Lab - 01
Installing NOOBS

Problem

This tutorial will show you the process of installing NOOBs on to a SD card, ready for you to plug into your Pi and to select an Operating System.

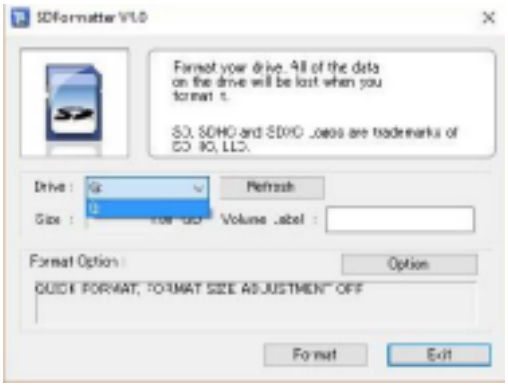
Solution

The following steps were performed on a Windows computer. The steps may vary if you are using Linux or a Mac, but the principle will be the same.

First we need to prepare our SD card. To do this, we will use a piece of software called SD Formatter, by SD Association. If you haven't already got this software, you will need to download and install it.

Plug your SD card into your computer and load up the SD Formatter software.

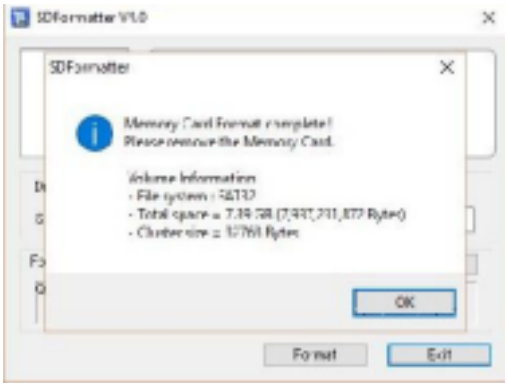
Make sure the correct drive letter for your SD card has been selected from the drop down.



Once you have selected the drive letter, click on the "Options" button, and select "ON" from the "FORMAT SIZE ADJUSTMENT" dropdown.

Now click the format button and click "OK" on the prompts.

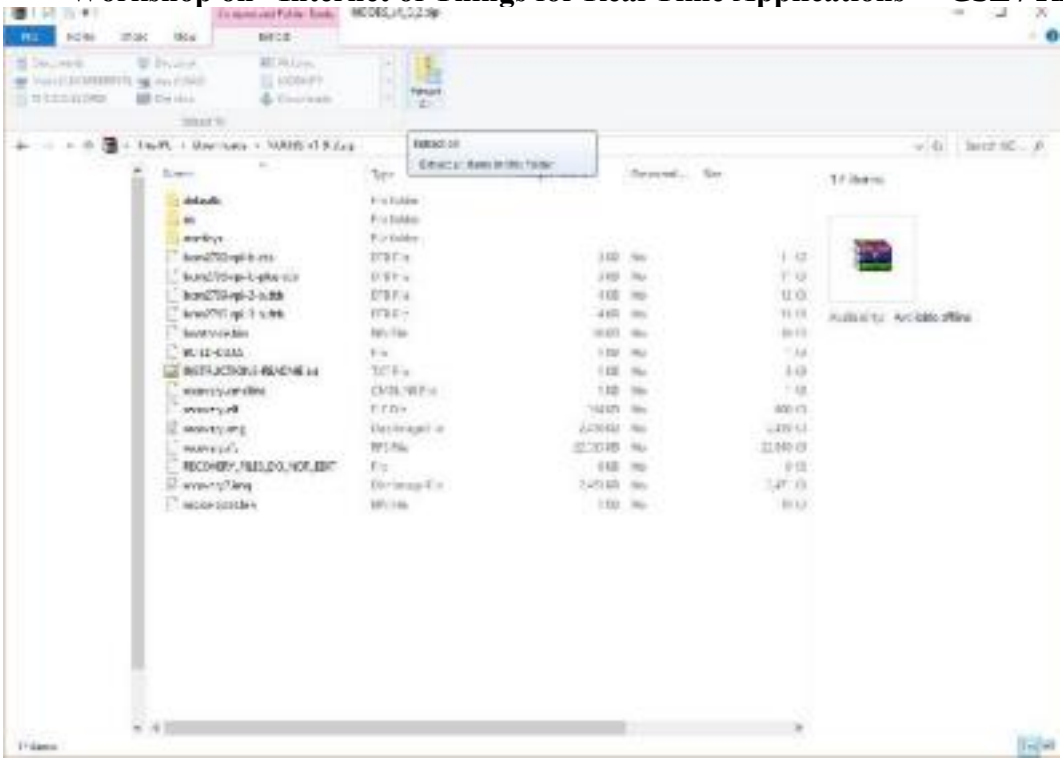
Your SD card has now been formatted and is ready to load NOOBs on to.



