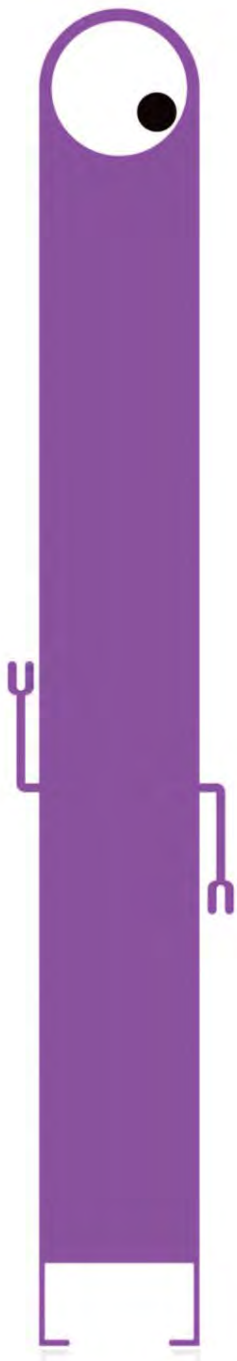
FIGURE 6: Controlling the camera using RPICameraGUI.

Programming Turtle Graphics with Turtle Art

ENTER THE TURTLE

START PROGRAMMING WITH TURTLE GRAPHICS.

BY PAUL C. BROWN



TURTLE GRAPHICS is a graphic notation system that was developed in the 1960s to work with an earlier generation of computers. The Turtle graphics system is easy to visualize, and, perhaps more importantly, it is easy to describe in a way that even a simple computer can implement.

The clarity and simplicity of turtle graphics makes it an ideal starting point for understanding computer programming. This article will help you take your first steps. You'll learn how to direct the computer using commands and how to assemble those commands into simple pro-

grams that draw pictures on the screen or on a printed page.

TURTLE CONCEPTS

Turtle graphics uses a moving cursor called the *turtle*. Imagine this turtle is holding a pen. You can make the turtle move by telling it how many steps to take. You can also tell the turtle to change direction. If the turtle puts the pen down, it draws a line as it walks. If the turtle holds the pen up, it simply moves to a new position and doesn't draw as it walks. You can create a won-

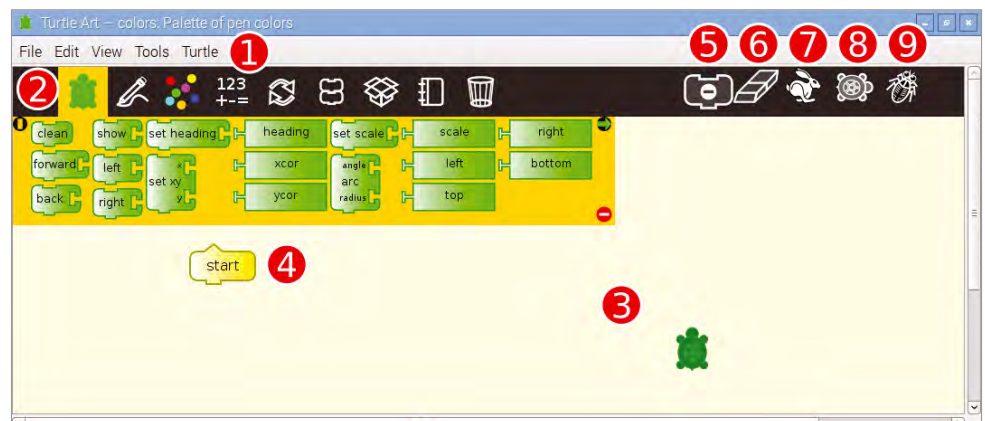


FIGURE 1: Turtle Art on start up: (1) Menu bar, (2) palettes, (3) workspace/drawing area, (4) block, (5) hide palette, (6) clear workspace, (7) run program fast, (8) run program slow, (9) help me look for mistakes in my program (debug).

TURTLE



drous range of pictures using these simple concepts. The turtle can draw almost anything if you give it a series of commands that specify:

- Direction
- Number of steps
- Whether to draw while walking or move to the new position without drawing.

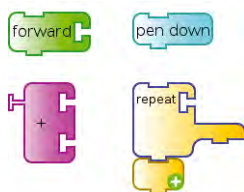
With a few enhancements, like pen color, line width, and some special programming techniques, the little turtle becomes a formidable artist.

Turtle Art is an easy turtle graphics program that runs on the Raspberry Pi. Because Turtle Art is not usually pre-installed on your Raspberry Pi, you need to grab it using the command line. Connect your Pi to the Internet, open a terminal window, and type:

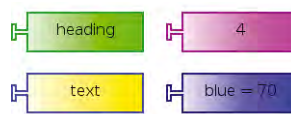
```
sudo apt-get install turtleart
```

You might have to answer *Y* to install the program, but, once it's done and installed, you'll find the *Turtle Art* app in the *Education* section of your menu. Click on the icon and the program will start.

Look at the picture in Figure 1: Across the top of the screen is a menu bar (1). Below that you have a dark bar that contains the *Palettes* (2) on the left. The Palettes have tabs with different blocks and tiles in them. Blocks look like this:



and they do things. Tiles look like the following:



and they contain values you can pass to the blocks. If you want to change the color of the pen to blue, you use a **blue** tile with the **set color** block. I will show plenty of examples of how to do this in a minute.

Click on the tab with the turtle icon (next to the (2) in the picture), and the first tab, *Turtle*, will open. It has blocks you can use to move and do stuff with the turtle. So, to get started, drag a **forward** block out onto the workspace:



It will come with a purple value tile that says **100**. If you click on the block, the turtle will move forward 100 steps. Click inside the purple tile and a black square will appear. Use your keyboard to change the value, to 50, for example. Click on the green **forward** block again, and you'll see how the turtle moves forward half of what it did before.

To clear the screen after each of your experiments, click on the eraser icon (6) in the upper right-hand corner of your screen.

You can make the turtle change direction two ways: you can use **left** or **right** blocks, or you can use **set heading**. These options work in different ways. Drag out



and click on it. You'll see the turtle turn and face right (the 90 refers to "90 degrees" – the angle the turtle has to turn). Click on it again and the turtle will turn and face downwards, because turning right 90 degrees twice makes you turn around and face in the opposite direction.

The `set heading` block works like a compass – look at Figure 2. Here, 0 and 360 indicate north or straight up, 90 is east or to the left, 180 is south or down, and 270 is west or to the right. Bring out a `set heading` block and set its purple value tile to 90:



Then, click the block, and the turtle will turn to look straight right, regardless of which direction it was looking in before.

The second tab, *Pen*, lets you do things with the pen that the turtle is carrying. You can make the turtle pull the pen up, so it does not draw (and then you can make the turtle push the pen down when you want it to start drawing again). You can also change the pen's color and size and make other changes to the way the turtle draws lines.

It may look like you have just a few colors to pick from in the *Colors* tab, but really there are many more. See how

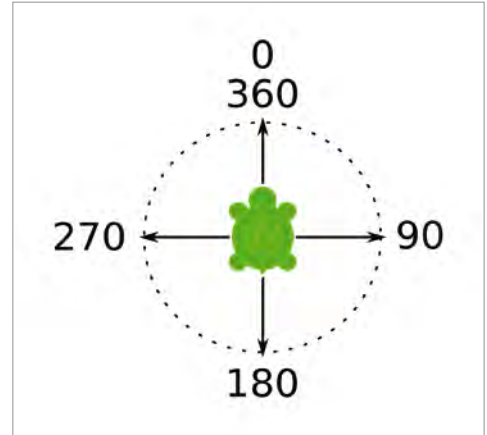


FIGURE 2: The `set heading` block acts like a compass.

each color has a number next to it? Red is 0, orange is 10, yellow is 20, and so on. You can use numbers in between to get slightly different colors. Then, you can also combine the colors into *shades*. Take out the `set shade` block, change the value tile to 20, and start drawing.



The color of the pen will be darker than normal. Change the number in the tile to 70 and draw a line again. The color of the pen will be much lighter. In

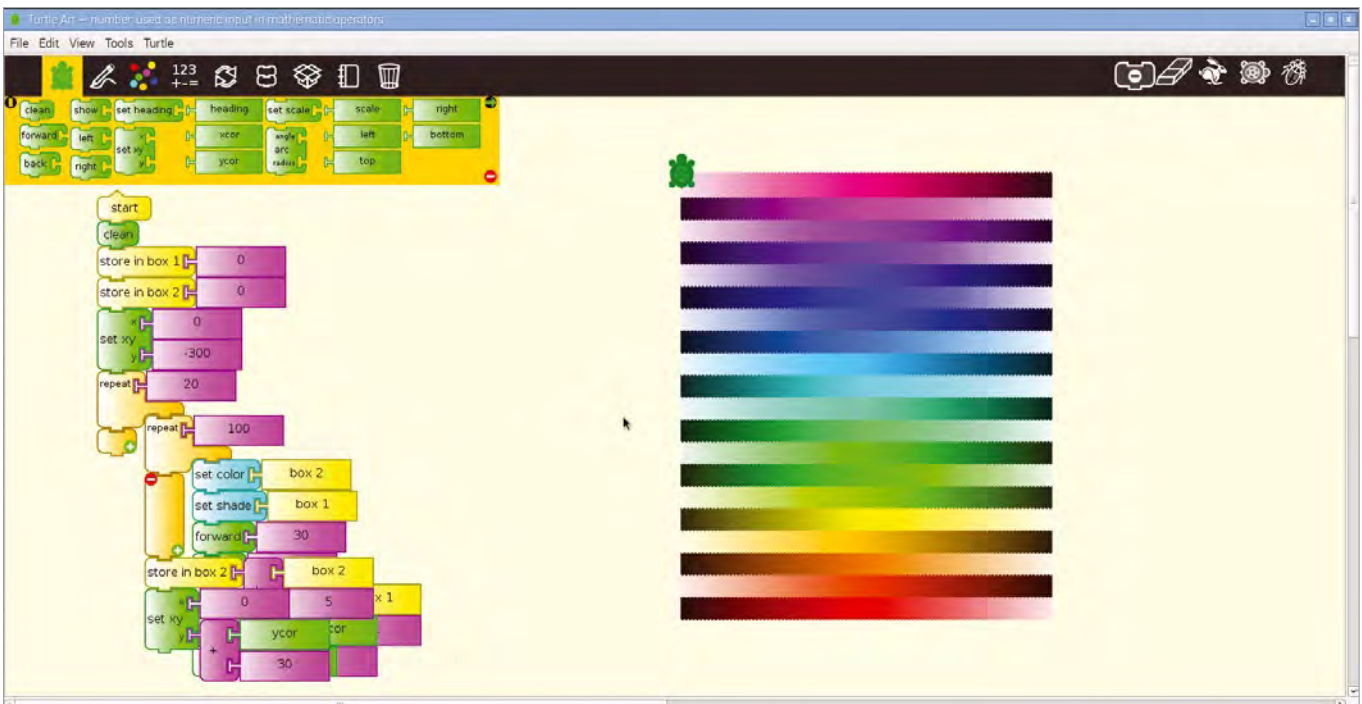


FIGURE 3: You have plenty of colors to choose from.

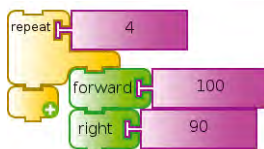
Figure 3, you can see some of the colors you can draw with.

The *Numbers* tab is all about math and numbers. You can use Turtle Art as a calculator! Drag out a division (/) block and do this:



Then, click on the / block. An orange bar will pop up at the bottom of the window with the result: 1.625.

The *Flow* tab is important, because it contains blocks that allow you to create loops. A loop lets you to do something again and again. You can use a loop to draw a square. Squares have four sides and four corners, right? You want to make you turtle move forward, say, 100 steps, turn right 90 degrees (to make the corner), and do that four times. In turtle art, you do that like this:



PRETTY PENTAGONS

Now that you've drawn a square, why not try a pentagon? First, though, I'm going to let you in on a secret.

Look at the picture in Figure 2 again. See at the top where it says 0 and 360? If you turn a full circle and get back to facing the way you were originally, you have to turn 360 degrees. Now, look at the degrees you have to use to make a turn in a square: 90. 90 is $360 / 4$. And 4 is the number of sides or angles you have in a square.

The interesting thing is this rule applies to all polygons, so, to work out the angle your turtle has to turn in a pentagon, you divide 360 by 5, in a hexagon you divide 360 by 6, and so on. That means your code for drawing a pentagon in Turtle Art could look like what you see in Listing 1.

Notice you should start using the **start** block at the beginning of your code. This block tells Turtle Art where your program starts. If you press the picture of the hare (7), the code that begins with **start** is what will get run (if you have two pieces of code that begin with

start, pressing the hare will execute the piece of code you wrote first).

That's one way to draw a pentagon, but you can also draw one using the blocks in the *Blocks* tab (Table 1).

ACTION PACKED

You can make the turtle draw any regular polygon you want, and you only ever have to change one tile. A *box* in Turtle Art is known as a *variable* in other programming languages. In the *Blocks* tab, you have all the bits and pieces to build a box.

Drag out the **store in** block and change it to look like this:



This means you can now use the value inside a box you just called **sides** whenever you need to tell Turtle Art how many sides your polygon has.

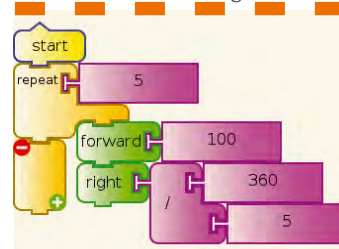
In your program, your code would look like what you can see in Listing 2. It may look like more work to use a box, but think about it: Before, if you wanted to draw a different polygon, say a hexagon, you had to change your program in two places. Now you only have to change the program in one place. That's half the work!

And, it gets better. Suppose you want to draw all the polygons from a triangle up to an octagon. Before, you had to write six programs, but with the help of boxes, an **action**, and a **repeat** loop, you only need one.

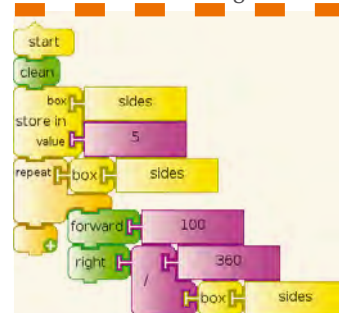
An **action** in Turtle Art is what other languages call a *function* or *module*. You give it a name (you're going to call your first action *polygon*), and it contains some blocks you're going to need over and over. When you need to draw a polygon in your program, instead of having to write all the code out each time, you can insert one simple **action** block with the name of your action (**polygon**) and, then your code gets run!

To make this work, the first step is to move the code that draws the polygon into your action. Drag from the *Blocks* tab a **start** action block and move your blocks from the program shown in Listing 2 to under the **start** action block. Your code should look like what is shown in Listing 3.

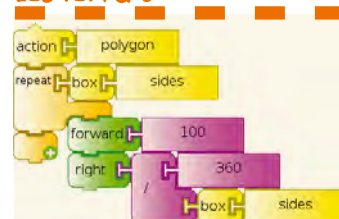
LISTING 1: A simple Pentagon



LISTING 2: A Smarter Pentagon



LISTING 3: Action Polygon

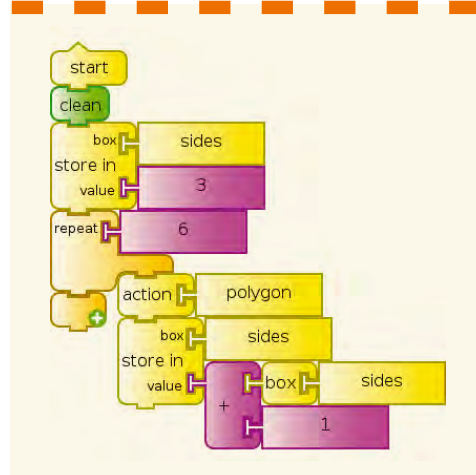


The next step is to set up the main body of the code. My version is in Listing 4.

You start out by cleaning up the work area. Then, you set the number of sides for the first polygon. The first polygon you want is a triangle, so you would put 3 into *sides*.

Then you have a repeat loop that will run six times, because you want to draw a triangle, a square, a pentagon, a hexagon, a heptagon, and an octagon...

LISTING 4: Many Polygons Program



That's six polygons. Each time the program runs it does two things: First, it calls the **polygon** action you made before. Second, it adds 1 to *sides*.

The first time the program runs through the loop, *sides* contains 3 and so **polygon** draws a triangle. Then, you add 1 to *sides*. The second time the program runs through the loop, *sides* contains 4, and so **polygon** draws a square. Then, you add 1 to *sides* and, well, you get the rest. The final picture looks like what you can see in Figure 4.

STARS AND SPIRALS

Now you're ready to draw a star in a pentagon. The code in Listing 5 makes use of the famous irrational number ϕ (*phi*). See the box titled "More on Phi" for additional information on this famous number, which has many uses in mathematics and design. All the code you will need for teaching the turtle to draw a star in a pentagon is shown in Listing 5.

See how the main program consists of amazingly simple actions! The first thing you do is clean up the workspace and set the turtle looking to the left. Then,

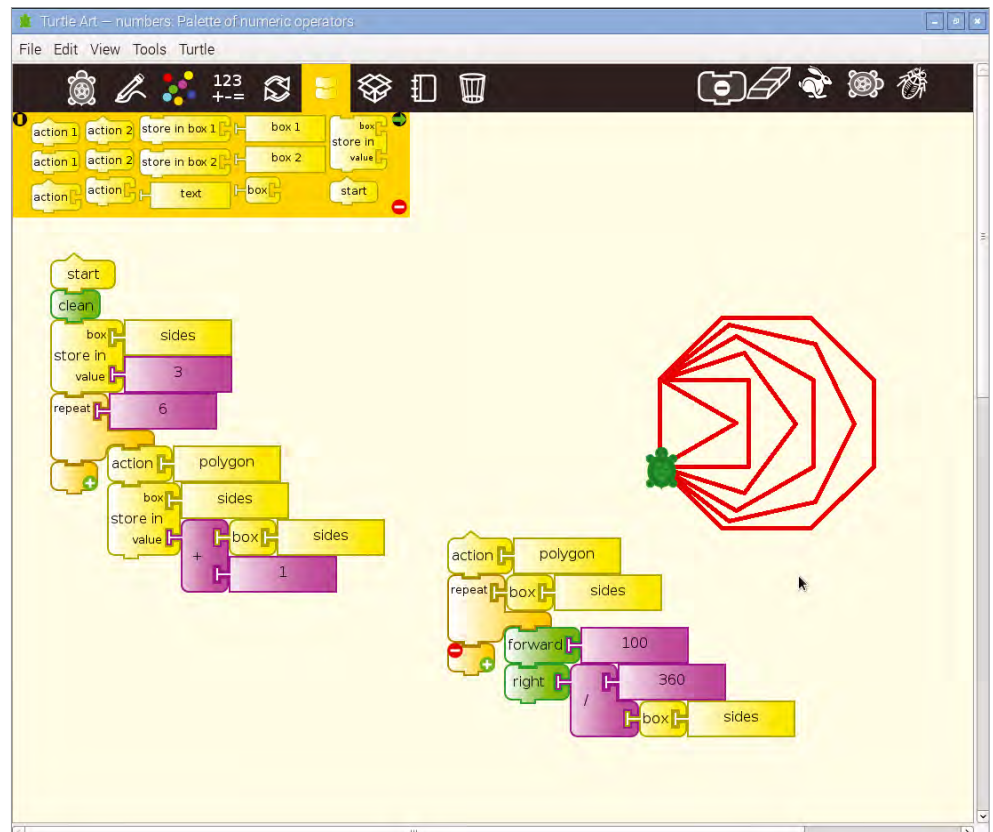
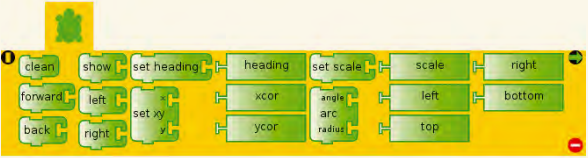
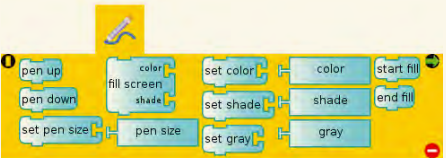
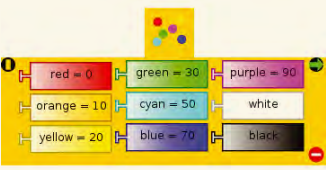
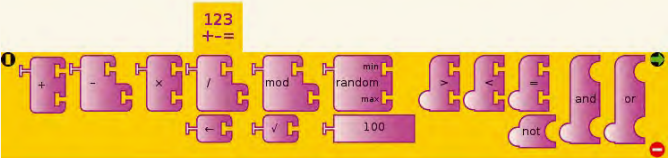

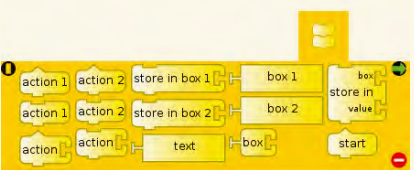
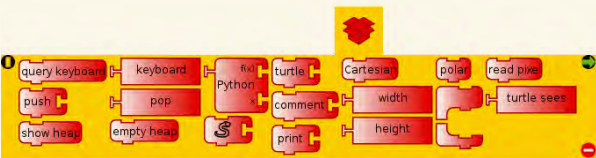
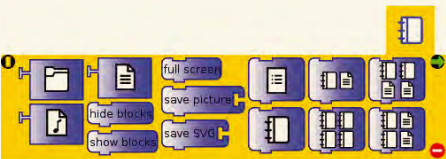
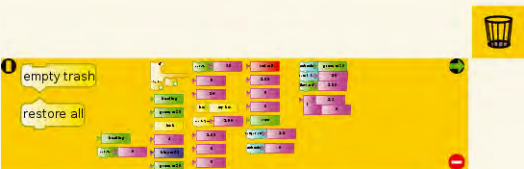


FIGURE 4: Six polygons, from a triangle all the way up to an octagon.

TABLE 1: All the Tabs in the Palette

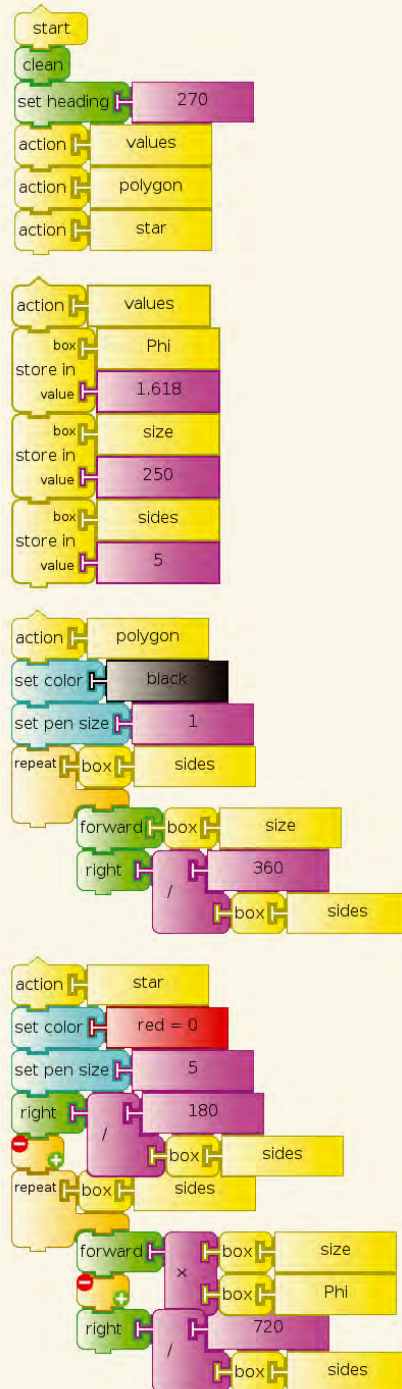
Name	Picture	Contains
Turtle		Commands that control the turtle
Pen		Commands that control the pen
Colors		Pen colors
Numbers		Numeric operations
Flow		Loops and conditions
Blocks		Variable and action blocks
Extra		Extra options
Portfolio		Presentation templates
Trashcan		Erased blocks and code

you set up some values (see the **values** action a bit below), such as Phi, the size of your pentagon and the number of sides. It's a pentagon, so the number of sides is going to be five. Then, you draw a polygon.

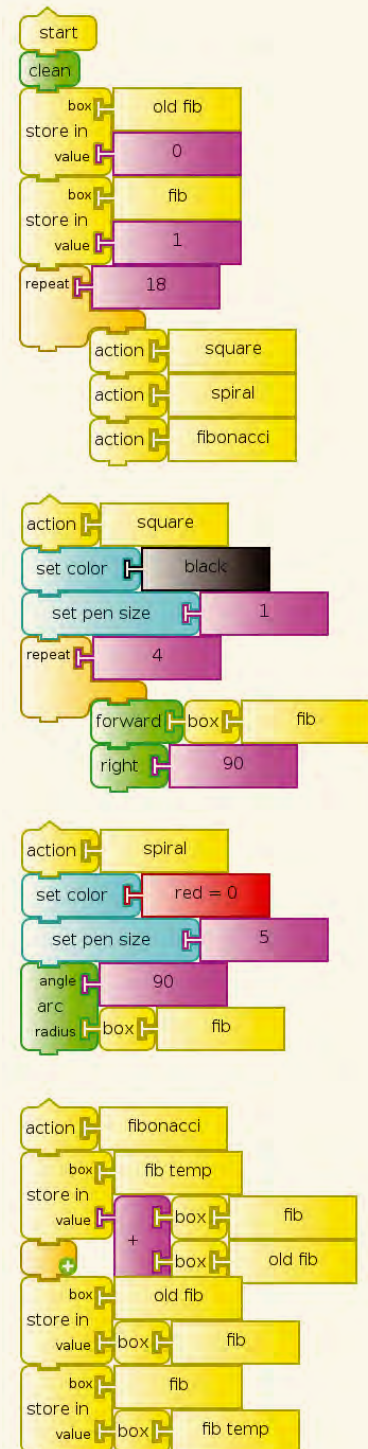
Finally, you draw a star inside the pentagon. The **star** action is a tiny bit more complicated than drawing a polygon, but not much. After changing the color and size of the pen, to place the turtle

facing the right way, you have to turn the turtle to the right by 180 degrees divided by the number of sides (i.e., $[360/2]/[\text{no. of sides}]$). Remember how the angles of a polygon measure 360 degrees divided by the number of sides? Well, the angles of a star measure 720 degrees divided by the number of sides (i.e., $[360 \times 2]/[\text{no. of sides}]$).

