your Raspberry Pi remotely, but as if you were using a terminal session directly from the Raspberry Pi.

If you are using a Mac or Linux computer, start a terminal session and enter the following command, replacing 192.168.1.15 with the IP of your Raspberry Pi that you found earlier.

```
$ ssh pi@192.168.1.15
```

Whether using Putty or a Mac or Linux terminal, you now need to log in. The username is "pi" and the password is "raspberry".

Congratulations, you can now connect to your Raspberry Pi from another computer on your home network and you can if you want disconnect the keyboard mouse and monitor from the Raspberry Pi.

Blinking an LED

To get the hang of running a program on the Raspberry Pi, we can start in the same way as you did on an Arduino by blinking an LED.

Since there is no built-in LED that is controllable from one of the normal GPIO (general purpose input/output) pins on a Raspberry Pi, you need to attach an external LED and resistor using breadboard as shown in Figure 7-8.

The LED is going to be controlled by pin GPIO 18.

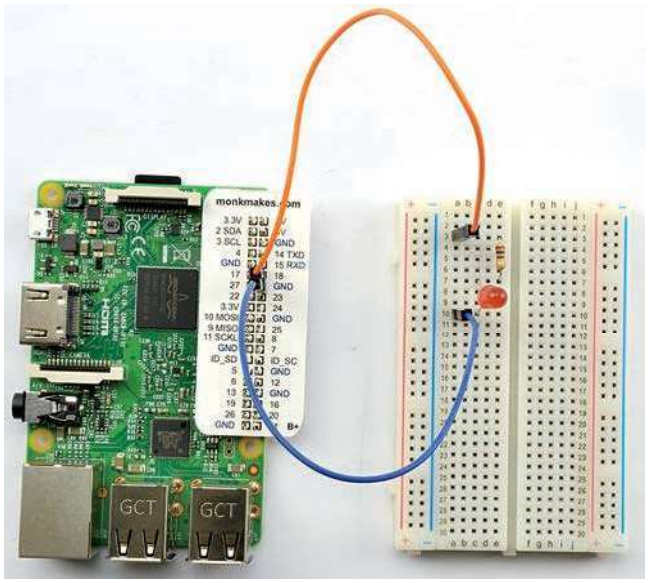


FIGURE 7-8 Attaching an LED and resistor to a Raspberry Pi

Python

Whereas Arduino is programmed in C, the most common language used to program a Raspberry Pi is Python.

Many of the features of Python are similar to those of C, but unlike C, Python uses indentation to mark off blocks of code. So in Arduino C, you might write

```
if (x > 10) {
  x = 0;
}
```

The equivalent in Python would be

```
if x > 10 :
  x = 0
```

The differences in the formatting are that in Python:

- You do not need parentheses around the condition "x > 10".
- The start of a block of code is indicated by ":" rather than "{".
- All lines of code inside the block must be indented to the same level. This is done by convention in Arduino C but is mandatory in Python.

There are lots of other differences between Python and Arduino C, and if you would like to find out more about learning Python, you might enjoy my book *Programming the Raspberry Pi: Getting Started with Python* from TAB DIY.

You Will Need

To connect your LED up, you will need the following items.

Quantity	Item	Appendix Code
1	Raspberry Pi Model B (Pi 2 or later preferred)	M11
1	Solderless breadboard	K1, T5
1	270Ω/470Ω resistor	K1
1	Red LED	K1
2	Male to female jumper wires	K1, T12

Raspberry Pi GPIO Connections

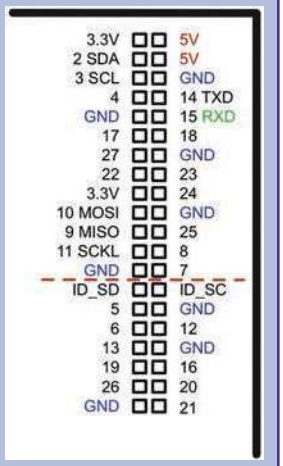
Figure 7-9 shows the GPIO pin out of a Raspberry Pi 3.

The dotted line separates the top 26 pins of the connector from the rest because in version 1 of the Raspberry Pi model B, there were only the top 26 pins rather than the full 40 pins of the newer models.

Some of the pins on the connector are to supply power to other devices attached to the connector. This includes 5V, 3.3V and GND connections. Most of the rest of the pins can be used as either digital inputs, digital outputs or analog outputs (PWM) but unlike the Arduino, the Raspberry Pi does not have any analog inputs.