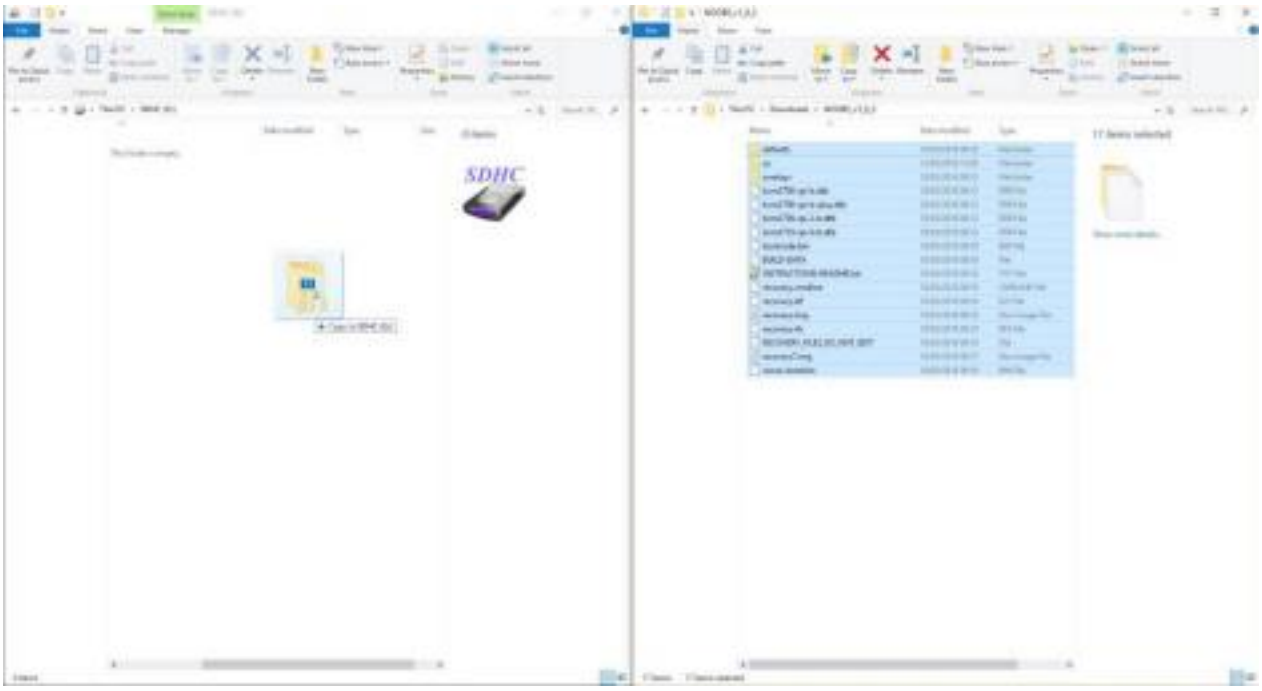
Head over to the Raspberry Pi Foundation Download page and select NOOBS.