LISTING 5: A Phitagon



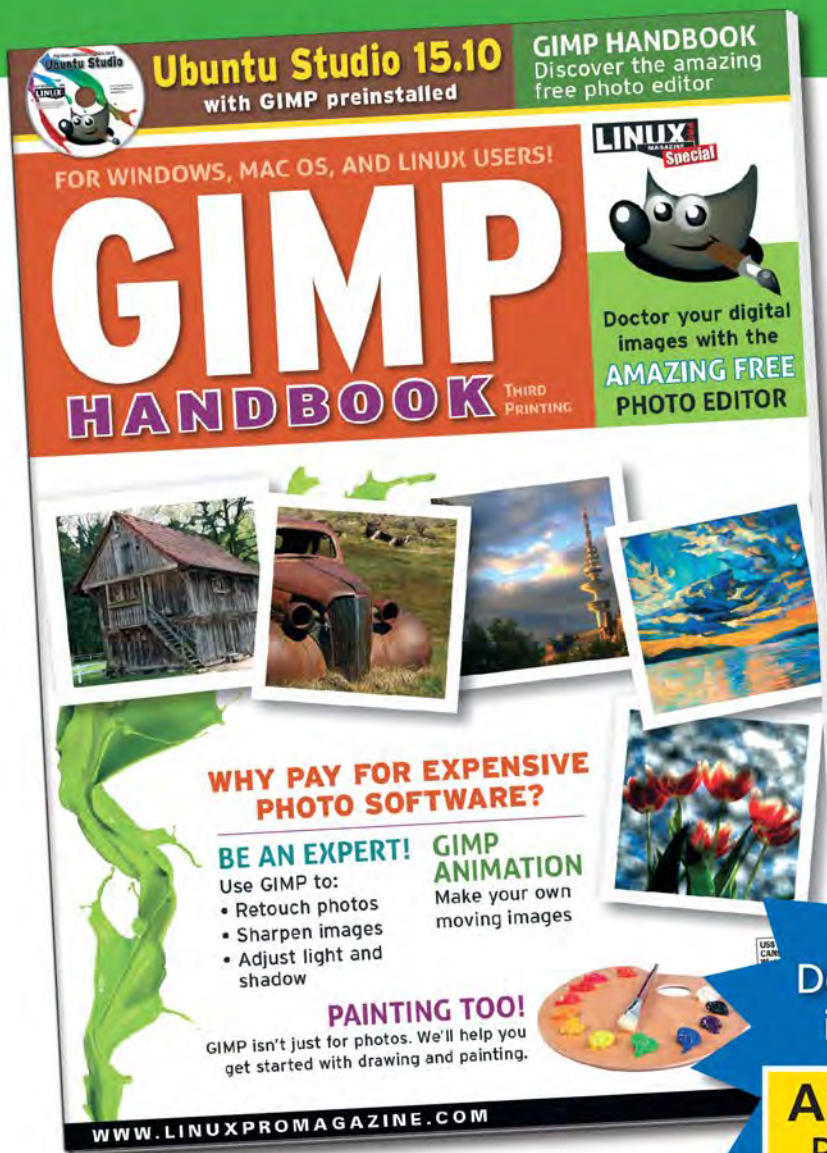
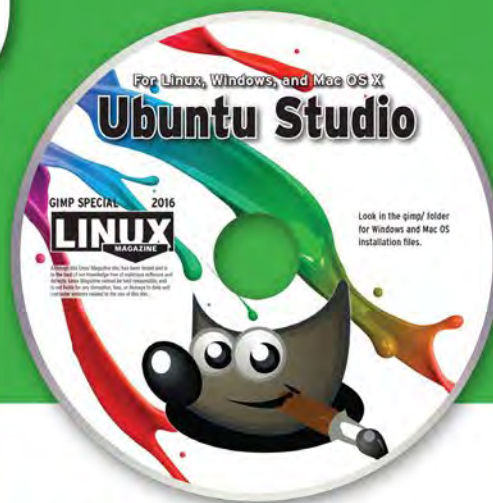
LISTING 6: The Fibonacci Spiral



Shop the Shop

shop.linuxnewmedia.com

GIMP HANDBOOK



**SURE YOU
KNOW LINUX...**
but do you know **GIMP?**

- Fix your digital photos
- Create animations
- Build posters, signs, and logos

Order now and become an expert in one of the most important and practical open source tools!

GIMP
Doctor your digital images with the
AMAZING FREE PHOTO EDITOR!

Order online:
shop.linuxnewmedia.com/specials



FOR WINDOWS, MAC OS, AND LINUX USERS!

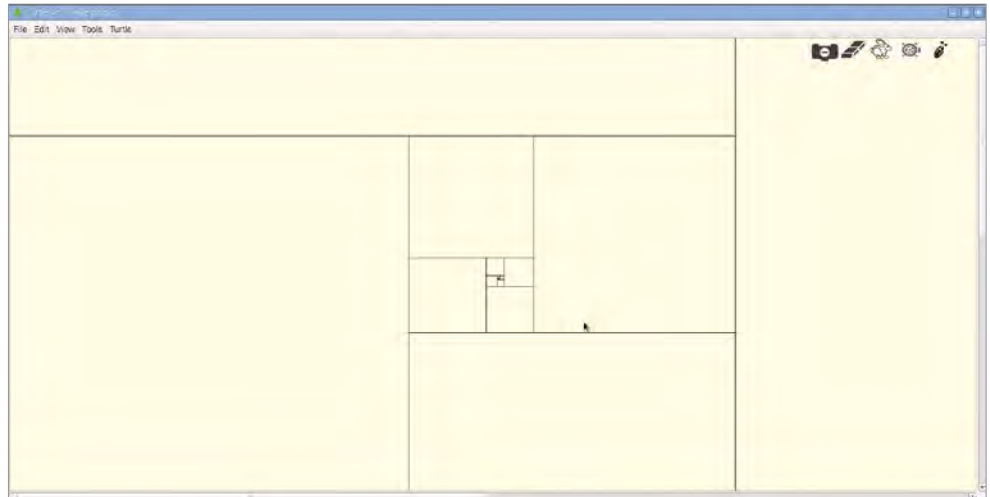


FIGURE 5: Drawing squares using the numbers in a Fibonacci series.

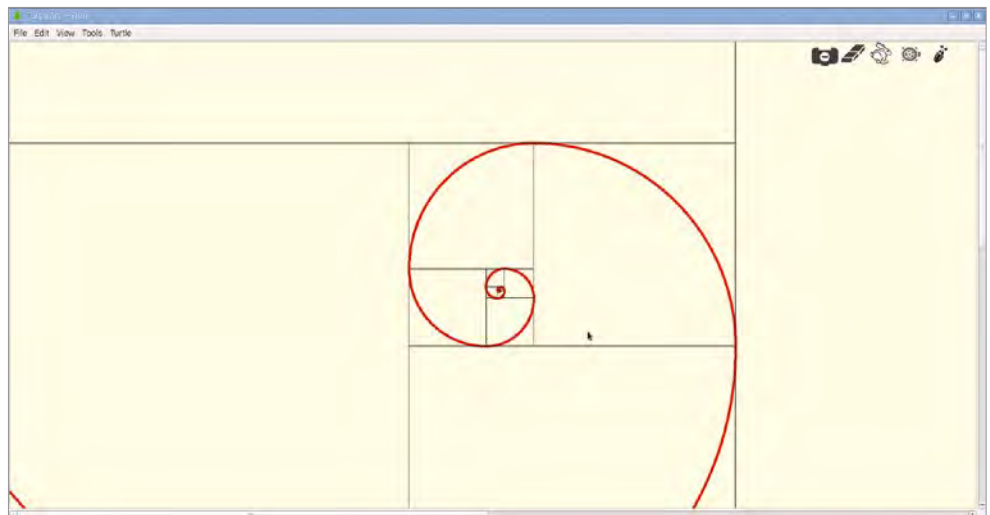


FIGURE 6: A Fibonacci spiral.

Also remember that the length of a star's sides are equal to the length of the side of the pentagon multiplied by ϕ , so you multiply by ϕ in the `repeat` loop.

Run the program with the `start` block, and the turtle will draw you a star enclosed in a pentagon.

There are two more pictures I would like to share that are associated with ϕ and Fibonacci numbers. For example, take a look at what happens if you draw some squares with sides the same as the numbers in a Fibonacci series.

If you draw the squares – not side by side – but around each other, as shown in Figure 5, you can see a kind of spiral emerge. Let me add in a quarter of a circle joining the first corner the turtle draws in each square with its third corner. See Figure 6. Listing 6 shows the turtle code for the spiral.

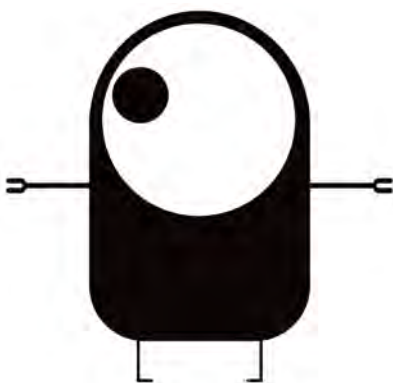
CONCLUSION

You can use Turtle Art to draw square houses, with square windows, a triangle roof, and a circle sun. Use your imagination to create your own projects using the turtle graphics. This adventure into pentagons, stars, and visual representations of Phi offers a lesson in using the turtle as a powerful programming tool.

If you want to know more about ϕ , Fibonacci, or how to draw more things using Turtle Art, write to me! My email address is p_brown@gmx.com. I'd also love to see your pictures made with Turtle Art! *

INFO

 Golden ratio: https://en.wikipedia.org/wiki/Golden_ratio



MORE ON PHI

Some people like π ; others like e . Me? I'm a 1.618033988749894848204586834 guy. For me ϕ (or *phi* – rhymes with “fly”) is the most beautiful number in the world.

Phi starts with 1.618033988749894848204586834..., and it goes on and on and on to infinity, never repeating itself. Phi is an irrational number, like the famous π (pi), which lends its name to the nano-computer Raspberry Pi. Irrational numbers contain an infinite number of digits in all possible combinations. You could, for example, find your age, your birthday, and your phone number all bunched together somewhere in the infinite realm of digits if you looked hard and long enough.

Phi isn't just beautiful because it is infinite. Why is it awesome? Because of the Fibonacci sequence, pinecones, and pentagons.

Allow me to explain: This...

1 1 2 3 5 8 13 21 34 55 89 ...

is the start of the Fibonacci Sequence. In the Fibonacci Sequence, you start with 1 and 1, and then you calculate the next number in the list by adding up the two that come before it. So, you take the first two numbers, 1 and 1, add them together and get the third number, 2. Then, you add the second and third numbers, 1 and 2, and you get the fourth number, 3. And so on.

What would come after 89? Add 55 (the second to last number) and 89 (the last number) and you get $55 + 89 = 144$.

Now, take any number in the Fibonacci sequence and divide it by the number before it (you can use a calculator):

$1/1 = 1$
 $2/1 = 2$
 $3/2 = 1.5$
 $5/3 = 1.666666...$
 $8/5 = 1.6$
 $13/8 = 1.625$
 $21/13 = 1.615384...$
 $34/21 = 1.619047...$
 $55/34 = 1.617647...$
 $89/55 = 1.618181...$
 $144/89 = 1.617977$

As you move further and further along the sequence, the result of the divisions get closer and closer to ϕ ! Let's look at pine cones now. Find a cone like the one shown in Figure 7. If it is closed, all the better. If you count the scales around each ring, starting at the tip (counting the tip as 1) and moving down, you'll usu-

ally find that each ring contains a number of scales that is in the Fibonacci sequence: 3, 5, 8, ... Not always, mind: sometimes it will be the Fibonacci sequence + 1 (2, 4, 6, 9, ...). If you look at the seeds in the center of the sunflower, you'll see the same thing.

You'll also see how the seeds form a spiral radiating from the center of the sunflower. A pine cone also has spirals of scales radiating from the top and bottom.

Suppose I ask you to draw a pentagon like the one shown Figure 8. Each side has to measure 100 steps. (I want a really big pentagon!) Now I give you a big bucket of red paint and tell you to join all the corners on the inside to create a star, again, just like in the picture. Do you know how long each of the red lines has to measure? Well, if each black side measures 100 steps, each red line measures 161.8033988749894848204586834 steps.... a red line measures whatever a side measures multiplied by ϕ (in this case, 100 steps $\times \phi$).

See that blue line? It measures exactly whatever a side measures divided by ϕ (100 steps/ ϕ). All of this comes in really handy when drawing pentagons and stars!



FIGURE 7: Pine cones knew about Fibonacci numbers before you did!

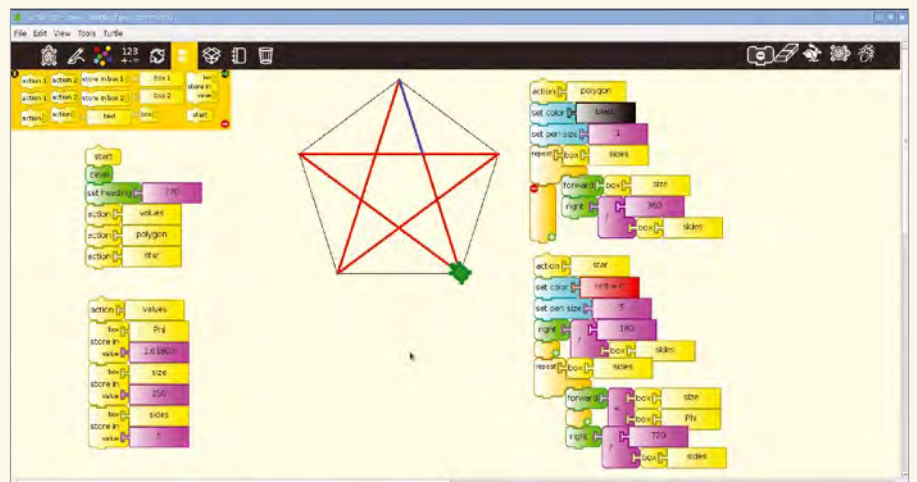


FIGURE 8: Drawing pentagons and stars requires knowing about ϕ .

GETTING STARTED WITH
SCRATCH PROGRAMMING


SCRATCH GOODE

SCRATCH MAKES LEARNING TO PROGRAM FUN AND EASY. THIS TUTORIAL STARTS WITH SOME BASICS AND MOVES INTO MORE ADVANCED USES WITH SIMPLE EXAMPLES. YOU'LL LEARN HOW TO DRAW, ANIMATE, AND CREATE A SHARK ATTACK GAME.
BY MICHAEL BADGER

SCRATCH is a free programming language that's designed to teach you how to program by creating stories, animations, multimedia projects, and games. The version of Scratch that ships with the Raspberry Pi will allow you to do all of these things and more.

One of the ways Scratch helps you quickly see your ideas on the page is through a collection of color-coded blocks that you stack together to create the instructions that tells your game, story, or animation exactly what to do. If the blocks snap together, they will run, allowing you to rapidly and successfully create programs. The visual drag-and-drop nature of Scratch means anyone can create with Scratch. First, I'll take a quick look at the Scratch interface.

SETTING UP SCRATCH

Scratch is a default application on the Raspberry Pi. You can open the program by going to the *Menu | Programming | Scratch*. A window resembling Figure 1 will open.

The interface is horizontally divided into three primary sections. The left third of the interface contains the blocks, organized by category. The blocks each contain a command, such as move (10) steps. I like to think of the categories of blocks (motion, looks, sound, pen, control, sensing, operators, and variables) as palettes because they are comparable to a painter's palette of colors. You can mix the colorful blocks together to form your art.

Lead Image © Alexandr Aleabiev, 123RF.com

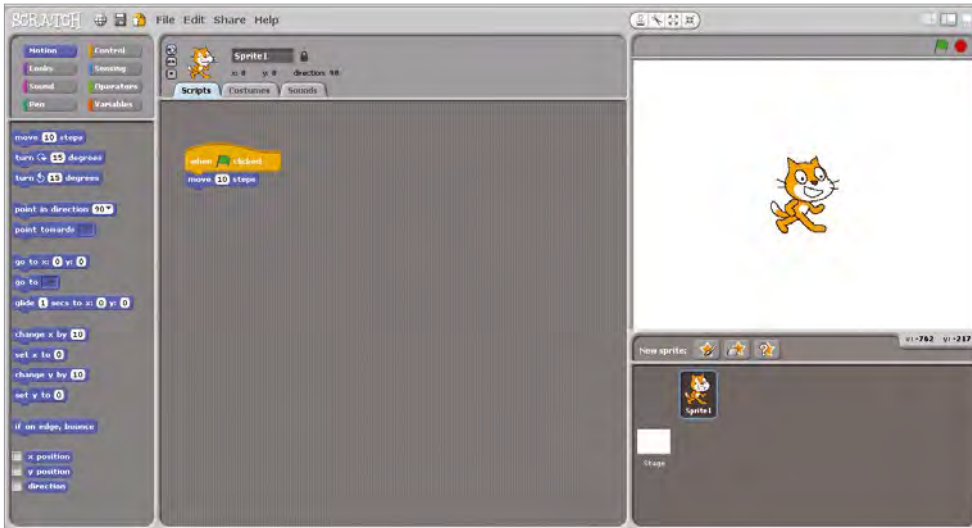


FIGURE 1: The Scratch interface.

The middle third of the interface contains tabs with specific information related to the project. The most common area you will work with is the Scripts area, which is a tab. The Scripts area will be the primary workspace where you stack the blocks to control your characters, called sprites in Scratch. You can see in Figure 1 that there are additional tabs for costumes and sounds, in addition to some sprite-specific information (name, rotation, positioning). This area describes everything you need to know about the selected sprite.

In the right third of the interface is the stage and the list of sprites. In Figure 1, you see two sprites (the cat and a block figure) on the main stage. The stage is where each of the characters play out their scripts. Directly below the stage is the cast of sprites in each project, and as you see, projects can contain much more than just sprites.

As I describe a few simple exercises, I will fill in more details. This short tour provides the necessary introduction for you to follow along with the sample scripts and should make it easy for you to imagine yourself learning to program with Scratch.

BASICS — DRAWING

Every time you create a new project, the Scratch Cat will be included, but you're under no obligation to keep the cat in your project. There are several ways to get characters into your project. You can draw, import, or edit existing sprites.

The first feature I want to explore with you is the drawing capabilities that are built-into Scratch. Drawing is accomplished in the Paint Editor, and it allows you to create characters, scenes, or art without worrying about programming. After you draw a character, I will show you how to animate it.

To open the Paint Editor, click the *Paint new sprite* icon found between the stage and sprite list. It's the icon that has a star and paint brush; as you move your mouse over the icon, a tool tip will display with the name of the tool.

Figure 2 shows the Paint Editor with my hand-drawn sprite. If you've used other graphics editing programs, such as Gimp or Photoshop, these tools will likely be familiar. Scratch provides a very small subset of the tools available

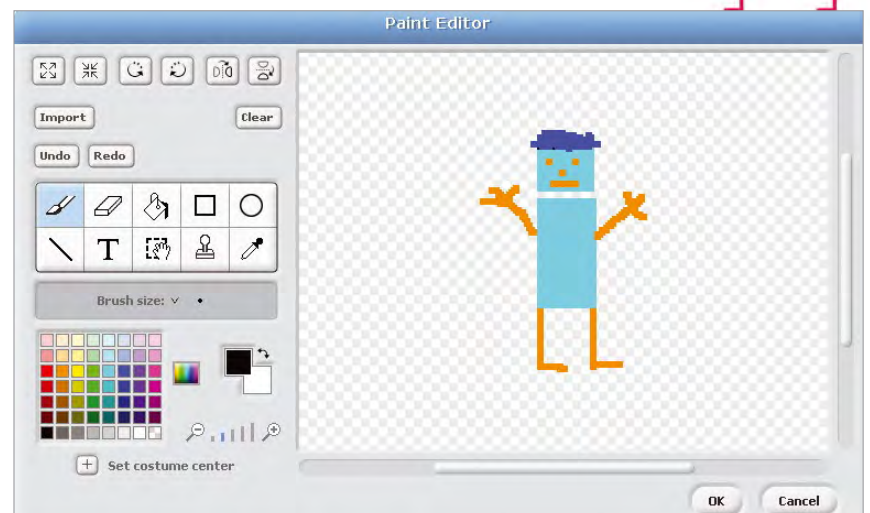
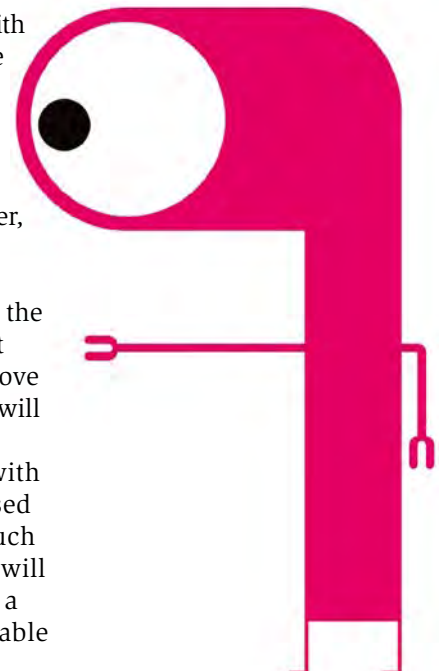


FIGURE 2: Scratch provides its own Paint Editor to allow you to create your own sprites and backgrounds.

in other applications, but it will be sufficient.

I drew my sprite using the rectangle tool and the paintbrush. Both of these tools let you select options that change how the paintbrush or rectangle draw. For example, both provide an option to select a color. For the paintbrush, you can select a brush size to draw in bigger or smaller strokes. For the rectangle tool, you can select whether or not you draw a solid rectangle or a transparent rectangle with a border. Several of the other tools work the same way.

If you make a mistake, there is an undo option, and if you want to start over, you can use *clear*. Take a moment and experiment with the drawing tools to create your own sprite. Browse through the other options in the Paint Editor.

After you click *OK* to save your sprite and exit the Paint Editor, you may find that your character is too big for the stage. Directly above the stage is a *shrink sprite* button. When you click *shrink sprite*, the mouse cursor will display as four inward-pointing arrows. Position the cursor over the sprite and click. Each click will reduce the size of the sprite.

SIMPLE MOVEMENT

Now, you can make your character move across the screen, but before you look at a script, go to the Motion palette and drag the *move (10) steps* block into the scripts area for your new sprite. Then, click on the block several times. Did you see your character move?

The sprite is moving a relatively small number of “steps” each time you click the block, which is why you may need to click the block several times to see any movement. The steps represented in the *move (10) steps* block are pixels, so with each move command, the sprite moves 10 pixels. In Scratch, the stage is 480 pixels wide and 360 pixels tall.

Now, reverse the movement of your sprite by changing the 10 to a negative 10 (-10). Click the block several times and observe that the sprite moves to the left. Take a moment to see what happens when you make the number of steps larger or smaller.

Just like in real life, if you took 10 steps forward and you wanted to return to where you came from, you would

take 10 steps backward. In Scratch, the 10 steps backward can be represented with a negative number.

The other subtle feature used in this exercise will be of great benefit to you as you create bigger and more complex projects. You were able to make the sprite follow the command on the block just by clicking on the block, and the changes were immediately reflected in the sprite. This allows you to test an individual block or an entire script by clicking on the stack of blocks to test your programming without running the entire project.

Now that you can move your sprite forward and backward, I’ll show how to create your first animation using costumes. With your sprite selected, click the Costumes tab to display a thumbnail view of your sprite’s costumes. You should have one thumbnail for the sprite you created.

ADDING COSTUMES

Costumes are different appearances for a sprite. It’s similar to when you put on a costume (e.g., clothes for school, uniform, or pajamas), and you change how you look. To create a second costume, click the *copy* button next to first costume to duplicate it. Now click the *edit* button for *costume 2* to open the Paint Editor, and you can change some appearances on the second costume.

You may choose to make any changes you want, such as recoloring parts of the costume or changing the positions of the hands or feet. The animation I describe will involve walking, so if you want to try to show the second costume taking a step, go for it. In my example, I clicked the *rotate clockwise* button in the Paint Editor to give my second costume a tilt. The goal is to have a second costume that’s different enough from the first costume so you can see the animation. Realism is not required.

After you create your second costume, build the script shown in Figure 3. You’ll find the *when green flag clicked*, *wait () secs*, and *forever* blocks in the Control palette. The next costume blocks are found in Looks.

When you click on the green flag, observe the animation. As the sprite moves across the stage, you should see your animation play out. You’ll immediately notice a problem, however. When the



FIGURE 3: A simple script to move the sprite around the stage.

sprite reaches the edge of the stage, it turns around and goes in the opposite direction, courtesy of the **if on edge, bounce** block; however, when the sprite bounces off the right side of the stage, it flips upside down.

To fix the rotation of the sprite, look in the sprite information panel above the Scripts tab for three small buttons that affect rotation. Click the *Only face left-right* button in the middle, and the sprite will stop walking upside down.

You have a lot of control over how this animation appears by changing the values in the **move** and **wait** blocks. In my script, I tell the sprite to wait .05 seconds after switching to the next costume. This slows down the animation because the next block in the stack does not run until the current block completes its command. If you take the wait block out of the script, you'll notice that the animation becomes very fast.

The **wait () secs** block shows us something important about the order in which blocks are run in Scratch. When you write a script in Scratch, the commands run in the order you stack the blocks. However, it is possible to have multiple scripts running at the same time for a sprite.

By now, you've noticed that this animation doesn't stop. That's because the animation is inside the **forever** block. A sequence of blocks that continually runs over and over again is called a loop.

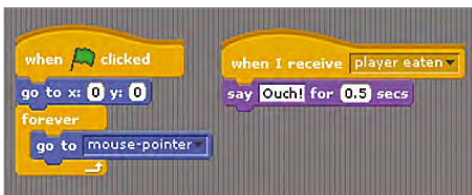


FIGURE 4. The player scripts in a game of Shark Attack.

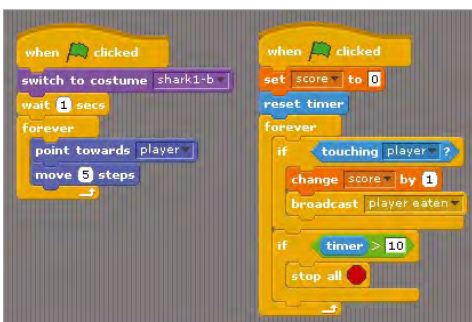


FIGURE 5. The shark scripts in a game of Shark Attack.

Before I move on, the **when green flag clicked** block warrants some discussion. Using this block is a common way to start your program in Scratch, and my simple script is demonstrating two fundamental uses. As you've seen, the animation is started when the user clicks the green flag in Scratch.

Using the green flag click is a great way to set initial values at the start of your project. In Figure 3, I'm ensuring that *costume1* is selected, because you may notice if you start and stop (using the stop sign) the script, the program may end on *costume2*. This way, I ensure I start with the first costume.

You may want to specifically set many things at the start of a game, story, or animation, including resetting scores to zero, positioning a sprite on the stage, showing/hiding sprites, clearing graphic effects, or clearing the pen drawings.