Some of the GPIO pins also have a second function relating to a serial interface:

FIGURE 7-9 The Raspberry Pi GPIO Connector



- GPIO 2 and 3 are used as SDA and SCL if the I2C interface is enabled to allow I2C displays and sensors to be connected to the Raspberry Pi.
- GPIO 9, 10 and 11 are used in the SPI (Serial Programming Interface).
- GPIO 14 and 15 will be used by the TTL Serial Interface if that is enabled.

By default, none of these extra interfaces are enabled when you install Raspbian, so all the GPIO pins can be used as inputs or outputs unless you enable the interface (I2C, SPI or Serial) as shown in Figure 7-6. In general, if you just need a digital output, then avoid these special purpose pins.

Digital outputs on a Raspberry Pi are not capable of providing to 40mA of an Arduino. The maximum that you should use for one pin is 16mA. The inputs and outputs of the Raspberry Pi operate at 3.3V rather than the 5V of an Arduino Uno. Connecting 5V to a Raspberry Pi GPIO is likely to damage your Raspberry Pi. You must always use 3.3V logic with a Raspberry Pi.

Using a 270Ω resistor will make the LED brighter, but a 470Ω resistor will also work fine. These items (apart from the Raspberry Pi) are all included in the Electronics Starter Kit for Raspberry Pi by MonkMakes (http://monkmakes.com/rpi_esk) and the Hacking Electronics Mega Kit (http://monkmakes.com/hacking2).

Software

With the LED connected, run the example Python program "ch07_blink.py" by issuing the following commands:

```
$ cd /home/pi/hacking2/pi
$ python ch07_blink.py
```

The first line makes sure that you are in the right directory to run the program and you only need to type this the first time that you want to run the program. The second line runs the Python program. Note that if you are using an older version of Raspbian then you may have to run the command with "sudo" in front like this:

```
$ sudo python ch07_blink.py
```

If all is well, the LED will start to slowly blink. When you have had enough, hold down the CTRL key and type **c** to quit the program.

The code for ch07_blink.py is listed below:

```
import RPi.GPIO as GPIO
import time

# Configure the Pi to use the BCM (Broadcom) pin names
GPIO.setmode(GPIO.BCM)

led_pin = 18
GPIO.setup(led_pin, GPIO.OUT)

try:
    while True:
        GPIO.output(led_pin, True)  # LED on
        time.sleep(0.5)             # delay 0.5 seconds
        GPIO.output(led_pin, False) # LED off
        time.sleep(0.5)             # delay 0.5 seconds

finally:
    print("Cleaning up")
    GPIO.cleanup()
```

The code has some similarities with its Arduino equivalent. Python uses the # symbol to denote comments that are not part of the code.

First, the *RPi.GPIO* Python library is imported. This is the library that allows Python programs to control the GPIO pins. The second import is for the *time* module that is used to produce the delay between turning the LEDs on and off.

The code *GPIO.setmode(GPIO.BCM)* is needed in all your Python programs and specifies that the standard numbering system for GPIO pins should be used rather than the alternative numbering system that relies on pin positions.

The variable *led_pin* is specified as the pin that is to be connected to the LED and in this case it is pin 18. The line after that sets *led_pin* to be an output.

The *try* and *finally* commands ensure that when you press CTRL-C to quit the program the code after *finally* is run that sets all the GPIO pins back to a safe state.

The code indented in from the *try* block, first sets the GPIO pin connected to the LED high, delays for half a second, sets it low, delays for another half second and so on indefinitely because of the *while* loop enclosing it.

Controlling a Relay with Raspberry Pi

Now that you can turn a GPIO pin on and off, we could replace the LED and resistor with a relay module, opening up more possibilities such as switching a toy on and off as described in Chapter 6 in the section "Hacking a Toy for Arduino Control."

Figure 7-10 shows a relay module attached to the Raspberry Pi using female to female jumper wires.

The connections are as follows:

- GND on the relay module to GND on the Raspberry Pi
- VCC on the relay module to 5V on the Raspberry Pi
- IN on the relay module to GPIO18 on the Raspberry Pi

As with the relay module experiments with Arduino in Chapter 6, you will need to run slightly different programs depending on whether your relay module has "inverted" inputs. So run either "ch7_relay_click.py" or "ch7_relay_click_inverted.py". Note that if you run the wrong program, it just won't work, you will not break anything.