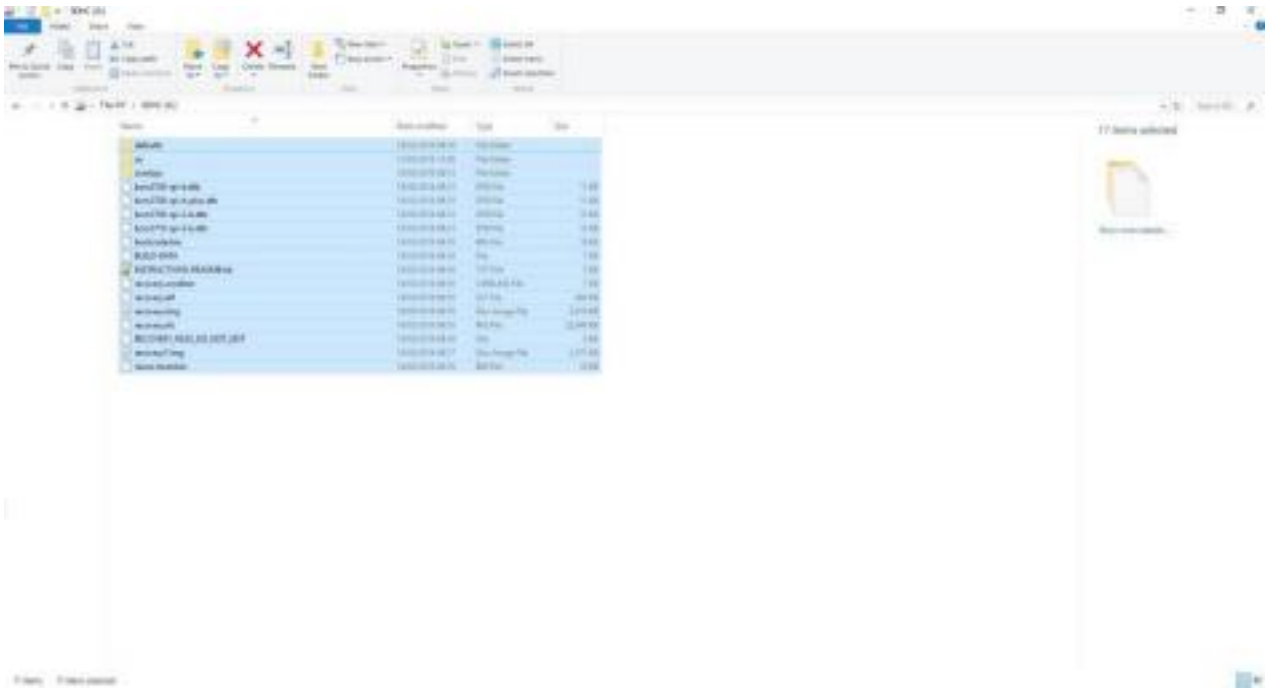
Then pick a NOOBS version. We recommend the full NOOBS download, unless you want to install an OS other than Raspbian

Once the download has complete, you'll need to extract the files from the download .zip file.



With the files extracted, you now need to copy them over to your formatted SD card.





And that's it! NOOBS has now been loaded on to your SD Card ready for you to plug into your Pi.

Lab - 02
Raspberry Pi Quick Start Guide - Connecting the System

Problem

You have everything that you need for your Raspberry Pi, and you want to connect it all together.

Solution

Unless you are embedding your Raspberry Pi in a project or using it as a media center, you need to attach a keyboard, mouse, monitor, and probably a WiFi dongle, unless you have a Raspberry Pi 3.

Figure shows a typical Raspberry Pi system.