ADVANCED

Now, I'll show you the scripts for a two-sprite "Shark Attack" game. The player has 10 seconds to avoid being "eaten" by the shark. Each time the shark touches the player, the score increases, meaning the player's objective is to keep the sharks' score low. How low can you score? Can you get zero?

Figure 4 shows the scripts for the player, and Figure 5 shows the script for the shark. I'll explain the important parts of these scripts below.

Figure 6 shows the game play. You can draw your sprites or use sprites from the Scratch library. If you use the *Choose new sprite from the file* option for a new sprite, then you can browse the list of available sprites in Scratch's library. In



FIGURE 6. A scene showing the shark attacking the player.

the Animals category, you will find several sharks, for example.

Figure 4 shows the script to control the player sprite's movement. At the start of the game, the script positions the player in the middle of the screen. That's the `go to x: (0) y: (0)`. Each location on the stage can be identified by a set of x and y coordinates. The x values range from -240 to 240, and the y values range from -180 to 180. For now, I'm only concerned with identifying the middle of the stage (x = 0, y = 0).

Next I use the `go to (mouse-pointer)` block to send the player sprite to the location of the mouse cursor. If you build and play this script, you will see that the player will follow the mouse, meaning that moving the mouse moves the player.

Of the two shark scripts in Figure 5, the script that sets the shark's costume and movement should be familiar now. The other script, however, introduces several new ideas.

I'll start with the `set score to (0)` block. `Score` is a variable I created to track when the shark touches the player. To create a variable in your game, select the shark sprite and then click on the *Variable* palette and then *Make new variable*. You will be prompted to enter a variable name. You will also have to choose whether or not you should make the variable accessible to this sprite only or for all sprites. If you make the `score` variable only available to the shark sprite, then the player sprite will not be able to see or use the score directly. For this demo, either option will work.

Next, I'll look at the sensing block `touching (player)?` that is used with the `if ()` block. The `if ()` block is commonly referred to as a conditional statement, meaning the script only runs the blocks inside the `if` statement when the statement is true.

In Figure 5, the condition I'm checking is whether or not the shark is touching the player. Think about all the things you may do in a day that are determined by a simple check first. Before crossing a busy street, for example, you have to check the crosswalk signal for the "walk" sign.

Look at the values in the `touching ()` and `point to ()` blocks. These blocks are capable of sensing other sprites and features in Scratch, such as the edge of the stage, the mouse pointer, and other

sprites by name. All the available things you can sense are contained in the dropdown menu of the block's value field. Identifying other sprites is relatively easy when you only have two sprites, but what if you have a project with four, a dozen, or more sprites? It helps to give your sprites meaningful names, such as `player` and `shark`, so you can clearly identify them. To rename a sprite, select it from the list of sprites and then type a new name in the *sprite properties* field above the scripts area.

When the shark touches the player, the `change (score) by (1)` block adds one to the score. By the way, when you create a variable, that variable will report its value on the stage. You can right-click on this stage reporter to hide or change the display.

The variable is useful because you would expect the value of `score` to change throughout the course of the game, but you only have to look at one variable to get the current value – `score`, in this case. Variables can contain any number or text value you may want to use in your game.

The script in Figure 5 uses one of Scratch's built-in variables (you don't have to create it) called `time`. This is the control that ends the game. If more than 10 seconds elapse, the game ends. Because the time value continually counts, the script initializes the timer to 0 using the reset timer block at the start of the game.

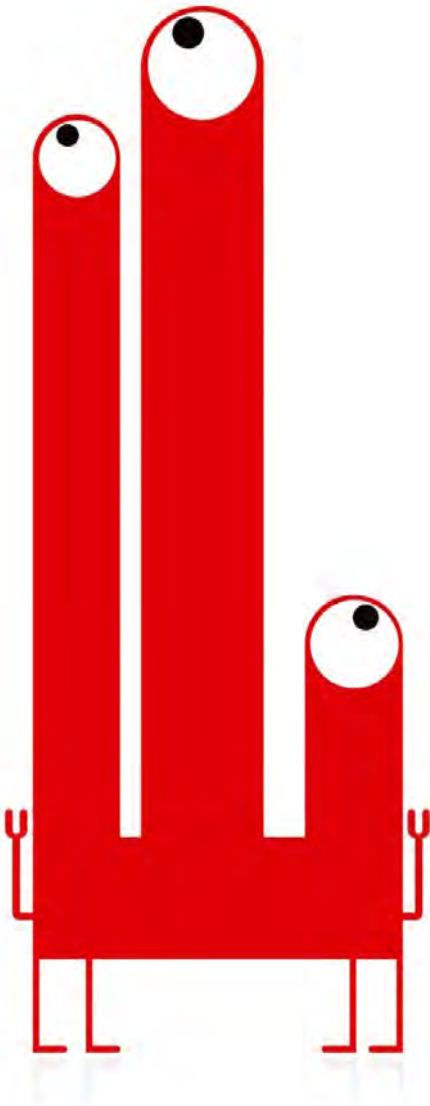
The way the script evaluates `time`'s value is with a `greater than` comparison block, which is available in the Operators palette.

BROADCASTS

One of the most important Scratch features you can learn is broadcasts. Broadcasts enable one sprite to send a message to all the other sprites, and it's a useful way to synchronize the events in your project.

Figures 4 and 5 demonstrate broadcasts. When the shark touches the player, it broadcasts the message "player eaten." You create the broadcast message by clicking in the value area of the `broadcast ()` block and choosing the `new` option.

Sending a broadcast is not enough to cause anything else to happen in the game. You need to tell each sprite to listen for the broadcast message. That's



what's happening in Figure 4; the player is waiting to receive the "player eaten" message, and when it receives the message, it says "Ouch." Of course, you can have multiple broadcast messages within your project, so it makes sense to choose meaningful names. In fact, anytime you name something, make the name meaningful and identifiable. It's similar to naming people Cameron, Christie, and Wally instead of boy1, girl1, and boy2.

What about the stage? I didn't spend any time on the stage, but it's important to know that the stage can have its own scripts and backgrounds, as well. Scratch also includes a library of backgrounds that you can import into the project via the Backgrounds tab when the stage is selected in the sprite list.

EXERCISES

Create a slideshow of your favorite images. Not only does Scratch allow you to draw your own sprites and backgrounds, it also lets you import your own image files for use in the project.

Draw a square by moving and turning the sprite around the stage. Check out the pen down, pen up, and clear blocks in the Pen palette to trace the steps of the sprite. Can you draw other shapes and designs using different values that move and turn blocks?

Try using broadcast messages to animate and tell a story between two or

more sprites. Incorporate backgrounds into the project.

You can build off the Shark Attack game to make it harder. Try making the sharks go faster after a certain amount of time passes or try adding a second shark. You could "level up," so that if the player is able to avoid being eaten for a few seconds, you give the player an extra health point that can be deducted from the shark's final score.

The installed version of Scratch cannot interact with the general purpose input/output (GPIO) pins on the Raspberry Pi. For that, you can download a third-party version of Scratch called ScratchGPIO from Cymplecy [1].

You can use ScratchGPIO to create specific broadcast messages and variables to control the input and output of the GPIO pins on the Pi. GPIO support can be useful for demonstrating circuits and incorporating lights, motors, and sensors into your Scratch project. Happy Scratchin'. *

INFO

[1] Cymplecy: <http://simples.net/scratchgpio/>

THE AUTHOR

Michael Badger authored the *Scratch (1.4 and 2.0) Beginner's Guide* series from Packt Publishing. Learn more and get this project source at scratchguide.com.

raspberry pi GEEK

Raspberry Pi Geek Newsletter SUBSCRIBE NOW – FREE!

Every two months you'll get a sneak peak at the newest *Raspberry Pi Geek* to hit the presses. Read about new projects for beginners and advanced users, tips and tricks, hacks, and the best distributions to try out on your Raspberry Pi!



PYTHON PROGRAMMING BASICS

Taming the Snake

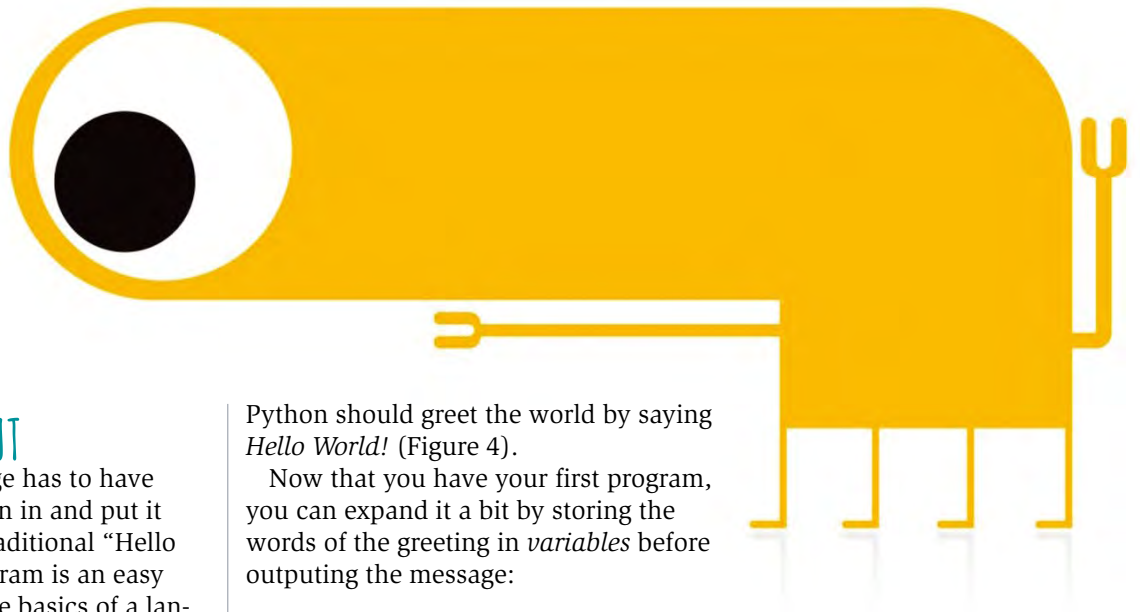
NOW THAT YOU'VE USED TURTLE ART AND SCRATCH, WE'LL SHOW YOU HOW TO USE THE HIGHER LEVEL PROGRAMMING LANGUAGE PYTHON. HOWEVER, THIS PYTHON DOESN'T BITE.
BY SCOTT SUMNER

PYTHON is an easy-to-read programming language that uses indented lines to map blocks of code visually. You can enter a line of code in the terminal while running the Python “interpreter” and receive results immediately, or you can collect a number of lines of code in a file and run them as a program. Raspbian comes with Python version 2.7.9 already installed.

For all of the examples in this article, you can use Raspbian’s text editor to

enter the programs. You’ll find it under the *Accessories* submenu (Figure 1). To enter or run programs, type any code in text that appears in **this font**, or any code in a listing box, exactly as it appears. To save the program, use *File | Save*.

In the left sidebar of the Save dialog, the `pi` folder (with a house icon) is the home directory, where you’ll save your programs. Figure 2 shows how your first code file will look when you finish typing and saving it.



INPUT AND OUTPUT

Any computer language has to have ways to get information in and put it back out again. The traditional “Hello World” computer program is an easy way to demonstrate the basics of a language. Python uses the `print` command to create output:

```
print "Hello World!"
```

Once that’s saved to a text file named `hello.py`, click in the top panel on the terminal icon (Figure 3). The terminal allows you to send commands directly to the Linux operating system instead of with a mouse in a graphical user interface. To run your program, type:

```
python hello.py
```

Python should greet the world by saying *Hello World!* (Figure 4).

Now that you have your first program, you can expand it a bit by storing the words of the greeting in *variables* before outputting the message:

```
greeting = "Hello"
who = "World"
print greeting + " " + who + "!"
```

To try it, type this listing in the text editor and save it to the file `hello2.py`. To run it, type

```
python hello2.py
```

in the terminal. When this program runs, you still see *Hello World!* in the terminal. Behind the scenes, though, things are happening a little differently.

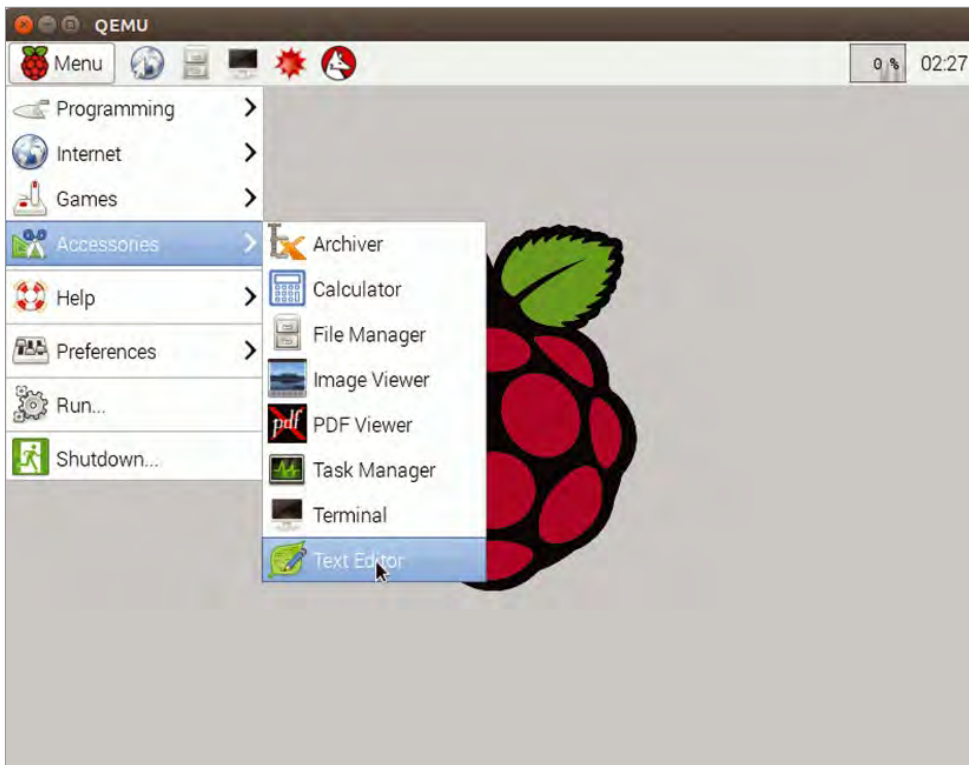


FIGURE 1. In Raspbian, installed programs are found in the main *Menu* in the upper left corner of the desktop.

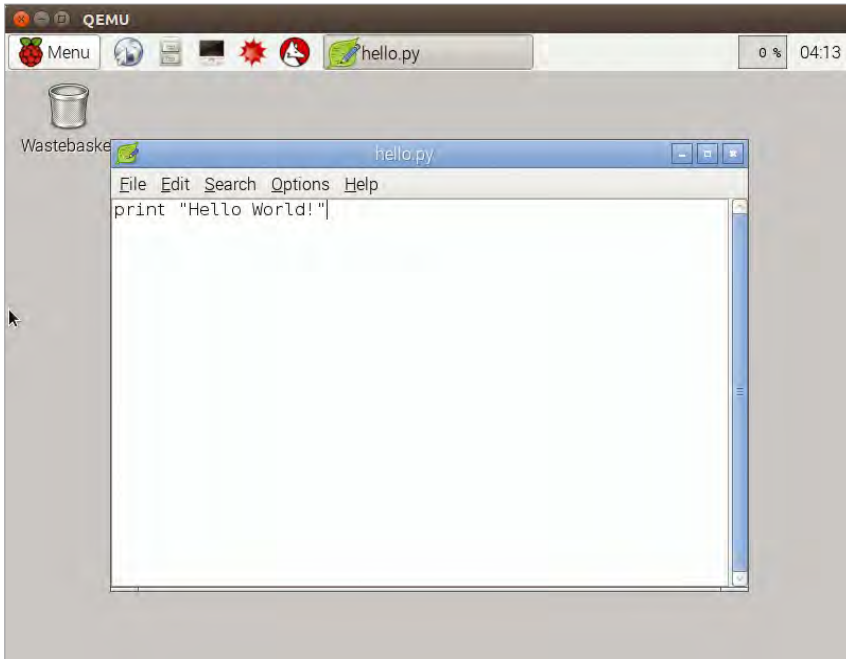


FIGURE 2: After you save your file, the name appears at the top of the window.

The first two lines define variables to the left of the equals sign; here, I've used `greeting` and `who`. Think of a variable as a bucket that Python uses to carry things around. I have a bucket

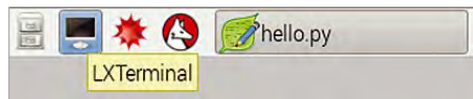


FIGURE 3: The icon that looks like a computer monitor is the terminal.

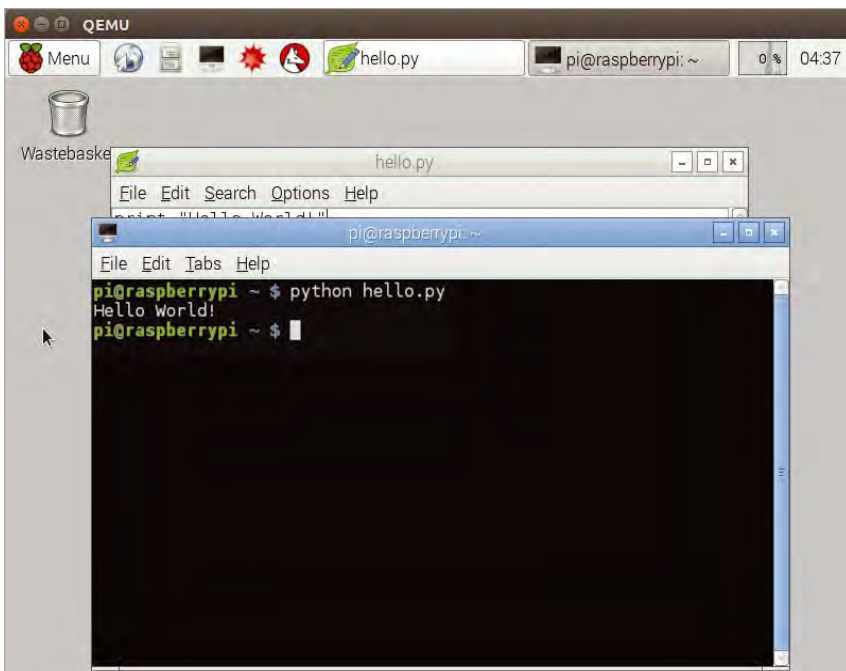


FIGURE 4: Your first program on the Pi!

named `greeting`, and I put the word *Hello* in it (Figure 5). In my bucket called `who`, I put the word *World*.

The `print` statement knows how to output variables, as well. When I give `print` a variable name, it looks in the bucket to see what's there and shows me. I can put multiple pieces of a line of output together by using plus signs.

INPUT, NEED INPUT!

So far, I've only printed words to the screen – output. Now I'm going to show you how to input information to your program so your Raspberry Pi can greet you by name. First, save the lines in Listing 1 to the file `hello3.py`. To run it, type `python hello3.py` in the terminal.

In the second line, your program asks you a question, which shows up in the terminal and waits for you to type and hit the Enter key. Whatever you type, including spaces, goes into the variable name. Once you press Enter, the program will greet you. Now you have a simple program that takes input of your choice (your name, here) and finishes with a customized greeting.

LOOPS

Often in programming it is useful to repeat code over and over again. Loops allow Python to run the same set of in-



FIGURE 5: A variable bucket. Each program can have a lot of buckets in which to carry values or text.

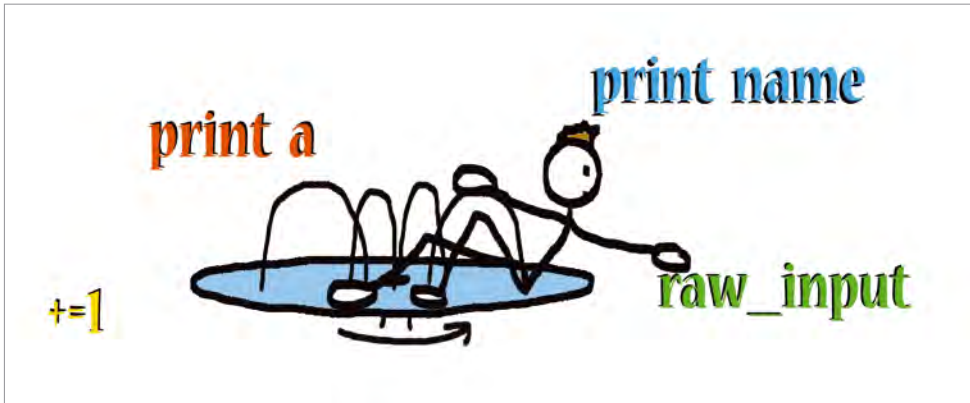


FIGURE 6: Each time the program goes around, it runs all the commands of the merry-go-round.

structions many times; yet, you only have to type it in once. Think of a loop like a merry-go-round. Each repeat of the loop is another trip around (Figure 6). A loop directs how many times to go around, or sometimes its instructions are to “keep riding until something specific happens.”

In this first example, I’ll use a loop to count to 10. To begin, type the lines in Listing 2 in the text editor; then, save the file as `count.py` before running it with `python count.py` in the terminal.

The first line in the example sets up the loop:

- `for` is the Python statement that means “start a loop.”
- `i` is a variable that keeps track of how many times the loop has run.
- `in range (10)` says to repeat the loop 10 times.
- The spaces after `range` and around the `10` are optional. I add them to make the code easier to read.

`range(10)` will work just as well. Also notice that this line ends with a colon (:). Whenever you see a colon in Python, it signals the start of an indented code block. As mentioned at the

top of this article, indentation, or white space at the beginning of a line, indicates a section of Python code that belongs together. It doesn’t matter how many spaces or tabs you use, as long as each line in the block has the same amount of white space.

The `print` statement in the second line should look familiar; I use it to print the content of variable `i`. Because the line is indented, Python recognizes it as a part of the loop. You might wonder what the `+ 1` does. At first, you might think that Python will print `11`,

LISTING 1: `hello3.py`

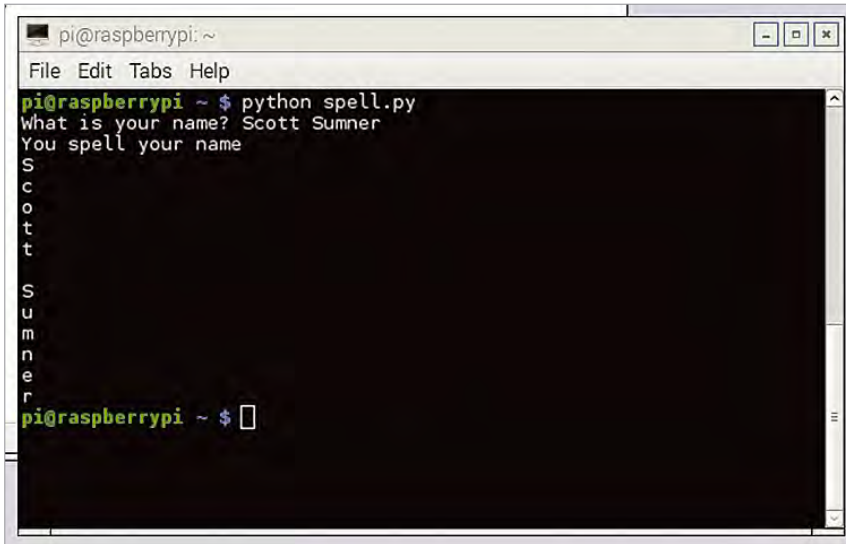
```
greeting = "Hello"
name = raw_input ( "What is your name? " )
print greeting + " " + name + "!"
```

LISTING 2: `count.py`

```
for i in range ( 10 ):
    print i + 1
print "I counted to 10!"
```

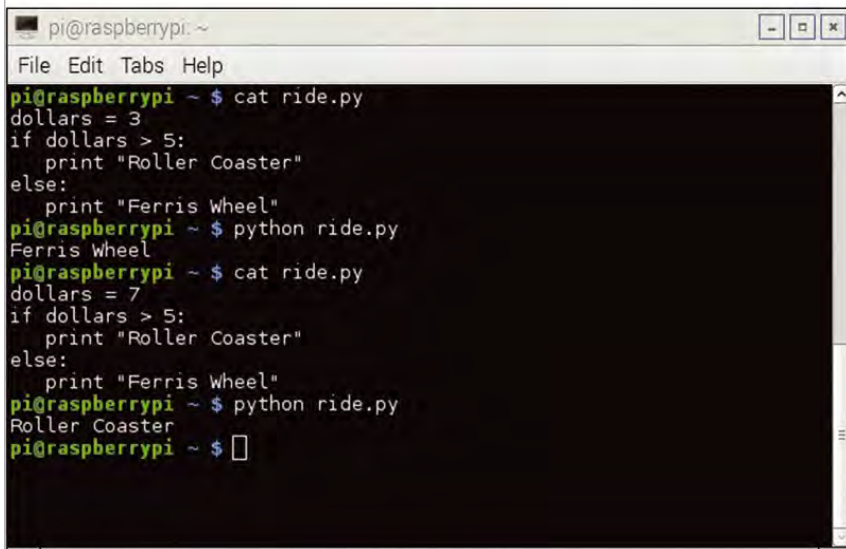
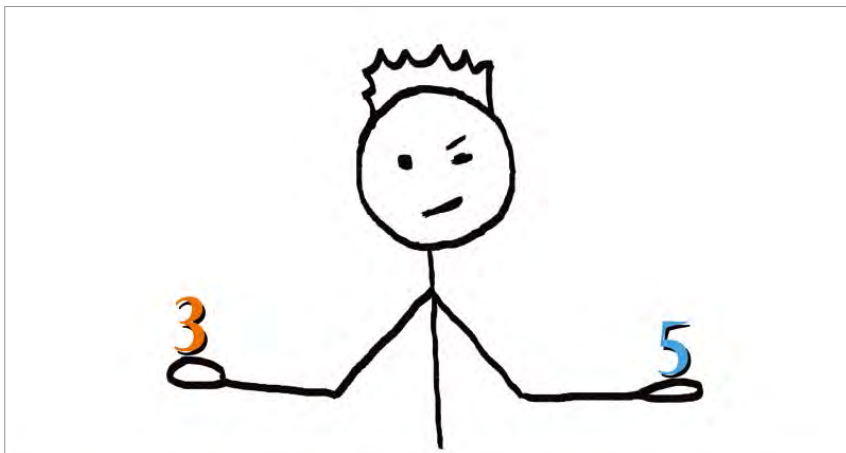


21, and so on to 101, as when combining words in the Hello World example of Listing 1; however, that's not what



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi ~ $ python spell.py  
What is your name? Scott Sumner  
You spell your name  
S  
c  
o  
t  
t  
S  
u  
m  
n  
e  
r  
pi@raspberrypi ~ $
```

FIGURE 7: Working with strings in a loop.



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi ~ $ cat ride.py  
dollars = 3  
if dollars > 5:  
    print "Roller Coaster"  
else:  
    print "Ferris Wheel"  
pi@raspberrypi ~ $ python ride.py  
Ferris Wheel  
pi@raspberrypi ~ $ cat ride.py  
dollars = 7  
if dollars > 5:  
    print "Roller Coaster"  
else:  
    print "Ferris Wheel"  
pi@raspberrypi ~ $ python ride.py  
Roller Coaster  
pi@raspberrypi ~ $
```

FIGURE 8: In the ride.py program, I'm trying to decide whether 3 is greater than 5. Bummer! I only had enough money for the Ferris Wheel, until I borrowed some cash.

happens when I run the code, so what's happening?

Remember that variables are like buckets. Python knows the value of the variable (what's in the bucket), and it also knows whether the bucket is carrying characters or numbers. What is stored in a variable determines its *type* (as in "what type of variable is this?"). Sets of characters are called *strings*. You can recognize them because they are surrounded by double quotes (""). Numbers, as used here, are *integers* and are just common digits in this program. Python treats strings and integers differently, just as you do when you go to Math classes.

Before, I used the plus sign to combine pieces of text when I built my "Hello World" message, but here, `i` represents an integer, so Python treats the plus as a math operation and instead adds 1 to my integer variable. Why am I adding one? Python (and most other languages) start counting at 0. Without the `+ 1`, my program would count from 0 to 9.

Loops can function with strings too, but the program will output each character individually. Try running the `spell.py` program in Listing 3 with the `python spell.py` terminal command. You should see results similar to those in Figure 7.

IF STATEMENTS

Python uses the `if` statement to make a decision by comparing two values. You decide how they are compared and what happens once the comparison is complete. For example, imagine you're at a carnival and you want to go on a ride. However,

LISTING 3: spell.py

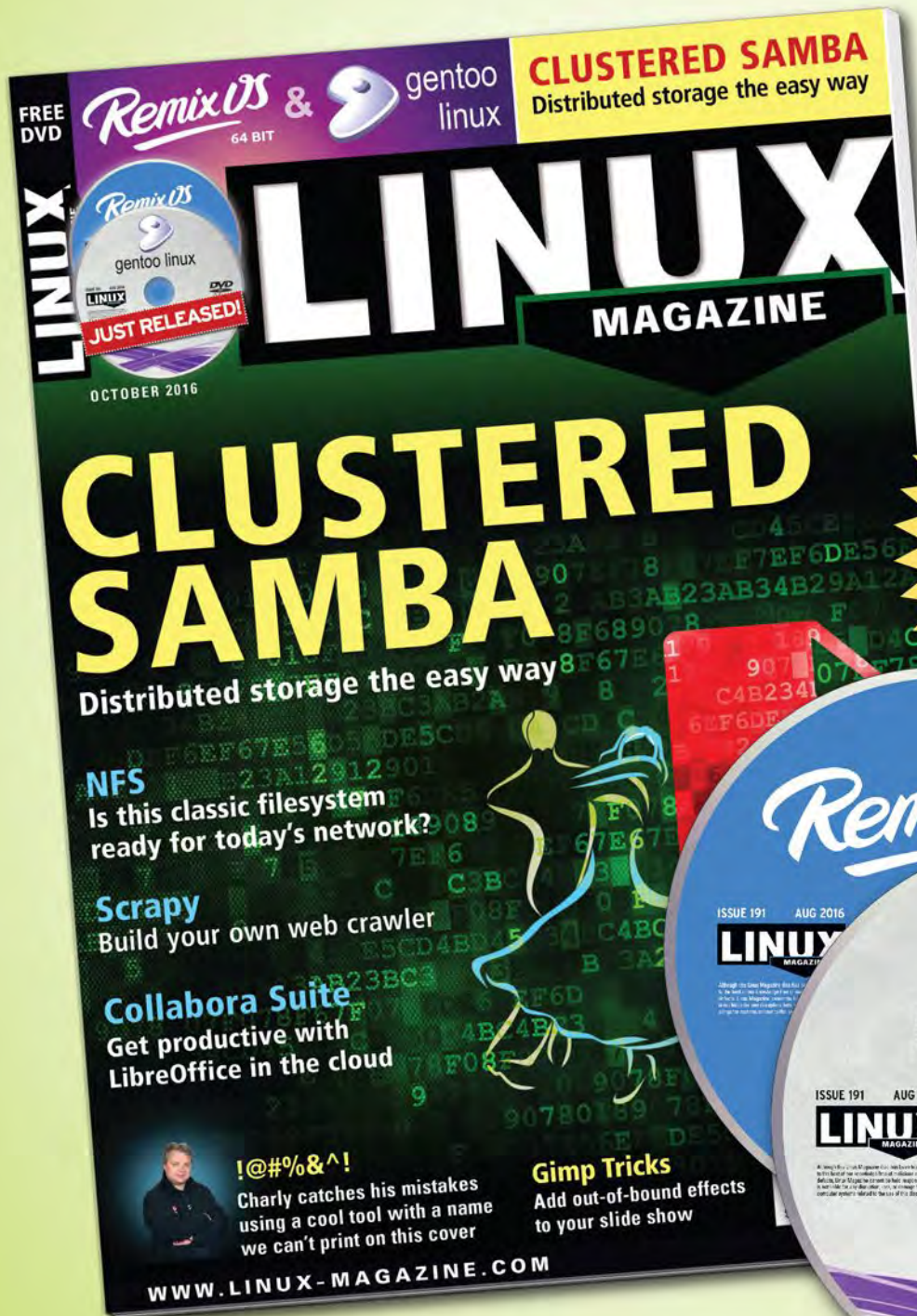
```
name = raw_input ( "What is your name? " )  
print "You spell your name"  
for char in name:  
    print char
```

LISTING 4: ride.py

```
dollars = 3  
if dollars > 5:  
    print "Roller Coaster"  
else:  
    print "Ferris Wheel"
```

Subscribe now!

Don't miss a single issue of the magazine that delivers the in-depth technical solutions you'll use everyday!



GET IT NOW!
SAVE TIME ON DELIVERY WITH OUR PDF EDITION



UK £ 49.90 | Europe € 79.90 | USA / Canada US\$ 99.95 Rest of World (by Airmail) US\$ 109.90
DVD included with print edition • Terms and conditions: <http://goo.gl/SSSQer>

shop.linuxnewmedia.com/subs

LISTING 5: ride2.py

```

01 money = raw_input ( "How much money do you have? " )
02 dollars = int ( money )
03 if dollars < 3:
04     print "You don't have enough money to do anything"
05 elif dollars >= 20:
06     print "Bring a friend"
07 elif dollars >= 15:
08     print "Lunch in the park"
09 elif dollars >= 10:
10     print "Roller Coaster + on-ride photo"
11 elif dollars >= 5:
12     print "Roller Coaster"
13 elif dollars >= 3:
14     print "Ferris Wheel"

```

you might not have enough money to go on the \$5.00 daredevil ride, so you have to look at your cash on hand and make a decision. The `ride.py` program in Listing 4 shows how Python makes that decision with an `if` statement.

You can change the number of `dollars` on the first line to see which ride you can afford (Figure 8). Line 2 works (and reads) just as it looks: “If `dollars` is greater than 5.” This statement is called a condition. Python will then decide whether the condition is `True` or `False`. The indented code immediately after the

`if` is executed when it is `True`. If an `else` is present (line 4), the indented lines that follow are executed when the condition is `False`.

Now I’ll expand the program a bit so that it asks how much money I have (Listing 5); then, the program can decide what I can do depending on my input. Call this program `ride2.py` and run it with `python ride2.py`. Run this program a number of times with different

input values to see the different responses. (See the “Using the Terminal” box for an easy way to repeat the same terminal command.)

You’ll probably recognize `raw_input` on line 1 from an earlier program. The `raw_input` function always returns a string, even if you enter numbers; therefore, the first thing you have to do is convert the string to a number with `int` (line 2). A “5” will become 5, “27” will become 27, and so on. However, “five” will not become 5. Unfortunately Python isn’t quite that smart.

USING THE TERMINAL

If you want to run the same command over, you can press the Up arrow to get your previous command back; then just press Enter to run it! The Up arrow works for an entire history of commands. If you continue to press the Up arrow, you will see older and older entries you made in the terminal. It will go all the way back to when you started up your Pi.

The Pi can also help you remember file names. Type `python ri` then press the Tab key twice. It will fill in the blanks or show you a list of choices. Then you can type a few more characters and press Tab again, or just type in the missing characters of the file name.

TABLE 1: Python Expressions

Expression	Name	Comparison
<code>==</code>	Equals	Is <i>a</i> the same as <i>b</i> ?
<code>!=</code>	Not equal	Is <i>a</i> different from <i>b</i> ?
<code><</code>	Less than	Is <i>a</i> less than <i>b</i> ?
<code>></code>	Greater than	Is <i>a</i> greater than <i>b</i> ?
<code><=</code>	Less than or equal to	Is <i>a</i> less than or equal to <i>b</i> ?
<code>>=</code>	Greater than or equal to	Is <i>a</i> greater than or equal to <i>b</i> ?

Line 3 makes the first comparison. If the number of `dollars` is less than 3, you get the message: *You don't have enough money to do anything*. On line 5 is a new command, `elif` (else if). If the comparison on line 3 is `False` (i.e., you have more than \$3), then Python checks the next unindented line. That's line 5, which checks for amounts of \$20 or more, in which case, you can bring a friend. Once any condition is `True`, the indented code directly under it runs, and the `if` is complete. None of the other conditions will be checked or run.

Notice that line 13 checks to see whether `dollars` is greater than or equal to 3; otherwise, you wouldn't get any output if you entered 3, because none of the other `elif` statements would apply. Table 1 shows other Python expressions you can use for comparisons in `if` statements. How could you change `ride2.py` to check for exactly the right amount of money?

Think of sets of `if/elif` statements as a set of doors in a school hallway. Each door posts its rule, which you can evaluate to determine whether it's `True` or `False`. Once you find a door that you can enter, you've found your class and don't have to check any other rules. As shown in Figure 9, if I've washed my hands, I can go in the first door; if not, I move on to the next one. Have I read chapter 3? If so, I can enter here; otherwise, I check the last door. If I've done my homework, I can enter. Of course, if none of these rules apply, I'm left standing in the hall.

GUESS MY NUMBER GAME

Listing 6 puts together everything you've learned so far into a "guess my number" game. Right at the beginning, you'll see a new command. The `import` statement asks Python to bring in new modules. A module is additional Python code someone wrote to do a specific task. Python has thousands of modules to do anything from showing pictures and graphics to controlling robots and everything in between. The `random` module lets Python generate random numbers.

You might notice that lines 2, 7, 9, and 23 are blank. Python doesn't care about blank lines, but they make code easier to read on a screen. It's never bad to add a blank line to separate different parts of your program.

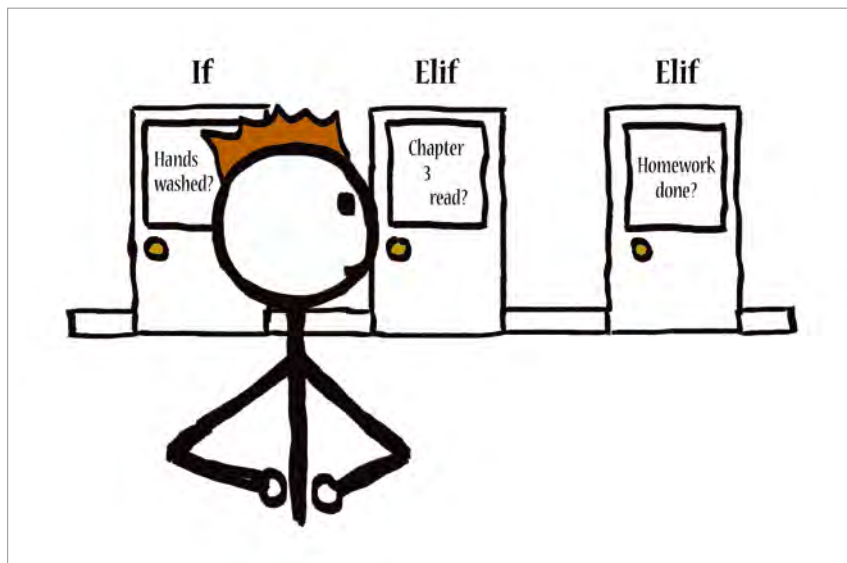


FIGURE 9. Making a decision with `if` and `elif`. As I start walking down the hallway, the rule on each door determines whether I can enter.

On lines 3 and 4, I set up variables the program uses to determine the lowest and highest number that Python can choose. The `guesses` variable on line 5 keeps track of how many guesses have been made during the game.

Line 6, starting with `target =`, works the same way as the `raw_input` lines in

LISTING 6: `.numberGuess.py`

```

01 import random
02
03 low = 1
04 high = 100
05 guesses = 0
06 target = random.randint ( low , high )
07
08 print "I've picked a number between {0} and {1}; start guessing, and I'll
tell you if it's higher or lower!".format ( low , high )
09
10 guessing = True
11 while guessing:
12     guess = int ( raw_input ( "Your guess?" ) )
13     if guess == target:
14         guesses += 1
15         print "You got it! My number is {0}".format ( target )
16         guessing = False
17     elif guess < target:
18         print "It's greater than {0}!".format ( guess )
19         guesses += 1
20     elif guess > target:
21         print "It's less than {0}!".format ( guess )
22         guesses += 1
23
24 print "You figured out my number in {0} guesses!".format ( guesses )

```

previous programs, except instead of asking the user for input, it's asking Python for a random number between 1 and 100 and storing it in `target`. The compound name `random.randint` tells Python to access the `random` module and call its `randint` function. The `(**low**, high**)` tells `randint` the lowest and highest numbers it can choose.

Line 8, prints a welcome message to describe the game. What's new here is the `{0}` in the middle of the words and `format` on the end. The `.format` structure is another way to combine multiple variables when printing. In `hello2.py`, plus signs were used to build and print a message. This is just another way to do the same thing. In this way, you can show a message that's easy to read, label your output, or fill in a sentence with values from a program.

To use `format`, I start with a string. (See the double quotes at the beginning and the end?) Whenever I want to add a value from my program, I add a placeholder. That's the numbers in curly braces. It tells `format` to "put the variable here." If I were to stop there, you would see:

```
I've picked a number between {0} and {1};
start guessing, and I'll tell you if
it's higher or lower!
```

That's not very helpful, so I add `.format` on the end. When I tack on `(low , high)`, it tells `format` what to plug in to the placeholders.

In line 10, I set the `guessing` variable, which I set to `True`. This will be used in the `while` on line 11.

The `while` statement is another type of loop that checks its condition and, if

`True`, runs the indented code block beneath it. When the code block finishes, it checks again. If it's still `True`, then the code block runs again. That process repeats until the condition becomes `False`. In the number-guessing game, that means the computer will keep asking for numbers until you get it right!

Line 12 combines a few things you've seen before: the `raw_input` and `int` from `ride2.py`. Here, I've just combined them so that they only use one line. Working from the inside out, I ask for `Your guess?` with `raw_input`, change the string to an integer with `int`, then store it in the variable called `guess`.

On line 13, I check to see if you picked the right number. If the contents of `guess` and `target` are equal (`==`) then you win! You still made a guess though, so line 14 adds 1 to the guess count stored in `guesses`. The `+= 1` expression is another shortcut. It says "add the number on the right to the value of the variable on the left and store that sum in the variable on the left".

Line 15 uses `print` and `format` to remind you what the number was, and line 16 changes `guessing` to `False`, so the next time Python checks `guessing` when it loops back to line 11 it will be `False` and the loop will end. Python then finds the next line of code that's not part of the `while` loop, which is on line 24.

Falling out of the loop only happens if the guess is correct! Until then, if you haven't guessed the correct number, you still have to handle that case. Code blocks 17-19 and 20-22 are almost identical. Can you spot the differences? The first section checks to see if your guess

LISTING 7: Error Message for oops2.py

```
File "oops2.py", line 3
    print My name is Scott and my favorite number is number
          ^
SyntaxError: invalid syntax
```

LISTING 8: Concatenation Error

```
Traceback (most recent call last):
  File "error.py", line 3, in <module>
    print "My name is Scott and my favorite number is " + number
TypeError: cannot concatenate 'str' and 'int' objects
```

TALKING TO PYTHON

It's also possible to talk to Python directly. Go to the terminal and type `python`. After a short introduction, you'll see the `>>>` prompt, which means you've entered the Python interpreter, and it is ready for you to talk to it (Figure 10). Python makes a great calculator, so enter a math problem and press Enter. You use the usual plus sign (+) for addition and the hyphen (-) for subtraction, but instead of an 'x' for multiply, use an asterisk (*); to divide, use the slash (/).

The Python console is also a great way to see if you have a module on your Pi. Try typing `import random`. Python immediately returns a new prompt (`>>>`), so you're good to go! If you try to load something that doesn't exist, you'll get an error; for example, try entering `import imaginaryModule`.

When you're done playing in the Python interpreter, press `Ctrl + D` or type `quit()` to exit; then, type `exit` to exit the terminal.

is less than the `target`, and the second section checks to see if your guess is greater than the selection. Each section has a different message to give you a hint about what direction your next guess should take.

Finally, once you've figured out the number, line 24 tells you how many guesses it took.

Now it's your turn! Start with `numberGuess.py` and see if you can make the following changes:

- Make Python ask you for the lowest and highest numbers to use
- Print the number of guesses used before asking for a guess each time
- Limit the number of guesses you have to figure out the number. If you run out of guesses, print a message that tells what the number was and stop asking for guesses.

ERRORS

Sooner or later, you'll accidentally ask Python to do something it doesn't understand or can't do. These are called errors and have to be fixed before your program will run. Sometimes the messages can be strange, but if you know where to look, they are very helpful. Here's an example I call `oops.py`:

```
01 number = 10
02 name = Scott
03 print My name is Scott and my
    favorite number is number
```

(The arrow at the end of the line just means to continue typing the line; don't try to type the arrow!) To try it,

type and save this listing in the text editor – without the line numbers – then enter `python oops.py` in the terminal. You should see a message like this:

```
Traceback (most recent call last):
  File "error.py", line 2, in <module>
    name = Scott
NameError: name 'Scott' is not defined
```

Python is telling you that something isn't right. But look right in the middle – a line number! The first clue. Can you spot what I did wrong?

I forgot to put double quotes around my name. The last line tells me what Python doesn't understand: `name 'Scott' is`

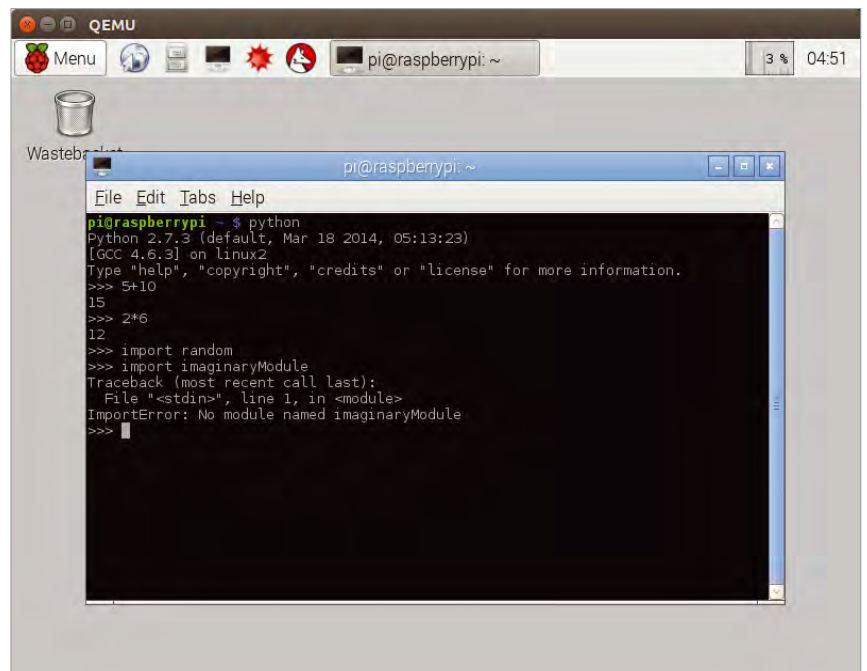


FIGURE 10: Using the Python interpreter.

not defined. Python doesn't know what *Scott* is, so it's stuck. Once I fix it my code (`oops2.py`), it looks like this:

```
01 number = 10
02 name = "Scott"
03 print My name is Scott and my
      favorite number is number
```

It still doesn't work, but I got another a hint (Listing 7). My problem is in line 3. Moreover, Python is pointing me to where it thinks the problem is with the caret (^). Any time Python talks about a *syntax* error, it means there's something wrong with the format of the code. Can you spot my error?

I forgot to put double quotes on my output string again. I'll try one more time:

```
01 number = 10
02 name = "Scott"
03 print "My name is Scott and my
      favorite number is number"
```

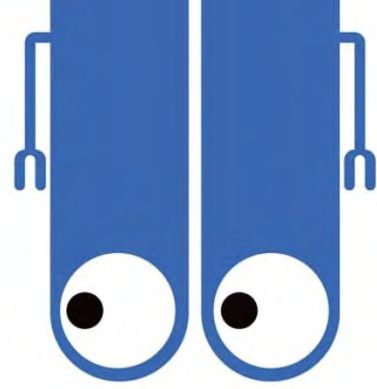
It works! Well, sort of. Python isn't giving me an error, but the program is not doing what I want. There's nothing wrong with the code I've typed, but I haven't successfully figured out how to tell Python what I want to do yet. These can be hard to figure out since you don't get any hints from Python as to what's wrong. I'll try to fix it with this:

```
01 number = 10
02 name = "Scott"
03 print "My name is Scott and my favorite
      number is " + number
```

Now I have an informative error message (Listing 8) that is telling me I cannot *concatenate*, or put together, string and integer objects on line 3. Remember that I fixed this problem in the `number-Guess.py` program with the `format` structure:

```
number = 10
name = "Scott"
print "My name is Scott and my favorite
      number is {}".format ( number )
```

The output for this code is *My name is Scott and my favorite number is 10*. Great, it's working! Now I'm going to show my friend Joe. His favorite number is 23, so lines 1 and 2 need to be:



```
number = 23
name = "Joe"
```

The output now says: *My name is Scott and my favorite number is 23*. Uh oh, I stole Joe's favorite number. Can you see what I did wrong? I need to change line 3:

```
print "My name is {1} and my favorite
      number is {0}".format ( number , name )
```

Great! It's all working! The output now says, *My name is Joe and my favorite number is 23*. Notice that the placeholders in line 3 are out of order; however, as long as the variables inside the `format` parentheses are in the right order, I don't have to renumber anything as I add more information.

Even with this simple program, it took a lot of tries to get it right. Finding errors and fixing them is a big part of the challenge of programming. Don't let it get you down; just keep trying and take it in small chunks.

EXERCISES

With the information I've given you in this article, you should be able to write a program to calculate your age:

- Ask what year it is
- Ask what year you were born
- Show the answer

In another program, you could roll a dice:

- Ask how many sides the dice has
- Ask how many times the dice should be rolled
- Show each roll and then the total of all rolls

In addition to running program files with your Python code, you can talk directly to Python in the terminal. See the "Talking to Python" box for more information. *

ABOUT THE AUTHOR

Scott has been programming in Python as a hobby for many years. He's very grateful to Eliese Donhardt for illustrating this special edition article.

Shop the Shop

shop.linuxnewmedia.com

Want to subscribe?

Searching for that back issue you really wish you'd picked up at the newsstand?

Discover the past and invest in a new year of IT solutions at Linux New Media's online store.

shop.linuxnewmedia.com

DIGITAL & PRINT SUBSCRIPTIONS



SPECIAL EDITIONS



BUILDING AN ELECTRONICS PROJECT WITH THE BREADBOARD AND GPIO PINS

Score Keeper!

THIS EASY ELECTRONICS PROJECT INTRODUCES YOU TO THE BREADBOARD AND OTHER TOOLS FOR INTEGRATING YOUR RASPBERRY PI WITH ELECTRICAL CIRCUITS.
BY BILL SUMNER

YOU CAN HAVE a lot of fun on a Raspberry Pi just playing with the software. Even beginning users can quickly learn to make music, play a game, surf the web, or write a letter using programs that run internally on the Raspberry Pi system. But many believe the real fun is in using the Raspberry Pi with electronics projects.

With a few extra parts and a little knowledge of electronics, you can make your Pi turn on lights, ring bells, and control motors. The possibilities for cool projects are endless. Of course, the best ideas are the ones you come up with yourself, but this article will help you get started with electronics on the Raspberry Pi. I'll show you how to use your Raspberry Pi to control a simple scoreboard for tracking the balls and strikes in a baseball game. Along the way, you'll discover the Raspberry Pi GPIO pins and learn about common electronic components like breadboards, jumpers, resistors, and pushbuttons.

CIRCUITS

Electrical current brings energy to an electronic device such as a light or a motor. Current flows through an electri-

cal circuit. The electrical properties of the circuit are typically expressed with the following parameters:

- **Volts:** Think of the voltage as the amount of pressure trying to push the electrons along the wire. The Pi uses 3.3 volts (3.3V) for providing power to circuits through the GPIO pins, which you'll learn about later in this article.
- **Ohms:** Ohms measure the amount of resistance to the flow of the electrons – how hard it is for the voltage to push electrons through a resistor. This project will use resistors to control how much electrical current will flow in the circuit. (See the box titled “Resistor Color Codes.”)
- **Amperes:** Amperes are a measure of the electrical current – the number of electrons that flow past any point of the wire each second. However, one ampere (or one *amp*) is a really big number of electrons – far too much current for anything in this article. None of the parts described in this article needs more than a few thousandths of an Ampere: a few milliamperes (mA).

These parameters are related through an equation known as Ohm's law, which states that the number of volts is equal

to the number of amperes multiplied by the number of ohms.

If you know two of these values, you can calculate the third. Since you almost always know the voltage (3.3V from the Pi), you can choose the resistor you need to provide the necessary electrical current to the electrical components on the circuit.

To calculate the ohms needed to provide a specific amount of current, divide the volts by the amperes of current you want:

$$\text{ohms} = \text{volts} / \text{amperes}$$

For example: most LEDs need about 3mA (.003 amps) or maybe just a little more. The Pi provides 3.3 volts, so:

$$\text{ohms} = 3.3 / .003$$

Use the calculator program on your Pi (*Menu | Accessories | Calculator*) and enter 3.3 / .003

$$\text{ohms} = 1100$$

so you will need an 1100-ohm resistor in the circuit with the LED to limit the current to 3mA.

However, 3mA is actually on the low-end of what LEDs like, and since a 1K (1000-ohm) resistor is a common value, you can use the 1K resistor. 1000 ohms gives the LED a little bit more current than the 1100 ohm resistor and it will work just fine.

To see just how much current the 1K resistor gives the LED, modify the equation as follows:

$$\text{amperes} = \text{volts} / \text{ohms}$$

On the calculator:

$$3.3 / 1000$$

The LED gets 3.3mA – about 10 percent more, and that works just fine.