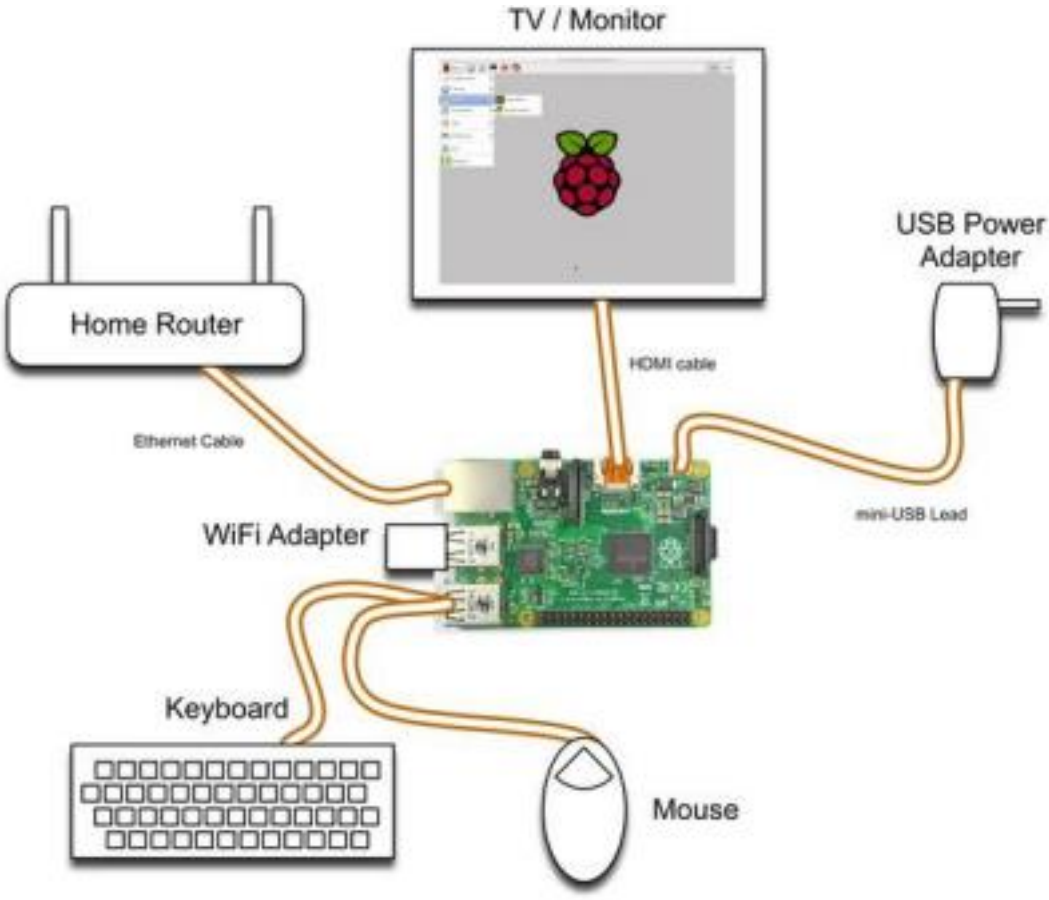


Figure. A typical Raspberry Pi system

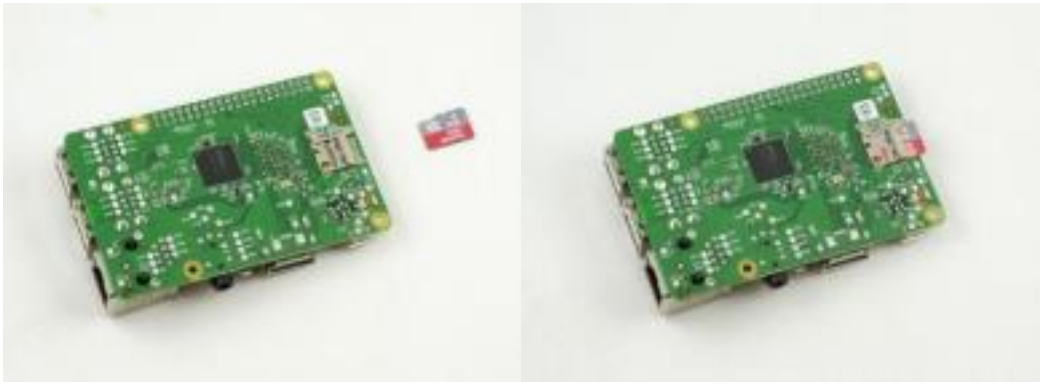
Discussion

The Raspberry Pi is perfectly happy with pretty much any keyboard or mouse, wired or

wireless. The exception to this is Bluetooth wireless keyboards and mice, which will not work with the Raspberry Pi.

If you have an older Raspberry Pi or a model A or A+ and run out of USB sockets, then you will also need a USB hub.

Start by inserting the micro SD card into the dedicated micro SD card slot on the Raspberry Pi. This is located on the underside of the Pi.