WARNING: Do not connect more than 3.3V to any of the GPIO pins! See the box titled “Warnings” for more precautions you’ll need to keep in mind when wiring your Raspberry Pi circuit.

PI IN A CIRCUIT

The Raspberry Pi is designed to serve as a power source for small electrical

circuits. As the previous section described, the Pi is capable of providing 3.3V for an electrical circuit. 3.3V is enough for a simple circuit like the ones described in this article. (Bigger and more powerful creations will need an external power source.)

The Raspberry Pi communicates electrical information with the outside world using a bank of pins on the Rasp Pi board known as the GPIO (General Purpose Input and Output) (Figure 1). GPIO pin layouts differ for different versions of the Raspberry Pi. The pin layout for the Raspberry Pi 2 is shown in Figure 2.

The GPIO pins serve as an interface for letting the software running on the

RESISTOR COLOR CODES

You can read the value of a resistor by decoding the colored rings around the resistor near one end. The colors give the amount of resistance (measured in ohms) provided by the resistor.

The color of each ring is code for one digit of the resistance value.

The color codes were chosen to make it easy to remember which color represents each digit from zero to nine:

- 0 Black
- 1 Brown
- 2 Red
- 3 Orange
- 4 Yellow
- 5 Green
- 6 Blue
- 7 Violet
- 8 Grey
- 9 White

To read the value of the resistor, hold the resistor so that the colored rings are closest to the left end, and then decode each ring from left to right into a digit. The leftmost color is the “tens” digit. The next color is the “ones” digit. The third color is “number of zeros to append to the first two.” The fourth color (if there is one) can be ignored for our purposes. It encodes the “tolerance” of the resistor value, which tells the maximum amount that this resistor

might be away from the value coded by the first three rings. With modern manufacturing, all of the resistors in the kit will be close enough to their encoded value for this project.

EXAMPLE A: Brown, Black, Red 1, 0, 2 zeros = 1000 ohms (1 thousand ohms) (1K)

EXAMPLE B: Yellow, Violet, Brown 4, 7, 1 zero = 470 ohms (470 ohms)

EXAMPLE C: Orange, Orange, Brown 3, 3, 1 zero = 330 ohms (330 ohms)

EXAMPLE D: Brown, Black, Orange 1, 0, 3 zeros = 10,000 ohms (10 thousand ohms) (10K)

Note: Large resistor values are often written as a value followed by K or M (kilo or mega):

EXAMPLE E: 4.7K is 4.7 thousand ohms or 4700 ohms

EXAMPLE F: 33M is 33 million ohms or 33,000,000 ohms

Sometimes the K or M is not at the end but in the position of the decimal point:

EXAMPLE G: 2200 ohms written as 2K2

EXAMPLE H: 5.6 million ohms written as 5M6

WARNINGS

Never connect more than 3.3V to any of the GPIO pins. A higher voltage will damage it. If you experiment with the breadboard, DO NOT connect anything to either of the 5V pins on the cobbler. This will almost surely damage the Pi.

Note that many other Integrated Circuit (IC) chips use 5V. When connecting any of these 5V ICs to the Pi's GPIO pins, always use a voltage level shifter to convert the 5V from the IC into 3.3V for the Pi. Any Pi project plan that uses some 5V chips should tell you how to do this.

The Pi has very low limits on current both for each individual GPIO pin and for the total current from all the pins. Keep in mind:

- Each GPIO pin is limited to 16mA.
- The total current supplied by the Pi is limited to 50mA.

All of the current you pull from all of the GPIO pins plus the current you pull from the 3.3V power pin MUST total no more than 50mA. If you pull more than 50mA total, you can overheat the Pi and damage it.

Raspberry Pi talk with electronic devices in the real world. But, as you can see in Figure 1, the GPIO pins are close together and are very difficult to work with if you wish to connect wires and other electrical components. People who play with Raspberry Pi electronics typically like to wire up their electronic circuits using a breadboard (Figure 3), then use a specially designed cable and interface called a cobbler (Figure 4) to connect the breadboard with the Raspberry Pi GPIO pins. The breadboard is designed to make it easy to build circuits by connecting wires, resistors, and other electronic components.

The project in this article uses a breadboard and cobbler to connect the Rasp-

berry Pi with electronic components through the GPIO.

THE PROJECT

This simple project manages an old-time baseball scoreboard like the ones that still exist at some fields: Separate light

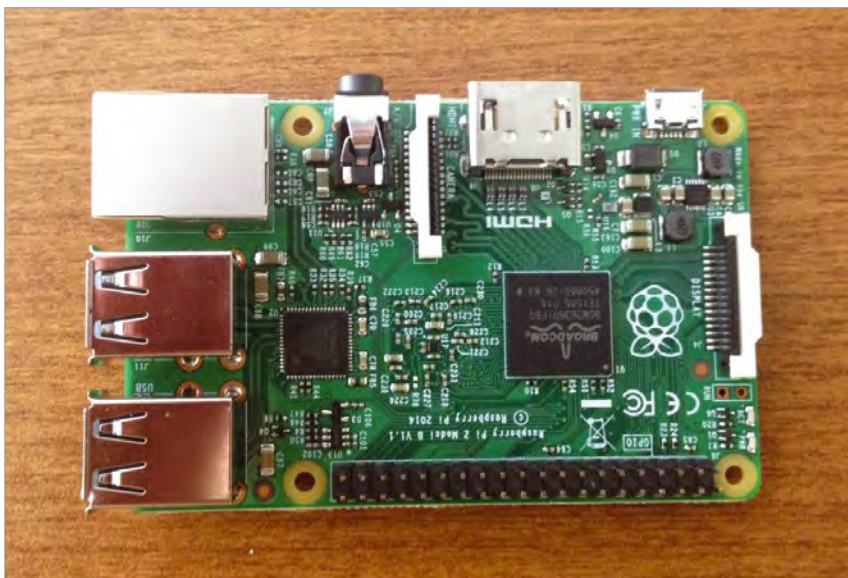


FIGURE 1: The GPIO pins let the Raspberry Pi communicate with external circuits connected through electrical circuits.

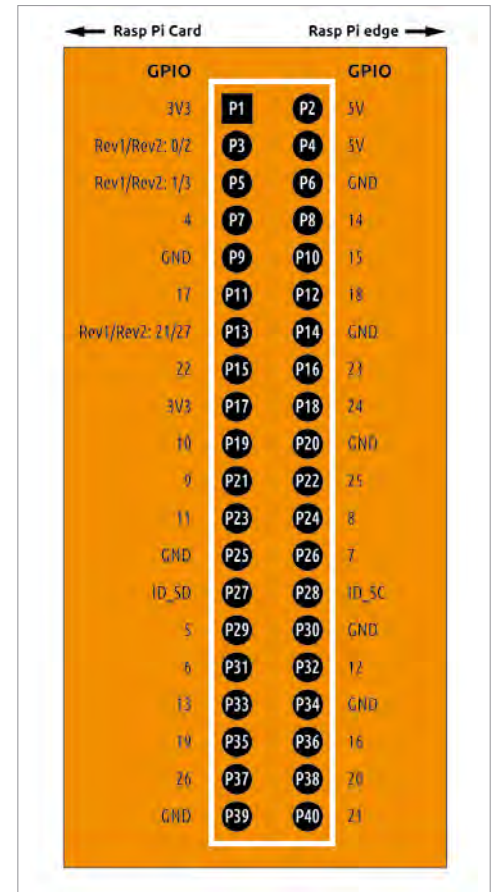
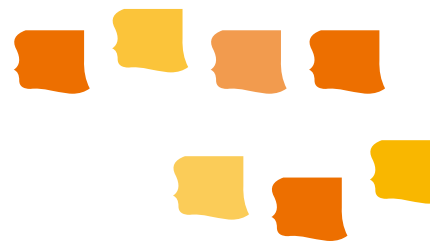


FIGURE 2: Pin layout for the Raspberry Pi GPIO pins.



bulbs track the balls, strikes, and outs in the inning. If you are more familiar with other sports, you could easily adapt this concept to implement a soccer, hockey, rugby, or basketball scoreboard.

Although I will only work with individual lights, a more advanced extension of this concept could illuminate patterns of lights that form numbers or letters. See the box titled “Batter Up” for some background on pitch counts and scoreboards in baseball.

The scoreboard provides three rows of lights, representing the “balls,” “strikes,” and “outs.” The scorekeeper pushes one pushbutton to record a “ball” and another pushbutton to record a “strike.” After four balls, the ball lights blink to signify a walk. After three strikes, the strike lights blink to signify a strikeout. Press the button again to reset the count. Another pushbutton records outs. After three outs, the “out” lights blink, meaning the teams switch and the other team gets to bat. Press the out button again to reset.

The project has two big parts (see the “What You’ll Need” box): A breadboard with seven LEDs (3 balls, 2 strikes, 2 outs) and three pushbuttons (“umpire

called a ball,” “umpire called a strike,” “team made an out”) serves as the scoreboard. There are seven copies of the output circuit (LEDs) and three copies of the input circuit (buttons).

The other important component is a Python program that runs the breadboard and knows enough baseball rules to run the scoreboard:

- Four balls is a “walk”
- Three strikes is an “out”
- Three outs and the other team comes to bat

I have already written the Python program and tested it with my breadboard on a real baseball game.

This project uses inexpensive parts, uses only the 3.3-Volt power supplied by the Pi, needs no soldering and trimming wires, and doesn’t even require any programming – I wrote the first program for you. The program is in Python, so with a little bit of “cut and paste,” you can modify the script and make this same hardware do a lot of other things.

THE CIRCUITS

This project uses two types of circuits – input and output. The input circuit lets you advance the scoreboard by pressing a pushbutton. The output circuit has an LED light that illuminates to display the score. For each of these circuits, the wires on the breadboard are exactly the same. The only difference is that the output circuit has an LED and the input circuit has a pushbutton.

The wiring for both circuits:

1. Starts at a pin on the cobbler (comes from the Pi)
2. Goes through a 1000-ohm resistor
3. Goes through either the LED (output) or the pushbutton (input)

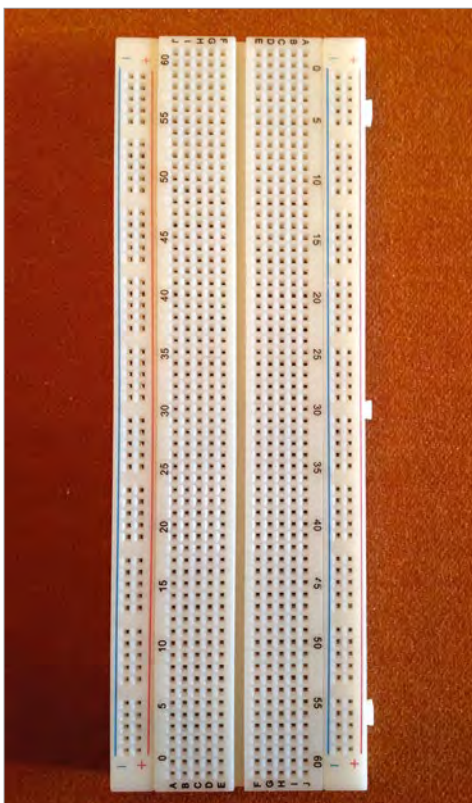


FIGURE 3: A breadboard offers a convenient space for wiring electrical components.

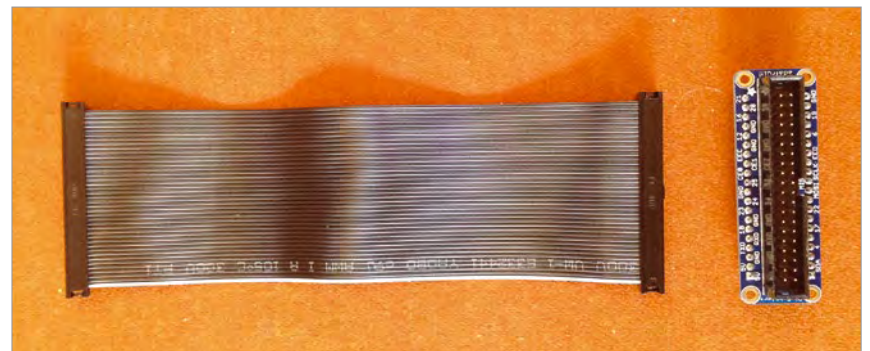


FIGURE 4: A cobbler connects the breadboard with the Raspberry Pi GPIO pins.

BATTER UP

Baseball looks a little like the game of cricket at first glance. A player (called the batter) stands with a bat (a big stick) and waits for the pitcher to throw the ball. The pitcher is expected to throw the ball at a well defined area in front of the batter (called the strike zone). If the pitch does not pass through the strike zone, the batter doesn't have to swing at it, and the pitch is called a "ball." After four balls, the batter can go to first base, and this is called a "walk."

If the pitch does land in the strike zone, the batter is expected to swing at it. If the batter swings and misses, or doesn't swing at all, the pitch is called a "strike." After 3 strikes, the batter is "out," meaning the batter has lost the chance to hit the ball and another batter now has a chance.

If the player manages to hit the ball, any number of scenarios might occur depending on where the ball is hit and what the defensive team does with it, but basically, the batter either reaches base or is "out." After three outs, the teams trade: the defensive team gets to bat and the team that was batting goes out to the field. Years ago, many baseball fields had a simple scoreboard like the one described in this article for tracking the number of balls, strikes, and outs. Today, major league fields have wild and colorful electronic scoreboards that light up like a video game screen and provide a lush, high-resolution display. Many small local fields, however, still have this simple type of scoreboard, with rows of lights representing the balls, strikes, and outs.

4. Goes to the "ground bus" – more about this later
5. Goes to the "gnd" pin on the cobbler (and back to the Pi)

A program running on the Raspberry Pi tells the Pi which GPIO pins are used as inputs and which pins are used as outputs.

An output pin on the GPIO acts just like a light switch on the wall. It's either *on* or *off* depending on which way the program wants it. The program can set the pin *on* or *off* any time it wants to.

An input pin is more interesting. The Pi hardware watches the voltage on the pin to see whether it is close to 3.3V or

close to zero volts. Any time the program wants to know, it can ask the Pi: "Which voltage do you see right now on this pin?" and the Pi returns the answer. For a more in-depth look at the circuits, see the "Circuit Diagram" box.

The designers of the Pi have made it easy for extra parts like our breadboard to set the voltage on an input pin high or low. When the program tells the Pi to set up a pin for input, it can also tell the Pi: "Unless the outside circuit connects the input pin to *gnd*, make the voltage on the pin close to 3.3V". All you need to do is let the button connect the pin to *gnd* through the 1K resistor or not. This pulls the voltage on the input pin "close to zero" or leaves it "close to 3.3V."

ASSEMBLING THE PARTS ONTO THE BREADBOARD

Once you have gathered all the parts, it's time to start assembling. The first step is to put together all the parts on the breadboard. Don't attach the ribbon cable that came with the cobbler to the cobbler or the Pi yet. Please refer to Figure 5, the "Completed Breadboard," frequently as you assemble your breadboard.

The empty breadboard should be oriented with the "long way" going left to right and the "short way" going from near you to farther away. This project will ignore the column numbers printed

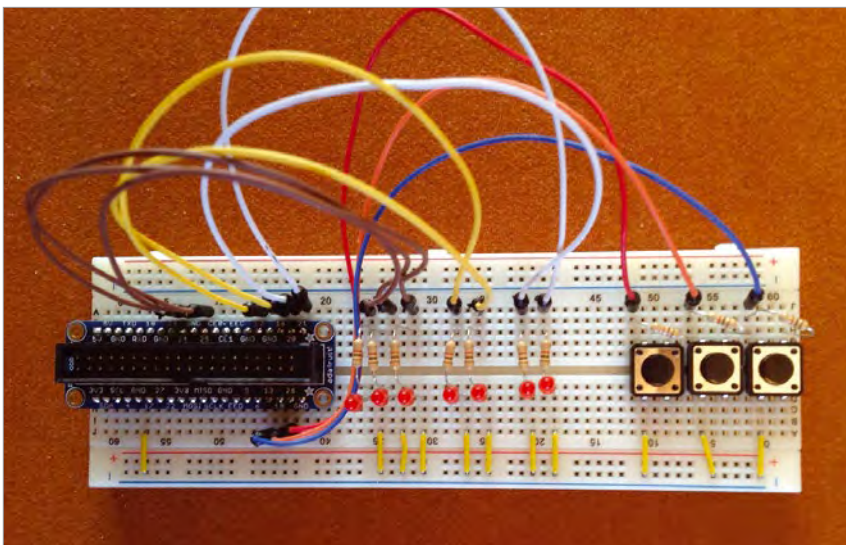


FIGURE 5: The completed project breadboard before it is attached to the Pi.

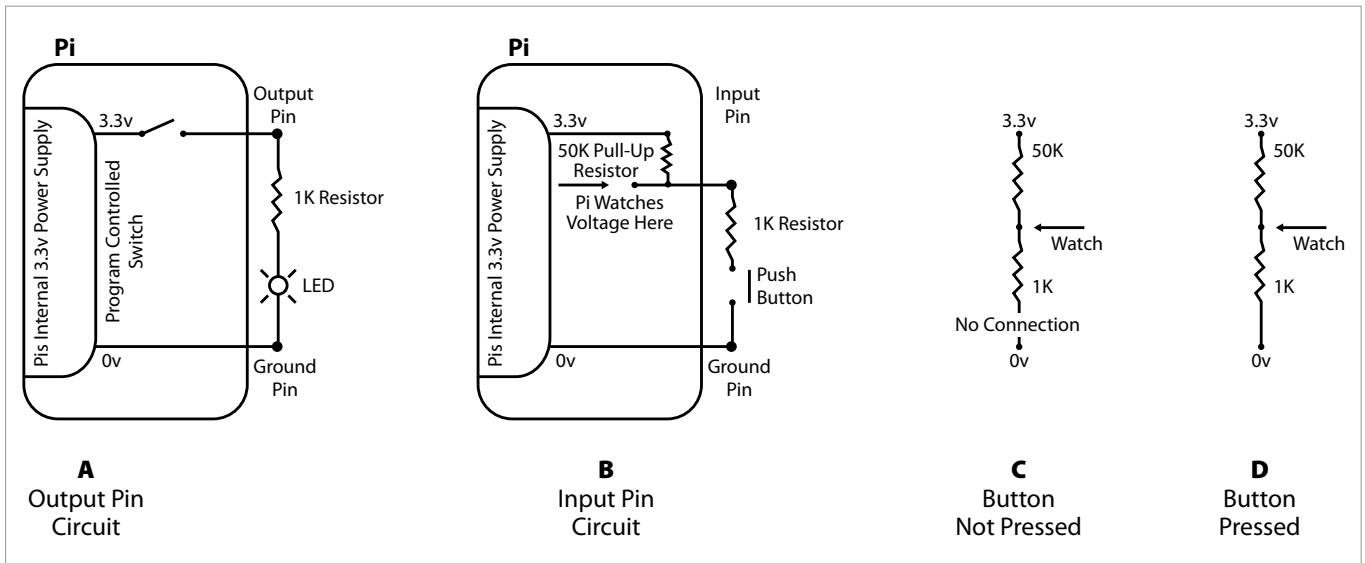


FIGURE 6: Circuit diagrams for both the input and output pins. The “Circuit Diagram” box explains the electrical workings of these circuits.

on the breadboard and just count rows and columns of holes.

Inside the breadboard are lots of long, flat, tiny metal springs that connect some of the holes on the breadboard together electrically – holes that are in a straight line.

When you push a wire into a hole, it connects to the spring underneath that hole inside the breadboard. If some other wire is pushed into another hole along that same spring, the two wires are connected together electrically.

The breadboard has three sections from top to bottom. Each section is electrically separate from the other sections. The top Section – the one farthest away from you – has two rows of holes, one row above the other. On each of these rows, all of the holes are connected together, but the two rows are not connected to each other. This forms two electrical “buses.” An electrical “bus” is just a wire with a lot of places to connect a bunch of other wires together. Each row is usually marked with a different colored stripe – often red, blue, or green. This project will not use this top section.

The middle section is 10 rows high and 60 columns wide. This section has two rows of five vertical holes, one row above the other. In each column of 10 holes, the upper five holes are connected together and the lower five are connected together. The top and bottom halves are not connected together. Each column is electrically separate – not connected to any

other column. Most breadboards have a divider between the upper and lower halves, either a printed line or a line molded into the plastic. All the parts for this project will be in this section.

Circuit Diagram

This project requires two types of circuits. The output circuit (Figure 6A) is very simple. The output pin is either *on* or *off* depending on which way the program wants it. When the pin is *on*, the current flows through the 1K resistor, which limits it to about 3mA. Current then goes through the LED and back to the Pi through the *gnd* pin.

When the program sets up an input pin, it can also tell the Pi to put a “pull-up” resistor into the circuit inside the Pi. Then you put the 1K resistor and the button on the breadboard. All this is shown in Figure 6B. Figures 6C and 6D show details for the right-hand side of Figure 6B.

I am interested in how much voltage is given to the 1K resistor because that tells me how close to ground (0V) the pin is. I calculate it this way using the Pi calculator:

$1000 + 50000$	$1000 \text{ ohms} + 50000 \text{ ohms}$
$= 51000$	The total resistance
$1000 / 51000$	Divide the lower ohms by the total.
$= 0.0196$	The fraction of the lower resistor divided by the total resistance.
$0.0196 * 3.3$	Take this fraction of the total voltage.
$= 0.0647 \text{ volts}$	The Pi “watcher” sees this voltage. (This is really close to zero.)

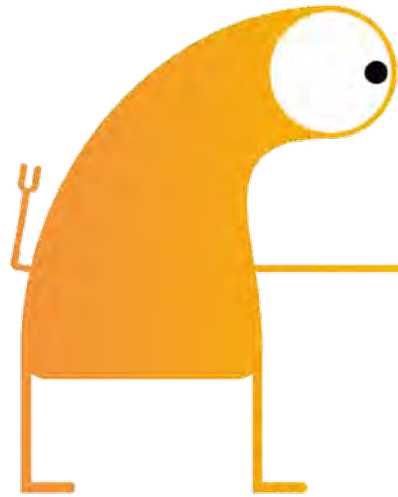
You might ask: “Why do I need the 1K resistor? Why not just connect the input pin directly to the button?” The answer is: “The 1K resistor protects your Pi against a common programming error – setting that pin to be an output.” With the pin set to “output,” pressing the button would short-circuit the Pi’s 3.3V power supply, which would damage your Pi.

Using a 1K resistor, as shown, limits the output current to about 3mA just as it does in LED output circuits.

The bottom section – the one closest to you – looks just the same as the top section: two single-row electrical buses that are not connected to each other. I will use one bus in this bottom section for the project’s electrical “ground.”

Electrical “ground” is the place in the circuit that always has zero volts. It is the place from which we measure the voltage at all other parts of the circuit. “Ground” is normally connected to the negative side of the power supply.

The *GND* pins on the cobbler are all connected to the negative side of the Pi’s power supply inside the Pi; and of course, all of the *GND* pins on the cobbler are connected together. They are all at zero volts.



THE COBBLER

Place the cobbler PC board, pins down, at the left end of the breadboard with the pin labeled *3V3* in the lower set of 5 holes and the end pin labeled *5V* in the upper set of 5 holes – both in the leftmost column of the breadboard. All the other pins will line up over the breadboard columns to the right. I left three rows of holes open below the PC board and two rows of holes open above the PC board.

Wiggle the cobbler board a little bit to make sure that each of the pins is directly over its hole in the breadboard.

Press the pins of the cobbler PC board into the breadboard. Press straight down, and the PC pins should all go in with no problems. I used both thumbs and both forefingers, pressing on the edges of the PC board to spread out the “push.”

THE PUSHBUTTONS

Next, place the three pushbuttons at the right end of the breadboard. On the buttons from Adafruit, the 4 pins on each button form a small rectangle (sized 3 holes by 6 holes). The 6-hole length goes across the center divider, letting the upper pins rest on the second row above the divider and the lower pins rest on the second row below the divider. The right 2 pins will be in the rightmost column of the breadboard and the left pins will be two columns to the left.

Skip two columns to the left and place the next button beside the first – so that you leave two empty columns between the nearest pins of these two buttons.

Skip two more columns to the left and place the third button just like you did the second one.

WHAT YOU’LL NEED

You’ll need the following parts for this project. I’ve suggested some vendors I have used in the past, but you will also find these components with many other electronic distributors. The Raspberry Pi Foundation website [1] lists the official distributors RS [2], Allied Electronics [3], and Element14 [4].

- Raspberry Pi – This project was designed and tested with the latest Pi 2 Model B. I have not tested this project on earlier Pi models. Some of them have only 26 GPIO pins instead of the 40 on the Pi 2 Model B.
- Pi Cobbler – by Adafruit [5]. A Pi cobbler connects all of the GPIO pins on the Pi to a breadboard and provides a set of labels to make it easy to connect the extra parts to the right pin. Get an assembled version of the cobbler so you won’t need to solder it together. Also, get the version of the cobbler for the version of your Pi.
- 10 1K resistors (1000 ohms each) – Jameco [6], Digi-Key [7], Mouser [8]. Use 1/4-watt resistors. Their wires are easy to work with. (See Figure 7.)
- 7 LEDs – Jameco, Digi-Key, or Mouser. My LEDs are red, size T1, standard LEDs. You can use any color. Red, green, yellow are popular. Don’t buy the infra-red ones;

human eyes can’t see infra-red. Most electronic distributors sell LEDs in packages of 10 or 12 or give a price break around that number.

- 3 pushbuttons – Adafruit. “Will it fit into the breadboard?” is the big question here. Holes in the breadboard are spaced 1/10th inch (2.54mm) apart both horizontally and vertically. You want a button whose leads (pins) are spaced to fit a breadboard because many buttons have pins that are too short and stiff to bend.
- 10 long breadboard jumpers – 7 to 8 inches (20cm) long. The ones I used are 26 gauge stranded wire with a reinforced pin at each end. (Figure 8.) This length will let you reach anywhere on the breadboard with wire to spare. From Jameco or Adafruit.
- 11 shorter breadboard jumpers – 1 to 2 inches (5cm) long. Same as the long jumpers, just shorter. From Jameco or Adafruit.
- 1 breadboard – full size. Adafruit or Jameco. Mine is 60 columns wide and has red and blue stripes along both the top and bottom of the board to mark the long rows that are often used for electrical power and ground.

On the buttons from Adafruit, the two pins on each long side of the 3x6 rectangle are connected together inside the button. When the button is pressed, all four pins are connected together.

Press the buttons into the breadboard so that the pins “click” into the breadboard. I used my thumbnails (mine are quite sturdy) to push on the button frame just above and below the button itself.

THE LEDs

Skip the 7 columns to the left of the leftmost button, then insert 2 LEDs into the next four columns below the center divide. Use the second hole below the center divider because you will need the first hole below the divider for a resistor. On each of the LEDs, the long wire goes into the right hole and the short wire goes into the left hole. These first 2 LEDs are for counting the number of outs. The left one is called `led_out1` and the right one is called `led_out2` in the Python program.

Skip two more columns to the left and then insert two more LEDs. These LEDs will be the strike count. The left one is called `led_strike1` and the right one is called `led_strike2` in the Python program.

Skip two more columns to the left and insert three LEDs. These LEDs will be the ball count. The left one is called `led_ball1`, the middle one is `led_ball2`, and the right one is `led_ball3`.

WIRING THE GROUND (GND) CONNECTIONS

On the cobbler are several pins marked *GND*. I will use the *GND* pin on the lower side of the cobbler – 5 pins from the left end of the cobbler. Connect this cobbler *GND* pin to the bus below it (the one nearest you) using a short jumper wire. On my breadboard, this bus is colored blue, so I will call it “the blue bus.”

Connect the right wire of each LED to the same blue bus across the bottom of the breadboard using seven more short jumpers.

Connect the lower left pin of each pushbutton to the same blue bus across the bottom of the breadboard using three more jumpers.

Note: On my breadboard, all of these short jumpers are yellow, solid-wire jumpers from a jumper kit that provides several lengths, each pre-bent to fit across a different number of holes in a breadboard. These kits are available from several parts houses.

I have now connected each of the parts to the blue “ground” bus and have connected the blue bus to the *GND* pin of the Pi. “Ground” is the negative side of the electrical circuit and will always be at zero volts.

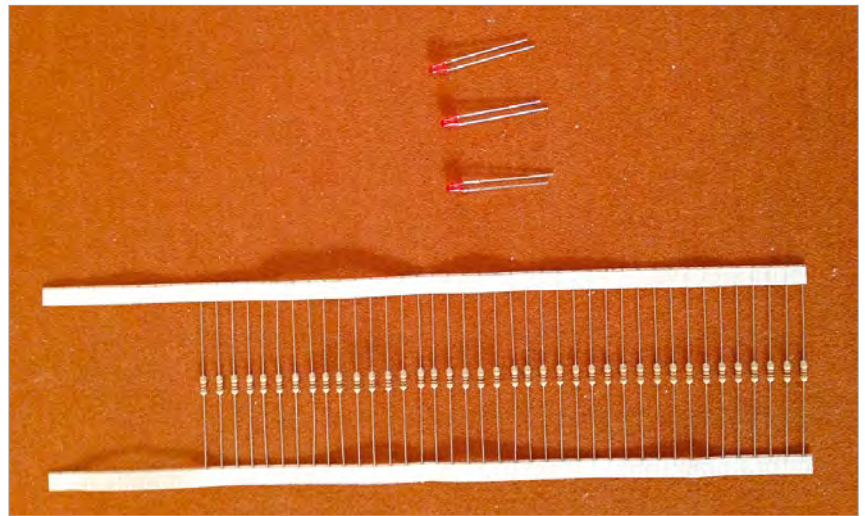


FIGURE 7. A package of 50 resistors minus the ten I used for this project. The brown-black-red colored bands tell that these are 1K resistors (1000 ohms). See the box titled “Resistor Color Codes.”

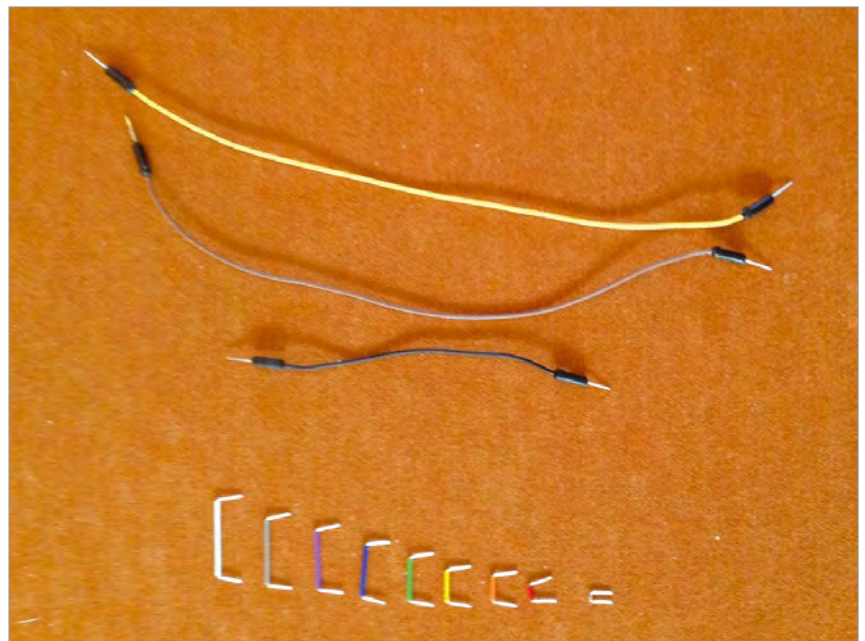


FIGURE 8. You will need the long, flexible jumpers to connect the cobbler pins to the LED and button resistors. The small stiff-wire jumpers on the left come in a kit with several dozen of each size in a plastic case. The colors encode the length as the number of breadboard holes they span from one to nine – using the same colors as the resistor color codes.

Now connect the other side of each part to its GPIO pin through a 1K resistor (1000 ohms). As you install the resistors, leave the top hole of the upper half of each column open so that later you can use a long jumper to connect that end of the resistor to one of the numbered GPIO pins on the Cobbler.

CONNECTING THE RESISTORS TO THE LEDs

The LEDs are in the lower half of the center section. For each LED, use a 1K (1000 ohm) resistor to connect the left wire of the LED to the upper half of the same column.

The wires on a new resistor go straight out from the ends. To bend the wire leads of the resistor at right angles like I did:

1. Pick up a resistor by its body.
2. Hold it between the thumb and forefinger of your left hand with the wires going up and down.
3. Place your right forefinger and thumb lightly on the top and bottom of the knuckles of your left finger and thumb and slowly slide them to the right. When they reach the wires, the wires will bend nicely. Keep sliding to the right until your right finger and thumb slide off the ends of the wires.

Insert one end of a resistor into the

top hole of the lower half of the column that holds the left wire of the leftmost LED. A pair of longnose pliers helps this greatly. Grip the wire about a quarter-inch (a little less than a centimeter) from the end of the wire and push it into the breadboard. Then, do the same to insert the other end of the resistor into the middle hole of the upper half of that same column above the center divider (Figure 9).

Repeat the preceding step to install resistors onto the left wire of each LED.

CONNECTING THE RESISTORS TO THE PUSHBUTTONS

For each pushbutton:

1. Find the upper-left pin of the button and notice that the column to the left of that pin is empty – no wires connected.
2. In this empty column, put one end of a resistor into the hole next to the top row. It is very important to connect this end of the resistor to an empty column.
3. Put the other end of this resistor into the hole just above the upper-right pin of that same button.

Notice that I put the button resistors at a slight angle with the left end one hole higher than the right end (Figure 10). This just gives a little more separation between the wires of the resistors. You do not want the wires to touch each other.

CONNECTING THE GPIO PINS TO THE RESISTORS

Now it's time to wire the GPIO pins on the cobbler to the top ends of the resistors. I will use GPIO pins on the cobbler that are named with numbers. The GPIO pins that are named with letters have additional special abilities that you won't need for this project.

Note: The pin numbers on the cobbler are in a strange order. Just ignore the order, find the pin named with the proper number on the cobbler PC board, and use that pin.

Notice that each GPIO pin will be connected to a resistor and to nothing else. This helps protect the Pi from a short-circuit just in case you run a program that does not match the wiring on the breadboard.

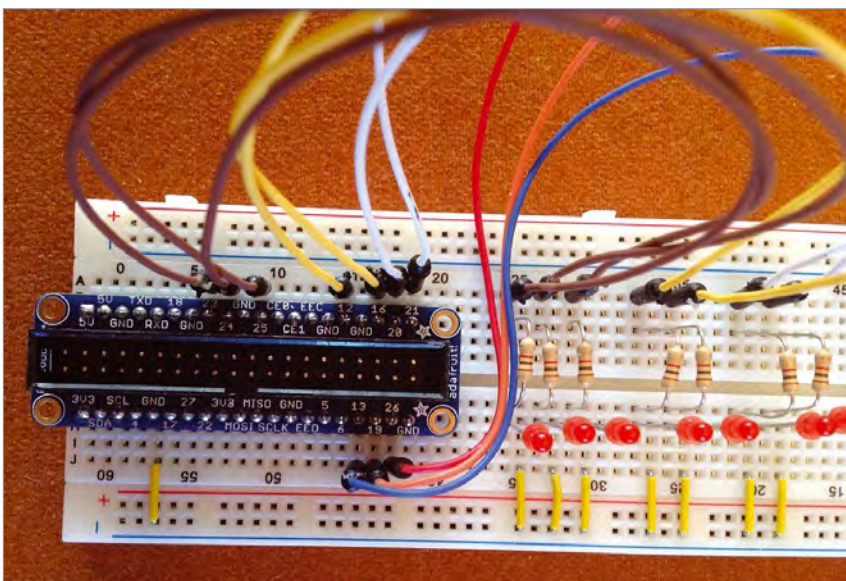


FIGURE 9: Close up of the left end of the completed breadboard. Note how the long jumpers go from the numbered cobbler pins to the upper end of the LED resistors. Also, note that the 3V3 pin on the cobbler is plugged into the left-end hole on the breadboard.

I chose GPIO pins on the upper side of the cobbler for the LEDs.

The leftmost LED is called `led_ball1` in the python program. Connect a jumper wire from cobbler pin 23 to the topmost hole in the column with the resistor for that LED.

The circuit for `led_ball1` now goes from the cobbler pin 23, through the resistor, into the left wire of the LED, out the right wire of the LED, through the short jumper, and into the blue “ground” bus at the bottom of the breadboard.

In the same way, connect the cobbler pins listed below to the resistors of the remaining LEDs from left to right.

- Pin 24 to `led_ball2`
- Pin 25 to `led_ball3`
- Pin 12 to `led_strike1`
- Pin 16 to `led_strike2`
- Pin 20 to `led_out1`
- Pin 21 to `led_out2`

Connect the cobbler pins listed below to the resistors of the pushbuttons from left to right. The jumper wire connects to the left end of the resistor – the end that is not connected to the button. I chose GPIO pins on the lower side of the cobbler for the buttons.

- Pin 26 to `add_ball` – The resistor of the left button
- Pin 19 to `add_strike` – The resistor of the middle button
- Pin 13 to `add_out` – The resistor of the right button

The pushbutton circuits now go from the cobbler pin, through the resistor, into the upper-right pin of the button, out of the lower-left pin of the button, and through the short jumper to the blue ground bus. The breadboard wiring is now complete.

Before you connect the breadboard to the Pi, take the time to trace the circuit for each LED and each button to make sure it is correct. Make sure the left end of each button resistor *is not* plugged into the same column as any button pin.

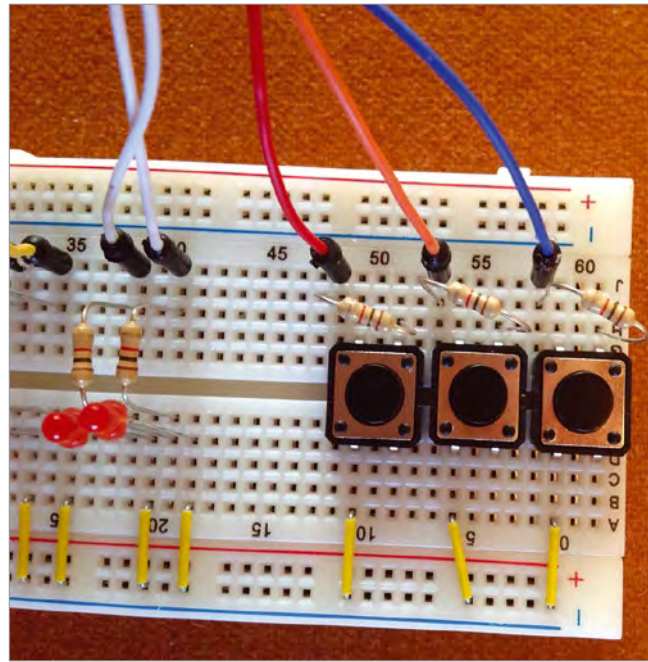


FIGURE 10. Close up of the pushbuttons. Note that, on the pushbutton at the end of the breadboard, the jumper from the cobbler and the left end of the resistor are in a column all by themselves. The left pin of the pushbutton is in the next column over. Each pushbutton jumper and resistor should be in a column by itself; it is just easier to see the wiring for the button on the end of the board.

CONNECTING THE BREADBOARD TO THE PI

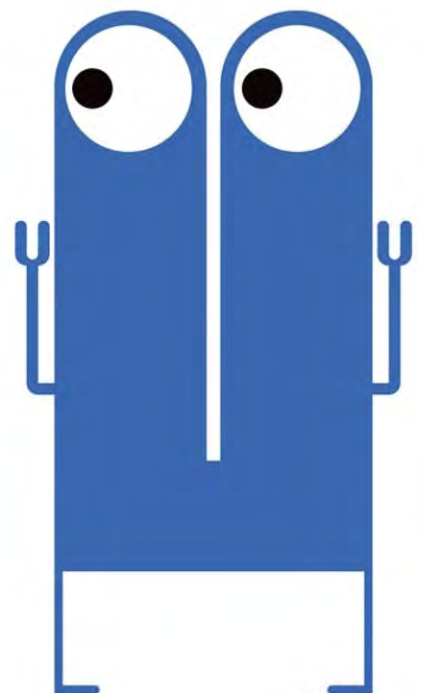
Shutdown the Pi and turn the power off. Next, turn the Pi so that the GPIO pins are in the upper-left corner. The GPIO pins will be on the side of the Pi away from you.

Because the connector on either end of the ribbon cable will connect to the cobbler, you now have a choice. You can place the breadboard closer to you than the Pi or farther away than the Pi.

Choose one, then place the breadboard so that its left end lines up with the left side of the Pi. The distance between them should be a little less than the length of the cable. My cable is about 6 inches (15 centimeters) long.

The wire at one side of the ribbon cable is a different color. My black ribbon cable has a white wire along one side. Place the ribbon cable between the breadboard and the Pi so that the different-colored wire is along the left end of both the breadboard and the Pi.

Lightly place the end of the cable closest to the cobbler into the socket on the cobbler. It will only fit one way. Using



both thumbs, press the cable header into the cobbler socket.

Connect the other end of the ribbon cable to the Pi.

1. Make sure that the different colored wire is at the left end of the Pi board.
2. Place the ribbon cable connector lightly over the GPIO pins on the Pi. The Pi has no “keyed socket” to force the cable to go only the right way (like the cobbler does), so you have to be careful. Compare your arrangement with Figure 11, which shows my breadboard and Pi connected together.
3. Make sure that the ribbon cable connector covers all of the of GPIO pins on the Pi and covers both rows, and that there are no Pi pins showing at the ends of the connector.
4. Press straight down on both ends of the ribbon cable connector.

Power-on the Pi and watch the Pi and the breadboard carefully. The Pi should boot up normally and nothing should happen on the breadboard.

If anything else happens or if the Pi does not boot up normally, disconnect the power immediately and find out what is wrong.

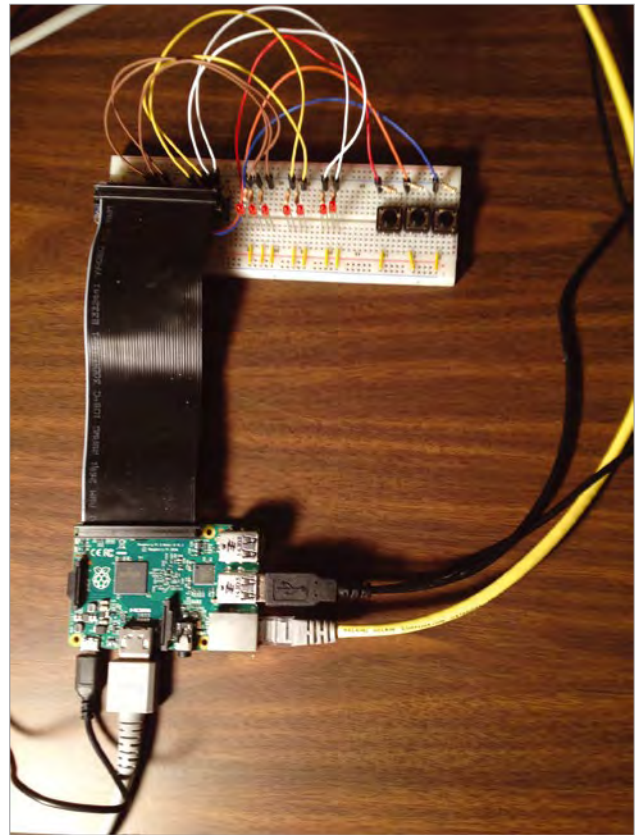


FIGURE 11. The completed breadboard connected to the Pi with the ribbon cable that came with the cobbler. Note that the white wire of the black ribbon cable is along the left end of the breadboard (which should be the only way it will fit into the cobbler socket). Note also that the white wire is along the left edge of the Pi.

```

pi@raspberrypi: ~/bill/pi_for_kids
File Edit Tabs Help
pi@raspberrypi ~/bill/pi_for_kids $ sudo python baseball.py
Imported: RPi.GPIO as GPIO
GPIO.VERSION 0.5.11
How to play:
On the breadboard:
Press the left button when the umpire calls a 'ball'
Press the middle button when the umpire calls a 'strike'
Press the right button when a player makes an out

'ball' LEDs will blink when the batter walks
'strike' LEDs will blink when the batter strikes out
'out' LEDs will blink when this side takes the field

To clear blinking 'ball' LEDs, press the 'ball' button
To clear blinking 'strike' LEDs, press the 'strike' button
To clear blinking 'out' LEDs, press the 'out' button

On the keyboard:
To exit the program, Press ctrl-C

```

FIGURE 12. A screenshot of `baseball.py` running the breadboard. Nothing changes on the screen while you operate the pushbuttons on the breadboard. When you press `ctrl-c` (hold the ‘ctrl’ key down and press ‘c’ with another finger) the program will print a one-line exiting message, reset the GPIO pins to default (the LEDs will turn off), then exit.

BASEBALL.PY

The last piece of the puzzle is the program called `baseball.py` (Listing 1) that runs on the Raspberry Pi. `baseball.py` listens for input from the pushbuttons and controls the LED lights on the scoreboard. A file with the complete source code for `baseball.py` is available for download [9] if you would rather not type it all in.

You'll need some background in Python to fully understand what the code in Listing 1 is doing, but, as you can see, the code is well documented, so you should be able to gain some insights into how the program works by reading through it.

The box titled "Notes on `baseball.py`" explains how the GPIO functions work. The box also explains three Python features in the program that you might not have seen yet.

Login to the Pi and change directory (`cd`) to the directory that the program is in.

Hint: If you type "clear" (no quotes) before starting the program, you will have a cleaner printout on the screen.

```
clear
```

Type the following on the keyboard:

```
sudo python baseball.py
```

The program will print a few lines onto the screen and then say that the baseball scoreboard is ready (Figure 12).

On the breadboard, press and release the left button. Don't release it too quickly. Hold the button pressed for just a moment before you release it. The `led_ball1` LED should light.

Press and release the same button, and the `led_ball2` LED should light.

Press and release the same button a third time and the `led_ball3` LED should light.

Press and release the same button the fourth time, and all of the `led_ball` LEDs should start blinking to say that the pitcher walked the batter.

Press and release the same button one more time to clear the ball count back to zero (all off).

Move to the middle button, and the `strikes` LEDs should work the same way as the ball LEDs. The lights should

LISTING 1: `baseball.py`

```
001 # baseball.py displays the status of a baseball or softball inning
002 # using LEDs on a breadboard just as the lights on the old-time
003 # "scoreboard" at the game used to do.
004 #
005 # balls      3 LEDs -- first 3 balls, blinks for a walk
006 # strikes   2 LEDs -- first two strikes, blinks for strikeout
007 # outs      2 LEDs -- first 2 outs, blinks for the third out
008 # -----
009 #
010 #
011 # -----
012 # Import the time module -- so the program can sleep occasionally
013 # -----
014 import time
015 #
016 # -----
017 # Import and Initialize the GPIO module
018 # -----
019 import RPi.GPIO as GPIO
020 print "Imported: RPi.GPIO as GPIO"
021 print "GPIO.VERSION", GPIO.VERSION
022 #
023 GPIO.setmode(GPIO.BCM)
024 GPIO.setwarnings(True)
025 #
026 # -----
027 # Choose which GPIO pin increments the balls, strikes, outs
028 # Note: This must match the wiring on the breadboard
029 # -----
030 add_ball = 26
031 add_strike = 19
032 add_out = 13
033 #
034 # -----
035 # Choose which GPIO pin each LED is attached-to
036 # Note: This must match the wiring on the breadboard
037 # -----
038 led_ball1 = 23
039 led_ball2 = 24
040 led_ball3 = 25
041 #
042 led_strike1 = 12
043 led_strike2 = 16
044 #
045 led_out1 = 20
046 led_out2 = 21
047 #
048 # -----
049 # Make lists of GPIO pins used for balls, strikes, outs, buttons
050 # -----
051 pinlist_balls = [led_ball1, led_ball2, led_ball3]
052 pinlist_strikes = [led_strike1, led_strike2]
053 pinlist_outs = [led_out1, led_out2]
054 pinlist_inputs = [add_ball, add_strike, add_out]
055 #
056 # -----
057 # Tell GPIO about the input pins (Connected to buttons)
```

blink after the third strike to say *strike out*. Push the middle button again to clear the strikeout. This will also add one out to the *out* LEDs.

Move to the right button and repeat the test for the outs. At the third out, the *out* LEDs will blink to say “It is the other team’s turn to bat.” Press the

NOTES ON BASEBALL.PY

Some additional commentary on `baseball.py` will be useful for those who wish to study the code. Following are some notes on the GPIO functions used in this program and on three Python features you might not have seen before: lists, the `+=` operator, and `try`:

```
GPIO.setmode(GPIO.BCM)
```

This line tells the GPIO software that this program will use the pin numbers that match the labels on the cobbler. These numbers are the same numbers used by the Broadcom processor chip in the Pi.

```
GPIO.setwarnings(True)
```

This line tells the GPIO software to check the status of the GPIO pins and print a warning about any pin that is not in the default state – the state that power-on sets the pins. The default state is: “pin is input, with none of its pull-up or pull-down resistors active.”

The warning says that the last program to use the GPIO pins did not reset them to default. Just press `Ctrl + C` to reset the pins to default values and then re-run `baseball.py`.

Choose which GPIO Pin?

These two sections name all the GPIO pins that the breadboard is going to use. The pin numbers must match the connections on the breadboard.

```
GPIO.setup
```

These lines tell the Pi to use each of the GPIO pins as either an input or output. Here they also set input pull-up resistors to active and set the output pins to zero volts.

```
GPIO.input(add_ball)
```

`GPIO.input(<pin number>)` reads the voltage on the named pin. If the voltage is close to 3.3, `GPIO.input()` returns 1. If the voltage is close to 0.0, `GPIO.input()` returns 0.

```
GPIO.output(led_ball1, 1)
```

`GPIO.output(<pin number>, <value>)` sets the voltage on the named pin according to `<value>`. If `<pin number>` is a list of pin numbers, set all pins in the list according to `<value>`. If the `<value>` is 1, set the pin voltage to 3.3 volts. If the `<value>` is 0, set the pin voltage to 0.0 volts.

```
GPIO.cleanup()
```

This line tells the Pi to reset the GPIO pins that this program used. The pins are reset to be input pins with no pull-up or pull-down resistors.

Python Lists

Make lists ...

In python, a list is just a list of things. You put variables into the list by putting the variable names between square brackets `[]` and separated by commas. Later in the program, the list passes to `GPIO.output()`, which will set all of the pins in the list to the same value.

```
blink_list = []
```

Scan on down the program to about line 125 and find this line. This line clears the list and makes it empty. After clearing, The list still exists – just has nothing in it.

```
if balls == 4: blink_list += pinlist_balls
```

A few lines further down, this line, and a couple more lines like it, copy the contents of one list into another. In this case, the program copies the list `pinlist_balls` into `blink_list`. If the receiving list already has things in it, the new things just go onto the end of the list. The things in the sending list are not removed from that list. After the copy, these things are on both lists.

The Python += Operator

`baseball.py` uses the `+=` operator in two ways:

1. to add 1 to a count
Example: `balls += 1`
2. to copy all the items in one list into another list.
Example: `blink_list += pinlist_strikes`

Exceptions

```
try: except: finally:
```

These lines are a standard way to handle certain interrupts (like you pressing `Ctrl + C` on the keyboard) or exceptions (errors) that happen while the program is running. If the program is running inside the `try` when the interrupt or exception happens, Python stops running the code inside the `try` and jumps to the code inside the `except`. When the `except` (exception) code is finished, the `try` is also complete.

When the `try` is complete, if it has a `finally`, any code inside its `finally` is run, whether or not there has been an exception within the `try`. It is possible for the `try` to complete with no exception by simply exiting the end of the `try` code.

In `baseball.py`, the `while True:` loop is an infinite loop, so the `try` will never complete without an exception. When you press `Ctrl + C` on the keyboard, you provide the exception that allows the program to clean up and exit.

add_out button again to clear all of the LEDs for the other team's turn at bat.

Clearing either a walk (blinking ball LEDs) or a strikeout (blinking strike LEDs) will clear all ball and strike LEDs to get ready for the next batter.