If you need an internet connection, plug in your Ethernet cable. (Optional)

Then plug in your HDMI cable. The other end of the cable should be plugged into your TV or Monitor. If you are using a TV, make sure you have the correct input selected. Make sure your TV/Monitor is turned on.

Now we need to plug in our Keyboard and Mouse into the USB ports on the Raspberry Pi.

Once you have double checked everything has been connected properly, its time to plug in your micro usb power supply. With the power supply plugged in, turn on the power supply.

If this is the first time you have powered on your Raspberry Pi, and you have a NOOBS Micro SD card, you will need to install an Operating System. Simply select one from the list. You can see more Operating Systems when you are connected to the internet via an Ethernet cable.

Once the OS has been installed your Pi should automatically reboot. This time your Pi should boot into the installed OS.

If you have installed Raspbian (the recommended OS) the default username and password is:

Username: pi
Password: raspberry

RECOVERY

If you managed to break your OS install, or you simply want to start from fresh again, you can do so easily using the NOOBs recovery feature.

To access this, you need to hold down the Left Shift key on the keyboard immediately after turning on the Raspberry Pi. Keep Shift pressed until the recovery menu displays.

You should see a list of the operating systems, and a x next to the OS you currently have installed. To simply overwrite your existing install, press the Install button. You will then have a prompt letting you know that all data on the SD card will be deleted. If you are happy with everything being removed, click YES to continue and re-install the OS.

If you would like to change operating systems, simply untick the currently installed OS and tick the OS you want to install. Like above, you'll be told that all data will be removed. Clicking YES will continue the process and delete your existing OS, and install the newly selected OS.

Lab - 03
Python Hello World Program

Problem

Run your first python program Hello World!

Solution

Things you will need:
Raspberry Pi + SD Card
Keyboard + Mouse
Monitor + HDMI Cable
Power Supply

Discussion

1. Start by opening a Terminal, by click the icon in the taskbar. 2. Start by creating a new directory, this is where we are going to store all our scripts for this series of worksshop, make a directory called iotworkshop:

mkdir iotworkshop

3. We can use the command ls -l to view the contents of the current directory:

ls l

4. You should be able to see the directory you created. 5. Now we need to change our current directory to the newly created iotworkshop directory:

cd iotworkshop

6. We can use the command nano, to open it in a text editor. sudo nano helloworld.py
7. Now enter the following code:

```
#!/usr/bin/python
print "Hello World!"
```

8. Press ctrl+X to exit, you will be prompted to save the file, simply press Y, and then be you will be prompted to give the file a name.
9. It will automatically use its current name helloworld.py, so just hit enter to accept.
10. To execute your script, simply enter the following command: sudo python helloworld.py

Lab - 04
Making Executable Python Program

Problem

How to make Python programs executable.

Solution

Normally, in order to run a Python program you have to tell the Python software to open the file. However it is possible to execute the file without having to call upon Python first. This allows you to call upon your own programs (that you created in Python) at the terminal by simply typing it's name.

Discussion

First you need to tell Linux to mark your Python file as executable, which means that the file is a program. For this example the target file to be made executable will be called helloworld.py. When you come to doing it yourself simply replace this with your own file name. We use the chmod +x command to make a file executable. In the terminal type the following:

```
chmod +x helloworld.py
```

You can now try running the program directly by typing:

```
./helloworld.py
```

Even though you didn't call upon Python the program should still run the same as if you'd typed python helloworld.py. The program can only be run by calling it with it's full location /home/pi/example.py or from the current directory by using ./ as the location.

To make the file accessible in the same way as any other command in the terminal, it needs to be copied (using the command cp) to /usr/local/bin with the following command:

```
sudo cp helloworld.py /usr/local/bin/
```

With the file now located in /usr/local/bin it can be executed from any directory by simply typing it's name. Try changing to another directory and then run the program again by typing the following: helloworld.py

To make your custom-made programs seem more like native utilities, you can rename (using the command mv) them to remove the .py file extension. To change helloworld.py in this way type the following line at the terminal:

```
sudo mv /usr/local/bin/helloworld.py /usr/local/bin/helloworld
```

Now the program can be run by simply typing example at the terminal!