Congratulations on a successful breadboard project. Play Ball! *

INFO

[1] Raspberry Pi Official Website:
<https://www.raspberrypi.org/>

[2] RS: <http://uk.rs-online.com/web/generalDisplay.html?id=raspberrypi>

[3] Allied Electronics:
<http://www.alliedelec.com/>

[4] Element14:
<http://www.element14.com/community/community/raspberrypi>

[5] Adafruit: <https://www.adafruit.com/>

[6] Jameco: <http://www.jameco.com/>

[7] Digi-Key: <http://www.digikey.com/>

[8] Mouser: <http://www.mouser.com/>

[9] Code for this article:
ftp://ftp.linux-magazine.com/pub/listings/magazine/RPi_Adventures

LISTING 1. baseball.py (continued)

```
058 # Activate the pull-up resistors.
059 # When the button is pressed, the input on the pin will be zero.
060 # -----
061 GPIO.setup(add_ball , GPIO.IN, GPIO.PUD_UP)
062 GPIO.setup(add_strike, GPIO.IN, GPIO.PUD_UP)
063 GPIO.setup(add_out , GPIO.IN, GPIO.PUD_UP)
064
065 button_pressed = 0
066
067 # -----
068 # Tell GPIO about the output pins (Connected to LEDs)
069 # Start with all LEDs off.
070 # -----
071 GPIO.setup(led_ball1 , GPIO.OUT, initial=GPIO.LOW)
072 GPIO.setup(led_ball2 , GPIO.OUT, initial=GPIO.LOW)
073 GPIO.setup(led_ball3 , GPIO.OUT, initial=GPIO.LOW)
074 GPIO.setup(led_strike1, GPIO.OUT, initial=GPIO.LOW)
075 GPIO.setup(led_strike2, GPIO.OUT, initial=GPIO.LOW)
076 GPIO.setup(led_out1 , GPIO.OUT, initial=GPIO.LOW)
077 GPIO.setup(led_out2 , GPIO.OUT, initial=GPIO.LOW)
078
079
080 # -----
081 # Clear counts of balls, strikes, outs for the first inning
082 # -----
083 balls = 0
084 strikes = 0
085 outs = 0
086
087 # -----
088 # On the screen, tell how to play
089 # -----
090 print "How to play:"
091 print "On the breadboard:"
092 print " Press the left button when the umpire calls a 'ball'"
093 print " Press the middle button when the umpire calls a 'strike'"
094 print " Press the right button when a player makes an out"
095 print " "
096 print " 'ball' LEDs will blink when the batter walks"
097 print " 'strike' LEDs will blink when the batter strikes out"
098 print " 'out' LEDs will blink when this side takes the field "
099 print " "
100 print " To clear blinking 'ball' LEDs, press the 'ball' button"
101 print " To clear blinking 'strike' LEDs, press the 'strike' button"
102 print " To clear blinking 'out' LEDs, press the 'out' button"
103 print " "
104 print "On the keyboard: "
105 print "To exit the program, Press ctrl-c"
106
107
108 # -----
109 # Clear the blink_list so that no LEDs blink yet
110 #
111 # 'try', 'except', 'finally' catches ctrl-c from the keyboard
112 # and stops the program.
113 #
114 # Loop while the ball game is being played
115 # Read the input pins to see if any has been pressed
116 #
117 # If an input is pressed, add 1 to its counter
118 # (balls, strikes, or outs)
119 #
120 # Turn-on the number of LEDs for the proper number of
121 # balls, strikes, outs or add LEDs to the blink_list
122 #
123 # If blink_list is not empty, blink the LEDs in the list
124 # -----
125 blink_list = []
126
127 try:
128     while True:
129         # -----
130         # Read the buttons
131         #
132         # If a button is pressed,
133         # Wait 1/20th of a second to make sure it was pressed
134         # If it is still pressed,
135         # Then wait for it to be released,
136         # and add 1 to the count
137         # Else (it was not pressed for 1/20th of a second)
138         # Then we want to ignore it.
139         #
140         # When a button is being pressed and is very near the bottom
141         # it sometimes bounces against the bottom -- too fast for
```

LISTING 1. baseball.py (continued)

```

142     # a human finger to feel-- but the computer will notice.
143     # If we do not "debounce" the button by waiting a short time,
144     # we might add several counts during a single press instead
145     # of adding only the single count that we expected.
146     # -----
147     if GPIO.input(add_ball) == button_pressed:
148         time.sleep(.05)
149         if GPIO.input(add_ball) == button_pressed:
150             while GPIO.input(add_ball) == button_pressed:
151                 time.sleep(.02)
152                 balls += 1
153
154     if GPIO.input(add_strike) == button_pressed:
155         time.sleep(.05)
156         if GPIO.input(add_strike) == button_pressed:
157             while GPIO.input(add_strike) == button_pressed:
158                 time.sleep(.02)
159                 strikes += 1
160
161     if GPIO.input(add_out) == button_pressed:
162         time.sleep(.05)
163         if GPIO.input(add_out) == button_pressed:
164             while GPIO.input(add_out) == button_pressed:
165                 time.sleep(.02)
166                 outs += 1
167
168     # -----
169     # Turn the LEDs on or off to show the count
170     # of balls, strikes, outs
171     #
172     #   If (count is zero) turn off all these LEDs
173     #
174     #   If (have an LED for this count) turn it on
175     #
176     #   If (have reached max count)
177     #       Then add all these LEDs to the blink list
178     #
179     #   If (LEDs are blinking and button was pressed)
180     #       Then clear the count to zero
181     #       (LEDs will turn off next time around the loop)
182     #
183     # Notes:
184     #   Clearing the ball count also clears the strike count.
185     #   Clearing the strike count also clears the ball count.
186     #   Clearing the strike count adds one to the out count.
187     #   Clearing the out count resets for a new inning
188     # -----
189     if balls == 0: GPIO.output(pinlist_balls, 0)
190     if balls == 1: GPIO.output(led_ball1, 1)
191     if balls == 2: GPIO.output(led_ball2, 1)
192     if balls == 3: GPIO.output(led_ball3, 1)
193     if balls == 4: blink_list += pinlist_balls
194     if balls == 5:
195         balls = 0
196         strikes = 0
197
198     if strikes == 0: GPIO.output(pinlist_strikes, 0)

```

LISTING 1. baseball.py (continued)

```
199     if strikes == 1: GPIO.output(led_strike1, 1)
200     if strikes == 2: GPIO.output(led_strike2, 1)
201     if strikes == 3: blink_list += pinlist_strikes
202     if strikes == 4:
203         balls = 0
204         strikes = 0
205         if outs < 3:
206             outs += 1
207
208
209     if outs == 0: GPIO.output(pinlist_outs, 0)
210     if outs == 1: GPIO.output(led_out1, 1)
211     if outs == 2: GPIO.output(led_out2, 1)
212     if outs == 3: blink_list += pinlist_outs
213     if outs == 4:
214         # -----
215         # Reset for a new inning
216         # -----
217         balls = 0
218         strikes = 0
219         outs = 0
220
221     # -----
222     # Handle any blink requests
223     #
224     # If blink_list is not empty:
225     #   Turn off all LEDs in the blink_list
226     #   Wait 1/5 of a second
227     #   Turn on all LEDs in the blink_list
228     #   Clear the blink list to empty
229     # Else
230     #   sleep to give the CPU some time to handle other programs
231     # -----
232     if len(blink_list) > 0:
233         GPIO.output(blink_list, 0)
234         time.sleep(.2)
235         GPIO.output(blink_list, 1)
236         time.sleep(.2)
237         blink_list = []
238     else:
239         time.sleep(.05)
240
241 except KeyboardInterrupt:
242     print "ctrl-c pressed -- Cleaning-up and exiting"
243
244 finally:
245     # -----
246     # When the above loop exits:
247     #   Reset the GPIO pins that this program used
248     #   Pins set to be inputs
249     #   with no active pull-up, pull-down resistors
250     # -----
251     GPIO.cleanup()
252
253     # -----
254     # Then fall out the end of the program to exit.
255     # -----
```

Drawing and Animating a Hungry Cat Racer

GO,
CAT,
GO!

WE SHOW HOW TO USE SCRATCH'S
BUILT-IN GRAPHICS EDITOR TO
CREATE ANIMATIONS FOR A
RACING GAME.
BY MICHAEL BADGER

USING SCRATCH'S built-in graphics editor is a perfect way to introduce Scratch to novice programmers of any age. Experienced Scratchers can create detailed animations that enhance any project. In the Hungry Cat Racer game, I'll show you how to draw multiple backgrounds and edit sprite costumes that will become the basis of the game's animated effects.

When I introduce Scratch to young programmers, the built-in graphics editor is often a perfect place to start; students can take an existing project and immediately customize it without any initial programming while still engaging in something creative. When customizing projects, the students invariably start drawing their own characters, and then,

of course, they want to program those characters to do something.

The setup for this project will involve the Paint Editor, and although I won't go over each option in a tool-by-tool review, I will show you the possibilities. In other words, a five-year-old can hack together a ninja character (that might look like blue spaghetti to you) or a more practiced artist can draw polished images.

GAME PLAY

The player (a.k.a. the Scratch Cat) drives along a road and tries to eat as many mice or other rewards as possible while avoiding dogs and other obstacles. The mice and dogs will appear in the road at random locations. The player

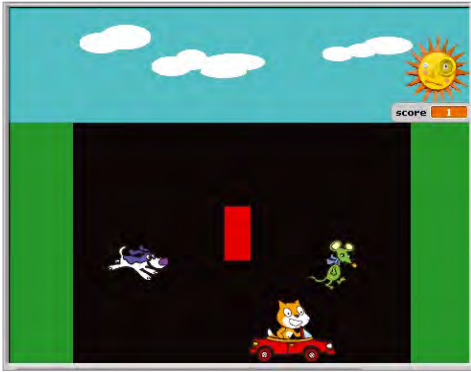


FIGURE 1: Hungry Cat game play. Use the cat to eat the mice and avoid the dogs.

can move left and right using the arrow keys to avoid the dog and eat the mice.

MOVING BACKGROUNDS

For this game, I created an animated background that simulates driving along a road. Figure 1 shows the background layout complete with a blue sky, a two-lane road, and a border. To draw this background, click on the *Stage* icon that's listed below the stage and select the *Backgrounds* tab. Click the *Edit* button next to the default white background. If you are using this tutorial with Scratch 2.0 outside of your Raspberry Pi, backgrounds have been renamed "backdrops."

If you mouse over the tool icons on the left side of the Paint Editor, a tool tip will display the name of the tool. As you can see in Figure 1, the background relies heavily on colorful rectangles. If you make a mistake while editing, you can use the *Undo* button or press **Ctrl+Z** to undo your changes. If something irreversible happens, cancel the Paint Editor and start again.

To aid in the drawing process and ensure the images are drawn big enough, I recommend using the zoom controls to zoom out as far as possible. Select the *Fill tool* and then select black from the color picker. Click on the stage, and the white background will be filled with black.

To draw the sky, select the *Rectangle tool*, make sure the solid square option is selected, and then choose blue from the color picker. On the stage, draw a rectangle in the top area of the stage by clicking and dragging the mouse. As you draw, a blue rectangle will appear. Release the mouse button to create the rectangle. Some trial and error may be needed to get the rectangle in the right

spot and to take up the entire area at the top of the image. You won't be able to drag the rectangle around the stage to position it, so keep undoing and redrawing until you get something you like. Then, use a green rectangle to draw a border on each side of the stage, using Figure 1 as a visual reference.

Voilà, you have a road without a center line. At this point, you could use the rectangle tool to draw a vertical center line in the middle of the road, but I found it easier to draw a new image and then import it into the background. This will help ensure a uniform design across all the backgrounds you will create. An inconsistent design from one background to another will be easily noticed.

As you see in my example, I've added some clouds to the sky, which you can do, too. Use the *Ellipse tool* to draw overlapping white ellipses. Click *OK* to save the background and exit Paint Editor.

DRAW, EXPORT, AND IMPORT SPRITES

To get the center line, click the *Paint new sprite* icon directly beneath the stage to open a blank Paint Editor canvas. Draw a vertical rectangle in the size and color you want. I used red.

After you save the new sprite, right-click on the sprite's icon and choose *export this sprite*. Save it to your filesystem. Now edit the stage background and, in the Paint Editor, click *Import*. Select the sprite you just saved.

When the Paint Editor imports the sprite, you will see a hand icon to signify that you can move the image around the stage to position it as needed. After you set the position of the imported sprite by clicking somewhere on the canvas or saving the image, you will not be able to reposition the imported image.

DUPLICATING AND EDITING BACKGROUNDS

Assuming you have a background that you like, I recommend making a backup copy. From the *Backgrounds* tab, you can right-click and export the image or you can drag the background and drop it onto one of the icons in the sprite list, which will add the image as a costume for that sprite.

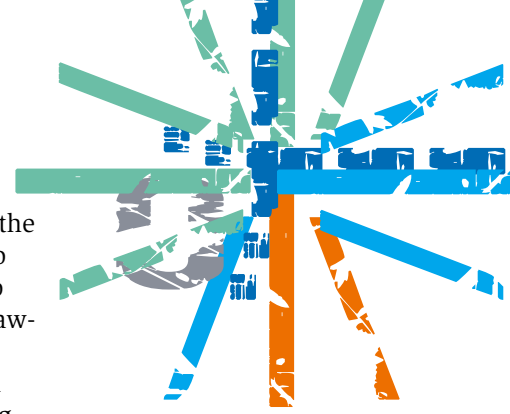


FIGURE 2: Thumbnail view of stage backgrounds showing how each image is changed.

By the way, if you drag a sprite's costume onto the stage icon, you'll add a background to the stage. I didn't use this method to import the road's center line because you cannot reposition the sprite with drag and drop; therefore, the method I showed you provides as much flexibility as possible.

Now, I'll get back to editing the backgrounds. A good starting point for the animation seems to be three backgrounds. If you copy the existing background two times, you can start to edit the road's center line.

Edit the first background and use a black rectangle to draw over the center line, leaving a small portion visible at the top of the road. Then repeat this process for the other backgrounds, and for each background, move the position of the center line down the image. Figure 2 shows the thumbnail view of my backgrounds for a reference. Note that Figure 2 also shows my starting image with a solid red line; it's there only as a reference to show the progression of the edits.

Figure 3 shows the actual stage script to animate the images to give the effect of driving down the road. It continually loops through costumes using the `next costume` block. Clicking the green flag will quickly cycle through each background in the order they are listed.

If you want to slow down the animation, add a `wait ()` to the `forever` loop in Figure 3.

CREATING AND ANIMATING THE PLAYER

Figure 4 shows the scripts to control the player. The smaller stack of blocks are initialization values to ensure the game always starts with the sprite in the right location, direction, size, and so forth.

The other `when green flag clicked` script shows the movement controls. Using the arrow keys is a fairly common way to move a sprite in Scratch, but this script actually detects the side of the road and inverts the sprite's movement when the player tries to go off the road. This prevents the cat from straying off the road. And, as you see in the script, I'm using a costume to show an animation when the cat touches the green border. I'll leave it to you to create that effect.

I'm detecting the side of the road to restrict the player's movement and create a more challenging game play. If you were creating a platform game, however, this script provides one way to detect a wall. It prevents the sprite from passing through the detected boundary by moving in the opposite direction when it detects the wall or boundary.

The `when I receive (game over)` script is running another animation using a combination of a costume and the `change () effect by ()` blocks. In some cases, the pixelate effect or the whirl effect may adequately cover your needs. I felt a compelling need to start with an exploding sprite, which I created by editing a costume with the `Select tool`. By selecting pieces of the costume and moving it outward from the costume's center, it creates a rudimentary exploding effect. You could, of course, draw several costumes to illustrate this. I chose to draw one explosion costume and then use the `change () effect by ()` blocks to further distort the image when the dog "catches" the cat.

In Figure 1, you can see that the cat is sitting in a car. This costume was created by importing and positioning a car into each of the costumes used to animate the player sprite. Like most image editing software, the Paint Editor has an option to flip the image horizontally and vertically to change the orientation of the costume. This also applies to an image that you first import into the Paint Editor. So, if you follow my lead and import a car into the cat, you can change the orientation right away before you commit the change because, as with the center line, you will not be able to make position changes after you save the change.

USING SIZE TO CHANGE PERSPECTIVES ON THE MOUSE

Figure 5 shows the scripts to animate the mouse. Note that the sprite starts in a random x position on the stage and at the top of the road. It is not constrained to the width of the road like the cat's movement is; I did restrict the placement of the dog to the width of the road. That's a decision I made to make the game more challenging.

The effect of the mouse as it moves from the top of the stage to the bottom is important here. After the sprite picks a

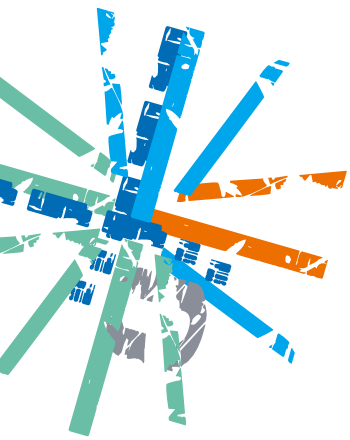


FIGURE 3: A stage script to initialize some game variables and to animate the backgrounds.

location to show itself, the size is reduced using the `set size to ()` block. As the mouse falls, the `turn () degrees` block rotates the sprite while increasing the size of the mouse with the `change size by ()` block.

The `set size by ()` changes the size relative to the current size. For example, in relation to the sprite's starting size, a 0.5% value, would make the sprite half the size, whereas using 200% would make it twice as big. Subsequently setting the size to the same percentage will not change the size of the sprite, because 200% of the original sprite size is always the same.

The `change size by ()` block in comparison grows the sprite by the specified amount based on the current size, so that each time you run the `change size by ()` block, it will grow or shrink the sprite by the size specified. This allows the mouse to get bigger each time the block runs.

The result of the animation is that the mouse starts small at the top of the screen (beginning of the road), and as the mouse tumbles closer to the cat, it gets bigger – to mimic real-life perspective.

ENDING THE GAME

The game ends when the dog eats the cat or, in this context, when the dog and the cat touch. I won't show the scripts for the dog, because positioning and moving the dog along the road can be very similar to the cat's movement already discussed. You can find the scripts if you download the project [1].

CUSTOMIZING GAME PLAY

As the game stands, there is one reward and one obstacle. One way to create challenging game play is to add more rewards and obstacles. In Scratch 1.4, the way to do that is to duplicate the sprites, which will duplicate the scripts for each sprite, too. Thus, it makes sense to create, play, troubleshoot, and update your scripts before you duplicate them.

I drew inspiration from a Scratch 2.0 project that handles the generation of the sprites using a clone feature, which is not available in Scratch 1.4. If you have access to a non-Pi computer using Flash, you can check out the project online [2].

Speeding up the rate of the rewards and obstacles based on either time or score will also create a more challenging environment. If the animations are running more slowly than you might like on the Raspberry Pi, you can try increasing the number of steps a sprite moves.

Of course, drawing the individual frames of an animation opens the opportunity for an infinite amount of customizations and personality. Happy Scratchin'. *

INFO

- [1] Scratch 1.4 project: <http://www.scratchguide.com/tag/raspberry-pi-geek/>
- [2] Scratch 2.0 project: <https://scratch.mit.edu/projects/26168007/>

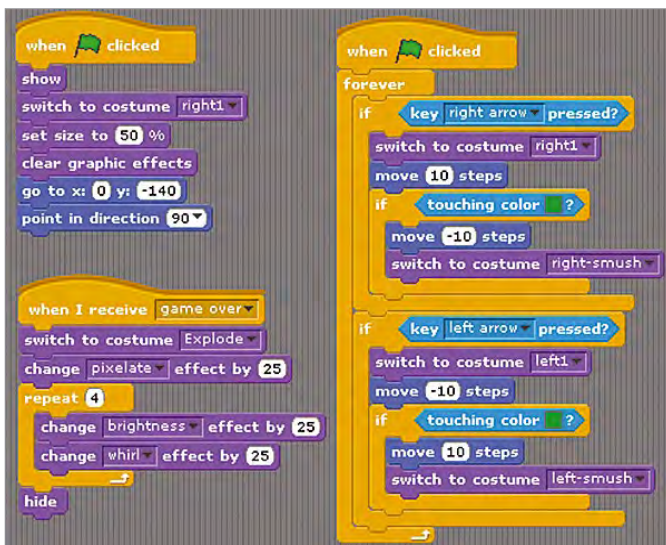
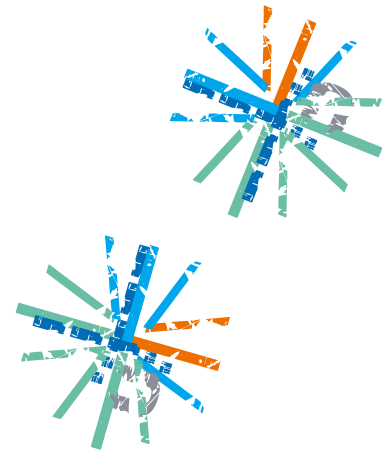


FIGURE 4: Scripts to control and animate the player.

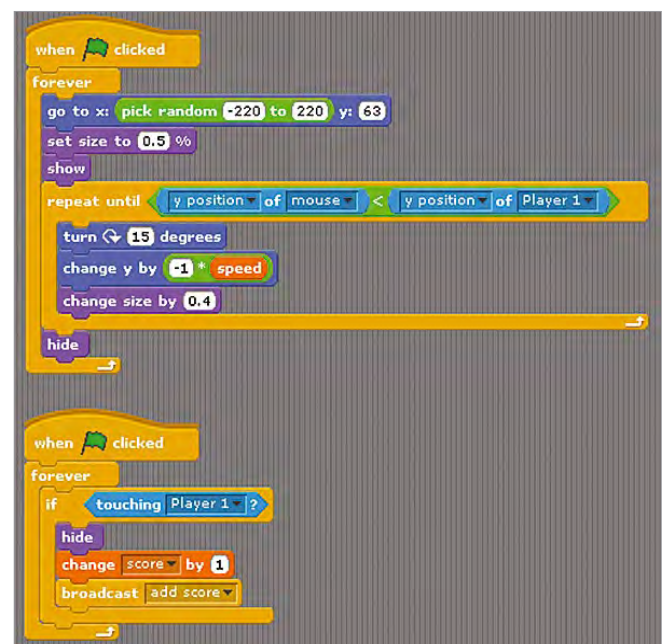
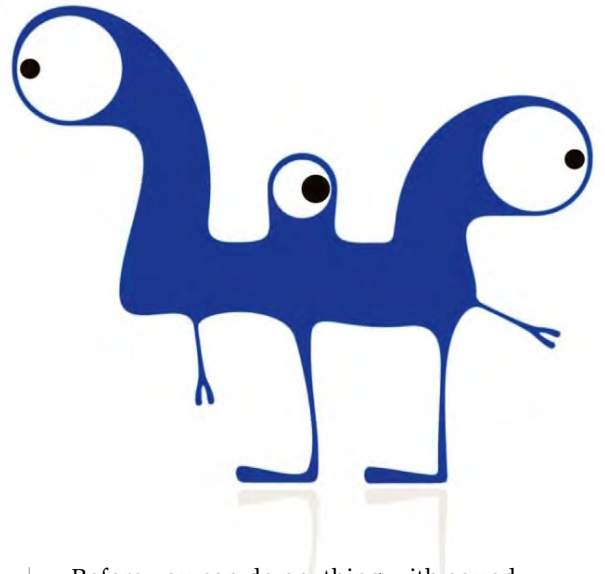


FIGURE 5: Scripts to display and move the mouse sprite.

MAKING MUSIC WITH SONIC PI

Music Box



MAKE MUSIC AND EXPLORE
THE WORLD OF DIGITAL
SOUND WITH SONIC PI.
BY DAVE PHILLIPS

SONIC PI [1] is a computer programming environment for music and sound composition, with features that also let you use it as a realtime performance instrument. The Sonic Pi project was originally intended as a tool for teaching music in schools. The beauty of Sonic Pi is that you can learn about programming and computer science while also learning basic concepts of digital sound and music production. Sonic Pi's live-coding capability can turn a programmer into a musical performer exactly like a player in a band.

FROM 0 TO SOUND IN THREE STEPS

I'll start by making some music with the Raspberry Pi board and the Sonic Pi software. First, you'll need to do a system check to be sure you have the required pieces. (See the box titled "Parts List.")

Before you can do anything with sound on the Raspberry Pi, you'll need to set the output volume level. In the Raspbian Main window, click on the *Menu* button and select *Preferences | Audio Device Settings*. The Raspberry Pi is a playback-only device, so there's only a single output volume control (Figure 1). For your first session, set the volume level to 60. Be careful with your first settings; you don't want to blow out your speakers or

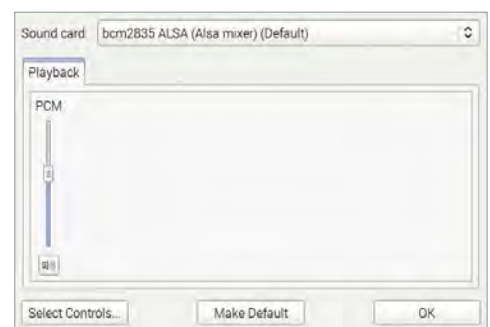


FIGURE 1: Volume control for a Raspberry Pi system running Raspbian.

Lead Image © Alexandr Alecbiev, 123RF.com

PARTS LIST

- A Raspberry Pi board
- Peripherals: power supply, monitor, keyboard, mouse
- Internet connection (WiFi or wired)
- A sound system (headphones, powered speakers, HDMI audio/video)
- The Sonic Pi software (which comes preinstalled on most Raspbian systems)

your eardrums. You can always raise the level later. If the volume control doesn't appear in the *AudioDeviceSettings* dialog, click the *Select Controls* button (Figure 1) and check the box for PCM.

Take a breath, you're ready to dive. The next steps provide a first demonstration of what you can do with Raspberry Pi and Sonic Pi.

To start Sonic Pi, click the top *Menu* button and choose *Programming | Sonic Pi*. The *Help* panel at the bottom-left corner of the Sonic Pi window includes a subpanel with three tabs. Click on the *Examples* tab, scroll to the item labeled *[Magician] Compus Beats*, then double-click it to load it into the code viewer (the other subpanel in the *Help* section). At this point, your display should look like Figure 2.

Now you need to copy the code from the viewer into a workspace. Select the text of the code by clicking and dragging the mouse from the first line through the last (Figure 3). Move the mouse pointer to an empty workspace, then middle-

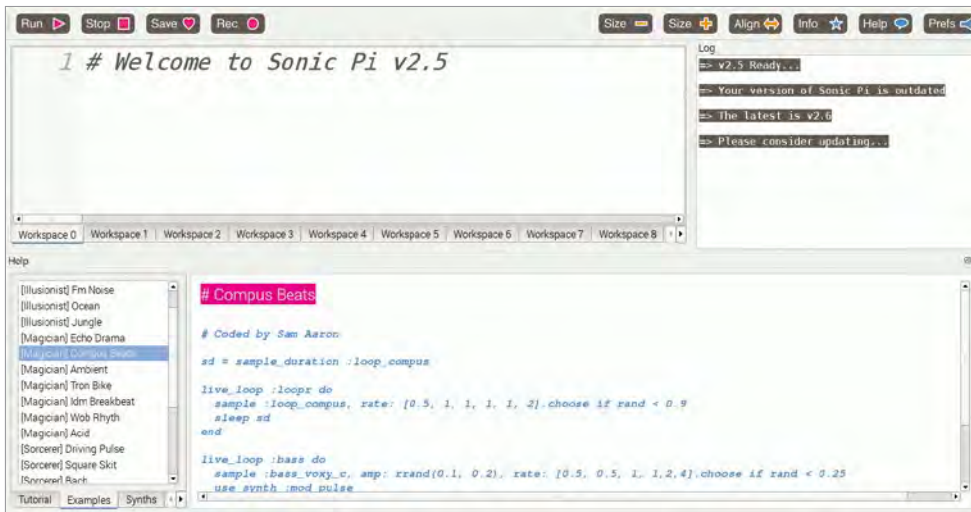


FIGURE 2: Copying an example into the code viewer.