Lab - 05
Finding Your Way Around the GPIO Connector

Problem

You need to connect electronics to the GPIO connector, but first you need to know more about what all the pins do.

Solution

Figure 1. The GPIO pinout (40-pin models)

At the top of the connector, there are 3.3V and 5V power supplies. The GPIO uses 3.3V for all inputs and outputs. Any pin with a number next to it can act as a GPIO pin. Those that have another name after them also have some other special purpose, so 14 TXD and 15 RXD are the transmit and receive pins of the serial interface. SDA and SCL form the I2C interface, and MOSI, MISO, and SCKL from the SPI interface.

Figure 1 shows the current 40-pin layout, which is the same for all 40-pin GPIO Raspberry Pi models.

The top 26 pins are the same as the 26 pins of the original Raspberry Pi Model B revision 2. This allows the 40-pin Raspberry Pi models to use hardware and software designed for the earlier 26-pin Raspberry Pi designs. The extra pins of the 40-pin connector are made up of three useful extra GND connections and 9 GPIO pins. The ID_SD and ID_SC pins are intended for use in communicating with a special serial memory chip, which can be included on interface boards that conform to the Hardware At Top (HAT) standard and allows the Raspberry Pi to identify the board.

Discussion

Working out which pin is which on a Raspberry Pi is quite error prone if you rely on counting down the pin connector to find the pin you need. A better way of finding the right pin is to use a GPIO template like the Raspberry Leaf shown in Figure 3.

This paper template fits over the GPIO pins telling you which is which. Other GPIO templates include the Pi GPIO Reference Board.

The Hardware At Top standard is an interface standard that you can use with the Raspberry Pi 2, B+ and A+. This standard does not in any way stop you from just using GPIO pins directly; however, interface boards that conform to the HAT standard can call themselves HATs and differ from regular Raspberry Pi interface boards in that a HAT must contain a little Electrically Erasable Programmable Read-Only Memory (EEPROM) chip that is used to identify the HAT so that ultimately the Raspberry Pi can auto
install necessary software. At the time of writing, HATs have not quite met that level of sophistication, but the idea is a good one. The pins ID_SD and ID_SC are used to communicate with a HAT EEPROM.

Lab 06
Verifying the RPi.GPIO Library

Problem

To access Raspberry Pi GPIO, we can use several GPIO libraries. **Solution**

If you are working with Python, Raspbian will have already installed the RPi.GPIO library to access Raspberry Pi GPIO.

Discussion

You can verify the RPi.GPIO library from a Python terminal by importing the RPi.GPIO module, as shown in the following screenshot:

If you don't find this library on Python runtime or get the error message **ImportError: No module named RPi.GPIO**, you can install it by compiling from the source code. For instance, we want to install RPi.GPIO 0.5.11, so type the following commands:

```
pi@raspberrypi ~ $ wget
https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO
0.5.11.tar.gz

pi@raspberrypi ~ $ tar -xvzf RPi.GPIO-0.5.11.tar.gz

pi@raspberrypi ~ $ cd RPi.GPIO-0.5.11/

pi@raspberrypi ~ $ sudo python setup.py install
```

Lab - 07
Connecting an LED

Problem

You want to know how to connect an LED to the Raspberry Pi. **Solution**

Connect an LED to one of the GPIO pins using a 470 or 1k series resistor to limit the current. To make this recipe, you will need:

Breadboard and jumper wires
470 resistor
LED

Figure shows how you can wire this LED using a solderless breadboard and male-to-female jumper leads.

Figure. Connecting an LED to a Raspberry Pi

Having connected the LED, we need to be able to turn it on and off using commands from Python.

Start a Python console from the Terminal with superuser access and enter these commands:

```
pi@raspberrypi ~ $ sudo python
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(18, GPIO.OUT)
>>> GPIO.output(18, True)
>>> GPIO.output(18, False)
```

This will turn your LED on and off.

Discussion

LEDs are a very useful, cheap, and efficient way of producing light, but you do have to be careful how you use them. If they are connected directly to a voltage source (such as a GPIO output) that is greater than about 1.7 volts, they will draw a very large current. This can often be enough to destroy the LED or whatever is providing the currentwhich is not good if your Raspberry Pi is providing the current.

You should always use a series resistor with an LED because the series resistor is placed between the LED and the voltage source, which limits the amount of current flowing through the LED to a level that is safe for both the LED and the GPIO pin driving it. Raspberry Pi GPIO pins are only guaranteed to provide about 3mA or 16mA of current (depending on the board and number of pins in use). LEDs will generally illuminate with any current greater than 1mA, but will be brighter with more current. Use Table as a guide to selecting a series resistor based on the type of LED; the table also indicates the approximate current that will be drawn from the GPIO pin.

LED type	Resistor	Current (mA)
Red	470	3.5
Red	1k	1.5
Orange, yellow, green	470	2
Orange, yellow, green	1k	1
Blue, white	100	3
Blue, white	270	1

Table. Selecting series resistors for LEDs and a 3.3V GPIO pin

As you can see, in all cases, it is safe to use a 470 resistor. If you are using a blue or white LED, you can reduce the value of the series resistor considerably without risk of damaging your Raspberry Pi.

If you want to extend the experiments that you made in the Python console into a program that makes the LED blink on and off repeatedly, you could type the following code into the IDLE or nano editors. Save the file as led_blink.py.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
while (True):
    GPIO.output(18, True)
    time.sleep(0.5)
    GPIO.output(18, False)
    time.sleep(0.5)
```

Remember that to run the program, you must have superuser privileges for the RPi.GPIO library, so you need to use this command:

```
pi@raspberrypi ~ $ sudo python led_blink.py
```

Lab - 08
Leaving the GPIO Pins in a Safe State

Problem

You want all the GPIO pins to be set to inputs whenever your program exits so that there is less of a chance of an accidental short on the GPIO header, which could damage your Raspberry Pi.

Solution

Use a try: finally: construction and the GPIO.cleanup method.

The blink example can be rewritten to exit safely as shown below. The file for the code is called led_blink_safe.py.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

try:
        while (True):
                GPIO.output(18, True)
                time.sleep(0.5)
                GPIO.output(18, False)
                time.sleep(0.5)
finally:
        print("Cleaning Up!")
        GPIO.cleanup()
```

Now, when you Ctrl-C the program to close it, GPIO.cleanup will be called before the program exits.

Discussion

If cleanup is not called or the Pi is not rebooted, then pins set to be outputs will remain as outputs after the program has finished. If you were to start wiring up a new project, unaware of this problem, your new circuit might accidentally short a GPIO output to one of the supply rails or another GPIO pin in the opposite state.

A typical scenario where this might happen would be if you were to connect a push switch, connecting a GPIO pin that you had configured as an output and HIGH to GND.

In summary, either be careful when swapping hardware or use GPIO.cleanup as shown earlier, or reboot your Pi. In any case, its a good idea to power down your Pi while you are connecting new hardware to it.

Lab - 09
Connecting a Push Switch

Problem

You want to connect a switch to your Raspberry Pi so that when you press it, some Python code is run.

Solution

Connect a switch to a GPIO pin and use the RPi.GPIO library in your Python program to detect the button press.

To make this recipe, you will need:

Breadboard and jumper wires
Tactile push switch

Figure shows how to connect a tactile push switch using a breadboard and jumper wires.

Figure. Connecting a push switch to a Raspberry Pi

An alternative to using a breadboard and tactile switch is to use a Squid Button (Figure). This is a push switch with female header leads soldered to the end, which can be connected directly to the GPIO connector.

Open an editor (nano or IDLE) and type in the following code switch.py.