FIGURE 3: Selecting text in the code viewer.

click on an empty line to insert the code into the workspace. What, there's no empty line? Position the cursor at the end of the welcome line, then either use backspace to remove the existing text, or press Enter to create a new line for the new code.

With the code in the workspace, your screen should look like Figure 4. Click on the *Run* control, and if all the pieces fit together, you'll hear a rather nice groove tune, complete with bass line, synthesizer loops, and drum parts. Click on the *Stop* control when you've heard enough, but leave the code in the workspace: you'll need it again.

Take a moment to consider what you just did:

1. You set up a computer system for making sound and music.
2. You loaded and edited computer code in Sonic Pi.
3. You compiled (ran) that code, converting it from a text file to an audio stream, which is pretty cool.

Now you can try something more exciting than just running a code block and sitting back to listen. It is time to get interactive and do a little live coding.

Return to the workspace with the code from the previous demonstration. Click on *Run*, and let the music begin. In the workspace, scroll to the line that reads

```
sleep sd / 4
```

This code effectively creates the rhythm of the piece by dividing the sample duration by 4, then waiting for that value before the next event is allowed to

sound. While the music is playing, change the 4 to 8. Click on *Run* again, listen for the audible difference, then change it to 2 and click *Run*. Don't click the *Stop* button; the process will update smoothly to the new code. Congratulations, you've just completed your first live coding session.

Now play with other values in the code, always with the same procedure: Edit, run, listen, repeat. Don't worry too much at first about what's what; just make some changes and listen for their effect on the music. You might not like all your changes, but your mistakes are an important part of your development as a live-coding player. Some changes will have a dramatic effect; others might be barely noticeable. As with most other arts, you'll learn the most by doing and through trial and error. Eventually, you'll become more comfortable with the instrument and its possibilities. As with any other instrument, your Sonic Pi skill improves with time and practice. See the box titled "Documentation" for tips on where to get additional information on the Raspberry Pi environment.

MAKING YOUR OWN MUSIC

Let's make a song. I'm not going to worry about whether it's a good or bad song; I just want to show how to make a simple song with Sonic Pi. Don't worry if you don't know much – or anything – about music composition, you can learn as you go with Sonic Pi's help.

First, you need to understand the difference between synthesized sounds and

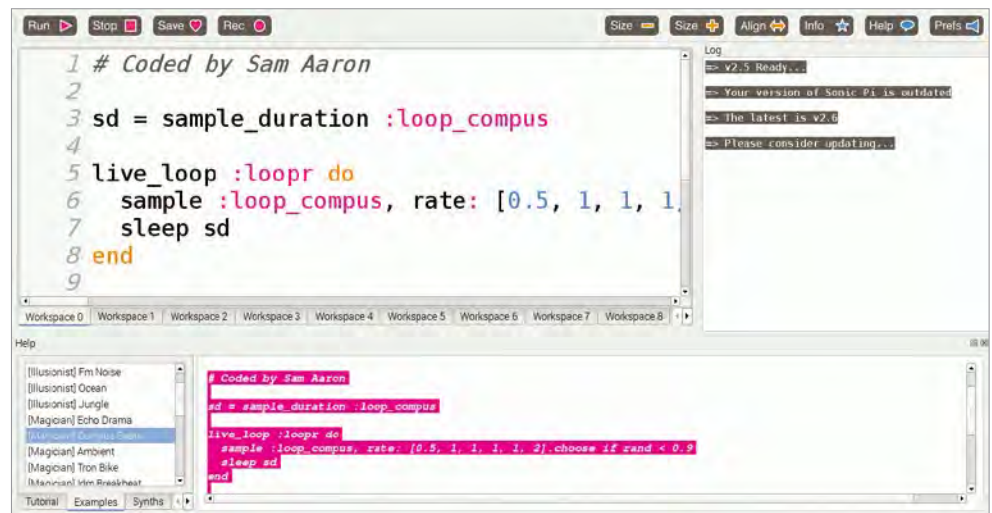


FIGURE 4: Code in the workspace – ready to run.

sampled sounds. A synthesizer (short form: synth) is an electronic sound-producing device that sounds notes played into it, just like a trumpet sounds the notes played by the performer. A sample is a recording of a sound already made. Typically, samples aren't very long, but they can be any length. Some samples are edited to function as audio loops, a very useful feature, as you're about to find out.

You also need to understand a few things about music composition. You don't have to worry about the rules and regulations. You just need to know how to create a musical form using those synths and samples.

Many songwriters start with a rhythm pattern. It doesn't really matter how you start, but I'll take the well-known way for now. I'll begin by making a drum part.

A drum set is a complex instrument. A typical set will have a bass drum, a snare drum, one or more mounted tomtoms, a floor tom, a hi-hat, and one or more cymbals. And, one person plays all those pieces. That means that on a real set, the player can play no more than four of those pieces at once. Keep that in mind if you want to make a realistic drum part. Speaking of realism, you can design drum set sounds with a synthesizer, but since drum samples sound more realistic, I'll go with the sampled sounds.

Start by defining the drum set. Copy the following code into an empty workspace:

```
load_samples [:drum_bass_soft, ↵
:drum_snare_soft, ↵
:drum_cymbal_soft]
```

That's a bass drum, a snare drum, and a cymbal. Now make a pattern for each piece of the set, starting with the bass drum:

```
define :bassdrum do
  8.times do
    sample :drum_bass_soft
    sleep 0.5
  end
end
```

Add this code to your workspace, then click the Run control. You should hear eight beats played by the soft bass drum

sample, with a half-second delay (sleep) between beats.

Now add a snare drum:

```
define :snare do
  sample :drum_snare_soft
  sleep 1
end
```

If you run your workspace code now, you'll hear the bass drum first, then the snare. That isn't what you want, but don't bother about it now. The piece isn't finished yet.

Complete the drum set with a cymbal sample:

```
define :cymbal do
  sample :drum_cymbal_soft
  sleep 8
end
```

The `sleep` value is longer because you only want the cymbal at a certain point in the progression of the piece.

Now that you've made a drum set from sampled sounds, the next step is to make a pattern for a synthesizer part. Copy this code into your workspace:

```
define :sinesynth do
  use_synth :sine
  notes = [:A2, :C3, :G2, :B2, ↵
:F2, :A2, :E2, :Gs2]
  notes.each do |n|
    play note(n)
    play note(n)+12
    sleep 1
  end
end
```

The note definition is a note name followed by octave specifier. For instance, the note `Gs2` means a G-sharp played in two octaves higher than `Gs0` (octaves range from 0 to 10 in Sonic Pi).

The `|n|` is shorthand for the notes list. A note from the list passes to the `play` command as the `n` value of the note.

DOCUMENTATION

The Sonic Pi environment includes a great tutorial by Sam Aaron, Sonic Pi's chief designer, and of course, you'll find Help examples for play and study. The Sonic Pi website includes links to a variety of music and sound examples. For more links, including audio and video examples, search for "sonic pi" on Google, YouTube, Vimeo, and other sources.

The extra `play` command doubles the note at the octave; where the list says `A2`, that line plays an `A3`. Doubling at the octave makes the part sound more full.

By the way, this code is an edited version of a synth definition from one of the Sonic Pi examples. I changed the synth to the sine instrument, revised the note list, and shortened the play block. I encourage you to do the same with any interesting or useful code you find. Borrowing open source material and changing it for your own purposes is a time-honored practice among programmers and musicians.

At this point, the instruments are defined, and they have patterns to play. But if you run the code now, you'll just hear each instrument play in sequence, not together. You need to arrange the parts into a composition.

Fortunately Sonic Pi provides a neat way of arranging the parts. This code shows how to do it:

```
in_thread(name: :bassdrum) do
  loop{bassdrum}
end

in_thread(name: :sinesynth) do
  sleep 4
  loop{sinesynth}
end

in_thread(name: :snare) do
  sleep 12.5
  loop{snare}
end

in_thread(name: :cymbal) do
  sleep 20
  loop{cymbal}
end
```

That `in_thread` function is the organizer here, binding the separate parts into a single piece. It starts the playback with the bass drum, with no delay. In these thread blocks, the sleep value functions as a delay time for starting the instrument and its pattern. So when you run this code, you'll hear the instruments enter in the order seen above (though, in fact, you can arrange the `in_thread` blocks in any order).

One more addition to the code, and I'll call it complete. The synth pattern is okay as it is, but I think it could be im-

proved a lot by adding some chords to it. All you need to do is define a new instrument:

```
define :sawsynth do
  use_synth :saw
  chords = [[:A :minor], :G, :F, :E]
  chords.each do |n|
    play_pattern_timed ↗
    chord(n), 0.25
    sleep 1.25
  end
end
```

And put it in the thread:

```
in_thread(name: :sawsynth) do
  sleep 20
  loop{sawsynth}
end
```

The `play_pattern_timed` function plays the chord notes in sequence instead of all together, a process called arpeggiation.

The instrument is timed to come in with the cymbal. Click on Run, and you should hear a steady bass drum beat, followed by a descending bass line, then the snare drum, and finally the arpeggiated chords, announced by the cymbal crash.

This little exercise produces a repeating pattern called a riff. A single riff can be sufficient material for a song, but typically more than one riff is used. You could write another riff for a different synth and drum set and then add it to the existing workspace, arranging the second riff to follow the first. I leave that suggestion as a further exercise in your studies. Meanwhile, you can download and browse the complete code [2].

Oh, and now that you're a 21st-century electronic music producer, you'll want to know how to save your work as a WAV or MP3 in order to distribute it on the Internet. Sites such as SoundCloud [3] and Bandcamp [4] are great ways to put your pieces out into the world and get constructive feedback from other producers. Once again, saving a WAV is easy with Sonic Pi. Before you start playing your piece, click on the Rec button. Start the piece, stop it when you like, then click REC again to stop the recording. You'll be prompted for a place to save your file. Sonic Pi won't make an

MP3 directly, but you will find many utilities online that will convert the file. (Search for “WAV to MP3.”)

MORE THINGS TO TRY

After you get started making music, try the following experiments:

- Change the synths. Some will work nicely, others will need more attention to their parameter sets, and some might not work well at all.
- Switch the intensity of the drum set components from soft to hard. Change the components of the set completely.
- Change the order and timing of the instrument entries into the performance thread.
- Change sleep values to add more variety to the rhythm.
- Work on the mix – the overall balance of the sounds. Add a reverb or other effect to the code.
- Use the code and your expansions on it to explore its use in a live-coding performance.

If you have the hardware resources, you can even make your own samples. Remember, a sampled sound can be anything, such as sounds made by kitchen utensils or other common household appliances; it doesn't have to be a “musical” sound. See the box titled “After the First Steps” for more on building and enriching your Raspberry Pi music environment.

LEARNING THE WALK

After your first steps into Sonic Pi, you might like to learn more about it and how you can use it to make your own sounds and music. If you're completely new to making music with a computer, you can work through the tutorial exercises. I began my exploration with the tutorial, and it was a great starting point. Of course, at any time you can run the various examples. The tutorials progress from simple pieces at the apprentice stage to more complicated works at the wizard level.

By the way, the tutorial covers a lot of material dealing with the basics of computer music and sound. You don't need to go through all of it right away, but as you become more interested in making your own pieces, you'll want to check out the whole tutorial.

The “hands-on” philosophy is at the core of Sonic Pi. The practice of live-coding music composition is essentially interactive, treating the computer as a musical instrument played in realtime by a performer (that is, you). Musicians have been using the computer since its earliest days, but modern hardware allows a deeper level of interaction at the code level. Live coding is a new performance art, limited only by your hardware capabilities and your imagination, and Sonic Pi is an excellent introduction to this new art. Most of the example programs are designed for interactive performance, so you can jump in at any time during a run.

TROUBLESHOOTING

If you have no sound with the examples, you'll need to do some problem-solving. Make sure your speakers are turned on and all physical connections are made properly. Check the plugs and cables to and from your speakers, and make sure all connections are made properly.

AFTER THE FIRST STEPS

As I mentioned earlier, the Raspberry Pi's audio is limited to output only. However, thanks to its USB ports, you can connect a high-quality digital audio interface to the Raspberry Pi. Many interfaces require power from the USB connection, so you might need to invest in a powered USB hub to avoid taxing the board's power distribution. You'll also need equipment typically found in recording studios, such as microphones and instruments. By now, you've probably figured out that while your Raspberry Pi is indeed a low-cost item, outfitting a recording studio around it can become rather expensive.

Live coding can be a group activity, so form a band of players. You can assign parts to each player in the same way any band is organized, with a bass player, a synthesizer player, a drummer, and so on. Making music with the computer by yourself can be a satisfactory activity. Making music with a group of people playing computers can be awesome. You'll also experience a new thing, a way of making music not possible until the development of modern computers. Everyone knows the computer is central to modern music production, but live-coding treats the machine as an instrument in a new way, one played by a performer programming the computer in realtime.

The Raspberry Pi can run the Guitarix [5] and Rakarrack [6] sound-processing programs, making it an ideal low-cost, multi-effects unit for electric guitar players. The board can also run software synthesizers, such as amSynth and ZynAddSubFX. A synth player can show up for a show fully equipped with only a keyboard or controller interface, a Raspberry Pi running a softsynth, and the proper cables. By the way, for the best quality sound from a Raspberry Pi-based synthesizer or effects processor, you should use a USB audio interface.



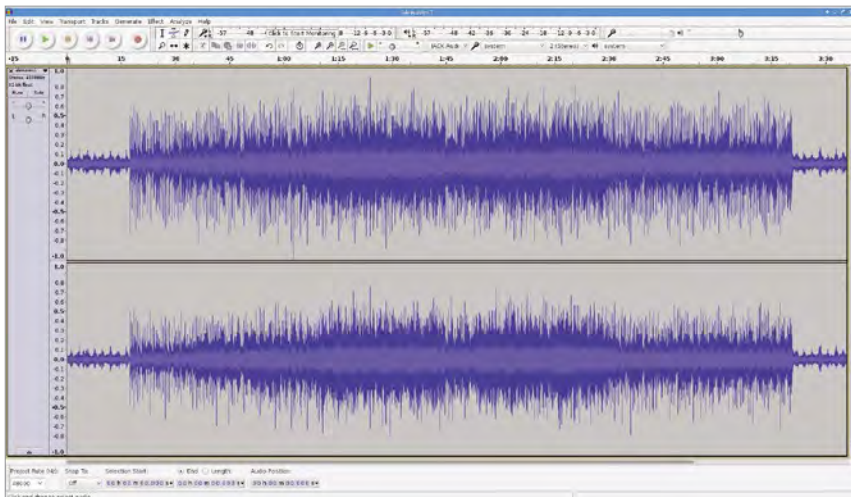


FIGURE 5: The Audacity soundfile editor.

Check the *Audio Output* settings in the *Sonic Pi Prefs* dialog, and make sure the correct device is selected (headphone/speakers or HDMI). Set the volume control on your speakers to 50% or higher. On the software side, set the output level to 50%, and verify that no output control is muted.

The Raspberry Pi is powerful enough for Sonic Pi, but it's also easy enough for a user to overwhelm the board's playback capabilities. Be careful how many audio streams run at one time, and watch out for overlapping parts that can distort and delay playback.

INTO THE BEYOND

If Sonic Pi ever starts to feel limited, you can look into running other environments for sound and music composition on the Raspberry Pi. Highly evolved systems such as Csound [7], Pure Data (Pd) [8], RtCmix [9], and SuperCollider [10] have active development branches focused on the Raspberry Pi.

In particular, SuperCollider has been the main influence on the development of Sonic Pi, making it a good choice to continue your studies. However, any one of these systems is a good choice for continuing your journey into the world of computer music. All have graphic development tools – in the case of Pure Data, the entire system is graphic – and all are very powerful.

As you progress with Sonic Pi, you'll want to pick up a few extra tools to help you in your work. A sound editor will be needed when you start making

your own samples and loops, preferably an editor that includes tools for loop definition and beat detection (to determine where the strong beats occur in a soundfile).

The Audacity [11] program is a superb cross-platform sound editor that is freely available for Linux, Mac OS X, Windows, and a version of Audacity will run on the Raspberry Pi (Figure 5).

By today's standards, the Raspberry Pi is a relatively low-powered device, and indeed; the central processor for my Raspberry Pi B+ is less than 1GHz in speed. By comparison, my desktop music workstation runs at 3.5 GHz with six cores. But consider this comparison: My first computer ran at 1/100th the speed of the Pi, at 10 times the cost, and took up the available space of my desk. Of course, the real capability of any computer is measured by its usefulness. For the purposes of this article, "usefulness" means its ability to run modern software for sound and music, and in that spirit, the Raspberry Pi has proven that it is quite useful.

I have to mention again that the Raspberry Pi's fun factor is very high. When I was 11 or so I got into amateur radio (ham radio), which led me into building my own gear, often with the famed Heathkit packages. I had a lot of fun with those kits, and I find myself getting into the Raspberry Pi with the same level of enjoyment. Which is to say, it's amazing great fun, and once you get into what the Raspberry Pi can do, you'll have a great time figuring out what you can do with it. ✨

INFO

[1] Sonic Pi: <http://sonic-pi.net/>

[2] Code for this article:
[http://linux-sound.org/misc/
first-riff-with-sonicpi.txt](http://linux-sound.org/misc/first-riff-with-sonicpi.txt)

[3] SoundCloud:
<https://soundcloud.com/>

[4] Bandcamp: <https://bandcamp.com/>

[5] Guitarix: <http://guitarix.org/>

[6] Rakarrack:
<http://rakarrack.sourceforge.net/>

[7] Csound: <http://www.csounds.com/>

[8] Pure Data (Pd): <https://puredata.info/>

[9] RtCmix: <http://rtcmix.org/>

[10] SuperCollider:
<http://supercollider.github.io/>

[11] Audacity: <http://audacityteam.org/>

REAL SOLUTIONS FOR REAL NETWORKS



**FREE
CD or DVD
in Every Issue!**

Each issue delivers technical solutions to the real-world problems you face every day.

Learn the latest techniques for better:

- network security
- system management
- troubleshooting
- performance tuning
- virtualization
- cloud computing

on Windows, Linux, Solaris, and popular varieties of Unix.

6 issues per year!

ORDER ONLINE AT: shop.linuxnewmedia.com

WRITE FOR US!

Raspberry Pi Geek is looking for fresh, original articles on Raspberry Pi and other maker hardware platforms. If you work with Raspberry Pi, Arduino, BeagleBone, Minnow-Board, Parallella, or another similar technology, and you have an interesting story about a recent project or configuration, drop us a line at edit@raspberrypi-geek.com.

We're also seeking articles on software tools for maker hardware environments – including applications in the repositories of the leading Raspberry Pi operating systems, as well as homegrown scripts for custom configurations.

We're especially interested in electronics projects that use Raspberry Pi's GPIO to control real-world hardware devices for a practical (or whimsical) purpose. Write for Raspberry Pi Geek and share your story.



AUTHORS

Michael Badger	52, 86
Paul Brown	42
Joe Casad	3, 8, 10, 18, 26
Heike Jurzik	26
Dave Phillips	90
Dmitri Popov	34
Bill Sumner	70
Scott Sumner	58

DISCLAIMER

All brand or product names are trademarks of their respective owners. Contact us if we haven't credited your copyright; we will always correct any oversight.

CONTACT INFO

Editor in Chief

Joe Casad, jcasad@linuxnewmedia.com

Managing Editor

Rita L Sooby, rsooby@linuxnewmedia.com

Copy Editor

Amber Ankerholz

Layout

Dena Friesen, Lori White

Cover Design

Dena Friesen and Lori White, Illustrations by Young-Sun Teh, Alexandr Aleabiev, and xalanx @123RF.com and Raspberry Pi

Advertising – North America

Ann Jesse, ajesse@linuxnewmedia.com
phone +1 785 841 8834

Advertising – Europe

Brian Osborn, bosborn@linuxnewmedia.com
phone +49 89 99 34 11 48

Publisher

Brian Osborn, bosborn@linuxnewmedia.com

Marketing Communications

Gwen Clark, gclark@linuxnewmedia.com

Customer Service / Subscription

For USA and Canada:
Email: cs@linuxnewmedia.com
Phone: 1-866-247-2802
(toll-free from the US and Canada)
Fax: 1-785-856-3084

For all other countries:
Email: subs@linuxnewmedia.com
Phone: +49 89 99 34 11 67
Fax: +49 89 99 34 11 98
Linux New Media USA
616 Kentucky St. Lawrence, KS 66044
www.linuxpromagazine.com

While every care has been taken in the content of the magazine, the publishers cannot be held responsible for the accuracy of the information contained within it or any consequences arising from the use of it. The use of the DVD provided with the magazine or any material provided on it is at your own risk.

Copyright and Trademarks © 2016 Linux New Media USA, LLC

No material may be reproduced in any form whatsoever in whole or in part without the written permission of the publishers. It is assumed that all correspondence sent, for example, letters, email, faxes, photographs, articles, drawings, are supplied for publication or license to third parties on a non-exclusive worldwide basis by Linux New Media unless otherwise stated in writing.

Linux Magazine Special (ISSN 1757-6369) is published by Linux New Media USA, LLC, 616 Kentucky St, Lawrence, KS, 66044, USA.

All brand or product names are trademarks of their respective owners. Contact us if we haven't credited your copyright; we will always correct any oversight.

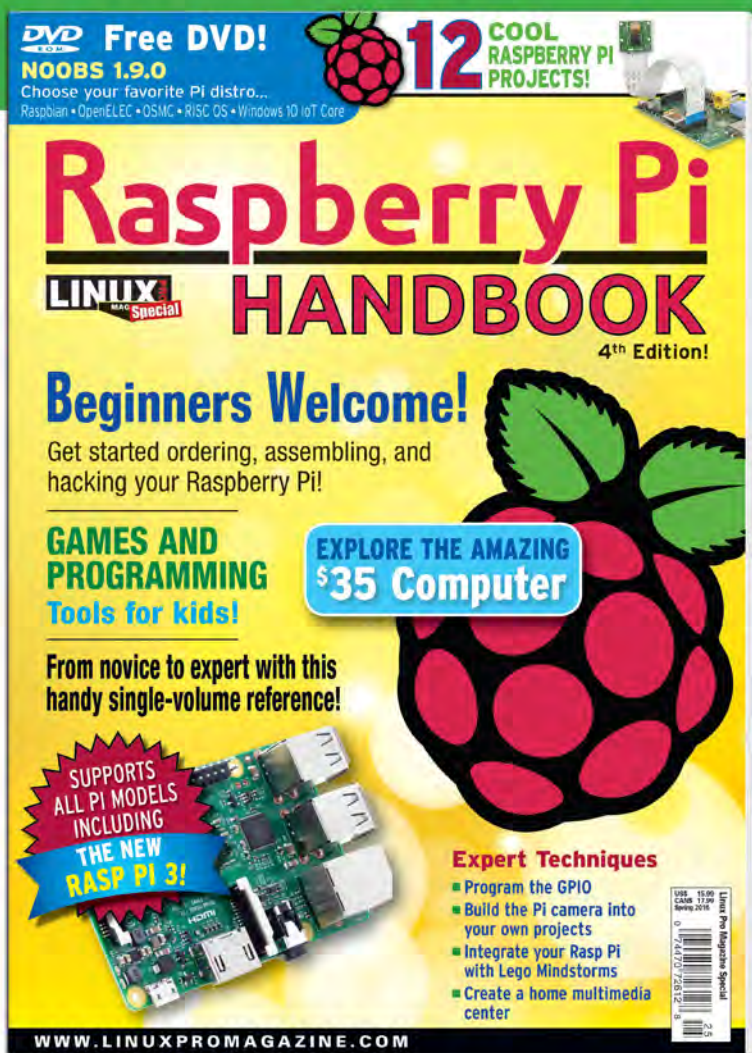
Printed in Germany

Distributed by COMAG Specialist, Tavistock Road, West Drayton, Middlesex, UB7 7QE, United Kingdom

Published in Europe by: Sparkhaus Media GmbH, Putzbrunner Str. 71, 81739 Munich, Germany

Raspberry Pi

HANDBOOK



In case you missed it last time...

**You ordered your Raspberry Pi...
 You got it to boot...what now?**

The Raspberry Pi Handbook takes you through an inspiring collection of projects. Put your Pi to work as a:

- media center
- web server
- IR remote
- hardware controller
- and much more!

Discover Raspberry Pi's special tools for teaching kids about programming and electronics, explore Wolfram Mathematica, and find out how to integrate your Rasp Pi system with LEGO Mindstorms.

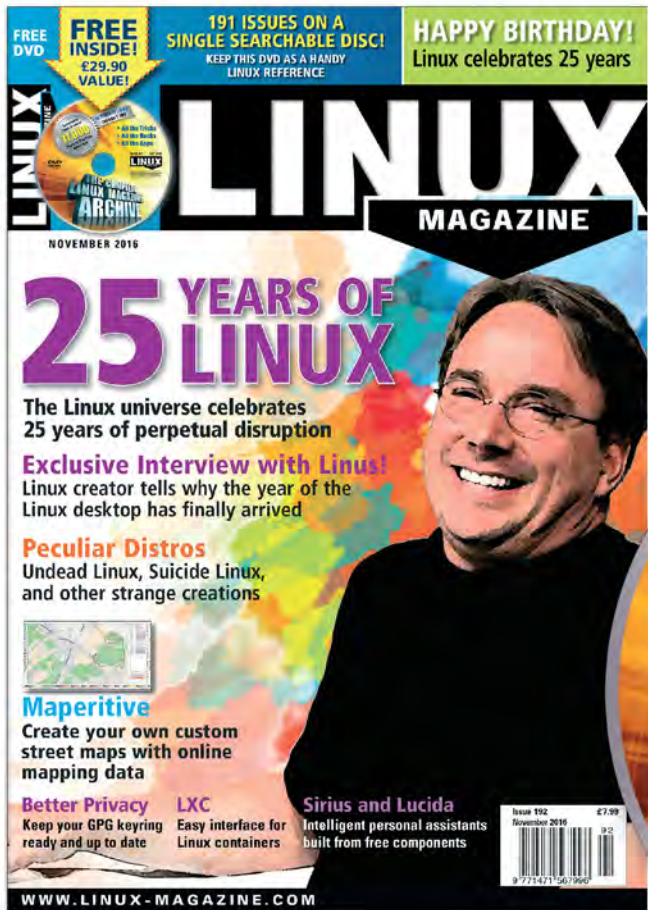
**THE ONLY RASPBERRY PI REFERENCE
 YOU'LL EVER NEED!**



ORDER ONLINE:

shop.linuxnewmedia.com/rpi

Happy 25th Anniversary to Linux!



Help us celebrate 25 years of Linux with the All-Time Archive DVD of Linux Magazine!



Order today and get all 191 issues of Linux Magazine on one handy DVD – FREE with Issue #192!

BEST VALUE: Subscribe to the print edition and get the archive DVD FREE with November 2016 issue (a \$39.90 / €29.90 value). You'll get Linux Magazine every month and 15+ years of content.

Order Now! Shop.linuxnewmedia.com