This example code displays a message when the button is pressed:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
        input_state = GPIO.input(18)
        if input_state == False:
                print 'Button Pressed'
                time.sleep(0.2)
```

You will need to run the program as superuser:

```
pi@raspberrypi ~ $ sudo python switch.py
```

Button Pressed
Button Pressed
Button Pressed
Button Pressed

Discussion

You will notice that the switch is wired so that when it is pressed, it will connect pin 18 configured as an input to GND. The input pin is normally pulled up to 3.3V by the optional argument pull_up_down=GPIO.PUD_UP in GPIO.setup. This means that when you read the input value using GPIO.input, False will be returned if the button is pressed. This is a little counterintuitive.

Each GPIO pin has software-configurable pull-up and pull-down resistors. When using a GPIO pin as an input, you can configure these resistors so that one, either, or neither of the resistors is enabled, using the optional pull_up_down parameter to GPIO.setup. If this parameter is omitted, then neither resistor will be enabled. This leaves the input floating, which means that its value cannot be relied upon and it will drift between high and low depending on what it picks up in the way of electrical noise.

If it is set to GPIO.PUD_UP, the pull-up resistor is enabled; if it is set to GPIO.PUD_DOWN, the pull-down resistor is enabled.

You might expect the push switch to have just two connections, which are either open or closed.

Lab - 10
Toggling with a Push Switch

Problem

You want to turn something on and off with a push switch so that it toggles between on and off each time you press it.

Solution

Record the last state of the button and invert that value each time the button is pressed.

The following example toggles an LED on and off as you press the switch.

To make this recipe, you will need:

Breadboard and jumper wires

Tactile push switch
LED
470 resistor

Figure shows how to connect a tactile push switch and LED, using a breadboard and jumper wires.

Figure. Connecting a push switch and LED to a Raspberry Pi

In addition to the male-to-female jumper wires connecting the Raspberry Pi to the breadboard, you will also need one male-to-male jumper wire or solid core wire.

Open an editor (nano or IDLE) and type in the following code switch_on_off.py:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

switch_pin = 18
led_pin = 23

GPIO.setup(switch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(led_pin, GPIO.OUT)

led_state = False
old_input_state = True # pulled-up

while True:
    new_input_state = GPIO.input(switch_pin)
    if new_input_state == False and old_input_state == True: led_state = not
        led_state
```

```
        old_input_state = new_input_state
        GPIO.output(led_pin, led_state)
```

Discussion

The variable led_state contains the current state of the LED (True for on and False for off). Whenever the button is pressed, the following line is run:

led_state = **not** led_state

The not command inverts the value of led_state, so if led_state is True, it becomes False and vice versa.

The variable old_input_state is used to remember the button position so that a button press is defined as occurring only when the input state changes from being True(switch not pressed) to False (switch pressed).

Lab - 11
Make a Buzzing Sound

Problem

You want to make a buzzing sound with the Raspberry Pi. **Solution**

Use a piezo-electric buzzer connected to a GPIO pin.

Most small piezo buzzers work just fine using the arrangement shown in Figure. The one I used is an Adafruit-supplied component. You can connect the buzzer pins directly to the Raspberry Pi using female-to female headers.

These buzzers use very little current. However, if you have a large buzzer or just want to play it safe, then put a 470 resistor between the GPIO pin and the buzzer lead.

Paste the following code into the IDLE or nano editors. Save the file as buzzer.py.


```
import RPi.GPIO as GPIO
import time

buzzer_pin = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(buzzer_pin, GPIO.OUT)

def buzz(pitch, duration):
        period = 1.0 / pitch
        delay = period / 2
        cycles = int(duration * pitch)
        for i in range(cycles):
                GPIO.output(buzzer_pin, True)
                time.sleep(delay)
                GPIO.output(buzzer_pin, False)
                time.sleep(delay)
```

```
while True:
    pitch = input("Enter Pitch (200 to 2000): ")
    duration = input("Enter Duration (seconds): ")
    buzz(pitch, duration)
```

When you run the program, it will first prompt you for the pitch in Hz and then the duration of the buzz in seconds:

```
$ sudo python buzzer.py
Enter Pitch (2000 to 10000): 2000
Enter Duration (seconds): 20
```

Discussion

Piezo buzzers dont have a wide range of frequencies, nor is the sound quality remotely good. However, you can vary the pitch a little. The frequency generated by the code is very approximate.

The program works by simply toggling the GPIO pin 18 on and off with a short delay in between. The delay is calculated from the pitch. The higher the pitch (frequency), the shorter the delay needs to be.